DReAM: Per-Task DRAM Energy Metering in Multicore Systems

Qixiao Liu[1,2], Miquel Moreto[1,2], Jaume Abella[1],
Francisco J. Cazorla[1,2,3], and Mateo Valero[1,2]

[1] Barcelona Supercomputing Center, Barcelona, Spain
[2] Universitat Politecnica de Catalunya, Barcelona, Spain
[3] Spanish National Research Council (IIIA-CSIC), Barcelona, Spain

Abstract. Interaction across applications in DRAM memory impacts its energy consumption. This paper makes the case for accurate per-task DRAM energy metering in multicores, which opens new paths to energy/performance optimizations, such as per-task energy-aware task scheduling and energy-aware billing in datacenters. 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 more accurate than these other methods.

1 Introduction

Energy demand and cost of computing systems have grown during the last years, and the trend is expected to hold in the coming future [1]. Conversely, computing hardware-related costs (e.g., servers) have remained roughly constant or even decreased in datacenters, desktops and laptops. This leads to scenarios where energy costs are as significant as hardware-related costs. For instance, energy already accounts for 20% of the total cost of ownership in a large-scale computing facility [2]. This cost virtually doubles if we also include the cost of the cooling infrastructure needed to dissipate the temperature induced by such a high energy consumption. Similarly, laptops and desktops may use in the order of 50-200 Watts depending on the computing power and peripherals attached. Assuming a cost of 0.11€/kWh and 3 years of non-stop operation (so 26,280 hours), a computer dissipating 120 Watts sustainedly would reach an energy cost of 350€. This cost is in the same order of magnitude as the computer itself and it is expected to grow since energy cost is expected to grow [1]. Therefore, managing energy consumption is of paramount importance.

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. Metering the energy consumed by each task accurately would provide the following benefits. 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

F. Silva et al. (Eds.): Euro-Par 2014, LNCS 8632, pp. 111–123, 2014.

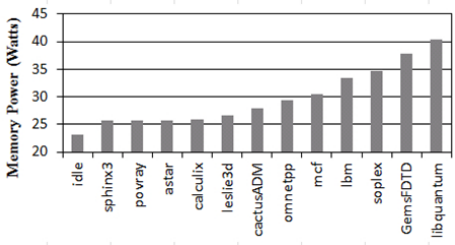

Fig. 1. Average memory power of a set of SPEC CPU 2006 benchmarks running alone on an Intel Sandy Bridge server, with 8 cores and a 64GB DDR3 memory running at 1.6GHz. Memory power is obtained using the Running Average Power Limit (RAPL) interfaces [6] and total system power with a *FitPC* external multimeter. We correlate total power data with the data collected from the hardware energy counters using time stamps. Representative benchmarks were selected based on previous characterization studies [7].

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 schedule tasks better so that energy consumption is minimized while still completing tasks when needed. And third, traditionally, datacenters charge users based on the resources they are allocated. The increasing fraction of energy-related costs in datacenters and the need for 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 running applications [3].

In that respect, despite memory power keeps increasing, reaching 30-50W in high-performance computers [4], there is a lack of understanding of per-task energy consumption in memory. To elaborate on the need of accurate per-task memory energy metering, we measured the power dissipation of different SPEC CPU 2006 benchmarks on an Intel Sandy Bridge server, see Figure 1. In this experiment memory represents between 25% and 34% of the total system power and it is comparable to the entire processor socket power. Further, different tasks incur different power consumption: e.g., 25.7W (`482.sphinx3`) versus 40.4W (`462.libquantum`). However, while per-task energy metering solutions exist for processors [5], to the best of our knowledge, no mechanism exists to accurately measure the per-task memory energy consumption in multicore systems.

We propose, for the first time, an ideal method and an efficient implementation of such method to fairly measure the energy consumed in DRAM memories when concurrently running several tasks. Our approach relies on tracking both the activity incurred by running tasks and the memory state they induce.

Overall, the contributions of this work are as follows:

- An ideal per-task energy metering model for DRAM memories, as needed for performance/energy optimization, task scheduling and billing in multicore systems. This is the reference model against which per-task energy metering mechanisms in DRAM memories can be compared to.
- `DReAM`, an accurate, yet low cost, implementation of the ideal model. `DReAM` is within 5% average error with respect to the ideal model at the expense of less than 0.1% power and area overhead in the processor.
- A comparison of `DReAM` with other energy metering approaches proving that `DReAM` is far more accurate than those other approaches.

