Limiting the Number of Dirty Cache Lines

Pepijn de Langen and Ben Juurlink
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
Email: pepijn@ieee.org, b.h.h.juurlink@tudelft.nl

***Abstract*—Caches often employ write-back instead of write-through, since write-back avoids unnecessary transfers for multiple writes to the same block. For several reasons, however, it is undesirable that a significant number of cache lines will be marked "dirty". Energy-efficient cache organizations, for example, often apply techniques that resize, reconfigure, or turn off (parts of) the cache. In such cache organizations, dirty lines have to be written back before the cache is reconfigured. The delay imposed by these write-backs or the required additional logic and buffers can significantly reduce the attained energy savings. A cache organization called the clean/dirty cache (CD-cache) is proposed that combines the properties of write-back and write-through. It avoids unnecessary transfers for recurring writes, while restricting the number of dirty lines to a hard limit. Detailed experimental results show that the CD-cache reduces the number of dirty lines significantly, while achieving similar or better performance. We also use the CD-cache to implement cache decay. Experimental results show that the CD-cache attains similar or higher performance than a normal decay cache, while using a significantly less complex design.**

I. Introduction

To hide the long latency of off-chip memories, most processors employ several levels of caches, either on- or off-chip. For writes, a distinction is made between two policies: write-back or write-through. The advantage of write-back over write-through is that it avoids unnecessary transfers for recurring writes to the same block, which results in higher performance and less power dissipation in the memory bus.

Although a write-back cache generally outperforms a write-through cache, it will contain a significant number of dirty cache lines. Table I depicts the average number of dirty cache lines in a 32kB write-back cache, measured using the experimental setup explained in Section IV. On average, 45% of the cache lines are marked dirty in a conventional write-back cache. For several reasons, a high number of dirty cache lines is undesirable. Dirty cache lines cause additional delay or complexity for caches that need to flush cache lines on certain occasions. Examples are caches that need to be flushed on context switches as well as caches that need to empty cache lines to use partial shutdown, drowse modes, or cache reconfiguration [1]. Another reason against a large number of dirty caches lines comes from the area of fault tolerance. In caches, error detection is sufficient for clean data, since date can be reloaded from the next memory level when an error is detected. For dirty data, however, error correction is required since there is no other up-to-date copy if an error occurs.

This research was supported in part by the Netherlands Organisation for Scientific Research (NWO).

TABLE I: Percentage of dirty cache lines in a 32kB cache.

gcc	mcf	parser	twolf	vortex	vpr
51.5%	26.3%	45.5%	51.9%	39.6%	57.1%

In this work we explore the potential of a cache organization called the *clean/dirty cache* (CD-cache). The proposed organization consists of a cache that is only used for loads, and a much smaller write-back cache that is used to store dirty data. Experimental results show that the CD-cache attains similar or higher performance than a write-back cache, while restricting the number of dirty cache lines to a hard upper limit. In addition, since accessing a smaller cache structure requires less energy than accessing a larger one, the CD-cache reduces the dynamic energy consumption by directing all writes and a significant number of loads to the smaller cache structure. This is also exploited in the *filter cache* [2].

This paper is organized as follows. Section II discusses related work. The design of the CD-cache is presented in Section III. Experimental results are presented in Section IV, and a case study with cache decay is provided in Section V. Section VI summarizes this work and presents directions for future research.

II. Related Work

The CD-cache shows some similarities with the *write cache* [3], which in turn shows similarities with the *miss cache* and *victim cache* [4]. The write cache is in principle a write-buffer except that instead of writing updates to the next memory level as soon as possible, it only writes back data in case it needs to make room for new entries. This way, it is possible to coalesce more writes and hence decrease the amount of write traffic. The write-cache, however, uses the allocate-on-write and the no-fetch-on-write policies. This requires valid bits for each sub-block in a cache line and imposes additional complexity to support stores smaller than the sub-block size.

Chu et al. [5] evaluated several write buffer configurations. They propose to either flush the write buffer at certain intervals or on certain events, or to write back entries from the write buffer in the background. This is done by writing updates both to the L1 data cache and to the write buffer, and having

978-3-9810801-5-5/DATE09 © 2009 EDAA

the write buffer clear the dirty bit in the L1 data cache after writing back data. Lee et al. [6] proposed a technique called *Eager Write-Back*. This write-back cache does not wait with writing back data until a line is evicted. Instead, write-backs are issued whenever the memory bus is idle. None of these approaches, however, put a hard limit on the number of dirty cache lines.

