EECache: Exploiting Design Choices in Energy-Efficient Last-Level Caches for Chip Multiprocessors

Hsiang-Yun Cheng, Matt Poremba, Narges Shahidi, Ivan Stalev,
Mary Jane Irwin, Mahmut Kandemir, Jack Sampson, Yuan Xie

Department of Computer Science and Engineering, Pennsylvania State University
University Park, Pennsylvania, USA 16802

hoc5108@cse.psu.edu, mrp5060@psu.edu, nxs314@cse.psu.edu, ids103@psu.edu,
{mji, kandemir, sampson, yuanxie}@cse.psu.edu

ABSTRACT

Power management for large last-level caches (LLCs) is important in chip-multiprocessors (CMPs), as the leakage power of LLCs accounts for a significant fraction of the limited on-chip power budget. Since not all workloads need the entire cache, portions of a shared LLC can be disabled to save energy. In this paper, we explore different design choices, from circuit-level cache organization to micro-architectural management policies, to propose a low-overhead run-time mechanism for energy reduction in the shared LLC. Results show that our design (EECache) provides 14.1% energy saving at only 1.2% performance degradation on average, with negligible hardware overhead.

Categories and Subject Descriptors: B.3.2 [Memory Structures]: Design Styles - *Cache memories*

Keywords: Cache; Power management; Energy-efficiency

1. INTRODUCTION

The power consumption of modern chip-multiprocessors has become a primary design constraint. Even though Moore's law continues to provide increasing transistor counts, the limited on-chip power budget restricts the percentage of active transistors [5, 16]. In recent years, an increasing percentage of those transistors are invested on the large LLCs utilized to bridge the gap between fast CPU cores and slow off-chip memory accesses. Specifically, LLCs occupy as much as 50% of the chip area and contribute to a significant amount of the chip's leakage power [17].

The high leakage power of the LLC comes from its large size that aims to accommodate most applications' memory footprints. However, not all workloads need the entire cache. Figure 1(a) illustrates the variable sensitivity of workloads to changes in LLC capacity. On the x-axis are multi-programmed workloads with different demands on capacity (see Section 6 for workloads and simulation details). For example, workloads LL1 and LL2 do not benefit from a larger capacity, while TH1 and TH2 perform better with larger LLCs. Further, the required cache size may also vary with different program phases, as shown in Figure 1(b). When the required cache size is smaller, some parts of the LLC can be disabled to reduce leakage power. In Figure 1(a), for example, if a 5% performance degradation is acceptable, more than half of the LLC can be disabled to save power in all but two workloads.

This research was funded by NSF grants 1218867, 1213052, 1409798, and Department of Energy under Award Number DE-SC0005026.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ISLPED'14, August 11–13, 2014, La Jolla, CA, USA.
Copyright 2014 ACM 978-1-4503-2975-0/14/08 ...\$15.00.
http://dx.doi.org/10.1145/2627369.2627661.

Figure 1: (a) Performance under different L3 sizes relative to 16MB L3 and (b) phase-dependent memory footprints.

Figure 2: Physical design of (a) uniform cache architecture (b) zoomed in portion of SRAM subarray and (c) slice-based cache organization.

In this paper, we propose low-overhead run-time mechanisms to manage LLC power consumption. We first introduce a slice-based cache organization that requires only minimal circuit overhead to shut down parts of the LLC. Based on this slice-based organization, we next propose a low-overhead approach to monitor cache access behavior and determine when to power-off/on cache slices. Specifically, we consider three important and complementary metrics that guide our slice turn-on and turn-off decisions: **utilization**, **hotness**, and **the distribution of dirty cache lines**. Our evaluation reveals that considering one metric alone may not be effective across workloads. By taking advantage of the strengths of different metrics, our comprehensive approach provides 14.1% energy savings with only 1.2% performance degradation on average.