2 Background and Related Work

In recent years, there has been an increasing interest for energy metering in different environments from datacenters [3] to smartphones [8,9]. Previous proposals, however, focus 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 is accounted the whole energy consumption of the system (e.g., using a simple meter). Other proposals [4,10] make use of performance-monitoring counters (PMCs) or system events, such as OS system calls, to breakdown the energy consumption of the system across its components (e.g., memory, processor, etc.). 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 [11].

Recently, Shen et al. [12] proposed a request-level OS mechanism to attribute power consumption to each server request based on PMCs [13]. Similarly, Kestor et al. [14] derive 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 interferences unless appropriate solutions provide accurate per-task energy metering in multicores. Our work in [5] provides Per-Task Energy Metering (PTEM) for on-chip resources (cores, caches, etc.). Our proposal in this paper, `DReAM`, provides such support for DRAM memories.

DRAM memory energy consumption can be split into dynamic, refresh and background. Dynamic energy corresponds to the energy spent to perform those *useful* activities triggered by the programs running. 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. Background energy includes the energy consumed due to *useless* activity not triggered by the program(s) being run as well as the energy wasted due to imperfections of the technology used to implement the circuit.

3 Metering Per-Task Energy Consumption

In this section we present an idealized model for per-task 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

Table 1. Memory commands, timing, power states and background power breakdown for a read operation in close-page mode

<table>
<tr><td rowspan="2">Command</td><td>T_0</td><td>–</td><td colspan="2">ACT</td><td>READ</td><td>PRE</td><td>–</td></tr>
<tr><td>T_1</td><td colspan="6">–</td></tr>
<tr><td>Timing</td><td>T_0</td><td>–</td><td>t_{XP}</td><td>t_{RCD}</td><td>t_{RTP}</td><td>t_{RP}</td><td>–</td></tr>
<tr><td rowspan="4">State</td><td>$Bank_0$</td><td rowspan="4">PD</td><td rowspan="4">S</td><td colspan="2">A</td><td rowspan="4">S</td><td rowspan="4">PD</td></tr>
<tr><td>$Bank_1$</td><td colspan="2" rowspan="3">S</td></tr>
<tr><td>$Bank_2$</td></tr>
<tr><td>$Bank_3$</td></tr>
<tr><td rowspan="3">Power</td><td>Rank</td><td>P_{PD}</td><td>P_S</td><td colspan="2">P_A</td><td>P_S</td><td>P_{PD}</td></tr>
<tr><td>T_0</td><td>$\frac{P_{PD}}{2}$</td><td>$P_S - \frac{P_{PD}}{2}$</td><td colspan="2">$P_A - \frac{P_{PD}}{2}$</td><td>$P_S - \frac{P_{PD}}{2}$</td><td>$\frac{P_{PD}}{2}$</td></tr>
<tr><td>T_1</td><td colspan="6">$\frac{P_{PD}}{2}$</td></tr>
</table>

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.

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 [15]. In this model, all devices in a rank are accessed upon every access. In each device, only one bank is accessed, in which all arrays 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 configuration. 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.

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, banks can be in different states. Note, however, that our approach can be easily adapted to other models. This is not detailed in this paper due to lack of space.

3.2 Memory Energy Consumption

The energy model for the main memory is based on the current profiles provided by Micron [16] and it splits energy consumption into dynamic, refresh and

background energy. This is analogous to the methodology used in [17], 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. 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.

Table 1 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 1. Eventually, some banks are activated so that the device as well as some banks transition to A state. The device and the accessed banks (Bank0 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). 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, the memory controller sends the clock disable signal returning the device to PD state.

As stated before, modern memory controllers may implement either open-page or close-page policies. The close-page policy is the focus in this paper, although we have observed similar trends for open-page policy.

3.3 Per-Task Energy Metering for Close-Page Policy

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. Memory occupancy is discarded as input for the model since background energy does not depend on it.

