ExCaD: Exploring Last-level Cache to Improve DRAM Energy Efficiency

Su Myat Min Haris Javaid Sri Parameswaran

University of New South Wales, Australia
{sumyatmins,harisj,sridevan}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201223
July 2012

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Embedded systems with high energy consumption often exploit the idleness of DRAM to reduce their energy consumption by putting the DRAM into deepest low-power mode (self-refresh power down mode) during idle periods. DRAM idle periods heavily depend on the last-level cache, and in this paper, we propose the exploration of last-level cache configurations to improve DRAM energy efficiency by selecting the one which maximally reduces the total energy consumption of last-level cache and DRAM.

To facilitate fast exploration, we propose a novel, simple yet high fidelity DRAM energy reduction estimator. Our framework, ExCaD, combines the estimator with a cache simulator and a novel cache profile transformation technique to avoid slow cycle-accurate processor-memory simulations. Our experiments with eight different applications from mediabench, two different DRAM sizes and 330 last-level cache configurations show that the cache configuration selected by ExCaD reduces DRAM energy consumption by at least 96% and 34% over systems without last-level cache and with largest last-level cache respectively. Use of self-refresh power down mode saved at least 93% more DRAM energy consumption compared to a system where it was not used. These results indicate that a suitable last-level cache configuration with self-refresh power down mode can significantly improve DRAM energy efficiency. ExCaD took only a few hours to explore last-level cache configurations compared to several days for cycle-accurate processor-memory simulations.

1 INTRODUCTION

Energy efficiency is of utmost concern in the design of electronic systems, especially embedded systems. In these systems, main memories consume up to 80% of total system power [1], making memory systems' energy efficiency of primary interest. Figure 1.1 shows that DRAM consumes up to 80% power in a uniprocessor system with L1 and L2 caches for eight different applications from mediabench [2]. Power consumption in DRAM consists of active, background and refresh power. Background power is a major contributor consuming 90% of DRAM power and 70% of total system power. This significant consumption is due to leakage power which is substantial in deep sub-micron technologies.

To reduce non-active power consumption, DRAM manufacturers provide multiple power modes (such as active StandBy (SB), self-refresh Power Down (PD), etc.), which are reported in Table 1.1. A DRAM consists of multiple ranks where a rank is the smallest granularity at which these power modes are applied. Different power modes consume different amounts of power, where self-refresh PD is the deepest low-power mode. In self-refresh PD mode, DRAM internally refreshes its contents without the need for refresh commands from memory controller and hence consumes only minuscule amounts of background and refresh power while retaining its contents. The transition to a low-power mode comes at the cost of a wakeup latency where deeper low-power modes incur higher wakeup latencies. The key is to choose a low-power mode only if the DRAM idle period is long enough to amortize its performance penalty. Several power management policies have been explored to determine when to transition to a given low-power mode during the execution of an application [3, 4, 5, 6, 7, 8]. More recently, to achieve even better energy efficiency for DRAMs, the focus has been on prolonging DRAM idle periods to transition it to the deepest low-power mode (self-refresh PD mode) more often than shallow low-power modes (precharge SB, etc.) [5, 9]. In this paper, we propose to exploit last-level cache to prolong DRAM idle periods for self-refresh PD mode.

Motivational Example. Consider JPEG encoder application running on a uniprocessor system with L1 and L2 cache and a DRAM memory. Figure 1.2 reports DRAM activity for several thousand cycles, where a value of 1 means the DRAM was accessed and a value of 0 means that it was idle. Two L2 cache configurations are used: (1) a 4KB, 4B line size, direct-mapped cache, denoted as [4KB, 4B, 1A] and shown in lower plot, and (2) a 128KB, 8B line size and 2-way associative cache, denoted as [128KB, 8B, 2A] and shown in the upper plot. The values inside the parentheses report DRAM idle cycles and the number of idle periods. For instance, in [128KB,

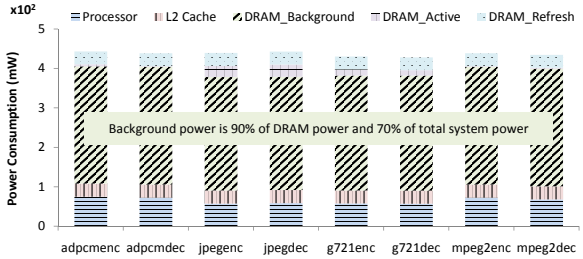

Figure 1.1: Power consumption breakdown in a uniprocessor system with on-chip L1 cache, off-chip L2 cache and DRAM memory.

Low-power Mode	Current (mA)	Wakeup Latency (clock cycles)
Active SB	57	0
Precharge SB	55	3
Active PD	35	4
Precharge PD	12	13
Self-refresh PD	6	64

Table 1.1: Power modes of Micron DDR3 DRAM. SB and PD stand for StandBy and PowerDown respectively.

8B, 2A] L2 cache system, DRAM was idle for 27.6 million cycles, distributed across 7,700 idle periods. The [128KB, 8B, 2A] L2 cache increased DRAM idle cycles by 22% which was expected due to its larger size. The number of idle periods has reduced from 270,000 to only 7,700, and this reduction will be advantageous in reducing DRAM energy consumption because: (1) idle periods will be longer enabling DRAM's transition to self-refresh PD mode (deepest low-power mode); (2) DRAM can remain in self-refresh PD mode for longer periods, and (3) fewer wakeups from self-refresh PD mode mean less performance penalty. The plot for [128KB, 8B, 2A] cache corroborates our belief, where several short idle periods of [4KB, 4B, 1A] cache that are not suitable for self-refresh PD mode, have mostly been consolidated into two longer idle periods which in fact are long enough to transition DRAM into self-refresh PD mode. Hence, [128KB, 8B, 2A] cache would reduce the energy consumption of DRAM to a larger extent than [4KB, 4B, 1A] cache. In this example, [128KB, 8B, 2A] cache reduced 38% more DRAM energy consumption than [4KB, 4B, 1A] cache. This example shows the importance of exploring last-level cache, and choosing a suitable configuration in the first place can tremendously reduce energy consumption of DRAM. Previously proposed memory controller based scheduling and power management techniques [3, 4, 5, 6, 7, 8] can still be applied, and hence our proposal is complementary to theirs.

