A

DReAM: an Approach to Estimate Per-Task DRAM Energy in Multicore Systems

Qixiao Liu, Miquel Moreto, Jaume Abella, Francisco J. Cazorla, and Mateo Valero

Barcelona Supercomputing Center (BSC), Barcelona, Spain.
Universitat Politecnica de Catalunya (UPC), Barcelona, Spain.
Spanish National Research Council (IIIA-CSIC). Barcelona, Spain

Accurate per-task energy estimation in multicore systems would allow performing per-task energy-aware task scheduling and energy-aware billing in data centers, among other applications. Per-task energy estimation is challenged by the interaction between tasks in shared resources, which impacts tasks' energy consumption in uncontrolled ways. Some accurate mechanisms have been devised recently to estimate per-task energy consumed on-chip in multicores, but there is a lack of such mechanisms for DRAM memories. This paper makes the case for accurate per-task DRAM energy metering in multicores, which opens new paths to energy/performance optimizations. In particular, the contributions of this paper are (i) an ideal per-task energy metering model for DRAM memories; (ii) `DReAM`, an accurate, yet low cost, implementation of the ideal model (less than 5% accuracy error when 16 tasks share memory); and (iii) a comparison with standard methods (even distribution and access-count based) proving that `DReAM` is much more accurate than these other methods.

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, Energy Metering, Processor and Memory Architecture

Additional Key Words and Phrases: Multicore Architecture, Modeling and Simulation, Power Modeling, Performance, Benchmark Characterization

1. INTRODUCTION

The energy demand and cost of computing systems have grown during the last years, and the trend is expected to hold in the coming future [Beloglazov et al. 2011]. Conversely, computing hardware-related cost (e.g., servers) remains constant or even decreases in data centers, desktops and laptops. This leads to scenarios where energy cost is as significant as hardware-related cost. For instance, energy already accounts for 20% of the total cost of ownership in a large-scale computing facility [Hamilton 2009]. This cost virtually doubles if we also include the cost of the cooling infrastructures needed to dissipate the heat induced by such a high energy consumption. Therefore, energy-related cost is as relevant as the cost of servers in data centers. Similar examples can be found for home computers whose energy cost during their lifetime is in the same order of magnitude than the computer itself (e.g. 500$) and it is expected to grow due to the foreseen energy cost increase [Beloglazov et al. 2011].

As processor design moves towards multi-threaded and many-core processors, in which an increasing number of different applications run simultaneously in the same processor, providing per-task energy metering becomes critical. Accurately metering the energy consumed by each task would provide several benefits, including the following: First, the amount of hardware resources allocated to a given task (e.g., cores, memory space) impact both its execution time and energy consumption. If per-task energy can be accurately estimated, one may optimize not only each task's performance, but its energy consumption or a combined energy-delay metric; Second, per-task energy metering can be used by the operating system (OS) to better schedule tasks so that energy consumption is minimized while still completing tasks when needed; And third, traditionally, data centers charge users based on the resources they are allocated. The increasing fraction of energy-related costs in data centers and the need for

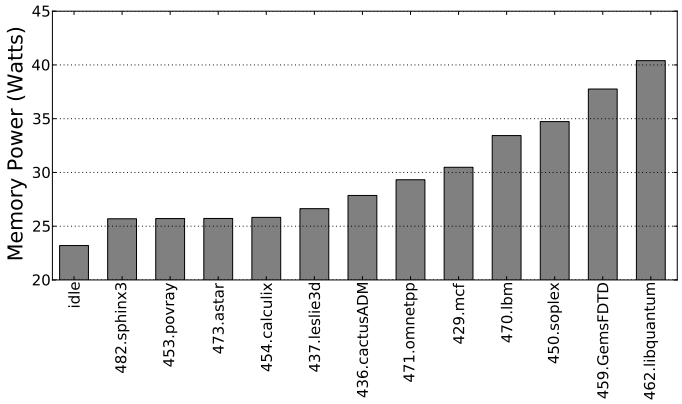

Fig. 1. Memory power of some SPEC CPU 2006 benchmarks running alone on an Intel Sandy Bridge server, with 8 cores and a 64GB DDR3 memory running at 1.6GHz. Power is obtained using the Running Average Power Limit (RAPL) interfaces [Intel Corp. 2012a]. FitPC external multimeter is used to measure wall power. We correlate wall power data with the data collected from the hardware energy counters using time stamps. Representative benchmarks were selected based on previous characterization studies [Phansalkar et al. 2007; Jaleel 2007].

more accurate billing pushes for new billing approaches based on the actual energy consumption of each task rather than on the nominal resources allocated or on simply distributing energy evenly among the running applications [Jimenez et al. 2011].

While energy can be easily estimated or measured in systems with no shared resources (e.g., single-core processors), the advent of multicores challenges accurate per-task energy metering due to shared resources. Some efforts have been done to split energy across hardware components (e.g., cores, caches, memory) and, to understand how on-chip hardware resources are shared [Bircher and John 2007; McCullough et al. 2011; David et al. 2010]. Those proposals rely on the use of performance monitoring counters (PMCs) or system events (such as OS system calls).

In that respect, despite memory power keeps increasing, reaching 30-50W in high-performance computers [Bircher and John 2007; David et al. 2011], there is a lack of understanding of how energy is consumed per-task in memory [Aggarwal et al. 2008]. This is aggravated by the fact that memory power profiles across tasks may vary significantly. For instance, a variation of up to 36% in memory power consumption is observed across different SPEC CPU 2000 workloads (from 33.9W to 46.4W) when running 4 instances of the same benchmark in each workload [Bircher and John 2007].

To elaborate on the need of accurate per-task memory energy metering, we perform an experiment with several representative SPEC CPU 2006 benchmarks running on an Intel Sandy Bridge server. In our experiments, memory represents between 24.6% and 33.9% of the total wall power. It is comparable to the entire processor socket power: on average, DRAM memory only consumes 6.3% less power than the processor. Thus, memory power accounts for a significant portion of the total power consumption in modern computing systems. Figure 1 shows the average memory power consumption of each benchmark when executing in isolation on the system. Different tasks incur different power consumption, with the maximum variation being 54%, between `482.sphinx3` and `462.libquantum` (from 25.7W to 40.4W). Hence, libquantum-like and sphinx3-like workloads executing for the same amount of time would incur significantly different energy consumption. However, to the best of our knowledge, no mechanism has been proposed to measure accurately the memory energy consumed by each task in multicore architectures.

This paper proposes, for the first time, an ideal method to fairly distribute the energy consumed in DRAM memories to concurrent running tasks and an efficient implementation of such method. Our approach relies on tracking both, the activity incurred by running tasks and the memory bank states they induce. Then, energy is attributed fairly to tasks based on their memory behavior. We show that an accurate, yet low cost, implementation of the ideal model is feasible. Overall, the contributions of this work are as follows:

— We propose an ideal per-task energy metering model for DRAM memories, including those based on close-page and open-page policies, as needed for performance/energy optimization, task scheduling and billing in multicore systems. To the best of our knowledge, it is the first reference model against which per-task energy metering mechanisms in DRAM memories can be compared to.
— We devise `DReAM`, an accurate, yet low cost, implementation of the ideal model. `DReAM` requires few counters and registers to be set up in the memory controller to gather the required information. Our results show that such implementation is within a 5% average error with respect to the ideal model.
— We compare `DReAM` with two other energy metering approaches: (i) Evenly Splitting (ES) energy across co-running tasks, and (ii) splitting energy Proportionally To memory Accesses (PTA), a simplified `DReAM` method that further trades accuracy and cost. Our results show that `DReAM` is far more accurate than ES and PTA with negligible hardware overhead.
— We characterize the SPEC CPU 2006 benchmark suite in terms of DRAM energy consumption. Our characterization allows identifying those properties of the applications that impact DRAM energy consumption the most, so that suitable scheduling algorithms can be devised.

The rest of this paper is organized as follows. Section 2 provides background on memory energy consumption and existing approaches for energy metering. Section 3 presents our approach to perform ideal per-task memory energy metering. `DReAM`, our efficient hardware implementation of the ideal model, is described in Section 4. Next, `DReAM` accuracy is evaluated in Section 5. In Section 6 we use `DReAM` to characterize DRAM energy consumption of the SPEC CPU 2006 benchmark suite. Finally, Section 7 draws the main conclusions of this work.

2. BACKGROUND AND RELATED WORK

2.1. Previous Energy Metering Approaches

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 Corp. 2007; Chung et al. 2011]. In previous proposals, however, the focus is on providing accurate energy metering for single-core architectures or multicore architectures in which a single (multi-threaded) application is executed. These scenarios are relatively easy to handle since, when an application is scheduled on the CPU, it accounts for the whole energy consumption of the system (e.g., using a simple meter). Many proposals [Bircher and John 2007; McCullough et al. 2011; David et al. 2010] use PMCs or system events (such as OS system calls) to break down the energy consumption of the system across its components (e.g., memory, processor, etc.). Those power models use a set of PMCs and predefined weights derived through correlation. In many cases, the results of the power model are compared against approaches using circuit-based mechanisms such as current sense resistors. Some Intel servers model DRAM power

per channel, but they are unaware of per-task interactions in each channel as well as DRAM bank state interactions across requests [Intel Corp. 2012b].

However, with the increasing number of computing cores in processor architectures, managing shared hardware resources on-chip becomes challenging [Nesbit et al. 2008]. Furthermore, the number and heterogeneity of the tasks that will coexist in a computing system will significantly increase. In this evolving scenario, it is of prominent importance performing accurate per-task energy *metering* and *accounting* [Liu et al. 2013b]. Given a workload composed by n tasks $T_1, T_2, \ldots, T_n$ running in a system with n cores, *per-task energy metering* consists in tracking the energy that a given task, T_i, consumes during a given period of time. *Per-task energy accounting* 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. Since energy accounting builds upon energy metering [Liu et al. 2013b], per-task energy metering is the first challenge to address.

