JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.16, NO.1, FEBRUARY, 2016 ISSN(Print) 1598-1657
http://dx.doi.org/10.5573/JSTS.2016.16.1.080 ISSN(Online) 2233-4866

A Locality-Aware Write Filter Cache for Energy Reduction of STTRAM-Based L1 Data Cache

Joonho Kong

Abstract—**Thanks to superior leakage energy efficiency compared to SRAM cells, STTRAM cells are considered as a promising alternative for a memory element in on-chip caches. However, the main disadvantage of STTRAM cells is high write energy and latency. In this paper, we propose a low-cost write filter (WF) cache which resides between the load/store queue and STTRAM-based L1 data cache. To maximize efficiency of the WF cache, the line allocation and access policies are optimized for reducing energy consumption of STTRAM-based L1 data cache. By efficiently filtering the write operations in the STTRAM-based L1 data cache, our proposed WF cache reduces energy consumption of the STTRAM-based L1 data cache by up to 43.0% compared to the case without the WF cache. In addition, thanks to the fast hit latency of the WF cache, it slightly improves performance by 0.2%.**

Index Terms—**Spin torque transfer random access memory, filter cache, energy efficiency, performance, L1 data cache**

I. INTRODUCTION

The advent of new memory cells has been providing a new opportunity for computer architecture design. Spin torque transfer random access memory (STTRAM) cells are one of the promising memory cells which can potentially replace SRAM cells. They have significantly lower leakage energy consumption and smaller cell size with comparable (or less) read access energy and latency compared to the conventional 6T SRAM cells. Due to their better leakage energy efficiency, there have been many proposals to use STTRAM cells for on-chip caches [1-7]. However, the main disadvantages of STTRAM cells are high write energy and latency. Particularly, a long write latency of STTRAM cells may lead to a severe performance loss in the case of L1 caches where accesses to the cache occur very frequently. Thus, the employment of STTRAM cells for on-chip caches have been explored mainly for large-scale L2 or last-level caches where write operations occur sporadically. In this case, a long latency of write operations can be sufficiently hidden by write buffers or queues as most of the write operations will be filtered by L1 caches.

On the other hand, write energy and latency of STTRAM cells can be optimized by adjusting the retention time of the cells [7, 8]. It enables a deployment of the STTRAM cells in L1 caches [8, 9] as it can reduce write latency as well as write energy of the STTRAM cells. Though write energy and latency of the optimized STTRAM cells are still higher than those of 6T SRAM cells, thanks to the lower read access energy and leakage power, energy consumption of the STTRAM-based L1 caches can be reduced by up to 40% with a negligible performance loss [8] compared to that of the SRAM-based L1 caches. However, a portion of write dynamic energy consumption of STTRAM-based L1 caches is still significant (~45%), which means there is a considerable room for energy optimization in STTRAM-based L1 caches.

In the case of L1 instruction caches, they do not have store operations and write energy is only consumed when

Manuscript received Jul. 26, 2015; accepted Nov. 2, 2015
Joonho Kong is with the School of Electronics Engineering, Kyungpook National University
E-mail : joonho.kong@knu.ac.kr

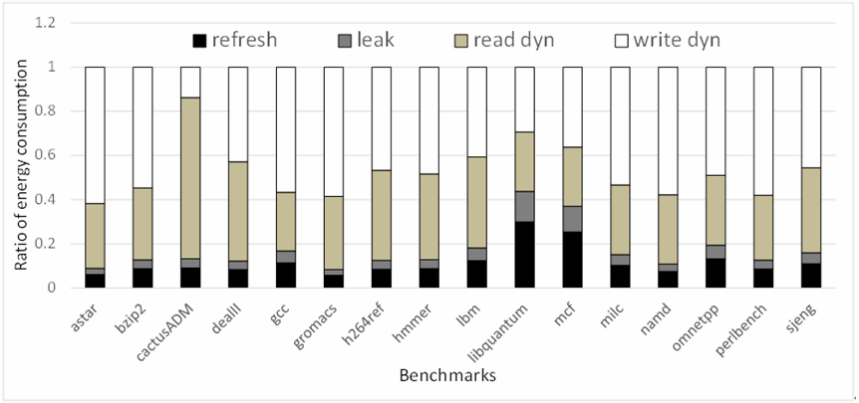

Fig. 1. An energy breakdown of STTRAM-based L1 data cache. The simulation framework and energy parameters are presented in Section 4.

a cache line is filled into the cache. Hence, just employing the optimized STTRAM cells to the L1 instruction cache without any further optimization would nearly yield an optimal energy consumption. However, in the case of L1 data caches, a situation is totally different because of the store operations. Since L1 data caches are accessed nearly every cycle due to superscalar and out-of-order executions, store operations frequently occur in the L1 data caches. Consequently, it leads to huge write dynamic energy consumption in STTRAM-based L1 data caches.

For energy reduction in STTRAM-based L1 data caches, we propose a locality-aware write filter (WF) cache which resides between the processor core's load/store queue (LSQ) and L1 data cache. It primarily filters write accesses (as well as read) to the L1 caches, reducing the energy consumption of the STTRAM-based L1 data cache. Moreover, thanks to fast hit latency of WF cache, it brings a little performance benefit. The main architecture of our WF cache is similar to the conventional filter caches [10, 11]. However, we use novel WF cache allocation and access policies optimized for STTRAM-based L1 data caches.

