Energy Efficient Last Level Caches via Last Read/Write Prediction

***Abstract*—The size of the Last Level Caches (LLC) in multi-core architectures is increasing, and so does their power consumption. However, a lot of this power is wasted on unused or invalid cache lines. For dirty cache lines, the LLC waits until the line is evicted to be written back to memory. Additionally, dirty lines compete with read requests (prefetch and demand), increasing pressure on the memory controller. This paper proposes a Dead Line and Early Write-Back Predictor (DEWP) to improve the energy efficiency of the LLC. DEWP early evicts dead cache lines with an average accuracy of 94%, and only 2% false positives. DEWP also allows scheduling dirty lines for early eviction, allowing earlier write-backs. Using DEWP over a set of single and multi-threaded benchmarks, we obtain an average of 61% static energy savings (while maintaining the performance) for inclusive and non-inclusive LLCs.**

I. Introduction

Chip Multiprocessors (CMPs) have become the de facto standard processor design. Chip manufacturers are increasing the number of cores per chip and the amount of on-chip memory to improve application performance. Additionally, power dissipation has become one of the major concerns for computer architects. As researchers have pointed out [5] the amount of resources that can be simultaneously powered in a chip is limited.

In order to increase the energy efficiency in current CMPs, cache memories need to be taken into consideration. According to [14], the amount of power consumed by cache memories in Niagara, Niagara2, Xeon (Tulsa) and Alpha 21364 processors corresponds to an average of 15% of the total chip power. To reduce it, researchers [1], [19] have proposed several prediction mechanisms to keep only useful information in the cache.

However, previous approaches do not take into account that modified or dirty cache lines remain turned on for long periods of time, wasting energy while they could be evicted early. The gains can be increased even for dirty cache lines, by performing an early write-back to the memory. Thereby, energy consumption, as well as pressure on the memory controller, can be reduced. Lee et al. [12] early write-back dirty lines at the LRU position, but their proposal still consumes a lot of energy by not evicting the line when the last write operation occurs. Using a perfect mechanism, we show that turning off invalid lines and dead lines can save 82% of static energy from the Last Level Cache (LLC) on average (see Section II).

This paper proposes the Dead Line and Early Write-Back Predictor (DEWP) mechanism, consisting of a last read/write predictor operating at the cache line granularity.

The *last read* prediction aims to save energy by turning off dead or invalid cache lines. The *last write* prediction performs early write-backs of dirty cache lines to main memory, since these lines will not be modified anymore. Both last read and last write predictions detect whenever a line receives its last access, prioritizing those lines for early eviction.

The last read predictor uses the access history to predict when a cache line becomes dead and can be turned off. The data is considered dead whenever the cache line receives its last read before it gets evicted or invalidated.

The last write predictor allows dirty cache lines to be early written back when it detects the last write operation, reducing the pressure on the memory controller between read and writes during bursts of requests. Furthermore, performing the early write-back of dirty lines also enables those lines to be turned off whenever a last read is predicted.

Both predictors reduces cache pollution, prioritizing the eviction of completely dead lines. All the cache lines that would be normally evicted from the LLC by the replacement policy are considered completely dead since their last access. By early evicting completely dead lines, cache lines that are still alive can stay longer inside the cache.

We make the following contributions:

- We propose DEWP, a last read/write predictor for the Last Level Cache on CMPs.
- Using the last read predictor, we turn-off cache lines after they receive the last read before the line gets evicted.
- Using the last write predictor, we early write-back the dirty cache lines after they receive the last write, reducing the pressure in the memory controller.
- Combining both prediction results, the mechanism detects the last access to a cache line, marking it for early eviction, thus, improving the cache utilization.
- We evaluate DEWP, correctly predicting 94% of the LLC accesses, with only 2% of false positives. This translates into 61% of static cache energy savings and 2% of performance improvements when averaged over SPEC-CPU2006, SPEC-OMP2001 and NAS-NPB.

II. Motivation

A. Sensitivity to Early Write-back

Dirty cache lines remain in the cache until they are evicted by another line request. However, they can be sent to write-back earlier when the last write operation is detected [19]. When predicting the last write operation, a dirty cache line is available for write-back earlier. Thus, the time window to write it back to memory becomes longer, reducing pressure to the memory controller. Additionally, by using a last write

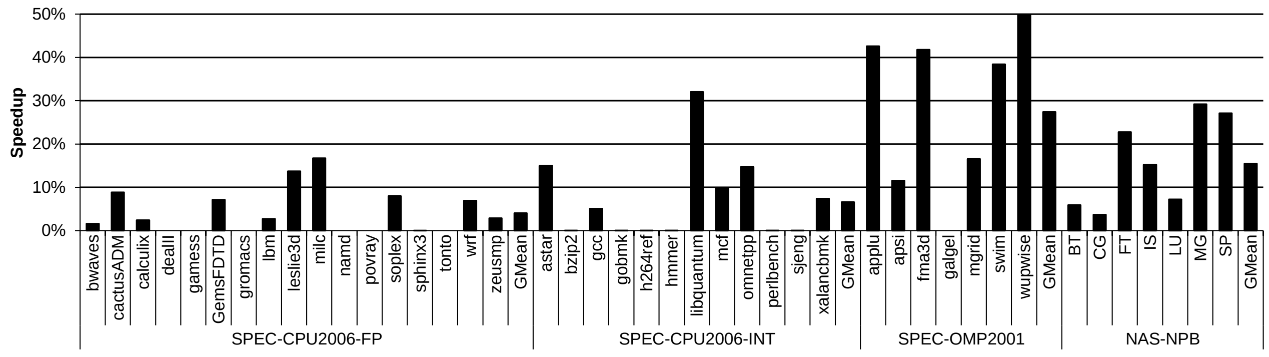

Fig. 1. Normalized performance of a perfect early write-back system with no write-back contention on a 8 core CMP.

