Cooperatively Managing Dynamic Writeback and Insertion Policies in a Last-level DRAM Cache

Shouyi Yin*, Jiakun Li*, Leibo Liu*, Shaojun Wei*, Yike Guo†
*Institute of Microelectronics, Tsinghua University, Beijing, 100084, China
†Department of Computing, Imperial College, London, SW7 2AZ, UK

***Abstract*—Stacked-DRAM used as the last-level caches(LLC) in multi-core systems delivers performance enhancement due to its capacity benefit. While the performance of LLC depends heavily upon its block replacement policy, conventional replacement policy needs redesigning to exploit the best of DRAM cache while avoiding its drawbacks. Existing DRAM cache insertion policy blindly forwards victim lines to the off-chip memory, regardless of the potential for increased hits by placing a fraction of them in the DRAM cache; nevertheless, a naïve design that steers all dirty victims to the DRAM cache introduces excessive writeback traffic which aggravates capacity misses and DRAM interference. To leverage insertions in terms of writeback or fill requests, we propose a cooperative writeback and insertion policy that adapts to the distinct access patterns of heterogeneous applications based on runtime misses and writeback efficiency, thereby increasing HMIPC (harmonic instruction per cycle) throughput by 22.2%, 13.7% and 14.5% compared to LRU and two static writeback policies.**

I. Introduction

The Memory Wall, which refers to the speed gap between data processing and data fetching, has been exacerbated by the trend towards integrating many cores on chip. [1] In response to memory pressure, the multi-core system utilizes a large, shared Last-level Cache (LLC) to enable flexible allocation of cache resource among cores and exploit the temporal locality of multiple applications. In this scenario die-stacked DRAM, which offers caches of tens or low hundreds of megabytes at twice the bandwidth of off-chip memory and less than half of the memory latency [2], [3], serves as a promising alternative to standard SRAM due to its capacity benefits [4], [5].

When multiple applications are executing in parallel, their requests to the shared cache may lead to *DRAM interference* [6]–[8] and *inter-core contention* [6], [9]–[11], increasing hit latencies and decreasing hit rate in an unpredictable way. DRAM interference can be caused either by the interleaving of cache requests from multiple applications or the competition for available bandwidth between non-critical commands (writebacks) and critical ones (read/write). Inter-core contention happens when the combined working set exceeds the cache size so that the incoming lines of one application evict the blocks of another.

This work was supported in part by the National NSFC grant (No.61274131), the International S&T Cooperation Project of China Grant (No. 2012DFA11170), the Tsinghua Indigenous Research Project (No.20111080997), the China National High Technologies Research Program (No. 2012AA012701 and 2012AA010904) and the National Science and Techonology Major Project (No.2013ZX01033001-001-003).

To mitigate DRAM interference and inter-core contention, several block replacement policies targeting DRAM caches are proposed. The Adaptive Multi-Queue Policy (*AMQ-policy*) [12] and the Utility-Based Cache Partitioning (*UCP-policy*) [13] dynamically partition cache sets across cores, whereas the former requires non-trivial modification to the existing scheme and the latter conducts unnecessary fill cache requests, which disruptively interact with necessary ones and aggravate DRAM interference. The Adaptive DRAM Placement Policy (*ADP-policy*) [6] fills the DRAM cache at an adaptive probability determined by the runtime misses of sampling sets. However, it indiscriminately steers all L3 writeback misses to main memory, unaware of the fact that the most frequently evicted L3 blocks are also the most likely to be found as misses in L3 cache and they can be serviced faster if placed in the DRAM cache.