The rest of this paper is organized as follows. Section 2 presents our motivational study which advocates a need for write dynamic energy optimization in STTRAM-based L1 data caches. Section 3 describes our proposed write filter cache architecture with new line allocation and access policies. Section 4 describes our evaluation results. Section 5 briefly skims recent literature regarding STTRAM-based on-chip cache designs and filter cache designs. Lastly, Section 6 concludes this paper.

II. MOTIVATION

STTRAM cells typically have lower (or comparable) read access latency and energy, and significantly less leakage power consumption compared to the conventional 6T SRAM cells. However, an immediate deployment of STTRAM cells for on-chip caches may have energy and performance inefficiency due to their high write dynamic energy consumption and latency. In order to mitigate those demerits, one can optimize write energy and latency of the STTRAM cells by reducing retention time of the cells [7, 8]. The shorter retention time the STTRAM cells have, the lower write energy and latency they tend to have.

A study in [8] shows a considerable energy reduction of the L1 cache by employing the optimized STTRAM cells that have a shorter retention time. By paying periodic refresh energy, it can reduce write dynamic energy in the STTRAM-based L1 cache. It sufficiently advocates the use of STTRAM cells in the L1 caches in addition to the large-scale L2 or L3 caches. However, write dynamic energy consumption in STTRAM-based L1 data cache is still significant even though the L1 data caches adopt the optimized STTRAM cells (which have reduced write energy and shorter retention time). As shown in Fig. 1 even with the optimized STTRAM cells, write dynamic energy consumption occupies 45% in energy consumption of the STTRAM-based L1 data cache, on average. It means there is still a significant room for optimizing write energy consumption in STTRAM-based L1 data caches.

In L1 data caches, there are two categories that cause

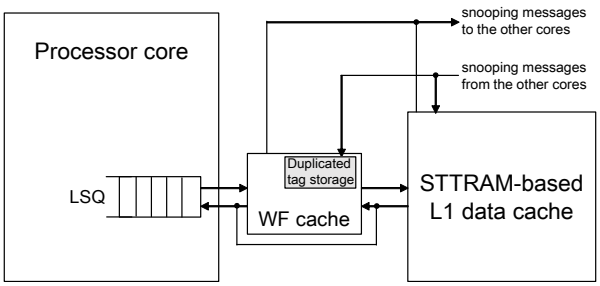

Fig. 2. The overall architecture.

write dynamic energy consumption: data store and block fill. Data store operations occur when the processor core executes store instructions while block fills occur when there is a L1 data cache miss. Since the miss rates in L1 data caches are typically very low (~5%) in general programs, most of the write energy consumption comes from the data store operations issued from the load-store queue (LSQ). Therefore, energy reduction in data store operations is a critical factor for energy optimization in STTRAM-based L1 data caches.

III. WRITE FILTER CACHE DESIGN

1. Architecture

To minimize store operations to the L1 data cache, we propose to use a small write filter (WF) cache between the load-store queue (LSQ) and L1 data cache as depicted in Fig. 2. The main architecture for our WF cache is similar to the ones previously proposed [10, 11]. These techniques are mainly for performance improvement and energy reduction by exploiting the fast hit latency and low access energy of the filter cache. However, the main purpose of our filter cache is to filter write (store) operations to the L1 data cache which eventually leads to a huge write energy reduction in the STTRAM-based L1 data cache. Thus, our WF cache has different filter cache allocation and access policies (which will be explained in Sections 3.2 and 3.3) with the hardware architecture similar to the conventional filter caches.

Our WF cache is composed of the conventional 6T SRAM cells as it should have fast access time and low read/write dynamic energy. The associativity of the WF cache is fully-associative and the replacement policy is least recently used (LRU). The line size of the WF cache is 64 Byte (same as the L1 data cache line size in our evaluation). For design simplicity, the WF cache has a strict inclusion property with the L1 data cache. Hence, when the cache line in the L1 data cache is evicted, the corresponding line in the WF cache is also evicted. In this paper, we use 4-entry and 8-entry configurations for the WF cache size. The higher number of entries will increase a hit rate of the WF cache while it also increases dynamic and leakage energy overhead from the WF cache.

In the chip multi-processor architecture with write-through caches, there is a write buffer to hide write latency. However, the typical write buffer resides between L1 and L2 cache while the WF cache is between the LSQ and L1 cache. Thus, our WF cache can be used along with the write buffers.

To support cache coherence, the snooping messages must be sent to the other core's L1 data cache whenever the store operation is done in either WF cache or L1 data cache. In order to address this case, as shown in Fig. 2, snooping messages are sent from the WF cache or L1 data cache. When a store hit occurs in the WF cache, the state of the corresponding shared cache line is changed to modified state and an invalidation message is sent to the other cores. The updated data is written in the WF cache and also goes to the write buffer for L2 cache update in the case of write-through cache. In this case, the data is not updated in the L1 data cache. Note that the data update in the L1 cache occurs when the dirty line is evicted from the WF cache. For MESI cache coherence protocol, two-bit storage per cache line in the WF cache is required for maintaining the coherency status. Since our WF cache has only small number of entries (~8), the area overhead for this storage is negligible.

