Improving Energy Efficiency of Embedded DRAM Caches for High-end Computing Systems

Sparsh Mittal
Oak Ridge National Laboratory
mittals@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory and Georgia Institute of Technology
vetter@computer.org

Dong Li
Oak Ridge National Laboratory
lid1@ornl.gov

ABSTRACT

The number of cores in a single chip in the nodes of high-end computing systems is on rise, due, in part, to a number of constraints, such as power consumption. With this, the size of the last level cache (LLC) has also increased significantly. Since LLCs built with SRAM consume high leakage power, power consumption of LLCs is becoming a significant fraction of processor power consumption. To address this issue, researchers have used embedded DRAM (eDRAM) LLCs which consume low leakage power. However, eDRAM caches consume a significant amount of energy in the form of refresh energy. In this paper, we propose ESTEEM, an energy saving technique for embedded DRAM caches. ESTEEM uses dynamic cache reconfiguration to turn off a portion of the cache to save both leakage and refresh energy. It logically divides the cache sets into multiple modules and turns off possibly different number of ways in each module. Microarchitectural simulations confirm that ESTEEM is effective in improving performance and energy efficiency and provides better results compared to a recently-proposed eDRAM cache energy saving technique, namely Refrint. For single and dual-core simulations, the average energy saving in memory subsystem (LLC+main memory) with ESTEEM is 25.8% and 32.6% respectively, and the average weighted speedup is 1.09× and 1.22× respectively. Additional experiments confirm that ESTEEM works well for a wide-range of system and algorithm parameters.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Embedded DRAM (eDRAM) cache, low-power, cache reconfiguration, refresh energy saving, leakage energy saving.

1. INTRODUCTION

Managing power consumption of high-end computing systems is extremely important to continue to scale their performance and avoid intolerable operating costs and failure rates [6, 13]. The Tianhe-2 supercomputer with largest performance in the top500 list [3] consumes 17 mega-watt power, which is enough to sustain a city of more than 50,000 people. Since the future exascale systems have a 20MW power budget, an exascale machine built with the technology used in modern supercomputers would consume giga-watts of power. Further, for every watt of power dissipated in the computing systems, an additional 0.5 to 1W of power is consumed by the cooling system also [33]. Given such tight power budgets, energy efficiency of current high-end computing systems must be significantly improved to achieve future exascale computing. For these reasons, energy efficiency has now become a first-order constraint in the design of high-end computing systems.

As single-core processor performance becomes power limited, processor designers are using large number of on-chip cores to improve performance. To feed data to these cores and offset the limitations posed by off-chip memory bandwidth, modern processors use large-size last level caches (LLCs) [28]. For example, Intel's Enterprise Xeon processor uses 30 MB LLC [18]. Conventionally, SRAM has been used to design on-chip caches due to its low access-latency. However, SRAM also consumes large leakage power and hence, large last level caches (LLCs) designed with SRAM consume significant fraction of processor power. As an example, leakage power of last level cache accounts for 20% and 30% of the total power in Intel Core 2 Penryn and Intel Xeon Tulsa processors [26]. Further, it has been shown that if large caches are designed with SRAM, they may occupy 90% of the chip-area in upcoming fourth CMOS generation [40] because of the relatively low density of SRAM.

To overcome the limitations of SRAM (i.e., high leakage power and low density), researchers have recently explored alternative device technologies such as non-volatile memory (NVM) and embedded DRAM (eDRAM) for designing on-chip caches. While NVMs, such as STT-RAM (spin transfer torque RAM) and ReRAM (resistive RAM), have the advantage of near-zero leakage energy and high-density, their limited write endurance and high write-latency [36] present a critical bottleneck in enabling their use for on-chip caches. EDRAM has the advantage of low-leakage (nearly 1/8th leakage power consumption compared to SRAM [4]), compatibility with CMOS process, and high write endurance.

These features make them suitable for use as on-chip caches. For this reason, eDRAM has been used to design the LLCs in IBM's Power 7 processor [21] and Blue Gene/L supercomputer chip [19]. Also, Intel's Haswell processor uses 128MB eDRAM L4 cache [25].

A critical limitation of eDRAM cells, however, is that they lose charge over time and, hence, require refresh operations to maintain data integrity. Thus, to avoid failures, an eDRAM cell must be refreshed before its retention period, which is the duration of time for which the cell can retain its state. More precisely, compared to the conventional DRAM, eDRAM uses faster logic transistors with high leakage current and hence, the retention period of eDRAM is in the range of tens of microseconds (e.g. $40\mu s$ [8]), which is nearly a thousand times shorter than that of conventional DRAM, which is in range of 64ms [45]. It has been demonstrated that refresh energy accounts to nearly 70% of the total energy in eDRAM LLCs, while the leakage energy accounts for most of the remaining fraction [4]. With ongoing CMOS scaling, this retention period is expected to reduce further due to increasing leakage and smaller storage capacitance [11], which will increase the overhead of refresh even further. Thus, reducing the refresh energy consumption of eDRAM is extremely important to enable their wide-spread use and also avoid complex cooling solutions (e.g. liquid cooling).

1.1 Contributions

In this paper, we present ESTEEM, an energy saving technique for embedded DRAM caches. ESTEEM uses periodic cache reconfiguration to turn-off a portion of LLC and avoids refreshing it. This leads to saving in both leakage and refresh energy (Section 3). Further, in the active portion of cache, only valid blocks are refreshed. For cache reconfiguration, ESTEEM logically divides the cache-sets into different modules. For example, with 4096 sets and 16 modules, each module has 256 sets. Then, in each module, only the required number of ways are kept active, such that the performance is not affected while largest possible saving in energy is achieved (Section 4). ESTEEM does not require offline profiling or manual tuning of its parameters. Also, its energy saving algorithm runs in software and uses lightweight hardware support. The overhead of ESTEEM is less than 0.1% of the L2 cache size (Section 5).

We perform single and dual-core microarchitectural simulations using an x86-64 simulator and workloads from SPEC06 suite and HPC (high-performance computing) field (Section 6). Also, we compare ESTEEM with a recently proposed technique for saving refresh energy in eDRAM caches, named Refrint polyphase-valid (RPV) ([4], Section 6.2). The experiments have shown that ESTEEM provides better performance and energy efficiency than RPV. For $50\mu s$ retention period and a baseline eDRAM LLC (which periodically refreshes all cache lines), the energy saving achieved using ESTEEM and RPV, for single-core system is 25.82% and 15.93%, respectively. For dual-core system, these values are 32.63% and 14.39%, respectively. ESTEEM also provides higher performance than both baseline and RPV. Experiments with $40\mu s$ retention period show that with lower retention period, the advantage of ESTEEM increases even further. Additional experiments show that ESTEEM works well for a wide range of system and algorithm parameters.

The major contributions of this paper are:

- We propose a dynamic cache reconfiguration technique for saving both leakage and refresh energy in eDRAM caches. Our technique addresses the major challenge that prevents eDRAM to be used as a viable and scalable solution for future high-end computing systems.
- We evaluate our technique with a spectrum of scientific computing applications and over a wide-range of system/algorithm parameters. After detailed comparison with a state-of-the-art energy saving mechanism for eDRAM cache, we demonstrate the effectiveness of our technique.

The remainder of the paper is organized as follows. Section 2 presents the background and related work on eDRAM and power management techniques for caches. Section 3 describes the working of ESTEEM. Section 4 presents the energy saving algorithm of ESTEEM and Section 5 presents its implementation details. Section 6 presents simulation platform, workloads, energy model and evaluation metrics. Section 7 presents the simulation results. Finally, Section 8 presents the conclusion.

