Rethinking On-chip DRAM Cache for Simultaneous Performance and Energy Optimization

Fazal Hameed* and Jeronimo Castrillon†
Center for Advancing Electronics Dresden (cfaed), Technische Universität Dresden, Germany
Email: *fazal.hameed@tu-dresden.de, †jeronimo.castrillon@tu-dresden.de

***Abstract*—State-of-the-art DRAM cache employs a small Tag-Cache and its performance is dependent upon two important parameters namely bank-level-parallelism and Tag-Cache hit rate. These parameters depend upon the row buffer organization. Recently, it has been shown that a *small row buffer organization* delivers better performance via improved bank-level-parallelism than the traditional *large row buffer organization* along with energy benefits. However, small row buffers do not fully exploit the temporal locality of tag accesses, leading to reduced Tag-Cache hit rates. As a result, the DRAM cache needs to be re-designed for small row buffer organization to achieve additional performance benefits. In this paper, we propose a novel tag-store mechanism that improves the Tag-Cache hit rate by 70% compared to existing DRAM tag-store mechanisms employing small row buffer organization. In addition, we enhance the DRAM cache controller with novel policies that take into account the locality characteristics of cache accesses. We evaluate our novel tag-store mechanism and controller policies in an 8-core system running the SPEC2006 benchmark and compare their performance and energy consumption against recent proposals. Our architecture improves the average performance by 21.2% and 11.4% respectively compared to large and small row buffer organizations via simultaneously improving both parameters. Compared to DRAM cache with large row buffer organization, we report an energy improvement of 62%.**

I. Introduction

Recently, industry has introduced die-stacked DRAM technologies namely hybrid memory cube (HMC) [1], hybrid bandwidth memory (HBM) [2], and DDR4-3DS [3], which are employed to satisfy the huge memory and bandwidth requirements from emerging applications [4] with large memory footprints. Die-stacked DRAM technologies have been adopted as Last-Level-Cache (LLC) [5]–[9] to improve the system performance by reducing the number of high-latency off-chip accesses. The reason is that LLC performance plays a significant role in determining the overall performance of a multi-core system due to increasing latency gap between processor and off-chip memory.

Fig. 1 depicts a typical multi-core cache hierarchy used in [5]–[7]. The DRAM cache is composed of multiple banks where each bank is provided with a **R**ow **B**uffer (RB) as shown in Fig. 2. When an access is made to a DRAM bank, one row of the DRAM bank is fetched into the bank's RB. The data in the RB can be accessed at much lower latency and lower energy than accessing it from the DRAM bank. State-of-the-art DRAM cache use a large RB size (i.e. 2KB or 4KB) per DRAM cache bank. This *large RB organization* incurs high energy consumption via buffering large number of unnecessary data. To reduce energy consumption, state-of-the-art employs multiple small RB's per bank instead of a single large RB per bank [10]. This *small RB organization* has shown improved performance benefits compared to the large RB organization in the context of off-chip memory due to an improved bank-level-parallelism. However, the small RB organization has not been studied in the context of DRAM cache.

State-of-the-art DRAM cache [5]–[7] is equipped with a small low latency Tag-Cache which holds the tags of the recently accessed DRAM cache sets. The Tag-Cache provides significantly fast tag lookup (see Table I) compared to the scenario when the tags are read from the DRAM cache. The performance of DRAM cache depends upon the Tag-Cache hit rate. However, we found that employing small RB organization [10] using existing DRAM cache suffers from reduced Tag-Cache hit rates. To address this problem, we rethink the design of DRAM cache to make it viable for small RB organization. More precisely, we make the following contributions:

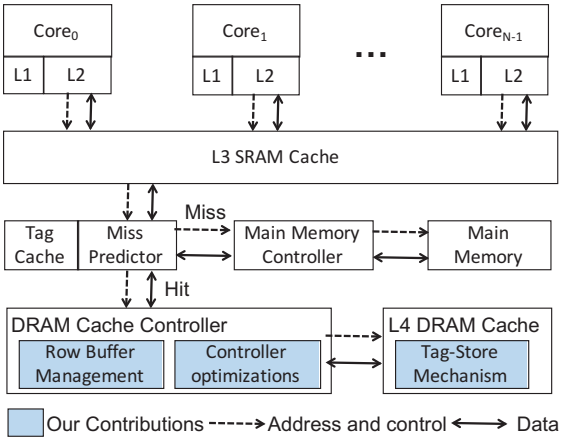

Fig. 1. Typical DRAM based multi-core cache hierarchy for an N-core system; see Section IV-A and Table III for core and cache parameters

1) We propose a novel tag-store mechanism (in Section III-A) that provides significantly high Tag-Cache hit rate compared to the small RB organization [10]. At the same time, it reduces the DRAM cache miss rate compared to both small [10] and large RB organizations [5]–[7].
2) We propose an efficient RB management policy at the DRAM cache controller (in Section III-C) that further improves the performance via a high RB hit rate for the critical read requests.
3) We also found that cache writebacks have reduced temporal locality with a low RB hit rate (less than 5%) that leads to increased latency for the critical read requests. Therefore, we propose an early precharge of writeback request to hide the latency of future read requests.