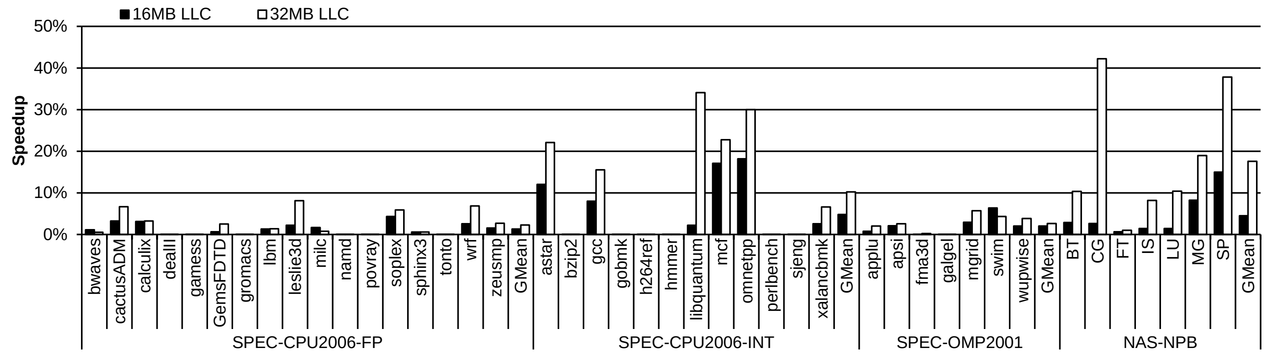

Fig. 2. Application speedup through LLC size sensitivity with 8 MB (baseline), 16 MB and 32 MB.

predictor, dirty lines can be evicted earlier, thereby increasing the potential cache capacity.

In order to show the potential benefit of a perfect last write predictor, Figure 1 shows the performance improvement of a system with no write-back operations to memory normalized to a conventional CMP with write-back. This way, we show the potential of performing write-back operations without interfering to read requests in the memory controller. Since most data accesses tend to occur in bursts [19], reducing memory pressure during those bursts is a key for memory performance.

Figure 1 shows an average 5% performance improvement for single threaded benchmarks (SPEC-CPU2006) and 21% for multi-threaded benchmarks (SPEC-OMP2001 and NAS-NPB). As expected, multi-threaded applications present higher performance gains due to their higher memory pressure.

B. Sensitivity to LLC Capacity

Cache capacity is a performance limitation for many benchmarks. In order to show the potential benefit of a dead line predictor, we perform a design space exploration with different LLC cache sizes to demonstrate the sensitivity to cache capacity for the benchmarks used in this paper.

Figure 2 shows the speedup obtained by increasing the LLC cache size. In this experiment, the baseline LLC is an 8 MB cache compared to a 16 MB and 32 MB LLC. In order to evaluate the LLC capacity sensitivity of all benchmarks, the LLC latency is maintained for all the configurations.

Benchmarks with a large working set such as NAS-NPB, or with a great amount of memory operations, such as *libquantum* or *mcf*, benefit more from a larger cache. A successful dead line predictor would eliminate cache pollution, by early evicting dead lines and therefore virtually increase the cache capacity. Such a predictor would increase performance of benchmarks with bigger working sets and it would decrease leakage for other benchmarks by turning off unused cache lines of the large LLC.

C. Energy Savings in LLC

The LLC static energy usage can account for more than 50% of the total energy consumption of the LLC [13]. Figure 3 shows the maximum theoretical static energy savings considering that cache lines could be turned off after their last access or whenever the cache line becomes invalid. For this experiment, we consider a perfect mechanism without any overhead in terms of energy consumption.

Figure 3 shows an average of 80% energy reduction for all benchmarks evaluated. The results show that benchmarks with higher energy savings are those which receive the least amount of accesses per cache line on average. These benchmarks have a low data reuse ratio and therefore offer higher opportunities for energy savings. For instance, the *sphinx3* benchmark has a high cache line reuse ratio, with more than 50% of the cache lines accessed more than 16 times before the line gets evicted. Therefore, this benchmark shows small energy savings with a perfect mechanism.