2. BACKGROUND AND RELATED WORK

The eDRAM cells are generally of two types, namely the gain cell eDRAM and the 1T1C eDRAM [11]. Both of these cells store data in the form of capacitor. For example, a gain cell utilizes the gate capacitance of its storage transistor and a 1T1C cell utilizes a dedicated capacitor to store its data. In this paper, we assume a gain cell eDRAM as the basis of our study.

Recently, several techniques have been proposed to mitigate the refresh energy in eDRAM devices. Some researchers propose use of error-detection/correction based approaches [39,45] which allow increasing the refresh period by tolerating some failures. Some researchers propose techniques to detect dead-blocks and avoid refreshing them to save refresh energy [4,11]. The Smart-Refresh technique [15] avoids refreshing the DRAM rows which are recently read or written. Reohr [38] discusses several approaches for optimizing refresh operations in eDRAM caches, for example, no-refresh, periodic refresh and line-level refresh based on time stamps. In this paper, we use cache-reconfiguration approach to save energy in eDRAM caches.

In literature, several cache reconfiguration techniques have been proposed to save leakage energy in SRAM caches [28]. On the basis of granularity of cache reconfiguration, these techniques can be divided into several categories, such as selective-sets [34], selective-ways [5], hybrid (selective-sets and ways) [30], cache-coloring [29], cache block-level [22–24] etc. ESTEEM uses selective-ways based cache reconfiguration approach, which has low implementation and reconfiguration overhead. Also, unlike selective-sets or cache coloring approach, selective-ways approach does not require a change in set-decoding on cache reconfiguration.

Some cache reconfiguration techniques (e.g. [5, 20]) statically reconfigure the cache and do not allow dynamic reconfiguration. In contrast, ESTEEM uses dynamic cache reconfiguration to easily adapt to intra-application variation in cache demand and provide large energy savings. For SRAM caches, FlexiWay [31] proposes cache reconfiguration at fine-granularity. However, FlexiWay uses complex and higher-overhead scheme for predicting LLC and memory energy, which is likely to be inaccurate due to dynamic behavior of workloads. By comparison, ESTEEM does not require prediction of energy. ESTEEM proposes a novel insight of simultaneously attacking leakage and refresh energy in eDRAM, whereas refresh operations do not happen in SRAM in case of FlexiWay. Also, compared to SRAM, presence of refresh in eDRAM totally changes the relative contribution of dynamic, leakage and refresh energies, thus changing the energy optimization scenario.

To leverage the different features of different memory cells, some researchers have proposed hybrid memory cells. Valero et al. [43] propose a macro-cell that combines SRAM and eDRAM at cell level. They implement an A-way set-associative cache with these macro-cells which consists of one SRAM cell, $A-1$ eDRAM cells and a transistor that acts as a bridge for transferring data between static and dynamic cells. Their approach is suitable for L1 caches but does not work well for lower-level caches. This is because due to filtering of access-stream by L1 cache, the access patterns at lower-level caches are not very predictable.

3. METHODOLOGY

Notations: Let N denote the number of cores. Let S, A, M denote the number of cache sets, cache associativity and number of cache modules, respectively. Let B and G denote the cache-line(block) size and tag size, respectively, which, in this paper, are taken as 512bits (64 byte) and 40 bits, respectively. In this paper, we use the terms 'cache line' and 'cache block', interchangeably.

3.1 Main Idea

Our technique works on the key idea that there exists large intra-application and inter-application variation in the cache requirement of different applications. Thus, by allocating just the right amount of cache to each application, the rest of the cache can be transitioned to low-power state, with minimum performance loss. This leads to reduction in the active-fraction of the cache. This saves both leakage and refresh energy as the inactive area of the cache need not be refreshed and also does not consume leakage energy. Further, in the active portion of the cache, only the valid blocks are refreshed, which further reduces the refresh energy.

To dynamically reconfigure the cache, we use the following approach. It has been shown that the associativity requirement of applications varies across different sets of the cache [31,41]. Conventional way-based reconfiguration techniques (e.g. [5]), turn-off exactly the same number of ways across all the sets. This, however, may lead to loss of flexibility which may lead to performance and/or energy penalty. To address this, we turn-off *possibly different* number of ways in each cache module. This is illustrated in Figure 1. This approach allows fine-grain cache reconfiguration, which also enables achieving a fine-balance between performance loss and energy saving.

To identify the blocks which store dead data (i.e. which are unlikely to be reused) in a low-overhead manner, we use the following observation. The LRU (least recently used) replacement policy works by ordering the cache lines based on their recency (or "age") and evicting the LRU-block on a cache miss. The intuition behind working of LRU is that the older (i.e. least recent or lower in LRU-stack) cache lines are less likely to be reused. Thus, the number of hits are expected to decrease with decreasing recency positions [7]. Hence, the cache ways with less recent positions are suitable

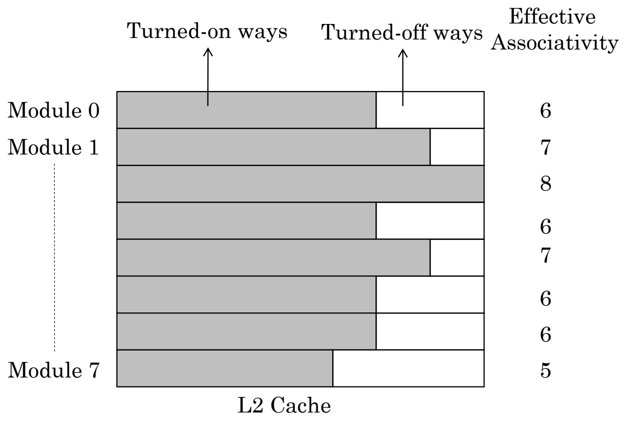

Figure 1: An illustration of ESTEEM approach for an 8-way L2 cache with 8 modules

candidates are turning-off, since turning these ways off leads to minimal impact on performance.

An exception to the above observation happens for the applications which do not show LRU behavior, i.e. for which the hits to the LRU-positions do not decrease monotonically with decreasing recency positions. Examples of such "non-LRU" applications from SPEC2006 suite include omnetpp, xalancbmk etc. [42]. To avoid incurring energy loss for such applications, ESTEEM detects whether in an interval, hits in a cache module show non-LRU pattern. This is detected by noting the hits to different LRU positions and seeing when the number of hits do not decrease monotonically with decreasing LRU-recency position. For such modules, the number of active-ways is not reduced below $A - 1$. In other words, for non-LRU applications, the aggressiveness of cache reconfiguration is reduced.

To decide the exact number of ways to turn-off, we use the following idea: if the total number of hits in all ways of a module are H, then we keep X ($\leq A$) ways turned-on, such that the total hits in X ways are equal to or greater than αH, where $\alpha < 1$ is a parameter. In other words, we keep a number of ways turned-on to cover at least α fraction of cache hits. We show an example to explain this. Assume that for an 8-way cache, the number of hits in different LRU-positions is $\{10816,4645,2140,501,217,113,63,11\}$, where the first value shows the hits in MRU (most-recently used) position while the last value shows the hits in the LRU position. Here, we have $H = 18506$. If $\alpha = 0.97$, then we get $X = 4$ since at least 4 ways need to be turned-on to achieve equal to or greater than αH hits. However, if $\alpha = 0.95$, then $X = 3$ and hence, only 3 ways need to be turned-on.

We also use a parameter A_{min}, which shows the minimum number of ways which are always turned-on. The typical values of A_{min} used in this paper are 2, 3 and 4. We do not take $A_{min} = 1$, since keeping only a single-way turned-on makes the LLC a direct-mapped cache, which leads to large performance loss due to increased off-chip accesses.

