Per-task Energy Accounting in Computing Systems

Qixiao Liu[1,2], Victor Jimenez[1,2], Miquel Moreto[1,2], Jaume Abella[2], Francisco J. Cazorla[2,3], Mateo Valero[1,2]
[1]Universitat Politècnica de Catalunya [2]Barcelona Supercomputing Center [3]Spanish National Research Council (IIIA-CSIC)

Abstract— We present for the first time the concept of *per-task energy accounting* (PTEA) and relate it to per-task energy metering (PTEM). We show the benefits of supporting both in future computing systems. Using the shared last-level cache (LLC) as an example: (1) We illustrate the complexities in providing PTEM and PTEA; (2) we present an idealized PTEM model and an accurate and low-cost implementation of it; and (3) we introduce a hardware mechanism to provide accurate PTEA in the cache.

1. Introduction

Energy is one of the most – if not the most – expensive resources in computing systems used in markets ranging from embedded to data centers. In the case of data centers, energy already accounts for 20% of the total cost of ownership in a large-scale computing facility [8] and 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). Energy consumption is also of prominent importance in the PC and embedded markets due to the limited energy stored in batteries, which calls for techniques to use it efficiently.

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 smartphones for instance, the diversity of the applications coming from different providers increases in every new generation. Data centers will observe a similar trend with the increasing number of offered cloud services. In this evolving scenario, we make a case for accurate per-task energy metering and accounting. Given a workload composed by n tasks $T_i, T_2, ..., T_n$ running in a processor with n cores, we define per-task energy metering and accounting as follows. *Per-task energy metering* (PTEM) consists in tracking the energy that a given task, T_i, consumes during a given period of time. *Per-task energy accounting* (PTEA) consists in 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. Both, per-task energy metering and accounting, are complex to derive with the increasing number of hardware shared resources that can serve requests from different tasks concurrently using different resources and/or with different latencies.

Accurately metering and accounting the energy consumed by each task in a computer has important applications across different computing domains including, but not limited to, the following:
1) *Selection of appropriate co-runners*. Task interaction in hardware shared resources may negatively affect tasks hurting performance and increasing energy requirements. Metering per-task energy can help the OS/runtime scheduler to decide which task to run and when, reducing systems's energy profile.
2) *Energy/Performance optimization*. Metering the energy consumed per task would allow finding the processor setup (e.g. number of cores) and software setup (e.g. mapping) that leads to the lowest system energy consumption.
3) In any power-capped system, when the system is reaching its power threshold, instead of reducing the activity and hence the energy and power of all tasks, a more fair capping could be implemented with PTEA by capping those tasks using more energy than that they would have consumed with a fair share of the resources.
4) *Billing*. Data centers charge users for the use of their resources. The fact that costs will be dominated by energy, makes billing directly dependent on the energy consumed by users' running jobs. In this scenario, ensuring consistent billing is a must. This requires PTEA such that a user running several times the same application with the same input data set will receive always the same billing regardless of the applications running in the other cores.

In recent years there has been an increasing interest for energy metering in different environments from data centers [3], [9] to smartphones [6], [16], [17]. In the case of smartphones the main focus is obtaining the overall system energy consumption or break it down per component (e.g. memory, chip). Many proposals [4], [13], [18] use performance monitoring counters (PMCs) or system events (such as OS system calls) to perform such breakdown. Power models are based on collecting data from a set of PMCs, voltage and temperature, and using predefined weights derived through correlation. Although previous approaches have shown to be very accurate in metering per-component and overall system's energy consumption, they neither provide per-task energy measurements nor allow performing per-task energy accounting, which are the focus of this paper. It is our position that as processor design moves towards multi- and many-core processors, in which an increasing number of different applications simultaneously run on the same chip, providing per-task energy metering and accounting becomes increasingly important.