On average, more than 90% of the LLC lines of the evaluated benchmarks receive only one access before the line

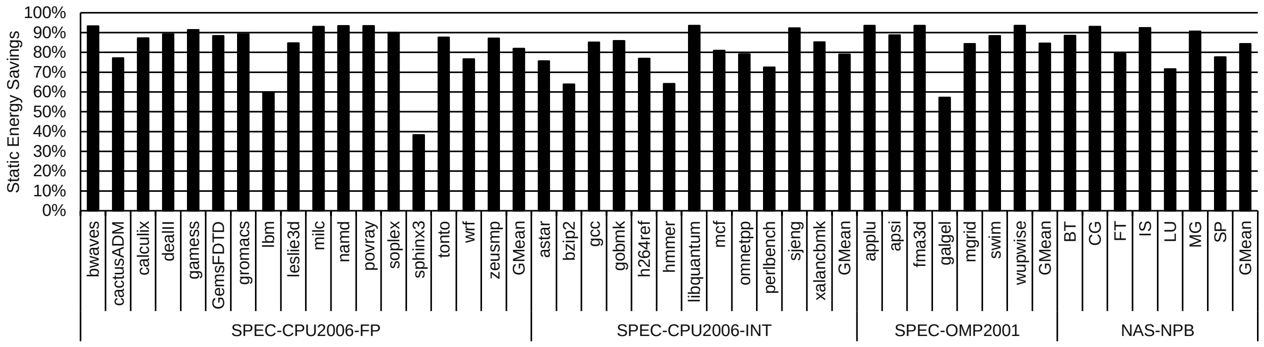

Fig. 3. Maximum static energy saving considering a perfect last access predictor using an 8 MB LLC.

gets evicted, enabling high potential energy gains for all the benchmarks, since a large part of the LLC lines are dead on arrival.

D. Overall Potential Benefits

The previous three subsections have shown the potential benefits of a perfect dead line predictor in terms of performance, sensitivity to LLC capacity and static energy consumption. First, we showed performance improvements of up to 21%, second, we detected benchmarks with high sensitivity to the LLC capacity, and finally, for all suites we showed the potential average static energy saving of 80%. All the experiments above show the potential benefits of the mechanism proposed in this paper.

III. Dead Line and Early write-back Predictor

DEWP is a Dead Line and Early Write-back Predictor to detect last read and write accesses to LLC cache lines. DEWP uses recent access information stored in an Access History Table (AHT) to predict usage patterns. The combination of traditional gated V_{DD} circuit techniques [17] and DEWP allows to power off cache lines once they are predicted dead, therefore saving static energy.

A. Overview of the Mechanism

Figure 4 shows for a set of LLC cache lines the structures required to build DEWP. These structures are: 1) *cache line metadata* which adds information for every cache line in the LLC, and; 2) An *Access History Table (AHT)* that stores the prediction information for the LLC. The cache line metadata guides the cache line predictions. Each cache metadata line includes the following fields:

- An *On/Off flag* to indicate if the cache line is switched on or off;
- A *Train flag* to indicate if accesses to the cache line should update the pattern in the AHT;
- A *Read Usage counter* to store the number of read accesses the cache line is predicted to receive before it becomes dead;
- An *Read Overflow bit* to indicate if the predicted number of read accesses exceeds the maximum value the *Read Usage counter* can hold. If set, the cache line remains powered until the line is evicted;
- A *Write Usage counter* to store the number of write accesses the cache line is predicted to receive before it gets evicted;
- An *Write Overflow bit* to indicate if the predicted number of write accesses exceeds the maximum value the *Write Usage counter* can hold. If set, the cache line is not be sent to early write-back;
- An *Early Evict flag* to indicate if the line should be queued to be evicted earlier after its last predicted access; and
- An *AHT Pointer* linking a cache line to its respective entry in the AHT.

The AHT is indexed by the program counter (PC) of the memory instruction that caused the cache miss and the requested cache line offset (byte within the line) of the address. The PC-offset combination has been shown to provide high accuracy and high coverage of patterns even with moderately sized AHTs [4] [9] [18].

An entry in the AHT includes:

- A *Pointer flag* indicating that a cache line has a pointer to that specific AHT entry;
- A *Read counter*;
- An *Read Overflow bit*;
- A *Write counter*;
- An *Write Overflow bit.*

The usage counters and overflow bits inside the AHT are identical to those in the cache line metadata.

B. Operations of DEWP

The main operations performed by DEWP are triggered by the following cache operations:

Cache Line Miss: The AHT is searched for an entry matching the PC and offset of the instruction that caused the miss. For an *AHT hit*, the mechanism copies the AHT's read/write counters and overflow bits into the cache metadata and resets the train and early evict flags. In the case of an *AHT miss*, the train flag is set, and all usage counters and overflow bits are reset in the cache metadata. The AHT resets the read/write counters and overflow bits and evicts the LRU entry to make room for the

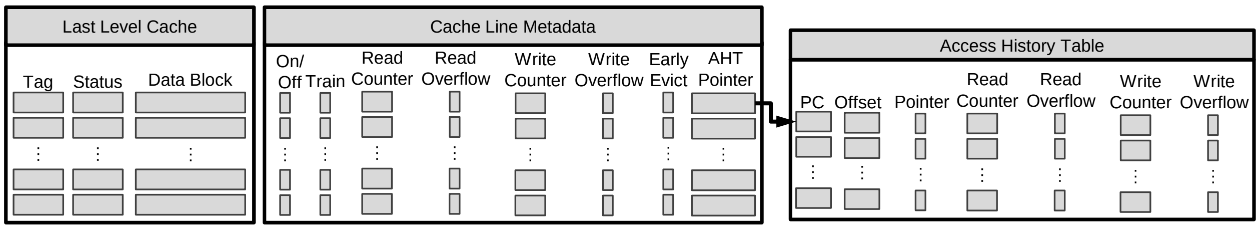

Fig. 4. Additions to the LLC required by DEWP.

new pattern (AHT line). An AHT pointer is created linking the cache metadata and the new entry. Because the train flag is set, future accesses to this line update the counters only in the AHT. In order to avoid multiple lines updating the same AHT entry, the point flag is used to inform if another cache line is already linked to that entry. In this case, the new link is not created.