3.2 Cache Profiling

For making dynamic cache reconfiguration decisions, ESTEEM collects data using dynamic profiling approach. For this purpose, it uses auxiliary tag directory (ATD) [37]. The ATD has the same associativity and replacement policy as the main tag directory (MTD) and uses set-sampling ap-

Algorithm 1 ESTEEM Energy Saving Algorithm

```
 1: INPUT: nL2Hit[0:$M$-1][0:$A$-1] showing the number of hits to different LRU-positions for different modules in last interval;
    $A_{min}$ showing the minimum number of ways to be always turned-on; $\alpha$ showing the hit threshold.
 2: OUTPUT: nActiveWay[0:$M$-1] showing algorithm decision about number of ways to keep turned-on in different modules
 3: Let Accumulated_L2Hit[0:$M$-1][0:$A$-1] be a variable
 4: for $m$=0 to $M$-1 do                                        ▷ First see, if the module is non-LRU
 5:     Let nLRUAnomaly ←0
 6:     for $i$=0 to $A$-2 do
 7:         if nL2Hit[m][i]<nL2Hit[m][i+1] then
 8:             nLRUAnomaly ← nLRUAnomaly+1
 9:         end if
10:     end for
11:     if nLRUAnomaly $\geq A/4$ then
12:         isModuleNonLRU ← TRUE
13:     end if
                                                                 ▷ Now decide the number of ways to turn-off
14:     for $i$=0 to $A$-1 do
15:         Accumulated_L2Hit[m][i] ← $\sum_{j=0}^{i}$ nL2Hit[m][j]
16:     end for
17:     totHitsThisModule ← Accumulated_L2Hit[m][$A$-1]
18:     for $i$=0 to $A$-1 do
19:         if Accumulated_L2Hit[m][i] $\geq$ ($\alpha\times$ totHitsThisModule) then
20:             nActiveWay[m] ← MAX($A_{min}$, i+1)
21:             if isModuleNonLRU then
22:                 nActiveWay[m] ← MAX($A$-1, i+1)
23:             end if
24:             Break from for loop
25:         end if
26:     end for
27: end for
    return nActiveWay[0:$M$-1]
```

proach [35] to keep its overhead small. The ratio of the number of sets in the L2 cache and that in ATD is shown as R_s and its typical values are 32, 64, 128 etc. As an example, for R_s =64, ATD monitors only 1/64 of the sets. We use an ATD, which is embedded in the MTD of the L2 cache. The sets which are monitored are called the leader sets and the remaining sets are called the follower sets. The leader sets do not undergo cache reconfiguration. Profiling information is only collected from the leader sets. The follower sets undergo cache reconfiguration based on the decision of the algorithm. Since the cache-sets are divided into multiple modules, statistics obtained from a leader set count towards the module in which this leader set falls.

4. ENERGY SAVING ALGORITHM

Algorithm 1 shows the energy saving algorithm in ESTEEM, which can be a kernel module. The algorithm runs after every few million cycles (e.g. 10M cycles). The algorithm can be understood as follows. The algorithm first checks whether a module is non-LRU, which is checked by comparing the number of hits at different LRU positions. For each LRU position, the algorithm computes the accumulated hits till that LRU position. The number of ways to keep active can thus be decided by the LRU position, at which total hits are α fraction (e.g. 95%) of the total hits. For a non-LRU module, at most 1 way is turned-off for the reasons explained the previous section. This process is repeated for each of the module.

The overhead of the algorithm is very small. Periodically, a kernel routine is triggered which executes algorithm. The output of the algorithm is the decision about the number of ways to be turned-off in each module, which can be easily implemented using per-way disable bits. Thus, the changes required in the cache and the hardware overhead are very small. The algorithm reads few counters and runs infrequently, so its power overhead is amortized over the length of the phase. The power saving provided by the algorithm easily allow adding small additional hardware within the power-budget. Many processors already use counters for OS or measuring performance [22], which can be leveraged by the algorithm. Modern processors already use write-back buffers, specific instructions, and MSHRs (miss-status holding registers), which handle writing-back of flushed data.

5. IMPLEMENTATION AND OVERHEAD ASSESSMENT

We assume that power-gating of eDRAM is achieved by a suitable circuit-level technique, as proposed by several authors [10, 12, 16, 32]. For each module, we use $A - A_{min}$ control bits which control turning-off or turning-on of the ways in that module.

Cache reconfigurations are handled as follows. When the number of ways is reduced, the clean cache lines in those ways are discarded and the dirty lines are written-back. When the number of ways is increased, the extra ways are simply turned-on and they are subsequently used for storing data. With ESTEEM technique, cache reconfigurations happen only at the end of a large interval and not throughout the execution of the application. Thus, cache line-switching does not lie on the critical access path of the cache. Also, unlike selective-sets approach used in previous works [34], the selective-ways approach used in ESTEEM does not require changing the set-decoding of the cache and hence, ESTEEM does not increase the cache access time. Further, ESTEEM provides fine-grained cache reconfiguration with caches of typical associativity and thus, does not require use of caches of large associativity which have significantly large access time and energy. Also note that ESTEEM does not require tables for offline profiling (as in [44]) or using per-block counters to monitor cache access intensity (as in [22]). Also, it does not require prediction of cache or memory energy (as in [31]) or hits/misses for different cache configurations (as in [30]).

ESTEEM uses counters for recording the number of hits to different LRU positions and execution of algorithm. For nL2Hit and Accumulated_L2Hit, total storage required is $2 \times M \times A$ counters (assuming $A_{min} = 0$ for simplicity) and for nActiveWay, the storage required is M counters. Assuming that each counter takes 40 bits, the total storage overhead of ESTEEM, as a percentage of L2 storage, can be expressed as

$$Overhead = \frac{(2A+1)M \times 40}{SA(B+G)} \times 100 \qquad (1)$$

For a 4MB cache with 16 modules and 16-way set-associativity, the overhead of ESTEEM is found to be 0.06% of the L2 cache size, which is extremely small. For this reason, we ignore the overhead of counters.

6. EXPERIMENTAL METHODOLOGY

6.1 Simulation Platform and Workload

We perform microarchitectural simulation using Sniper x86-64 simulator [9]. The processor has 2GHz frequency. All caches use a line size of 64B. Both L1D and L1I are 32KB, 4-way, LRU caches and have a latency of 2 cycles. The L2 cache is a 16-way, LRU cache with 12 cycle latency. Its size for single and dual-core system is 4MB and 8MB, respectively. L1 caches are private to each core and L2 cache is shared among cores. The latency of main memory is 220 cycles and memory queue contention is also modeled. The main memory bandwidth for single and dual-core system is 10 GB/s and 15GB/s, respectively.

All eDRAM L2 caches have a 4-bank structure. We assume that each bank of L2 cache has dedicated logic to process refresh requests and using pipelining, a line can be refreshed in a single cycle [4]. For eDRAM cells, Barth et al. [8] report a retention period of 40μs at 105°C. In this paper, we assume working temperature of 60°C and since retention periods are exponentially dependent on temperature [4], we present most of the results with a retention period of 50μ. In Section 7.3, we also present results for a retention period of 40μs.

We use all 29 SPEC2006 benchmarks [17] with *ref* inputs and 5 benchmarks from HPC field (shown as italics in Table 1) [1, 2]. Using these benchmarks, we randomly make 17 dual-core multiprogrammed workloads, such that each benchmark is used only once. These workloads are shown in Table 1.