When the cache invalidation signal comes from the other cores, the cache line in both WF cache and L1 data cache must be invalidated. To support this, the invalidation signal is also sent to the WF cache and the corresponding cache line is invalidated if there is a cache line that corresponds to the address designated in the invalidation message. To eliminate a latency overhead due to the tag-lookup during the cache snooping, we can have duplicate copy of the tags in WF caches. Since the tag size of the WF cache entry is small (42-bit per entry when using 48-bit physical address), the area overhead is very small.

2. WF cache Line Allocation Policy

The line allocation policy of the WF cache plays a key role for reducing write dynamic energy consumption in STTRAM-based L1 data caches. In this paper, we propose two different line allocation policies for the WF cache: WF_RD and WF_WR. The WF_RD policy is similar to the allocation policy of Hit Cache [10]. The WF_RD policy does not allocate the cache line to the WF cache along with the block (line) fill in the L1 data cache (i.e., when there is a cache miss). When the read (load) hit occurs after the line fill in the L1 data cache, then the corresponding cache line is allocated to the WF cache. The main advantage of the WF_RD policy is that it can filter both consecutive read and write accesses to the L1 data cache when there is a high read/write temporal locality. Since the WF cache hit latency is smaller than the L1 data cache hit latency, it would also lead to better performance. However, the main disadvantage of the WF_RD policy is that the WF cache is mostly used for filtering read operations in the L1 data cache. Recall that typical programs tend to execute more number of load instructions than store instructions. It implies the WF_RD policy incurs frequent line replacements in the WF cache, potentially resulting in more evictions of the cache lines that have high temporal locality on write operations. Because filtering write operations is a critical factor for STTRAM-based L1 data cache energy reduction, the WF_RD policy may lead to suboptimal energy reduction in the L1 data cache.

The other policy (WF_WR) allocates the cache line to the WF cache when the cache write (store) hit occurs in the L1 data cache. The WF_WR policy operates as a prefetching mechanism for the cache line which will be written (by store instructions) in the near future. With the WF_WR policy, the WF cache tends to have a lower replacement rate than the WF_RD, which in turn leads to fewer evictions of the cache lines that have a high temporal locality on write operations. Hence, the WF_WR can filter the write operations better than the WF_RD policy.

The conventional filter caches (i.e., L0 caches) allocates the cache line in the filter caches whenever there is a cache miss in the filter cache. It implies a new cache line is always allocated in both L1 cache and filter cache when there is a cache miss in the L1 cache. In contrast, our new line allocation policy adopts a lazy allocation of the cache lines in the WF cache. The rationale behind this decision is that too frequent cache line replacement between the WF cache and L1 data cache would result in more evictions of the cache line that will be accessed (particularly for store operations) in the near future. To summarize, Table 1 describes the cases when the line allocation in the WF cache occurs across various allocation policies.

Table 1. Line allocation policies in the WF cache and the conventional filter cache (FC) [11]

	Cache miss	Cache load hit	Cache store hit
WF_RD	X	O	O
WF_WR	X	X	O
FC	O	O	O

As in the general write-back cache, a dirty line eviction from the WF cache incurs a cache line update (write operation) in the L1 data cache for data coherence. In the case of the clean line eviction, it can be silently evicted (i.e., no write operation in the L1 data cache) since we enforce the strict inclusion property between the WF cache and L1 data cache.

3. WF Cache Access Policy

Fig. 3 shows the line access policy and required clock cycles of the WF cache. The WF cache is always accessed first before accessing the L1 data cache. In the case of data load hit in the WF cache, the data is served within 1 cycle. Similar to the case of data load, data store hit in the WF cache also takes 1 cycle for the data to be updated in the WF cache. In the case of WF cache misses, it takes different clock cycles for load and store since the STTRAM cells have a longer write latency than the read. In our configuration, the read operation takes 2 cycles for an L1 data cache access while it takes 4 cycles for a write operation [8]. Thus, in the case of WF cache miss and L1 data cache hit, it takes total 3 cycles and 5 cycles for load and store, respectively.

4. WF Cache Line Allocation and Write-Back Overhead

Along with the access latency overhead of the WF cache, there are line allocation and write-back latency overhead. However, the line allocation to the WF cache

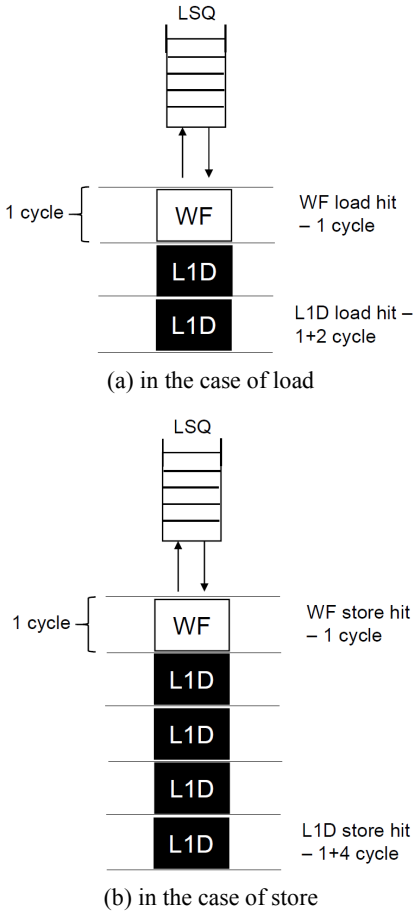

Fig. 3. The WF cache access policy.