2. SLICE-BASED CACHE

We craft our design starting from the physical implementation of a uniform cache architecture. A uniform cache consists of several smaller subarrays of SRAM cells, shown as squares in Figure 2(a). An H-tree interconnect provides equal wiring distance to all subarrays in the cache. Typically, each cache set spans multiple subarrays in the horizontal direction, highlighted in grey. Further, the bits of each way can be interleaved for less wiring on each subarray output, designated by the black striped cache line. At the die layout level, the bitlines run perpendicular to the wordlines (WL), the power (Vdd), and the ground (gnd) rails, as shown in Figure 2(b).

This results in a large trade-off between area overhead and the shutdown granularity. Since the power rails are perpendicular to bitlines, they span multiple ways in the same set. This makes it difficult to turn off single ways without either adding multiple wire routes for different ways or re-routing wires parallel to the bitlines. Both of these methods would increase the area of all subarrays and incur high overheads (>10%). For example, in Figure 2(a), additional wire routing is required to power gate only 1/4 of a single

subarray when turning off way0. The other option is to force a few ways, rather than all ways, to reside in a single subarray. Figure 2(c) shows an example that forces each subarray to store data from only a single way. Forcing a subset of ways into a single subarray increases the width of the H-tree. However, the size of the H-tree near the center of the cache, closest to the cache output, will be the same size. By utilizing CACTI [12] to analyze the area overhead, we found that the wider H-tree only incurs less than 0.5% additional area overhead, which is much smaller than the area overhead of larger subarrays in way-based shutdowns.

In this work, we utilize the second approach of constraining ways into subarrays, an organization which we call *slice*. A slice is a generically sized shut-down granularity which may range from one to all ways in a cache. Data ways in a slice are placed in the same subarray or group of subarrays sharing a sleep transistor. This work chooses a slice size of 1/16 of the total cache ways; 4-ways in our experimental design. Such design allows us to turn off entire subarrays. This results in lower overhead both due to (1) subarray sizes remaining static and (2) the need for fewer power gating transistors.

3. RELATED WORK

Prior studies have proposed circuit-level methods, including drowsy cache [6] and the gated-Vdd approach [13], to reduce the leakage power of on-chip caches. In this paper, we use the gated-Vdd technique to power-off slices for lower supply voltage overhead. Based on the circuit level techniques, several architectural approaches have been proposed [1, 2, 8, 9, 11, 15]. Some techniques [1, 15] attempt to partition caches by ways, and disable useless ways across the whole cache or sub-group of sets [11]. Basu et al. [2] exploited cache coherence to identify stale data and resize the cache. Kadjo et al. [8] facilitated power gating and migrated high temporal locality blocks to live partitions. Kaxiras et al. [9] disabled cache lines that are not likely to be reused. These techniques either require offline profiling, high hardware overhead to track the dead blocks, or utilize additional hardware that consumes non-negligible power to monitor cache accesses. Moreover, these prior way-based schemes do not easily generalize to the slice-based organization which incurs less power-gating circuit overhead.

4. METRICS OF INTEREST

In order to save energy by disabling cache slices, we need to exploit variability in cache size requirements. There are three main factors that can be used to make slice turn-on/turn-off decisions: utilization, hotness, and the distribution of dirty cache lines.

4.1 Utilization

Ideally, the cache capacity should be large enough to fit the active cache footprint of workloads, i.e. the unique cache lines referenced in a time epoch, and it indicates the utilization of the cache. Figure 3(a) and (b) show the utilization of two types of workloads. We define the utilization as the percentage of cache lines that are referenced in a time epoch in each slice. As shown in the figure, ML1 has low utilization, while TL1 has high utilization. Furthermore, the utilization varies across different cache slices and time epochs.

Low-utilization slices represent potential power-off opportunities, as disabling these slices would incur few additional cache misses. Figure 4 shows that if we shutdown slices with utilization less than 30%, we can turn off 68.1% of the LLC on average. Utilization alone can capture the power-off opportunity of most of the workloads. However, if the data are seldom reused, such as in TL1, TL2, TH1, and TH2, it misses some power-off opportunities. This observation motivates us to consider additional metrics.