Since entering self-refresh PD mode comes at the cost of a wakeup latency, greedily using it for every idle period may significantly degrade performance, offsetting any gains in energy reduction, and hence decreasing the overall system energy efficiency. Typically, a pre-determined threshold is used to wait for several cycles before entering into a low-power mode [3, 4, 5, 9]. Figure 1.2 shows such a threshold with a vertical

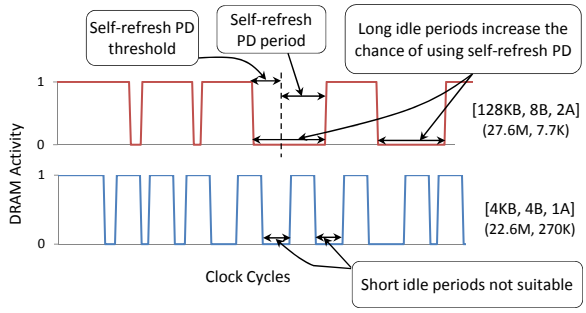

Figure 1.2: Effect of two distinct L2 (last-level) cache configurations on DRAM idle periods. PD stands for PowerDown.

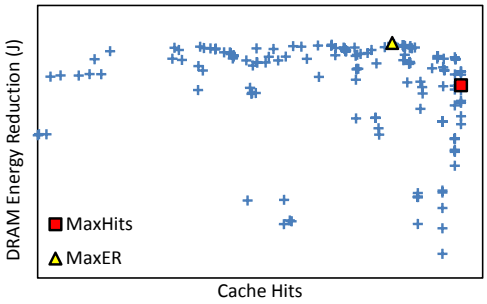

Figure 1.3: DRAM energy reduction for different L2 (last-level) cache configurations for mpeg2enc application's execution on target system of Figure 3.1. The MaxHits and MaxER stand for cache configurations with maximum number of hits and energy reduction respectively.

dotted line, and the part of the idle period before it labelled as self-refresh PD threshold. The DRAM is transitioned to self-refresh PD mode only after the threshold number of cycles to skip short idle periods such as the first two idle periods in the upper plot of Figure 1.2. The duration of self-refresh PD period in cycles (self-refresh PD cycles) is the number of cycles in DRAM idle period subtracted by self-refresh PD threshold cycles. Several studies show that a pre-determined threshold works well in most situations [3, 4, 5, 9].

We seek to answer two questions: one, for a given self-refresh PD threshold, which last-level cache configuration should one choose?; and two, can cache exploration be done quickly? One might choose the largest cache configuration because it will result in the least number of misses, thereby increasing DRAM idle cycles to maximum. Although true, largest cache configuration: (1) does not necessarily mean that all or most of the idle periods would be long enough for self-refresh PD mode because DRAM idle periods depend on application memory access pattern and hit and miss profile of the given cache configuration, and (2) will have high area and energy overhead. Hence, a more suitable approach is to explore cache configurations through cycle-accurate simulations of the system under design. However, cycle-accurate simulations of systems with multi-level caches and DRAM are exorbitantly slow [10]. For instance, in our experiments, one cycle-accurate simulation of g721enc application took 57 mins on average. Given we have to explore 300 cache configurations, which is typical of an embedded system design [10, 11], it will take several days to find the most suitable cache configuration for only one application. Hence, relying on a simulation-only solution is not feasible.

Several researchers have proposed fast cache simulators such as DEW [11], Dinero IV [12], and CRCB1 and CRCB [13]. These cache simulators use the memory trace of an application to output the number of hits and misses for all the cache configurations. Calculation of DRAM energy consumption is a complex process [14, 15], and using only the number of last-level cache hits or misses is not sufficient. Figure 1.3 plots DRAM energy reduction against the number of hits for different L2 cache configurations where it is the last-level cache in a uniprocessor system executing mpeg2enc application, in addition to cache configurations with maximum hits (MaxHits) and maximum energy reduction (MaxER). The MaxHits configuration is not the same as MaxER configuration. In addition, the area of MaxHits (3.22 mm^2) is far more than the area of MaxER (0.052 mm^2). This plot illustrates the fact that cache hits alone are not

sufficient to select the cache configuration with maximum DRAM energy reduction.

Hence, *the challenge is to quantify the effect of last-level cache configurations on DRAM energy consumption with minimal number of cycle-accurate simulations.* To this end, we make the following contributions:

- A novel DRAM energy reduction estimator. The estimator uses cache line size, cache misses, self-refresh PD cycles and number of memory requests to predict energy reduction of DRAM for a given last-level cache configuration. The estimator uses a model that is derived through linear regression and a small number of cycle-accurate simulations. In Section 4, we illustrate that our estimator is applicable across different applications. In addition, the estimator can be used for different memory sizes, although its coefficients need to be recomputed.
- ExCaD framework that integrates our estimator with a cycle-accurate processor-memory simulator and a cache simulator for fast exploration of last-level cache. We use a novel technique to transform hit and miss profile (that lacks timing information) of the last-level cache configuration into a DRAM idle profile, which is then processed to produce the total number of self-refresh PD cycles for the estimator. ExCaD uses only one cycle-accurate simulation per application, and hence can quickly explore last-level cache to select the configuration with maximum DRAM energy reduction.

2 RELATED WORK

Recent research has focused on reducing DRAM energy consumption, consisting of memory architectures and controller policies [3, 4, 5, 6, 7, 8, 16], cache replacement policies [9], OS-level policies [17, 18, 19, 20] and compiler-directed optimizations [21, 22, 23, 24, 25].

Delaluz et al. [3] predicted memory idle periods for cacheless systems to control the use of low-power modes. Fan et al. [4] extended their work for multi-level cache hierarchy through an analytical model of Rambus DRAM. The work in [5] proposed an adaptive history based scheduler with memory commands' throttling to increase memory idle periods. Liu et al. [6] introduced a temperature and power management policy by buffering DRAM write operations to improve page hit rate, resulting in reduction of DRAM active power. Prefetching of data from DRAM into an intermediate buffer has also been explored [7, 8] with the focus on reducing DRAM active power only. The authors of [16] proposed an architectural technique where a DRAM rank is divided into multiple mini-ranks to reduce the number of active devices for a memory access. In contrast to above memory architectures and controller policies, Amin et al. [9] proposed a cache replacement policy to skew cache-memory traffic to prolong DRAM idle periods for use of self-refresh PD mode.

OS-level approaches [17, 18, 19, 20], on the other hand, manage memory traffic at kernel layer through page migration, power-aware page allocation and similar policies. OS-level policies work at a much coarser granularity than memory controller policies. Compiler-directed approaches [21, 22, 23, 24] statically analyze application code to detect memory idle periods in addition to data access patterns to optimally place both code and data in DRAM.

Unlike all the above works where a pre-configured last-level cache was used, our work differs in its very proposal of exploring the last-level cache to improve DRAM energy efficiency. As such, our proposal is complimentary to above mentioned proposals.