We argue that writeback and insertion policies should be collaboratively optimized in an effort to enhance DRAM cache performance. In this paper, we propose a Cooperative Writeback and Insertion Policy (*CWIP*) which simultaneously manages the writeback and insertion policies, while the existing ADP-policy only handles insertion requests from main memory. We introduce the Integrated Set Monitors (*ISM*) that tracks the runtime misses under different combinations of writeback and fill policies and determines the current policy for each workload based on this information. However, the ISM can be misled by non-uniform access patterns across cores and incur excessive writeback traffic to the DRAM cache. So we propose the Lose to Gain Dispatcher (*LGD*) on top of ISM that estimates the writeback accuracy on a per-core basis and reduces the cache resources allocated to futile writebacks. LGD assists with the decision-making of the ISM and alleviates the DRAM interference caused by the interleaving of critical and noncritical requests. We compare our CWIP with two static writeback policies, Never Writeback Dynamic Insertion Policy (NW-DIP) and Always Writeback Dynamic Insertion Policy (AW-DIP) and demonstrate CWIP outperforms both.

II. Motivation

Traditional cache architecture needs redesigning to maximize DRAM cache capacity benefits while minimizing the negative aspects of its heavy latency. State-of-the-art DRAM cache employs a low-overhead SRAM-based structure namely *MissMap* [5], [6], [9], [14] to answer block membership

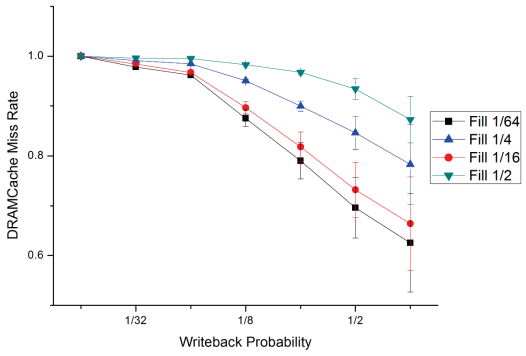


Fig. 1. DRAM cache miss rates under static fill and writeback probabilities

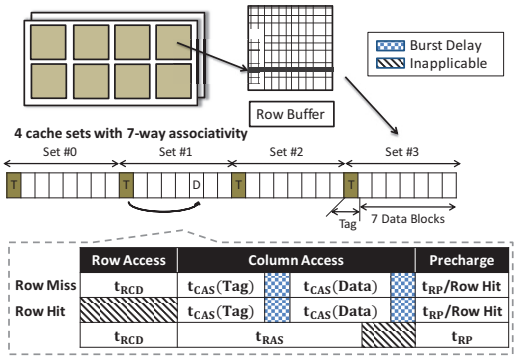


Fig. 2. A typical DRAM cache of 7-way associativity

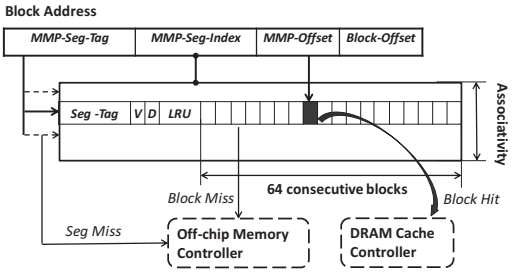


Fig. 3. MissMap structure

queries (i.e. Is block X in DRAM cache?) before real cache accesses (details in Sec.III B). MissMap accurately tracks the residence information of all data blocks in the DRAM cache so that pre-checking MissMap can eliminate any cache accesses on misses.

The MissMap design opens new opportunity for handling writeback. A writeback command is issued to the lower-level cache when a dirty(modified) block is victimized from the upper-level cache. If the writeback hits, its lower-level copy must be updated to guarantee consistency; otherwise a later request to the stale data can result in incorrect execution [15], [16]. But an accurate predictor MissMap allows the freedom to choose whether to conduct writeback to the DRAM cache when it is identified as a miss in advance. The ADP-policy [6] only writes back to the DRAM cache when MissMap indicates cache hit and forwards all writeback misses to the off-chip memory. We argue that 1) writing back blocks at misses has a distinguishable advantage in improving cache hit rate and 2) the optimal performance arises from the cooperative management of fill and writebacks.