We also present a case study using the CD-cache for a cache energy reduction technique called *cache decay* [1]. Cache decay is based on a technique called the *gated-Vdd* [7], which saves energy by turning of unused memory cells. In cache decay, this technique is used to turn off individual data cache lines. Our work extends upon this work by providing a method to handle dirty data efficiently.

III. Clean/Dirty Cache

The proposed design comprises two caches: a primary *clean cache* and a much smaller secondary cache called the *dirty cache*. While the clean cache is only used to store 'clean' data, the cache lines in the dirty cache are marked 'dirty' by definition. Furthermore, the contents of these caches are mutually exclusive. Figure 1 shows a schematic description of the CD-cache.

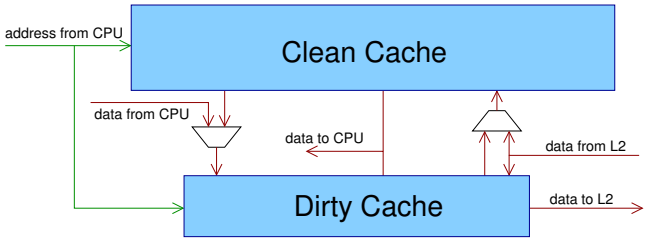


Fig. 1: Schematic representation of the clean/dirty-cache

Writes always go to the dirty cache and are never allocated in the clean cache. If a write misses in the dirty cache, a lookup is performed in the clean cache. If the data is found in the clean cache, this data is invalidated and copied to the dirty cache, thereby keeping the two structures mutually exclusive. If the data is not found in either cache, the cache line is fetched from the next memory level and stored in the dirty cache. The dirty cache employs the write-back policy. Data that is evicted from the dirty cache is always written back to the next memory level, since it is dirty by definition. When this happens, the clean cache is probed again to see if the corresponding cache line is still available, i.e., this cache line is not used to store other data and is therefore in *invalid* state. If this is the case, the data is also written back to the clean cache. Our experimental results show that this causes a small reduction of L2 accesses.

Data that is read can reside in either the clean or the dirty part. Therefore, when a read is issued, a lookup has to be performed in both caches. When a read misses in both caches, the data is fetched from the next memory level and is stored in the clean cache.

TABLE II: Baseline memory hierarchy.

L1 Data Cache	32 or 36kB, 2-way, 32B lines, 2 cycles
L1 Inst. Cache	32kB, 2-way, 32B lines, 2 cycles
L2 Unif. Cache	1MB, 4-way, 128B lines, 12 cycles
Memory Latency	100 cycles

Using such a split cache design can impact both the access time and the energy consumption. Since writes always go to the dirty cache and since both structures are mutually exclusive, there is no negative impact on the *hit time* for writes. For reads, the data can reside in either cache. This implies the tags in both caches have to be checked before the data can be read out. It is therefore assumed that read hits experience a one cycle additional delay compared to a normal write back cache of the same size. This furthermore implies that for every read, the proposed cache organization spends twice as much energy on comparing tags as compared to a normal cache. However, since the dirty cache is much smaller than the clean cache, reading data from the dirty cache takes less energy than reading from the clean cache or from a normal cache. The same is true for writes, which always go to the smaller dirty cache but often incur additional tag checks. A more detailed discussion of the energies involved with comparing tags and reading out data is presented in Section IV.

An important property of the proposed design is that data allocated by load instructions can only be replaced by other load instructions, and that data allocated by store instructions can only be replaced by other store instructions. This implies that issuing a load will never cause a write-back event, and as such, are not delayed by these. Loads can, however, be delayed by write-back events due to previous instructions.