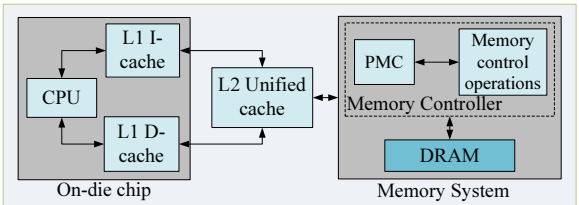

Figure 3.1: An example of target system.

3 Problem Statement

We target a uniprocessor system with multi-level cache hierarchy and DRAM based system memory. An example system is shown in Figure 3.1, where the processor has on-chip separate L1 instruction and data caches, which are connected to a unified off-chip L2 cache. The L2 cache is interfaced with DRAM memory through a memory controller to contain both application instructions and data. Here, L2 is the last-level cache and uses Least Recently Used (LRU) replacement policy with write-back mechanism. We use this system as an example throughout the paper in addition to its use in our experiments.

Power Mode Controller (PMC) is a module that is typically embedded into memory controller firmware to set DRAM into one of the low-power modes when it is idle [3, 4, 5, 6, 7, 8, 26]. DRAM memory also contains an internal mode controller which sets it into one of shallow low-power modes rather than the deepest low-power mode whenever DRAM is idle. PMC, on the other hand, typically uses a pre-determined threshold based policy or a history based adaptive policy to set DRAM into the deepest low-power mode (self-refresh PD). In our work, PMC transitions DRAM to self-refresh PD mode if it is idle for at least threshold cycles as used in [5, 9]. We use 30 cycles as the self-refresh PD threshold (PDthreshold) which was obtained through experimentation [5, 9].

Our goal is to determine which last-level cache configuration will maximally reduce energy consumption of DRAM in a uniprocessor system given a PDthreshold and other architectural parameters such as processor type, memory type, lower level caches, etc. remain unchanged. Parameters varied in last-level cache are its size, line size and associativity.

4 DRAM Energy Reduction Estimator

Estimation of DRAM energy consumption is a complex process [14, 15] due to the involvement of different DRAM states and their transitions during the execution of an application. The authors of [15] used a model consisting of twenty equations to estimate the power consumption of DRAM as accurately as possible. Their model used detailed trace of memory commands collected from a cycle-accurate processor-memory simulator. These memory controller traces change from one last-level cache configuration to another, and hence makes their detailed model unsuitable for fast exploration of last-level cache. During last-level cache exploration, the energy reduction trend between distinct cache configurations matter more than the actual energy consumption of DRAM, and it is this fact which we exploit in our estimator. In fact, the authors of [27] reported a similar observation, but for a general design space exploration problem.

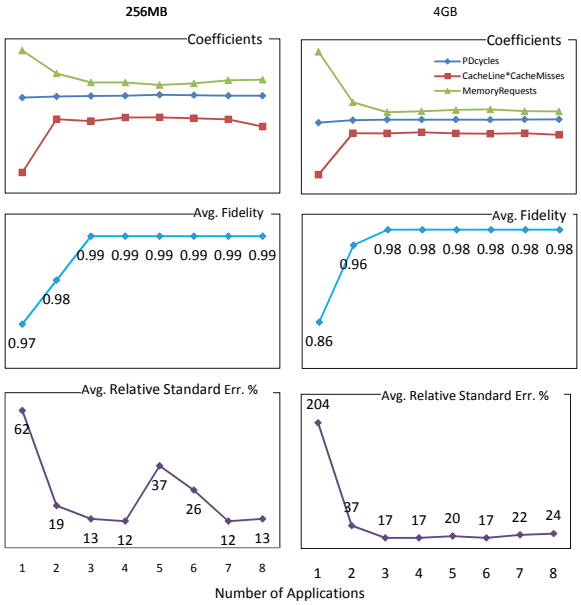

Figure 4.1: Estimator coefficients, average fidelity and relative standard error (%) against training set consisting of different number of applications for two DRAM sizes.

Our goal is to produce a high fidelity yet simple DRAM energy reduction estimator that uses minimum number of parameters rather than an absolutely accurate DRAM energy consumption estimator. We used the following three parameters to estimate the reduction in DRAM energy consumption:

- **PDcycles:** the total number of cycles for which DRAM is in PD mode during the execution of an application. This parameter depends on the total number of DRAM idle cycles and the PDthreshold. More PDcycles mean DRAM remains in PD mode for longer periods, providing more energy reduction.
- **Cache Line Size and Cache Misses:** The product of last-level cache line size and total number of last-level cache misses measures the amount of traffic going to DRAM during the execution of an application. More traffic to DRAM means that it will be active for longer time, increasing its energy consumption. We did not convert the amount of traffic into cycles using a fixed read and write latency because reading and writing from DRAM takes different number of cycles depending on its state [26], which is unknown until the cycle-accurate processor-memory simulation.
- **Memory Requests:** In DRAM, data is brought to a row-buffer before it can be accessed [14, 15]. If data from another row of DRAM needs to be accessed, then the current contents of row-buffer are written back before loading the row-buffer with new data. Hence, memory requests that result in more row-buffer conflicts will increase DRAM energy consumption. We approximate the number of row-buffer conflicts by the number of memory requests because actual number of row-buffer conflicts is not known until the cycle-accurate processor-memory simulation.

Using these parameters, the DRAM energy reduction estimator can be described

as:

$$E_r = \beta_0 PDcycles + \beta_1 [Cache\ Line\ Size \times Cache\ Misses] + \beta_2 MemoryRequests \quad (4.1)$$

where β_0, β_1 and β_2 are parameter coefficients. We used linear regression to compute the coefficients.

Determining Coefficients. We performed several experiments to collect data for the three parameters and the actual reduction in energy consumption of DRAM from a cycle-accurate processor-memory simulator. The training set consisted of three types of cache configurations: one, changing cache size with constant cache line size and associativity; two, changing cache line size with constant cache size and associativity; and three, changing associativity with constant cache size and line size. The constant cache parameters were set to both smallest and largest values. Such a training set considers all the possible values of individual cache parameters, and is based on the proposals in [28, 29]. Hence, the training set consisted of 44 out of 330 cache configurations for an application. The 330 cache configurations in the last-level cache design space were created by changing cache size from 4KB to 4MB, line size from 4B to 128B and associativity from 1 to 16. The 44 cache configurations in the training set included [4KB – 4MB, 4B, 1A], [4KB – 4MB, 128B, 16A], [4KB, 4B – 128B, 1A], [4MB, 4B – 128B, 16A], [4KB, 4B, 1A – 16A] and [4MB, 128B, 1A – 16A] configurations.