Table 1: Workloads Used in the Paper

Single-core workloads and their acronyms
As(astar), Bw(bwaves), Bz(bzip2), Cd(cactusADM)
Ca(calculix), Dl(dealII), Ga(gamess), Gc(gcc)
Gm(gemsFDTD), Gk(gobmk), Gr(gromacs), H2(h264ref)
Hm(hmmer), Lb(lbm), Ls(leslie3d), Lq(libquantum)
Mc(mcf), Mi(milc), Nd(namd), Om(omnetpp)
Pe(perlbench), Po(povray), Sj(sjeng), So(soplex)
Sp(sphinx), To(tonto), Wr(wrf), Xa(xalancbmk)
Ze(zeusmp), *Am(amg2013), Co(comd), Lu(lulesh)*
Ne(nekbone), Xb(xsbench)
Dual-core workloads and their acronyms
GmDl(gemsFDTD-dealII), AsXb(astar-xsbench)
GcGa(gcc-gamess), BzXa(bzip2-xalancbmk)
LsLb(leslie3d-lbm), GkNe(gobmk-nekbone)
OmGr(omnetpp-gromacs), NdCd(namd-cactusADM)
CaTo(calculix-tonto), SpBw(sphinx-bwaves)
LqPo(libquantum-povray), SjWr(sjeng-wrf)
PeZe(perlbench-zeusmp), HmH2(hmmer-h264ref)
SoMi(soplex-milc), McLu(mcf-lulesh)
CoAm(comd-amg2013)

6.2 Comparison With Other Technique

We compare ESTEEM with Refrint polyphase-valid (RPV) policy [4]. RPV works on the idea that on a read or a write, an eDRAM cache block is automatically refreshed and hence, it need not be refreshed for the duration of one retention period. RPV divides the retention period into a number of phases. Each cache block maintains the information about the phase in which it was last updated. Afterwards, to reduce the number of refresh operations, RPV refreshes the block at the beginning of *this phase* in the next retention period, instead of refreshing at beginning of refresh period itself. Also, RPV only refreshes the valid blocks. We use RPV with four phases, since this has been shown to provide significant energy savings [4].

Agrawal et al. [4] also propose Refrint polyphase-dirty (RPD) policy which eagerly invalidates valid blocks to avoid refreshing them and refreshes only dirty blocks. For applications where the fraction of dirty data is small, RPD policy would aggressively invalidate almost the whole cache which will greatly increase the access to main memory and hence, we do not evaluate this. Further, RPV policy has been shown to perform better than another policy proposed by Agrawal et al., namely the periodic-valid refresh policy [4] and hence, we do not evaluate periodic-valid refresh policy.

6.3 Energy Model

We account for the energy consumption of L2 cache (E_{L2}), main memory (E_{MM}) and energy cost of algorithm (E_{Algo}), since the techniques evaluated here affect the other components only minimally. We use the following notations. LE_{L2}, DE_{L2} and RE_{L2} show the leakage, dynamic and refresh energy consumed in L2 cache, respectively. E^{dyn}_{xyz} and P^{leak}_{xyz} show the dynamic energy per access and leakage energy per second, respectively, in a component xyz (e.g. L2 or MM). For ESTEEM, N_L shows the number of cache blocks which are turned on or off (i.e. undergo state transition); E_χ shows the energy consumed in a single such block transition. F_A, H_{L2} and M_{L2} show the active fraction of cache, number of L2 hits and L2 misses in an interval, respectively. N_R shows the number of cache lines which are refreshed within all refresh-events in an interval. T denotes the time length of an interval (or any time period of measurement) in seconds. A_{MM} shows the number of main memory accesses.

We assume that the L2 leakage energy scales with the active fraction of cache [22,31]. Also, we assume that an L2 miss consumes twice the dynamic energy as that of an L2 hit [29–31]. Thus, we have

$$E = E_{L2} + E_{MM} + E_{Algo} \tag{2}$$
$$E_{L2} = LE_{L2} + DE_{L2} + RE_{L2} \tag{3}$$
$$LE_{L2} = P^{leak}_{L2} \times F_A \times T \tag{4}$$
$$DE_{L2} = E^{dyn}_{L2} \times (2M_{L2} + H_{L2}) \tag{5}$$
$$RE_{L2} = N_R \times E^{dyn}_{L2} \tag{6}$$
$$E_{MM} = P^{leak}_{MM} \times T + E^{dyn}_{MM} \times A_{MM} \tag{7}$$
$$E_{Algo} = E_{\chi} \times N_L \tag{8}$$

We ignore the energy overhead of RPV algorithm, thus, for experiments with baseline eDRAM cache and RPV, we have $E_{Algo} = 0$ and $F_A = 1$. Note that F_A for ESTEEM duly takes into account the active area due to leader and follower sets (see Section 3.2).

We use CACTI [27] to obtain the values of E^{Dyn}_{L2} and P^{Leak}_{L2} at 32nm for eDRAM cache. These values are shown in Table 2. Following [4], we assume that for eDRAM cache, the time and energy consumed in refreshing a line is equal to the time and energy to access the line, respectively.

Table 2: Energy values for 16-way eDRAM cache

	E^{dyn}_{L2} (nJ/access)	P^{leak}_{L2} (Watts)
2 MB	0.186	0.096
4 MB	0.212	0.116
8 MB	0.282	0.280
16 MB	0.370	0.456
32 MB	0.467	1.056

E^{dyn}_{MM} and P^{leak}_{MM} are taken as 70 nJ and 0.18 Watt, respectively [23, 29, 46] and E_{χ} is taken as 2 pJ [29, 30].

6.4 Evaluation Metrics

Our baseline is as an eDRAM cache which periodically refreshes all the cache lines at the given retention period and does not use any refresh-minimization technique. We show the results on the following metrics.

1. Percentage energy saving (as defined above)

2. Weighted speedup (WS) [24, 29, 31], referred to as relative performance. It is defined as

$$WS = \frac{\sum_{n=0}^{N-1} \frac{\text{IPC}_n(\text{technique})}{\text{IPC}_n(\text{base})}}{N} \tag{9}$$

   Here technique refers to either ESTEEM or RPV.

3. Absolute reduction in number of cache lines refreshed per kilo instructions (RPKI).

For ESTEEM technique, we also show the results on the following metrics:

1. Absolute increase in MPKI (miss-per kilo instructions) [30] due to use of ESTEEM

2. Active ratio (the fraction of active lines averaged over entire execution [22, 29])

The decrease in RPKI helps us to evaluate the efficacy of a technique in reducing refresh operations. Active ratio enables us to evaluate the aggressiveness of cache turn-off of ESTEEM and the increase in MPKI helps in evaluating the increase in off-chip traffic. Note that since RPV does not turn-off the cache or cause early invalidation, its ActiveRatio is always 100% and the increase in MPKI is always zero.

The benchmarks were fast-forwarded for 10B instructions. Then, each workload is simulated for 400M instructions. For dual-core system, the benchmark which finished its 400M instructions early was allowed to run, however, its IPC (for computing speedup) was only recorded for the first 400M instructions, following well-established simulation methodology [14,29]. For dual-core system, we have also computed the value of fair speedup [29,31] and found its average value to be close to the weighted speedup. Thus, our technique does not cause unfairness. For the sake of brevity, we omit these results. Across the workloads, speedup (weighted and fair) values are averaged using geometric mean and the remaining metrics are averaged using arithmetic mean, since they can be zero or negative.

7. RESULTS AND ANALYSIS

We now present the results. In Sections 7.1, 7.2 and 7.3, we use the following parameters. The size of L2 cache for single and dual-core systems are 4 and 8 MB, respectively. For ESTEEM, we use the following parameters: $\alpha = 0.97$, $A_{min} = 3$, R_s =64, an interval length of 10M cycles and 8 modules for single-core system and 16 modules for dual-core system.