4.2 Hotness

In addition to the active cache footprint, the access frequency of the stored data also helps to capture the power-off opportunity. Dis-

Figure 3: Utilization and hotness of ML1(with small active footprint but frequently reused data) and TL1(with large active footprint but rarely reused data).

Figure 4: Power-off opportunity captured by (a) Utilization (utilization<0.3 slices) (b) Hotness (Hotness<0.075 slices) and (c) Static Optimal (statically select minimum cache size that incurs <5% performance degradation).

abling a frequently accessed slice would incur more cache misses than shutting down a seldom reused slice. In this paper, we define the hotness of a slice as the number of hits to the slice divided by the total number of LLC misses in a time epoch. Thus, the hotness implies the increase in the cache miss rate if the slice is disabled.

Disabling cold slices provides different power-off opportunities, than disabling low-utilization slices. Figure 3(c) and (d) show the hotness of two type of workloads. The active footprint of ML1 is small, but the referenced data are highly reused. Disabling only cold slices for ML1 would lose considerable power-off opportunities provided by the small active footprint. On the other hand, TL1 references a large number of cache lines in each epoch, but these referenced data are seldom reused. Therefore, disabling cold slices for TL1 may provide higher power savings than disabling low-utilization slices. Figure 4 illustrates that if we shutdown the cold slices with hotness less than 7.5%, we can disable 47.6% of the LLC on average. For workloads with large but seldom reused cache footprint, such as TL1, TL2, TH1, and TH2, the hotness of slices can better capture the power-off opportunity than utilization.

4.3 Writeback of Dirty Data

When a slice in the LLC is turned off, the dirty data need to be written back. Disabling a slice with a higher number of dirty cache lines would reduce power savings, since the slice can only be powered down once all the dirty data have been written back. Furthermore, these writes would frequently fill up the write-buffer in the memory controller and could delay critical reads. Thus, a slice with less dirty data should be chosen among the slices with the same level of utilization or hotness, when deciding which slice should be powered off.

In summary, cache utilization indicates spatial access behavior, while cache hotness indicates temporal access behavior. Low-utilization slices can be disabled to save the leakage power, while cold slices can be turned off when the stored data are seldom reused to further increase the power-off opportunity. Also, when choosing which slices to power off, the number of dirty lines should play an important role. The discussion above suggests that an ideal slice power-off/on strategy should consider all these metrics.

5. EECACHE

In this paper, we propose low-overhead methods to monitor the cache access behavior, and design the power-off, power-on, and data migration policies accordingly. Figure 5 shows the overview of the proposed scheme. In the cache controller, we add a power management unit (PMU) to determine the power state of each cache slice. At the beginning of each epoch, i.e., time t in the figure, the PMU collects the cache access status from the previous epoch. Based on the cache access behavior, the PMU first decides whether the workload will benefit from a higher capacity. It then checks whether some of the slices can be turned off to save even more

Figure 5: Overview of EECache and power management process.

Figure 6: Profiling cache access behavior in each cache slice. Uc: Utilization counter; CHc/DHc: Clean/Dirty hit counter; Dc: Dirty counter.

power. Before turning off the victim slices selected by the power-off policy, the PMU decides whether each dirty block in the victim slices should be written back to the main memory or migrated to other slices. The clean blocks also need to be either discarded or migrated. After all the blocks in the victim slices are flushed, i.e., at time $t + T_{migrate}$ in Figure 5, the victim slices can be turned off to save leakage power. Below, we explain how to monitor the cache access behavior with low-overhead hardware, and describe our power management policies in detail.

