# Scrabble: A Fine-Grained Cache with Adaptive Merged Block

Chao Zhang, *Student Member, IEEE*, Yuan Zeng, *Student Member, IEEE*, and Xiaochen Guo, *Member, IEEE*

**Abstract**—A large fraction of the microprocessor energy is consumed by the data movement in the system. One of the reasons is the inefficiency in the conventional cache design. Cache blocks larger than a word are used in conventional caches to exploit spatial locality. However, many applications only use a small part of a cache block before its eviction. Transferring and storing unused data wastes bandwidth, energy, and limited cache space. Prior work on fine-grained caches can reduce data access and storage granularity to reduce the amount of unused data. However, small data blocks typically require greater metadata and control overhead. Sharing the common bits among tags of fine-grained blocks can reduce the metadata overhead but the constraints on which fine-grained blocks can share tag bits can cause fragmentation. This work proposes scrabble, a fine-grained cache that can merge multiple non-contiguous fine-grained blocks into a variable size merged block. The length of the shared tag is maximized to reduce the metadata overhead. The space utilization is improved by supporting merged blocks with variable size. The control overhead can be reduced by moving the merged block together from memory to the last level cache. For applications with poor spatial locality, Scrabble cache can achieve more than 40 percent of performance improvement. Even for application with good spatial locality, the speedup is still more than 7 percent. In general, for an evaluated set of benchmarks, Scrabble cache achieves an average of 2.41× effective capacity over the baseline cache with the same cache capacity which leads to a 16.7 percent performance improvement and an 11 percent on-chip energy reduction. As compared to a state-of-the-art fine-grained cache, Scrabble cache achieves a 1.25× effective capacity, a 7.9 percent speedup, and a 5.8 percent on-chip energy reduction.

**Index Terms**—Fine granularity, energy efficiency, caches, data movement, effective capacity, cache utilization, tag sharing, index selection, control overhead, bandwidth efficiency, variable size block, spatial locality, space utilization, data utilization

## 1 Introduction

Data movement contributes to a large portion of the total system energy consumption. The inefficiency of data movement is rooted in the conventional memory hierarchy design. Memory hierarchy designs have been following the same design principle for decades, which is to hide the memory latency assuming good locality. Conventional caches are organized using fixed-size blocks (i.e., 32-128 B) to make use of spatial locality. However, data-intensive workloads do not always have good locality. Even for applications with good locality, the benefit of large access granularity diminishes when running these applications on multicore processors due to the increased contentions in shared cache. Data over-fetching results in a waste of energy, bandwidth, and a reduced *effective capacity*.

To avoid data over-fetching, many fine-grained caches have been proposed [1], [2], [3], [4], [5], [6], [7], [8]. These mechanisms adapt to application locality characteristics to improve the *data utilization* and data movement efficiency. However, the size of the metadata needs to be increased to support data identification, state tracking, and replacement at a smaller data granularity. Furthermore, moving fine-grained data leads to a greater control overhead, which causes longer queueing delay and consumes additional energy. While keeping the total chip area the same, the data capacity is reduced due to an increased metadata overhead. To reduce the metadata overhead for fine-grained caches, some fine-grained caches [9], [10], [11] share common bits in the tag of multiple fine-grained blocks and store the remaining *private tag* to identify each fine-grained block. These designs increase the data utilization and reduce the metadata overhead. However, the constraints on sharing the tag bits reduce the *space utilization*. When there are not enough blocks that share a fixed subset of the tag bits, the data in each merged block cannot fill out the entire cache line. A low cache space utilization reduces the effective capacity.

An ideal fine-grained cache should have 1) low metadata and control overhead, 2) high data utilization (used data/fetched data), and 3) high space utilization (fetched data/data capacity) to maximize the effective capacity and data movement efficiency. In this paper, a new fine-grained cache named scrabble is proposed to systematically optimize for effective capacity. The scrabble cache can store multiple non-contiguous words together in a *merged block* for both L1 and L2. The L2 merged block size varies dynamically to improve the cache space utilization and the L1 merged block size is fixed to maintain a low access latency. The length and position of the private tag and the cache index are carefully

• *The authors are with the Lehigh University, Bethlehem, PA 18015. E-mail: {chz616, yuz615, xig515}@lehigh.edu.*

*Manuscript received 22 Nov. 2018; revised 26 Apr. 2019; accepted 16 Aug. 2019. Date of publication 6 Sept. 2019; date of current version 19 Dec. 2019. (Corresponding author: Chao Zhang.) Recommended for acceptance by A. Gordon-Ross. Digital Object Identifier no. 10.1109/TC.2019.2939809*

TABLE 1
Comparisons of Different Cache Designs

| Designs | Metadata Overhead | Space Utilization | Data Utilization |
|---|---|---|---|
| **Conventional** | Low | High | Low |
| **Word cache** [5] | Very High | High | High |
| **Line distillation** [2] | High | High | High |
| **Amoeba cache** [8] | High | High | High |
| **DSC** [3] | Very Low | High | Low |
| **Elastic cache** [9] | Medium | Low | High |
| **Tag-Split cache** [10] | Medium | Low | High |
| **DyCache** [11] | Medium | Low | High |
| **This work** | Medium | High | High |

selected to optimize for the merging efficiency. Multiple fine-grained blocks in the same merged block share the common tag bits and are stored and transferred together to amortize the control overhead.

This paper makes the following contributions:

- Improving the cache space utilization by caching non-contiguous words with variable size merged blocks.
- Maximizing the merging and tag sharing possibility by a private tag and index selection scheme.
- Reducing the control overhead by fetching fine-grained blocks within a merged block together from memory to cache through grouped line-fill.

## 2 Background and Related Work

The scrabble cache is built upon and inspired by prior work on fine-grained caches. This section summarizes these fine-grained caches and their remaining issues.

### 2.1 Effective Capacity

The primary design goal of fine-grained caches is to increase the effective capacity of on-chip storage and thereby reduce cache misses and off-chip data movement. The effective capacity is the cache space that stores useful data. Effective capacity can be calculated using the following equation:

$$\begin{aligned}\text{EffectiveCapacity} = \text{DataCapacity} \\ \times \text{SpaceUtilization} \times \text{DataUtilization}.\end{aligned}$$

Under a fixed area, the lower the metadata overhead, the higher the data capacity. Cache space utilization quantifies the amount of valid data stored in the data array. Data utilization quantifies the amount of used data as a percentage of valid data before eviction. As shown in Table 1, The proposed Scrabble cache targets the best trade-off among metadata overhead, space utilization, and data utilization.

#### 2.1.1 Data Utilization

Data utilization is the ratio of the useful data to the fetched data. Conventional caches have low data utilization because a fixed size of contiguous data is fetched regardless of spatial locality. Reducing the size of the cache block can improve the data utilization but can also degrade performance for applications or program phases that have good spatial locality. Amoeba cache [8] keeps the tag in the data array for storing different sizes of cache blocks. A tag-bitmap is required per set to indicate where are the tags stored. Each of the tags has the size of one word (64 bits) no matter how many words are stored in a fine-grained block. Amoeba cache requires more storage for the tag. Therefore, Amoeba cache has an increased data utilization but a decreased data capacity as compared to the conventional cache. Moreover, Amoeba cache [8] can only fetch contiguous words from a *physical block*, which prevents further improvement of data utilization. Line distillation [2] combines line-organized cache with the word-organized cache to keep useful words in cache on evictions. The utilization of cache capacity is improved but each cache line needs to be first placed in the line-organized cache, which can not improve the data movement efficiency when cache lines are initially fetched. Spatial prefetching [4] improved the data utilization by skipping the unused words, but for applications with poor or variable spatial locality, this design still wastes cache space and bandwidth. Word organized cache [5] improves cache utilization, bandwidth, and energy consumption. However, 50 percent of space is used to store tags. In Scrabble cache, non-contiguous data from the same block or different blocks can be merged into one block, which improves data utilization without adding significant metadata overheads.

#### 2.1.2 Metadata Overhead

The conventional cache uses relatively large block to minimize the metadata, which includes tag, valid bit, LRU bits. Fine-grained caches reduce the size of the cache blocks to avoid over-fetching when application do not have good locality. However, fine-grained caches typically require a higher metadata overhead as compared to the conventional cache. Decoupled sector cache (DSC) [3] can reduce metadata overhead and improve cache utilization by allowing multiple sectors to share cache space. In this design, tag array is indexed by selection tag and index, whereas the data array is indexed by index and sector offset. Therefore, multiple blocks which have the same index but different selection tags can be stored in the same set. Selection tags are required to be stored in the data array for identifying blocks in the data array and finding the corresponding tags in the tag array. Elastic cache [9] shares the common bits in the tag as the shared tag for multiple fine-grained blocks. The metadata can be effectively reduced. However, the Elastic cache cannot change the granularity of the fine-grained blocks at the runtime, which hurts performance for applications that have good locality. Tag-Split cache [10] and DyCache [11] use a similar technique to share the MSB of the tag among multiple fine-grained blocks. They can adaptively choose the granularity of the fine-grained blocks based on the spatial locality. Tag-Split provides fine-grained modes and coarse-grained mode selection. it samples a number of cache sets to determine the operating mode. However, the sampler sets cannot change their operating mode. Upon the granularity changing, DyCache has to flush all of the cache blocks. In Scrabble cache, multiple fine-grained blocks can be merged into a variable-size merged block. Increasing the number of blocks that can be merged reduces