In this paper, we make a case for accurate online per-task energy metering and accounting. We introduce both terms, show their differences and relate them. Next, using the shared LLC as illustrative example, we propose a model to derive the energy that should be metered to a task when running with other tasks simultaneously sharing several hardware resources. To that end, we break down resources into different types based on their energy profile and show how per-task energy metering is quite sensitive to the type of the resources under consideration. Finally, we motivate the need for providing hardware support for energy accounting and the importance of ensuring that the energy accounted to a task is always the same regardless of the other tasks it runs with. We also propose an initial model to perform per-task energy accounting for a shared LLC.

2. Formalizing the definition of per-task energy metering and accounting

We break energy into its main three components. Dynamic energy corresponds to the energy spent to perform those useful activities that circuits are intended to do, including short-circuits from supply to ground when switching gates. Static energy corresponds to the energy consumed due to useless activity not triggered by the program(s) being run (e.g. bitline precharge if no access occurs)[1]. 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.

2.1. Per-Task Energy Metering

Per-task energy metering consist in tracking the energy that a given task consumes during its execution. This requires deriving the energy consumed per task in private hardware components (i.e. components only used by the task in a given point in time) such as the cores in a multicore CPU, and shared resources, such as caches.

The difficulty with shared resources resides on the fact that they can serve requests from different tasks concurrently, and each request type may generate different internal activity in the resource with variable duration. This seriously challenges per-task energy metering. Current

[1]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

methods for energy metering focus mostly on time-shared resources (e.g. CPUs) and are based on usage time and allocated resources. This may be adequate if leakage and static power dominate the total power consumption. However, this is no longer true with the shift towards energy-proportional systems [2] where most of the energy consumed by an application - and hence, its cost - is due to its activity. Hence, in an energy proportional system two customers that incur different utilizations across similarly allocated resources for similar usage time, will be accounted the same energy consumption while in reality their energy consumption profiles can be quite different. In [9] authors run several homogeneous programs in isolation on the same platform for a fixed period of time. Results show that power dissipation across these homogenous programs with similar resource and time allocation may vary more than 20%. More heterogeneous workloads including database processing, I/O-intensive applications as well as high-performance ones exhibit higher power variations.

Our view is that, the energy metered to a given task should be proportional to its resource usage. This includes the number and type of accesses to the different resources and, for stateful resources (e.g., Branch Target Buffer, caches and TLBs) the fraction of the space occupied by the task. The accuracy in per-task metering depends on the characteristics of the hardware resources used and the hardware support enabled for energy metering. Note that energy metering for multithreaded tasks (applications) simply consists in adding up the energy consumed by each of its constituent threads.

Per-task leakage energy metering: Splitting leakage energy among running tasks requires considering two different types of resources. Those whose leakage is proportional to time and those whose leakage is proportional to both time and space. Under the former type we find resources like logic blocks in cores that consume leakage energy proportionally to the time they are enabled. Under the latter type we find resources like caches or memory, where leakage is proportional to the fraction of the resource that a task demands and the duration that fraction of the resource is used.

Per-task static energy metering: Static energy is consumed by useless activities in idle resources. It must be attributed to tasks depending on the amount of space they occupy if those resources are stateful (e.g. caches). However, if resources are stateless, one fair way to meter such energy across different tasks is distributing it uniformly since no task triggers on-going activity in those resources.

Per-task dynamic energy metering: In order to split dynamic energy across running tasks we consider the number and type of accesses that each task performs to each resource. This requires per-task access counters and the energy consumption per access to be provided by the chip vendor. While this is highly accurate it may be costly to track, so simpler and higher-level approaches based on tracking CPU utilization may also correlate well with dynamic power consumption.

2.2. Per-Task Energy Accounting

The concept of Energy Accounting is inherited from CPU accounting that was first introduced in [11] and then develop for multicores in [10], [12] and for SMTs in [7]. CPU accounting measures the CPU utilization of a given task during a period of time when it runs in a multicore processor. CPU utilization depends on both the time the task is scheduled in the CPU and the progress (or slowdown) the task does in the multicore processor. The latter is computed by determining which accesses 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% (hence its progress is 70%) with respect to its execution with a fair share of the resources, it is only accounted $1,000 \times 0.7 = 700$ cycles.

