SEDEA: A Sensible Approach to Account DRAM Energy in Multicore Systems

Qixiao Liu[◇], Miquel Moreto[†,‡], Jaume Abella[†], Francisco J. Cazorla[⋆,†], Mateo Valero[†,‡]

[◇] Shenzhen Institute of Advanced Technology, Chinese Academy of Science (CAS), Shenzhen, China
[†] Barcelona Supercomputing Center (BSC), Barcelona, Spain
[‡] Universitat Politecnica de Catalunya - Barcelona Tech (UPC), Barcelona, Spain
[⋆] Spanish National Research Council (IIIA-CSIC), Barcelona, Spain.

***Abstract*—As the energy cost in today's computing systems keeps increasing, measuring the energy becomes crucial in many scenarios. For instance, due to the fact that the operational cost of datacenters largely depends on the energy consumed by the applications executed, end users should be charged for the energy consumed, which requires a fair and consistent energy measuring approach. However, the use of multicore system complicates per-task energy measurement as the increased Thread Level Parallelism (TLP) allows several tasks to run simultaneously sharing resources. Therefore, the energy usage of each task is hard to determine due to interleaved activities and mutual interferences. To this end, Per-Task Energy Metering (PTEM) has been proposed to measure the actual energy of each task based on their resource utilization in a workload. However, the measured energy depends on the interferences from co-running tasks sharing the resources, and thus fails to provide the consistency across executions. Therefore, Sensible Energy Accounting (SEA) has been proposed to deliver an abstraction of the energy consumption based on a particular allocation of resources to a task.**

In this work we provide a realization of SEA for the DRAM memory system, SEDEA, where we account a task for the DRAM energy it would have consumed when running in isolation with a fraction of the on-chip shared cache. SEDEA is a mechanism to sensibly account for the DRAM energy of a task based on predicting its memory behavior. Our results show that SEDEA provides accurate estimates, yet with low-cost, beating existing per-task energy models, which do not target accounting energy in multicore system. We also provide a use case showing that SEDEA can be used to guide shared cache and memory bank partition schemes to save energy.

I. Introduction

As the energy price keeps rising, energy has already become one of the most expensive resources in current computing systems [6]. Therefore, in the case of a datacenter or supercomputer facility, it is fair charging users for the energy consumed, since energy usage is proportional to the cost of their operations, the same as for the time and computing resources (CPU, Memory). However, as multi- and many-core processors have already become the reference platform, several tasks run at the same time, thus challenging per-task energy measurement. Furthermore, the number and heterogeneity of the tasks that coexist in a computing system significantly increases over time. Amongst the main energy contributors in a multi/many-core system, we find the memory subsystem, whose energy share is almost as important as the processor socket one [5], [25]. However, due to the fact that increased TLP in multi/many-core systems leads to a large number of tasks sharing the memory system simultaneously, and the internal operations of the memory system are often opaque to the operating system, particularly complicates the per-task memory energy usage measurement.

Per-Task Energy Metering (PTEM) has been introduced to meter the actual runtime energy of tasks based on their individual resource utilization when they are running in a multicore system, including their incurred activities and interactions on-chip and off-chip in the DRAM devices [14], [15]. However, PTEM metered energy is affected by the behavior of other co-running tasks in the same system, we regard it inappropriate for energy accounting as the same program with the same inputs would be assigned different energy costs based on factors beyond the end user's control. Therefore, if end users were charged based on PTEM energy measurements, their energy bill would vary drastically from run to run despite executing exactly the same program with the same inputs, which would be inconsistent and unfair from the end user perspective.

Therefore, novel mechanisms are needed to account the actual energy a task would consume when using a specific fraction of the hardware resources, discounting the effects due to interactions with other tasks running simultaneously in the multicore. To this end, Sensible Energy Accounting (SEA) delivers an abstraction of the energy consumption of tasks in multicores discounting interferences, so that energy accounted is fair and consistent [12]. A realization of such concept for the on-chip resources has been shown by Liu et al. [13], in which each task is accounted the energy it would consume when it has been allocated a particular fraction of on-chip resources. However, this work only takes into account the energy cost of on-chip resources. The side effect on off-chip resources, such as the memory system, has been ignored. As the interactions between resources have significant impact on the energy profile of tasks, accounting accurately the per-task memory energy can enhance the fairness and consistency for the energy to account to users. Moreover, accounting task-level memory energy based on resource utilization can also help optimizing the resource allocation for energy efficiency, which in turn helps reducing the operation costs.