1) During PD only background power is consumed, which cannot be attributed to any task since during PD no task has any memory activity. Hence, we divide background power evenly across all tasks running in the processor. 2) Whenever a device transitions 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 inflight commands that force the memory devices to stay in S state. 3) When a 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 1 shows the case where one task, T_0, issues a *read* command (first row) and the other task T_1 issues no command. Assuming that those are the only tasks using the

Table 2. Memory commands, timing, power states and background power breakdown for several operations in close-page mode.

Command	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	$\frac{P_{PD}}{2}$			$\frac{P_S}{2}$	$\frac{P_A}{2}$	$P_A - \frac{P_S}{2}$	$\frac{P_S}{2}$	$P_S - \frac{P_{PD}}{2}$	$\frac{P_{PD}}{2}$				

memory system, during the whole period T_1 is responsible only for half of the P_{PD} power (last row). 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 to be on a given state. In the example in Table 2, 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, devices transition 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.

3.4 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:

$$E^{mem}_{bg,\ total}(Tk_i) = P_{PD} \times ExecTime(Tk_i)/\#Tk + \sum_{j=0}^{ExecTime(Tk_i)} \left((P_S - P_{PD}) \times \frac{\delta^S_{i,j}}{\#Tk_{S,j}} \right) + \sum_{j=0}^{ExecTime(Tk_i)} \left((P_A - P_S) \times \frac{\delta^A_{i,j}}{\#Tk_{A,j}} \right) \quad (1)$$

In the first addend each running task is metered an even part of P_{PD}, where $ExecTime(Tk_i)$ stands for the execution time of task i in cycles and $\#Tk$ 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. $\#Tk_{S,j}$ and $\#Tk_{A,j}$ 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, $\delta^A_{i,j}$ 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 operations it performs, as shown in the following equations:

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

where E^{mem}_{read}, E^{mem}_{write}, E^{mem}_{ACT} and E^{mem}_{PRE} stand for the energy of each command, and $\#RD(Tk_i)$, $\#WR(Tk_i)$, $\#ACT(Tk_i)$ and $\#PRE(Tk_i)$ 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 split evenly across all running tasks. Thus, refresh energy per task is as follows:

$$E^{mem}_{refr,\ total}(Tk_i) = E^{mem}_{refr} \times \#Ref \times ExecTime(Tk_i)/\#Tk \tag{3}$$

E^{mem}_{refr} corresponds to the dynamic energy of a refresh command. $\#Ref$ 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 end, 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 ($\#Ref$). These parameters are already provided by chip vendors like Micron for DDR2/3 memories [16], so our model imposes no change to current DDR2/3 memories. In the memory controller,

Table 3. 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}, $\#Ref$	$\#RD$, $\#WR$, $\#ACT$, $\#PRE$, $\#RD(Tk_i)$, $\#WR(Tk_i)$, $\#ACT(Tk_i)$, $\#PRE(Tk_i)$, $IntMem$ cycle counter

we only require per-task activity counters, namely $\#RD(Tk_i)$, $\#WR(Tk_i)$, $\#ACT(Tk_i)$ and $\#PRE(Tk_i)$. Total background energy, $E^{mem}_{bg,total}$ can be obtained by metering memory energy consumption [10] and subtracting dynamic and refresh energy. The PD background power is constant and hence easy to track. Meanwhile, 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:

$$E^{mem}_{bg,\ total}(Tk_i) = \frac{P^{mem}_{PD} \times ExecTime(Tk_i)}{\#Tk} + \sum_{j=0}^{\frac{ExecTime(Tk_i)}{IntMem}} \#accesses^{Tk_i}_j \times \frac{E^{mem}_{rem}(j)}{\#TOTaccesses_j} \tag{4}$$

where P^{mem}_{PD} is the PD background power, $\#accesses^{Tk_i}_j$ tracks the number of memory accesses of task i during interval j, and $\#TOTaccesses_j$ 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 All Together

The DReAM approach requires little hardware overhead. DReAM mostly requires setting up some 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 3 summarizes those parameters required from the memory vendor and the extra logic (i.e. counters) that must be set up. Counters with the "(Tk_i)" suffix must be replicated for each task.