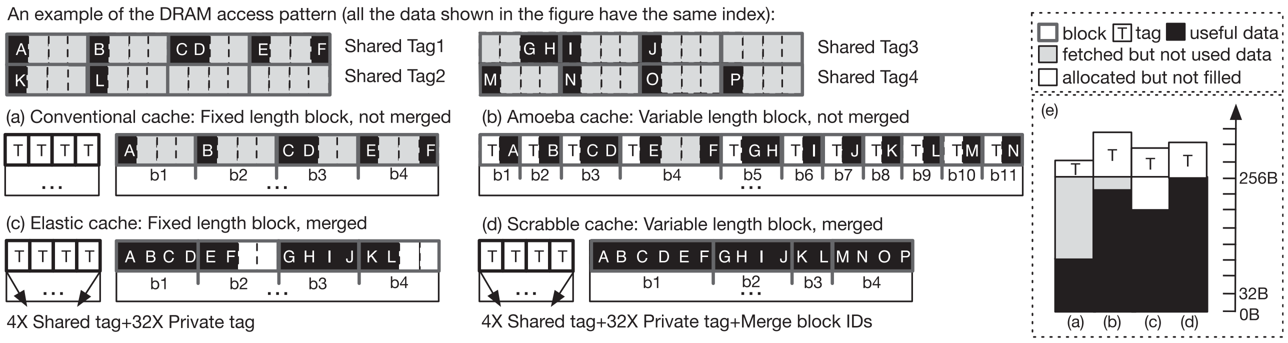

An example of the DRAM access pattern (all the data shown in the figure have the same index):
Shared Tag1
Shared Tag2
Shared Tag3
Shared Tag4
block
tag
useful data
fetched but not used data
allocated but not filled
(a) Conventional cache: Fixed length block, not merged
(b) Amoeba cache: Variable length block, not merged
(c) Elastic cache: Fixed length block, merged
(d) Scrabble cache: Variable length block, merged
4X Shared tag+32X Private tag
4X Shared tag+32X Private tag+Merge block IDs
(e)
256B
32B
0B

Fig. 1. Comparison of the data organization for conventional cache, Amoeba cache, Elastic cache, and Scrabble cache with an example data access pattern (A, B, C, ...). All of the blocks in the figure have the same index. Each of the grey block and the labeled black block is 16B.

the metadata overhead. The private tag and index are carefully selected to maximize the length of the shared tag.

### 2.1.3 Cache Space Utilization

Cache space utilization is the ratio of the fetched data to the data capacity. For the conventional cache, the cache space utilization is typically 1 because the fetching granularity equals to the allocation granularity. For fine-grained caches that merge multiple fine-grained blocks with shared common tag bits [9], [10], [11], all of the fine-grained blocks must have the same shared tag and index to be merged. A merged block may not be completely filled and therefore can waste cache space and reduce the cache space utilization. Due to fragmentation, fine-grained caches might need to evict existing blocks for a new block allocation even if it has enough space. Scrabble cache increases the cache space utilization by adaptively selecting the sizes of the merged blocks. When not enough fine-grained blocks can be merged, the size of the merged block is decreased to reduce wasted space. Moreover, Scrabble cache can allocate non-contiguous space in a set for one merged block, which can effectively mitigate the fragmentation problem.

## 2.2 Control Overhead

Fine-grained cache has a reduced block size. A cache access can hit in the original physical block but miss in the partial block. When this kind of partial miss occurs, the fine-grained cache needs to fetch the missing part of the physical block [8], [10]. As a result, the control overhead for fine-grained caches is typically higher as compared to it is in the conventional cache. To reduce partial misses, a better prediction mechanism is required. Prior work uses spatial pattern predictor [4], [8], [12] to determine the size of a contiguous range of data that will be used. Scrabble cache improves the prediction accuracy by allowing non-contiguous predictions. Hence, Scrabble reduces control overhead due toon partial misses.

Another type of control overhead happens when refilling fine-grained blocks. To fill the same amount of the data, fine-grained caches require more line-fill operations. The control signals and the queuing delay of writing these fine-grained blocks are increased. This work is the first to address this issue by grouping multiple fine-grained blocks into one merged block and write it within one line-fill. The control overhead can be reduced by using the same set of control signals.

## 3 Key Ideas and Design Overview

Scrabble cache aims to optimize the effective cache capacity and control overhead. This section presents the key ideas and an overview of Scrabble cache.

*Private Tag and Index Selection.* Sharing the common bits in the tag among multiple fine-grained blocks can effectively reduce the metadata overhead. However, fine-grained blocks that can be merged must have the same index bits as well as shared tag bits. Prior works [9], [10], [11] keep the position of the index bits the same as the conventional cache and choose the LSBs of the original tag bits to be the private tag of each fine-grained blocks. In this type of tag and index selection, fine-grained blocks that are accessed back-to-back tend to be mapped to different cache set and lose merging opportunities. As a result, the merged blocks tend to have low space utilization. To overcome this problem, the private tag should choose the bits with the higher variations in the address; whereas the shared tag and index should have lower variations. However, selecting the index bits with lower variations increases load imbalance among cache sets and hence increases the conflict misses. In Scrabble cache, the private tag bits are chosen from the LSBs of the address bits except for the block offset. Then, the index bits are chosen from the rest of the LSBs. The MSBs are selected to be the share tag. A hash function that XORed the LSB of the shared tag and the index bits is used to mitigate the load imbalance problem among sets [13].

*Variable Length Merged Block.* For different data regions and different application phases, the spatial locality varies and the number of the fine-grained blocks that can be merged is different. Keeping a fixed sized merged block cannot maximize the merging efficiency. A large merged block can accommodate more fine-grained blocks if they have the same shared tag and index. Therefore, using large merged block can reduce the metadata overhead and maximize the tag bits sharing. However, there may not always be enough blocks to be merged. A small merged block can reduce the unused space and improves the space utilization. Therefore, Scrabble cache allows variable size merged block.

The variable length merged block has advantages over the previous cache design when the locality is not good. Fig. 1 shows an example of how the fetched data are organized in different cache design. In this example, the conventional cache has a fixed 64 B block size which can only store contiguous words. After fetching useful data A-F, the cache

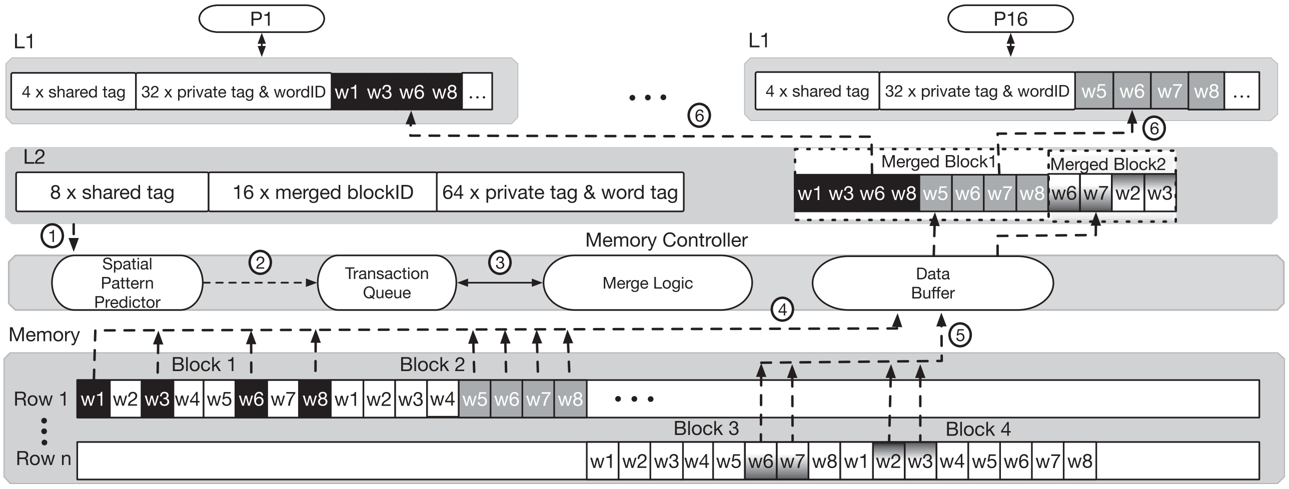


Fig. 2. Scrabble cache design with 16 private 4-way L1s and a shared 8-way L2 cache.

set is full. Future misses will cause block eviction. Amoeba cache avoids fetching some non-useful words by using variable length blocks. However, since this design can only store contiguous words, when the first and the last 16 B of a block are useful (E, F), Amoeba fetches the entire block. The elastic cache can merge non-contiguous words that have the same shared tag into a merged block, which avoids fetching non-useful data. However, because the merged block has a fixed length, some allocated cache space is not filled. Scrabble cache can merge non-contiguous words while having blocks with variable length. In the example, A-F with shared tag1 are stored in a 96 B merged block1. G-J, K-L, M-P are stored in merged blocks with size 64 B, 32 B, and 64 B, respectively. More design details will be discussed in Section 4. Fig. 1e compares the size of the useful data, fetched but not used data, allocated but not filled space, and the metadata size of the four cache design with the same allocated data storing space. Among all of the listed cache designs, Scrabble has the highest percentage of the useful data over the allocated cache space with moderate metadata overhead, which means it has the highest effective capacity.

*Grouped Line-Fill.* For fine-grained caches, the total control overhead of placing the same amount of data into the cache is higher as compared to it is in the conventional cache. In Scrabble cache, multiple fine-grained blocks are merged together at the memory controller. The control overhead can be amortized by using the same set of control signals to write multiple fine-grained blocks.

In Scrabble cache, supporting merged blocks with variable length can improve the effective capacity and thereby reduce the miss rate. However, additional logic and tag bits should be added, which increases the cache access time. Therefore, in the proposed design, the L1 cache has a fixed merged block size to maintain low access delay; whereas the L2 cache has merged blocks with variable length to improve effective capacity and reduce the miss rate.