Consequently, the paper is structured as follows. We first refer to the prior work in Section II. Afterwards, our tag-store mechanism along with controller optimizations is presented in Section III, which is evaluated in Section IV. Section V finally concludes the paper.

II. Background and Related Work

This section presents the details of the state-of-the-art **R**ow **B**uffer (RB) and DRAM cache organizations.

A. Large RB Organization

Typically a DRAM cache is organized into a number of logically independent banks (see Fig. 2-a) where each bank consists of multiple

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

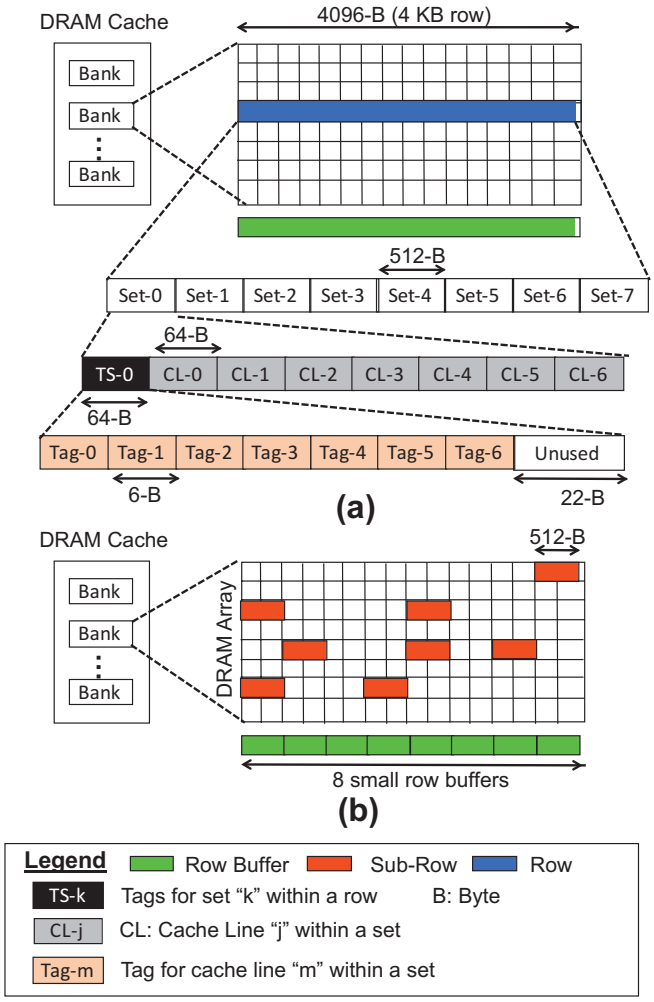

Fig. 2. (a) High level view of DRAM LLC based on commonly used large RB organization and LLC set organization employed in [6], [9] for a 4KB (4096 Bytes) row size (b) Small RB organization [10]

TABLE I
LATENCY OF A READ REQUEST FOR DIFFERENT SCENARIOS IN STATE-OF-THE-ART DRAM CACHE [5]–[7] EXCLUDING CONTROLLER LATENCY

RB Hit	Tag-Cache Hit	Tag Check Latency	Cache Line Latency	Total Latency
No	No	24 cycles	40 cycles	64 cycles
Yes	No	24 cycles	22 cycles	46 cycles
No	Yes	2 cycles	40 cycles	42 cycles
Yes	Yes	2 cycles	22 cycles	24 cycles

rows of data. In a large RB organization [5]–[9], [11], a single large RB is available at each bank. When an access is made to a DRAM bank, one row of the DRAM bank is fetched into the bank's RB. This operation is called as *row-activate*. Any subsequent access to the row residing in the RB (called RB hit) will not require the row-activate operation. When an access is made to a different row of the same bank (called RB miss), the contents of the RB are substituted by the new row after the current row residing in the RB is written back to the DRAM bank. This operation is called as *precharge*. An RB miss requires to store the contents of the current row (precharge operation), time to read the new row (row-activate operation) and the RB read/write access time. On the other hand, an RB hit only requires the RB read/write access time which significantly reduces access latency and energy compared to an RB miss.