In this paper, we introduce SEnsible DRAM Energy Accounting (SEDEA), a mechanism to account DRAM energy to tasks, thus predicting the energy they would have consumed under a given (arbitrary) allocation of on-chip Last Level Cache (LLC) ways. SEDEA predicts the energy a task would consume when running with a specific resource partition with no interferences from other tasks, based on the runtime behavior of the target task when it runs within any workload with a fully shared LLC. SEDEA accounts task energy by predicting the LLC hit/miss behavior and the memory bank activity when the task under study would run in isolation. Overall, the contributions of this work are as follows:

- We propose an ideal model to sensibly account a task the DRAM memory energy it would have consumed if it had been allocated an arbitrary fraction of the LLC exclusively. To the best of our knowledge, it is the first reference model against which energy accounting mechanisms for DRAM memories can be compared to.
- We propose SEDEA, a model extending current energy accounting techniques by adding the off-chip memory system. We account the DRAM energy to a task in each case based on predicting its on-chip activities, its execution time and its DRAM bank usage. We compare the prediction accuracy of SEDEA with an Evenly Split (ES), a Proportional To Access (PTA) and a PTEM technique in DRAM memory, DReAM [14], to show that SEDEA provides consistent and accurate estimates for the energy to account.
- We provide a use case of SEDEA that co-partitions cache and memory banks and optimizes the system energy consumption. We show that such technique obtains higher energy savings than state-of-the-art approaches.

The rest of this paper is organized as follows. Section II introduces the background of this work and the related work. Section III presents SEDEA, our approach to account DRAM memory energy and its efficient hardware implementation. The accuracy of SEDEA is evaluated in Section IV. Finally, Section V draws the main conclusions of this paper.

II. Background and Related Work

A. Energy Metering in Computing System

There are many approaches to meter the energy usage in a computing system. Next, we classify those approaches into two main categories as follows:

Firstly, Per-Component Energy Metering (PCEM) derives the energy consumed by the main hardware components such as CPU and memory [2], [3], [17], [20], [23]. These techniques first estimate the overall system energy consumption, and then break it down per component (e.g. CPU, memory), by using performance monitoring counters (PMCs) or system events such as system calls to carry out such measurements. Power models rely on collecting data from a set of PMCs, and voltage and temperature information, to estimate power through correlation.

In contrast, PTEM [14], [15] has been proposed to break down the system energy across tasks based on their actual resource utilization, when they run simultaneously in a workload in a multicore system. The main challenge of PTEM is dealing with shared hardware resources, as the energy consumption of tasks significantly changes depending on the co-running tasks.

As a result, PTEM metered energy depends on the behavior of other tasks. Such dependence is particularly problematic in environments where users are charged for the usage of resources, including energy. Users running the same applications with the same inputs would observe different energy profiles for their applications and hence, would be billed inconsistent amounts across runs. Therefore, Liu et al. introduced Sensible Energy Accounting (SEA) [12], [13]. For each task T_i, SEA aims at dynamically accounting (i.e. while they run with other tasks in a workload), the energy T_i would have consumed, $E_{T_i}^{fhr}$, if T_i had been run in isolation with a certain fraction of the hardware resources, fhr.

In this work, we aim at extending SEA to the memory system. That is, to predict the DRAM energy that a task would have consumed if it had been allocated a fraction of the on-chip resources (the LLC in our case), fhr. When running in isolation with fhr of resources, T_i may behave in DRAM devices differently, compared to the case when it runs as part of a workload. The causes of these differences are as following: 1) T_i may suffer extra LLC misses, known as inter-task misses, caused by co-running tasks; 2) T_i may experience extra hits when running in a workload if it manages to use more than its LLC fraction, fhr; 3) Memory requests from T_i may be delayed since the memory controller schedules additional memory requests belonging to co-running tasks. In contrast, SEDEA predicts the in-isolation DRAM energy consumption (with a fraction of the LLC) based on its behavior when running in a workload. To the best of our knowledge, SEDEA is the first model to do so.

B. DRAM Memory System

The DRAM memory system adopted in this work strictly follows the JEDEC DDR standard [7]. Memory requests sent from the chip are dispatched to the memory system through the memory controller. In this paper, we inherit the memory controller model from DRAMsim2 [22], which uses a typical scheduling policy, known as first-ready first-come-first-serve (FR-FCFS). Such policy prioritizes the ready and old commands over the non-ready and newly arrived commands. By applying such policy, the commands are not issued exactly following their arriving order.

DRAM memory energy variation across workloads can be large [4] and is likely to increase in the future as system manufacturers pay increasing attention to energy efficiency. The energy in DRAM memory can be modeled by current profiles provided by the hardware vendor [18]. We build upon such an approach to break down the energy into three components: active, refresh and background energy. Active or dynamic energy correspond to the energy spent to perform those *useful* activities that circuits are intended to do triggered by the running programs. For instance, the energy spent to retrieve data from memory on a read operation. Refresh energy corresponds to the energy consumed to refresh periodically all memory contents in DRAM cells to avoid losing their contents. Background energy includes maintenance and leakage energy, which refers to all energy consumed except dynamic and refresh energy. Maintenance energy corresponds to the energy consumed due to *useless* activities not triggered by the program(s) being run. Leakage energy corresponds to the energy wasted due to imperfections of the technology used to implement the circuit.

We use the current profiles listed in Section IV-A to compute each component of energy consumed in the memory system. Then, we make use of the methods proposed by Liu et al. [14] to attribute those energy components to each running task by correlating their running stats.