To determine how many applications should be used in the training set, we adopted the following methodology. First, we computed the estimator coefficients using the smallest application, then using two smallest applications, and then three smallest applications and so on until the maximum number of applications. Figure 4.1 illustrates the three estimator coefficients for two different memory sizes (256MB and 4GB) against the number of applications used in the training set (from a total of eight applications – adpcmenc/dec, jpegenc/dec, g721enc/dec and mpeg2enc/dec). The plot suggests that the estimator coefficients remain fairly constant for training sets with more than 2 applications. Hence, any combination of three or more applications could be used in the training set.

Evaluating Model's Accuracy and Fidelity. We evaluated the fidelity[1] and average relative standard error of the estimator by comparing the estimated energy reductions from the estimator with the actual energy reductions from cycle-accurate simulations for all the cache configurations. Figure 4.1 illustrates the average fidelity and average relative standard error of all the 8 applications. The average fidelity is above 0.99 when more than 2 applications are used in the training set. Note that our goal was to produce a high fidelity estimator; that is why in some cases high relative standard errors were observed.

For the rest of the paper, we trained the estimator on 44 cache configurations of each of the adpcmenc, jpegenc, g721enc and mpeg2enc applications to compute its coefficients. Estimator coefficients computed as such are used in the experimental section as well. For visual illustration, Figure 4.2 plots the actual and estimated DRAM energy reductions of all the cache configurations across the 8 applications for 256MB DRAM. Although estimated values have high absolute errors, the trend of both actual and estimated values is similar. The estimator for 4GB DRAM revealed similar findings. It

[1]Fidelity measures the correlation between the ordering of the actual and estimated values to quantify the similarity between the trends of actual and estimated values. A value close to 1 means that an analytical model has high fidelity. We used one of the fidelity metrics from [27].

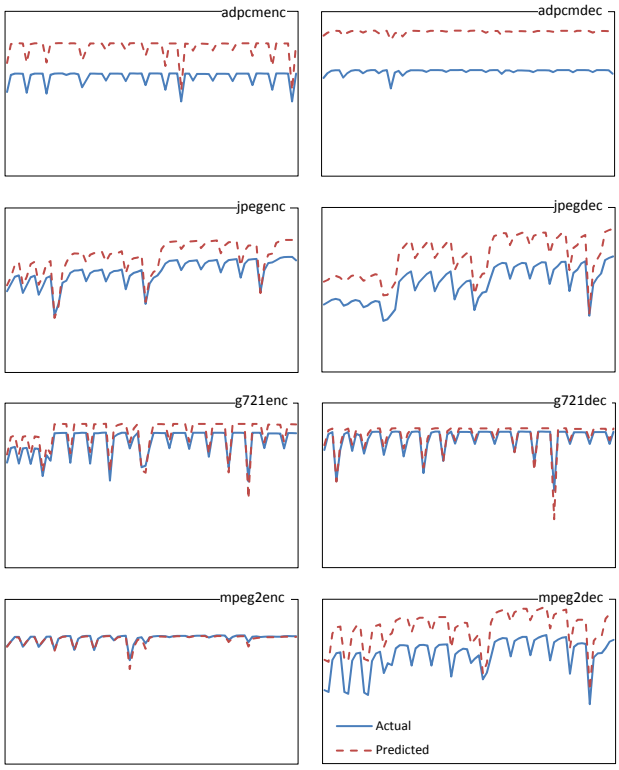

Figure 4.2: Actual and predicted DRAM energy reduction for L2 (last-level) cache configurations in the test set on target system of Figure 3.1 with 256MB DRAM. Although absolute accuracy is poor, the estimator has high fidelity.

is important to note that only $44 \times 4 = 176$ out of a total of $330 \times 8 = 2{,}640$ cycle-accurate simulations were used to compute estimator's coefficients. Furthermore, our estimator is applicable across different DRAM sizes; however, estimator coefficients need to be recomputed.

5 ExCaD Framework

Our framework, ExCaD, to quickly explore last-level cache in the context of DRAM energy reduction is shown in Figure 5.1. The input to ExCaD consists of: applications; last-level cache configurations; PDthreshold and last-level cache area constraint(s).

At high level, for each application, ExCaD analyzes all the requested cache configurations to produce the improvements in DRAM energy consumption. To this end, ExCaD cycle accurately simulates an application without last-level cache to record its memory trace. This memory trace is then fed to a cache simulator to record statistics for all the last-level cache configurations. Finally, our estimator is used to quantify the improvements in energy consumption of DRAM. Then, the designer can choose the cache configuration with maximum DRAM energy reduction. Alternatively, the designer can also find Pareto-optimal front to select a cache configuration with maximum energy reduction under an area constraint; however, in our experiments we select

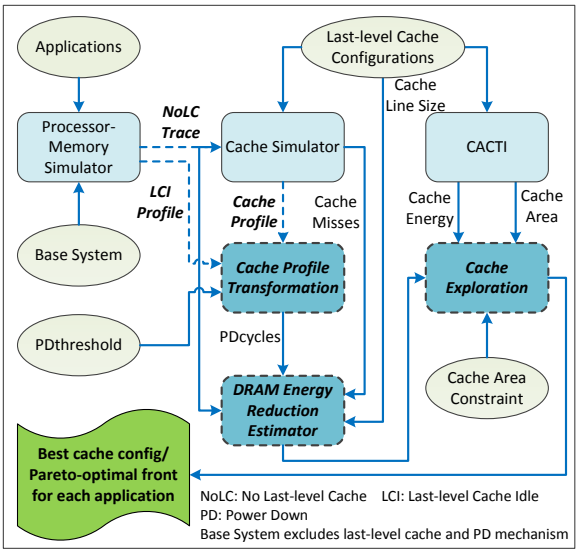

Figure 5.1: ExCaD Framework. Dotted-lined rectangles and broken arrows show our novel contributions.

the cache configuration with maximum DRAM energy reduction without any area constraint. The following paragraphs explain core components of ExCaD in more detail from the perspective of one application.

Application Trace Generation. An application is simulated in a cycle-accurate processor-memory simulator without last-level cache and PMC's power down mechanism to record two entities at the second-last-level cache and memory interface: (1) No Last-level Cache (NoLC) memory trace, and (2) Last-level Cache Idle (LCI) profile. NoLC trace will contain only those memory requests that will be missed in lower level caches. For instance, in Figure 3.1 without L2 cache, the memory trace captured at L1–memory interface will only contain L1 cache misses. The number of memory requests in NoLC trace are recorded for the third parameter of the estimator. *An LCI period refers to an application's execution period that does not access DRAM.* For instance, at L1–memory interface without L2 cache in Figure 3.1, LCI periods will be the execution periods with no memory requests from the application (consecutive non-load and non-store instructions) and execution periods with memory requests that will hit in L1 caches. Figure 5.2 illustrates such an LCI period, where L2 cache will not be accessed due to non-load and non-store instructions and hits in L1 cache. Hence, L2 cache will

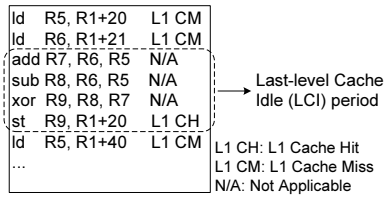

Figure 5.2: An example of LCI period for target system of Figure 3.1 (L2 is the last-level cache).

be idle during the marked LCI period. *An LCI profile captures all LCI periods in clock cycles from the execution of an application.* For an in-order processor, which is typical of embedded processors, an LCI profile of the application will not change across different last-level cache configurations because the processor pipeline is stalled during each memory request and lower level caches remain unchanged. Note that DRAM will be idle during all LCI periods and hence these periods will contribute to DRAM idle periods. The DRAM energy consumption from this simulation is used as reference point for calculation of energy reductions for all the last-level cache configurations. This is the only step in ExCaD where an application is cycle-accurately simulated.