B. DRAM Cache Organization

Various DRAM cache organizations [5]–[9], [12] have been proposed in the past. Here we discuss the most recently organization proposed in [5]–[7], shown in Fig. 2-(a). These works propose to store the tags and data of DRAM cache in the same row. Each 4KB DRAM row consists of 8 sub-rows where each sub-row comprises one cache set with 7-way associativity (i.e. one sub-row consists of one tag block and seven cache lines). Once the entire row is fetched into the RB, the tag block must be be accessed before the cache line. Both of these operations are directly performed on the RB. The authors also propose a small low latency hardware structure namely Tag-Cache that caches the tags of recently accessed DRAM cache sets. Table I shows the latency of a read request for different scenarios in state-of-the-art DRAM cache. Note that the latency values do not show the queuing delay in the DRAM cache controller (time spent in the DRAM cache controller before having an access to a DRAM bank). Accesses that hit in the Tag-Cache are serviced with much lower latency because they do not require DRAM cache access for the tags. Similarly, accesses that hit in the RB are serviced with much lower latency because they do not require DRAM bank access for the cache line. The performance of a DRAM cache depends upon both RB and Tag-Cache hit rates.

C. Small RB Organization

The small RB organization proposed in [10] employs multiple small RBs per bank instead of a single large RB per bank. Their approach divides the large DRAM row and RB into multiple sub-rows and sub-RBs respectively as shown in Fig. 2-(b). They only fetch the requested sub-row into one of the sub-RB instead of fetching the entire row. Compared to large RB organization, the small RB organization significantly improves the energy consumption due to reduction in the energy consumed by the activate and precharge operations on the smaller sub-rows. Additionally, in a large RB organization the entire bank is unavailable while an operation is being performed on the RB, and therefore any access to other rows of the same bank will be delayed until the current operation on the RB completes. Serialized accesses to two different rows of the same bank suppresses the bank-level-parallelism, which increases the queuing delay at the DRAM cache controller. The use of multiple RB's in small RB organization allows multiple accesses to the same bank in parallel which improves bank-level-parallelism, herby reducing the queuing delay.

III. PROPOSED DRAM CACHE ARCHITECTURE

Fig. 1 shows the high level overview of a DRAM-based multi-core cache hierarchy enumerating our novel contributions. Similar to [10], we employ small RB organization as shown in Fig. 2-(b). The small RB organization has been shown to have performance and energy benefits compared to the large RB organization [5]–[7] as explained earlier. However, the main drawback of existing small RB organization is that it suffers from reduced Tag-Cache hit rate. To improve it, we developed the tag-store mechanism depicted in Fig. 3-(a) and subsequently explained in Section III-A. We further improve the row buffer hit rate via novel RB management policy which is explained in Section III-C. Consequently, we present the latency hiding techniques in Section III-D. Finally, the overheads of our approach are discussed in Section III-E.

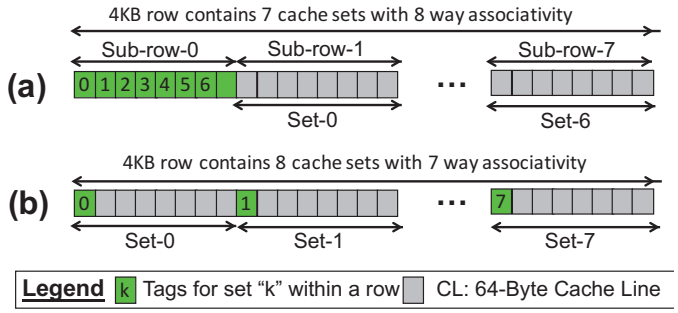

Fig. 3. (a) Our proposed Tag-store mechanism (b) State-of-the-art Tag-store mechanism [6], [7], [9]

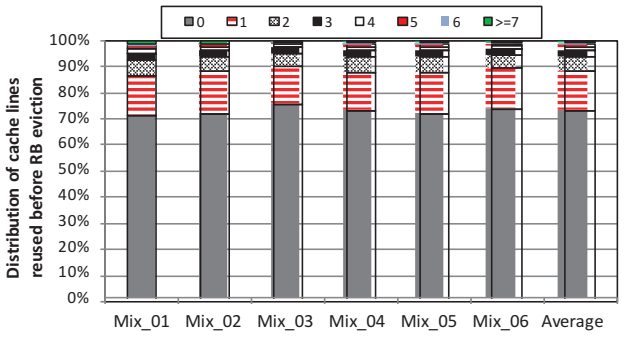

Fig. 4. Distribution of number of cache lines reused in the RB before RB eviction for 4KB RB size and using large RB organization for SPEC2006 application mixes (see Table II)

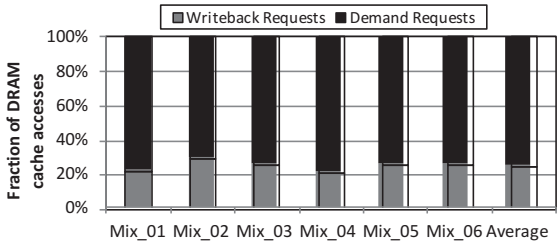

Fig. 5. Distribution of DRAM cache accesses for SPEC2006 application mixes (see Table II)