Fig.1 shows the average DRAM cache miss rates at different fill cache probability when writeback probability (at misses) varies. (parameters listed in TableI,II). All results are normalized to the case of Never Writing back (NW) at misses. It indicates that writing back 50% dirty lines at misses reduces the demand miss rates by 6.6%, 21.7%, 26.8% and 30.4% at a fill probability of 1/64, 1/16, 1/4 and 1/2. A more aggressive approach writing back all evicted lines cuts down miss rates further by 13.6%, 27.7%, 33.1% and 36.5%.

However, Always Writing back (AW) at misses, though improving DRAM cache hit rate, can degrade system performance via increased hit latencies since it introduces excessive L3-L4 writeback traffic interfering with critical requests. Besides, a large number of futile blocks written back into the DRAM cache can evict useful blocks that would be demanded in near future. To overcome the problems, our CWIP seeks to strike a balance between NW and AW through dynamically managing writeback policies. We compare our results with two static writeback policies in Sec.VI and demonstrate CWIP outperforms both.

III. Background

A. DRAM Cache Organization

A DRAM cache is composed of multiple banks where each bank contains multiple rows and owns a row buffer. The common practice of DRAM cache uses a Tag-In-DRAM approach. It arranges the tag and data alongside in a same row of DRAM array cells, so as to reduce hit latency through combining tag lookup and data fetching in one row access. [5], [14] This paper assumes an 8-bank DRAM cache of 2KB row size. A row is divided into 4 cache sets where each contains 1 tag block and 7 data blocks, as is shown in Fig.2. [17] After the DRAM cache controller receives a request, it issues a command for row activation (t_{RCD}) to load a whole row into the row buffer and two following commands for column activation (t_{CAS}) to access the tag and data blocks. If the subsequent request hits in the same row, the row buffer is preserved (so called a row hit); otherwise all contents in the row buffer are written back into DRAM bitcell array by a precharge command (t_{RP}) and a new cycle begins.

B. MissMap Structure

State-of-the-art DRAM cache architecture also uses an SRAM-based structure MissMap (MMP) [5], [6], [14] to reduce miss penalty by eschewing cache accesses on misses. The MMP structure is illustrated in Fig.3. One MMP entry contains a bit-vector accurately representing the presence of consecutive 64 blocks, a tag for the tracked memory segment, a valid bit, a dirty bit, 4-bit Least-Recently Used (LRU) information. Each time a block is inserted into the DRAM

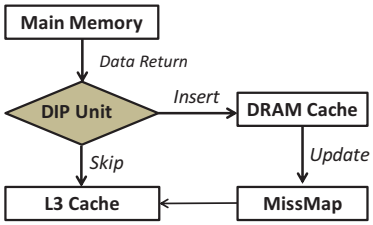


Fig. 4. The decision flow chart of DRAM cache DIP

cache, its corresponding bit in MissMap will be set. The bit will be cleared after its block is invalidated. The data request is only sent to the DRAM cache controller after MissMap reports a hit; otherwise it is forwarded to the main memory, making a miss faster.

C. Dynamic Insertion Policy for DRAM cache

An application is called thrashing when its data reuse distance exceeds cache associativity so that its lines are always evicted before re-referenced. The multi-cores employing LRU may cause the applications with large working sets to exhibit thrashing behavior because LRU implicitly allocates cache resources based on the rate of demand, unaware of the distinct cache requirements of heterogeneous workloads.

Previous studies proposed several thrashing-resistant block replacement policies for SRAM caches. The Bimodal Insertion Policy (BIP) [18] inserts the majority of incoming lines at the LRU side and few at the MRU side; thus securing hits to a fraction of working set retained in cache. The Dynamic Insertion Policy (DIP) [18] dedicates a few cache sets to LRU or BIP and uses the policy incurring fewer misses for the rest.

Fig.4 illustrates the decision flow chart of DRAM cache DIP [6], which is similar to conventional DIP except that it makes the decision of inserting or bypassing the cache instead of where to insert in the LRU stack. This is because DRAM accesses are slow and unnecessary accesses add delay to the critical requests. The DRAM cache DIP decreases hit latency of non-thrashing applications by reducing the intensity of fill requests from thrashing ones.