Recently, Shen et al. [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. Similarly, Kestor et al. [Kestor et al. 2013] estimate the energy of moving data along the memory hierarchy by designing a set of micro-benchmarks. However, both approaches cannot take into account the impact of inter-task interference unless appropriate solutions provide accurate per-task energy metering in multicores, as stated by the authors in [Shen et al. 2013]. Liu et al. [Liu et al. 2013a] have recently provided the hardware layer that delivers accurate Per-Task Energy Metering (PTEM) for on-chip resources (cores, caches, etc.), attributing the energy to running tasks according to their hardware utilization. Our proposal in this paper, `DReAM`, complements PTEM providing such support for DRAM memories, thus delivering the hardware support needed by [Shen et al. 2013; Kestor et al. 2013]. In particular, we make the first proposal of (i) an idealized reference per-task memory energy metering model and (ii) the hardware support to accurately measure per-task memory energy consumption in multicores with multiple tasks executing concurrently.

2.2. Breaking Down Energy Consumption

DRAM memory energy variation across workloads can be large [Bircher and John 2007; David et al. 2011] and is likely to increase in the future as system manufacturers pay increasing attention to energy efficiency [Barroso and Holzle 2007]. We break DRAM memory energy consumption down into three components: dynamic, refresh and background.

Dynamic energy corresponds 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 or the termination power due to terminating signals of other ranks on the same channel.

Refresh energy corresponds to the energy consumed to refresh periodically all memory contents. Unlike SRAM memory cells, DRAM cells are unable to retain contents indefinitely. Instead, DRAM cells discharge over time and eventually, they lose their contents. Therefore, they must be read and written back at a given minimum frequency to keep their contents. Although this has some implications in energy consumption (to read/write memory contents) and bandwidth (refresh operations may delay program's accesses), DRAM cells are smaller and less power-hungry than SRAM ones, so they are used to implement main memory.

Background energy includes maintenance and leakage energy. Maintenance energy corresponds to the energy consumed due to *useless* activities not triggered by the program(s) being run. For instance, DRAM memory may stay in a higher energy consump-

tion state during idle cycles so that it can quickly react and serve a new access. Alternatively, it may remain in a much lower power mode with lower maintenance power dissipation, but it may take longer to serve a new access due to the time required to transition to an active mode. Leakage energy corresponds to the energy wasted due to imperfections of the technology used to implement the circuit. Note that if circuits are 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]. For the sake of clarity, we make use of the term background energy to refer to all energy consumed except dynamic and refresh energy.

3. METERING PER-TASK ENERGY CONSUMPTION

In this section we present an idealized model for per-task DRAM energy metering without considering hardware cost. The result of this model is later used as the reference for `DReAM` model to meter per-task energy with a low-cost implementation. We assume a multicore architecture where an on-chip memory controller serves as the bridge to the off-chip memory. Next we describe the memory model considered in this paper, how energy is consumed in the different memory blocks, and our models to split energy among different tasks.

3.1. Memory Model

We focus on DDRx SDRAM as it is one of the most common memory technologies. A DDRx SDRAM memory system is composed by a memory controller and one or more DRAM devices. The memory controller controls the off-chip memory system acting as the interface between the processor and DRAM devices.

A memory *rank* consists of multiple devices, which in turn consist of multiple banks that can be accessed independently. Each bank comprises rows and columns of DRAM cells (organized in arrays) and a row-buffer to cache the most recently accessed rows in the bank. Rows are loaded into the row-buffer using a row activate command (ACT). Such command opens the row by moving the data from the DRAM cells to the row-buffer sense amplifiers. Once a bank is open, any read/write operation (R/W) can be issued. Finally, a precharge command (PRE) closes the row-buffer, storing the data back into the row. The memory controller can use two different policies to manage the row-buffer: close-page that precharges the rows immediately after every access, and open-page that leaves the rows in the row-buffer open for potential future accesses to the same rows. The memory controller uses a First Ready, First Come First Serve (FR-FCFS) policy. In this policy, all memory requests arriving from all cores are buffered in arrival order in the memory controller, and dispatched to the memory system depending on the states of the banks they access.

Different models can be adopted to access memory. Those models determine which ranks, devices, banks and arrays are accessed on each operation. We adopt the same model as *DRAMsim2*, which in turn models Micron DDR2/3 memories [Rosenfeld et al. 2011]. In this model, all devices in a rank are accessed upon every access. In each device, all arrays of exactly one bank are accessed. Each array provides the specified row to the sense amplifier on every access, where a number of contiguous columns are accessed over successive cycles to serve an incoming access. In our model, we use a single rank, 8 devices per rank, 8 banks per device and 8 arrays per bank. In one cycle, one bank per device is accessed, thus providing 64 bits in total for the rank. A burst of 8 cycles provides 64 bytes on every access to memory, therefore matching the cache line size for the last level cache (LLC) in the processor. In our model, each rank connects to the memory controller through a logically independent channel. In each channel, the commands are ordered sequentially, and the data transfer occurs as determined by the burst length timing parameter.

Table I. Memory commands, timing, power states and background power breakdown for a read operation in close-page mode.

Command	T_0	–	ACT		READ	PRE	–
	T_1	–					
Timing	T_0	–	t_{XP}	t_{RCD}	t_{RTP}	t_{RP}	–
State	$Bank_0$	PD	S	A		S	PD
	$Bank_1$			S			
	$Bank_2$						
	$Bank_3$						
Power	Rank	P_{PD}	P_S	P_A		P_S	P_{PD}
	T_0	$\frac{P_{PD}}{2}$	$P_S - \frac{P_{PD}}{2}$	$P_A - \frac{P_{PD}}{2}$		$P_S - \frac{P_{PD}}{2}$	$\frac{P_{PD}}{2}$
	T_1	$\frac{P_{PD}}{2}$					

Under this configuration, all devices are always in the same power state, which is equivalent to consider the power state at rank level. In each device, those banks being accessed – if any – can be in a higher-power state.

We build our model upon Micron DDR2/3 power model, which provides temperature-independent data. However, in practice energy consumption can be affected by temperature. If such temperature-dependent data were available, per-temperature-range energy constants should be used accordingly as already pointed out by Liu et al. [Liu et al. 2013a] to track per-task energy consumption considering temperature variations.

3.2. Memory Energy Consumption

The energy model for the main memory is based on the current profiles provided by Micron [Micron 2007] and it splits energy consumption into dynamic, refresh and background energy. This is analogous to the methodology used in [Deng et al. 2011], where the same data from Micron is used as input. Micron energy model determines the background electric current level, and so the background power dissipation of each rank, similarly to the methodology introduced in [Vogelsang 2010]. Devices can be in three different states: Power Down (*PD*), Standby (*S*), and Active (*A*). In each state, power dissipation is P_{PD}, P_S and P_A, respectively. *PD* state is the one with the lowest power dissipation. Note that *PD* refers to disabling the clocking in the memory system. This process, which may take several cycles, is carried out by the memory controller. However, memory contents are preserved at all times. Also the Micron model determines the electric current level caused by each command, and so their energy, except for the ACT and PRE commands, whose energy is not segregated. We have used a similar approach to the one introduced in [Chandrasekar et al. 2011] to separate the energy of those commands when needed, for example, under open-page policy.

Table I shows the effect on memory of a read command. We observe that the device is in *PD* state when the memory controller is not processing any request. Note that in our configuration all devices in the rank are in the same state and therefore, rank and device states match. When the memory controller receives a memory access request from task 0 (T_0), it sends a clock enable (*CKE*) signal to transition the rank from *PD* to *S* state. The device stays in S state as long as all banks are powered up and idle. This includes the time the device is waiting for the memory controller to send those commands corresponding to the requests in the memory controller's queues. During the *S* state, background power is higher than in *PD* state ($P_S > P_{PD}$). *S* state lasts t_{XP}, as depicted in Table I. Eventually, some banks are activated so that the device as well as some banks transition to *A* state. Note that in this model, when the ACT command is issued the device (and so the rank) switches to *S* state, and whenever the corresponding bank has been activated, the device switches to *A* state. The device and the accessed banks (Bank_0 in the example) are in *A* state during part of the activation

period (t_{RCD}) and while the read/write command is served (t_{RTP} in the example for a read command). Note that there is another timing constraint: each bank can only be precharged after t_{RAS}. Therefore, in the case when $t_{RAS} > (t_{RCD} + t_{RTP})$, the bank stays in *A* state at least for t_{RAS} after being activated. While in *A* state, the device incurs the highest power dissipation, P_A, with $P_A > P_S$. Once the only command being processed is the *PRE* command, the device and accessed banks transition to *S* state. When no command is executed and no memory access request exists in the memory controller buffer for a certain time interval, the memory controller returns the device to *PD* state.

Most modern memory controllers implement open-page and/or close-page policies. They differ on how the data array row-buffer is managed (for how long the row-buffer keeps open). Next, we present how per-task energy is metered under both policies.

3.3. Per-Task Energy Metering for Close-Page

Our idealized model relies on the fact that background power dissipation of a device depends solely on its current state, which can be induced by different, concurrent accesses. Therefore, our model attributes background energy to each task based on the state it imposes on memory. As reported in [Deng et al. 2011], background energy accounts for over 50% of the memory energy consumed by programs. Memory occupancy is discarded as input for the model since background energy does not depend on it. Thus, distributing background energy according to resource utilization is crucial to meter per-task memory energy accurately.