An overview of Scrabble cache is shown in Fig. 2. Both L1 and L2 caches can store merged blocks. In a 4-way L1 cache, each set stores 32 words in four fixed 64 B block. Four shared tags, 32 private tags, and 32 word tags are used to uniquely identify the data. For an 8-way L2 cache, the number of the merged blocks in one cache set can vary from one to sixteen. To identify each merged block, 16 shared tags are required to support the maximum number of merged blocks (if they are all 32 B), which leads to a large metadata overhead. To reduce the overhead, Scrabble cache limits the number of merged blocks such that the number of the shared tag is the same as the number of tags in the conventional cache. In the proposed L2 cache, 8 shared tag, 16 *merged blockIDs*, 64 private tags, and 64 word tags are stored in the tag array. The merged blockIDs are used to indicate which shared tag should be used for each 32 B space in the data array. This mechanism allows non-contiguous cache space be allocated to one merged block. In the worst case, all of the merged blocks are 32 B and half of the set space can be empty. Fortunately, 84 percent of the merged blocks have the size greater than 64 B for the evaluated applications. Data transferring between L1 and L2 is chosen to be physical block granularity. A study on the data movement between L1 cache and the L2 cache is shown in Section 7.3. A spatial pattern predictor is located at the memory controller and trained by the eviction information. The merge logic in the memory controller determines which partial blocks can be merged together and the sizes of each merged block.

When cache accesses miss in the L2 cache, ① the read requests are first sent to the spatial pattern predictor. ② The spatial pattern predictor generates a bitmap for each read request and the bitmaps are stored in the transaction queue. ③ The merge logic determines which requests can be merged and the sizes of each merged block. ④ When all of the data from merged block1 are returned to the data buffer, the merged block is sent back to the cache together. ⑤ Similarly, a merged block2 with different size is sent to the L2 cache. If there is a miss in the L1 cache, but hit in the L2 cache, ⑥ All the data from the same physical block in the same cache set will be transferred to L1 cache instead of the entire merged block.

## 4 Scrabble Cache

To implement the ideas discussed in Section 3, modifications to the cache hierarchy and the memory controller are required.

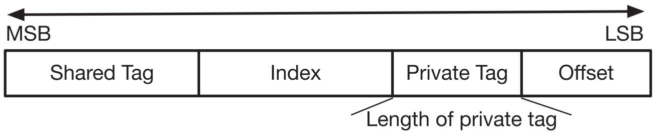


Fig. 3. Private tag selection.

## 4.1 Private Tag and Index Selection

To reduce the metadata overhead and improve the block utilization: length of the private tag and the position of the index bits must be carefully determined. Fine-grained blocks must have the same shared tag and index to be merged together. To increase merging possibility, the shared tag and index should be chosen from the address bits that have lower variations (i.e., LSB) and the private tag should be selected from the ones with higher variations (i.e., MSB). As shown in Fig. 3, the first $n$ LSBs after block offset are selected to be the private tag. The following LSBs are used as the index to allow load balancing among cache sets. The MSBs are used as the shared tag. To reserve the bits with the highest variations to be the private tag, the index bits are shifted towards left, which may cause the cache to have load imbalance. To mitigate this problem, a modified cache index is calculated by XORing the LSB of the shared tag and the index. This new index scheme can effectively reduce the conflict misses without reducing the number of fine-grained blocks that can be merged. The length of the private tag is chosen to be six bits. A detailed discussion about choosing the length of the private tag is in Section 7.1.

## 4.2 Merge Logic

To improve the control efficiency, partial blocks are merged at the memory controller based on the predicted spatial pattern and the merged partial blocks are written back together. The spatial pattern predictor design will be discussed in Section 4.4. To support the dynamic merge architecture, additional circuits are required in the memory controller, which are shown in Fig. 4: ① an MSign and a bitmap per transaction queue entry to indicate whether a block has been merged and to store spatial patterns; ② a merge logic to determine which blocks are merged together; ③ a merge map to record the merge logic results and data return information; and ④ a return logic to determine whether a merged block is ready to be sent back to the cache.

Memory controller typically requires scheduling logic (e.g., FR-FCFS) to choose the ready command with the highest priority, which already requires search capability in a transaction queue and a priority encoder to pick the highest priority request from multiple ready candidates. In the proposed design, the largest merged block size at the memory controller is 128 B [1] and a maximum of 16 fine-grained blocks can be merged into one block. A request selected to be served searches the transaction queue for other requests with the same shared tag and index. The priority encoder in the memory controller selects one request per cycle and it takes up to 16 cycles to find all of the candidate blocks. When the memory controller needs to use the search port and priority encoder, the merge logic is paused, which happens rarely because the average number of blocks that have the same index and shared tag is less than four. While the bitmaps of each candidate block are buffered, the merge logic starts to determine the size of the merged block. The algorithm to generate a merged block is shown in Algorithm 1. The input of the algorithm is 16 8-bits bitmaps from up to 16 mergeable blocks. The output is 16 valid bits which indicate the merged block lengths and 16 MSigns. MSigns for merged candidates are set to one in the transaction queue to avoid merging them again. The merge algorithm counts the number of valid bits for each merging candidates. If the remaining space of a merged block is larger than the number of valid bits of a candidate, the candidate is merged and the remaining space is updated. The corresponding MSigns and valid bits are updated too.

1. This is different from the largest merged block in cache because merging can also happen during refills when two merged blocks in a set have the same shared tag.

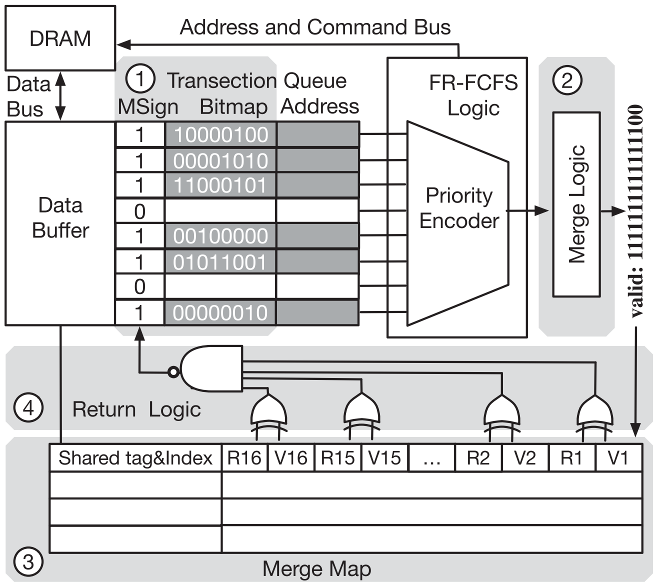


Fig. 4. An illustration of the memory controller logic (the shaded components are new hardware).

**Algorithm 1.** Merge Algorithm

```
1  Input: map[16][8]
2  Output: valid[16], MSign[16]
3  Initial: valid[16]=0, MSign[16]=0, N=0, space=16, p=0
4  for $i \leftarrow 0$ to 16 do
5    for $j \leftarrow 0$ to 7 do
6      if map[i][j] then
7        N++
8      end
9    end
10   while N $<=$ space do
11     MSign[i]=1
12     for $k \leftarrow 0$ to $N$ do
13       valid[p+N]=1
14     end
15     p+=N
16     space-=N
17   end
18 end
```

Part ② in Fig. 4 shows an example of merging 14 words from six fine-grained blocks. After the merged result is

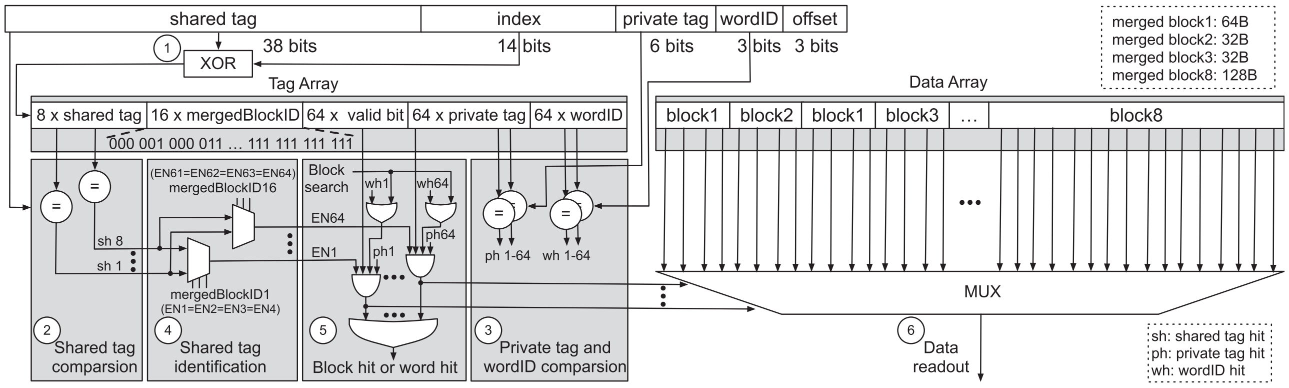


Fig. 5. An example of cache lookup in an 8-way cache with variable size merged blocks.

generated, a new entry will be added into the merge map to store the data return status, cache index, and shared tag for the merged block. One valid bit and one return bit are assigned for each word in the merged block. Return bits are set to all zeros initially. Each time a new data is fetched from the DRAM to the data buffer, the merge map will be checked. If hit, the corresponding return bit of that data will be set to one. if all of the valid words are returned, the merged block can be sent back to the cache together. The first returned fine-grained block in a merged block has to wait for the rest of the merged block. In order to reduce the waiting time, the FR-FCFS scheduling policy is modified to prioritize fine-grained blocks that are determined to be merged.

According to the synthesis results in Section 6, the merging logic takes up to 18 cycles to make merging decision (including the 16 cycle selection). The merging logic can be considered as one additional pipeline stage before accessing to DRAM. The best-case throughput of the merging logic is when merging 16 fine-grained blocks in 18 cycles, whereas the worst-case throughput is when merging two fine-grained blocks in four cycles. Since the maximum DRAM throughput is $1/t_{BURST}$, which is smaller than the worst-case merging throughput, merging will not hurt the system throughput. Latency, area, and power overhead of the additional hardware to support the merge capability is shown in Section 6.