does not need to stall the processor pipeline since the required data for load instructions can be directly forwarded from the L1 data cache to the LSQ regardless of the WF cache line allocation. In the case of write-back from the WF cache to the L1 data cache, it takes 4 cycles to update the L1 data cache. It can be done in the background and does not make the CPU core be stalled. However, when the CPU core accesses the cache line which are currently written (by write-back from the WF cache or line-fill from the L2 cache), the cache access cycle is increased by the remaining cycles for the completion of write operations in L1 data cache. We carefully modeled these overheads in our evaluation to obtain an accurate simulation results. Since long physical distance between the WF cache and L1 data cache would result in additional latency overhead for line allocation and write-back. To minimize data transfer latency overhead, the WF cache designer should carefully determine the distance between the WF cache and the L1 data cache as close as possible. Careful P&R (place & route) process during the processor design would result in negligible data transfer latency overhead.

Table 2. The meaning of abbreviations used in Eqs. (1-3)

Abbre.	Meaning
HR_{L1D}	L1 data cache hit rate
HL_{L1D}	L1 data cache hit latency
MR_{L1D}	L1 data cache miss rate
HR_{L2}	L2 cache hit rate
HL_{L2}	L2 cache hit latency
MR_{L2}	L2 cache miss rate
L_{MEM}	Main memory access latency
HR_{FC}	Filter cache hit rate
HL_{FC}	Filter cache hit latency
MR_{FC}	Filter cache miss rate
$Ratio_{LD}$	The ratio between hit load instructions and hit memory access instructions (load+store)
$Ratio_{ST}$	The ratio between hit store instructions and hit memory access instructions (load+store)
RL_{L1D}	L1 data cache read latency
WL_{L1D}	L1 data cache write latency

5. Analytical Performance Model for WF Cache

In this subsection, we demonstrate an analytical performance model for our WF cache. To estimate memory-side performance, we use AMAT (average memory access time) for performance metric. Based on the model shown in [12], we extend the analytical model to describe AMAT when there is filter cache between the LSQ and L1 data cache.

The conventional (i.e., without filter cache) AMAT with two-level cache hierarchy can be calculated as follows:

$$AMAT_{conv} = HR_{L1D} * HL_{L1D} + MR_{L1D} * (HR_{L2} * HL_{L2} + MR_{L2} * L_{MEM.}) \quad (1)$$

The explanations of abbreviations in Eqs. (1-3) are shown in Table 2. For extension of this model to cover filter cache (i.e., L0 cache), we add terms for filter caches as follows:

$$AMAT = HR_{FC} * HL_{FC} + MR_{FC} * AMAT_{conv} \quad (2)$$

For STTRAM-based L1 data caches, the read and write latency will be different. To reflect this impact, the HL_{L1D} can be calculated as follows:

$$HL_{L1D} = Ratio_{LD} * RL_{L1D} + Ratio_{ST} * WL_{L1D} \quad (3)$$

Table 3. Energy and latency parameters (45 nm process node) for SRAM- and STTRAM-based L1 data cache [8]

	SRAM-based	STTRAM-based ('lo2' in [8])
Read energy (nJ)	0.075	0.035
Write energy (nJ)	0.059	0.187
Leakage power (mW)	57.7	1.98
Read latency (cycle)	3	2
Write latency (cycle)	3	4
Retention time (us)	-	26.5

Since full performance simulation often takes huge time, our simple analytical performance model can reduce the development cycle of STTRAM-based L1 cache with filter caches. Please note that the simulation statistics shown in Table 2 can easily be collected via fast cache-only simulation tools (e.g., DINERO [13]).

IV. EVALUATION

For energy evaluations, we use an SRAM and STTRAM energy parameters from [8]. For STTRAM-based L1 data caches, we assume that the cell configuration from [8] (denoted as 'lo2' in [8]) is used. For program-dependent energy evaluations, cache access traces and counts (for both WF and L1 data caches) are extracted from M-SIM [14] which is originated from SimpleScalar [15] architectural simulator. We evaluate performance in terms of IPC (instruction per clock cycle) which is also extracted from M-SIM. The micro-architectural parameters in our M-SIM simulator are tuned for ARM Cortex-A15 [16] as close as possible. The L1 data cache configuration is 4-way set associative 32 KB cache with LRU replacement policy and line size is 64 B. Table 3 summarizes the parameters of SRAM and STTRAM-based L1 data caches for energy and performance evaluations.

We also take the filter cache energy consumption into consideration. In this paper, we show two different configurations for the filter cache size: 4-entry (SR_E4, ST_E4, ST_WF_WR_E4 and ST_WR_RD_E4) and 8-entry (SR_E8, ST_E8, ST_WF_WR_E8 and ST_WR_RD_E8). Table 4 summarizes energy and latency parameters for the filter (as well as WF) cache which are derived from CACTI [17]. Please note that energy evaluation results shown in Section 4.1 already include the filter cache energy consumption except for the case of not using filter caches (SR and ST configurations).