7.1 Example of working of ESTEEM

To get insights into working of ESTEEM, we first show the cache reconfiguration taking place with ESTEEM for a selected workload (namely h264ref) in Figure 2. Notice

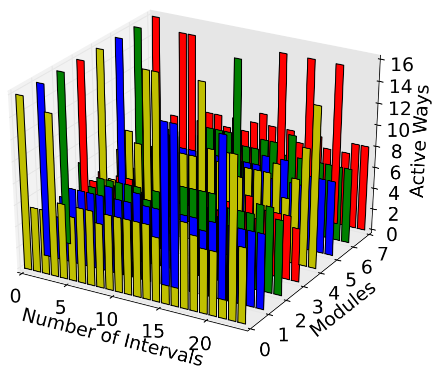

Figure 2: Example of working of ESTEEM for h264ref benchmark. Note that, in any interval, the number of active ways in different modules may be different.

that depending on the cache requirement, the cache active

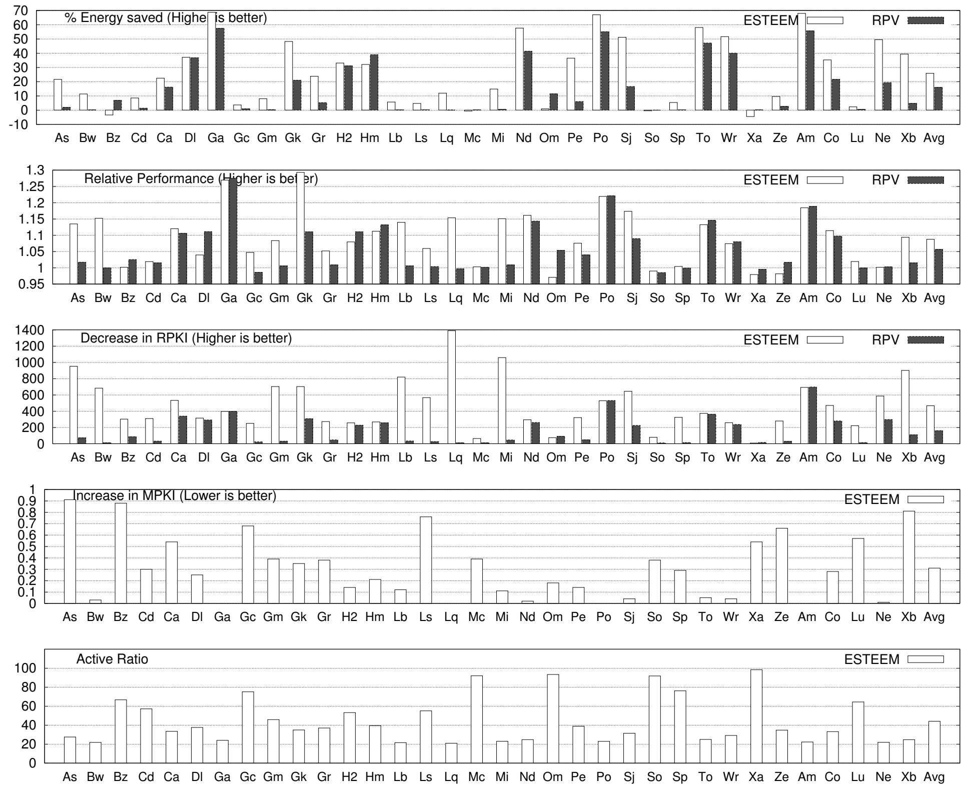

Figure 3: Single-core results for different techniques at 50 μs refresh period.

ratio changes with time (i.e. over different intervals). More importantly, the number of active ways in different modules can possibly be different. This clearly shows the advantage of ESTEEM in exercising fine-grained cache reconfiguration.

7.2 Results with 50μs Retention Period

Figures 3 and 4 show the results for single and dual-core systems, respectively. We now discuss and analyze the results.

For single and dual-core systems, the average saving in memory sub-system energy on using ESTEEM (resp. RPV) are 25.82% (resp. 15.93%) and 32.63% (resp. 14.3%), respectively. Also, the average relative performance on using ESTEEM (resp. RPV) are 1.09$\times$ (resp. 1.06$\times$) and 1.22$\times$ (resp. 1.09$\times$), respectively. With ESTEEM, for single and dual-core systems, the largest amount of energy saving is seen for gamess (68.7%) and GkNe (77.2%), respectively. The largest improvement in performance is seem for Gk (1.29$\times$) and GkNe (1.48$\times$). The performance is improved despite cache-turnoff due to reduction in refresh operations which offsets the penalty of increased cache misses.

The actual performance/power improvement achieved with ESTEEM depends on interaction of several factors. Turning off large fraction of cache saves large amount of leakage energy and avoids the need of large number of refresh operations. Reduction in refresh operations leads to performance improvement (i.e. reduction in execution time), which further reduces the refresh operations and the refresh and leakage energy consumed in the cache. However, the extra misses and writebacks introduced due to reduced cache size increase the energy consumed in main memory.

RPV saves energy when only a small portion of the cache stores valid data, since in such cases, a large number of refresh operations can be avoided. For some applications, nearly the whole cache stores valid data and hence, the scope for avoiding refresh operations is minimal. Moreover, it has been shown that [22] cache lines typically have a flurry of frequent use when first brought into the cache, and then see a period of "dead time" before they are evicted. Thus, for any cache line, after the last access and before its eviction, RPV does not avoid or minimize refresh operations to the cache line. For these reasons, the effectiveness of RPV in minimizing refresh operations is limited. In contrast, ESTEEM intelligently reduces the active fraction of cache, which intrinsically minimizes the number of cache lines which need to be refreshed.

In general, for dual-core system, the intensity of cache access is higher than that in single-core system. For this reason, the fraction of invalid cache blocks is reduced. Hence, RPV saves smaller percentage of energy in the dual-core system. This is also evident from the smaller value of decrease in RPKI in the case of dual-core system for RPKI. For single-core system, the reduction in RPKI on using ESTEEM and RPV are 467 and 161, respectively, and for dual-core system, these values are 511 and 134, respectively. Large reduction in RPKI also reflects in improved performance, since the performance overhead of refresh operations is avoided. Also note that compared to RPV, ESTEEM achieves nearly 4$\times$ reduction in RPKI.

Since ESTEEM turns-off a fraction of cache, it also has the advantage of saving leakage energy. With single and dual-core systems, the average active-ratio achieved on us-