Tag Hit and Data Turned On: If the train flag is disabled, the read usage counter in the metadata is decremented and the cache line is turned off if its read usage counter and overflow bit are zero. The AHT will only be updated when the train flag is enabled. The AHT to be updated is determined by the pointer in the metadata.

Tag Hit and Data Turned Off: The requested cache line is brought into the cache and its read overflow bit is set. If the cache metadata has a valid pointer to an AHT entry, the train flag is enabled and the mechanism increments the corresponding usage counter in the AHT entry.

Cache Line Eviction: If the cache line contains a valid link to an AHT pointer, the pointer flag must be disabled in the corresponding AHT. Also, if the read/write usage counters in the metadata is non-zero (indicating that the cache line was accessed less than the predicted number of times), the usage counter in the AHT entry is updated by decrementing the counter by the non-zero value.

Cache Line Invalidation: If the write usage counter is zero, it means that the last write was mispredicted. In this case, similar to the tag hit and data turned off case, the write overflow bit is set. If the cache metadata has a valid pointer to an AHT entry, the train flag is enabled, so future writes increment the write usage counter. Moreover, the cache line is turned off until it receives valid data.

Cache Line write-back: Similar to what happens during a tag hit and data turned on case, the write usage counter is decremented and the cache line is sent to early eviction in the case its write usage counter and overflow bit are zero. The AHT is only updated when the train flag is enabled. The AHT to be updated is determined by the pointer in the metadata. If the cache line was turned off, it is turned on again.

The proposed mechanism does not modify the coherence protocol at all. The protocol states are kept untouched even when the cache line is turned off. The tag store is always kept turned on.

C. *Augmenting the Cache Replacement Policy*

We also use our mechanism to improve the traditional LRU cache replacement policy by prioritizing lines that have the early evict flag set. This flag is set when the train flag is disabled and the read/write usage counters and overflow bits indicate that the cache line already received the predicted number of reads and writes.

Evicting dead lines early before they actually become victims (being at the LRU position, for example) can reduce the cache miss ratio by letting the alive lines stay longer in the cache [2].

Although we gave examples with the LRU replacement policy, other policies could also be easily modified to take advantage of our mechanism.

IV. Methodology

For our mechanism, experimental evaluation showed that using 2 bits in the read and write usage counter covers more than 95% of the LLC lines. This means that those lines receive less than 4 accesses before their eviction. In our experiments, we used 512 entries per AHT which proved to be enough to generate accurate results.

To maintain the metadata information, 2 bytes per cache line were added, which represents an overhead of 2.9% of the total cache size, assuming a tag size of 32 bits.

For the AHT, using only 16 bits to store the least significant part of the PC demonstrated to be enough to obtain accurate results. Moreover, since most of the accesses are aligned inside the cache line in sub-blocks of 8 bytes, only 3 bits are necessary to maintain the sub-block accessed inside the cache line (instead of using the full offset).

The total size of the AHT used in our experiments is 2.6 KB per LLC bank, which represents less than 0.25% of the total LLC size. Each AHT is organized as an 8 way set-associative cache in order to reduce the conflicts and increase the accuracy of the predictions.

A. *Simulation Environment*

We use an in-house cycle-accurate x86 processor simulator for our evaluation. Table I shows the baseline configuration for the processor, cache memory and the main memory system.

For our evaluation, we use a total of 43 benchmarks from 3 different suites: all (12 integer and 17 floating-point) from the SPEC-CPU2006 suite, (7 parallel) from the SPEC-OMP2001 suite, and (7 parallel) from the NAS-NPB-3.3.1 [3] suite. The SPEC-CPU2006 benchmarks were run using the reference input set, the SPEC-OMP2001 benchmarks were run using the Medium-ref input set and NBP-OMP using size the A input set. Each benchmark from SPEC-CPU2006 was

TABLE I. BASELINE SYSTEM CONFIGURATION.

OoO Execution Cores	2 GHz; 8 cores, in-order front-end and commit;
	14 stages (3-fetch, 3-decode, 3-rename, 2-dispatch, 3-commit);
	16 bytes fetch block size, fetch up to 6 instructions
	Decode and commit up to 5 instructions;
	Rename/dispatch/execute up to 5 micro instructions;
	18-entry fetch buffer, 28-entry decode buffer, 168-entry ROB;
	3-alu, 1-mul. and 1-div. integer units (1-3-20 cycle);
	1-alu, 1-mul. and 1-div. floating-point units (5-5-20 cycle);
	1-load and 1-store functional units (1-1 cycle);
	MOB entries: 64-read and 36-write;

Branch Predictor	1 branch per fetch; 8 parallel in-flight branches;
	4 K-entry 4-way set-associative, LRU policy BTB;
	Two-Level PAs predictor; 16 K-entry BHT, 2-bits prediction;