(1) During *PD*, only background power is consumed. Such energy is constantly consumed during a given period of time, independent from the percentage of capacity used by a task. If a task runs alone, all this power is assigned to it. Thus, when running multiple tasks, the powerdown background power cannot be attributed to any specific task since no task has any memory activity during *PD*. Hence, we divide background power evenly across all tasks running in the system.
(2) Whenever a DRAM device switches from *PD* to *S* state, the extra background power incurred due to *S* state (i.e. $P_S - P_{PD}$) is distributed uniformly across all tasks with in-flight commands that force the DRAM devices to stay in *S* state.
(3) When a DRAM device is in *A* state (active), the extra power incurred (i.e. $P_A - P_S$) is distributed evenly across all tasks enforcing *A* state.

For instance, Table I shows the case where one task, T_0, issues a *read* command (first row) while another task, T_1, issues no command. Let us assume that those are the only tasks using the memory system. During the whole period, T_1 is responsible only for half of the P_{PD} power (last row), while T_0 is responsible for half of the P_{PD} and all P_S and P_A extra power (penultimate row).

When multiple commands are processed in parallel, we follow the same principle of attributing power to those tasks that impose the memory chip to be on a given state. In the example in Table II, we show a particular case where both T_0 and T_1 issue commands in parallel. First, the device is in *PD* state. Eventually, T_0 makes the device transition to *S*, so T_0 is responsible for the extra background power. Then, the device transitions to *A* state and T_1 starts its activate command. Both tasks are equally responsible for P_{PD} and P_S power, but only T_0 is responsible for P_A power. Later, T_1 also enforces memory to be in *A* state so that the total power must be uniformly distributed across both tasks. Finally, as commands finish, tasks T_0 and T_1 stop enforcing high-power states and power dissipation is attributed only to those tasks imposing each particular state.

Regarding refresh operations, according to the JEDEC standard of DDR2/3 SDRAM memory [JEDEC Solid State Technology Association 2012], it is required to issue eight

Table II. Memory commands, timing, power states and background power breakdown for several operations in close-page mode.

Comm.	T_0	–	ACT		READ	PRE		–	
	T_1	–		ACT		READ	PRE		–
Timing	T_0	–	t_{XP}	t_{RCD}	t_{RTP}	t_{RP}		–	
	T_1	–		t_{RRD}	t_{RCD}	t_{RTP}	t_{RP}		...
State	$Bank_0$	PD	S	A		S	S		PD
	$Bank_1$			S	A				
	$Bank_2$				S				
	$Bank_3$								
Power	Rank	P_{PD}	P_S	P_A			P_S		P_{PD}
	T_0	$\frac{P_{PD}}{2}$	$P_S - \frac{P_{PD}}{2}$	$P_A - \frac{P_S}{2}$	$\frac{P_A}{2}$	$\frac{P_S}{2}$		$\frac{P_{PD}}{2}$	
	T_1	$\frac{P_{PD}}{2}$		$\frac{P_S}{2}$	$\frac{P_A}{2}$	$P_A - \frac{P_S}{2}$	$\frac{P_S}{2}$	$P_S - \frac{P_{PD}}{2}$	$\frac{P_{PD}}{2}$

Table III. Memory commands, timing, power states and background energy breakdown when a hit in the row buffer occurs in open-page mode.

Comm.	T_0	–	ACT		READ	–	–	
	T_1	–				–	READ	–
Timing	T_0	–	t_{XP}	t_{RCD}	t_{RTP}	–		
	T_1	–					t_{RTP}	–
State	$Bank_0$	PD	S	A				
	$Bank_1$		S					
	$Bank_2$							
	$Bank_3$							
Power	Rank	P_{PD}	P_S	P_A				
	T_0	$\frac{P_{PD}}{2}$	$P_S - \frac{P_{PD}}{2}$	$P_A - \frac{P_{PD}}{2}$		$\frac{P_{PD}}{2}$		
	T_1	$\frac{P_{PD}}{2}$				$P_A - \frac{P_{PD}}{2}$		

refresh commands during a given time window. Thus, the memory controller has some flexibility to schedule those refresh commands minimizing interference on tasks' commands. The refresh energy is guaranteed to be constant in the memory system during a given period of time, regardless of the activities of running tasks. Given that refresh commands occur in all banks simultaneously, they cannot happen in parallel with any other command. Thus, both dynamic and background energy incurred during refresh is accounted as refresh energy. Although refresh energy is not triggered by the execution of tasks, it is consumed as long as the system is powered up. Thus, tasks running in the system are assumed to be responsible for the system being up, and so refresh energy is evenly split across those tasks.

3.4. Per-Task Energy Metering for Open-Page

As opposed to the close-page policy, in open-page, ACT/PRE commands may not be needed by a memory access, since banks remain open after being accessed. However, energy consumed by open banks is still attributed to those tasks that opened the banks. Regarding background energy, the same principle as for close-page is followed: attributing the energy to tasks based on the state they impose to memory.

As in close-page policy, devices are powered up and activated (A state) to execute commands. However, once the corresponding read/write operation finishes, those devices remain open in A state. This is illustrated in the example in Table III that reflects the case of a row-buffer hit. The task that opened the bank (T_0 in the example) is responsible for the extra background energy of the activated devices (after the first t_{RTP}). Eventually, another access to the open banks can occur. If this is the case, no precharge command is needed. Since T_1 read access is a row-buffer hit, it can directly read data

Table IV. Memory commands, timing, power states and background energy breakdown for multiple interleaved accesses from two tasks accessing the same bank in open-page mode.

Comm.	T_0	READ	–	READ	–	PRE
	T_1	–	READ	–		
Timing	T_0	t_{RTP}	–	t_{RTP}	–	t_{RP}
	T_1	–	t_{RTP}	–		
State	$Bank_0$	A				S
	$Bank_1$	S				
	$Bank_2$	S				
	$Bank_3$	S				
Power	Rank	P_A				P_S
	T_0	$P_A - \frac{P_{PD}}{2}$	$\frac{P_{PD}}{2}$	$P_A - \frac{P_{PD}}{2}$		$P_S - \frac{P_{PD}}{2}$
	T_1	$\frac{P_{PD}}{2}$	$P_A - \frac{P_{PD}}{2}$	$\frac{P_{PD}}{2}$		

from the row buffer. Consequently, T_1 becomes responsible for the extra background energy, while T_0 is only responsible for half of the PD energy.

Analogously, the same principle also applies when multiple accesses are interleaved, as shown in Table IV. In this particular case, T_0 has already opened one bank ($Bank_0$), which imposes the A state to the rank and the corresponding bank. Eventually, T_1 accesses the same rows which incurs a row-buffer hit. During this process, the extra background energy attribution switches like in the previous example. Then, after T_1 finishes its operation, T_0 accesses the same rows which incurs another row-buffer hit. Thus, the attribution of extra background energy switches back to T_0 again. Whenever the page is closed, T_0 is also responsible for the precharging dynamic energy, which should have been attributed to T_1 if T_0 had not accessed the open bank. The main reason why we distribute the extra background energy this way is that, when the bank is firstly opened, it is impossible to predict its future accesses, thus the activation energy is attributed to the first user. Similarly, the precharging energy is attributed to the last user, who triggered the PRE command. Regarding background energy, we also assume that the last task imposing a particular device state accounts for the extra energy. Although our choice is, to some extent, arbitrary, we regard it as fair.

In summary, activate and read/write dynamic energy is attributed to the task performing the access, whereas precharge energy is attributed to the last task accessing such row. Note that on a refresh command all banks need to be closed, and so precharge energy for open pages is attributed to the last task accessing each of them. Other than that, energy distribution is analogous for close-page and open-page policies.

3.5. Ideal Per-Task Energy Metering Model

We generalize the memory energy consumed by each task as follows.

1) The background (bg) energy attributed to a task can be generalized as follows for both open- and close-page policies:

$$E_{bg}^{mem}(T_i) = P_{PD} \times ExecTime(T_i)/N_T + \sum_{j=0}^{ExecTime(T_i)} \left((P_S - P_{PD}) \times \frac{\delta_{i,j}^S}{N_{S,j}^T} \right) + \sum_{j=0}^{ExecTime(T_i)} \left((P_A - P_S) \times \frac{\delta_{i,j}^A}{N_{A,j}^T} \right) \quad (1)$$

In the first addend each running task is metered an even part of P_{PD}, where $ExecTime(T_i)$ stands for the execution time of task i in cycles and N_T for the number of tasks running in the processor – not necessarily the maximum number of tasks allowed in the processor. The second and third addends meter $P_S - P_{PD}$ and $P_A - P_S$

for tasks enforcing those states. $N^T_{S,j}$ and $N^T_{A,j}$ correspond to the number of tasks imposing S and A states respectively in cycle j; and $\delta^S_{i,j}$ and $\delta^A_{i,j}$ indicate if the task i makes memory be in S and A state respectively, in cycle j. In other words, for instance, for close-page $\delta^A_{i,j}$ is 1 if task i is executing a *read*, *write* or *activate* (last t_{RCD} cycles) command in cycle j, and 0 otherwise; and $\delta^S_{i,j}$ is 1 if task i is executing a *precharge* or activate (first t_{XP} cycles) command or if it has pending commands in the memory controller while all banks are idle in cycle j, and 0 otherwise. Note that, as stated before, memory occupancy is not considered for metering energy to tasks since the memory regions not used by the task under consideration cannot be turned off when idle. Hence, background power remains the same regardless of the memory space used.

2) Dynamic energy for a task depends on the number of commands it performs, as shown in the following equation:

$$\begin{aligned} E^{mem}_{dyn}(T_i) = E^{mem}_{read} \times N_{RD}(T_i) + E^{mem}_{write} \times N_{WR}(T_i) \\ + E^{mem}_{ACT} \times N_{ACT}(T_i) + E^{mem}_{PRE} \times N_{PRE}(T_i) \end{aligned} \tag{2}$$

where E^{mem}_{read}, E^{mem}_{write}, E^{mem}_{ACT} and E^{mem}_{PRE} stand for the energy of each command, and $N_{RD}(T_i)$, $N_{WR}(T_i)$, $N_{ACT}(T_i)$ and $N_{PRE}(T_i)$ stand for the number of memory internal commands executed by task i.