C. Memory Resource Partition

The memory hierarchy is the main shared resource in multicore architectures, and also the main source of interferences between co-running tasks. To isolate the tasks performance, cache and memory bank partitioning techniques have been proposed and proved to improve the system throughput and the individual task performance [19], [21], [27].

a) ***Cache partition:*** In most current multicore architecture, on-chip LLC is shared among cores. However, it sometimes becomes the main cause of performance loss due to the different application characteristics and increasing core count. LLC partitioning has been used to reduce the conflicts between co-running tasks, which allows a task to use its own allocated cache space. Some authors proposed the Utility-based Cache Partition (UCP) to increase the performance contribution of cache based on the tasks needs, by devising a monitoring hardware mechanism [21]. In contrast, Lin et al. [10] propose to partition the cache through coloring the page at operating system level.

b) ***Memory bank partition:*** Also, in the memory system, some works have shown that increasing the memory bank-level parallelism and row-buffer locality can improve system performance. Kim et al. [9] propose to prioritize the thread that imposes less interference to the others to increase the memory bandwidth utilization. Another work proposes the Heterogeneous Multi-Channel (HMC), which divides the memory system into sub-banks, and applies the FR-FCFS policy locally in each sub-bank [27]. HMC reduces bank-level conflicts and improves the throughput of the DRAM memory system.

Cache and memory bank partitioning can be combined [11], where a task is allocated the same portion of cache and memory space vertically through page coloring. However, the cache and memory bandwidth requirements of a task are complementary [26]. This claim builds on the observation that, when a task has allocated a sufficiently large cache partition, it needs less memory bandwidth. Hence, cache and memory bandwidth partitioning methods can be improved by using heterogeneous and independent policies.

In this work, we show the energy saving potential of SEDEA, by improving the cache and memory bank co-partition mechanism.

III. Sensible DRAM Energy Accounting

In this section, we introduce our approach to account DRAM energy to a task based on its number of allocated LLC ways. When a task T_i is running in a workload in a multicore system, we aim at accounting the energy it would have consumed when run in isolation with a given fraction of the LLC space. Thus, in a N-way set-associative LLC, T_i could own $n \leq N$ ways, where n refers to the particular number of ways in the LLC used by T_i to account its DRAM memory energy.

A. Ideal Model

We begin with a theoretical model to discuss how the energy should be accounted to a task. Then, we describe our approach, SEDEA, which is implementable with affordable cost.

a) **Active energy:** In the DRAM memory system, some *useful* activities are performed by each running task. For each read/write memory request received in the memory controller, a set of Activate (ACT), Precharge (PRE) and Read/Write (R/W) commands will be dispatched to the off-chip memory system. In particular, ACT command activates a row in the memory bank and loads the data from DRAM cells into the row-buffer (through sense amplifiers). Upon the completion of the ACT command, any R/W command can be issued to read or write the data in specific columns of the row-buffer. After that, a PRE command is send to close the row-buffer while the data being read (in case of read commands) is transferred back to the on-chip memory controller, storing the data back into the DRAM cells.

Therefore, to account the active energy, $EA_n^{Act}(T_i)$, to task T_i, we need to account T_i the useful activities it would incur when it has n ways of the LLC.

$$EA_n^{Act}(T_i) = \sum N_n^{Comm}(T_i) \times E^{Comm} \tag{1}$$

where E^{CM} represents the energy consumed by each command type, and $N_n^{CM}(T_i)$ is the number of commands task T_i incurs when allocated n LLC ways.

b) **Background energy:** Background energy includes both, energy consumed due to leakage power and maintenance power. Note that the background power of the memory system has different levels corresponding to different states of the DRAM device: power down (P), standby (S) and active (A). The power in P state is incurred when the memory system is clock-disabled. After enabling clocking, the DRAM device enters S state, which largely rises the background power, but can quickly respond to requests. When executing the ACT command, the background power further rises to A state, but the increment is relatively small. The A state also holds for the duration of the corresponding R/W command and the PRE command. After the PRE command precharges the open row in the DRAM device, the background power returns to S state.

After the devices stay in idle state for a given time period, the clock is disabled to save energy, thus returning the device to P state.

As a result, the background energy we account to task T_i when it uses n ways in LLC, is determined by the time DRAM devices spend in each state. We can calculate the background energy to account to task T_i when running alone with n-way LLC as follows:

$$EA_n^{BG}(T_i) = T_{i,n}^A \times P^A + (T_{i,n}^S + T_{i,n}^A) \times P^S + T_{i,n}^{Exe} \times P^P \quad (2)$$

where P^A, P^S and P^P represent the background power increment under A, S and P states. Note that P^P is consumed across the whole execution of T_i, P^S is consumed when the bank state has been raised to S or A (additionally to P^P). In contrast, P^A will only be consumed when the bank is in A state (additionally to P^P and P^S). We use $T_{i,n}^{Exe}$, $T_{i,n}^S$ and $T_{i,n}^A$ to refer to the execution time of task T_i, and the time it spends in S state and A state respectively.

c) **Refresh energy:** The DRAM memory system needs to refresh periodically all memory contents. Such period is set according to the JEDEC standard [7] so that refreshes occur at the minimum frequency that guarantees that DRAM contents will be preserved. For example, we use 40μs in our configuration. Therefore, the refresh energy of T_i depends on its execution time:

$$EA_n^{REF}(T_i) = T_{i,n}^{Exe} \times P^{REF} \quad (3)$$

where P^{REF} stands for the refresh power in the specific DRAM memory system.