### 4.3 Cache Lookup

To support the proposed variable size merged block, the cache lookup logic needs to be modified. In a conventional cache, a cache block can be uniquely identified by its tag stored in a cache set. In Scrabble cache, each word can be uniquely identified by the shared tag concatenated with the private tag and wordID. On a cache lookup, *word hit* is when all three parts match between the request and a stored word. In addition to the word hit, the cache lookup logic needs to support a *block hit*, which requires a match for both the shared tag and the private tag. Block hit is used to read all partial blocks belonging to a physical block (Sections 4.5 and 4.6).

Fig. 5 shows an example of the tag comparison logic for an 8-way set-associative cache. Each merged block has a shared tag. Each word has its own private tag, wordID, and valid bit. A merged block can be stored non-contiguously in a cache set (e.g., block1 in Fig. 5). Therefore, a mergedBlockID is needed for each word to identify which merged block it belongs to. Since the smallest merged block size is 32B, every four words share a mergedBlockID. In the given example, mergedBlockID1 "000" indicates that the first 32 B in the data array corresponds to the first shared tag. The third 32 B in the data array has the same merged-BlockID, and hence is in the same merged block as the first 32 B.

Cache lookup for a word can be broken down into five steps: ① index to a set; ② check whether the shared tag matches; ③ check whether the private tag and the wordID matches for each word, which happens in parallel with step two; ④identify the shared tag based on the mergedBlockID; ⑤ check whether a word has matched shared tag, private tag, and wordID (when doing the block search, the block search signal is set to one and the words are always hit); and ⑥ read out the words on hit. For the L1 cache, which has a fixed merged block size, step ④is removed.

Area, latency, and energy overhead are estimated by synthesizing the logic and adding tag storage overhead in the cache models. As compared to the conventional cache lookup, Scrabble cache requires more gates to support simultaneous comparison for the shared tag, the private tag, and the wordID of the merged block. Additional logic latency is added to the critical path for checking whether all of the metadata match (step ⑤). Synthesize results in Section 6 show that the additional latency can fit in the same cycle of a conventional L1 lookup. For L2 cache, the latency is longer. Because supporting variable size merged block requires shared tag identifying (step ④), which takes a longer time and is on the critical path. However, results in Section 6 show that the overhead is tolerable.

### 4.4 Spatial Pattern Predictor

When a cache miss happens, Scrabble cache only fetches partial blocks that are predicted to be accessed. Different spatial pattern predictors (SPP) [4], [8], [12], [14], [15] could be used to generate the word access patterns within a block. Scrabble cache uses a table-based SPP. Because the same type of data objects are typically allocated in a contiguous address space and tend to exhibit a similar spatial behavior, each entry in the table stores the spatial pattern of a contiguous memory address region. Word access information is recorded by adding one reference bit per word in the L1 tag array. This section introduces how SPP works in Scrabble

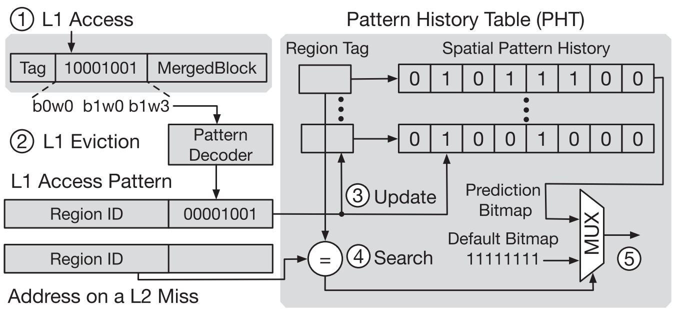


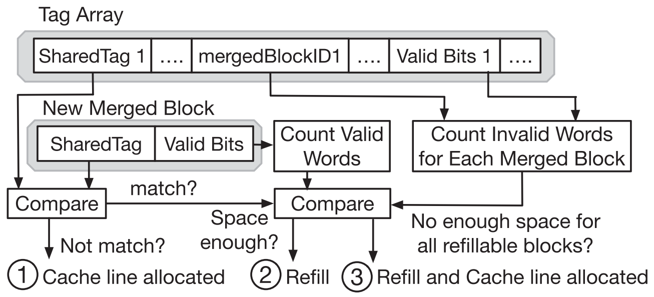
Fig. 6. Spatial pattern predictor.

cache. Detailed trade-offs of different SPP design choice is analyzed in Section 7.2.

In Scrabble cache, SPP is updated on L1 evictions and read on L2 misses. An example is shown in Fig. 6. ①Word access information is updated on each L1 access in the tag array. ② When a merged block is evicted from L1, the recorded access pattern is decoded into a bitmap. In this example, word 0 in physical block 0, word 0 in physical block 1, and word 3 in physical block1 are referenced, therefore bit 0 and bit 3 are one in the decoded bitmap. ③The decoded bitmap is used to update the corresponding entry in the pattern history table at the memory controller. ④On an L2 miss, the regionID of the missed address is used to search the spatial pattern. ⑤If the regionID hit in the PHT, the prediction bitmap will be read out. Otherwise, a whole block is fetched. In Scrabble cache, the SPP is located in the memory controller. Therefore, required partial blocks can be merged and moved to cache together to reduce the control overhead and data movement energy. The overhead for SPP is evaluated in Section 6.1.

### 4.5 Cache Refill and Replacement Policy

In Scrabble cache, the number of merged block is fewer than or equal to the number of cache ways. Words within a merged block are always evicted together. Therefore, no additional LRU bits are needed. If a merged block is not full, it can be refilled by an incoming mergeable block. A new refill logic is needed to support this function. As shown in the Fig. 7, ① the refill logic checks whether there is a shared tag hit for a returned merged block. If not, a new cache line is allocated. If an existing merged block with a matching shared tag is found, ② the refill logic checks whether the existing merged block has enough space. If it does, the new merged block can be refilled to the existing merged block. If space is not enough, ③ refill the existing merged block and allocate a new cache line for the remaining words. L2 cache has variable size blocks. Therefore, allocating space for a new merged block may require an eviction of multiple merged blocks. Merged blocks are evicted based on the LRU policy, if an evicted block does not provide enough space, the next LRU block will be evicted. In Scrabble cache, up to four merged blocks may be evicted to allocation space for a new merged block. Refill logic shares hardware with the cache lookup logic. The only major overhead is the counter for calculating the remaining space for each merged block. The refill logic overhead is reported in Section 6.



Tag Array
SharedTag 1 | .... | mergedBlockID1 | .... | Valid Bits 1 | ....
New Merged Block
SharedTag | Valid Bits
Count Valid Words
Count Invalid Words for Each Merged Block
Compare
match?
Compare
Space enough?
No enough space for all refillable blocks?
Not match?
① Cache line allocated
② Refill
③ Refill and Cache line allocated

Fig. 7. Scrabble cache refill and replacement.

### 4.6 Coherence and MSHR

Scrabble cache can adopt many existing coherence protocols [16], [17], [18], while maintaining the coherent metadata at original physical block granularity.

For example, Protozoa [18] is designed for variable granularity cache block. This work keeps the metadata at a conventional fixed cache block granularity while supporting variable read and write caching granularity.

In this paper, Scrabble cache uses a directory-based coherence protocol that is the same as it is in the baseline system. Similar with the Amoeba cache [8] , the coherence granularity and directory information are maintained at the original physical block size. Multiple cores can cache different partial blocks from the same physical block in shared state. Merging different partial blocks does not create any new false sharings or remove any existing ones. This allows the programmers to optimize program behaviors based on conventional cache blocks.

The proposed architecture requires minimum modifications to the MSHRs. In the conventional MSHRs, once a primary miss occurs, the block address is added to the MSHR. If there are multiple misses on the same block, these misses are recorded in the address stack. After the block returns, the MSHR entry and the associated address stack entries can be invalidated. For Scrabble cache, the challenge is that after the partial block returns, it can not guarantee that all of the misses in the address stack are satisfied by the returned partial block. In the proposed design, the MSHR entry will be recycled only when the returned bitmap can cover all of the misses in the address stack. When any of the associated address stack entry is not satisfied, the corresponding MSHR entry cannot be recycled. An unsatisfied secondary miss becomes the primary miss and issues a read request to the lower level of the cache hierarchy. This inefficient MSHR operation rarely happens because the spatial pattern predictor has a low false negative rate (Section 7.2.4).

## 5 Experiment Setup

This work compares the following fine-grained caches with the conventional cache:

- *Amoeba*: a fine-grained cache that stores tag in data array [8].
- *mDSC*: a modified decoupled sector cache [3] with 64 B sector size and 8 B line size with the same SPP in Amoeba cache.
- *Elastic cache*: a fine-grained cache with tag sharing [9].
- *Scrabble (fixed)*: the proposed Scrabble cache with 64 B fixed block size for both L1 and L2.

TABLE 2
Baseline Configuration

| | |
|---|---|
| **System** | 16 cores, 32 threads, 2.2 GHz, 22 nm node [19]<br>OoO, SMP, issue width = 4<br>L1i: 64 KB, direct-map, 64 B block |
| **Cache** | L1d: 64 KB: 4-way LRU, 64 B block<br>miss delay = 1 cycle, hit delay = 1 cycle<br>L2: 8 MB, 8-way LRU, 64 B block<br>miss delay = 3 cycles, hit delay = 5 cycles |
| **Memory Controller** | FR-FCFS scheduler<br>open-page policy<br>4 Gb x8 I/Os, DDR4-2400 17-17-17 [20]<br>16 banks, 4 channels, 2 ranks |
| **DRAM** | 1 KB page size per chip (Baseline)<br>tRC = 55cycles, tRAS = 38cycles<br>tRP = 17cycles, tCAS = 20cycles |

- *Scrabble (variable)*: the proposed Scrabble cache with 64 B fixed block size for L1 and variable size block for L2.

To be fair, the total storage for tag and data is kept the same among all of the evaluated caches at the same level of the memory hierarchy. The baseline configuration is listed in Table 2. The configurations of the fine-grained caches are listed in Table 3.