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 will happen 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

Table 4. 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, 16 cores, single-threaded
Fetch, decode, issue, commit bandwidth	2 instructions/cycle
Instruction & Data L1	32KB, 4-way, 32B/line (2 cycles hit)
Instruction & Data *TLB*	256 entries fully-associative (1 cycle hit)
LLC Size	256KB/core, 16-way, 64B/line (3 cycles hit + 12 cycles L1 miss penalty and bus round trip) 256KB (1 core), 1MB (4 cores), 4MB (16 cores)

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 [5], 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 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 *DRAMsim2* [15] to model off-chip main memory, a cycle-accurate memory system simulator for DDR2/3 memories including a memory controller and DRAM memory. The processor is modeled with *MPsim* [18]. *DRAMsim2* has been connected to *MPsim* so that *LLC* misses are propagated to the memory controller, which manages those memory requests. A power model based on Micron memories has been implemented in *DRAMsim2*.

We consider three Chip Multi-Processor (CMP) configurations with 1, 4 and 16 single-threaded cores. The second level cache (L2) is partitioned with 256KB 16-way per core. Therefore, L2 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 L2 cache), thus allowing to consider memory effects only. Details about the configuration can be found in Table 4. Other parameters are analogous to those in [5].

For the DRAM memory we model a 8GB memory since it is 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. DRAM memory row-buffer management policy is close-page across all the evaluation section.

Benchmarks. We use traces collected from the whole SPEC CPU 2006 benchmark suite using the reference input set. Each trace contains 100 million instructions, selected using the SimPoint methodology [19]. 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 randomly, except for the 1-core setup where all benchmarks run in isolation. In the case of X, half of the benchmarks belong to L and the other half to H.

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

$$WldPredError = \frac{\sum_{i=0}^{N} |Energy_{ideal_i} - Energy_{model_i}|}{Energy_{measured}} \quad (5)$$

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 Proportionally To memory Accesses model (PTA) that splits energy across tasks proportionally to their memory accesses.

`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, by regulating the interval period from 128 to half million processor cycles. 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 *WldPredError*). 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. Thus, we have chosen two different interval sizes with different accuracy/cost tradeoff: 512 and 50K cycles sampling intervals.

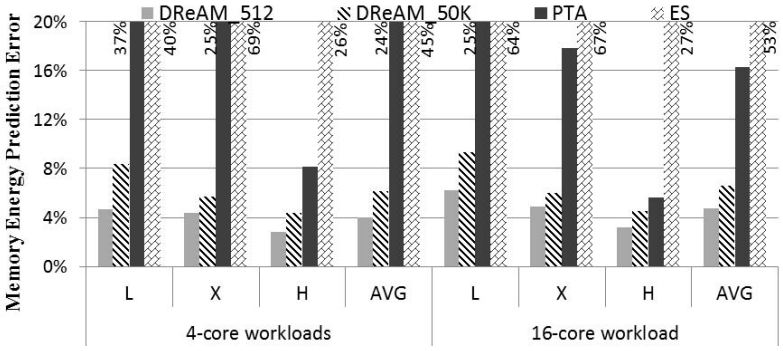

Fig. 2. Per-task DRAM energy prediction error for 4-core workloads

DRAM Energy Consumption Prediction. Next we evaluate the off estimation for 4-core and 16-core processor setups with respect to the ideal model. The left half of Figure 2 shows the result for the 24 workloads (8 of each type) under the 4-core setup. We observe that, in general, the ES model is highly inaccurate averaging over 45% prediction error across all workloads, and ranging from 26% to 69% for all workload types. 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. PTA model improves the estimation accuracy, with an average prediction error around 24%. PTA accuracy is high for H workloads since the large number of accesses of H benchmarks makes energy cost more proportional to the number of accesses (dynamic energy becomes dominant). However, benchmarks in L group seldom access memory, so their memory energy is mainly background energy, which PTA fails to predict accurately.

Our `DReAM` model improves prediction accuracy significantly over both ES and PTA. When the sample period granularity is 512 cycles, the prediction error is always below 10%, and 3.9% on average. If the sampling period increases to 50K cycles, the prediction error may reach 14.0% at most for one particular workload, and 6.1% on average. The right half of Figure 2 shows results under 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 error increment mainly comes from L workloads. A similar effect occurs for `DReAM`, thus making L workloads to exhibit the lowest prediction accuracy. 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 in 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 above 10%, and around 23% for a particular workload. Opposably, DReAM error is below 5% on average (512-cycles interval) and always below 8% across all workloads. Note that the gap between 512 and 50K 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.