Let's assume that a given task T_A runs for a period of X cycles executing Y instructions. Per-task energy accounting consists in determining, while task T_A runs concurrently with other tasks, the energy T_A would have consumed to execute the same Y instructions with a fair share of the multicore processor resources. For instance, in a 4-core processor with private data and instruction caches and shared L2, if we run a workload composed by 4 tasks, we want to determine the energy each of those tasks would have consumed with a single core, $\frac{1}{4}$ of the L2 cache, $\frac{1}{4}$ of the memory bandwidth and $\frac{1}{4}$ of the on-chip interconnection network bandwidth.

We aim at maintaining the same *Principle of Accounting* that rules when measuring CPU accounting [11]: the energy accounted to a task should be independent from the workload in which this task runs.

Performing accurate per-task energy accounting requires a per-task energy metering mechanism. Given a task T_i for which we can derive its energy metering EM_{Ti} and its energy accounting EA_{Ti} we define the Energy Accounting ratio (*EAratio*) as $EAratio = \frac{EM_{Ti}}{EA_{Ti}}$. If $EAratio > 1$, the task has consumed more energy than it would have consumed with a fair share of the resources. This may happen for instance if task T_i gets less than $\frac{1}{4}$ L2 space in the example above because other tasks make an intense use of L2, making T_i running longer than with a fair share of the resources. If $EAratio < 1$, T_i has consumed less energy than with its fair share of the resources. This may be caused because T_i is using more resources than its fair share which speeds up T_i.

One of the major difficulties in deriving per-task energy accounting resides in the fact that there is not a direct relation between the amount of resources used by a task and its performance. For instance, a task receiving x% more resources than its fair share can decrease its execution time much more (or much less) than x%.

The principle of PTEA is inherited from Per-task CPU accounting though with CPU accounting mechanisms the same functionality that PTEA achieves cannot be provided. In that sense, the concept of PTEA bases on two main pillars: PTEM and Energy Metering Modulation (EMM). The former derives the actual energy consumed by a given task when it runs as part of a workload. The latter, modulates the figure obtained with the PTEM mechanism in order to derive the actual energy that the task would have consumed with a fair share of the resources (e.g. by using the $EAratio$ defined above). One of the ways to achieve EMM is through CPU accounting. CPU accounting by itself provides the time it would have taken a given task t_i to execute a certain set of instructions I_j with a fair share of the resources. Interestingly, it can be the case that a task t_i that executes I_j can take x% longer to execute than with a fair share of the resources if (1) either t_i receives lower bandwidth to the shared bus or if (2) its LLC lines are evicted. However, in the latter case t_i consumes more energy than in the former since every miss in the LLC leads to energy-hungry memory accesses. Hence, under both scenarios the CPU accounting for the task is the same though its energy profiles may significantly vary.

Note that PTEA dynamically derives the energy that a task would have consumed with a fair share of the resources, while such task runs as a part of a workload. This removes the need for any profiling.

3. Per-Task Energy Metering for a Shared Cache

We focus on the case of a shared cache as it is a good representative of the challenges to address when carrying out per-task energy metering and accounting. We assume a multicore architecture where each core has private data and instruction first level caches plus a shared on-chip Last-Level Cache (LLC). Note that many multicore architectures in the high-performance and embedded domains include an on-chip shared LLC, such as the Aeroflex Gaisler NGMP and the IBM POWER7 processors.

We now present our idealized utilization-based model for per-task energy metering for the LLC. Next we present a first approach of

hardware support for per-task energy metering.