## 5.1 Circuit and Architecture

A modified SESC simulator [21] is used with a cycle-accurate memory simulator to evaluate the systems. CACTI 7.0 [22] is used to model the cache access latency, energy, and area. Cadence Encounter RTL Compiler [23] is used to synthesize all of the additional logics to estimate the timing, power, and area overhead. NanGate FreePDK45 open cell library [24] is used and the results are scaled from 45 nm to 22 nm technology node according to the parameters reported in [25].

TABLE 3
Fine-Grained Cache Configurations

| | |
|---|---|
| | **L1 Cache** |
| **Amoeba** | 2.1 KB(T&V) + 67.9 KB(tag&data) = 70 KB<br>miss delay = 1 cycle, hit delay = 1 cycle |
| **Modified DSC** | 10.96 KB(tag) + 59.04 KB(data) = 70 KB<br>miss delay = 1 cycle, hit delay = 1 cycle |
| **Elastic cache** | 11.25 KB(tag) + 58.75 KB(data) = 70 KB<br>miss delay = 1 cycle, hit delay = 1 cycle |
| **Scrabble** | 11.4 KB(tag) + 58.6 KB(data) = 70 KB<br>miss delay = 1 cycle, hit delay = 1 cycle |
| | **L2 Cache** |
| **Amoeba** | 0.3 MB(T&V) + 8.3 MB(tag&data) = 8.6 MB<br>miss delay = 6 cycles, hit delay = 6 cycles |
| **Modified DSC** | 1.21 MB(tag) + 7.39 MB(data) = 8.6 MB<br>miss delay = 4 cycles, hit delay = 6 cycles |
| **Elastic cache** | 1.3 MB(tag) + 7.32 MB(data) = 8.62 MB<br>miss delay = 4 cycles, hit delay = 6 cycles |
| **Scrabble** | 1.3 MB(tag) + 7.32 MB(data) = 8.62 MB<br>miss delay = 4 cycles, hit delay = 6 cycles |

TABLE 4
Applications

| Benchmarks | Suite | Input |
|---|---|---|
| | **Multi-thread** | |
| **CG**, **MG** | NAS OpenMP | Class A |
| **FFT** | SPLASH-2 | 1M points |
| **OCEAN** | SPLASH-2 | 514 x 514 ocean |
| **CHOLESKY** | SPLASH-2 | Reference |
| **WATER-NSQUARED** | SPLASH-2 | Reference |
| **WATER-SPATIAL** | SPLASH-2 | Reference |
| **RADIX**, **LU** | SPLASH-2 | Reference |
| **SWIM**, **ART** | SPEC OpenMP | MinneSpec-Large |
| **HPCCG**, **miniAMR** | Mantevo | Default |
| **GUPS** | HPCC | Default |
| | **Multi-Program** | |
| **LIBQUANTUM**, **GOBMK**, **SOPLEX**, **OMNETPP**, **DEALII**, **MILC**, **LBM**, **GCC**, **ASTAR**, **MCF**, **SJENG**, **Xalan** | SPEC 2006 | Reference |

All of the DRAM parameters are from the datasheet of a SAMSUNG 4 Gb DDR4 device [20]. CPU model parameters are from the Intel Xeon Processor Scalable Family datasheet [19] with a reduced L2 cache capacity to accommodate the small working set size of the simulated workloads [26]. McPAT [27] is used to estimate power consumption of the rest of the system. The L1-L2 and L2-memory controller interconnection energy are estimated to be 6.8pJ per Byte transfer [28]. The total on-chip power calculation includes L1 caches, L2 cache, L1-L2 interconnect, L2-memory controller interconnect, memory controller, and cores power.

## 5.2 Applications

26 applications (Table 4) from a variety of benchmark suites are used in the evaluation: HPCC [29], SPLASH-2 [30], SPEC-OMP [31], SPEC2006 [32], NAS-OMP [33], and Mantevo [34]. These applications are divided into multi-thread and multi-program groups, which exhibit a wide range of spatial locality. Totally one billion instructions are simulated for selected simulation points [35] of each of SPEC applications. 16 copies of each SPEC applications are simulated on the 16-core system.

## 6 Evaluations

This section presents evaluations of circuit overhead, performance, energy, and effectiveness of Scrabble cache.

## 6.1 Area, Latency, and Power Overhead

The proposed architecture requires additional circuits in caches and memory controller. The overhead is summarized in Table 5. As compared to the total cache area (1 mm$^2$ for 16 L1 caches and 9 mm$^2$ for an L2 cache), the total area overhead for L1 and L2 is 0.3 percent. L1 lookup can fit in the same cycle as the conventional L1 does. The additional latency of scrabble comes from one OR gate delay. The synthesis tool is able to adjust the other gates on the critical path to fit the OR

TABLE 5
Overhead

| Component | Area (um$^2$) | Power (mW) | Additional Latency(ps) |
|---|---|---|---|
| **Cache Logic** | | | |
| **L1 Tag comparison** | 1163.8 | 11.7 | 0 |
| **L2 Tag comparison** | 8882.3 | 44.9 | 52.3 |
| **L1 Refill Logic** | 81.5 | 0.5 | 2.8 |
| **L2 Refill Logic** | 1415.3 | 0.6 | 21.4 |
| **Total (16L1s+L2)** | 30222.4 | 240.7 | |
| **Component** | **Area** (um$^2$) | **Power** (mW) | **Latency** (ps) |
| **Memory Controller ($\times$4)** | | | |
| **Spatial Pattern Predictor** | 13844.8 | 14.1 | 182.8 |
| **Merge and Return Logic** | 423.3 | 2.4 | 635.1 |
| **MSigns and Bitmaps** | 840 | 6.6 | 8.1 |
| **Merge Map** | 386.5 | 4.1 | 67.9 |
| **Total ($\times$4)** | 61978.4 | 108.8 | |

gate delay. L2 lookup adds an additional 52.3 ps latency, which leads to one cycle additional access time as compared to conventional cache. The total on-chip area overhead for Scrabble is less than 0.16 percent of the 55.8 mm$^2$ processor chip area. And the total power overhead is less than 0.45 percent of the 77.86 W total on-chip peak power.

## 6.2 Performance

To understand the performance of Scrabble, in this section, the merged block utilization, the L1 and L2 miss rate, the fragmentation issue, and the speedup contribution are discussed.

*Block Utilization.* Scrabble cache can avoid fetching and storing useless data. As compared to baseline, average block utilization before its eviction for both L1 and L2 is improved from the 30 percent to 87.3 percent as shown in the Fig. 8. Across all of the evaluated applications, Scrabble cache can significantly improve the block utilization by merging non-contiguous data into variable size blocks.

*Miss Rate.* The increased block utilization leads to a lower miss rate. For applications that have low spatial locality, the L1 and L2 misses per kilo instructions (MPKI) can be reduced by 29.1, and 27.2 percent respectively (Fig. 9). This is because Scrabble cache can improve the effective capacity (Section 6.4). Even for applications with good locality (lbm and astar e.g.,) the L1 and L2 MPKI can still be reduced by 8.4 percent. For some applications with low block utilization (mcf e.g.,) increasing the effective capacity does not influence the performance much. This is because these applications tend to have a relatively long reuse distance.

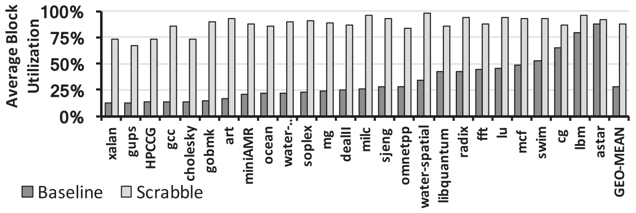
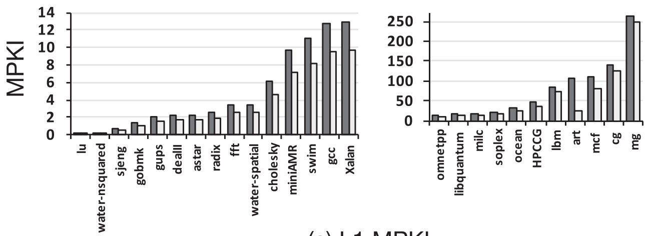


Fig. 8. Block utilization for both L1 and L2.



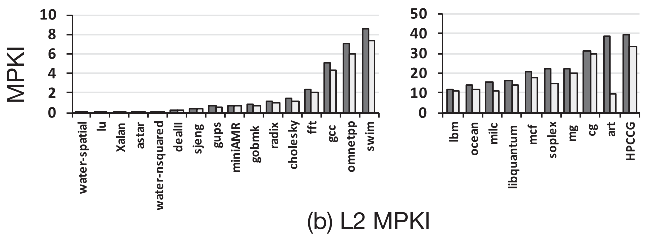


Fig. 9. L1 and L2 MPKI.

The 26 applications are divided into four groups based on their block utilization and L2 MPKI of the baseline configuration as shown in Table 6.

*Fragmentation.* Fine-grained caches can have fragmentation. On misses, fragmentation can cause evictions when the cache set still has enough space. Amoeba cache has this issue due to its adjustable size of the block and T-bitmap mapping. Elastic cache has this issue due to its unmatched number of shared tag and private tag. Scrabble cache also has a similar issue. However, Scrabble can support variable size of merged block and allocate non-contiguous space for one merged block, which can effectively mitigate the fragmentation issue (Fig. 10).

As shown in Fig. 12, low block utilization groups A and B can significantly reduce global miss ratio. However, existing fine-grained caches have more than 20 percent of cache misses that cause evictions due to fragmentation. Scrabble can reduce these evictions to 12 and 9 percent for group A and B respectively. This helps to further increase the cache effective capacity and reduce data movement. For Group C and D, the global miss ratio is not reduced as much as it is in group A and B, because these applications have a relatively high block utilization. However, the percentage of fragmentation can still be reduced by Scrabble. As a result, Scrabble cache can mitigate the fragmentation issue by allocating non-contiguous space for one merged block.