Cache Simulator. In this step, we feed NoLC trace to a cache simulator to generate cache profile for each of the last-level cache configuration. *A cache profile reports whether DRAM will be accessed or not for each memory request in the application trace.* For each cache hit, DRAM will not be accessed; however, a cache write miss may or may not result in DRAM access in a write-back mechanism. The decision depends on the dirtiness of the data; that is, if data is already dirty then it will be written to DRAM resulting in its access, otherwise the data will only be marked dirty resulting in DRAM non-access. Note that in a write-through cache, each write miss will result in DRAM access.

Cache Profile Transformation. The DRAM energy reduction estimator uses total number of DRAM PDcycles which is missing in the cache profile. Here, we transform DRAM access and non-access information from the cache profile into a more useful representation, DRAM Idle (DI) profile. *A DI profile captures the duration of each DI period from a given cache profile and LCI profile of the application where a DI period refers to consecutive cycles of DRAM idleness.* Both DRAM non-accesses from cache profile and LCI periods from LCI profile will contribute to DI periods. Hence, first we insert LCI periods from LCI profile between appropriate DRAM accesses and non-accesses in the cache profile. Then, all DI periods are marked where an idle period consists of consecutive DRAM non-accesses and LCI periods. Note that a DRAM non-access will contribute same number of cycles as last-level cache hit latency because it is equivalent to last-level cache hit.

Figure 5.3 shows an example of how PDcycles are derived from a cache profile. The cache profile reports DRAM accesses and non-accesses while the LCI profile reports L2 cache idle periods. First, each LCI period is inserted after appropriate Memory Request (MR) in the cache profile. For example, second LCI period occurs after MR3 and hence is inserted between MR3 and MR4 in the DI profile. The DI profile shows all the DI periods marked in dotted-lined rectangles. The cycles in each DI period are calculated using a 2 cycle DRAM non-access latency (L2 cache hit latency). For example, for the first DI period, there are 20 and 110 cycles from two LCI periods and 8 cycles from 4 DRAM non-accesses, totalling to 138 cycles.

Finally, the DI profile is converted to DRAM PDcycles by applying the pre-determined PDthreshold. In each DI period, the initial threshold number of cycles will not contribute to PDcycles. Likewise, all DI periods with duration less than PDthreshold will be filtered. For example, with a PDthreshold of 30 cycles in Figure 5.3, the last DI period will be filtered since its duration is only 24 cycles. For the rest of DI periods, 30 cycles are subtracted to calculate a total of 148 PDcycles. Hence, for this example, DRAM will remain in PD mode for 148 cycles. Note that this step allows a designer to apply any threshold that he/she deems suitable. Our Cache Profile Transformation accurately captures DI periods, however PDcycles might not be perfectly accurate because DRAM consumes arbitrary yet small number of cycles before transitioning into PD mode depending on the state of DRAM; however, this small inaccuracy did not

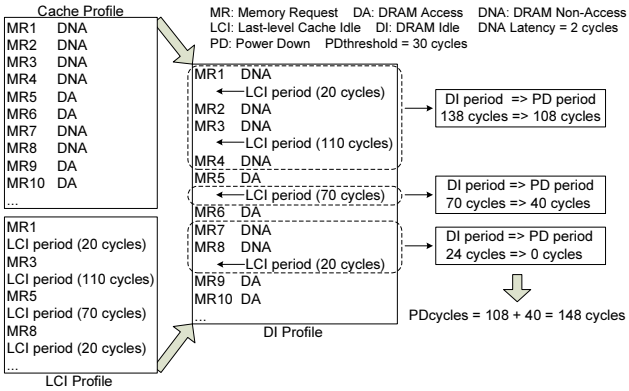

Figure 5.3: An example of Cache Profile Transformation for target system of Figure 3.1 (L2 is the last-level cache).

affect the selection of most suitable last-level cache configuration (see Section 6).

Cache Exploration. At the final step, DRAM energy reduction is computed for each last-level cache configuration using the PDcycles from cache profile transformation, cache misses from cache simulator, cache line size and number of memory requests from NoLC trace. The predicted energy reductions are adjusted by subtracting the energy consumptions of respective cache configurations which are obtained from CACTI [30]. Once net DRAM energy reductions for all last-level cache configurations are available, the cache configuration with maximum DRAM energy reduction is selected. We argue that an exhaustive search of the design space to find an optimal cache configuration is better than a heuristic search to find a near-optimal cache configuration because 1) estimators are simple and fast (see Section 6), and 2) the number of last-level cache configurations is in hundreds rather than thousands or millions (see Section 6).

5.1 Advantages & Limitations

ExCaD features several advantages over cycle-accurate simulations and standalone cache simulators. ExCaD: (1) is fast as it only uses one cycle-accurate simulation per application, (2) integrates cache simulator and cache profile transformation technique to quickly provide DRAM PDcycles for all last-level cache configurations and (3) uses DRAM energy reduction estimator for fast exploration of last-level cache configurations. Although several cycle-accurate simulations are required to build the DRAM energy reduction estimator, the number of simulations is far less than the whole design space (176 out of a total of 2,640 simulations, see Section 4) and are performed only once using the training set.

ExCaD is very flexible as any processor-memory simulator can be used given memory trace at second-last-level cache and memory can be captured. Likewise, any cache simulator can be used given above defined cache profile can be produced which requires minor modifications to already available cache simulators [11, 12, 13]. Finally, the designer has the flexibility to use PDthreshold according to his/her system under design.

ExCaD can also be used to find best last-level cache configuration for a class of applications. In this case, the trace from combined execution of the applications should

be captured and input to cache simulation. A truly representative trace from multiple applications' execution might not be possible due to indeterminism; however, this is a different problem and not the focus of our work. ExCaD will explore last-level cache configurations to find the best one for a given trace irrespective of whether that trace captures execution of single or multiple applications.