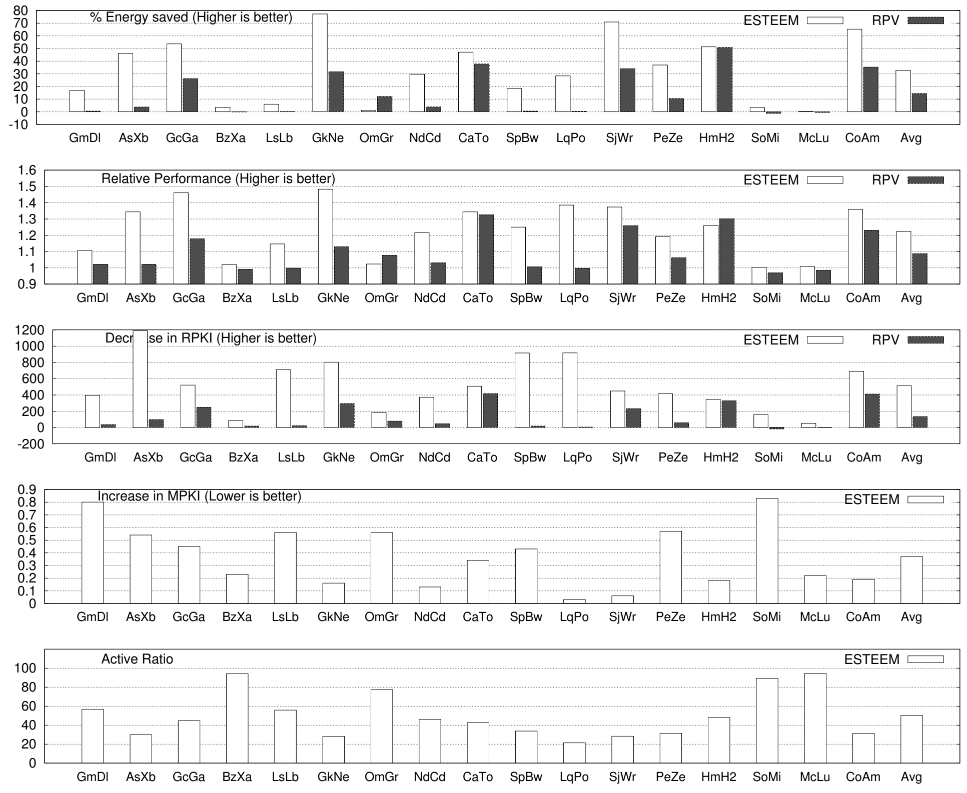

Figure 4: Dual-core results for different techniques at 50μs refresh period.

ing ESTEEM are 44.1% and 50.2%, respectively. For some applications, such as libquantum and milc, the data reuse is very small (i.e. miss-rate is nearly 100%) and thus, the working set size of the application is much smaller compared to the cache size. In such cases, ESTEEM aggressively reduces the cache active fraction and the number of refresh operations and provides large saving in energy.

On using ESTEEM, the average increase in MPKI for single and dual-core system are 0.31 and 0.37, respectively. Thus, the increase in off-chip traffic on using ESTEEM is very small. This is because, by virtue of using dynamic cache reconfiguration, ESTEEM closely adjusts the size of cache in response to intra- and inter-application variation in cache requirement of different applications. Also, it always keeps 3 ways active and keeps as many ways active as required for achieving at least 97% cache hits (assuming the parameters shown above are chosen).

For some single-core workloads, a small loss in performance/energy is seen on using ESTEEM. This is due to either the non-LRU behavior (e.g. omnetpp and xalancbmk) or large application working-set size (e.g. mcf and soplex). Also, in general, the reconfiguration overhead resulting from increased off-chip traffic slightly offsets the energy saving achieved in cache. These overheads can be minimized by increasing the value of α, A_{min} and interval length. We study the effect of these parameters in Section 7.4. The reconfiguration overhead can also be minimized by restricting the maximum number of change in associativity in each interval or detecting and avoiding frequent reconfigurations. This extension of the energy saving algorithm is planned as a future work.

7.3 Results with 40μs Retention Period

Since an increase in temperature and process variations reduce the retention period, we now test ESTEEM and RPV with reduced 40 μs retention period. The results are shown in Figures 5 and 6. In this section, we only comment on the effect of the retention period, compared to the case with 50 μs retention period, since the results can be easily understood based on the above explanation.

With reduced retention period, a large fraction of energy is consumed in the form of refresh energy for the baseline cache. Also, the energy consumption of L2 is increased and hence, it becomes a larger fraction of memory subsystem energy. Furthermore, with reduced retention period, the same number of blocks need to be refreshed within smaller amount of time. These refresh operations also make the cache unavailable, leading to performance loss. Hence, at lower retention period, the scope of and benefits from reducing refresh operations are further increased. This is also reflected in the reduction in RPKI value for both ESTEEM and RPV. For the same reason, both ESTEEM and RPV show larger improvement in performance and energy efficiency. With ESTEEM, the largest improvement in energy saving is seen in gamess (73.6%) and GkNe (83.2%). Also, the largest improvement in performance is seen in gobmk (1.40×) and GcGa (1.72×).

It is clear that a reduction of merely 10μs in retention period can increase refresh energy significantly. Thus, for smaller retention periods, the need of a technique for reducing refresh operations is increased even further. This highlights the importance of our technique.

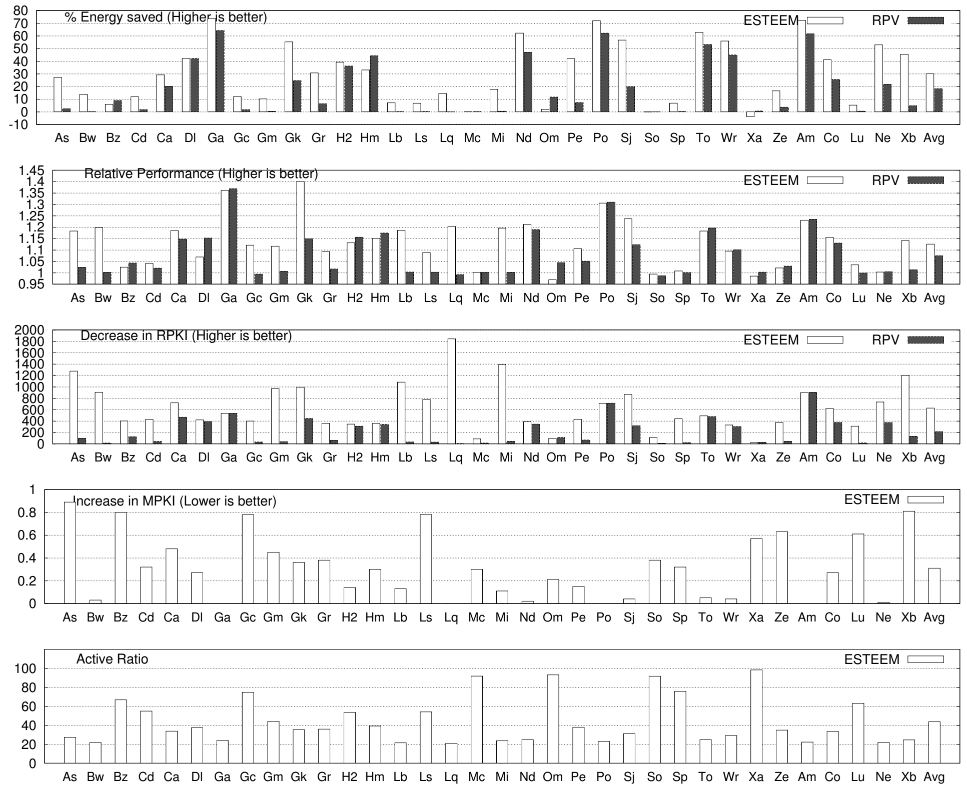

Figure 5: Single-core results for different techniques at 40μs refresh period.