IV. Cooperative Writeback and Insertion Policy

Additional writebacks to the DRAM cache at misses improves hit rate but competes with the servicing of demand hits. Our Cooperative Writeback and Insertion Policy (CWIP) attempts to manage the writeback and fill policies simultaneously, which dynamically adapts to the phasic change of a single application and the distinct access patterns of a workload mix.

A. Integrated Set Monitors

The Integrated Set Monitors (ISMs) adaptively change the combination of writeback and fill policies based on set dueling, shown in Fig.6. It counts the misses in a fraction of sets dedicated to using fixed policies, annotated with Writeback/Fill Monitors (WBM/FM) and applies the policy combination that incurs fewest misses to the rest. W_i/P_i denotes the

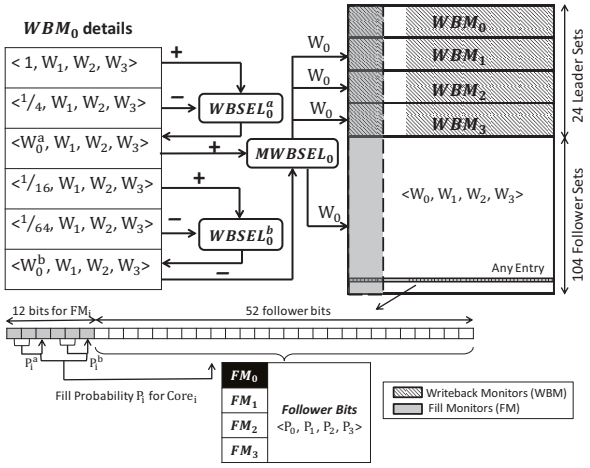


Fig. 5. The Integrated Set Monitors

writeback/fill probability for blocks from $core_i$, chosen from four candidates, 1, 1/4, 1/16 and 1/64. WBMs account for 24 out of 128 MissMap sets and the rest are followers. WBM_i determines the writeback probability for $core_i$ with other cores using their current policies. [19] A miss detected in the set employing $<1, W_1, W_2, W_3>$ increments the 10-bit saturating counter $WBSEL_0^a$ and a miss in that of $<1/4, W_1, W_2, W_3>$ decrements $WBSEL_0^a$. The most significant bit of $WBSEL_0^a$ decides the better policy from 1 and 1/4 as W_0^a and applies it to the third set. W_0^b is obtained from 1/16 and 1/64 through the same approach. $MWBSEL_0$, which is incremented or decremented depending on the misses in the third set or fifth set, determines the winner from W_0^a and W_0^b and applies it to $WBM_{j\neq 0}$ and followers. To realize writing back at different probabilities, we use an 8-bit Linear Feedback Shift Register to generate a pseudo-random number and compare it to four threshold values (5, 16, 64, 256 for 1/64, 1/16, 1/4, 1, respectively).

The fill policy selection executes in a similar way except it groups the first 12 bits of a MissMap entry as samplers and the rest as followers. The MissMap sets are dedicated to $core_0$, $core_1$, $core_2$, $core_3$ in a cyclic pattern. We adopt an integrated sampling method that the result of writeback/fill policy selection counters is directly used by the monitors for the other to incorporate feedback between two parties.