Using the same evaluation methodology, we have also validated the prediction accuracy of DReAM under open-policy. However, results obtained did not offer any further insight. Since many current DRAM chips implement low-power mode, and so is *DRAMsim2*, the open banks under open-page policy transition quickly to power down state when there is no incoming request. This fact makes open-page policy perform similarly to close-page in multicore systems. Results are not shown due to space constraints.

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

As shown in Table 3, 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 each memory access and at the end of a sampling interval. We have considered the energy consumption for two different sampling intervals: 512 and 50K cycles. Area and power overheads have been derived with power models analogous to those of Wattch [20]. Wattch-like power models are built on top of CACTI 6.5 simulation tool [21]. 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 memory system.

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 50K intervals, but still under 0.1% for the whole chip.

6 Conclusions

Different programs show highly different energy profiles in different components. However, per-task memory energy metering has not been considered so far. 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% w.r.t. 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 even distribution and proportional-to-accesses distribution.

Acknowledgements. This work has been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557, the HiPEAC

Network of Excellence, by 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.

References

1. Barroso, L.: The Price of Performance. Queue 3(7) (2005)
2. Hamilton, J.: Internet-Scale Service Infrastructure Efficiency. In: ISCA (2009)
3. Jimenez, V., Gioiosa, R., Cazorla, F., Valero, M., Kursun, E., Isci, C., Buyuktosunoglu, A., Bose, P.: Energy-aware accounting and billing in large-scale computing facilities. IEEE Micro 31(3), 60–71 (2011)
4. Bircher, W.L., John, L.K.: Complete system power estimation: A trickle-down approach based on performance events. In: ISPASS (April 2007)
5. Liu, Q., Moreto, M., Jimenez, V., Abella, J., Cazorla, F.J., Valero, M.: Hardware support for accurate per-task energy metering in multicore systems. ACM Trans. Archit. Code Optim. 10(4) (December 2013)
6. Intel Corp.: Intel 64 and ia-32 architectures software developer's manual (2012)
7. Phansalkar, A., Joshi, A., John, L.K.: Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In: ISCA, pp. 412–423 (2007)
8. Pathak, A., Hu, C., Zhang, M., Bahl, P., Wang, W.M.: Fine-grained power modeling for smartphones using system call tracing. In: EuroSys. (2011)
9. Chung, Y.F., Lin, C.Y., King, C.T.: ANEPROF: Energy profiling for android java virtual machine and applications. In: ICPADS (2011)
10. David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: RAPL: Memory power estimation and capping. In: ISLPED (2010)
11. Intel Corp.: Intel xeon processor E5-2600 product family uncore performance monitoring guide (March 2012)
12. Shen, K., Shriraman, A., Dwarkadas, S., Zhang, X., Chen, Z.: Power containers: an os facility for fine-grained power and energy management on multicore servers. In: ASPLOS (2013)
13. Bellosa, F.: The benefits of event-driven energy accounting in power-sensitive systems. In: ACM SIGOPS European Workshop, pp. 37–42 (2000)
14. Kestor, G., Gioiosa, R., Kerbyson, D., Hoisie, A.: Quantifying the energy cost of data movement in scientific applications. In: IISWC, pp. 56–65 (September 2013)
15. Rosenfeld, P., Cooper-Balis, E., Jacob, B.: DRAMSim2: A cycle accurate memory system simulator. IEEE Comput. Archit. Lett. (2011)
16. Micron: Calculating memory system power for DDR3. Micron Technical Notes (2007)
17. Deng, Q., Meisner, D., Ramos, L., Wenisch, T., Bianchini, R.: Memscale: Active low-power modes for main memory. In: ASPLOS (2011)
18. Acosta, C., Cazorla, F., Ramirez, A., Valero, M.: The MPsim simulation tool. Technical Report UPC-DAC-RR-CAP-2009-15, UPC (2009)
19. Sherwood, T., Perelman, E., Calder, B.: Basic block distribution analysis to find periodic behavior and simulation points in applications. In: PACT (2001)
20. Brooks, D.M., Tiwari, V., Martonosi, M.: Wattch: A framework for architectural-level power analysis and optimizations. In: ISCA (2000)
21. Muralimanohar, N., Balasubramonian, R., Jouppi, N.: CACTI 6.0: A tool to understand large caches. HP Tech Report HPL-2009-85 (2009)