B. Implementation

Next, we introduce SEDEA, an implementable, yet accurate, model that follows the principles of the ideal model.

a) **Active energy:** Accounting the active energy for T_i depends on accurately estimating the number of DRAM internal commands with different LLC allocations, as shown in Equation 1. This is a challenging task as we can only record the memory request forwarded by the LLC due to LLC misses when T_i runs in a workload. The number of LLC misses reported with the Performance Monitoring Unit (PMUs) does not match the activities to account, since it also includes the inter-task misses that are produced due to sharing the LLC with the co-running tasks in the workload, and excludes capacity misses that would be incurred otherwise by using only n LLC ways.

Therefore, we rely on the Auxiliary Tag Directory (ATD) [21], which focuses on a least recently used (LRU) replacement policy. While the LLC is shared among all tasks, the ATD keeps a local (per-core) copy of the tag directory, that is only updated with the accesses of the corresponding (owner) task. Besides, if the LLC implements LRU, one can predict whether an access would miss in the LLC for any number of cache ways n lower or equal to the total number of LLC ways (N). This occurs because LRU keeps in each set the position of each address in the LRU stack, and therefore, the order in which they would be evicted if they were not reused, is maintained. With the LRU stack one can determine whether a given access would hit or miss with n ways (where $n \leq N$) by simply checking if it hits any of the n Most Recently Used (MRU) entries of the ATD.

To keep hardware overheads low, we implement a Sampled ATD (SATD), which only monitors a sampled number of sets instead of the whole LLC [13], [21]. Moreover, the SATD can also be used for pseudo-LRU caches with negligible impact on accuracy [8]. Adapting the ATD to other replacement policies is left as future work and beyond the scope of this paper.

The overall hit probability for the different number of ways, $h_1, ..., h_N$, is computed for the sampled sets. Whenever a non-sampled set is accessed, which will likely be the case of most accesses, the access can be predicted to be a hit or a miss by using a *Monte Carlo* approach, which offers a high degree of accuracy and can be applied to each access at execution time [13].

Given that the memory controller dispatches all LLC misses as memory requests, we can rely on the miss estimates and their type (either read or write) as predicted by the SATD to account the active energy to task T_i as follows:

$$EA_n^{Act}(T_i) = (E^{ACT+PRE+R/W}) \times N_n^{R/W}(T_i) \quad (4)$$

where $ACT, PRE, R/W$ refer to activate, precharge and read/write commands respectively, and E represents the energy cost of each command that is provided by the hardware manufacturer [18]. $N_n^{R/W}(T_i)$ stands for the number of R/W memory requests estimated with the SATD.

b) **Background energy:** To account the background energy, we need to estimate the execution time that the task would take in isolation as well as the time the DRAM banks spend in high power states. Both parameters are determined by the activities performed by the task.

We use the following metrics to infer these parameters:

- Given a memory request that experiences no contention when it is served, its latency will be a fixed value that is specified by the JEDEC standard [7] and hardware vendor implementations, namely *DL* (Default Latency).
- The second metric we use corresponds to the count of all cycles that a task spends with at least one memory request in the memory system, which we name as *IMC* (In-flight Memory Cycles). *IMC* represents the time a task induces the memory system to a high power-consuming state.
- For each memory request, we also monitor its service time, from the cycle it is issued to the memory controller, till the cycle it completes. This metric represents the whole miss penalty this particular LLC miss suffers, denoted as *MP*. During the MP period, the execution of the task in the processor pipeline may stall for a certain time till the miss is served.

Note that for IMC and MP we need to setup a set of counters for each task: one for IMC, and as many as pending requests allowed plus one for MP. The additional MP register stores the final latency of the last request served. The IMC counter and the MP register are read and reset periodically (every 1,000 cycles in our setup). The size of the IMC and MP counters relates to the duration of the period. In our case, given a 1,000 cycles period, 10-bit counters are guaranteed to suffice, although 8 bits are enough in practice. Given that MP is the actual latency of a LLC miss and IMC represents the aggregate effect after overlapping, we calculate a Bank-level Parallelism Factor (BPF) in a period as follows:

$$BPF = \frac{MP \times N_{MR}}{IMC} \tag{5}$$

where N_{MR} represents the number of memory requests monitored in that period.

Next, we extend the CPU accounting method proposed by Luque et al. [16] to estimate the execution time of T_i, since we use a detailed memory system instead of a memory system with a constant latency. We also aim at accounting the execution time when T_i uses n LLC ways exclusively instead of the whole LLC shared with other tasks. The number of memory requests N_n when T_i runs alone with n LLC ways can be estimated using the SATD, as described before. In particular, N_n is obtained as follows:

$$N_n = N_{WL} - N_{ITM} + N^n_{CM} \tag{6}$$

where N_{WL} represents the actual memory requests of task T_i during the monitored period (while running in a workload), N_{ITM} stands for the number inter-task misses, and N^n_{CM} is the number of capacity misses T_i would experience when using n ways instead of the full LLC. Note that an inter-task miss is detected when an LLC access hits in the SATD, but misses in the LLC. In contrast, the capacity hit is detected on an LLC hit and SATD miss.