B. Lose to Gain Dispatcher

Although ISM enables the cooperative management of fill and writeback, it may lead to misprediction for only considering one parameter runtime misses. Imagine two applications A and B. Application A writes back a large number of dirty blocks into the DRAM cache and frequently requests these blocks, while B writes back a medium number of blocks but seldom demands its blocks. In this case, the cache resource consumed by unnecessary writebacks of B should be reallocated to application A; nevertheless, the ISM mechanism would report more misses of application A and reduce its

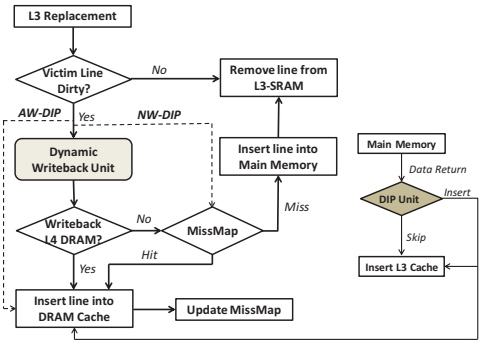


Fig. 6. The decision flow chart of NW-DIP, AW-DIP and CWIP

cache resource via lowering its writeback probability. Furthermore, ISM is designed to remove a portion of writebacks that victimize useful blocks but cannot identify futile writeback blocks that are never demanded. These blocks, though not contending for cache capacity, interleaves with data requests and increases hit latencies.

Our Lose to Gain Dispatcher (LGD) aims to correct the possible misjudgment of ISM by initiatively redirecting useless writebacks to off-chip memory. The efficiency of writeback is described with E_{WB}, the ratio between the references to blocks inserted by writeback and the amount of writeback blocks, see (1).

$$E_{WB} = \frac{The\ number\ of\ references\ to\ writeback\ blocks}{The\ number\ of\ writeback\ blocks} \quad (1)$$

Note that E_{WB} includes repeated demands to the same block. Though it is possible to exclude repeated references, the latter approach needs to modify the blocks on every access or add additional counters and the simulation result shows little speedup difference.

To implement LGD, each core maintains two counters. The first counter wb_ref_i counts the number of demand hits to the writeback blocks from $core_i$. The second counter wb_total_i tracks the number of total blocks from $core_i$ written back into DRAM cache. The counters are refreshed every 64 million of cycles to adapt to phasic change of the applications. If wb_total_i reaches a threshold and wb_ref_i is smaller than a given value, the owner application is identified as a writeback-futile one and a signal $sacrifice_i$ is sent to ISM to minimize its writeback probability to 1/128 and reserves its writeback policy selection counters until next refresh cycle begins. Each cache block is extended with 1-bit *WbFlag* which identifies whether it is inserted into the DRAM cache through writeback.

C. Implementation of CWIP policy

Fig.6 compares the writeback schemes of Never Writeback Dynamic Insertion Policy (NW-DIP), Always Writeback Dynamic Insertion Policy (AW-DIP) and our CWIP. AW-DIP and CWIP are initialized with low fill probability to trigger writeback streams and the same is applied to NW-DIP to enable direct comparison among three. The Dynamic Writeback Unit (*DWU*), composed of ISM and LGD, is not invoked until the victim lines of the DRAM cache exceeds a threshold (*8191 per core* in the experiment), which indicates a full cache. Then the DWU forwards every L3 writeback request to either the DRAM cache controller or MissMap. If the block address hits in the MissMap, it must be updated in the DRAM cache to maintain consistency.

Algorithm 1 Lose to Gain Dispatcher

```
// L3-SRAM evicts a dirty line B of workload i
// W_i: The threshold for current writeback policy
if sacrifice_i then
    Minimize W_i;
    Reserve policy selection counters WBSEL_i^a, WBSEL_i^b, MWBSEL_i until next refresh cycle begins;
else
    Choose the current policy W_i from four candidates based on the Integrated Set Monitors;
end if
Generate an 8-bit random number R;
if R < W_i then
    Writeback B to the DRAM cache;
    wb_total_i ++ ;
    if wb_total_i > 4096 && wb_ref_i is small then
        sacrifice_i ← True.
    end if
else if B hits in MissMap then
    Writeback B to the DRAM cache;
else
    Writeback B to the main memory;
end if
// Or a demand hit in the DRAM cache
if the requested block has a true WbFlag then
    i ← block's core ID;
    wb_ref_i ++;
end if
```

Our CWIP policy comes at negligible hardware overhead. The ISM needs six multiplexers, six 10-bit policy selection counters per core and two LFSRs (each requires 8 XOR gates and 8 flip flops). The LGD needs two 13-bit counters (wb_total_i and wb_ref_i), 2 comparators per core. The invoking unit needs additional four 13-bit counters and four comparators.