3) *Refresh* operations may have some side effects such as delaying some commands issued by running tasks. However, this fact does not alter the energy model. Also, refresh commands consume some energy to access the corresponding rows. Since refresh operations are distributed evenly over time at a fixed rate and they are not originated by any particular task, their energy is evenly split across all running tasks. Thus, refresh energy per task is as follows:

$$E^{mem}_{refr}(T_i) = E^{mem}_{refr} \times N_{Ref} \times ExecTime(T_i)/N_T \tag{3}$$

where E^{mem}_{refr} corresponds to the dynamic and background energy of a refresh command. N_{Ref} corresponds to the average number of refresh operations performed per cycle.

4. DREAM, A PRACTICAL APPROACH TO PER-TASK ENERGY METERING

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, the frequency at which they must be tracked, and the lack of information that the processor has about the memory state. On the other hand, metering memory energy evenly among running tasks or proportionally to the number of accesses that they perform requires minor changes to current architectures. However, these approaches exhibit low estimation accuracy as shown later in Section 5.2. Therefore, we propose DReAM, our per-task energy metering approach that trades off energy metering accuracy and implementation complexity.

In DReAM memory model, dynamic and refresh energy can be easily tracked as in the idealized model. This requires the memory vendor to provide the dynamic energy per access type, namely E^{mem}_{read}, E^{mem}_{write}, E^{mem}_{ACT} and E^{mem}_{PRE} for tracking dynamic energy and E^{mem}_{refr} for tracking refresh energy, as well as the average number of refresh operations per cycle (N_{Ref}). These parameters are already provided by chip vendors like Micron for DDR2/3 memories [Micron 2007], so our model imposes no change to current DDR2/3 memories. In the memory controller, we only require per-task activity counters, namely $N_{RD}(T_i)$, $N_{WR}(T_i)$, $N_{ACT}(T_i)$ and $N_{PRE}(T_i)$. Total background en-

Table V. DReAM hardware requirements.

Block	Memory Vendor	Extra Logic
Memory	E^{mem}_{read}, E^{mem}_{write}, E^{mem}_{ACT}, E^{mem}_{PRE}, E^{mem}_{PD}, E^{mem}_{refr}, N_{Ref}	N_{RD}, N_{WR}, N_{ACT}, N_{PRE}, $N_{RD}(T_i)$, $N_{WR}(T_i)$, $N_{ACT}(T_i)$, $N_{PRE}(T_i)$, $IntMem$ cycle counter

ergy, $E^{mem}_{bg,total}$ can be obtained by metering memory energy consumption [David et al. 2010] and subtracting dynamic and refresh energy. The PD background power is constant and hence easy to track. The remaining background energy, E^{mem}_{rem}, is due to active and standby periods (i.e. $E^{mem}_{bg,total}$ = E^{mem}_{PD} + E^{mem}_{rem}).

Our model distributes E^{mem}_{PD} uniformly across all tasks, while E^{mem}_{rem} is distributed based on access frequencies per task. To that end, we divide the execution into intervals of *IntMem* processor cycles and track the number of memory accesses sent to the memory controller (in a per-task basis) in the current interval. Thus, background energy is obtained as follows:

$$E^{mem}_{bg,\ total}(T_i) = \frac{P^{mem}_{PD} \times ExecTime(T_i)}{N_T} + \sum_{j=0}^{\frac{ExecTime(T_i)}{IntMem}} \frac{N^{T_i}_{acc,j}}{N^{TOTacc}_{j}} \times E^{mem}_{rem,j} \quad (4)$$

where P^{mem}_{PD} is the *PD* background power, $N^{T_i}_{acc,j}$ tracks the number of memory accesses of task i during interval j, and N^{TOTacc}_{j} tracks the total number of memory accesses in interval j. $E^{mem}_{rem,j}$ is the non-power-down background energy in interval j, obtained by subtracting all other sources of energy consumption from the total energy measured in the interval. Sensitivity to the sampling interval ($IntMem$) is studied in the evaluation section.

Putting it All Together

DReAM requires little hardware overhead, since DReAM mostly requires setting up a reduced set of counters similar to the PMCs currently available in most high-performance processors. DReAM support does not interfere the execution of programs since it is not in any critical path. Table V summarizes those parameters required from the memory vendor and the extra logic (i.e. counters) that must be set up. Counters with the "(T_i)" suffix must be replicated for each task. Thus, the number of required counters is dictated by the number of tasks that run simultaneously in the chip.

Regarding the interface with the software, the OS is responsible for keeping track of the energy consumed by every task running in the system. DReAM exports a special register, called Memory Energy Metering Register (MEMR), that acts as the interface between DReAM and the OS. The OS can access that register to collect the energy estimates made by DReAM. This typically happens when a context switch takes place. At that moment, the OS reads the MEMR using the hardware-thread index (or CPU index) for the task that is being scheduled out (T_{out}). Then, the OS aggregates the energy consumption value read in the *task struct* for T_{out}. Right after the new task (T_{in}) is scheduled in, the memory state may remain at a particular state due to an access triggered by the task that has been scheduled out. Although, DReAM attributes background energy consumption to T_{in}, this occurs during few cycles (in the order of tens or hundreds of cycles). Under a processor frequency of 2GHz, 500 cycles are equivalent to $0.25\mu s$, while context switches occur at much higher granularity, every 10-100ms.

As in [Liu et al. 2013a], the time the OS spends working on behalf of a given task is attributed to the calling task. The remaining energy consumed by the OS can be

Table VI. System Configuration.

Main memory	
Size	8GB
Frequency	933MHz
Row-buffer policy	Close-page or open-page
Address mapping scheme	Shared bank
Power-down mode	Fast
Supply voltage	1.35V
Technology	65nm
Core details	
Core count	1, 4, 16 cores, single-threaded
Fetch, decode, issue, commit bandwidth	2 instructions/cycle
Issue queues size	32/32/32 entries for INT/FP/Load-store queues
Register file	80 INT, 80 FP
Instruction & data L1	32KB, 4-way, 32B/line (2 cycles hit)
Instruction & data TLB	256 entries fully-associative (1 cycle hit)
Last-level Cache (LLC)	
Size	256KB/core 256KB (1 core), 1MB (4 cores), 4MB (16 cores)
Other parameters	16-way, 64B/line (3 cycles hit)

evenly attributed to all running tasks. In any case, DReAM provides the hardware support needed to attribute OS energy to tasks as required.

5. EVALUATION

5.1. Experimental Setup

We use MPsim [Acosta et al. 2009], an enhanced version of SMTSim [Tullsen et al. 1995] to model the processor. Off-chip main memory is modeled with DRAMsim2 [Rosenfeld et al. 2011], a cycle-accurate memory system simulator for DDR3 memories including a memory controller and DRAM memory. DRAMsim2 has been connected to MPsim so that last level cache (LLC) misses are propagated to the memory controller, which manages those memory requests. DRAMsim2 implements a power model based on Micron memories.

We consider three CMP processor configurations with 1, 4 and 16 single-threaded cores. The LLC is partitioned with 256KB 16-way per core. Therefore, the LLC size is 256KB, 1MB, and 4MB for 1, 4, and 16 cores, respectively. These configurations have been chosen to discount the effect of on-chip inter-task interferences due to shared resources (e.g., shared LLC cache), thus allowing to consider the effects of the interferences within the memory system only [Aggarwal et al. 2008]. Details about the configuration can be found in Table VI.

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. We have evaluated close-page and open-page DRAM memory row-buffer management policies, but differences are negligible: since many current DRAM memories have a low-power mode, the open banks under open-page policy quickly transition to power down state when there is no incoming request. In this case, open-page policy performs similarly to close-page in most of the cases. Thus, we only report results for one of the policies: close-page.

Average power consumption for the 8GB setup is 5.4W, 8.6W and 18.8W for 1-thread, 4-thread and 16-thread workloads respectively. For a setup of 64GB (results not shown in this paper) power increases by a 2x-3x factor (e.g., 14.7W for 1-thread workloads). Note that this is around half the power consumption reported in Section 1, which is

consistent since our setup is less aggressive than that of the particular server used in the real experiment. In particular, we assume a processor operating at 2GHz and DRAM operating at 1GHz, whereas the CPU of the server used operates at 3.2GHz and its memory at 1.6GHz. Nevertheless, our proposal is orthogonal to those parameters.

5.1.1. ***Benchmarks.*** We use traces collected from the whole SPEC CPU 2006 benchmark suite using the reference input set [Henning 2006]. Each trace contains 100 million instructions, selected using the SimPoint methodology [Sherwood et al. 2001]. Using these benchmarks, we generate different workloads with different number of benchmarks. Running all N-task combinations is infeasible as the number of combinations is too high. Hence, we classify benchmarks into two groups depending on their memory 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, that is: *mcf*, *milc*, *lbm*, *libquantum*, *soplex*, *gcc*, *bwaves*, *leslie3d*, *astar*, *bzip2*, *zeusmp*, *sphinx3* and *omnetpp*. The rest of the benchmarks access with low frequency (denoted L). From these two groups, we generate 3 workload types denoted L, H and X depending on whether all benchmarks belong to group L, H or a combination of both.

We generate 8 workloads per group and processor setup, except for the 1-core setup where all benchmarks run in isolation. 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 L and the other half to H. We do not put any constraint on whether benchmarks can repeat in a particular workload since the random selection of benchmarks is always performed out of the corresponding (original) group of benchmarks.