Thus, as the basic idea behind the CPU accounting concept is to eliminate the redundant CPU cycles caused by inter-task misses and add those saved due to the use of additional LLC space [16]. We extend such method as follows: 1. If there is any capacity hit detected, we add the due cycles to the CPU cycles accounted to T_i. 2. If there is any inter-task miss detected, we decrease the due cycles in the CPU cycles accounted to T_i. The particular number of cycles to add on a capacity hit corresponds to the contention-free latency (MP_n) when running with n LLC ways, which we estimate as follows:

$$MP_n = MP_{WL} \times \frac{N_{T_i}}{N_{WL}} \tag{7}$$

where N_{WL} and N_{T_i} are the number of memory requests of the whole workload in the multicore system and of T_i respectively. MP_{WL} stands for the monitored miss penalty when T_i runs in the workload, since the memory latency that T_i experiences in the workload is affected by the memory requests from the co-running tasks.

In contrast, in the case of an inter-task miss, the number of cycles to subtract from the accounted cycles corresponds to the actual MP_{WL} monitored when the task runs in the workload.

Next, the time T_i would impose the banks into high-power consuming state depends on the activities performed. In general, several memory requests may be served in parallel to exploit memory bank level parallelism. Therefore, the time T_i activates the memory system does not correspond to its number of memory requests linearly since their miss penalties overlap. Thus, estimating IMC for T_i when it runs alone with n LLC ways precisely based on its behavior in the workload is virtually impossible. However, we observe that BPF is a relatively stable indicator to predict the variation between workload and isolation conditions. Thus, we estimate the BPF in isolation with n LLC ways (BPF_n) as follows:

$$BPF_n = BPF_{WL} - (1 - \frac{N^{MR}_n}{N^{MR}_N}) \tag{8}$$

where BPF_{WL} stands for the bank level parallelism factor that have been calculated during T_i's execution in the workload, and N^{MR}_n and N^{MR}_N correspond to the estimated number of memory requests when T_i has n LLC ways and the measured memory requests in the workload respectively. Then, we calculate the IMC of T_i as follows:

$$IMC_n = \frac{N^{MR}_n}{BPF_n} \times MP_n \tag{9}$$

Finally, we obtain the background energy accounted to T_i with n LLC ways as follows:

$$EA^{BG}_n(T_i) = P_P \times T^{Exe}_{i,n} + \frac{P_A \times DL_{12} + P_S \times DL_3}{DL_{123}} \times IMC_n \tag{10}$$

here we use $DL_{1,2,3}$ to represent the default latency of ACT, R/W and PRE commands together. DL_{12} stands for the default latency of ACT and R/W commands together. $T^{Exe}_{i,n}$ stands for the accounted execution time. Note that default latencies of R/W are often similar, so using read latency, write latency or any combination of both (e.g. average) does not bring any noticeable difference. In our case, we use read latency, which is marginally different to write latency.

c) **Refresh energy:** As the refresh energy is determined by the execution time and refresh power, we calculate it using Equation 3, but with the estimated execution time $T^{exe_i}_n$ for task T_i.

C. Putting it All Together

SEDEA has some hardware overhead, since it requires setting up a reduced set of additional counters and logic apart from the existing PMUs in most current architectures. However, most of those overheads are inherited from previous works. Such as SATD [13], [16], [21], which has been reported to introduce around 0.7% area overhead for the LLC. In addition to that, few registers and little logic are needed by DReAM [14] and ITCA [16], which need to be in place anyway for on-chip energy accounting. Besides, PTEM incurs 0.1% energy and area overhead for metering energy in LLC [15]. Thus, SEDEA reuses such hardware as it is.

TABLE I
SYSTEM CONFIGURATION

Main memory	
Frequency and size	1000MHz, 8GB
Technology and supply voltage	65nm, 1.2V
Row-buffer management policy	close-page
Address mapping scheme	Shared Bank
Chip details	
Core count	1, 4 and 8 cores, single-threaded
Instruction & Data L1	32KB, 4-way, 32B/line (2 cycles hit)
Instruction & Data *TLB*	256 entries fully-associative (1 cycle hit)
LLC Size	4MB, 16-way, 64B/line (3 cycles hit)

Specifically in this work, a 6-bit counter is needed for each task to record its number of in-flight memory requests per period, as well as MP and IMC registers to sample memory state and latencies. The estimated energy is stored in a register for each task, called Memory Energy Accounting Register (MEAR). Thus, the incurred area and energy overheads of SEDEA are negligible and do not affect performance.

Regarding the software interface, the OS is responsible for keeping track of the DRAM energy accounted to each task in the system. SEDEA exports its output through MEAR, which acts as interface between SEDEA and the OS. The OS can reset the MEAR (typically when a task is scheduled in) and access that register for collecting the energy accounted by SEDEA (typically when a task is scheduled out). In general, these actions happen during context switches. On the event of a context switch, the OS reads the MEAR using the hardware-thread index (or CPU index) of the scheduled out task T_{out}. Then, the OS aggregates the DRAM energy accounted value received in the *task struct* of T_{out}.