L1 Data Cache	32 KB, 8-way, 2-cycle; 64 bytes line size; LRU policy;
	MSHR entries: 4-request, 6-write-back, 2-prefetch;
	Stride Prefetcher: 2-degree, 64-strides table;

L1 Inst. Cache	32 KB, 8-way, 2-cycle; 64 bytes line size; LRU policy;
	MSHR entries: 4-request, 2-prefetch;
	Stride Prefetcher: 2-degree, 64-strides table;

L2 Cache	Private 256 KB, 8-way, 4-cycle; 64 bytes line size; LRU policy;
	MSHR entries: 8-request, 12-write-back, 4-prefetch;
	Stream Prefetcher: 2-degree, 16 prefetch distance, 128-streams;

L3 Cache	Shared 8 MB (8-banks), 1 MB per bank;
	16-way, 10-cycle; 64 bytes line size; LRU policy;
	Inclusive LLC; MOESI coherence protocol;
	MSHR entries: 32-request, 32-write-back;
	Bi-directional ring interconnection;

DRAM Controller and Bus	On-chip DRAM controller, Open-row first policy, 4-channels;
	8 DRAM banks per channel, 8 KB row buffer per bank;
	DDR3, 8 burst length at 2:1 frequency ratio;
	9-9-9-28 cycles CAS, RP, RCD and RAS latency;

run for a representative 200M instruction slice which was selected with Pinpoints [16]. The parallel benchmarks (SPEC-OMP2001 and NPB-OMP) where run with 8 threads, executing the parallel region from one time step of each benchmark. All benchmarks were compiled for x86-64, using GCC 4.6.3 or GFORTRAN 4.6.3 with the -O3 option.

B. Modeling Energy Consumption

In order to increase the energy efficiency of the LLC, we turn off the data array part of the cache line using gated V_{DD} circuit techniques, as in [17]. Gated V_{DD} techniques use a transistor to gate the supply voltage (V_{DD}) of the cache SRAM cells.

In order to model the static energy savings using the DEWP predictor, we model both the baseline cache architecture and our proposed mechanism with CACTI 6.5 [15] at 32 nm technology. We model tag and data power consumption.

Since our proposed mechanism requires extra metadata, the cache lines were also modeled with the extra bits necessary. We also consider that the metadata and the tag array are always turned on. The additional energy consumption of the AHTs is not modeled due to its negligible impact.

V. EVALUATION

A. Prediction Accuracy

To analyze the accuracy of our mechanism, every time an LLC line is invalidated, due to a processor write operation or a cache eviction, we classify the line as: training, overprediction, correct prediction and underprediction.

The training classification corresponds to the lines used to train a new pattern. The overprediction refers to the case that the cache line could be turned off/written back earlier. The correct prediction means that the line was correctly turned off/written back. The underprediction means that the cache line was read after predicted to be dead or it received a write after it was written back.

Notice that under predictions can hurt the performance, by early evicting alive lines and thus generating extra cache misses for those lines that have a clean copy of the data, and also generate extra write-backs for dirty lines.

Figure 5 presents the accuracy results for our mechanism. They show that DEWP requires an average of 1% to train the mechanism, overpredicts 3%, underpredicts 2% and correctly predicts 94%.

B. Static Energy

The cache energy efficiency is increased by using our mechanism to turn off dead and invalid lines. Moreover, our mechanism will implicitly cause further energy savings by reducing the execution time of the applications.

The results in Figure 6 are shown in terms of static energy savings at the LLC normalized to the baseline using an inclusive cache hierarchy. DEWP achieves on average a 61% energy savings compared to the baseline.

Figure 7 presents the results for a non-inclusive cache using our predictor. For most of the benchmarks, the energy savings achieved are very similar to the inclusive LLC results. Comparing the results achieved by the mechanism to the theoretical results in Section II, we find that in most of the cases our energy savings are not as high as the perfect mechanism suggests. However, the energy savings was considerable for all benchmarks, reaching up to 91% in the case of *namd*, with the smallest saving for *sphinx3* at about 22%.

C. Performance Evaluation

Our mechanism can influence the execution time of an application in different ways. DEWP can increase the performance with early evictions of dead lines, enabling more virtual space in the LLC cache, while early write-backs of last written lines can potentially reduce the memory controller contention. On the other hand, our mechanism can hurt the performance by causing extra cache misses because of underpredictions.

The performance influence of our mechanism can also vary if the LLC is inclusive or non-inclusive. This happens because, whenever our mechanism predicts a cache line as dead and prioritizes that line for early eviction, if the LLC is inclusive, the early eviction will evict the line from the other cache levels of the system as well. On the other hand, if the LLC has a non-inclusive policy, the early eviction will only affect the LLC. Furthermore, extra cache misses caused by underpredictions of our mechanism would be alleviated in a non-inclusive LLC because the data might still be available in other cache memories (L1 and L2).