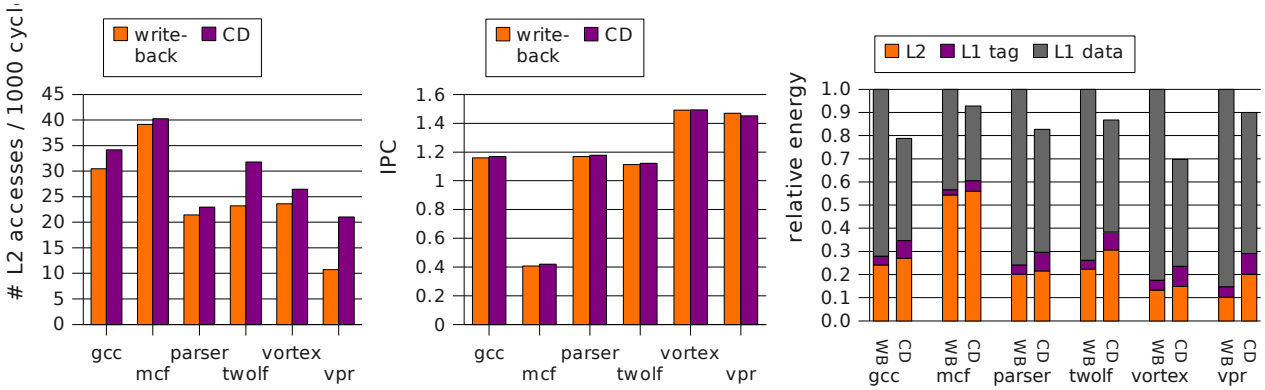
By limiting the size of the dirty cache, the maximum number of cache lines marked dirty is significantly reduced compared to a cache that employs a write-back policy. By keeping a small windows of recent stores in the L1 cache, a significant number of L2 cache accesses can be avoided compared to a cache that employs write-though.

IV. Experimental Evaluation

A. Experimental Setup

For the experiments in this paper, a selection of the SPEC2000 [8] integer benchmarks has been used. These benchmarks could be compiled and simulated without errors. All benchmarks were simulated for at least 500 million cycles, using sim-outorder from the SimpleScalar toolset [9], which was extended with a detailed memory hierarchy, including write-buffers and miss-status-holding-registers (MSHRs). Sim-outorder was configured to resemble a modern wide-issue superscalar processor. The most important parameters for the baseline system are shown in Table II. For the CD-cache, the same parameters were used except from a 1 cycle higher L1 read hit time.

We calculate energy by multiplying counters produced by sim-outorder with the corresponding costs. The exact energies consumed by certain events, such as L1 accesses, depend



(a) L2 accesses per 1000 cycles (b) IPC (c) Dynamic energy consumed in L1 and L2

Fig. 2: Results for write-back of 36kB and CD-caches of 32+4kB

on the used implementation and technology. Table III depicts results produced by CACTI 5.3 [10] for 32kB 2-way set-associative caches with 32-byte cache lines, using high-performance transistors. Although there are significant differences for different implementations, the relative differences between the energy consumption of various parts is fairly constant. More specifically, the dynamic energy involved with one access to the cache is approximately twice as high as the energy dissipated through leakage in a single cycle. Similarly, CACTI simulations show that approximately 5% of the energy is dissipated by tag comparisons, and that a 4kB cache requires less than half the energy of a 32kB cache.

The following model is therefore used to measure energy consumption: An L1 cache access is assumed to consume twice as much dynamic energy as is dissipated through leakage in one cycle. Of this energy, 5% is used for the tag comparison and 95% for reading out data. CACTI results show that accesses to the 1MB L2 cache require approximately 5 times as much energy as accesses to the L1 cache. This implies that an access to L2 requires 10 times as much energy as is dissipated through leakage in the L1 cache. It is noted that Hu et al. [1] use the same energy relation between L2 accesses and L1 leakage in most of their work. For the CD-cache, twice as many tag comparisons are required. On the other hand, some read accesses and all write accesses are serviced by the much smaller dirty cache. According to CACTI, accesses to a 4kB cache require less than half the energy of accesses to a 32kB cache. It is therefore assumed that accesses to the dirty cache reduce the energy involved with reading out data by half. Leakage energy is dissipated every cycle, even when there are no cache accesses.

The CD-caches are equipped with a 32kB cache structure to store the clean data, and a 4kB structure to store dirty data. Since the capacity of a cache has a significant impact on its performance, these CD-caches are compared with baseline caches of 36kB.