Integrating SEDEA with the SEA [13] improves the completeness of the SEA concept. Both mechanisms rely on the same input: the fraction of the resources that need to be accounted. Also, both mechanisms provide energy accounting estimates based on the same assumptions (the specific allocation of on-chip resources).

IV. EVALUATION

A. Experimental Setup

We use *DRAMsim2* [22] to model off-chip main memory, a cycle-accurate memory system simulator for DDR memories including a memory controller and DRAM memory. For the DRAM memory, we model an 8GB memory as it is large enough to support the workloads used in this paper. DRAM memory is single-rank with 8 devices per rank, 8 banks per device and 8 arrays per bank. Many current DRAM memories use low-power modes, in which the open banks under open-page policy quickly transition to power down state when there is no incoming request. Therefore, open-page policy performs similarly to close-page, at least in our workloads. Thus, we report results only for close-page row-buffer management policy, but conclusions hold also for open-page.

The multicore processor is modeled with *MPsim* performance simulator [1]. We consider three multicore processor configurations with 1, 4 and 8 single-threaded cores. The LLC is 4MB and shared among all the cores. To generate the reference set, we run each benchmark in isolation with each fraction of LLC, for instance, with 1-way 256KB LLC, 4-way 1MB LLC, etc. DRAMsim2 has been integrated with *MPsim* so that *LLC* misses are propagated to the memory controller, which manages those memory requests.

TABLE II
DRAM DEVICE CURRENT PARAMETERS USED IN DRAMSIM2 POWER MODEL

Current	Description	Value (mA)
IDD_{2P}	Precharge power-down current	10
IDD_{2N}	Precharge standby current	70
IDD_{3N}	Active standby current	90
IDD_{4W}	Operating burst write current	255
IDD_{4R}	Operating burst read current	230
IDD_{5}	Burst auto refresh current	305

A power model based on Micron current profiles [18] has been integrated in the simulation framework, which calculates the energy for all the activities and the power dissipated in every cycle.

The details of the system configuration are listed in Table I, and the main current profiles of the configuration we use in the power model are listed in Table II. We have implemented SEDEA on the described simulation infrastructure as it builds on hardware support currently unavailable in commercial off-the-shelf processors. Other works have followed an analogous methodology [4], [5].

We use traces collected from the whole SPEC CPU 2006 benchmark suite with the reference input set, using the SimPoint methodology [24]. Running all N-task combinations is infeasible since the number of combinations is too high. Therefore, we classify benchmarks into two groups depending on their memory access frequency. Benchmarks in the high-frequency group (denoted H) are those presenting a memory access frequency higher than 5 accesses per 1,000 cycles when running in isolation, and the rest of them in the low-frequency group (denoted L). From these two groups, we generate 3 workload types denoted L, H and X (combination of both), each consisting in 8 workloads generated randomly with benchmarks from the appropriate groups.

To evaluate the prediction accuracy of SEDEA, we use as reference the actual memory energy consumption of a benchmark when it runs alone with the corresponding LLC fraction. Hence, in each experiment, we measure the *prediction error* of each model with respect to the actual energy consumed when one task runs with the specified n-way LLC alone.

B. SEDEA Evaluation in Multicore Systems

We evaluate SEDEA in a 4-core architecture, with 24 randomly composed workloads using benchmarks of different LLC miss frequency level. We can observe in Figure 1 that SEDEA delivers stable prediction across all 1-16 ways of the LLC. The average prediction error across all benchmarks is relatively low, 6.5% on average and standard deviation under 13%, except for the case of 1-way. The reason that the deviation for 1-way is higher than others is because many

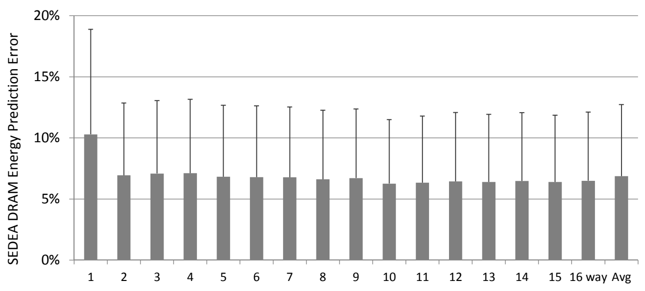

Fig. 1. Prediction error to account DRAM memory energy to benchmarks for different LLC ways allocated running in 4-core multicore system.

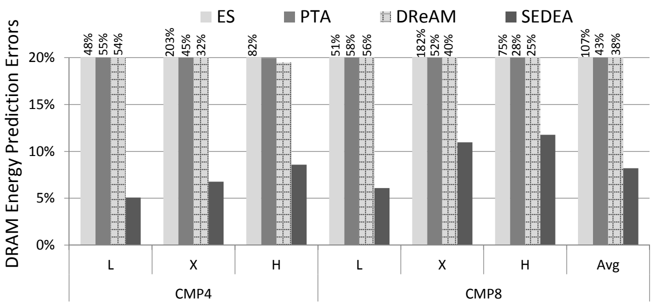

Fig. 2. Prediction error to account DRAM memory energy for workloads running in a 4-core processor using models: ES, PTA, DReAM and SEDEA.

benchmarks experience a drastically different number of LLC misses when only 1 LLC way is allocated. This huge variation translates into a significantly different memory energy profile that is, in turn, very hard to predict.