5.1 Monitoring Cache Access Behavior

Figure 6 shows how to dynamically monitor the cache access behavior with small hardware overhead. To capture hotness, we use two counters (CHc and DHc) to count the number of hits to each clean and dirty cache line. The number of LLC misses is captured by a global counter (Mc) for all the cache slices. To monitor the distribution of dirty blocks, we utilize a counter (Dc) to count the number of dirty cache lines. For profiling the utilization, we develop a sampling-based method that is inspired by a prior set-sampling approach [14] to reduce the hardware overhead. We sample the utilization of only $1/64$ sets, which is enough to provide high accuracy ($>80\%$). When a cache line is inserted into the sample set, the corresponding utilization bit is set to one. The utilization counter (Uc) of the slice is then increased by one. When the cache line is evicted or invalidated, the mapped utilization bit is reset to zero and Uc is decreased by one. All the utilization bits are reset to zero at the beginning of each epoch to filter the stale data. Suppose that there are N_w ways and N_s sets in each slice, the utilization of a slice can be estimated by $Uc / (N_w * N_s * (1/64))$, and the hotness can be calculated by $(CHc + DHc) / Mc$.

5.2 Power-off Policy

As observed in Section 4, using the utilization alone to decide the power state would lose some power-off opportunities when the stored data are seldom reused. The hotness characteristic can help to identify these seldom reused slices. Moreover, with similar utilization and hotness, the slice with fewer dirty cache lines would incur a lower writeback penalty. Therefore, we consider all three factors when designing the power-off policy, as illustrated in Figure 7. We first select the power-off victims from the slices with less than U_{th} utilization. If there is no low-utilization slice, we instead select the slices with less than H_{th} hotness. Among the power-off candidates, at most N_{off} slices with fewer dirty cache lines are chosen to be turned off. We analyze the impact of different threshold settings, and empirically set $U_{th}=30\%$, $H_{th}=7.5\%$, and $N_{off}=4$.

5.3 Power-on Policy

After some slices are powered off, the power consumption is reduced but the cache misses may increase due to the smaller LLC size. Since the cache access behavior changes in different program phases, we need to determine whether the workload would benefit from a larger LLC at each epoch. We keep the whole tag array powered-on to monitor the potential hits to the powered-off portion

Figure 7: Flow chart of the power-off policy.

of the LLC. The tag arrays of the powered-off slices are called victim tags [4], and store the evicted cache lines from the active slices. A potential hit counter, V_{hit} , is increased by one when a hit occurs to the victim tags. When the hit rate to the victim tags (V_{hit}/Mc) is higher than a threshold, $HonTh$, N_{off} slices are powered on to improve performance. Increasing the value of $HonTh$ provides more power saving by turning on slices less often, but the performance degradation would also increase. We analyze the impact of different $HonTh$ settings, and find that $HonTh=10\%$ is a good value.

5.4 Data Migration Policy

The goal of the data migration process is to guarantee data coherency while reducing the miss penalty due to the loss of data in the powered-off slices. One possible solution is to migrate useful data to other active slices. During each migration, a replacement victim is selected from the active slices according to the underlying replacement policy, i.e., the LRU block in active slices. The migration of a cache line may incur an additional conflict miss if the evicted replacement victim is reused later. Thus, we choose to migrate only the clean blocks in hot-clean slices ($CHc/Mc > M_{th}$) and dirty blocks in hot-dirty slices ($DHc/Mc > M_{th}$). The M_{th} is the migration threshold and is empirically set to 4%. Note that the data migration is performed at background and does not delay the demand requests to the cache.

Our power management mechanism requires only 0.005% storage overhead in a 16MB LLC, as shown in Table 1.

Uc	Utilization counter per slice	32bits * 16(slice) = 64B
CHc	Clean hit counter per slice	32bits * 16 = 64B
DHc	Dirty hit counter per slice	32bits * 16 = 64B
Mc	Miss counter for entire LLC	32bits = 4B
Dc	Dirty counter per slice	32bits * 16 = 64B
Vhit	hit counter for victim tags	32bits = 4B
Uarray	Utilization set-sampling array	64(way) * 64(sampled set) = 512B
All		776B/16MB $\approx$ 0.005%

Table 1: Storage overheads.