5.1.2. ***Metrics.*** In order to evaluate the accuracy of DReAM, we use as reference the ideal model. In each experiment, we measure the *off estimation* or *prediction error* of each model with respect to the ideal model, which is computed as follows, where N is the number of tasks in a workload.

$$WldPredError = \frac{\sum_{i=1}^{N} |Energy_{ideal_i} - Energy_{model_i}|}{Energy_{measured}} \quad (5)$$

We then take the average *WldPredError* across all benchmarks in each workload analyzed in each processor setup.

5.2. DReAM Energy Estimation

In this section we show the accuracy of DReAM with respect to the ideal model presented in Section 3. We also include the ES model that uniformly splits energy across all running tasks regardless of their activity and memory behavior, together with a simple PTA model that splits energy across tasks proportionally to their memory accesses.

5.2.1. DReAM ***Sampling Interval (IntMem).*** The memory energy consumption prediction of DReAM varies with different sample period (interval) lengths. When choosing the interval length, we seek for a reasonable tradeoff between accuracy and hardware cost. Figure 2 shows the average *WldPredError* for each task in a particular workload. This workload belongs to group X and runs in a 4-core configuration. We explore sampling periods from 128 to 500K processor cycles. Trends for most workloads are similar, so we have used this particular one to illustrate the sensitivity of DReAM to the particular sampling period.

As expected, higher sampling frequency increases accuracy. However, discrepancy between short and long sampling periods is not huge (from 4.6% to 7.4% average *Wld-*

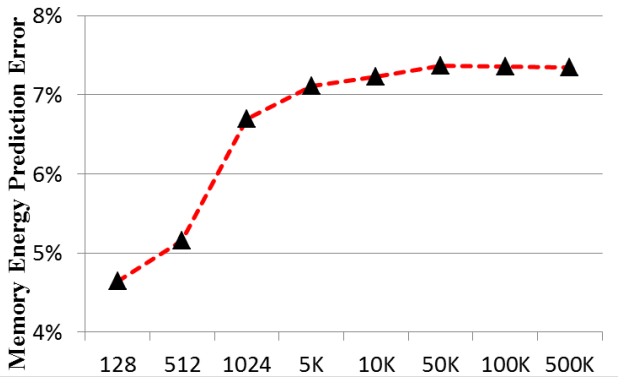

Fig. 2. Per-task DRAM memory energy prediction of a 4-core workload *soplex+sjeng+gcc+namd* with different sampling intervals.

PredError). Some meaningful average *WldPredError* increase is observed when moving from a 512-cycles sampling interval to a 1024-cycles interval. Further increasing the interval size until reaching half million cycles has little impact on accuracy since deviation from the ideal model quickly flattens[1]. Thus, we have chosen two different interval sizes with different accuracy/cost tradeoff: 512 and 500K cycles sampling intervals.

5.2.2. ***DRAM Energy Consumption Prediction.*** Next we evaluate the off estimation for 4-core and 16-core processor setups with respect to the ideal model. Note that the ideal model is the only reference model as no existing hardware provides accurate per-task DRAM energy metering.

Figure 3 shows the result for the 24 workloads (8 of each type) for the 4-core setup. We observe that, in general, the ES model is highly inaccurate averaging over 45% prediction error across all workloads. Prediction is more accurate for L and H workloads than for X ones. This is expected since benchmarks in L and H workloads are more homogeneous, so their individual power consumption is also more homogeneous than in X workloads. In some particular workloads, the prediction error is even below 10%. Nevertheless, ES model prediction error is very high in general, ranging from 30% to 85% for most workloads. For X workloads, the prediction error is always above 58%. PTA model improves the estimation accuracy, with an average prediction error around 23%. PTA accuracy is high for H workloads (the errors are all under 10%) since the large number of accesses of H benchmarks makes energy more proportional to the number of accesses (dynamic energy becomes dominant). However, benchmarks in L group infrequently access memory, so their memory energy is mainly background energy, which PTA fails to predict accurately. This fact is particularly noticeable for workload $w4$ where, although all tasks have few memory accesses and so, their energy is dominated by PD background energy, the fact that one task has a number of accesses relatively much higher than the others makes it account for most of the energy, thus producing very high error prediction. Conversely, in this workload the ES model is far more accurate than PTA since energy is quite homogeneous across tasks in the workload. Our `DReAM` model improves prediction accuracy significantly over both ES and PTA. When the sample period granularity is 512 cycles, the prediction error is al-

[1]Longer sampling period is also applicable, however, `DReAM` aims to provide the estimation at a finer granularity than the operating system quantum to offer a flexible use.

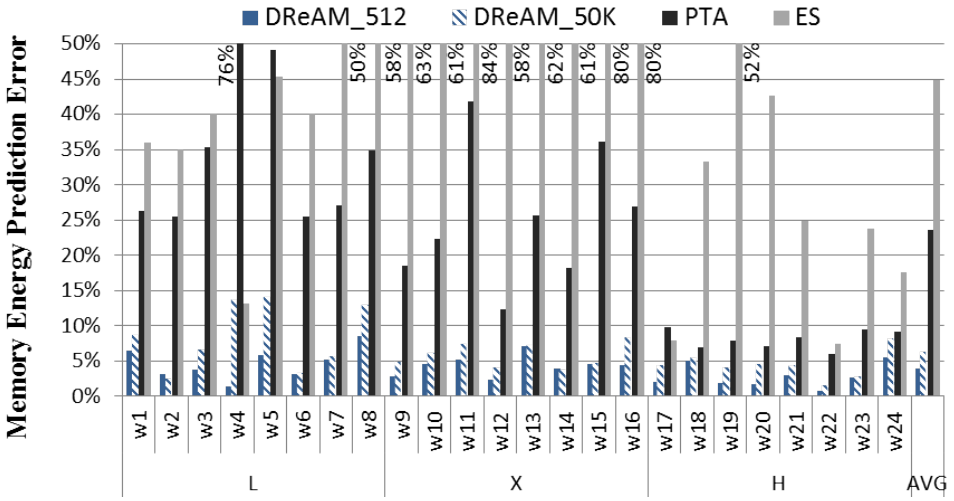

Fig. 3. Per-task DRAM energy prediction error for 4-core workloads.

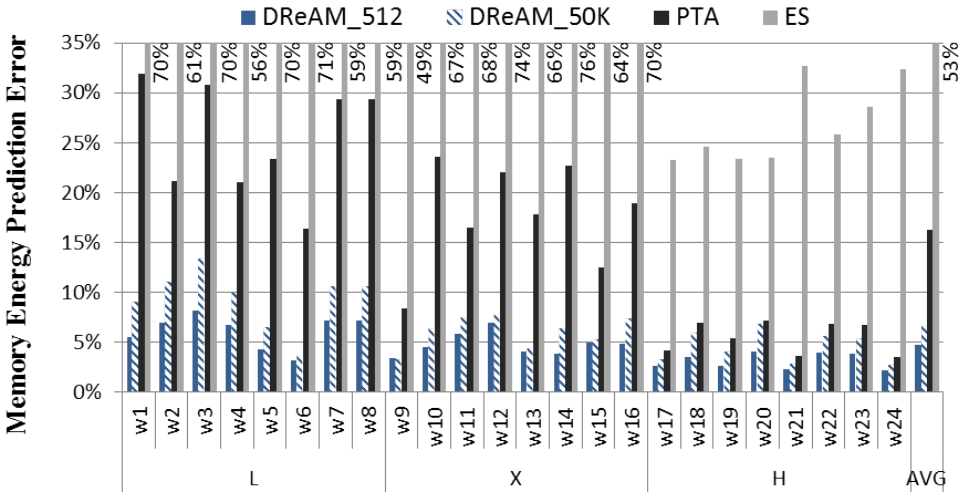

Fig. 4. Per-task DRAM energy prediction error for 16-core workloads.

ways below 10%, and 3.9% on average. If the sampling period increases to 500K cycles, the prediction error may reach 14.0% at most for one particular workload, and 6.1% on average. As shown, DReAM successfully predicts the energy consumed by each task consistently across workloads. In particular, this holds (i) when PTA works well and ES not (e.g., workload $w12$), (ii) when ES works well and PTA not (e.g., workload $w4$), and (iii) when both PTA and ES work badly (e.g., workload $w5$).

Figure 4 shows results for the 16-core setup. First, we observe that ES and DReAM accuracy remains similar to that of the 4-core setup. In contrast, PTA accuracy slightly improves. The average prediction error across all workloads for the ES model rises to 53%. The increase is particularly noticeable for L workloads. Since total power for L

workloads is relatively low, low deviations (in absolute numbers) become high in relative numbers. A similar effect occurs for DReAM, thus making L workloads to exhibit the lowest prediction accuracy, followed by X workloads, where half of the benchmarks are L benchmarks. Conversely, H workloads consume higher power and relative deviations become less significant for all models. Trends for PTA are similar to those for the 4-core setup, thus exhibiting higher accuracy for H workloads, although accuracy for the 16-core setup is higher. This is due to the fact that, with 4 cores, a large deviation for one benchmark has significant impact on average results, but such average impact becomes lower across 16 tasks. However, maximum error for individual benchmarks in each workload still remains high. Nevertheless, PTA has an average prediction error around 17%, and around 32% for a particular workload. Opposedly, DReAM error is below 5% on average (512-cycles interval) and always below 8% across all workloads. Note that the gap between 512 and 500K cycles sampling intervals for DReAM is still around 2%, as in the 4-core case. Our results prove that DReAM is far more accurate than ES and PTA models across all workload types, and average prediction error remains nearly the same for 4 and 16 cores, thus proving that DReAM scales well.

In conclusion, DReAM model greatly improves per-task DRAM energy estimation over ES and PTA at low cost.

5.2.3. DReAM ***Area and Energy Overhead****.* DReAM requires some hardware support in the form of counters to track memory activity. Those counters are in the memory controller, which in general is on-chip, so the DRAM devices remain unchanged.