V. Experimental Results

We use the gem5 simulator to model a quad-core system of 3-level SRAM caches and a DRAM LLC, with parameters listed in Table I. We assume the stacked DRAM timing latencies are approximately half of that of main memory in accord with [5], [6], [12]. The system performance is evaluated on various workload mixes from SPEC2006 benchmarks, listed in Table II.

A. System Performance

Fig.7 shows the performance improvement of NW-DIP, AW-DIP, CWIP without LGD correction and full CWIP over the

TABLE I
SYSTEM PARAMETERS

CPU	
Core	4-core, ARM out-of-order, 3.2GHz
SRAM Cache	
L1 Cache	32KB, 8-way, 2 cycles, private
L2 Cache	256KB, 8-way, 5 cycles, private
L3 Cache	4MB, 16-way, 15 cycles, shared
DRAM Cache	
MissMap	2MB, 16-way, 8192 sets
DRAM Cache Bus	1.6 GHz, 128 bits per channel
DRAM Cache Size	64MB, 8-bank, 2048 bytes per row
tCAS-tRCD-tRP-tRAS	10-10-10-25
Main Memory	
Main Memory Bus	800MHz, 64 bits per channel
Banks-Ranks-Channel	8-1-1
tCAS-tRCD-tRP-tRAS	11-11-11-28

TABLE II
WORKLOAD MIX

Name	Benchmarks	Name	Benchmarks
MIX_01	lbm-milc.r-libquantum-bzip2	MIX_07	astar.r-milc.r-mcf-lbm
MIX_02	bzip2-leslie3d.t-astar.r-lbm	MIX_08	astar.r-leslie3d.t-lbm-milc.r
MIX_03	zeusmp-hmmer-lbm-leslie3d.t	MIX_09	lbm-astar.t-zeusmp-gobmk
MIX_04	bzip2-leslie3d.r-libquantum-mcf	MIX_10	lbm-milc.r-bzip2-soplex.r
MIX_05	mcf-milc.r-gobmk-lbm	MIX_11	omnetpp-astar.r-lbm-libquantum
MIX_06	milc.r-leslie3d.r-leslie3d.t-lbm		

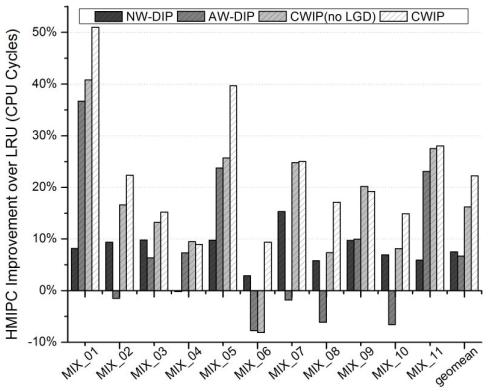
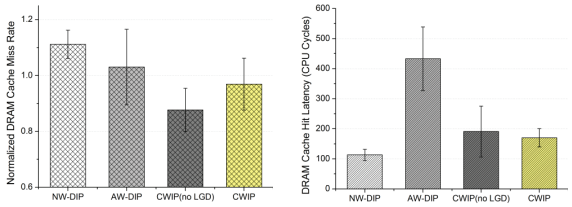
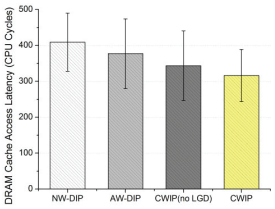


Fig. 7. Normalized HMIPC throughput improvement over LRU of NW-DIP, AW-DIP, CWIP(no LGD) and full CWIP