Figure 8 shows the execution time of our mechanism normalized to the baseline using an inclusive LLC configuration. We can observe that in most of the cases, the performance

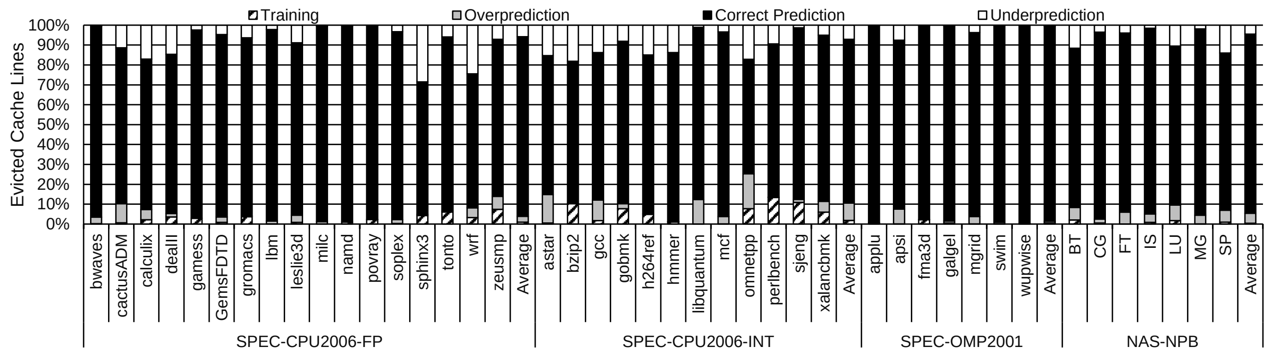

Fig. 5. Accuracy results for the DEWP mechanism.

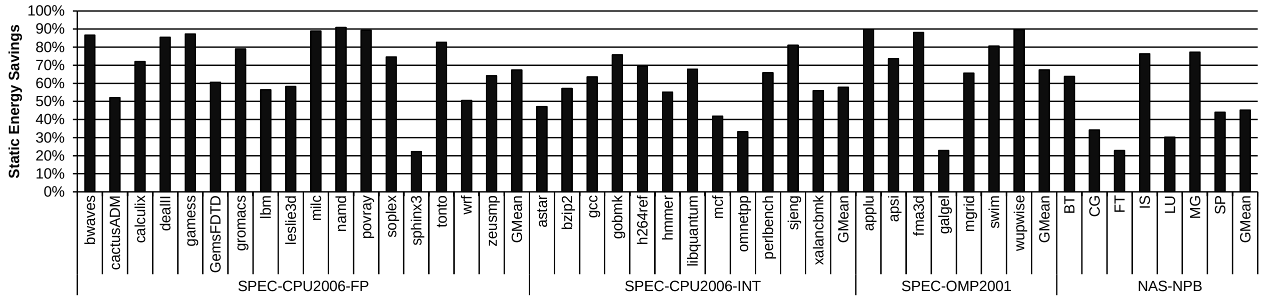

Fig. 6. Energy savings using DEWP in inclusive LLC.

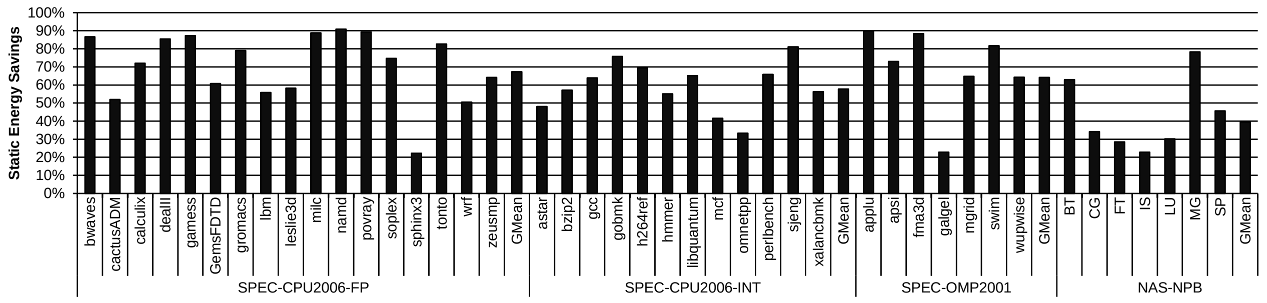

Fig. 7. Energy savings using DEWP in non-inclusive LLC.

gains correlate with the sensitivity study presented in Section II and the prediction accuracy results. However, some benchmarks, such as *swim* and *sp*, had a performance degradation, because our predictor failed to recognize some cache access patterns.

The results shown in Figure 9 are relative to our mechanism and the baseline running with a non-inclusive LLC. For some benchmarks, such as *applu*, *wupwise* and *IS*, the relative gains of our mechanism are less than the gains with an inclusive LLC. This is because the non-inclusive baseline achieved a higher performance for these applications, due to the larger effective cache size.

On average, the performance was improved by 2%, which shows that despite saving a lot of energy, the performance was not negatively impacted.

VI. Related Work

The following subsections describe the most significant work in cache line behavior prediction.

A. Line Usage Predictors

Chen et al. [4] proposed a Spatial Pattern Predictor (SPP) to predict cache line usage patterns. The mechanism uses the program counter (PC) and the referenced data offset to correlate historical data about line usage in order to predict future usage patterns of L1 cache lines. The goal of this technique is to reduce leakage energy by bringing into the cache just those sectors predicted to be useful. The authors also introduce a prefetching technique to bring only the predicted spatial patterns for contiguous groups of up to 512 bytes. Lee et al. [12] propose to perform an early write-back operation of cache lines that reach the LRU position, while DEWP uses a predictor that detects the last write operation much earlier. Also, their evaluation is limited to single threaded benchmarks, while we broadly study the impact of our predictor for single and multithreaded workloads.