As shown in Table V, DReAM needs few counters (5 shared counters and 4 extra counters per thread). 32-bit counters suffice to track the corresponding events. Further, few of those counters are accessed on a memory access and at the end of a sampling interval. Although computing the energy consumed by each thread in a particular interval involves few arithmetic operations, low-area and low-power arithmetic units (e.g., iterative multipliers [Santoro and Horowitz 1989] and dividers [Juang et al. 2008] operating at low frequency) can be set up for that purpose. We have considered the energy consumption for two different sampling intervals: 512 and 500K cycles. Area and power overheads have been estimated with power models analogous to those of Wattch [Brooks et al. 2000] built on top of CACTI 6.5 simulation tool [Muralimanohar et al. 2009]. CACTI is a flexible tool modeling delay, energy (dynamic and leakage) and area of cache memories and SRAM-based arrays. Results for 4-core and 16-core configurations show that the total energy and area overhead for DReAM is largely below 0.1% of the entire chip. If we compare DReAM energy overhead with DRAM energy consumption, it is also largely below 0.1% of total DRAM energy consumption. Furthermore, relative overheads do not change noticeably if the core count is increased, which proves that DReAM scales well. Energy overheads for 512 cycles sampling intervals are higher than for 500K intervals, but still under 0.1% for the whole chip. Due to its higher accuracy and still low overheads, the sampling interval considered in the rest of the paper is 512 cycles.

5.3. Metering Per-task Memory Energy for Multithreaded Applications Using DReAM

The support required by DReAM in the case of multithreaded applications is simple. In fact, no hardware changes in the DReAM logic are actually required, but only on how the OS handles the MEMR: The OS or the parallel runtime, simply needs to aggregate the energy consumption estimates stored for all the threads belonging to the same multi-threaded application, so $Emeter_{App} = \sum_{i=1}^{N} MEMR_i$ where N is the number of threads of the 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 through a

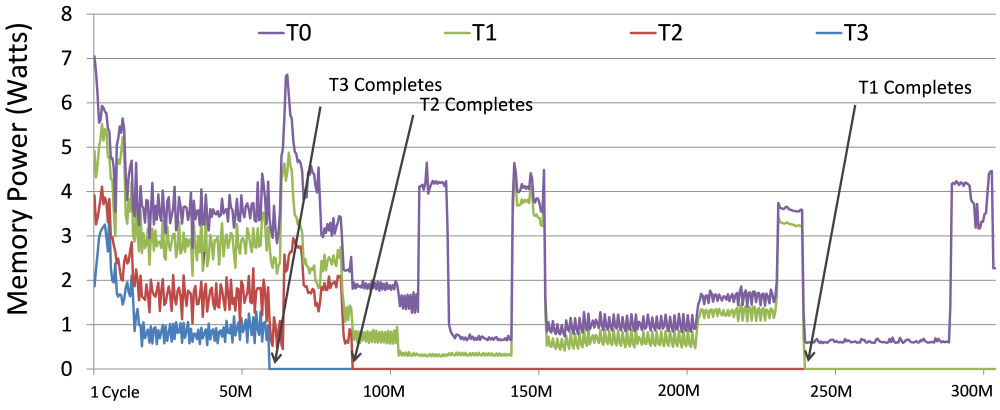

Fig. 5. `wrf` power consumption evolution between two barriers in a 4-core setup.

particular example. The information provided helps understanding the effects in terms of energy of unbalanced thread execution times.

In this section, we evaluate DReAM 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 non-hydrostatic 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 non-linear filtering and spectral analysis techniques to determine the internal structure of the trace and detect periodicity of applications. Afterwards, we use a clustering algorithm to determine the most representative computation bursts inside an iteration of the application.

We obtain 4 representatives for the 5 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 5.1. We simulate all threads sharing the LLC cache (4 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 5 shows the evolution of the per-task memory energy breakdown (using DReAM) in the system between two barrier communications. Note that energy components are stacked in the plot. At the beginning, all 4 threads dissipate dynamic, background and refresh power in memory. The memory power varies, and DReAM provides a way to monitor their power separately. Eventually, T3 reaches the barrier and becomes inactive. Thus, we attribute the powerdown background and refresh energy to the remaining 3 active tasks and stop accounting energy to T3 at this point. The behavior when T1 and T2 reach the barrier is analogous to that of T3, but it takes longer for them to reach the barrier. Upon their completion, we stop attributing the powerdown and refresh energy to them. T0, after all other tasks complete, is responsible for all the memory power and all memory energy is attributed to it.

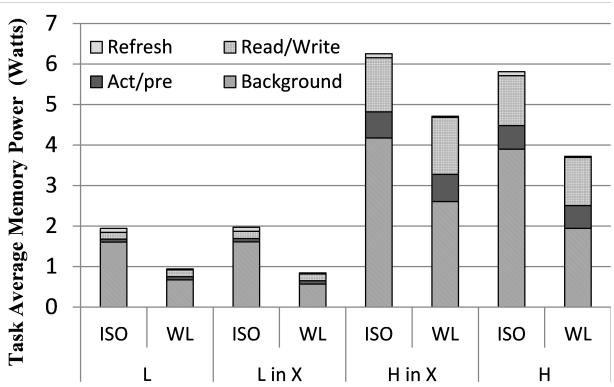

Fig. 6. All workloads power consumption comparison in a 4-core setup.

6. CASE STUDY: WORKLOAD CHARACTERIZATION WITH DREAM

In this section, we analyze how programs with different memory access profiles interact in terms of memory power consumption. For that purpose, we use DReAM, our proposed method for accurate per-task memory energy metering.

6.1. Workload as a Whole

We first analyze the different workloads paying attention to the power consumption of the different types of benchmark rather than individual benchmarks.

Figure 6 shows the average[2] memory power consumption of benchmarks in L, H and X workloads under a 4-core setup, and the average memory power they would consume if they ran in isolation. The figure has 4 sets of columns. From left to right: L workloads, L benchmarks in X workloads, H benchmarks in X workloads, and H workloads. For each set of columns, there are two columns labeled as *ISO* and *WL*. WL column shows the average data per benchmark in the corresponding category. For instance, the WL column in the L category shows the average memory power consumption per benchmark for the 32 benchmarks in those workloads (8 workloads with 4 benchmarks each). The ISO column corresponds to the average power of those 32 benchmarks when run in isolation. Note that separating results across benchmarks in workloads would not be possible without DReAM.

The first observation is that simultaneously running benchmarks in a multicore system decreases their individual memory power consumption. This fact is particularly noticeable for L benchmarks, whose average memory power decreases to less than half. Power consumption of H benchmarks decreases as well, but less than for L benchmarks. We also observe that those trends for L and H benchmarks hold independently of whether they run with benchmarks with similar or different characteristics in terms of memory access frequency.

The second observation is that, as expected, dynamic power (activate, precharge, read and write) remains roughly constant regardless of whether benchmarks run in isolation or simultaneously with other programs. However, background and refresh power decrease remarkably since they are shared across benchmarks in the workload. In particular, L programs observe a significant reduction in terms of background power when running with other programs since they keep memory in PD state most

[2]In fact, we use the harmonic mean for power in Figure 6 and 7 to take into account that slower (and lower power) programs run longer. Otherwise, we could not compare power and memory energy per instruction values fairly.

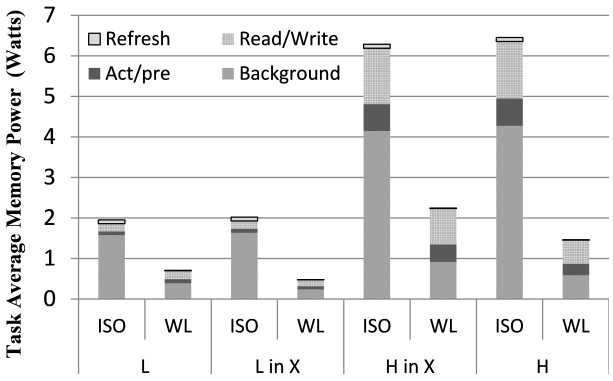

Fig. 7. All workloads power consumption comparison in a 16-core setup.

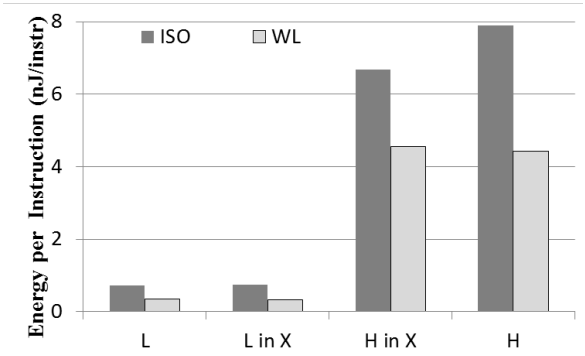

Fig. 8. Average benchmark MEPI comparison in a 4-core setup.

of the time, and PD power is shared homogeneously across running tasks. Conversely, H programs experience a lower relative reduction in terms of background power because background power during A and S states is their main source and typically few programs trigger those high-power states simultaneously. Therefore, A and S states account for most of the background energy and such energy is accounted quite often to a single task (the one inducing the high-power state). This occurs because accesses from different programs do not overlap often in time, and when they do, it is often the case that they need the same bank and thus, occur serially. Therefore, background energy due to A and S states is very similar in the workloads and in isolation.

Results for the 16-core setup, shown in Figure 7, resemble those for the 4-core setup with two main differences: (1) average memory power per program further decreases for the 16-core setup since power sources are shared across a larger number of programs; And (2) dynamic power (activate, precharge, read, write) decreases for H benchmarks because energy for those operations remains constant, but since memory contention increases execution time, power decreases.