TABLE III: Energy consumption of 32kB 2-way set-associative caches with cache lines of 32 bytes.

technology [nm]	45	45	65	65
number of banks	1	2	1	2
access time [ns]	0.828	0.750	1.392	1.261
dynamic read energy [nJ]	0.061	0.048	0.107	0.084
leakage power [W]	0.040	0.031	0.046	0.035
dynamic power by tags	4.9%	5.9%	3.1%	5.9%

B. Experimental Results

Figure 2a depicts the number of accesses to the unified L2 cache for both write-back and CD-caches. As expected, the CD-cache in general increases the number of L2 accesses compared to the baseline write-back cache. In most cases these increases are moderate, but for the twolf and vpr benchmarks they are quite significant, both in absolute and relative terms.

Although the CD-cache increases the number of L2 accesses, this does not necessarily imply a performance reduction. Figure 2b depicts the IPC for the baseline and CD-caches of 36kB. The difference between the baseline and CD-cache is rather small, and in most cases are in favor of the CD-cache. Even for the twolf and vpr benchmarks, which showed significant increases in L2 accesses, there is no real reduction in performance. This is due to the following reasons. First, the out-of-order processor is able to effectively hide the latency of the additional L2 accesses, especially if they do not occur concurrently with other L2 accesses. Second, due to the limited number of dirty cache lines in the CD-cache, the accesses to L2 are higher in frequency but also better spread over time. More importantly, in a normal write-back cache a read may result in both a transfer from the next memory level to retrieve the requested data and a transfer back to the next memory level when dirty data is evicted from the cache. In the CD-cache, reads can never result in writing back data, and as a result experience less delay.

Improving performance is, however, not the primary objective of the CD-cache. The goal is to limit the number of dirty cache lines while maintaining comparable performance. Furthermore, since the CD-cache is targeted at cache energy

reduction techniques, it is important to not increase energy consumption in other parts. Figure 2c depicts the dynamic energy consumption. The energy consumption in these figures is measured by using the results produced by SimpleScalar and multiplying these by the corresponding energy cost, as described in Section IV-A. Fig. 2c depicts separate results for the energies involved with tag comparisons, reading out data, and L2 accesses.

For all benchmarks the CD-cache actually reduces energy consumption. While twice as many tag comparisons are performed and more requests to the L2 cache are issued, the energy dissipated by reading/writing data to/from the L1 cache is reduced. All write operations and also some reads are performed on the much smaller dirty cache, which requires significantly less energy per access.

V. Cache Decay Case Study

Cache decay [1] reduces the static power consumption in caches by switching off the power supply to cache lines that have not been used for a certain number of cycles. Each line is associated with a 2-bit saturating local counter. The local counters are incremented on ticks from a global counter, while a local counter is reset to 0 when a cache line is used. When a local counter saturates, it generates a signal to switch off the cache line. The valid bits are never switched off and indicate whether a cache line is powered on.

In [1] it was shown that, based on the decay interval, a significant number of cache lines will be switched off. By switching off cache lines, however, the cache miss rate increases. This can have a negative impact on performance. Furthermore, the energy savings obtained by switching off cache lines has to be offset against the increase in accesses to the next memory level. The results show that a decay cache can result in the same miss rate as a normal cache, while having fewer powered-on cache lines, or in an improved miss rate while having the same number of active cache lines.

Dirty cache lines, however, pose a problem as dirty lines have to be written back to memory before they can be decayed. Hu et al. [1] proposed to avoid bursts of write-backs on the global tick signal by cascading the global signal from one local counter to the next with a one cycle delay. This, however, assumes that write-backs can always be written from the cache to a buffer without experiencing stalls. The memory subsystem is a known bottleneck and the same bandwidth that is used for write-backs is also used for other means. This makes it hard, or at least very costly, to always guarantee sufficient bandwidth for writing back one cache line per cycle. Other solutions are to stall the decay process whenever the write-back buffer is full or to only decay dirty cache lines in case there is sufficient room in the write buffers. Such solutions, however, would require significantly more complex decay hardware and would therefore consume additional energy.

We now show by experiments how the CD-cache can be used to efficiently implement cache decay. The main advantage of separating clean and dirty data in the CD-cache is that data in the clean cache can be decayed without a problem. This makes it possible to implement cache decay efficiently in this part of the cache, without the need to include additional write-back buffers.

Cache decay using the CD-cache is compared to a baseline write-back cache that employs cache decay. The write-back cache is assumed to have an additional write buffer with a limited number of entries (4-16) for writing back data. It is furthermore assumed that cache lines cannot be decayed if there is no room in the write buffer.