We evaluate and compare SEDEA in 4-core and 8-core scenarios, with Evenly Split (*ES*), Proportional To Access (*PTA*) and DReAM [14] in Figure 2. Note that in this figure, the outcome of *ES*, *PTA* and DReAM are compared with the cases where a task runs alone with a fair share of the LLC[1], and this is the only case they can be compared with. For example, for K tasks running in an N-way LLC, each task is given n LLC ways, where $n = N/K$. In contrast, SEDEA is capable to account energy for any LLC way count, achieving a much lower average error when predicting over $1-16$ ways.

As can be observed in Figure 2, *ES*, *PTA* and DReAM fail to sensibly account the memory energy to a task. This is expected as they lack any hardware support to estimate the behavior a task has in different scenarios. On average, *ES*, *PTA* and DReAM have prediction error over 38% across all setups. In contrast, SEDEA achieves an average 7.8% prediction error. In general, the predictions for M workloads and higher core-count scenarios are less accurate due to the higher interferences from co-running tasks, and thus harder discounting their effect. In general, SEDEA keeps inaccuracy low enough to make it usable in practice.

C. A Use Case of SEDEA

Apart from the main goal to provide a consistent energy metric, we also show a use case where some energy can be saved exploiting the synergy between cache partitioning (by ways) and memory partitioning (by banks) building upon SEDEA. We use a 4-core architecture, with a 4MB 16-way associative LLC and 8GB 8-bank memory system. We compose ten 4-task workloads for this experiment by using SPEC CPU 2006 benchmarks. We use the case when the LLC and memory are both shared as the reference to illustrate the potential of partitioning in energy savings.

[1]We consider even allocation of LLC ways across tasks as the fair share.

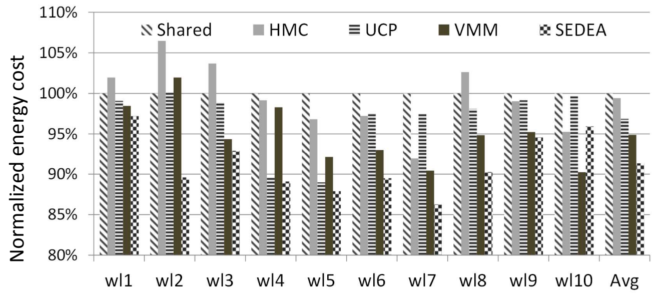

Fig. 3. The energy cost of 10 workloads for approaches HMC, UCP, VMM and SEDEA normalized to the energy cost of sharing cache and memory.

We compare our approach based on SEDEA against state-of-the-art mechanisms such as HMC, which groups the memory banks to reduce the bank-level conflicts and thus improve the throughput of the memory system when LLC is shared [27]; UCP, which partitions the LLC ways among tasks and lets tasks share the memory system [21]; Vertical Memory Management (VMM), which partitions the LLC and memory bank vertically, in which a task is given the same fraction of LLC ways and memory banks, depending on its profiled LLC behavior [11]. In the case of SEDEA, we have devised an algorithm to partition the cache and memory separately. We partition the LLC ways across tasks to minimize the overall memory energy based on SEDEA energy predictions. We also partition DRAM banks across tasks based on the BPF estimated as part of SEDEA. In particular, when the overall BPF is higher than the number of banks, we let tasks use a number of banks proportional to their individual BPF. Thus, the tasks with higher demand get larger bank counts whereas the tasks with lower BPF may share some banks.

Since the execution time of tasks varies across different partition methods, we meter the actual energy cost of each task in the chip and memory using PTEM [15] and DREAM [14]. Then, we calculate the energy per instruction of each task, and use their average value as the energy cost of the workload. We show the normalized energy savings (including processor and memory) of HMC, UCP, VMM and SEDEA in Figure 3 w.r.t. the case where cache and memory are fully shared. We can observe that SEDEA achieves higher energy savings than the other approaches in almost all workloads, reaching 8.7% energy savings on average. HMC, UCP and VMM save 0.6%, 3.2% and 5.2% energy, respectively. In our experiment, HMC does not perform well as a large fraction of the energy savings come from LLC miss reduction, and HMC only manages memory banks. Besides, its method for memory bank management, although avoids some conflicts, does not account for the bank requirements of tasks. In the case of UCP, the memory access conflicts from multiple tasks increase request duration, thus saving less energy. VMM partitions cache and memory vertically based on their needs. However, cache and memory partitions are correlated. Thus, SEDEA improves energy savings by exploiting independent partitions. Moreover, SEDEA also achieves higher performance improvement against these approaches, which further decreases leakage/-maintenance energy consumption.