This second effect can be better observed in Figure 9, where Memory Energy Per Instruction (MEPI) across workloads is shown. MEPI of each benchmark for 16-task workloads is lower than for executions in isolation, but the ratio is not as favorable as in terms of power for H benchmarks. This is due to the longer execution time produced by bank conflicts, memory access contention and limitations on the number of si-

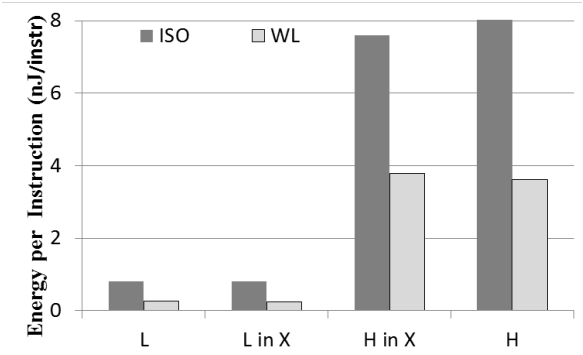

Fig. 9. Average benchmark MEPI comparison in a 16-core setup.

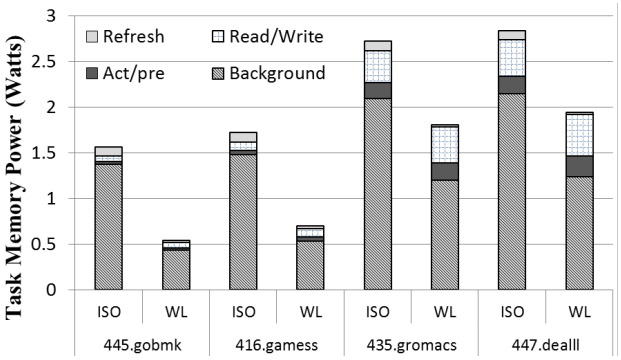

Fig. 10. L type workload power consumption comparison in a 4-core setup.

multaneously opened banks [JEDEC Solid State Technology Association 2012], which increases overall background and refresh energy, thus increasing MEPI.

Figures 8 and 9 show MEPI for 4-core and 16-core setups respectively. We observe that MEPI ratios between WL and ISO remain the same as for power for all workload types in the 4-core setup and L workloads in the 16-core setup. This occurs because the impact in execution time due to memory contention is negligible. However, H workloads and H benchmarks in X workloads in the 16-core setup experience some MEPI increment due to contention with concurrent memory requests as explained before. Note that power and energy for H (L) workloads and H (L) benchmarks in X workloads differ simply because benchmarks have been picked randomly and therefore, those sets contain different benchmarks (still of the same type). The same happens when comparing the MEPI in isolation in different processor setups.

6.2. Per-Benchmark Analysis

In this section we further study the behavior of benchmarks individually in different workloads. `DReAM` enables this study, which could not be done otherwise. For that purpose, we picked the workload with the most varying behavior with respect to the average case for each of the workload types (L, X and H) and core count (4 and 16), for a total of 6 workloads. In many cases, the most-varying behavior workload does not show big discrepancies with the average behavior for most of the benchmarks.

L ***Type Workloads***. Figure 10 shows the power consumption in an L type workload with 4 cores. As shown before, power is reduced to less than half on average

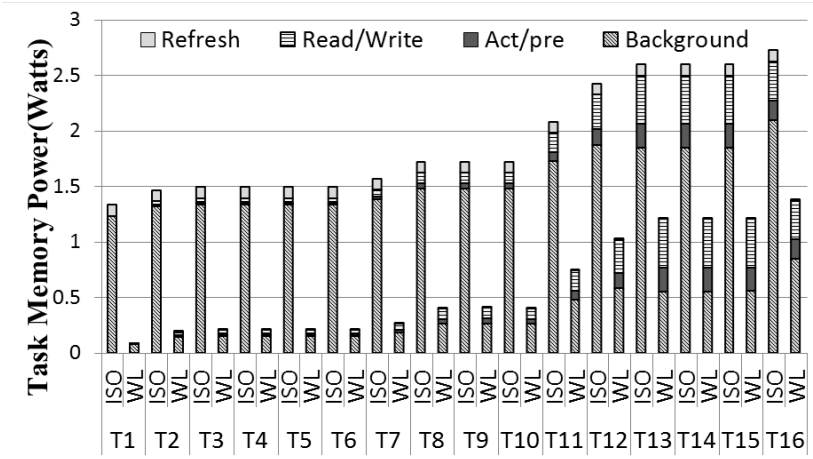

Fig. 11. L type workload power consumption comparison in a 16-core setup. Tn stands for task n in the workload.

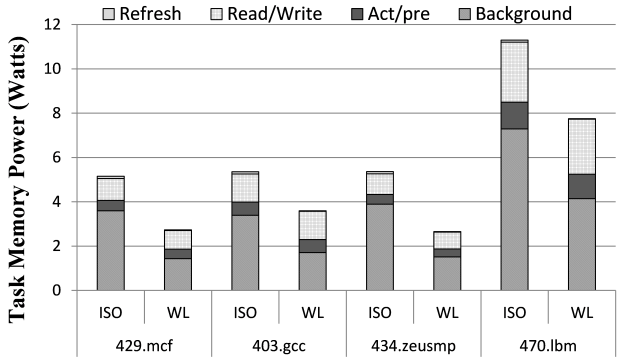

Fig. 12. H type workload power consumption comparison in a 4-core setup.

for L workloads in comparison with the ISO case. However, when we analyze benchmarks individually, we observe that those benchmarks with higher memory access frequency (*gromacs* and *dealII*) have higher WL case power consumption. This is expected since workloads are not fully homogeneous and discrepancies in the memory access frequency lead to higher background power for those programs keeping the memory in a higher power state longer. The fact that PD state background power is very low makes programs with a relatively higher memory access frequency increase their background power noticeably in relative numbers. Therefore, they are responsible for a larger fraction of the total energy consumption (and so of the power consumption). Dynamic power remains basically the same for ISO and WL since energy per access is constant and execution time barely changes.

Results for an L workload in a 16-core setup are shown in Figure 11. Trends are analogous to those reported for the 4-core setup with the only difference that power reductions are larger as already pointed out for the average results across all workloads.

H ***Type Workloads***. Figure 12 shows the power consumption in an H type workload on a 4-core setup. We can observe that, on average, power decreases moderately in the WL case with respect to the ISO case. Analogously to the trends in L workloads, the higher the memory access frequency, the lower the power reduction in the WL case

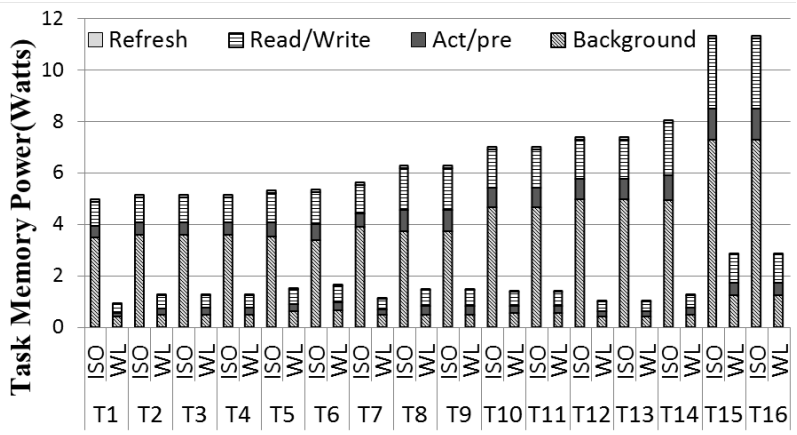

Fig. 13. H type workload power consumption comparison in a 16-core setup. Tn stands for task n in the workload.

since access frequency strongly correlates with background power. This is the case for benchmark *lbm*, whose background power consumption decreases only by around 40% instead of the average 55% for the whole workload. Note also that, although *zeusmp* and *gcc* have nearly the same DRAM power in isolation, *zeusmp* experiences a more significant reduction in power when running in the workload. A larger background power in the case of *zeusmp* explains this different behavior. For a 16-core workload, we also observe similar trends in Figure 13 to those in the average case. This is expected because H workloads are much more homogeneous than the others (L and X) since relative variations in access frequency across benchmarks is low (all of them access memory at least 5 times every 1000 cycles in isolation). Again, we observe that power in WL is much lower than in ISO, and such power decrease is much higher than for the 4-core case.

X ***Type Workloads***. Figure 14 shows a 4-core X workload. In this workload, *bzip* and *soplex* are H programs whereas *gromacs* and *gamess* are L programs. Notably, the same trends observed in pure H and L workloads still hold for each H and L benchmark in X workloads. As expected, *soplex* is the program experiencing a lower power reduction when moving from ISO to WL due to its high access frequency. In the 16-core setup (see Figure 15), those trends still hold. Only *T11* behaves differently since its power reduction in WL is not as significant as for the other benchmarks with similar access frequency. The reason is that this program accesses memory frequently (therefore its dynamic power is high), but it does it in bursts, so that the amount of time that DRAM devices are imposed to be at high power states (active or standby) is relatively low, and it makes its ISO background power low (e.g., compared to that of *T10* or *T12*). Therefore, its relative background power reduction in the WL case cannot be as significant as for other benchmarks with similar average access frequency but with different access patterns.

We do not further discuss MEPI for those particular workloads since the conclusions are similar as those for power.

6.3. Summary

We have shown that multicore architectures help reducing per-task memory power and energy. Energy savings are more significant for those programs with lower memory ac-

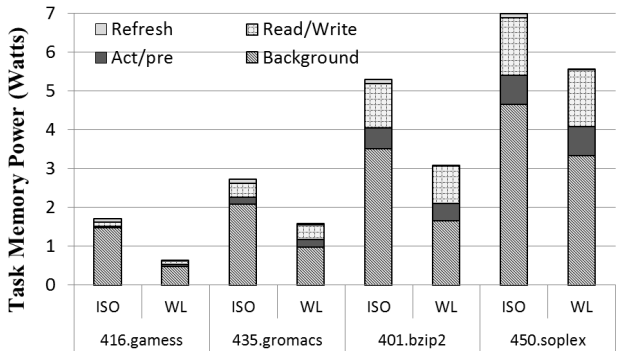

Fig. 14. X type workload power consumption comparison in a 4-core setup.