ExCaD as such cannot explore last-level cache in a multi-processor system because the LCI periods will be different across different last-level cache configurations (unlike a uniprocessor system where LCI periods are the same across different last-level configurations) due to inter-processor dependencies and cache coherency. In future, we will look into extending ExCaD for multi-processor systems.

6 EXPERIMENTS & RESULTS

We evaluate ExCaD framework by exploring L2 cache configurations in the target system of Figure 3.1. The target system is implemented using Tensilica's LX4 processor [31] with 2KB L1 instruction and 1 KB L1 data caches, both direct-mapped with 4B line size. We used -125E DDR3 DRAM (1600 Million Transfers per second) from Micron [14] to implement system memory. We created two target systems with 256MB and 4GB memories to observe the scalability of our estimator. The 256MB and 4GB memories were created using 1Gbit MT41J64M16 and 1Gbit MT41J256M4 types respectively, all from Micron [14]. These DRAMs had an interface width of 4B and used open-page row buffer policy and internal refresh time of 64 ms. A threshold of 30 cycles and a wakeup latency of 64 cycles from Table 1.1 was used for self-refresh Power Down (PD) mode to implement the PD mechanism for DRAM.

We used the cycle-accurate simulator from [26][1] and DineroIV [12] as processor-memory simulator and cache simulator in ExCaD. In addition, CACTI 6.5 [30] was configured for a given 90nm technology to obtain energy consumption and area of L2 cache configurations. For each application and DRAM size, 330 L2 cache configurations were explored by changing cache size from 4KB to 4MB, line size from 4B to 128B and associativity from 1 to 16. These configurations are typically explored in an embedded system's design [10, 11]. We computed estimator coefficients for the two DRAM sizes as described in Section 4. For evaluation purposes, we used adpcmenc/dec, jpegenc/dec, g721enc/dec, mpeg2enc/dec applications from mediabench [2]. All experiments were conducted on an Intel Xeon 64 core machine with 256GB RAM.

For each application, we used ExCaD to explore L2 cache to select the cache configuration with maximum DRAM energy reduction for the two DRAM sizes. Table 6.1 reports the L2 cache configurations selected by ExCaD with their areas. Out of 16 cache configurations (two configurations for two different DRAM sizes per application), ExCaD found the optimal cache configurations 14 times where an optimal cache configuration refers to the one with maximum DRAM energy reduction from cycle-accurate processor-memory simulations. The suboptimal configurations had a maximum increase of just 3% in the energy consumption of DRAM.

Once the best cache configuration was known from ExCaD, for comparison purposes, we simulated the following four systems in processor-memory simulator to obtain their actual DRAM energy reduction:

[1]Their cycle-accurate simulator integrated DRAMSim [32] with LX4's simulator to create a detailed processor-memory simulator.

- **BS:** Base system without L2 cache and PD mechanism.
- **LC_PD:** System with Largest Cache ([4MB, 128B, 16A]) and PD mechanism.
- **BC_SPD:** System with Best Cache configuration (maximum DRAM energy reduction from ExCaD) and DRAM's standard (internal) mode controller that transitions DRAM to only shallow low-power modes.
- **BC_PD:** System with Best Cache configuration and PD mechanism.

The energy reduction comparison of above four systems for all the 8 applications is reported in Figures 6.1 – 6.4. Here, we only discuss the interesting results of the smallest application (adpcmdec) and the largest application (mpeg2enc). Figures 6.1 and 6.4 report the normalized energy consumption breakdown for adpcmdec and mpeg2enc applications.

Apps.	256MB DRAM	4GB DRAM
adpcmenc	[4KB, 16B, 16A] (0.027)	[256KB, 64B, 2A] (0.632)
adpcmdec	[4KB, 16B, 16A] (0.027)	[8KB, 16B, 16A] (0.049)
jpegenc	[256KB, 128B, 1A] (0.523)	[256KB, 128B, 1A] (0.523)
jpegdec	[64KB, 16B, 16A] (0.31)	[64KB, 16B, 16A] (0.31)
g721enc	[8KB, 16B, 16A] (0.0489)	[8KB, 16B, 16A] (0.0489)
g721dec	[8KB, 16B, 16A] (0.0489)	[8KB, 16B, 16A] (0.0489)
mpeg2enc	[16KB, 64B, 8A] (0.043)	[1MB, 128B, 2A] (1.501)
mpeg2dec	[16KB, 16B, 16A] (0.084)	[256KB, 128B, 8A] (0.052)

Table 6.1: L2 cache configurations with maximum DRAM energy reduction (BC_PD) from ExCaD for different DRAM sizes. The numbers in parentheses are areas in mm^2.

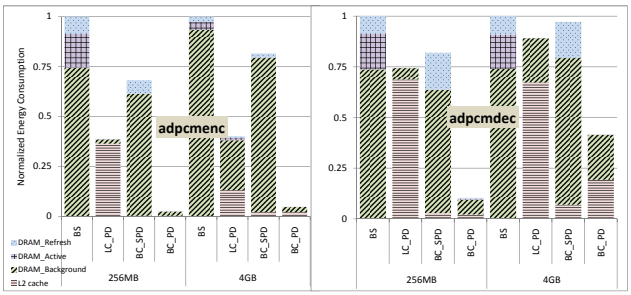

Figure 6.1: Normalized DRAM energy consumption breakdown of adpcmenc/dec for different L2 caches and DRAM sizes.

Since LC_PD system uses the largest cache, it significantly reduced DRAM's active, background and refresh energy consumptions due to very low cache misses and PD mechanism. The BC_PD system reduces DRAM energy consumption on par with LC_PD; however its total energy consumption is less than that of LC_PD due to the high energy consumption of the largest cache itself. In addition, LC_PD cache's area was 6.21 mm^2 compared to only 0.038 mm^2 and 0.772 mm^2 for adpcmdec and mpeg2enc applications which is an increase of 99% and 88% respectively. This illustrates that largest cache (or a randomly selected cache) is not an optimal option when the overheads of cache itself are considered.

The BC_SPD system uses DRAM's internal mode controller. Unlike PMC, internal mode controller transitions DRAM to only shallow low-power modes rather than

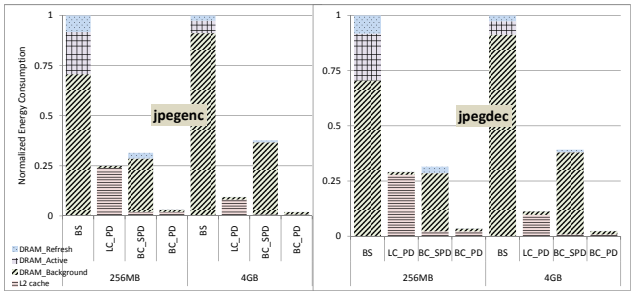

Figure 6.2: Normalized DRAM energy consumption breakdown of jpegenc/dec for different L2 caches and DRAM sizes.