1) Idealized Per-Task Energy Metering: The dynamic energy consumption in the shared LLC for a given task i is proportional to the number of accesses. It is 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. $\#action^{LLC}_k(tk_i)$ stands for the number of LLC accesses of type k performed by the task i. Access types include: read hits, read misses evicting a dirty line, write hits, invalidations, etc. The energy consumption of an access changes depending on the characteristics of the access. For instance, a read hit will consume less energy than a read miss that evicts a dirty line.

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 [1]. 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} is 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, where E^{LLC}_{leak} is the leakage energy per cycle consumed by the LLC. This value is also an input parameter for the idealized model:

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

2) Hardware Support for Per-Task Energy Metering: The ideal model for the LLC tracks two main per-task parameters: access (activity) counts per access type and cache occupancy. Our hardware support 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: (1) tagging each cache line with a *task id*, (2) keeping a counter per task with the number of owned cache lines (*instant counter*), and (3) 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 [14]. Therefore, we propose sampling the LLC occupancy in two different 'directions'. First, only some cache sets will be monitored [19], 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.

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

$$Occ^{LLC}(tk_i) = \frac{Occ^{LLC}_{cum}(tk_i) \times SmpFreq \times SmpSets}{\#Sets^{LLC} \times ExecTime(tk_i)} \quad (4)$$

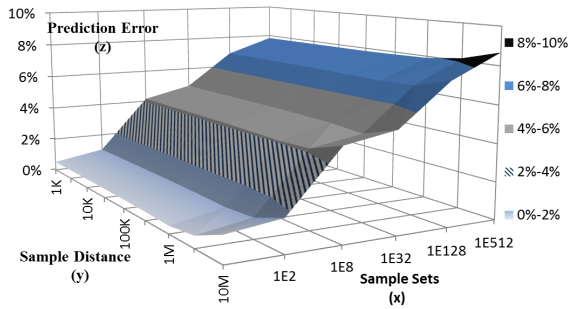

Fig. 1. LLC energy metering error with sample distance and sets

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 sets (1,024 in the example). The impact of sampling in both time and sets is analyzed next.

3) Results: We use an enhanced version of SMTSim [20] extended with power models analogous to those of Wattch [5]. Wattch-like power models are built on top of CACTI 6.5 simulation tool [15]. We consider a multicore setup in which each core has its private 32KB 4-way instruction and data L1 caches, while the 16-way 2MB L2 cache is shared among all cores. We use traces collected from the SPEC CPU 2006 benchmarks. For this experiment, we construct 16 varying-behavior 4-task workloads, all including the *astar* benchmark.

Figure 1 shows the effect of sampling sets and period on the average LLC energy prediction accuracy for *astar* in all 16 workloads. 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 1 every 8 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 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.

4. Per-Task Energy Accounting for a Shared Cache

In order to evaluate our PTEA mechanism we ran experiments with SPEC CPU 2006 benchmarks. We focus on 3 representative benchmarks: *namd*, a pure CPU-bound benchmark, with negligible LLC usage; *astar*, a benchmark with medium cache usage but whose performance is sensitive to cache space; *libquantum*, a cache-hungry benchmark. We study each benchmark as part of 4 different workloads composed by *libquantum* (L) and *namd* (N). The benchmark under study is labeled as *X*. Then the particular workloads studied are *(X,N,N,N)*, *(X,L,N,N)*, *(X,L,L,N)*, and *(X,L,L,L)*. In all 4 cases, we measure and account *X*'s LLC energy consumption.