*Speedup Contribution.* To analyze the performance impact of each of Scrabble cache modifications, Fig. 11 shows the

TABLE 6
Application Groups

| |
|---|
| **GroupA (L2 MPKI $<$ 10, block utilization% $<$ 40%)** cholesky, dealII, gcc, gobmk, gups, miniAMR, omnetpp, sjeng, water-nsquared, water-spatial, xalan |
| **GroupB (L2 MPKI $>$ 10, block utilization% $<$ 40%)** art, HPCCG, mg, soplex, milc, ocean |
| **GroupC (L2 MPKI $<$ 10, block utilization% $>$ 40%)** astar, fft, lu, radix, swim |
| **GroupD (L2 MPK I $>$ 10, block utilization% $>$ 40%)** cg, lbm, libquantum, mcf |

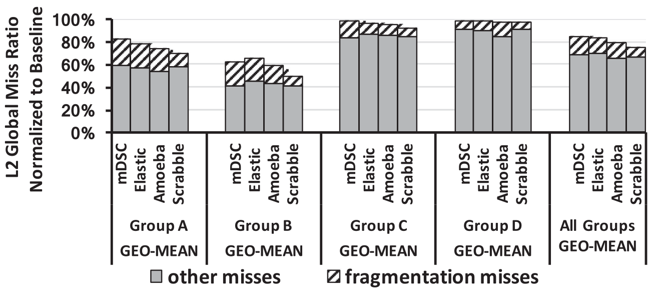


Fig. 10. Fragmentation analysis.

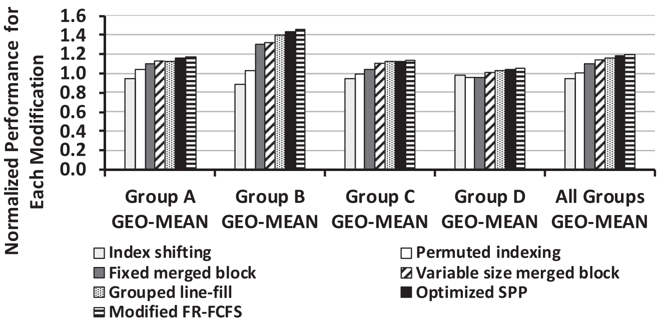


Fig. 11. Speedup Contribution.

average speedup of adding each of the following features one at a time: 1) index shifting, 2) permuted index- ing, 3) fixed size merged block, 4) variable size merged block for L2, 5) grouped line-fill, 6) optimized SPP, and 7) modified FR-FCFS.

Geo-mean result of the four groups (Table 6) is shown in the Fig. 11. For all of the groups, shifting index towards left hurts the performance, because using more significant bits that have lower variations causes load imbalance issue among cache sets. For applications with relatively higher locality and high miss rate (Group D), changing the index does not hurt the performance much. This is because these applications tend to access memory densely but do not reuse much. After XORing the shared tag with index, the original data mapping is permuted. Most of the applications can have more balanced cache sets. Only Group D have a reduced performance, and it is because new set conflicts are created after the permutation. Having a fixed merged block and an SPP can effectively reduce the useless data stored in the cache, the effective capacity can be significantly increased for the applications that have low locality (Group A & B). Therefore, the performance can be significantly improved. However, for the applications with low miss rate (Group A & C), merging has less speedup because these applications are not sensitive to capacity increase. Variable size merged block can further improve space utilization because it can effectively solve the fragmentation issue. Grouped line-fill can improve the performance by increasing the bandwidth utilization of L2 line-fill. All of the blocks in the same merged block need to wait at the memory controller until the merged block is ready to be grouped and filled into L2. However, this additional latency does not hurt the performance much as shown in the Fig. 11. Optimized SPP has more performance improvement for applications with poor locality (Group A & B) because improving the SPP accuracy can further increase the effective capacity. Modified FR-FCFS prioritizes the memory requests in the transaction queue that have been decided to be merged based on the merge logic. For most of the applications, the performance are not changed much. This is because Scrabble cache tends to merge partial blocks in the same DRAM row, which does not affect the FR-FCFS scheduling order for most of the cases.

*Performance Comparison.* A performance comparison of Scrabble, other fine-grained caches, and the baseline is shown in the Fig. 12a. Amoeba cache, mDSC, Elastic cache, the proposed Scrabble cache with the fixed size of merged blocks, and variable sizes of merged blocks can achieve on an average of 8.8, 3.7, 6.2, 10.2, and 16.7 percent of performance improvement respectively as compared to baseline. Having variable size of merged blocks can achieve better performance because increased effective capacity. For applications that have poor spatial locality (e.g., art, radix), the performance improvements of Amoeba cache and Scrabble cache are above 40 percent. Even for applications with high spatial locality(e.g., cg, mcf), Scrabble cache achieves better performance than other fine-grained caches do. This is because the grouped line-fill can effectively increase the bandwidth utilization when writing multiple blocks into the L2 cache.

## 6.3 Energy

Amoeba, mDSC, Elastic cache, and the proposed fine-grained cache with fixed merged block and variable merged

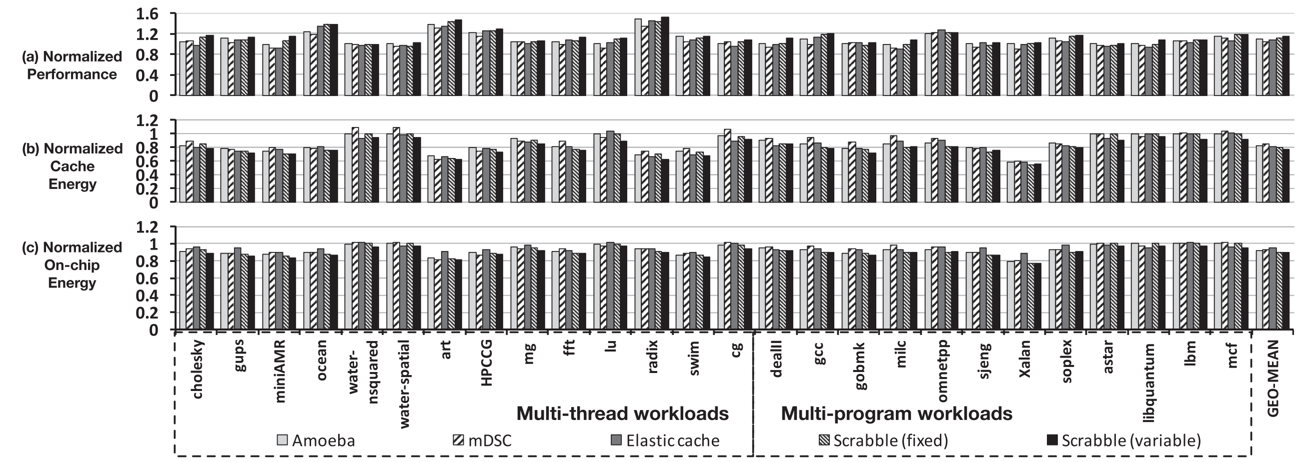


Fig. 12. Normalized performance, cache energy, and total on-chip energy comparison.

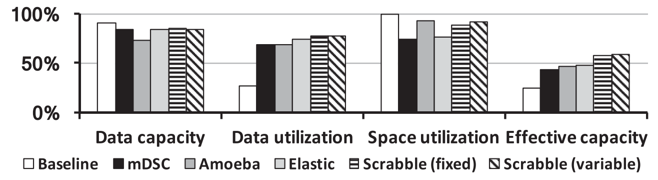


Fig. 13. Normalized effective capacity (GEO-MEAN).

blocks can reduce cache energy by 17.8, 15.2, 14.5, 19.0, and 24.3 percent respectively as shown in Fig. 12b and reduce the total on-chip energy by 9.2, 5.5, 5.2, 10.1, and 11 percent respectively as shown in Fig. 12c. Amoeba, mDSC, and Elastic cache reduce the granularity of the cache blocks, thereby reducing the cache energy by transferring and storing fewer words. However, tag accesses in Amoeba cache need to go through the data array, which costs more energy as compared to accessing a dedicated tag array. Modified decoupled sector cache uses a dedicated tag array but increases the size of the tag and adds selection tag in the data array, which increases power consumption. Elastic cache keeps fine-grained memory accesses, which consumes more energy for the applications that have good locality. For applications such as art and radix, which have poor spatial locality, the cache energy can be significantly reduced by these three fine-grained caches. Scrabble cache has a dedicated tag array and combines partial blocks at the memory controller. It can adaptively change the size of the merged block to maximize the space utilization and the data movement efficiency. In addition, Scrabble cache achieves the best performance as compared to other fine-grained caches. Grouped line-fill can significantly improve the bandwidth utilization and reduce control overhead. Therefore, Scrabble cache can reduce the cache energy the most.

### 6.4 Effective Capacity

As mentioned in the Section 2.1, the effective capacity is determined by both metadata overhead, data utilization, and space utilization. For all of the fine-grained caches, they increase the effective capacity improving one or multiple of these three factors. The mDSC, Amoeba, Elastic and Scrabble with fixed and variable merged blocks achieve an average of 1.75x, 1.91x, 1.93x, 2.34x, and 2.41x effective capacity improvement (Fig. 13).

*Metadata Overhead*. In fine-grained caches, maintaining smaller blocks typically requires a higher tag overhead as compared to conventional caches. Amoeba cache needs a word-sized tag even for the partial block that only has one word. The data capacity percentage of Amoeba is the smallest because the tag capacity is significantly increased when the applications have poor locality. The mDSC have a reduced data capacity as compared to the baseline because the selection tags require additional space. Elastic cache merges multiple fine-grained blocks, the data capacity is higher than Amoeba cache has. In Elastic cache, the smallest granularity is 16 B; whereas Scrabble support 8 B word granularity. The data capacity of Scrabble is similar to it is in the Elastic cache. This is because the proposed fine-grained cache carefully selects the private tag and index bits to merge as many fine-grained blocks as possible into one block.