A. Tag-store Mechanism

Our proposed Tag-store mechanism is illustrated in Fig. 3-(a) where each 4 KB DRAM row comprises 7 cache sets with 8-way associativity. The tag block (shown in green box in Fig. 3) stores the tags of all cache lines within a set. Each DRAM row consists of eight 512-Bytes sub-rows and we dedicate one sub-row for storing all the tag blocks. In contrast, state-of-the-art Tag-store mechanism [6], [7], [9] stores the tag block with in the same sub-row along with the cache lines as shown in in Fig. 3-(b). An access to a cache line in our proposed approach requires two sub-rows accesses (one for the tag block and other for the cache line access) in contrast to a single sub-row access (tag block and the cache line reside in the same sub-row). To reduce the number of accesses to the tags in DRAM, we employ a small low-latency Tag-Cache similar to [5]–[7] that exploits the temporal locality by caching the tags of spatial adjacent DRAM cache sets. Therefore, a Tag-Cache hit bypasses the sub-row access dedicated for the tags while reading the tags directly from the Tag-Cache. Let us assume that there is a DRAM cache read request to a cache line that belongs to Set-6 in Fig. 3-(a). To elaborate how our approach works along with the Tag-Cache, we describe the implementation of the following important events to service the above request:

Tag-Cache miss: On a Tag-Cache miss, the sub-row dedicated for the tag blocks (i.e. Sub-row-0) is accessed to read the requested tag block (i.e. tag block "6"). This tag block indicates the location of the cache line in Set-6 (stored in Sub-row-7). After reading the tag block, the DRAM cache controller issues a read request to access the requested cache line in Sub-row-7 (i.e. Set-6) which is forwarded to the requesting core. After that, subsequent read requests are sent to access the remaining tag blocks (i.e. tag blocks "0" to "5") from Sub-row-0 which are placed in the Tag-Cache. We exploit the temporal locality of applications that these prefetched tag blocks will likely be accessed in the near future. In contrast, using existing tag-store mechanism [6], [7], [9] employing small RB organization [10] need to fill only the requested tag block (i.e. tag block "6") in the Tag-Cache. The reason is that accessing adjacent tag blocks will require multiple sub-row accesses (see Fig. 3-b) which is not a viable option in the small RB organization.

Tag-Cache hit: A Tag-Cache hit does not require any DRAM cache lookup for the tag block (i.e. it is directly accessed from the Tag-Cache), so a read request is directly sent to Sub-row-7 to access the requested cache line in Set-6. The tag block "6" is updated in the Tag-Cache (e.g. to update LRU and dirty information etc.) but not in the DRAM cache. In contrast to our approach, existing DRAM cache [6], [7] always update the tag block both in the Tag-Cache and DRAM cache. In our implementation, the tags in the DRAM cache are only updated after its eviction from the Tag-Cache.

RB miss: If there is an RB miss for the sub-row (i.e. Sub-row-0) containing the tag block, then it is fetched into one of the small sub-RBs available at the DRAM bank shown in Fig. 2-(b). Note that Sub-row-0 only needs to be accessed after a Tag-Cache miss. If the sub-row containing the requested cache line (i.e. Sub-row-6 in the above example) does not reside in the RB, then it is fetched into one of the small sub-RB's. In contrast to our approach, state-of-the-art large RB organization [**?**], [5]–[9] fetches all sub-rows (i.e. Sub-row-0 to Sub-row-7) into the the large RB after an RB mis. Our proposed approach only fetches the requested sub-rows instead of fetching all sub-rows that provides significant energy saving compared to the large RB organization. On the other hand, the small RB organization [10] only fetches one sub-row after an RB miss because the tag block and the cache line resides in the same sub-row (see Fig. 3-b). Compared to the small RB organization, the additional latency incurred in fetching the sub-row dedicated for the tag-blocks in our proposed approach is compensated by significant improvement in the Tag-Cache hit rate (see Section IV-B and Fig. 9 for evaluation).

B. Motivation for optimizations

Fig. 4 shows the distribution of the number of cache lines reused in the RB before RB eviction for a 4KB RB size and using large RB organization. Note that each 4KB RB contains 56 cache lines. Fig. 4 shows that majority of the cache lines are unnecessarily fetched into the large RB while they are not reused again. The inefficient RB utilization motivates us to use small RB organization for energy saving because smaller sub-rows requires less energy for the activate and precharge operations compared to large rows. The low utilization of the RB is mainly due to low RB hit rate (less than 5%) of cache writeback accesses as illustrated in Fig. 8. These writeback accesses causes eviction of highly reused rows from the RB which degrades both performance and energy. Note that cache writeback accounts