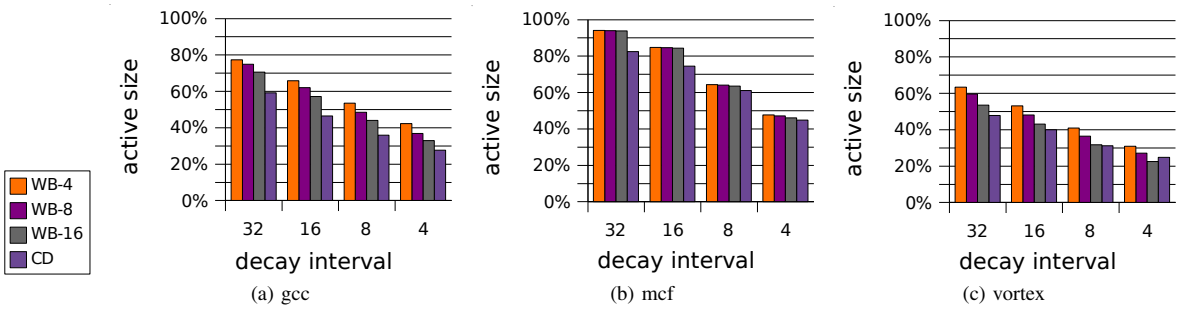
To limit the complexity and simulation time, cache decay is used only in the L1 data cache. Furthermore, performing cache decay in L2 would give unrepresentative results as the employed benchmarks do not significantly stress the L2 data cache. In reality, cache decay would benefit any on-chip cache, and especially large on-chip L2 caches as are common in many modern CPUs. The results presented here are applicable to any cache level.

Since cache decay is performed only in the L1 data cache, the static energy saved by cache decay can be estimated by the average fraction of L1 that is decayed multiplied with the duration of the benchmark. This has to be offset against the increased number of accesses to the next memory level. An exact comparison between these two is very implementation and technology specific. We therefore use the same energy model as used in Section IV, which related the dynamic energy involved with various operations in the caches to the leakage current of the whole L1 data cache during 1 cycle.

Figure 3 depicts the average active size when employing cache decay with varying periods on both a write-back and a CD-cache. The active size of a cache is defined as the percentage of cache lines that are powered on. As explained before, cache decay in a normal write-back cache can be limited due to the limited size of the write buffer. The results depicted in this figure include data for write-back caches with additional write buffers of 4, 8, and 16 entries. These are labelled WB-4, WB-8, and WB-16 respectively.

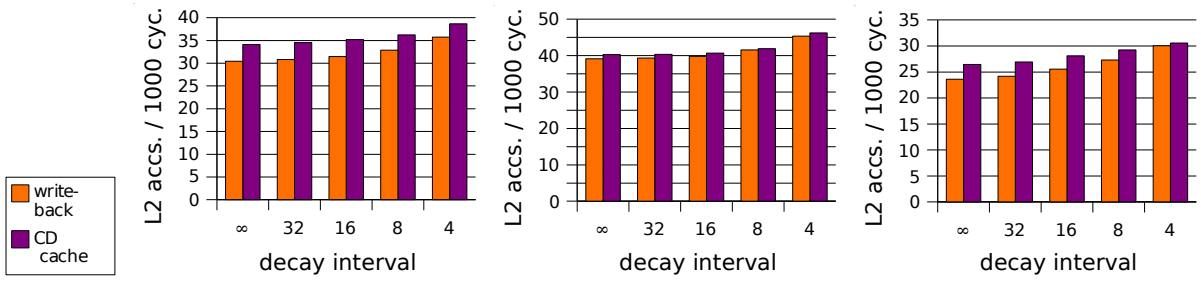
Figure 3 shows that the effect of cache decay can be limited if there is insufficient room to write back dirty cache lines. While for some benchmarks, like gcc, a larger write buffer leads to a decreased active size, a write buffer of more than 16 entries is required to attain the same number of decayed cache lines as the CD-cache. This is because the CD-cache can always decay cache lines in the clean part that have not been used recently. The write-back cache, on the other hand, may need to write back significantly more dirty cache lines than there is room for in the write buffer.