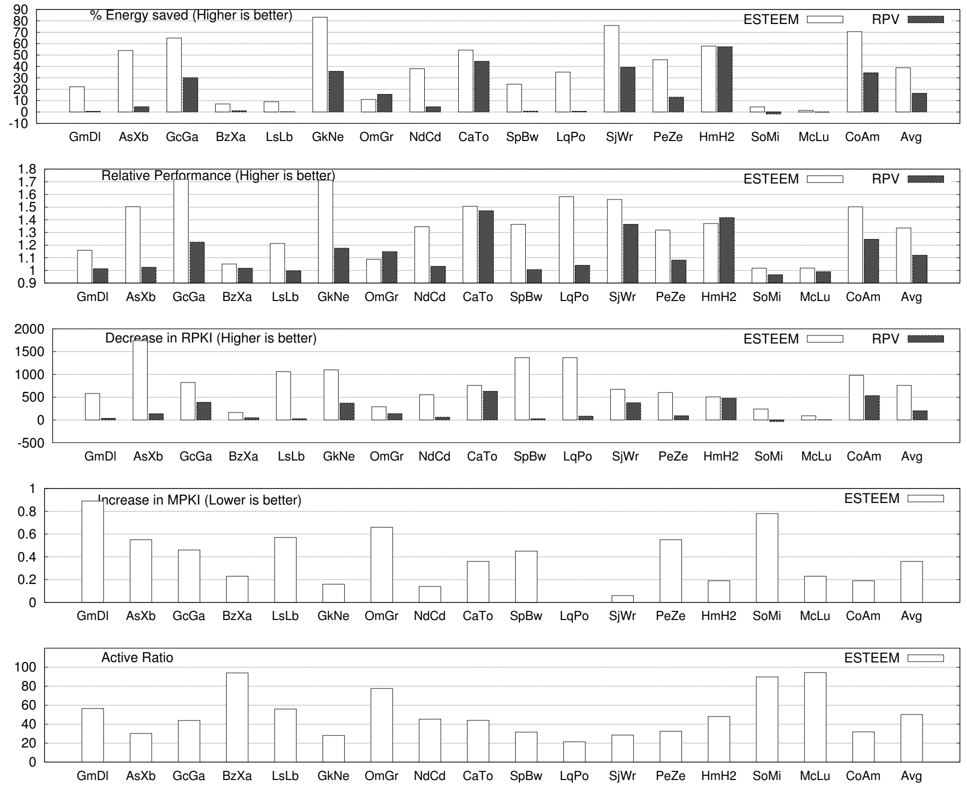

Figure 6: Dual-core results for different techniques at 40μs refresh period.

Table 3: Parameter sensitivity results for ESTEEM (Rel. Perf. = relative performance, Dec. = decrease, Inc. = increase). Default parameters are shown in the beginning of Section 7.

	% Energy Saving	Rel. Perf.	RPKI Dec.	MPKI Inc.	Active Ratio
Single-core System					
Default	25.82	1.09	467.4	0.31	44.10
A_{min}=2	25.46	1.08	482.4	0.36	41.60
A_{min}=4	25.76	1.09	449.1	0.26	47.00
α=0.95	24.95	1.08	473.9	0.37	42.70
α=0.99	26.56	1.09	458.2	0.24	46.10
2 modules	24.52	1.08	458.5	0.34	44.93
4 modules	25.96	1.09	457.7	0.27	45.20
16 modules	24.87	1.09	478.2	0.37	42.40
32 modules	19.41	1.06	491.0	0.62	38.97
5M interval	24.07	1.09	491.4	0.43	40.40
15M interval	25.82	1.09	456.5	0.27	46.00
R_s =32	25.79	1.09	458.9	0.28	45.80
R_s =128	24.30	1.08	477.7	0.38	42.20
8-way L2	23.68	1.08	397.9	0.20	55.94
32-way L2	24.39	1.08	499.3	0.49	38.27
2MB L2	10.18	1.02	204.4	0.38	48.00
8MB L2	49.42	1.29	1257.3	0.37	41.70
Two-core System					
Default	32.63	1.22	511.9	0.37	50.20
A_{min}=2	32.04	1.22	525.0	0.47	48.50
A_{min}=4	32.44	1.22	495.1	0.31	52.40
α=0.95	32.01	1.23	524.5	0.43	48.10
α=0.99	32.90	1.22	490.9	0.29	53.50
4 modules	31.22	1.19	482.9	0.35	51.40
8 modules	32.15	1.21	497.1	0.35	51.30
32 modules	32.13	1.23	526.1	0.42	47.90
64 modules	28.75	1.21	546.2	0.59	43.69
5M interval	32.41	1.23	543.4	0.49	46.60
15M interval	32.16	1.21	493.5	0.33	52.30
R_s =32	32.69	1.22	500.5	0.35	51.90
R_s =128	32.13	1.23	526.2	0.43	47.90
8-way L2	30.00	1.19	424.7	0.25	60.73
32-way L2	31.91	1.23	541.8	0.56	45.70
4MB L2	8.04	1.06	181.9	0.45	55.70
16MB L2	66.25	2.11	2438.0	0.68	43.70

7.4 Parameter Sensitivity Results

We now focus exclusively on ESTEEM and study its sensitivity for different parameters. The retention period is fixed to 50μs. Each time, we only change one parameter from the default parameters used above. The results are summarized in Table 3. For comparison purposes, the results with default parameters are also shown.

Change in A_{min}: On changing A_{min} (the minimum number of ways which are always kept on) from 3 to 2, the algorithm turns off a larger fraction of cache and hence, the active ratio is reduced and MPKI is increased further. This also reduces the refresh operations further. However, due to increased off-chip accesses, a small reduction in overall saving is achieved. The results on increasing A_{min} to 4 can be similarly understood.

Change in α: On changing α from 0.97 to 0.95, the aggressiveness of cache reconfiguration is increased, as reflected in results on active ratio, MPKI increase and RPKI decrease. However, it also increases the main memory energy and hence, the overall energy saving is slightly reduced. The opposite is seen on changing α to 0.99.

Change in number of modules (M): We experiment with both smaller and higher number of modules. On reducing the number of modules, both granularity and aggressiveness of reconfiguration is reduced, as reflected from values of MPKI and active ratio. This also reduces the decrease in RPKI and off-chip accesses. The net energy saving and performance depends on the interaction of these parameters. Conversely, on increasing the number of modules, cache can be reconfigured in more fine-grain manner. Thus, the active ratio and RPKI are reduced further, although the off-chip accesses are increased. Using very large number of modules leads to reduced energy savings due to increased overhead of cache reconfiguration.

Change in interval size: On changing the interval size from 10M cycles to 5M cycles, reconfiguration algorithm is more frequently executed. Thus, the active ratio is reduced further, with corresponding reduction in RPKI. However, due to frequent reconfiguration, the overhead of reconfiguration is also increased, as evident from the value of increase in MPKI, which also increases the main memory energy. Hence, the overall energy saving is slightly reduced. The opposite phenomenon is seen on increasing the interval size to 15M cycles and the overall energy saving achieved depends on the interaction of the above mentioned factors.

Change in sampling ratio (R_s): On changing the sampling ratio from 64 to 32, the energy and performance improvement are enhanced. Also, even with the sampling ratio of 128, ESTEEM achieves large improvement in performance and energy efficiency. Thus, a designer can choose a suitable value of sampling ratio to achieve a balance between profiling overhead and the energy saving achieved.

Change in associativity (A): We experiment with both 8-way and 32-way set-associative cache. For 8-way cache, using A_{min} = 3 leads to keeping at least 3 out of 8 ways always on. This leads to higher active ratio and smaller increase in MPKI. However, due to this, the aggressiveness of ESTEEM in reducing the refresh operations is also reduced, which leads to smaller energy saving. Conversely, for 32-way cache, using A_{min} = 3 leads to keeping only at most 3-ways always on and thus, the active ratio is reduced, although the increase in MPKI is also enhanced. Due to larger fraction of turned-off cache and larger reduction in refresh operations, the energy saving is increased.