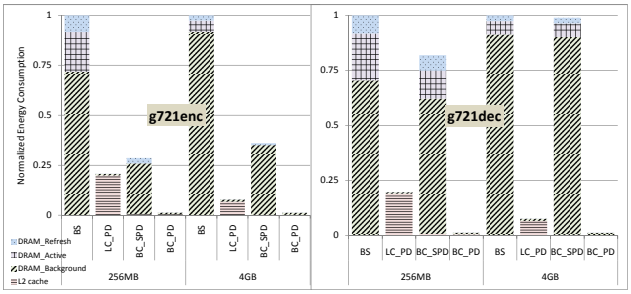

Figure 6.3: Normalized DRAM energy consumption breakdown of g721enc/dec for different L2 caches and DRAM sizes.

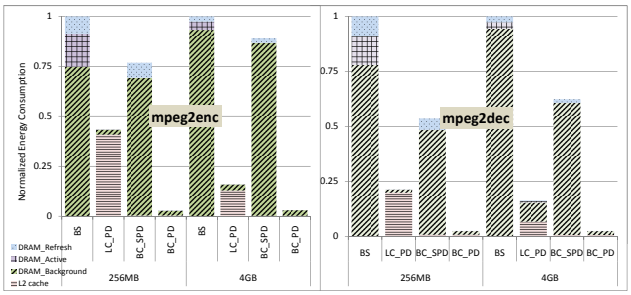

Figure 6.4: Normalized DRAM energy consumption breakdown of mpeg2enc/dec for different L2 caches and DRAM sizes.

the deepest low-power mode and comes as the default mode controller with DRAM memories. The BC_PD system is more energy efficient than BC_SPD configuration because it saves DRAM background energy as illustrated in Figures 6.1 and 6.4. The performance penalty of PD mechanism (wakeup latency of PD mode) can be measured by comparing the execution times of BC_PD and BC_SPD systems.

The normalized performance result for adpcmdec and mpeg2enc applications is reported in Figures 6.5 – 6.8 where the normalization is done w.r.t BS. In these applications, a maximum penalty of 1% was observed due to PD mechanism when BC_SPD and BC_PD systems are compared. These results illustrate the fact that a suitable cache not only increases DRAM idle periods but also consolidates them into longer periods to make them suitable for PD mode and to reduce the number of DRAM wakeups,

hence reducing the overall performance penalty. It is important to note that none of the systems incurred any performance penalty when compared to BS system because the reduction in execution time due to cache hits amortized the wakeup latency of PD mechanism. The above results point to the fact that PD mechanism with a suitable last-level cache can tremendously increase DRAM energy efficiency.

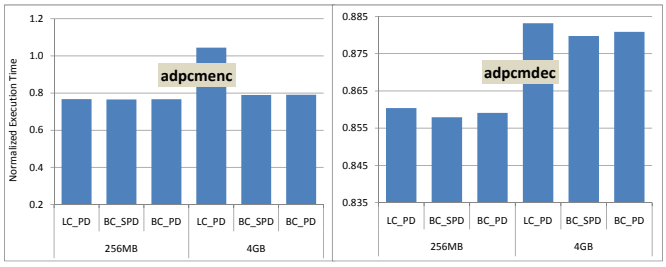

Figure 6.5: Normalized (to BS) Execution time of adpcmenc/dec for different L2 caches and DRAM sizes.

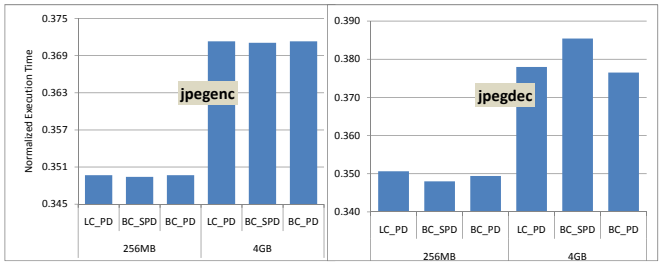

Figure 6.6: Normalized (to BS) Execution time of jpegenc/dec for different L2 caches and DRAM sizes.

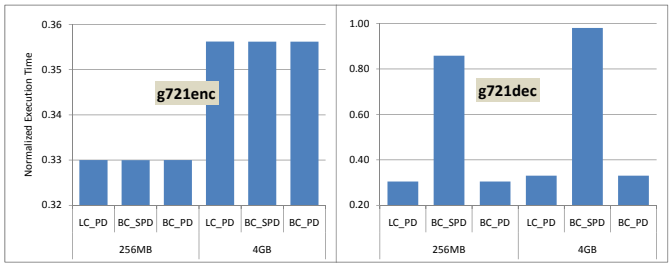

Figure 6.7: Normalized (to BS) Execution time g721enc/dec for different L2 caches and DRAM sizes.

In summary, BC_PD system from ExCaD saved a minimum of 95%, 53% and 47% DRAM energy when compared to BS, LC_PD and BC_SPD systems respectively for all applications and DRAM sizes. On average, BC_PD system saved 95%, 85% and 89% DRAM energy when compared to BS, LC_PD and BC_SPD systems respectively. These results indicate the usefulness of ExCaD in selecting a suitable cache configuration through our high fidelity DRAM energy reduction estimator and cache profile transformation technique.

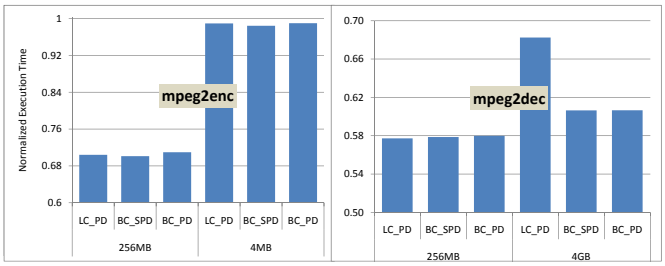

Figure 6.8: Normalized (to BS) Execution time of mpeg2enc/dec for different L2 caches and DRAM sizes.

Application	Processor-memory simulator	ExCaD
adpcmenc	3h	20m
adpcmdec	2h	14m
jpegenc	19h	3h
jpegdec	7h	1h
g721enc	13d	2d
g721dec	14d	2d
mpeg2enc	144d	8d
mpeg2dec	46d	4d

Table 6.2: Time comparison of cycle-accurate processor-memory simulator and ExCaD for exploration of 330 L2 cache configurations for only one DRAM size.