Processor	16 cores; 2GHz; Out-of-order; issue width=4; 192 ROB; 32 LSQ
L1 cache	Private; 32KB per core; 4-way; 64B line size; 2 cycles
L2 cache	Shared; 4MB total; 16-way; 64B line size; 20 cycles
L3 cache	Shared; 16MB total(1MB per slice); 64-way(4-way per slice) 30 cycles; MSHR=32; 5 slices are always on for inclusion property
Memory	120 cycles access latency
Epoch	10M cycles

Table 2: System Configuration

6. EXPERIMENTAL SETUP

We evaluate our designs using gem5 [3], augmented with McPAT [10]. The area overhead is evaluated using CACTI [12]. When calculating the power consumption, the leakage power saving, the increase in dynamic power due to additional cache misses, and the additional power consumption of our monitoring mechanism, are all included. The baseline system configuration is shown in Table 2. We use a set of SPEC2006 benchmarks for multi-programmed workloads and PARSEC benchmarks for multi-threaded workloads. For the SPEC benchmarks, we fast forward 500M instructions, and run in detailed mode for 1 billion instructions. The SPEC benchmarks are classified into four categories, as shown in Table 3, according to their active cache footprint. We create the multi-programmed workloads by combining two different categories of applications to cover a broad range of cache access behavior. For the PARSEC benchmarks, we run 1 billion instructions starting from the Region of Interest (ROI) [7]. When reporting performance results, we use overall throughput ($\sum IPC_i$).

	Benchmark	Util	Benchmark	Util
LL1	GemsFDTD,tonto,tonto,povray	3.6	bodytrack	0.1
LL2	GemsFDTD,povray,povray,tonto	2.0	swaptions	0.1
ML1	dealII,xalancbmk,GemsFDTD,povray	6.8	blackscholes	0.1
ML2	gromacs,h264ref,povray,tonto	14.4	fregmine	0.7
HL1	gobmk,omnetpp,GemsFDTD,tonto	12.5	x264	4.7
HL2	hmmr,zeusmp,povray,tonto	52.9	ferret	7.5
HM1	gobmk,hmmr,dealII,gromacs	37.9	rtview	8.8
HM2	omnetpp,zeusmp,h264ref,namd	58.3	dedup	15.4
TL1	astar,mcf,GemsFDTD,povray	61.9	fluidanimate	25.9
TL2	leslie3d,libquantum,povray,tonto	98.6	canneal	72.0
TH1	astar,leslie3d,gobmk,hmmr	97.0		
TH2	libquantum,sjeng,omnetpp,zeusmp	91.6		

Table 3: Workload characteristic and classification. Util=average LLC utilization per Epoch (%). Active cache footprint: $L < M < H < T$.

Figure 8: (a) Performance and (b) L3 power saving of different power-off policies, and (c) System power, energy, and EDP saving, compared to baseline 16MB LLC.

7. EXPERIMENTAL RESULTS

We first analyze the performance impact and power-saving of the proposed EECache, as shown in Figure 8(a) and (b), comparing to using only utilization (utilization $< 30\%$ slices) or hotness (hotness $< 7.5\%$ slices) alone as the power-off metric. Powering-off low-utilization slices (Uoff) can provide 48.4% power saving on average. However, when the access footprint of the workloads is large, such as in TL2, TH1, TH2, and canneal, Uoff loses the power-off opportunity when the stored data are seldom reused. Using hotness as the power-off metric (Hoff) can disable the rarely reused slices in these high-utilization workloads. However, the Hoff policy misses the power-off opportunity when the workload frequently accesses small amount of hot data, such as in ML1, ML2, HL2, HM1, and ferret. Within similar performance degradation, our EECache can better capture the power-off opportunity by taking advantage of the strengths of different metrics. On average, EECache can provide 52.5% power-saving in the LLC, while incurring only 1.2% performance degradation. EECache can also reduce the power and energy consumption of the entire system. Figure 8(c) shows that EECache provides 14.8% system power saving. The energy consumption and the energy-delay-product (EDP) of the whole system are also reduced by 14.1% and 13.4% on average.

	PGM	EECache
Storage overhead	103.25KB area=0.907mm ² (3.3%) leakage=0.516W (2.5%)	776B area=0.100mm ² (0.4%) leakage=0.057W (0.3%)
Gated-Vdd circuit	3.0%	1.7%
Routing overhead	larger subarrays (14.3%)	wider H-tree (0.4%)
Total area overhead	20.6%	2.5%

Table 4: Hardware overheads of PGM and EECache in a 16MB LLC.