(a) Normalized DRAM cache miss rate relative to LRU

(b) DRAM cache hit latency



(c) DRAM cache access latency

Fig. 8. DRAM cache utility of NW-DIP, AW-DIP, CWIP(no LGD) and full CWIP

baseline LRU using the metric normalized Harmonic Instruction per Cycle (HMIPC) for multi-programmed workloads. The bar labeled *geomean* represents the geometric mean of the 11 workload mixes. In some scenarios (i.e. MIX_01, MIX_04, MIX_11), AW-DIP performs better than NW-DIP, for the blocks written back into the DRAM cache add to demand hits. But in some other mixes AW-DIP delivers performance degradation (i.e. MIX_06, MIX_08, MIX_10) because additional dirty blocks from L3 contend for DRAM cache resource and give rise to thrashing behavior, or because the heavy L3-L4 writeback traffic exerts pressure on cache bandwidth and delays data requests. We first evaluate CWIP without LGD, where ISM mechanism executes alone. It shows a performance benefit over LRU, NW-DIP and AW-DIP of 16.2%, 8.08% and 8.90%, which is primarily due to its attempt to minimize cache misses through cooperative management of writeback and fill probabilities. On top of ISM we implement LGD to leverage writeback traffic to DRAM cache and off-chip memory. On average, CWIP increases the HMIPC throughput by 22.2%, 13.7% and 14.5% compared to LRU, NW-DIP and AW-DIP.

B. Miss rate, hit latency and access latency

To get a better insight on the effects of using CWIP, Fig. 8(a) and Fig. 8(b) illustrates the utility of the DRAM cache from its miss rate and hit latency. The average cache access latency(T_{acc}) is determined by the cache hit latency(T_{hit}), miss penalty(T_{miss}) and miss rate(R_{miss}), as shown in (2).

$$T_{acc} = T_{hit} \times (1 - R_{miss}) + T_{miss} \times R_{miss} \qquad (2)$$

NW-DIP suffers from high miss rates for it steers all dirty blocks victimized from L3 to main memory, keeping future requests from accessing them within on-chip latencies. AW-DIP suffers from heavy hit latencies for it triggers substantial L3-L4 writeback traffic which exhausts DRAM cache bandwidth while leaving off-chip bandwidth idle. Our first design CWIP (no LGD), where ISM works alone, reduces miss rate through adopting the policy combination that incurs fewest misses. On average it decreases miss rate by 21.1% and 14.9% compared to NW-DIP and AW-DIP. Our second innovation LGD relieves pressure on cache bandwidth and reduces hit latencies at the expense of increased miss rate. On average, the proposed mechanisms (ISM + LGD) provide 22.7% and 16.1% access latency reduction compared to NW-DIP and AW-DIP, see Fig.8(c).

C. Cooperation between fill and writeback

CWIP leverages DRAM cache insertions in terms of fill or writeback, hence realizing the cooperation. Fig. 9 shows the fill/writeback accuracy, indicated by the ratios of demanded blocks to total blocks inserted via fill/writeback (writeback is

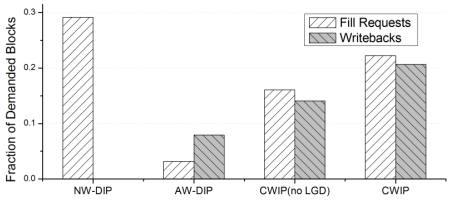


Fig. 9. Fill and writeback accuracy of DRAM cache

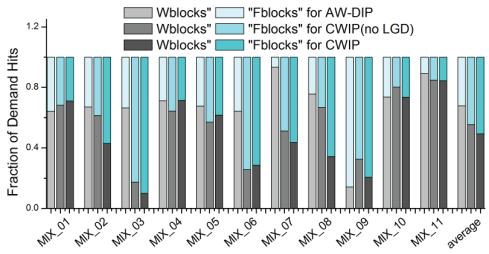


Fig. 10. Distribution of DRAM cache demand hits

inapplicable in NW-DIP). NW-DIP achieves the highest fill accuracy for its DIP unit eschews writing back the DRAM cache at misses and only optimizes for fill requests. AW-DIP causes significant accuracy degradation due to its uncontrolled writeback. In contrast, our proposed ISM and LGD improves the fraction of beneficial insertions by collaboratively managing the fill and writeback policies.