Change in cache size: We experiment with both double and half size cache, compared to the cache size used in Section 7.2. On increasing the cache size, the scope for saving both leakage and refresh energy is increased. This is because the applications have fixed working set size and with larger cache size, a large fraction of energy is wasted due to unnecessary refresh operations. Also, for larger cache size, the contribution of L2 cache in memory subsystem energy is also larger. Further, with larger cache size, more cache blocks need to be refreshed in the same amount of time, which significantly degrades the performance. This clearly shows that use of a technique for minimizing refresh energy is inevitable for large-sized last-level caches. From the results, we conclude that ESTEEM provides large energy saving and performance improvement with large cache size, in fact, with double cache size, for dual-core system, 66.25% energy is saved. The opposite trend is seen on reducing the cache size.

The results shown in this section confirm that ESTEEM is effective in saving energy in eDRAM caches and also improves performance. Also, by adjusting α, A_{min} and the interval size, a designer can achieve fine balance between the performance gain and energy saving.

8. CONCLUSION

Embedded DRAM caches present as a promising alternative to SRAM due to their low leakage value, however, addressing their refresh overhead is crucial to enabling their use in designing on-chip caches. In this paper, we presented ESTEEM, a technique for saving both leakage and refresh energy in eDRAM caches. The experimental results have shown that ESTEEM provides significant energy savings, while improving performance. Also, it outperforms a recently-proposed technique for mitigating refresh overhead in eDRAM caches.

Acknowledgements

Support for this work was provided by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 to the U.S. Government. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

9. REFERENCES

[1] `http://oxbow.ornl.gov/apps.html`.

[2] `https://github.com/jtramm/XSBench`.

[3] Top 500 Supercomputers. `www.top500.org`, 2013.

[4] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas. Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies. *HPCA*, 2013.

[5] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *MICRO*, pages 248–259, 1999.

[6] C. S. Bae, L. Xia, P. Dinda, and J. Lange. Dynamic adaptive virtual core mapping to improve power, energy, and performance in multi-socket multicores. *HPDC*, pages 247–258, 2012.

[7] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, C. Prete, and P. Stenström. Leveraging data promotion for low power D-NUCA caches. In *EUROMICRO Conference on Digital System Design (DSD)*, pages 307–316, 2008.

[8] J. Barth et al. A 500 MHz random cycle, 1.5 ns latency, SOI embedded DRAM macro featuring a three-transistor micro sense amplifier. *IEEE Journal of Solid-State Circuits*, 43(1):86–95, 2008.

[9] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *SC*, 2011.

[10] M.-T. Chang, P.-T. Huang, and W. Hwang. A 65nm low power 2T1D embedded DRAM with leakage current reduction. In *IEEE International SOC Conference*, pages 207–210, 2007.

[11] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob. Technology Comparison for Large Last-Level Caches (L^3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM. *HPCA*, 2013.

[12] K. C. Chun, P. Jain, and C. H. Kim. Logic-compatible embedded DRAM design for memory intensive low power systems. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 277–280, 2010.

[13] J. Dongarra et al. The International Exascale Software Project roadmap. *IJHPCA*, 25(1):3–60, 2011.

[14] J. Gaur, M. Chaudhuri, and S. Subramoney. Bypass and insertion algorithms for exclusive last-level caches. *ACM SIGARCH Computer Architecture News*, 39(3):81–92, 2011.

[15] M. Ghosh and H.-H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D Die-Stacked DRAMs. In *40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 134–145, 2007.

[16] K. Hardee et al. A 0.6 V 205MHz 19.5 ns tRC 16Mb embedded DRAM. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 200–522, 2004.

[17] J. L. Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[18] Intel. `http://ark.intel.com/products/53575/`.

[19] S. Iyer, J. Barth Jr, P. Parries, J. Norum, J. Rice, L. Logan, and D. Hoyniak. Embedded DRAM: Technology platform for the Blue Gene/L chip. *IBM Journal of Research and Development*, 49(2.3):333–350, 2005.

[20] X. Jiang et al. ACCESS: Smart scheduling for asymmetric cache CMPs. In *17th International Symposium on High Performance Computer Architecture (HPCA)*, pages 527–538, 2011.

[21] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. Power7: IBM's next-generation server processor. *IEEE Micro*, 30(2):7–15, 2010.

[22] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *ISCA*, pages 240–251, 2001.

[23] S. K. Khaitan and J. D. McCalley. A hardware-based approach for saving cache energy in multicore simulation of power systems. In *IEEE Power and Energy Society General Meeting (PES)*, pages 1–5, 2013.

[24] S. K. Khaitan and J. D. McCalley. Optimizing cache energy efficiency in multicore power system simulations. *Energy Systems*, pages 1–15, 2013.

[25] N. Kurd et al. Haswell: A family of IA 22nm processors. In *IEEE ISSCC*, pages 112–113, 2014.

[26] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *International Conference on Computer-Aided Design (ICCAD)*, pages 694–701, 2011.

[27] CACTI 5.3. `http://quid.hpl.hp.com:9081/cacti/`, 2013.

[28] S. Mittal. A survey of architectural techniques for improving cache power efficiency. *Sustainable Computing: Informatics and Systems*, 2013.

[29] S. Mittal, Y. Cao, and Z. Zhang. MASTER: A Multicore Cache Energy Saving Technique using Dynamic Cache Reconfiguration. *IEEE Transactions on VLSI*, 2013.

[30] S. Mittal and Z. Zhang. EnCache: Improving Cache Energy Efficiency Using A Software-Controlled Profiling Cache. In *IEEE International Conference On Electro/Information Technology*, USA, May 2012.

[31] S. Mittal, Z. Zhang, and J. Vetter. FlexiWay: A Cache Energy Saving Technique Using Fine-grained Cache Reconfiguration. In *IEEE International Conference on Computer Design (ICCD)*, pages 100–107, 2013.

[32] F. Morishita et al. A 312-MHz 16-Mb random-cycle embedded DRAM macro with a power-down data retention mode for mobile applications. *IEEE Journal of Solid-State Circuits*, 40(1):204–212, 2005.

[33] C. D. Patel, C. E. Bash, R. Sharma, M. Beitelmal, and R. Friedrich. Smart cooling of data centers. *Pacific RIM/ASME International Electronics Packaging Technical Conference and Exhibition (IPACK03)*, 2003.

[34] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar. Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In *international symposium on Low power electronics and design (ISLPED)*, pages 90 – 95, 2000.

[35] T. Puzak. *Cache Memory Design*. PhD thesis, University of Massachusetts, 1985.

[36] M. K. Qureshi, S. Gurumurthi, and B. Rajendran. Phase change memory: From devices to systems. *Synthesis Lectures on Computer Architecture*, 6(4):1–134, 2011.

[37] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO*, pages 423–432, 2006.

[38] W. R. Reohr. Memories: Exploiting them and developing them. In *IEEE International SOC Conference*, pages 303–310, 2006.

[39] P. Reviriego, A. Sánchez-Macian, and J. A. Maestro. Low Power embedded DRAM caches using BCH code partitioning. In *IEEE International On-Line Testing Symposium (IOLTS)*, pages 79–83, 2012.

[40] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 371–382, 2009.

[41] D. Rolán, B. B. Fraguela, and R. Doallo. Adaptive line placement with the set balancing cache. In *MICRO*, pages 529–540, 2009.

[42] A. Samih, A. Krishna, and Y. Solihin. Understanding the limits of capacity sharing in CMP Private Caches. In *HPCA*, 2009.

[43] A. Valero et al. An hybrid eDRAM/SRAM macrocell to implement first-level data caches. In *MICRO*, pages 213–221, 2009.

[44] W. Wang, P. Mishra, and S. Ranka. Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. In *DAC*, pages 948–953, 2011.

[45] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. *ACM SIGARCH Computer Architecture News*, 38(3):83–93, 2010.

[46] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled DIMM: building high-bandwidth memory system using low-speed DRAM devices. In *ISCA*, pages 255–266, 2009.