Table 4. The filter cache energy and latency parameters (45 nm process node) derived from CACTI [17].

	*_E4	*_E8
Access energy (nJ)	0.003890	0.004683
Leakage power (uW)	21.6	26.6
Read latency (cycle)	1	1
Write latency (cycle)	1	1

Table 5. Various configurations compared in Sections 4.1 and 4.2. The difference between the FC, WF_RD, and WF_WR is explained in Table 1

Configurations	Meaning
SR	SRAM-based L1 data cache
SR_E4	SRAM-based L1 data cache + 4-entry FC
SR_E8	SRAM-based L1 data cache + 8-entry FC
ST	STTRAM-based L1 data cache
ST_E4	STTRAM-based L1 data cache + 4-entry FC
ST_WF_RD_E4	STTRAM-based L1 data cache + 4-entry WF_RD
ST_WF_WR_E4	STTRAM-based L1 data cache + 4-entry WF_WR
ST_E8	STTRAM-based L1 data cache + 8-entry FC
ST_WF_RD_E8	STTRAM-based L1 data cache + 8-entry WF_RD
ST_WF_WR_E8	STTRAM-based L1 data cache + 8-entry WF_WR

For comparison with the prior art, we also show the energy and performance results of the conventional filter cache. The SR_E4, SR_E8, ST_E4, and ST_E8 correspond to the case where SRAM-based L1 cache with the 4 and 8-entry conventional filter cache and STTRAM-based L1 cache with 4 and 8-entry conventional filter cache, respectively. To summarize various configurations, Table 5 shows the various configurations and their abbreviations for conciseness.

We run selected 16 programs from SPEC2006 benchmark suite. We fast-forward 2 billion instructions and actually run 1 billion instructions for energy and performance evaluations. We assume the clock frequency of the processor as 2 GHz.

1. Energy

Fig. 4 shows normalized L1 data cache energy consumption results across various configurations. Compared to SRAM-based L1 data cache, STTRAM-based L1 data cache (ST configuration in Fig. 4) reduces energy consumption by 58% on average due to low read access energy and leakage energy of STTRAM cells.

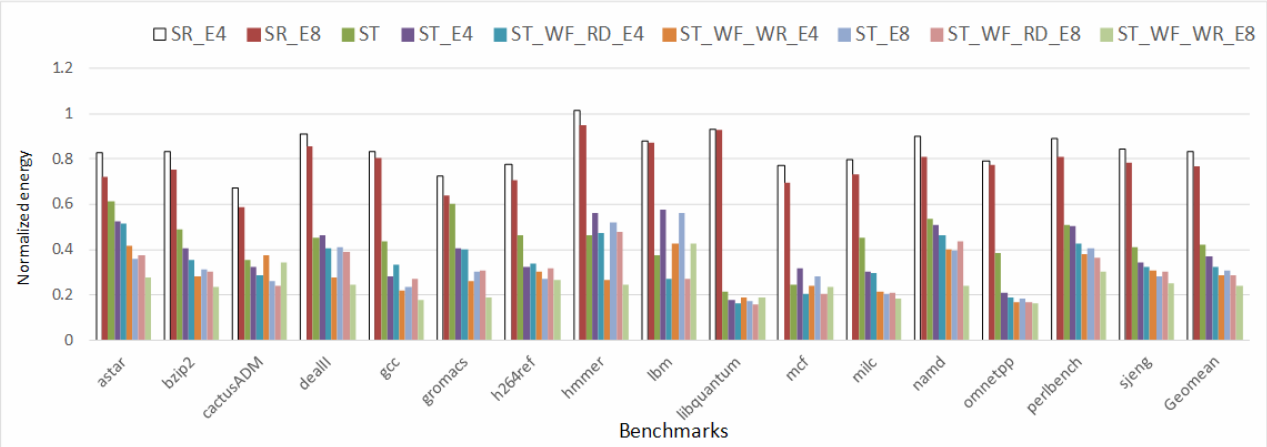

Fig. 4. Normalized energy consumption across the various configurations. The results are normalized to the energy results when using SRAM cells in the L1 data cache.

However, with the WF cache, one can further reduce STTRAM-based L1 data cache energy. The ST_WF_RD_E8 configuration can reduce energy by 32% compared to the STTRAM configuration. In the case of ST_WF_WR_E8, our WF cache reduces L1 data cache energy by 43% compared to the STTRAM configuration. The ST_WF_WR policies show more energy reduction compared to the ST_WF_RD policies. This is because of higher WF cache write hit rates with the WF_WR policies than those with the WF_RD as shown in Fig. 5. Though the ST_WF_RD policies show far higher WF cache read hit rate, increasing WF cache write hit rates contributes to energy reductions much more than the WF cache read hit rates. In addition, a larger number of the WF cache entries also contributes to energy reductions as it yields higher WF cache hit rates (for both read and write).