Table 6.2 reports the time taken by cycle-accurate processor-memory simulator and ExCaD for exploration of 330 L2 cache configurations for each of the eight applications for only one DRAM size. ExCaD reduces exploration time from several days to a few hours, enabling quick exploration of last-level cache for DRAM energy reduction.

7 Conclusion

In this paper, we have proposed the exploration of last-level cache in a uniprocessor system to improve DRAM energy efficiency. To this end, we propose the ExCaD framework which integrates a cache simulator with a simple yet high fidelity DRAM energy reduction estimator and cache profile transformation technique to avoid slow cycle-accurate processor-memory simulations. This enables fast exploration of last-level cache in ExCaD to select the configuration that provides maximum DRAM energy reduction. Our results illustrate that the last-level cache configuration selected through ExCaD reduces DRAM energy consumption by at least 96% and 34% over systems without last-level cache and with largest last-level cache respectively. Furthermore, the cache configuration from ExCaD when used with self-refresh power down mode saved at least 93% more DRAM energy compared to the use of shallow low-power modes only. These results indicate that a suitable last-level cache configuration with self-refresh power mode can tremendously improve DRAM energy efficiency. ExCaD took only a few hours to explore last-level cache configurations compared to several days of cycle-accurate processor-memory simulations.

Bibliography

[1] F. Catthoor, E. d. Greef, and S. Suytack, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[2] "MMediabench benchmark suite." http://euler.slu.edu/ fritts/mediabench/.

[3] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "Dram energy management using software and hardware directed power mode control," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, (Washington, DC, USA), pp. 159–, IEEE Computer Society, 2001.

[4] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for dram power management," in *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, (New York, NY, USA), pp. 129–134, ACM, 2001.

[5] I. Hur and C. Lin, "A comprehensive approach to dram power management," in *HPCA*, pp. 305–316, 2008.

[6] S. Liu, S. Ogrenci Memik, Y. Zhang, and G. Memik, "An approach for adaptive dram temperature and power management," in *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, (New York, NY, USA), pp. 63–72, ACM, 2008.

[7] J. Lin, H. Zheng, Z. Zhu, Z. Zhang, and H. David, "Dram-level prefetching for fully-buffered dimm: Design, performance and power saving," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pp. 94 –104, April 2007.

[8] J. Trajkovic, A. V. Veidenbaum, and A. Kejariwal, "Improving sdram access energy efficiency for low-power embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 24:1–24:21, May 2008.

[9] A. M. Amin and Z. A. Chishti, "Rank-aware cache replacement and write buffering to improve dram energy efficiency," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED '10, (New York, NY, USA), pp. 383–388, ACM, 2010.

[10] L. Benini, A. Macii, and M. Poncino, "Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques," *ACM Trans. Embed. Comput. Syst.*, vol. 2, pp. 5–32, February 2003.

[11] M. Haque, J. Peddersen, A. Janapsatya, and S. Parameswaran, "Dew: A fast level 1 cache simulation approach for embedded processors with fifo replacement policy," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 496 –501, march 2010.

[12] J. Edler and M. D. Hill, "Dinero iv trace-driven uniprocessor cache simulator." http://pages.cs.wisc.edu/ markhill/DineroIV/.

[13] N. Tojo, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Exact and fast l1 cache simulation for embedded systems," in *ASP-DAC '09: Proceedings of the 2009 Conference on Asia and South Pacific Design Automation*, (Piscataway, NJ, USA), pp. 817–822, IEEE Press, 2009.

[14] I. Micron, "Micron ddr3." http://www.micron.com/products/dram/ddr3/.

[15] K. Chandrasekar, B. Akesson, and K. Goossens, "Improved power modeling of ddr sdrams," in *DSD*, pp. 99–108, 2011.

[16] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," *Microarchitecture, IEEE/ACM International Symposium on*, vol. 0, pp. 210–221, 2008.

[17] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," *SIGPLAN Not.*, vol. 35, pp. 105–116, November 2000.

[18] Y.-H. Lu, L. Benini, and G. De Micheli, "Operating-system directed power reduction," in *Proceedings of the 2000 international symposium on Low power electronics and design*, ISLPED '00, (New York, NY, USA), pp. 37–42, ACM, 2000.

[19] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, "Improving energy efficiency by making dram less randomly accessed," in *Proceedings of the 2005 international symposium on Low power electronics and design*, ISLPED '05, (New York, NY, USA), pp. 393–398, ACM, 2005.

[20] M. Lee, E. Seo, J. Lee, and J. soo Kim, "Pabc: Power-aware buffer cache management for low power consumption," *IEEE Transactions on Computers*, vol. 56, pp. 488–501, 2007.

[21] Z. Wang and X. S. Hu, "Power aware variable partitioning and instruction scheduling for multiple memory banks," in *Proceedings of the conference on Design, automation and test in Europe - Volume 1*, DATE '04, (Washington, DC, USA), pp. 10312–, IEEE Computer Society, 2004.

[22] M. Kandemir, "Impact of data transformations on memory bank locality," in *Proceedings of the conference on Design, automation and test in Europe - Volume 1*, DATE '04, (Washington, DC, USA), pp. 10506–, IEEE Computer Society, 2004.

[23] C.-G. Lyuh and T. Kim, "Memory access scheduling and binding considering energy minimization in multi-bank memory systems," in *Proceedings of the 41st annual Design Automation Conference*, DAC '04, (New York, NY, USA), pp. 81–86, ACM, 2004.

[24] G. Chen, F. Li, and M. Kandemir, "Compiler-directed channel allocation for saving power in on-chip networks," *SIGPLAN Not.*, vol. 41, pp. 194–205, January 2006.

[25] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Energy-oriented compiler optimizations for partitioned memory architectures," in *Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '00, (New York, NY, USA), pp. 138–147, ACM, 2000.

[26] S. Min, J. Peddersen, and S. Parameswaran, "Realizing cycle accurate processor memory simulation via interface abstraction," in *VLSI Design (VLSI Design), 2011 24th International Conference on*, pp. 141 –146, jan. 2011.

[27] H. Javaid, A. Ignjatovic, and S. Parameswaran, "Fidelity metrics for estimation models," in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pp. 1 –8, 2010.

[28] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, HPCA '03, (Washington, DC, USA), pp. 281–, IEEE Computer Society, 2003.

[29] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," *SIGARCH Comput. Archit. News*, vol. 34, pp. 185–194, Oct. 2006.

[30] HP, "Cacti 6.5." http://www.hpl.hp.com/research/cacti/.

[31] Tensilica, Inc., "Xtensa Configurable Processors." http://www.tensilica.com.

[32] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. L. Jacob, "Dramsim: a memory system simulator," *SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 100–107, 2005.