When, based on the decay interval, the number of active cache lines becomes really small, the size of the dirty cache starts to play a significant role. This happens, for example, with vortex with a decay interval of 4000 cycles, depicted in Figure 3c. In this case, the CD-cache has on average 287 active cache lines in total. With a clean part of 32kB (1024 cache lines) and a dirty part of 4kB (128 cache lines), this implies that only 159 clean cache lines are active on average, and that almost half of the active cache lines are dirty cache lines that cannot be decayed. When targeting such a significant



(a) gcc (b) mcf (c) vortex

Fig. 3: Average active size for write-back and CD-caches using cache decay (intervals in $\times 1000$ cycles).



(a) gcc (b) mcf (c) vortex

Fig. 4: L2 accesses per 1000 cycles for a write-back and a CD-cache using cache decay (intervals in $\times 1000$ cycles).

reduction in active cache lines, a smaller dirty cache should be employed.

Figure 4 depicts the number of L2 accesses per 1000 cycles for the write-back cache and the CD-cache, using cache decay with various decay periods. This figure also includes results for the write-back and CD-caches without cache decay. These are labelled ∞, as they can be viewed as having an infinite decay period. The write-back caches in these experiments are equipped with a 16-entry write buffer.

As expected, the number of active cache lines decreases and the amount of traffic to and from L2 increases with decreasing decay periods. With a large decay period, hardly any of the cache lines are turned off. With a very small decay period, on the other hand, a significant number of additional L2 accesses is generated. The optimal decay period depends on the application and on the difference in energies due to leakage currents and L2 accesses. Hu et al. [1] describe a technique to adjust the decay period to the behavior of an application. This, however, is beyond the scope of this work.

The exact energy savings attained by cache decay depends on the relative amount of static energy consumed by the caches versus the amount of dynamic energy required for a transaction from or to the next memory level. In some cases, it might be worthwhile to have a few additional accesses to the next memory level to allow a significant part of the cache to be shut down. In other cases, it might be better to shutdown a smaller part of the cache, while making sure the number of accesses is not increased.

Figure 5 depicts the total energy consumption in the L1 data cache, scaled to the baseline cache. The energy model described in Section IV is used, where an L1 access is assumed to consume twice as much energy as is dissipated through leakage in one cycle, and where an L2 access is assumed to cost 5 times as much as an L1 access. The results indicate the amounts of energy consumed in the L1 data cache, separated by static energy consumption due to leakage currents (L1 leakage), dynamic energy for reading and writing data (L1 data), and dynamic energy for tag comparisons. Since cache decay increases the number of L2 accesses, we also include the energy consumption due to additional L2 accesses in this figure. As before, the write-back caches are equipped with a 16-entry write buffer.

The results show that the CD-cache with cache decay consumes significantly less energy than a write-back cache using cache decay. The additional energy consumed by the increased number of tag comparisons in the CD-cache is easily offset by the decrease in energy required for reading and writing data. By separating clean and dirty data, the CD-cache can employ cache decay on the clean part much more successfully than a write-back cache, leading to a decrease in the total energy.

So far, we have assumed that L2 accesses consume 10 times as much energy as is dissipated by the L1 cache in a single cycle. This ratio may be different when using other cache configurations or a different technology. From Figure 5, however, it can be seen than even when L2 accesses would dissipate twice as much energy, the CD-cache would still consume less energy than a write-back cache.