Our proposed policies (ST_WF_RD and ST_WF_WR), which are geared towards STTRAM-based L1 data caches, show better energy-efficiency compared to the conventional filter cache (FC configurations). The ST_WF_RD_E8 and ST_WF_WR_E8 policies show less energy consumption by 6% and 21% (on average) compared to the ST_E8. It means our ST_WF_RD and ST_WF_WR policies are more suitable for STTRAM-based L1 cache and filter cache.

As shown in Fig. 4, there are some programs which can be regarded as outliers. In the case of *CactusADM*, the ST_WF_WR_4E shows more energy consumption than the ST configuration. This is because there is much fewer number of store operations in *CactusADM* than the other programs, meaning that there is less room for write energy reduction (as already depicted in Fig. 1). Consequently, the energy overhead from the WF cache becomes more than the energy reduction from the L1 data cache. In the case of *lbm*, the ST_WF_WR policies also incur more energy consumption than the STTRAM configuration. The main reason for *lbm* is low write hit rates (~0%) in the WF cache due to the low temporal locality on store operations. In the case of *hmmer*, the ST_WF_WR shows significant energy reduction while the ST_WF_RD consumes more energy than the STTRAM configuration. This is because the ST_WF_WR shows much higher WF cache write hit rates (~96% for both ST_WF_WR_4E and ST_WF_WR_8E) than those in the case of the ST_WF_RD (only 22% and 40% for ST_WF_RD_4E and ST_WF_RD_8E, respectively).

We also estimate the system-level energy consumption by using McPAT 1.3 [18]. Fig. 6 shows the component-level energy breakdown in dual-core ARM Cortex-A15 based system-on-a-chip (SOC). We also use 45nm process node for the energy estimation. In the results shown in Fig. 6, the SRAM-based L1 data cache is used. Since L1 data cache consumes ~7% energy in the entire SOC, our ST_WF_WR_E8 can reduce system-level energy by 5.4% compared to the SR configuration. Compared to the ST configuration, our ST_WF_WR_E8 configuration reduces system-level energy consumption by 1.3%.

Though we do not compare energy results when using the optimized STTRAM cells in the L1 data cache [8] with ours, our WF cache will reduce energy consumption

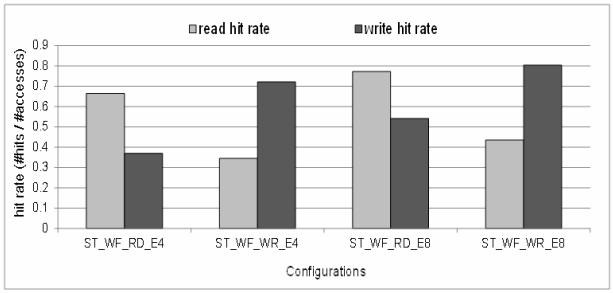

Fig. 5. Average WF cache read and write hit rates.

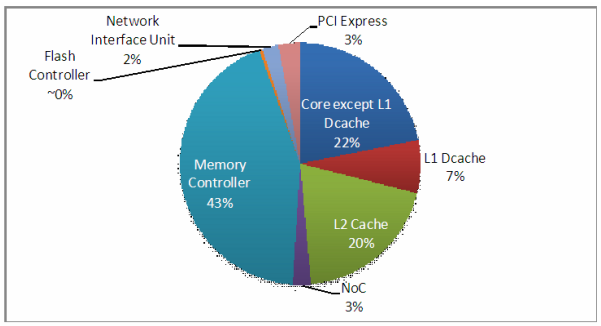

Fig. 6. Energy breakdown in dual-core ARM Cortex-A15 based system-on-a-chip (SOC) derived from McPAT [18].

of STTRAM-based L1 data cache regardless of which type of optimized STTRAM cells is used in the L1 data cache. This is because our WF cache is orthogonal to the STTRAM cell optimization. We leave quantifying the impact of STTRAM cells in the L1 data cache when using the WF cache as our future work.

2. Performance

Fig. 7 shows performance evaluation results across various configurations. As in Section 4.1, performance results for each configuration are also normalized to those for SRAM-based L1 data cache configuration. The ST configuration shows almost identical performance compared to the SR configuration. Though the ST has higher write latency, lower read latency of STTRAM cells offsets this latency overhead. When using the WF caches, one can slightly improve performance by up to 0.2% (in the case of using ST_WF_RD_8E) compared to the ST configuration since a WF cache hit enables faster data access. Between the ST_WF_RD and ST_WF_WR, the ST_WF_RD policies show slightly better performance compared to the ST_WF_WR policies. This is because ST_WF_RD yields much higher WF cache read hit rates as shown in Fig. 5. Since the processor pipeline is out-of-order issue and programs shown in Fig. 7 are not quite load/store-intensive (except for *mcf*), performance improvement from the WF_RD policies is marginal.

As an outlier, in the case of *mcf*, the ST_WF_RD significantly improves performance by 25~27% compared to the ST configuration while the ST_WF_WR shows little performance benefit. For *mcf*, the WF_WR only shows WF cache read hit rates of 24~29% while the WF_RD shows those of 64~78%. The huge differences in the WF cache read hit rates for *mcf* translate into the performance difference between the ST_WF_RD and ST_WF_WR.