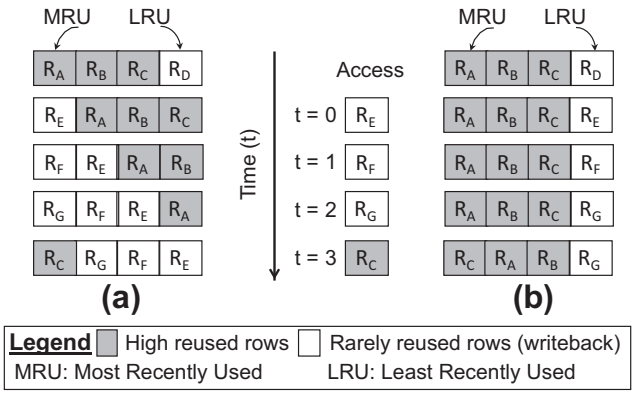

Fig. 6. Example illustrating (a) Traditional LRU policy (b) Our Row Buffer Management policy

for 25% of total cache accesses as shown in Fig. 5. Therefore, the writeback accesses requires special attention when allocating a sub-RB from multiple sub-RB's at the DRAM bank (details in Section III-C). In addition, we propose early precharge of writeback accesses to hide the latency of future read requests which is discussed in Section III-D.

C. Row Buffer Management Policy

The use of small RB organization with multiple sub-RB's at the DRAM bank requires the DRAM cache controller to determine which of the sub-RB to assign for a new sub-row activation. The commonly used RB management policy for multiple RB's is the traditional **L**east **R**ecently **U**sed (LRU) policy employed in [10], [13]. However, the LRU policy does not work well with writeback accesses that have a low RB hit rate. Fig. 6-(a) illustrates a DRAM bank with four small RB's servicing a mix of highly reused row accesses (shown in grey boxes) and rarely reused writeback row accesses (shown in white boxes) for the LRU policy. On an RB miss, a newly requested row is inserted into the **M**ost **R**ecently **U**sed (MRU) position while the RB in the LRU position is evicted to make room for the new row. For instance, at time $t = 0$ following an RB miss, the new row R_E is inserted into the MRU position after evicting the row R_D at the LRU position. As writeback have a high access rate (see Fig. 5), they cause eviction of some of the highly reused rows (e.g. R_B and R_C) from the RB as shown in Fig. 6-(a). Subsequent accesses to these highly reused rows will result in an RB miss, hereby affecting both performance and energy. For instance, at time $t = 3$, an access to highly reused row R_C will result in an RB miss for the LRU policy because R_C was evicted at time $t = 1$ to make room for the rarely reused row R_F.

In contrast to the LRU policy used in [10], [13], our RB management policy inserts the rarely reused writeback accesses in the LRU position as shown in Fig. 6-(b). In our approach, the useful highly reused rows are maintained near the MRU position which prevents their eviction by rarely reused rows. For the example shown in Fig. 6-(b), using our policy, an access to the row R_C will result in an RB hit. Thus, our proposed approach improves the RB hit rate compared to state-of-the-art RB management policy (see Section IV-B and Fig. 8 for evaluation).

D. Early Precharge of writeback requests

State-of-the-art DRAM cache [5]–[9], [11] use an open-page [14] policy for precharging rows (i.e. writing the contents of the row from the RB back to the DRAM array). This policy retains the rows in the RB until they are evicted. When a row is evicted from the RB in an open page policy, the content of that row must be precharged before activating a new row. Since, a row is rarely reused following writeback access, an early precharge of that row will reduce the latency of subsequent RB miss. Thus, our approach performs an early precharge of writeback access to hide the latency of future read request.

E. Overheads

Since our tag-store mechanism has a non-power-of-two number (i.e. 7 sets) of sets with in a row, the address bits cannot be simply used to determine a set. We perform a modulo operation with respect to 7 on the cache line address to identify one of the 7 sets within a row. This implies that the tag entry needs to store additional 3-bits for the set-id, which increases the tag entry size. Note that the tag entry size per cache line in previous tag-store mechanism is 6-byte (48-bits) as shown in Fig. 2-(a). In contrast, our tag entry size per cache line is 51-bits instead of 48-bits. Each cache set in our tag-store mechanism consists of one tag block (64 bytes = 512-bits) and 8 cache lines with 8-way associativity as shown in Fig. 3-(a). The 8 cache lines needs (51 x 8 = 408 bits) for their tag entries with 104 bits (512 - 408 = 104 bits) still left unused. The increased tag size entries for our policy does not incur additional DRAM storage overhead as it utilizes the unused bytes in the tag block. A modulo with respect to 7 will require four 3-bit adders using residual arithmetic. This small modulo circuit takes one additional cycle and negligible energy consumption compared to the large DRAM cache. We further assume that this modulo operation happens in parallel with L3-SRAM cache access. As the modulo operation latency is much smaller compared to L3-SRAM cache access latency (see Table III), this does not incur any additional latency overhead. Our RB management policy insert a cache writeback in the LRU position instead of the MRU position, which does not require any design changes to the traditional LRU policy.