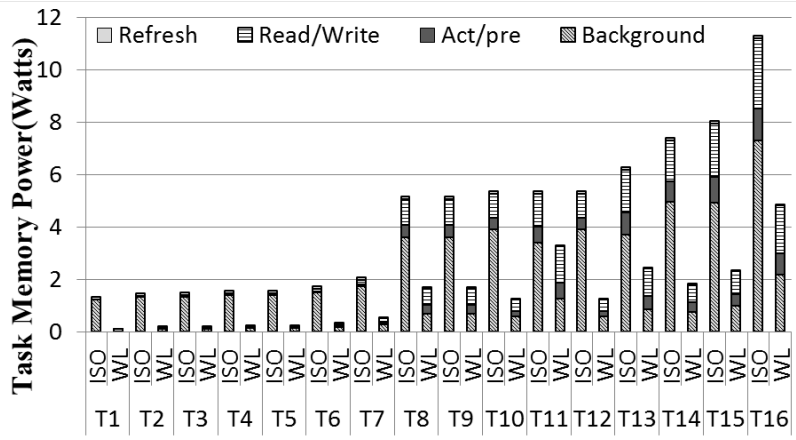

Fig. 15. X type workload power consumption comparison in a 16-core setup. Tn stands for task n in the workload.

cess frequency on higher core count setups, and trends do not change across workloads. Furthermore, exceptions do not deviate much from the average case, and when they do, it is because of their access patterns (burst versus scattered).

We have also shown that the impact of memory contention highly correlates with the access frequency of benchmarks. Our results show that high-access-frequency programs decrease their power at the expense of increasing their energy. Our study proves that memory energy profiles are quite stable for applications despite programs running simultaneously. Besides, it is preferable to run H programs with L programs to reduce the negative impact of memory contention in terms of energy consumption (once discounted LLC interferences). This information is very useful to perform task scheduling on multicore setups.

7. CONCLUSIONS

Per-task energy metering is needed in multicores for a number of performance/energy optimizations. So far such support has been only provided for on-chip resources, but not for DRAM memories. In this paper, we propose, for the first time, an ideal model to measure per-task DRAM memory energy and devise `DReAM`, an efficient and accurate implementation of such ideal model. We show how `DReAM` achieves a prediction error

between 3.9% and 4.7% with respect to the ideal model with negligible overhead for 4- and 16-core setups respectively. The error is largely below the error introduced by approaches such as distributing energy evenly or proportionally to memory accesses. Moreover, we illustrate how DReAM allows characterizing DRAM power and energy variations due to the interaction of programs with different energy profiles in multicores. Such information enables efficient on-line power and energy estimation and energy-aware task scheduling.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P, the HiPEAC Network of Excellence, the European Research Council under the European Union's 7th FP ERC Grant Agreement n. 321253, and by a joint study agreement between IBM and BSC (number W1361154). Qixiao Liu has also been funded by the Chinese Scholarship Council under grant 2010608015. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

REFERENCES

C. Acosta, F.J. Cazorla, A. Ramirez, and M. Valero. 2009. *The MPsim Simulation Tool*. Technical Report UPC-DAC-RR-CAP-2009-15. UPC.

N. Aggarwal, J.F. Cantin, M.H. Lipasti, and J.E. Smith. 2008. Power-Efficient DRAM Speculation. In *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA)*.

L. Barroso and U. Holzle. 2007. The Case for Energy-Proportional Computing. *IEEE Computer* (October 2007).

F. Bellosa. 2000. The benefits of event-driven energy accounting in power-sensitive systems. In *ACM SIGOPS European Workshop*.

A. Beloglazov, R. Buyya, Y. Lee, and A. Zomaya. 2011. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers* (2011).

R. Bertran, Y. Becerra, D. Carrera, V. Beltran, M. Gonzilez, X. Martorell, N. Navarro, J. Torres, and E. Ayguade. 2012. Energy accounting for shared virtualized environments under DVFS using PMC-based power models. *In Futu. Gene. Comput. Syst.* (Feb. 2012).

W. Lloyd Bircher and Lizy K. John. 2007. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. In *IEEE International Symposium on Performance Analysis of Systems & Software(ISPASS)*.

D. M. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

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

K. Chandrasekar, B. Akesson, and K. Goossens. 2011. Improved Power Modeling of DDR SDRAMs. In *14th Euromicro Conference on Digital System Design (DSD)*.

Y.F. Chung, C.Y. Lin, and C.T. King. 2011. ANEPROF: Energy Profiling for Android Java Virtual Machine and Applications. In *17th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE.

H. David, C. Fallin, E. Gorbatov, U.R. Hanebutte, and O. Mutlu. 2011. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*.

H. David, E. Gorbatov, U.R. Hanebutte, R. Khanna, and C. Le. 2010. RAPL: memory power estimation and capping. In *ACM/IEEE international symposium on Low power electronics and design (ISLPED)*.

Q. Deng, D. Meisner, L. Ramos, T.F. Wenisch, and R. Bianchini. 2011. MemScale: active low-power modes for main memory. In *International conference on Architectural support for programming languages and operating systems (ASPLOS)*.

J. Gonzalez, J. Gimenez, M. Casas, M. Moretó, A. Ramírez, J. Labarta, and M. Valero. 2011. Simulating Whole Supercomputer Applications. *IEEE Micro* 31, 3 (2011), 32–45.

J. Hamilton. 2009. Internet-Scale Service Infrastructure Efficiency. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

J. L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* (Sept. 2006).

Intel Corp. 2012a. Intel 64 and IA-32 Architectures Software Developer's Manual. (2012). http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Intel Corp. 2012b. Intel Xeon Processor E5-2600 Product Family Uncore Performance Monitoring Guide. (2012).

A. Jaleel. 2007. *Memory Characterization of Workloads Using Instrumentation-Driven Simulation - A Pin-based Memory Characterization of the SPEC CPU2000 and SPEC CPU2006 Benchmark Suites*. Technical Report. http://www.glue.umd.edu/~ajaleel/workload/

JEDEC Solid State Technology Association. 2012. JEDEC DDR3 SDRAM standard. (2012).

V. Jimenez, R. Gioiosa, F.J. Cazorla, M. Valero, E. Kursun, C. Isci, A. Buyuktosunoglu, and P. Bose. 2011. Energy-Aware Accounting and Billing in Large-Scale Computing Facilities. *IEEE Micro* (2011).

T.B. Juang, S.H. Chen, and S.M. Li. 2008. A novel VLSI iterative divider architecture for fast quotient generation. In *IEEE International Symposium on Circuit and Systems(ISCAS)*.

A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. 2010. Virtual Machine Power Metering and Provisioning. In *ACM symposium on Cloud Computing (ISCC)*.

G. Kestor, R. Gioiosa, D.J. Kerbyson, and A. Hoisie. 2013. Quantifying the energy cost of data movement in scientific applications. In *IEEE International Symposium on Workload Characterization (IISWC)*.

Q. Liu, V. Jimenez, M. Moreto, J. Abella, F.J. Cazorla, and M. Valero. 2013a. Hardware Support for Accurate Per-task Energy Metering in Multicore Systems. *ACM Tran. on Archi. and Code Opti.* (2013).

Q. Liu, V. Jimenez, M. Moreto, J. Abella, F.J. Cazorla, and M. Valero. 2013b. Per-task Energy Accounting in Computing Systems. *IEEE Comput. Archi. Lett.* (2013).

Q. Liu, M. Moreto, J. Abella, F.J. Cazorla, and M. Valero. 2014. DReAM: Per-Task DRAM Energy Metering in Multicore Systems. In *Euro-Par 2014 Parallel Processing*.

J.C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A.C. Snoeren, and R.K. Gupta. 2011. Evaluating the Effectiveness of Model-based Power Characterization. In *2011 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'11)*.

J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. 2004. The Weather Reseach and Forecast Model: Software Architecture and Performance. In *ECMWF*.

Micron. 2007. *Calculating Memory System Power For DDR3*. Technical Report tn41-01ddr3-power.

N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi. 2009. *CACTI 6.0: A Tool to Understand Large Caches*. Technical Report HPL-2009-85. HP.

K.J. Nesbit, M. Moreto, F.J. Cazorla, A. Ramirez, M. Valero, and J.E. Smith. 2008. Multicore resource management. *IEEE micro* (2008).

Nokia Corp. 2007. Energy Profiler. (2007). http://nokia-energy-profiler.en.softonic.com/symbian

A. Pathak, C.Y. Hu, M. Zhang, P. Bahl, and W.-M. Wang. 2011. Fine-grained power modeling for smartphones using system call tracing.. In *EuroSys*.

A. Phansalkar, A. Joshi, and L.K. John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Comput. Arch. Lett.* (2011).

M.R. Santoro and M.A. Horowitz. 1989. SPIM: a pipelined 64*64-bit iterative multiplier. *IEEE Journal of Solid-State Circuits* (1989).

K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. 2013. Power containers: an OS facility for fine-grained power and energy management on multicore servers. In *International conference on Architectural support for programming languages and operating systems (ASPLOS)*.

T. Sherwood, E. Perelman, and B. Calder. 2001. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

D.M. Tullsen, S.J. Eggers, and H.M. Levy. 1995. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

T. Vogelsang. 2010. Understanding the Energy Consumption of Dynamic Random Access Memories. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO),*.

N.H.E. Weste and K. Eshraghian. 1988. *Principles of CMOS VLSI Design. A Systems Perspective*. Addison-Wesley.