Compared to the conventional filter cache (FC), our proposed policies (WF_RD and WF_WR) show only a small performance loss. Compared to ST_E8, the ST_WF_RD_E8 and ST_WF_WR_E8 show only 0.5% and 0.6% performance losses on average. Considering an efficient trade-off between the energy and performance, our ST_WF_RD and ST_WF_WR show much better results thanks to the huge energy benefit.

Table 6. The WF cache area overhead

	vs. SRAM-based	vs. STTRAM-based
ST_WF_*_4E	0.84%	3.77%
ST_WF_*_8E	1.00%	4.47%

3. Area Overhead

The area overhead of our WF cache is <5% of the STTRAM-based L1 data cache area. As shown in Table 6, 4-entry WF cache area overhead is only 3.8% compared to the STTRAM-based L1 data cache. Compared to the SRAM-based L1 data cache, the area overhead of the WF cache becomes ~1%. For multi-core cache coherence, the WF cache requires duplicated tag arrays as explained in Section 3.1. However, tag array size in the cache is small (<10% of data array size). Thus, even with the duplicated tag storage, one can conservatively estimate that the WF cache area overhead is under 5% of the entire STTRAM-based L1 data cache area. Considering the L1 data cache occupies only a small portion in the entire processor area, the area overhead of the WF cache will be negligible compared to the whole processor area.

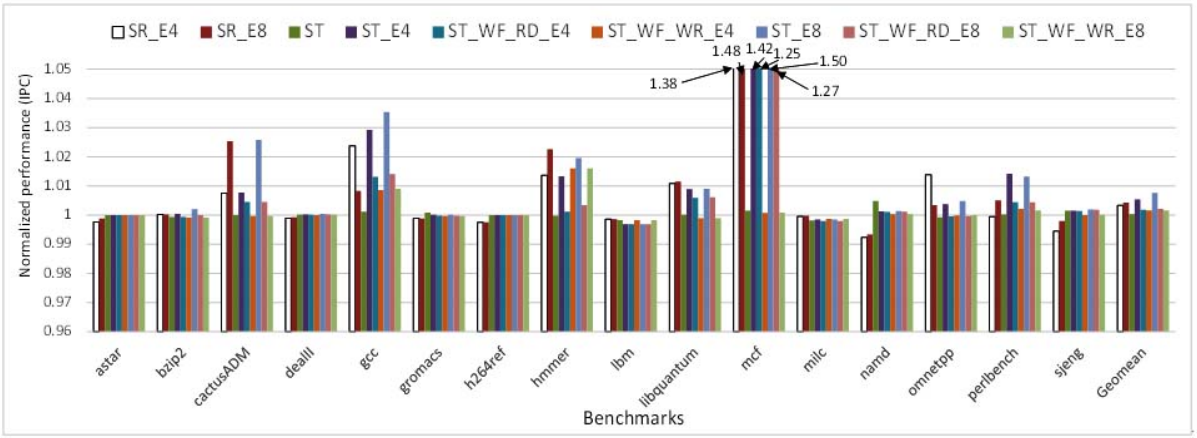

Fig. 7. Normalized performance (IPC) across the various configurations. The results are normalized to the performance results when using SRAM cells in the L1 data cache.

V. Related Work

Many proposals for STTRAM-based cache architecture have been introduced. However, many of those proposals focus on energy reduction in L2 or last-level caches which exploit lower leakage power consumption and smaller cell area of STTRAM cells compared to SRAM cells [1-7]. Several proposals have been introduced for STTRAM-based L1 caches. In [8], STTRAM-based L1 and L2 cache architectures are explored by using the write energy-optimized STTRAM cells with shorter retention time. In [19], an SRAM-STTRAM hybrid L1 cache design is introduced for multi-core cache coherence. For refresh energy reduction in STTRAM-based L1 caches, a 'no refresh' scheme was proposed in [9]. A compiler-assisted STTRAM-based L1 cache architecture was also proposed in [20]. In [20], a small loop cache is used to reduce energy consumption in STTRAM-based L1 instruction caches. Unlike the proposals introduced above, we exploit a small write filter cache for STTRAM-based L1 data cache energy reduction. As already demonstrated in Section 4, our technique not only reduces L1 data cache energy but also improves performance with small hardware cost.

For filter cache design, several different types of filter cache or victim caches have been proposed. In [11], the filter cache architecture was proposed for power reduction and energy-delay product improvement. Several studies have focused on reducing energy consumption in L1 instruction caches. In [22], a different type of filter cache was proposed to reduce processor's front-end power consumption. The authors of [22] proposed decode filter cache that contains high-locality decoded instructions. In [23] and [24], novel types of filter cache architectures are explored for multi-core processors and temperature management, respectively. Different from the previous studies on filter cache architecture, our work focuses on reducing energy of STTRAM-based L1 data cache with new block allocation policies: WF_RD and WF_WR.