The gray bars in Figure 2 show the energy metered to all three benchmarks under study in the LLC normalized to their LLC energy consumption in isolation with a fair share of the resources ($\frac{E_{shared}}{E_{fairshare}}$). We observe that depending on the workload, the energy measured to *X* varies significantly. The main reason for this behavior corresponds to the shared LLC interaction. For instance, *namd* gets very little LLC space if it runs with cache-hungry benchmarks in the workload, which decreases its static and leakage energy. Meanwhile, its few extra LLC misses barely affect its LLC energy consumption. Conversely, *astar* suffers an increase in LLC misses due to inter-task interferences. It increases its dynamic energy due to extra accesses to replace cache lines and also executes longer, which increases its leakage and static energy consumption in the LLC. In contrast, it occupies significantly less LLC space, leading to lower power consumption. *libquantum* demands high LLC space;

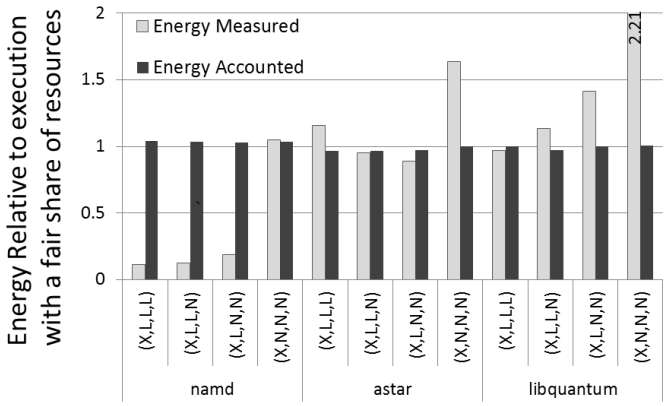

Fig. 2. LLC energy metered and accounted to *namd*, *astar*, and *libquantum* when they use a fair share of resources and as part of different workloads. *L* stands for *libquantum* and *N* for *namd*

however, it suffers few extra misses because of its extremely high miss rate. *libquantum* decreases total static energy in all workloads with plentiful LLC accesses (thus fewer idle cycles). Anyway, its static and leakage energy increase due to its high LLC occupancy.

Although not analyzed in this paper, note that execution time variations due to LLC interaction will also have an impact in the leakage and static energy consumed in other resources.

In our solution for computing the LLC energy a task would consume with a fair share of the resources, we combine the proposed per-task energy metering mechanism with hardware support for fair CPU accounting [7], [10]–[12]. In particular, we use ITCA, the technique proposed in [12] that focuses on the progress a task would have with a fair share of the resources. ITCA makes use of dedicated hardware monitoring support to track the interferences between tasks running in different cores. This hardware support tracks through an Auxiliary Tag Directory, the conflicts in the LLC.

Our approach to compute per-task energy accounting scales leakage, static and dynamic energy according to the LLC behavior in isolation. Since static and leakage LLC energy are proportional to occupancy and depend on execution time, we scale these values with the factor $\frac{Occ^{LLC}_{fairshare}}{Occ^{LLC}_{shared}} \cdot Progress_{Ti}$, where $Progress_{Ti}$ is the progress done by the task with a fair share of the resources. If the task in isolation occupies more LLC space or runs faster, these values are scaled accordingly. The dynamic energy depends on the LLC activity, so we scale energy based on the factor given by the number of accesses since hits perform a single LLC access and misses two (the actual miss and the cache line fill): $\frac{Hits^{LLC}_{fairshare}+2\cdot Misses^{LLC}_{fairshare}}{Hits^{LLC}_{shared}+2\cdot Misses^{LLC}_{shared}}$. The CPU mechanism proposed in [12] provides the progress of each task $Progress_{Ti}$. Further, its deployed Auxiliary Tag Directory (ATD) could easily provide the hits ($Hits^{LLC}_{fairshare}$) and misses ($Misses^{LLC}_{fairshare}$), and the occupancy $Occ^{LLC}_{fairshare}$ under the setup with a fair share of the cache in this work.

The black bars in Figure 2 show the results of applying our solution for per-task energy accounting. We observe that the predicted LLC energy accounting is really close to the fair energy metered across all benchmarks under study: 2.1% average error. In contrast, the energy metered has a 44.7% average error.