IV. Evaluation

A. Experimental Setup

For the evaluation, an x86 simulator [15] with cycle accurate DRAM timing model is extended to model our proposed DRAM cache architecture. We simulate an 8-core system using various application mixes from SPEC2006 [16], [17] shown in Table II. The details of our system parameters are summarized in Table III for all evaluated configurations. For all evaluated configurations, we employ a MAP-I predictor for predicting DRAM cache misses from [11] and Tag-Cache from [5]–[7]. The DRAM device energy consumption values are taken from [18] which are shown in Table IV. We compare our DRAM cache organization and controller policies with the state-of-the-art for DRAM memory [10] and for DRAM cache [5]–[9]. We compare the following different configurations when applied on top of DRAM cache:

TABLE II
Application mixes. Value in parenthesis denotes the number of instances used for that particular application

Mix_01	astar.t, bzip, leslie3d.r, libquantum, omnetpp, milc, soplex.r, leslie3d.t
Mix_02	astar.t(2), leslie3d.r(2), libquantum(2), lbm(2)
Mix_03	bzip(2), leslie3d.t(2), milc, omnetpp, soplex.r, lbm
Mix_04	astar.t, leslie3d.r, milc, omnetpp(2), soplex.r(2), leslie3d.t
Mix_05	bzip, leslie3d.r(2), astar.t, lbm, milc(2), libquantum
Mix_06	soplex.r, lbm, omnetpp(2), libquantum, leslie3d.t(2), bzip

TABLE III
CONFIGURATION DETAILS FOR THE EXPERIMENTS

Core	3.2 GHz, out-of-order, 4-issue
Private L1$	32 KByte, 8-way, 2-cycles
Private L2$	512 KByte, 8-way, 5-cycles
Shared L3$	8 KByte, 8-way, 20-cycles
Shared L4$ (DRAM)	4 channels, 512 MByte, 64-banks 128-bit wide channel, 4-cycle bus latency tRAS-tRCD-tRP-tCAS-tWR = 72-18-18-18-18 (cycles)
Tag-Cache	38 KByte 1-cycle [5]–[7]
Miss Predictor	Map-I [11], 256 entries
Main Memory (DRAM)	2 channels, 16 KB row buffer, 64-bit wide channel, 800 MHz bus frequency tRAS-tRCD-tRP-tCAS-tWR = 144-36-36-36-36 (cycles)

TABLE IV
PER-BIT ENERGY CONSUMPTION OF MEMORY OPERATIONS NORMALIZED TO THE ENERGY OF ACCESSING THE ROW BUFFER TAKEN FROM [18]

Operation	Normalized Energy
Array Read/Write	1.19
Precharge	0.39
Row Buffer Access	1.00

- **LH-Cache-2KB:** DRAM cache organization proposed by Loh and Hill [8] with a 2 KB RB size employing large RB organization.
- **LAMOST-2KB and LAMOST-4KB:** The best performing DRAM set mapping policy, namely LAMOST, which employs a RB size of 2 KB [9] and 4 KB [6], [7]. Both of these configurations employ large RB organization (recall Fig. 2-(a) and Section II-A).
- **LAMOST-Small:** LAMOST with a small RB organization (recall Fig. 2-(b) and Section II-C). The size of the DRAM row and the sub-row is assumed to be 4 KB and 512 bytes respectively.
- **Ours-Small:** Our Tag-store mechanism along with RB management and early writeback policies build on top of small RB organization (i.e. sub-row size is 512 bytes) which are explained in Section III.

B. Performance Analysis

The main performance results for the evaluated configurations are illustrated in Fig. 7, which depicts the performance speedup normalized to LH-Cache. As shown, our proposal improves the overall performance by 44.8%, (21.2%, 19.2%) and 11.4% compared to LH-Cache, LAMOST with (2KB, 4KB) RB sizes, and LAMOST-Small configurations, respectively. The performance of DRAM cache is primarily affected by two metrics namely DRAM cache read hit latency and DRAM cache miss rate (depends upon associativity). The DRAM cache hit latency comprises two components: the DRAM array latency and the queueing delay at the DRAM cache controller. The DRAM array latency strongly relies on the row buffer hit rate (higher is better; Fig 8), Tag-Cache hit rate (higher is better; Fig 9) while the queueing delay is strongly influenced by bank-level-parallelism.