B. Counter Based Dead Line Predictor

Kharbutil et al. [8] present two counter-based mechanisms (AIP and LvP). The paper indicates that the Live-time Predic-

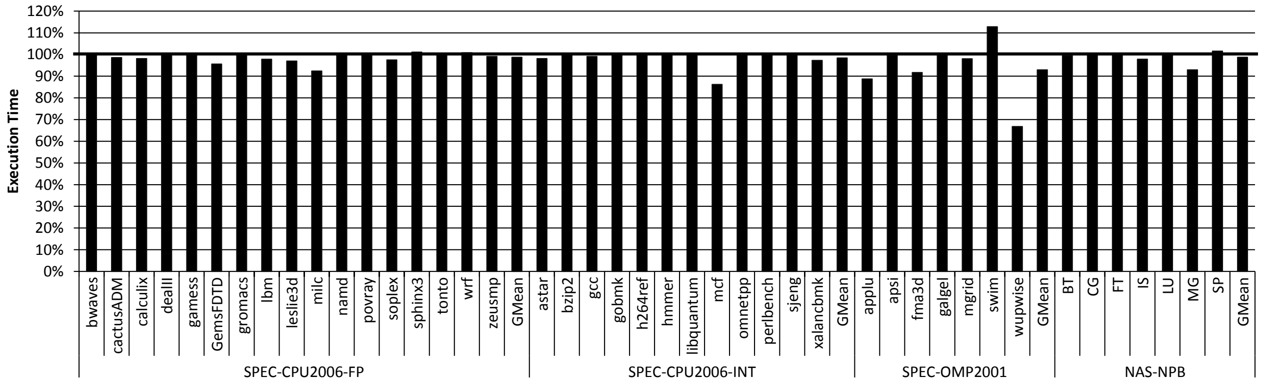

Fig. 8. Performance using DEWP in inclusive LLC

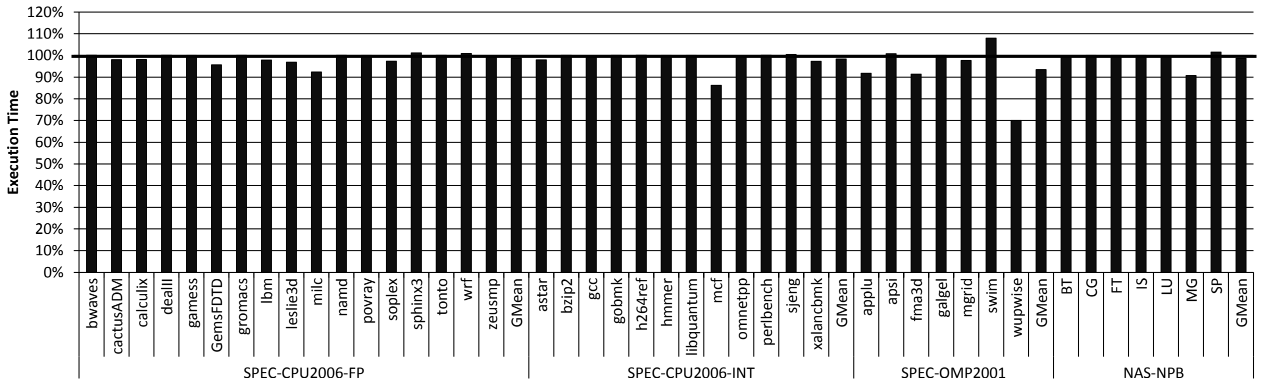

Fig. 9. Performance using DEWP in non-inclusive LLC

tor (LvP) delivers higher accuracy with less complexity. LvP records the number of accesses to a cache line and predicts the line as dead when the access counter reaches a certain threshold. The mechanism uses a hash of the PC which caused the cache miss to index into a table that stores the history of the number of accesses to previously evicted lines. The mechanism is used to identify dead lines early, and also to bypass dead-on-arrival cache lines.

C. Trace Based Dead Line Predictors

Lai et al. [10] [11] introduce the Last-Touch Predictor (LTP) which uses an execution trace to predict the last touch to a cache line. The mechanism generates a signature based on a trace of instructions that access a cache line. By matching the current signature with previously stored signatures that lead to dead cache lines, the mechanism can predict when a given line becomes dead. The goal of this work is to allow the lines to self-invalidate when their last access is detected.

Kahn et al. [7] propose a Skewed Dead Block Predictor (SDP) to predict dead lines and use these lines as a virtual victim cache. This skewed predictor is very similar to the LTP mechanism but uses two global tables indexed by different hash functions to reduce the impact of conflicts between them.

D. Time Based Dead Line Predictors

Kaxiras et al. [6] present a cache decay mechanism which that uses theories from competitive algorithms to create a time-based strategy. They exploit long dead periods by turning off cache lines during such periods. This approach aims to reduce leakage power dissipated by the cache. Once the algorithm indicates that a decay interval on the order of thousands of cycles arrives, a hierarchical counter mechanism is adopted to reduce the bits required for the counters per cache line.