*Data Utilization*. The data utilization for all of the fine-grained caches are improved as compared to it is in the conventional caches. Amoeba and mDSC improved the data utilization by adding a spatial pattern predictor to reduce data over-fetching. However, they have to fetch contiguous data within a physical block, which prevents further improvement on the data utilization. Elastic cache has a reduced cache block size to improve the data utilization. The proposed cache has the best data utilization because the spatial pattern predictor is optimized to support non-contiguous data fetching.

*Space Utilization*. For the conventional caches, the space utilization is the highest, because both the tag array and data array are equally partitioned to store fixed size blocks. Amoeba cache has a relatively high space utilization because each fine-grained block can be placed in any available space in a set together with its tag. mDSC has a relatively low space utilization because fine-grained blocks can be stored only in space allocated to its corresponding sector. Elastic cache has a low space utilization because the constraint on which fine-grained blocks can be merged results in empty space in merged blocks. Scrabble with fixed merged block has a relatively high space utilization because the selection of private tag and the index can allow more fine-grained blocks to be merged. Moreover, having merged blocks with variable length can further improve the space utilization as compared to the fixed one does. This is because when there are not enough blocks that can be merged, a small merged block is used to avoid wasting space.

*Control Overhead*. The proposed fine-grained cache merges different partial blocks into a merged block, then transfers and stores them together. For writing into the L2 cache from the DRAM, a merged block usually contains more data as compared to a single partial block does. An average of 26.9, 26.6, and 12.2 Bytes of effective data in a partial block are transferred respectively by each of the Amoeba, mDSC, and Elastic cache L2 refill. Amoeba cache and mDSC has similar effective data per block because they used the same spatial pattern predictor. Elastic cache increases the control overhead significantly by keeping fixed fine-grained memory accesses. Scrabble cache transfers an average of 34.5 Bytes per line-fill with a fixed merged block size, and an average of 38.2 Bytes per line-fill with variable merged block sizes. The control overhead of writing into the L2 cache is amortized by transferring more data for each line fill. However, the proposed fine-grained cache also introduces new control overhead. As mentioned in Section 7.3, a partial block is read from L2 to L1. In the proposed design, one physical block could be distributed into different merged blocks. In this case, more control signals will be generated. Fortunately, the cases that one physical block is distributed into multiple merged blocks is less than 1 percent for the evaluated applications.

## 7 Sensitivity Study

### 7.1 Private Tag Length

As shown in the Fig. 14, the performance speedup becomes saturated after the length of the private tag is increased to six bits for both L1 and L2 caches. The space utilization is increased when a longer private tag is used. This is because a longer private tag implies a shorter shared tag, which reduces the constraint on which fine-grained blocks can be merged. However, a longer private tag increases the size of tag array

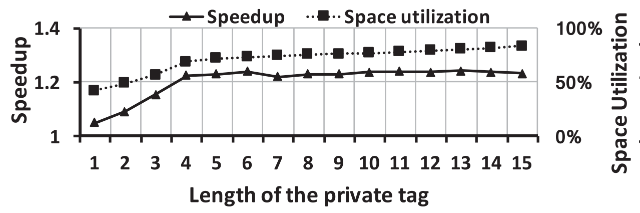


Fig. 14. Comparison of different private tag length.

and the access latency. When the private tag is longer than six bits, L2 miss latency is increased from 4 cycles to 5 cycles. Therefore, the length of the private tag is chosen to be six bits.

## 7.2 Spatial Pattern Prediction

A good spatial pattern predictor (SPP) should have a high coverage and a high accuracy. Coverage is the number of SPP hits divided by the total L2 misses. Accuracy is the number of correct predictions divided by the total predictions. An incorrect prediction could lead to either partial block miss (false negative) or over-fetch (false positive). Partial block miss happens when the SPP predict to fetch part of the block, however, words in the unfetched part of the block are required. Partial block miss requires an additional fetch to the same block. Over-fetch means some fetched words are not used before eviction. Both scenarios increase the data movement. Over-fetch wastes bandwidth and cache space, while the partial block miss leads to latency and energy overhead. In this section, the region size, update policy, pattern representation, and accuracy of SPP are studied.

### 7.2.1 *Region Size*

The proposed SPP table is a 128-set, 8-way cache indexed by the MSB of the regionID (similar with Amoeba cache [8]). The region size can influence the SPP efficiency. For the same prediction table size, a larger region leads to a higher coverage. However, the same spatial pattern is less likely to be shared across a large region. Using a large region size can increase the inaccurate prediction. The proposed design chooses a 4 KB region size because it leads to the highest system speed up and the second highest bandwidth reduction as shown in Fig. 15a.

### 7.2.2 *Update Policy*

When a merged block is evicted from the L1 cache, its access pattern needs to be recorded in the SPP. If the region address for the evicted block miss in the SPP, an SPP line is allocated. If the region address hits, two different update policy can be applied. One is to replace the previous pattern

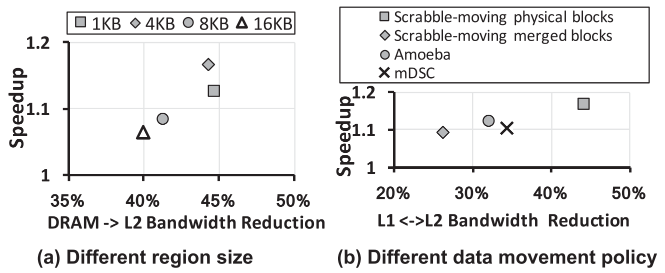


Fig. 15. Comparison of different design choice for SPP and data movement between L1 and L2.

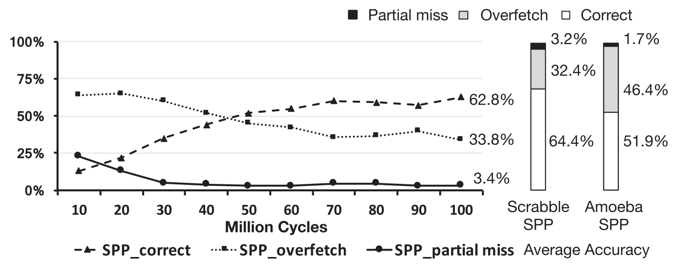


Fig. 16. SPP accuracy and warm-up time.

with the new pattern for the same region, the other is to apply a bitwise OR of the new pattern and the existing pattern. SPP design in Amoeba cache uses the bitwise-OR approach. Results show that, for the proposed cache, replacing the previous pattern with the new pattern has 1.167× speedup and 44.29 percent higher bandwidth reduction as compared to baseline, which has 3.3 percent more speedup and 7.89 percent more bandwidth reduction as compared to bit-wise OR approach. This is because adding different patterns together for the same region and fetch them all will increase the bandwidth consumption and pollute the cache due to over-fetch. Experiment results show that the bit-wise OR approach will lead to 26 percent more over-fetch while reduce only 1.5 percent partial miss. Therefore, Scrabble chooses the replacement approach when updating the spatial pattern predictor.

### 7.2.3 *Pattern Representation*

In the proposed SPP design, bitmaps are used to record the spatial pattern information. This is different from the SPP design in Amoeba cache, where saturation counters are used to indicate the start and end point of a spatial pattern. Amoeba cache only predicts and fetch consecutive words, while the proposed design can predict and fetch any words in a block, which can save bandwidth and improve the system performance. The Amoeba cache has a 32.16 percent bandwidth reduction and has a 1.122 × speedup compared to the baseline, while Scrabble reduces 44.29 percent bandwidth and has a 1.167× speedup as compared to the baseline.

### 7.2.4 *Accuracy Analysis*

As shown in Fig. 16, the proposed SPP takes around 70 million cycles to warm-up, which is about 5 percent of the application executing time (average execution time is 1.5 billion cycles). SPP partial miss, which can decrease the performance, reduces to less than 4 percent after 40 million cycles. The SPP has an overall of 64.4 percent average accuracy, where the prediction is exactly correct. There are 32.4 percent cases, where some unused data is fetched. After warm-up, the average SPP hit rate is around 90 percent. When SPP miss occurs, the entire physical block is requested.

## 7.3 Data Movement Between L1 and L2

In the proposed design, data movement between a private L1 cache and a shared L2 cache is at the granularity of a physical block instead of an entire merged block. An example is shown in Fig. 17. Word A, B, C, D in physical block 1 are distributed into multiple merged blocks in the L2 cache.

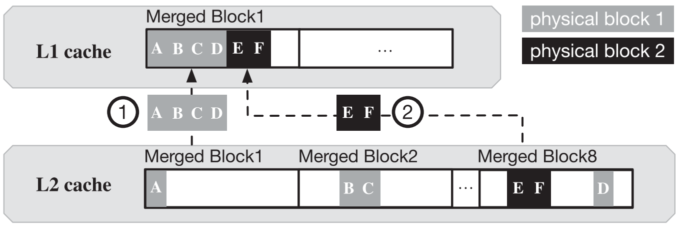


Fig. 17. An example of data movement between L1 and L2.

If the requested word A misses in L1 and hits in L2, ①all of the words belonging to physical block 1 (L2 block hit) will be moved to L1 together. The number of words to be moved is a physical block is typically fewer than it is in a merged block. Fig. 15d shows a comparison between moving the physical block and merged block. Moving physical block provides the opportunity for future merged block refill. As shown in Fig. 17, if word E in physical block 2 misses in L1 later, ②E and F are moved to L1 together. In this case, moving physical block can put A, B, C, D, E, F in the same L1 merged block. Since cache lookup among different blocks in the same cache set happens in parallel, finding words from the same a physical block does not cost additional latency. Therefore, the proposed design choose to move physical blocks between L1 and L2. Both L1 and L2 are write-back caches. When a merged block gets evicted from the L1 cache, instead of writing back the entire merged block, the proposed design only write back dirty physical blocks within the merged block in order to save bandwidth for the system that has fine-grained DRAM [36], [37]. In the proposed design, each merged block is composed of up to eight physical blocks. To support the partial write back for each physical block, 56 additional dirty bits per set are needed.