V. Conclusion

Achieving a consistent energy accounting for tasks is of prominent importance to charge users based on their resource usage in datacenters. To the best of our knowledge, such a mechanism only exists for on-chip resources, but not for the memory system. In this paper, we pursue accounting a task the memory energy it would have consumed when it runs with a fraction of the LLC in isolation, but based only on the measurements obtained while running in a workload with shared resources. This requires discounting the effect of interference from co-runners and of using a different amount of hardware resources. We propose an ideal DRAM energy accounting model, as well as SEDEA, a low-cost yet accurate mechanism to account the memory energy to tasks based on estimating the memory behavior a task would have when running in isolation with a given fraction of the LLC. Our results show that SEDEA provides accurate estimates for memory energy under different cache allocations, beating alternative mechanisms, and it can help to save energy by optimizing cache and memory co-partitioning methods.

Acknowledgements

This work has been supported by the RoMoL ERC Advanced Grant (GA 321253) and National Key R&D Program of China under No.2016YFB1000204, by the European HiPEAC Network of Excellence, by the Spanish Ministry of Science and Innovation (contracts TIN2015-65316-P), by Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272) and by the IBM-BSC Deep Learning Center initiative. Also by the major scientific and technological project of Guangdong province (2014B010115003), and NSFC under grant no 61702495, 61672511. M. Moretó has been partially supported by the Ministry of Economy and Competitiveness under Juan de la Cierva postdoctoral fellowship number JCI-2012-15047. J. Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

References

[1] C. Acosta, F. J. Cazorla, A. Ramirez, and M. Valero, "The MPsim simulation tool," in UPC, Tech. Rep. UPC-DAC-RR-CAP-2009-15, 2009.

[2] W. Bircher and L. K. John, "Complete system power estimation using processor performance events," *IEEE Tran. on Comp.*, vol. 61, no. 4, pp. 563–577, 2012.

[3] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *USENIX ATC*, 2010, pp. 21–21.

[4] H. David, C. Fallin, E. Gorbatov, U. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *8th ACM ICAC*, 2011.

[5] Q. Deng, D. Meisner, L. Ramos, T. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," in *14th ASPLOS*, 2011.

[6] European Comission, *Energy prices and costs in Europe*, http://ec.europa.eu/energy/doc/2030/20140122_communication_energy_prices.pdf, 2016.

[7] JEDEC Solid State Technology Association, "JEDEC DDR3 SDRAM standard," https://www.jedec.org/standards-documents, 2012.

[8] K. Kedzierski, M. Moretó, F. J. Cazorla, and M. Valero, "Adapting cache partitioning algorithms to pseudo-lru replacement policies," in *IEEE IPDPS*, 2010, pp. 1–12.

[9] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *43rd IEEE/ACM MICRO*, Dec 2010, pp. 65–76.

[10] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *14th HPCA*, Feb 2008, pp. 367–378.

[11] L. Liu, Y. Li, Z. Cui, Y. Bao, M. Chen, and C. Wu, "Going vertical in memory management: Handling multiplicity by multi-policy," in *41st ISCA*, 2014.

[12] Q. Liu, V. Jimenez, M. Moreto, J. Abella, F. Cazorla, and M. Valero, "Per-task energy accounting in computing systems," *IEEE CAL*, vol. 13, no. 2, 2013.

[13] Q. Liu, M. Moreto, J. Abella, F. J. Cazorla, D. A. Jimenez, and M. Valero, "Sensible energy accounting with abstract metering for multicore systems," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 60:1–60:26, Dec. 2015.

[14] Q. Liu, M. Moreto, J. Abella, F. J. Cazorla, and M. Valero, "DReAM: An approach to estimate per-task dram energy in multicore systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 16:1–16:26, Nov. 2016.

[15] Q. Liu, M. Moreto, V. Jimenez, J. Abella, F. J. Cazorla, and M. Valero, "Hardware support for accurate per-task energy metering in multicore systems," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 34:1–34:27, Dec. 2013.

[16] C. Luque, M. Moreto, F. J. Cazorla, R. Gioiosa, A. Buyuktosunoglu, and M. Valero, "Cpu accounting for multicore processors," *IEEE Trans. Comp.*, vol. 161, 2012.

[17] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *USENIX ATATCC*, 2011, pp. 12–12.

[18] Micron, "Calculating memory system power for DDR3," 2007.

[19] M. Moreto, F. Cazorla, A. Ramirez, and M. Valero, "MLP-aware dynamic cache partitioning," in *3rd HiPEAC*, 2008, pp. 337–352.

[20] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *EuroSys*, 2011, pp. 153–168.

[21] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *39th MICRO*, 2006, pp. 423–432.

[22] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE CAL*, vol. 10, no. 1, pp. 16–19, Jan 2011.

[23] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: an os facility for fine-grained power and energy management on multicore servers," in *14th ASPLOS*. ACM, 2013, pp. 65–76.

[24] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *PACT*, 2001, pp. 3–14.

[25] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *38th ISCA*, 2011, pp. 283–294.

[26] X. Wang and J. F. Martłnez, "Xchange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures," in *21st HPCA*, Feb 2015, pp. 113–125.

[27] G. Zhang, H. Wang, X. Chen, S. Huang, and P. Li, "Heterogeneous multi-channel: Fine-grained DRAM control for both system performance and power efficiency," in *49th DAC*, ser. DAC '12, 2012, pp. 876–881.