Abella et al. [1] introduce the Inter- Access Time per Access Count (IATAC) mechanism to predict and turn off dead lines with the objective of reducing L2 cache leakage energy. This mechanism predicts a cache line to be dead when it detects that the line has not received any accesses for a period greater than the average time between different accesses. The mechanism keeps track of the average time between accesses in a global table. Our mechanism does not require the broadcast signals from all the cache lines to detect dead lines, and they only predict last read operations.

E. Last Write Predictor

Wang et al. [19] propose a Last Write Predictor (LWP) to predict whenever the cache line receives its last write. The prediction mechanism uses three tables with a skewed organization similar to the SDP mechanism to detect the last-written blocks and store pointers of these blocks into a last-write buffer. The objective of this mechanism is to make the last-write blocks available for the main memory scheduling before the line gets evicted. They require a complex internal simulator while our predictor is much simpler and cheaper in terms of storage/area.

None of the previous approaches take into account that dirty lines remain turned on for long periods of time, wasting energy while these lines could be evicted early. Thereby, energy can be saved and memory contention reduced. Our work introduces a mechanism that performs the prediction of last read, last write and last access on a cache line basis, exploring the energy savings achievable by turning off invalid and dead lines, and performing early write-backs.

VII. Conclusions

In this paper, we introduced the DEWP mechanism to optimize the energy efficiency by keeping only alive data in the LLC. Our mechanism achieves this by predicting access patterns of the cache lines. Using this information, DEWP is able to turn off the cache lines as soon as their data becomes dead, to write-back early dirty cache lines after their last write operation happens and also to reduce cache pollution by prioritizing the eviction of completely dead cache lines. DEWP works independently of the cache replacement algorithm and it does not modify the cache coherence protocol.

The DEWP mechanism requires a low storage size overhead to achieve accurate predictions (94% correct predictions and only 2% of under predictions). DEWP achieves a 61% energy reduction on average compared to the baseline. The execution time is reduced by 2% on average for single-threaded and multi-threaded applications. DEWP saves 74% of the potential savings that a perfect (oracle) mechanism would achieve. Additionally, DEWP achieves very similar results in terms of energy and performance for inclusive and non-inclusive LLCs.

References

[1] J. Abella, A. González, X. Vera, and M. F. P. O'Boyle. Iatac: a smart predictor to turn-off l 2 cache lines. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2(1):55–77, 2005.

[2] M. Alves, E. Ebrahimi, V. Narasiman, C. Villavieja, P. Navaux, Y. Patt, et al. Energy savings via dead sub-block prediction. In *Proc. IEEE Int. Symp. on Computer Architecture and High Performance Computing*, SBAC-PAD'12, pages 51–58. IEEE, 2012.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. Technical report, The International Journal of Supercomputer Applications, 1991.

[4] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proc. IEEE Int. Symp. on High Performance Computer Architecture (HPCA '04)*, pages 276–287. IEEE, 2004.

[5] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[6] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *Proc. IEEE/ACM Int. Symp. on Computer Architecture (ISCA '01)*, pages 240–251. IEEE, 2001.

[7] S. Khan, D. A. Jiménez, D. Burger, and B. Falsafi. Using dead blocks as a virtual victim cache. In *Proc. IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques (PACT '10)*, pages 489–500. ACM, 2010.

[8] M. Kharbutil and Y. Solihin. Counter-based cache replacement and bypassing algorithms. *IEEE Transactions on Computers (TC)*, 57(4):433–447, 2008.

[9] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *ACM SIGARCH Computer Architecture News*, volume 26, pages 357–368. IEEE, 1998.

[10] A.-C. Lai and B. Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. In *Proc. IEEE/ACM Int. Symp. on Computer Architecture (ISCA '00)*, pages 139–148. IEEE, 2000.

[11] A.-C. Lai, C. Fide, and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proc. IEEE/ACM Int. Symp. on Computer Architecture (ISCA '01)*, pages 144–154. IEEE, 2001.

[12] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager writeback - a technique for improving bandwidth utilization. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, MICRO 33, pages 11–21, New York, NY, USA, 2000. ACM.

[13] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. IEEE/ACM Int. Symp. on Microarchitecture (MICRO '09)*, pages 469–480. IEEE, 2009.

[14] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 469–480. ACM, 2009.

[15] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Architecting efficient interconnects for large caches with cacti 6.0. *IEEE Micro (IEEE MICRO)*, 28(1):69–79, 2008.

[16] H. Patil, R. S. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large intel® itanium® programs with dynamic instrumentation. In *MICRO*, pages 81–92, 2004.

[17] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-v_{DD}: a circuit technique to reduce leakage in deep-submicron cache memories. In *Proc. ACM Int. Symp. on Low Power Electronics and Design (ISLPED '00)*, pages 90–95. ACM, 2000.

[18] P. Pujara and A. Aggarwal. Cache noise prediction. *IEEE Transactions on Computers (TC)*, 57(10):1372–1386, 2008.

[19] Z. Wang, S. M. Khan, and D. A. Jiménez. Improving writeback efficiency with decoupled last-write prediction. In *ISCA*, pages 309–320, 2012.