### 7.4 Last Level Cache Sizes

The proposed cache organization is applicable to mobile platform, which tends to have more cores with a relatively smaller LLC due to the area and energy constraint [38]. This section presents the performance of scrabble cache on systems with smaller LLC. As shown in the Fig. 18, for most of the evaluated fine-grained caches, the performance improvement are greater when the system has only 1 MB or 2 MB L2. This is because improving the cache capacity can be more effective when the application working sets do not fit in the cache. Some of the evaluated benchmarks have relatively small working sets. When the last level cache of the baseline has a large capacity, the working sets can already fit in and hence do not benefit from increasing the effective capacity.

There are many applications that are not cache capacity sensitive. Improving the cache effective capacity doesn't help much on those applications. Therefore, in the general purpose CPU design, it is difficult to achieve more than 30 percent of average performance improvement. A performance upper bound for fine-grained caches is estimated by increasing the cache capacity to hold equivalent data of 100 percent space utilization of the baseline cache, in which the tag array is compressed using the C-PACK+Z algorithm [13], [39]. As shown in Fig. 18, Scrabble cache has the highest speedup and is close to the upper bound.

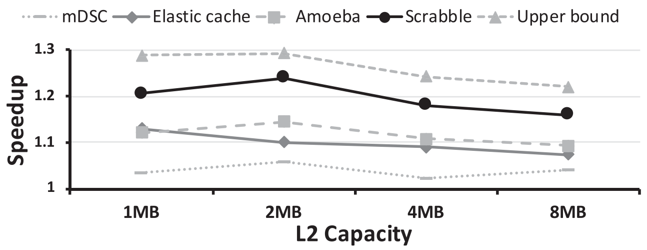


Fig. 18. Speedup on different LLC sizes.

## 8 Conclusion

In this paper, a new fine-grained cache, Scrabble, is proposed, which improves effective cache capacity by co-optimizing metadata overhead, data utilization, and space utilization. The effective capacity is significantly improved as compared to other fine-grained caches. Scrabble cache also uses grouped line-fill and optimized spatial pattern predictors to reduce control overhead and improve data movement efficiency. As a result, scrabble cache achieves better performance and lower energy consumption.

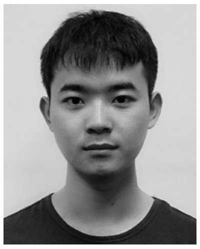
## Acknowledgments

This material is based upon work supported by the National Science Foundation at Lehigh University under Grant CCF-1750826. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

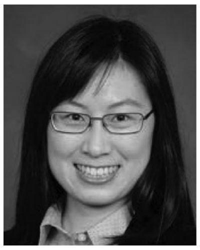
## References

[1] J. B. Rothman and A. J. Smith, "Sector cache design and performance," in *Proc. 8th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2000, pp. 124–133.
[2] M. K. Qureshi, M. A. Suleman, and Y. N. Patt, "Line distillation: Increasing cache capacity by filtering unused words in cache lines," in *Proc. IEEE 13th Int. Symp. High Perform. Comput. Archit.*, Feb. 2007, pp. 250–259.
[3] A. Seznec, "Decoupled sectored caches: Conciliating low tag implementation cost," in *Proc. 21st Annu. Int. Symp. Comput. Archit.*, 1994, pp. 384–393.
[4] P. Pujara and A. Aggarwal, "Increasing the cache efficiency by eliminating noise," in *Proc. 12th Int. Symp. High-Perform. Comput. Archit.*, 2006, pp. 145–154.
[5] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting cache line size to application behavior," in *Proc. 13th Int. Conf. Supercomputing*, 1999, pp. 145–154.
[6] D. Sanchez and C. Kozyrakis, "Scalable and efficient fine-grained cache partitioning with vantage," *IEEE Micro*, vol. 32, no. 3, pp. 26–37, May/Jun. 2012.
[7] C.-C. Huang and V. Nagarajan, "Increasing cache capacity via critical-words-only cache," in *Proc. 32nd IEEE Int. Conf. Comput. Des.*, 2014, pp. 125–132.
[8] S. Kumar, H. Zhao, A. Shriraman, E. Matthews, S. Dwarkadas, and L. Shannon, "Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2012, pp. 376–388.
[9] B. Li, J. Sun, M. Annavaram, and N. S. Kim, "Elastic-cache: GPU cache architecture for efficient fine-and coarse-grained cache-line management," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 82–91.
[10] L. Li, A. B. Hayes, S. L. Song, and E. Z. Zhang, "Tag-split cache for efficient GPGPU cache utilization," in *Proc. Int. Conf. Supercomputing*, 2016, Art. no. 43.
[11] H. Guo, L. Huang, Y. Lü, S. Ma, and Z. Wang, "Dycache: Dynamic multi-grain cache management for irregular memory accesses on GPU," *IEEE Access*, vol. 6, pp. 38881–38891, 2018.

[12] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos, "Accurate and complexity-effective spatial pattern prediction," in *Proc. 10th Int. Symp. High Perform. Comput. Archit.*, 2004, pp. 276–287.
[13] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 331–342.
[14] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," *ACM SIGARCH Comput. Archit. News*, vol. 34, pp. 252–263, 2006.
[15] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 69–80, 2009.
[16] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the dash multiprocessor," in *Proc. 17th Annu. Int. Symp. Comput. Archit.*, 1990, pp. 148–159.
[17] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," *ACM SIGARCH Comput. Archit. News*, vol. 11, no. 3, pp. 124–131, 1983.
[18] H. Zhao, A. Shriraman, S. Kumar, and S. Dwarkadas, "Protozoa: Adaptive granularity cache coherence," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 547–558, 2013.
[19] Intel, "Intel Xeon Processor Scalable Family Datasheet," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/processors/xeon/scalable/xeon-scalable-datasheet-vol-1.html
[20] Samsung, "4 Gb D-die DDR4 SDRAM," 2016. [Online]. Available: http://www.samsung.com/semiconductor/global/file/product/2016/03/DS_K4A4G085WD-B_Rev17-0.pdf
[21] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," Jan. 2005. [Online]. Available: http://sesc.sourceforge.net
[22] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optimization*, vol. 14, pp. 14:1–14:25, Jun. 2017.
[23] Cadence, "Cadence genus synthesis solution," 2019. [Online]. Available: https://www.cadence.com/content/dam/cadencewww/global/en US/documents/tools/digital-design-signoff/genus-synthesis- solution- ds.pdf
[24] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *Proc. IEEE Int. Conf. Microelectronic Syst. Edu.*, Jun. 2007, pp. 173–174.
[25] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative computing with STT-MRAM," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 189–200.
[26] A. KleinOsowski and D. J. Lilja, "Minnespec: A new SPEC benchmark workload for simulation-based computer architecture research," *IEEE Comput. Archit. Lett.*, vol. 1, no. 1, pp. 7–7, Jan.-Dec. 2002.
[27] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2009, pp. 469–480.
[28] S. Hambrusch, T. Mudge, J. Clarke, K. Bannerjee, and P. Chi-ang, "NSF workshop on emerging technologies for interconnects," Washington, DC, Feb. 2-3, 2012.
[29] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC challenge (HPCC) benchmark suite," in *Proc. 2006 ACM/IEEE Conf. Supercomputing*, vol. 213, Citeseer, 2006.
[30] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Symp. Comput. Architecture*, 1995, pp. 24–36.
[31] Standard Performance Evaluation Corporation, "SPEC OMP2001," 2001. [Online]. Available: https://www.spec.org/omp2001/
[32] Standard Performance Evaluation Corporation, "SPEC CPU2006," 2006. [Online]. Available: https://www.spec.org/cpu2006/
[33] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS parallel benchmarks—summary and preliminary results," in *Proc. ACM/IEEE Conf. Supercomputing*, 1991, pp. 158–165.
[34] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2009–5574, 2009.
[35] A. A. Nair and L. K. John, "Simulation points for spec cpu 2006," in *Proc. IEEE Int. Conf. Comput. Des.*, 2008, pp. 397–403.
[36] C. Zhang and X. Guo, "Enabling efficient fine-grained dram activations with interleaved I/O," in *Proc. Int. Symp. Low Power Electron. Des.*, Jul. 2017, pp. 1–6.
[37] Y. Lee and S. Kim, "Partial row activation for low-power DRAM system," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 217–228.
[38] ARM, "Arm CortexA75 Core Technical Reference Manual," 2018. [Online]. Available: https://developer.arm.com/docs/100403/latest
[39] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas, "C-Pack: A high-performance microprocessor cache compression algorithm," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 18, no. 8, pp. 1196–1208, Aug. 2010.


**Chao Zhang** received the BE degree in electronic science and technology from the University of Electronic Science and Technology of China, in 2016. He is working toward the PhD degree in the Department of Electrical and Computer Engineering, Lehigh University. He is a student member of the IEEE.

**Yuan Zeng** received the BE degree in electronic science and technology from Beijing Jiaotong University, China, in 2015. She is working toward the PhD degree in the Department of Electrical and Computer Engineering, Lehigh University. She is a student member of the IEEE.


**Xiaochen Guo** received the BE degree in computer science and engineering from Beihang University, in 2009, and the MS and PhD degrees in electrical and computer engineering from the University of Rochester, in 2011 and 2015. She is an assistant professor with the Department of Electrical and Computer Engineering, Lehigh University. Her research focuses on energy-efficient computer architectures, feature-rich memories, and computing platforms based on emerging technologies. She is a member of the IEEE.