We compare EECache against a state-of-the-art way-based power-gating approach (PGM) [8]. PGM uses a high-overhead sampling array that stores the tags of the sample sets and the instructions that recently access the blocks. They determine the required cache size by analyzing the hits to the sample tags, and rely on a high-overhead prediction scheme that hashes the instructions into a counter array to determine whether a cache line is useful and should be migrated to other active ways during the transition phase. Our EECache design consumes negligible storage overhead (0.4% extra L3 area), which is smaller than the PGM scheme, as shown in

Figure 9: (a) System performance and (b) System power saving of prior way-based power-gating (PGM) [8] and our EECache.

Table 4, resulting in less leakage power consumption. Since we use slice-based shutdowns, the gated-Vdd circuit and routing overhead are smaller than the fine-grained way-based shutdowns in PGM. As a result, the overall area overhead of EECache is only 2.5%, much less than the 20.6% overhead in the PGM scheme.

Figure 9 illustrates the system performance and power saving of prior PGM and our EECache. Prior PGM approach relies on the hits to LRU ways to determine the required cache size. However, their scheme does not consider the active footprint of the LLC, thus sacrificing the performance of some high-utilization workloads, such as TH1 and TH2, to provide higher power saving. The PGM also incurs higher than 5% performance degradation for ferret and dedup, due to the failure to detect the power-on demand in these workloads. Our EECache better tradeoff performance degradation and power savings, and incurs less than 5% performance degradation in all workloads, while providing 14.8% system power saving on average. These results show that our EECache can provide similar power savings to finer grained approaches with both smaller hardware overheads and less performance degradation.

8. CONCLUSION

This paper explores low-cost LLC power management policies for multi-programmed and multi-threaded benchmarks. Based on our extensive experimental analysis, we can conclude that simultaneously exploiting three key metrics, i.e., utilization, hotness, and the distribution of dirty cache lines, is necessary to design the power management policies for an energy-efficient LLC. Our EECache achieves 14.1% energy saving with less than 2% performance degradation, and consumes negligible hardware overhead.

9. REFERENCES

- [1] D. Albonesi. Selective cache ways: on-demand cache resource allocation. In *MICRO*, 1999.
- [2] A. Basu et al. Freshcache: Statically and dynamically exploiting dataless ways. In *ICCD*, 2013.
- [3] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 2011.
- [4] J. Cong et al. An energy-efficient adaptive hybrid cache. In *ISLPED*, 2011.
- [5] H. Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.
- [6] K. Flautner et al. Drowsy caches: simple techniques for reducing leakage power. In *ISCA*, 2002.
- [7] M. Gebhart et al. Running parsec 2.1 on m5. Technical report, The University of Texas at Austin, Department of Computer Science, October 2009.
- [8] D. Kadjo et al. Power gating with block migration in chip-multiprocessor last-level caches. In *ICCD*, 2013.
- [9] S. Kaxiras et al. Cache decay: exploiting generational behavior to reduce cache leakage power. In *ISCA*, 2001.
- [10] S. Li et al. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.
- [11] S. Mittal et al. Flexiway: A cache energy saving technique using fine-grained cache reconfiguration. In *ICCD*, 2013.
- [12] N. Muralimanohar et al. Cacti 6.0: A tool to model large caches. Technical report, HP Lab, 2009.
- [13] M. Powell et al. Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In *ISLPED*, 2000.
- [14] M. K. Qureshi et al. A case for mlp-aware cache replacement. In *ISCA*, 2006.
- [15] K. Sundararajan et al. Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps. In *HPCA*, 2012.
- [16] G. Venkatesh et al. Conservation cores: reducing the energy of mature computations. In *ASPLOS*, 2010.
- [17] D. Wendel et al. The implementation of power7: A highly parallel and scalable multi-core high-end server processor. In *ISSCC*, 2010.