Fig. 10 illustrates the distribution of the DRAM cache hits. The gray bars represent the fraction of demand hits to "Wblock", which refers to the blocks inserted via writeback. The blue bars represent the fraction of hits to "Fblocks", which refers to the blocks inserted via fill requests on the response from memory. In MIX_01 and MIX_09, "Wblocks" account for a larger proportion of hits in CWIP compared to AW-DIP, while in some other cases (i.e. MIX_03, MIX_06, MIX_07), the fill requests from the off-chip memory contribute more to cache hits under the cooperation.

VI. Conclusion

While stacked-DRAM brings about large on-chip cache capacity, it requires novel replacement policy to exploit its utility. Existing DRAM cache replacement policy indiscriminately steers L3 writeback misses to the off-chip, leaving room for performance improvement since the victim blocks can be serviced within on-chip latencies if placed in the DRAM cache. However, blindly forwarding victim blocks to the DRAM cache compromises hit latencies via increased interleaving of cache requests. Our proposed scheme CWIP, composed of the Integrated Set Monitors and the Lose to Gain Dispatcher, cooperatively manages writeback and fill policies in order to mitigate inter-core contention and DRAM interference. We compare CWIP with two static writeback dynamic insertion policies. On average, CWIP achieves a speedup(in harmonic instruction per cycle) of 22.2%, 13.7% and 14.5% compared to LRU, NW-DIP and AW-DIP.

References

[1] D. Burger, J. R. Goodman, and A. Kägi, *Memory bandwidth limitations of future microprocessors*. ACM, 1996, vol. 24, no. 2.

[2] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *ACM SIGARCH computer architecture news*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 453–464.

[3] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. Lee, "An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–12.

[4] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ACM, 2013, pp. 404–415.

[5] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 454–464.

[6] F. Hameed, L. Bauer, and J. Henkel, "Adaptive cache management for a combined SRAM and DRAM cache hierarchy for multi-cores," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 77–82.

[7] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez, "Balancing dram locality and parallelism in shared memory cmp systems," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 2012, pp. 1–12.

[8] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 63–74.

[9] F. Hameed, L. Bauer, and J. Henkel, "Reducing inter-core cache contention with an adaptive bank mapping policy in dram cache," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*. IEEE, 2013, pp. 1–8.

[10] H. Dybdahl, P. Stenström, and L. Natvig, "A cache-partitioning aware replacement policy for chip multiprocessors," in *High Performance Computing-HiPC 2006*. Springer, 2006, pp. 22–34.

[11] K. Nikas, M. Horsnell, and J. Garside, "An adaptive bloom filter cache partitioning scheme for multicore architectures," in *Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on*. IEEE, 2008, pp. 25–32.

[12] G. H. Loh, "Extending the effectiveness of 3d-stacked dram caches with an adaptive multi-queue policy," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 201–212.

[13] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2006, pp. 423–432.

[14] G. H. Loh and M. D. Hill, "Supporting very large dram caches with compound-access scheduling and missmap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, 2012.

[15] E. Speight, H. Shafi, L. Zhang, and R. Rajamony, "Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 346–356.

[16] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A mostly-clean dram cache for effective hit speculation and self-balancing dispatch," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*. IEEE, 2012, pp. 247–257.

[17] J. H. F.Hameed, L. Bauer, "Simultaneously optimizing dram cache hit latency and miss rate via novel set mapping policies," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2013.

[18] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 381–391.

[19] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely Jr, and J. Emer, "Adaptive insertion policies for managing shared caches," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 208–219.