VI. Conclusions

In this paper, we propose the WF cache that efficiently filters write operations in the STTRAM-based L1 data cache. We propose two different line allocation policies: WF_RD and WF_WR. The WF_RD is geared towards filtering both read and write operations in the L1 data cache while the WF_WR is primarily for filtering write operations. According to our evaluation results, our WF cache (in the case of ST_WF_WR_E8) reduces energy consumption of STTRAM-based L1 data cache by up to 43%. Furthermore, it slightly improves performance by 0.2% (in the case of ST_WF_RD_E8). We believe that our WF cache can be a low-cost alternative that enables energy- and area-efficient STTRAM-based L1 data cache. As our future work, we are planning to extend our work as follows:

1) we will develop an adaptive scheme which switches between the WF_WR and WF_RD policies in runtime by referring to the workload characteristics;

2) we will evaluate our WF cache with the adaptive

scheme in full system simulator with running operating systems;

3) we will quantify energy and performance impact of our WF cache with the adaptive scheme when using the various STTRAM configurations as in [8].

ACKNOWLEDGMENTS

This research was supported by Samsung Electronics.

REFERENCES

[1] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs," in Proceedings of 2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC), 2012, pp. 243–252.

[2] J. Ahn, S. Yoo, and K. Choi, "DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture," in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014, pp. 25–36.

[3] Z. Wang, D. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive Placement and Migration Policy for an STT-RAM-based Hybrid Cache," in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014, pp. 13–24.

[4] Y.-T. Chen, J. Cong, H. Huang, C. Liu, R. Prabhakar, and G. Reinman, "Static and Dynamic Co-optimizations for Blocks Mapping in Hybrid Caches," in Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, 2012, pp. 237–242.

[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy Reduction for STT-RAM Using Early Write Termination," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD) - Digest of Technical Papers, 2009., 2009, pp. 264–268.

[6] S. P. Park, S. Gupta, N. Mojumder, A. Raghunathan, and K. Roy, "Future Cache Design Using STT MRAMs for Improved Energy Efficiency: Devices, Circuits and Architecture," in Proceedings of 2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC), 2012, pp. 492–497.

[7] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. Stan, "Relaxing Non-volatility for Fast and Energy-efficient STT-RAM Caches," in Proceedings of 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), Feb 2011, pp. 50–61.

[8] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 2011, pp. 329–338.

[9] J. Yao, J. Ma, T. Chen, and T. Hu, "An Energy-Efficient Scheme for STT-RAM L1 Cache," in Proceedings of 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp. 1345–1350.

[10] N. Duong, T. Kim, D. Zhao, and A. V. Veidenbaum, "Revisiting Level-0 Caches in Embedded Processors," in Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2012, pp. 171–180.

[11] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, 1997, pp. 184–193.

[12] D. Patterson and J. Hennessy, "Computer Architecture: A Quantitative Approach. 5th ed., Morgan Kaufmann; 2011.

[13] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator", [Online]. Available: http://www.cs.wisc.edu/~markhill/DineroIV/.

[14] J. J. Sharkey, D. Ponomarev, and K. Ghose, "M-Sim: A Flexible, Multithreaded Architectural Simulation Environment," in Technical Report CS-TR-05-DP01, Department of Computer Science, State University of New York at Binghamton, 2005.

[15] "SimpleScalar toolset." [Online]. Available: http://www.simplescalar.com

[16] "ARM Cortex-A15." [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a15.php

[17] N. Muralimanohar and R. Balasubramonian, "CACTI 6.0: A Tool to Model Large Caches."

[18] 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 Micro-architecture, 2009, pp. 469–480.

[19] J. Wang, Y. Tim, W.-F. Wong, Z.-L. Ong, Z. Sun, H. H. Li, "A coherent hybrid SRAM and STT-RAM L1 cache architecture for shared memory multicores". In Proceedings of 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014, pp. 610–615.

[20] Y. Li, Y. Zhang, H. LI, Y. Chen, and A. K. Jones, "C1C: A Configurable, Compiler-guided STT-RAM L1 Cache," ACM Transactions on Architecture and Code Optimization, vol. 10, no. 4, pp. 52:1–52:22, 2013.

[21] J. Ahn and K. Choi, "LASIC: Loop-Aware Sleepy Instruction Caches Based on STT-RAM Technology," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 5, pp. 1197–1201, 2014.

[22] W. Tang, R. K. Gupta, and A. Nicolau, "Power Savings in Embedded Processors through Decode Filer Cache", in Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 443–448, 2002.

[23] Young Jin Park, Hong Jun Choi, Cheol Hong Kim, and Jong-Myon Kim, "Energy-aware Filter Cache Architecture for Multicore Processors", in Proceedings of Fifth IEEE International Sympo-sium on Electronic Design, Test & Applications (DELTA), pp. 58–62, 2010.

[24] Hong Jun Choi, Young Jin Park, Seung Gu Kang, Cheol Hong Kim, Sung Woo Chung, Jong-Myon Kim, and Dongseop Kwon, "Thermal-aware Duplicated Filter Cache for Improving Processor Reliability", in Proceedings of the 2010 International Conference on Computer Design (CDES), pp. 160–168, 2010.

Joonho Kong received the BS degree in Computer Science from Korea University, Seoul, Korea, in 2007. He received the MS and PhD degrees in Computer Science and Engineering from Korea University, Seoul, Korea, in 2009 and 2011, respectively. He also worked as a postdoctoral research associate in the Department of Electrical and Computer Engineering, Rice University. He is now an assistant professor in the School of Electronics Engineering at Kyungpook National University. His research interests include computer architecture design, temperature-aware microprocessor design, reliable microprocessor cache design, and hardware security.