Table V provides a quantitative and qualitative comparison of important parameters for the evaluated configurations. The performance of our approach is primarily enhanced via an improved DRAM cache read hit latency compared to all configurations as shown in Fig 10. This is because we simultaneously improve (i.e. close to the best value) all of the important parameters. Although our proposal (8-way associative cache) slightly increases the DRAM cache miss rate compared to LH-Cache (29-way associative cache),

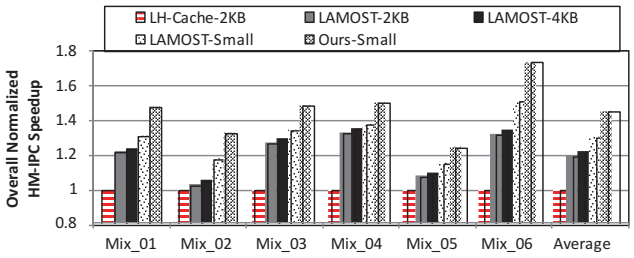

Fig. 7. Performance comparison of different configurations

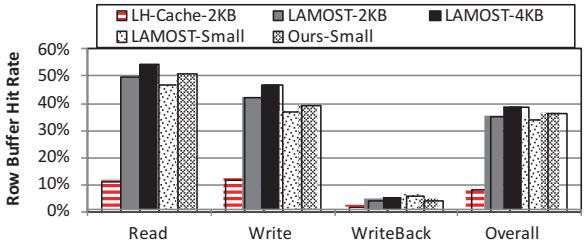

Fig. 8. DRAM Row Buffer hit rate for different configurations

but that is compensated by significant reduction in the read hit latency (58.5%). LH-Cache provides the worst hit latency compared to all configurations that is the primary reason for its worst performance. LH-Cache has a low row buffer and Tag-Cache hit rate due to reduced temporal locality as illustrated in Table V. In addition, LH-Cache also suffers from reduced bank-level-parallelism due to the use of large RB organization that further worsens the hit latency.

Compared to all variants of LAMOST configurations, our proposal provides simultaneous improvement in DRAM cache read hit latency and miss rate. It has a better miss rate compared to all variants of LAMOST configurations because it provides high associativity (i.e. 8-way) compared to LAMOST (i.e. 7-way). The hit latency compared to LAMOST with (2KB, 4KB) RB sizes is reduced via an improved bank-level-parallelism (see Section II-C) because we employ small RB organization. On the other hand, the hit latency compared to LAMOST-Small configuration is improved via an enhanced Tag-Cache (70% improvement) and row buffer hit rate (6.7% improvement). The Tag-Cache hit rate improvement using our proposal compared to LAMOST-Small configuration is achieved because our tag-store mechanism exploits the spatial locality by prefetching all adjacent tag blocks in the Tag-Cache (details in Section III-A). In contrast, LAMOST-Small configuration only fill the requested tag block in the Tag-Cache. Another drawback of LAMOST-Small configuration is that it inserts writeback sub-row accesses in the highest priority MRU position which causes eviction of some of the highly reused sub-rows. Our proposed RB management policy mitigates the negative impact of low locality writeback accesses by inserting them in the low priority LRU position which improves the row buffer hit rate by 6.7% compared to LAMOST-Small configuration.

C. Energy Analysis

The results of our energy analysis are summarized in Fig. 11. As shown, reducing the RB size significantly reduces DRAM cache energy consumption due to reduction in the energy required for the row-activate and precharge operations. The DRAM cache energy savings of our proposal using small RB organization are 72%, 62%, and 79% compared to LH-Cache-2KB, LAMOST-2KB and LAMOST-4KB configurations, respectively, because they employ

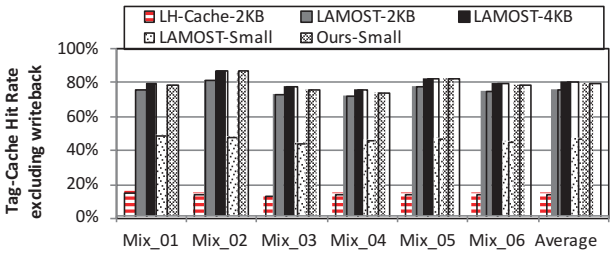

Fig. 9. Tag-Cache hit rates for different configurations

TABLE V
COMPARISONS OF DIFFERENT CONFIGURATIONS. THE RED COLOR INDICATES A BAD VALUE FOR A PARAMETER

Configuration	Tag-Cache Hit Rate	Avg. RB Hit Rate	Avg. Miss Rate	Bank-level-parallelism
LH-Cache	**14.4%**	**8.6%**	17.7%	**Worst**
LAMOST-2KB	75.8%	35.0%	**20.8%**	**Worst**
LAMOST-4KB	80.5%	38.4%	**20.8%**	**Worst**
LAMOST-Small	**46.5%**	33.9%	**20.8%**	Best
Ours-Small	79.4%	36.2%	19.0%	Best