5. Other resources

PTEA and PTEM are to be applied to the main energy-consumption contributors such as cores, caches, memory and storage devices (e.g., disks), which is part of our current work. The principles we have presented for the LLC can be generalized for other resources. Dynamic energy requires activity counters per resource (e.g., logic blocks in cores, storage device accesses). Static/leakage requires tracking occupancy whenever unused parts can be put into a low power state (e.g., LLC if lines/banks can be turned off). Static/leakage energy requires tracking the power state induced by each running task, so that tasks keeping devices in active state are metered higher energy than those accessing devices in bursts so that devices would remain in a low power state for long periods. The way in which each of these requirements can be accomplished for the resources with high energy profiles in a computing system is part of our future work.

6. Conclusions

The increasing cost of energy calls for accurate per-task energy metering and per-task energy accounting. The advent of multicores challenges both since co-running tasks share (and interact in) shared hardware resources which affects their energy profiles.

To our knowledge, this is the first paper presenting the concept of per-task energy accounting on multicores and relating it to energy metering. We illustrate how to perform both by proposing and evaluating new models for the case of a representative resource like a shared cache. In particular, we present several proof-of-concept mechanisms to compute the energy that a task consumes in a multicore when it runs as part of a workload (metering) and a mechanism to derive the energy that task would have consumed if executed the same instructions with a fair share of the resources (accounting).

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. Qixiao Liu has also been funded by the Chinese Scholarship Council under grant 2010608015.

References

[1] J. Abella et al. IATAC: a smart predictor to turn-off L2 cache lines. *ACM TACO*, 2005.
[2] L. Barroso and U. Holzle. The case for energy-proportional computing. *IEEE Computer*, 40:33–137, October 2007.
[3] R. Bertran et al. Energy accounting for shared virtualized environments under DVFS using PMC-based power models. *FGCS*, 2012.
[4] W. Bircher and L. John. Complete system power estimation: A trickle-down approach based on performance events. In *ISPASS*, 2007.
[5] D. M. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*, 2000.
[6] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX Annual Technical Conference*, Jun 2010.
[7] S. Eyerman and L. Eeckhout. Per-thread cycle accounting in smt processors. In *ASPLOS*, 2009.
[8] J. Hamilton. Internet-Scale Service Infrastructure Efficiency. In *ISCA*, 2009.
[9] V. Jiménez et al. Energy-aware accounting and billing in large-scale computing facilities. *IEEE Micro*, 31(3):60–71, 2011.
[10] C. Luque, M. Moreto, F. Cazorla, and M. Valero. Fair CPU time accounting in cmp+smt processors. In *HiPEAC*, 2013.
[11] C. Luque, M. Moreto, F. J. Cazorla, R. Gioiosa, A. Buyuktosunoglu, and M. Valero. CPU accounting in cmp processors. In *IEEE Computer Architecture Letters(CAL)*, volume 9, 2009.
[12] C. Luque, M. Moreto, F. J. Cazorla, R. Gioiosa, A. Buyuktosunoglu, and M. Valero. Cpu accounting for multicore processors. *IEEE Trans. Comput.*, 161, 2012.
[13] J. McCullough et al. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conference*, 2011.
[14] M. Moreto, F. Cazorla, A. Ramirez, and M. Valero. MLP-aware dynamic cache partitioning. In *HiPEAC*, 2008.
[15] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. CACTI 6.0: A tool to understand large caches. *HP Tech Report HPL-2009-85*, 2009.
[16] Nokia. Energy profiler. S60 platform: Effective power and resource management v3.1, 2007.
[17] A. Pathak et al. Fine-grained power modeling for smartphones using system call tracing. In *EuroSys*, 2011.
[18] K. Pusukuri, D. Vengerov, and A. Fedorova. A methodology for developing simple and robust power models using performance monitoring events. In *WIOSCA*, 2009.
[19] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. MICRO 39, 2006.
[20] D. Tullsen, S. Eggers, and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *ISCA*, 1995.