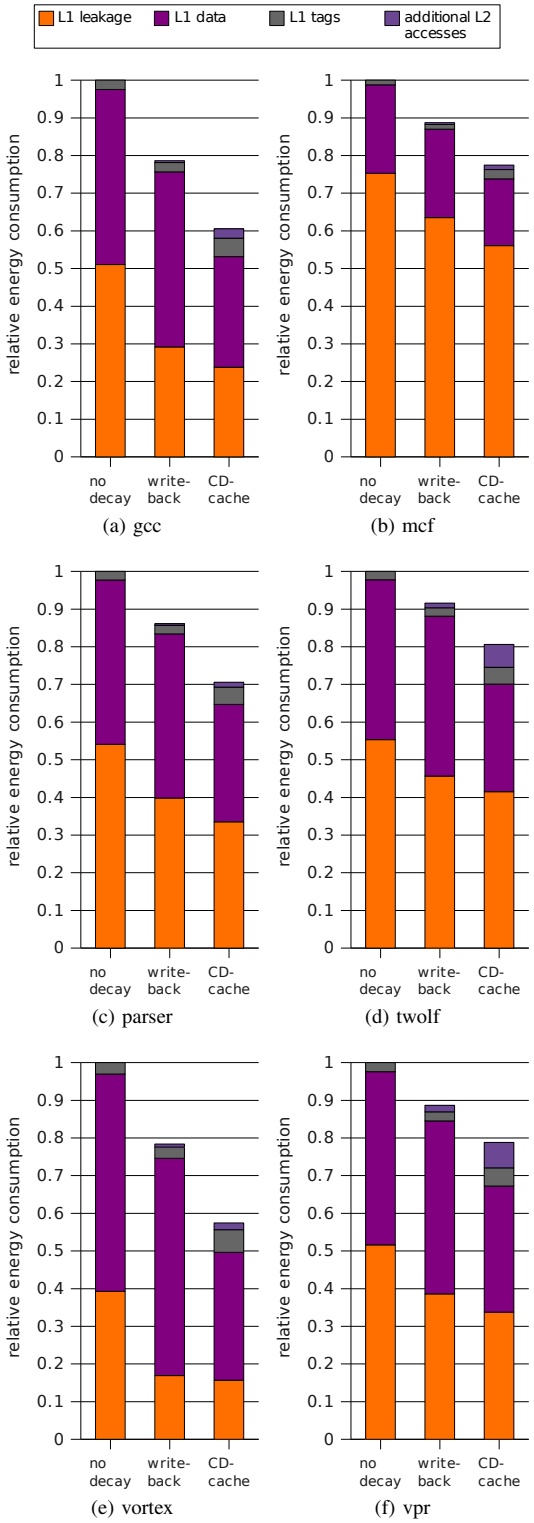


Fig. 5: Relative energy consumption in the L1 data cache when using cache decay with a period of 16000 cycles.

VI. Conclusions

We have proposed the clean/dirty-cache, a novel cache organization for handling writes in on-chip caches. The most important property of the CD-cache is that it limits the number of 'dirty' cache lines without having to 'write-through'. As a result, its performance is comparable to that of a conventional write-back cache of the same size, while the number of dirty cache lines is reduced by an order of magnitude. We have also shown that the CD-cache, although generating more write-backs, improves performance slightly compared to a write-back cache.

In the proposed organization, all writes and some reads are serviced by much smaller cache structure. It was shown that this reduces the total cache energy consumption compared to a normal write-back cache.

Reducing the number of dirty cache lines can be beneficial in a number of ways. In this paper, it was shown how the CD-cache can be used for a simple and efficient cache decay implementation. Similarly, the CD-cache should be a prime candidate for implementing other cache energy reduction techniques, as well as for fault-tolerant caches.

In future research, we intend to investigate how the CD-cache can be used to assist other energy reduction techniques, and how it can be used to build energy-efficient fault-tolerant caches.

References

[1] Z. Hu, S. Kaxiras, and M. Martonosi, "Let Caches Decay: Reducing Leakage Energy via Exploitation of Cache Generational Behavior," *ACM Trans. on Computer Systems*, vol. 20, no. 2, pp. 161–190, 2002.

[2] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in *Proc. of the ACM/IEEE Int. Symp. on Microarchitecture*, 1997, pp. 184–193.

[3] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *Proc. Int. Symp. on Computer Architecture*, 1990, pp. 364–373.

[4] ——, "Cache Write Policies and Performance," in *Proc. Int. Symp. on Computer Architecture*, 1993, pp. 191–201.

[5] P. P. Chu and R. Gottipati, "Write Buffer Design for On-Chip Cache," in *Proc. IEEE Int. Conf. on Computer Design*, 1994, pp. 311–316.

[6] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens, "Eager Writeback - A Technique for Improving Bandwidth Utilization," in *Proc. ACM/IEEE Int. Symp. on Microarchitecture*, 2000, pp. 11–21.

[7] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: a Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in *Proc. of the Int. Symp. on Low Power Electronics and Design*, 2000, pp. 90–95.

[8] S. P. E. Corporation, http://www.spec.org/.

[9] T. Austin *et al.*, "Simplescalar 3.0," http://www.simplescalar.com/.

[10] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.3," http://www.hpl.hp.com/research/cacti/.