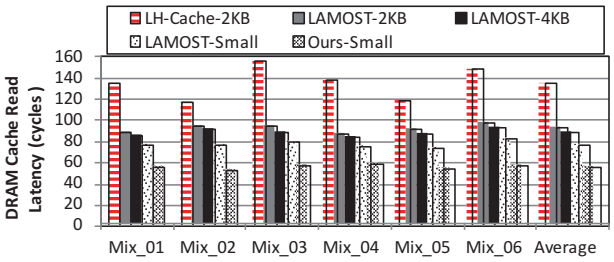

Fig. 10. DRAM cache latency for read request for different configurations

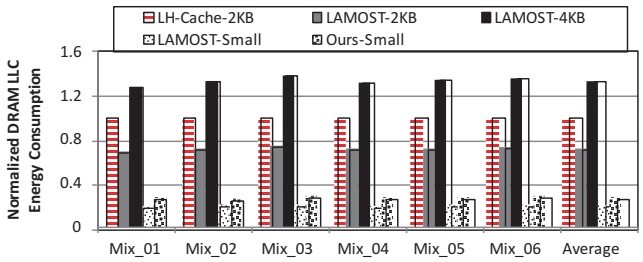

Fig. 11. Energy comparison of our proposed organization compared to state-of-the-art approaches

large RB organization. Our proposal slightly increases the energy consumption compared to LAMOST-Small configuration. However, this slight increase in the energy consumption is compensated by 11.4% improvement in performance. The reason for this increase is that our proposed Tag-store mechanism requires additional sub-row accesses for the tag block because the tag block and the cache line resides in different sub-rows. However, caching the tag blocks in the Tag-Cache bypasses any future sub-row accesses for the tags while reading the tags directly from the Tag-Cache.

V. CONCLUSIONS

This paper introduces our novel policies for on-chip DRAM cache that simultaneously improve important parameters namely bank-level-parallelism (via a small RB organization), Tag-Cache hit rate (via exploiting temporal locality) and RB hit rate (via mitigating the negative impact of writeback accesses). Our proposal not only retain the energy benefits of small RB organization but it also improves the performance by considering the locality characteristics of applications. Our proposed row buffer management policy and the optimization at the DRAM cache controller minimizes the negative impact of low locality writeback accesses that further improves the latency of critical read requests. We have performed detailed comparisons of our proposed policies with state-of-the-art approaches applied on top of DRAM based cache hierarchy. For an 8-core system, our approach improves the performance by 48.8%, 21.2%, and 11.4% compared to the state-of-the-art. The proposed policies comes with negligible hardware overheads which makes them applicable to wide range of DRAM devices.

REFERENCES

[1] D. U. Lee *et al.*, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 432–433.

[2] "HIGH BANDWIDTH MEMORY (HBM) DRAM," http://www.jedec.org/standards-documents/docs/jesd235.

[3] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM using Through-Silicon-Via Technology," in *IEEE Journal of Solid State Circuits*, vol. 45, no. 1, January 2010, pp. 111–119.

[4] D. Gove, "CPU2006 Working Set Size," *SIGARCH Computer Architecture News*, pp. 90–96, March 2007.

[5] C.-C. Huang *et al.*, "ATCache: Reducing DRAM Cache Latency via a Small SRAM Tag Cache," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2014, pp. 51–60.

[6] F. Hameed *et al.*, "Reducing Latency in an SRAM/DRAM Cache Hierarchy via a Novel Tag-Cache Architecture," in *Proceedings of the 51st Design Automation Conference (DAC'14)*, 2014.

[7] F. Hameed *et al.*, "Architecting On-Chip DRAM Cache for Simultaneous Miss Rate and Latency Reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 651–664, April 2016.

[8] G. Loh *et al.*, "Supporting Very Large DRAM Caches with Compound Access Scheduling and MissMaps," *IEEE Micro Magazine, Special Issue on Top Picks in Computer Architecture Conferences*, pp. 70–78, 2012.

[9] F. Hameed *et al.*, "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'13)*, 2013.

[10] N. D. G. *et al.*, "Multiple Sub-row Buffers in DRAM: Unlocking Performance and Energy Improvement Opportunities," in *Proceedings of the 26th ACM International Conference on Supercomputing*, 2012, pp. 257–266.

[11] M. Qureshi *et al.*, "Fundamental Latency Trade-offs in Architecting DRAM Caches," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 235–246.

[12] F. Hameed *et al.*, "Architecting STT Last-Level-Cache for Performance and Energy Improvement," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*, March 2016, pp. 319–324.

[13] G. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *Proceedings of the 35th International Symposium on Computer Architecture (ISCA)*, 2008, pp. 453–464.

[14] S. Rixner *et al.*, "Memory Access Scheduling," in *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, June 2000, pp. 128–138.

[15] G. Loh *et al.*, "Zesto: A Cycle-Level Simulator for Highly Detailed Microarchitecture Exploration," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[16] "Standard Performance Evaluation Corporation," http://www.spec.org.

[17] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Computer Architecture News*, pp. 1–17, September 2006.

[18] E. Kultursay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 256–267.