Micro-Sector Cache: Improving Space Utilization in Sectored DRAM Caches

MAINAK CHAUDHURI, Indian Institute of Technology, Kanpur
MUKESH AGRAWAL, Intel Architecture Group
JAYESH GAUR and SREENIVAS SUBRAMONEY, Intel Microarchitecture Research Lab

Recent research proposals on DRAM caches with conventional allocation units (64 or 128 bytes) as well as large allocation units (512 bytes to 4KB) have explored ways to minimize the space/latency impact of the tag store and maximize the effective utilization of the bandwidth. In this article, we study sectored DRAM caches that exercise large allocation units called sectors, invest reasonably small storage to maintain tag/state, enable space- and bandwidth-efficient tag/state caching due to low tag working set size and large data coverage per tag element, and minimize main memory bandwidth wastage by fetching only the useful portions of an allocated sector. However, the sectored caches suffer from poor space utilization, since a large sector is always allocated even if the sector utilization is low. The recently proposed Unison cache addresses only a special case of this problem by not allocating the sectors that have only one active block.

We propose Micro-sector cache, a locality-aware sectored DRAM cache architecture that features a flexible mechanism to allocate cache blocks within a sector and a locality-aware sector replacement algorithm. Simulation studies on a set of 30 16-way multi-programmed workloads show that our proposal, when incorporated in an optimized Unison cache baseline, improves performance (weighted speedup) by 8%, 14%, and 16% on average, respectively, for 1KB, 2KB, and 4KB sectors at 128MB capacity. These performance improvements result from significantly better cache space utilization, leading to 18%, 21%, and 22% average reduction in DRAM cache read misses, respectively, for 1KB, 2KB, and 4KB sectors at 128MB capacity. We evaluate our proposal for DRAM cache capacities ranging from 128MB to 1GB.

CCS Concepts: • **Computer systems organization** → *Multicore architectures;*

Additional Key Words and Phrases: DRAM cache, sectored cache, space utilization

ACM Reference Format:
Mainak Chaudhuri, Mukesh Agrawal, Jayesh Gaur, and Sreenivas Subramoney. 2017. Micro-sector cache: Improving space utilization in sectored DRAM caches. ACM Trans. Archit. Code Optim. 14, 1, Article 7 (March 2017), 29 pages.
DOI: http://dx.doi.org/10.1145/3046680

1. INTRODUCTION

The recent advances in die-stacked and in-package embedded DRAM technologies have motivated the industry to explore DRAM caches as a viable option for designing very large last-level caches [Arroyo et al. 2011; IBM 2012; Intel 2013; Stuecheli 2013; Kurd et al. 2014]. Recent research studies exploring the architecture of the DRAM caches have focused on traditional cache organizations with fine-grain (e.g., 64 or 128 bytes) [Loh and Hill 2011; Meza et al. 2012; Qureshi and Loh 2012; Sim et al.

Authors' addresses: M. Chaudhuri, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Uttar Pradesh 208016, India; email: mainakc@cse.iitk.ac.in; M. Agrawal, Intel Architecture Group, Hilsboro, OR 97124, USA; email: mukesh.agrawal@intel.com; J. Gaur and S. Subramoney, Intel Microarchitecture Research Lab, Bengaluru, Karnataka 560103, India; emails: {jayesh.gaur, sreenivas.subramoney}@intel.com.

DOI: http://dx.doi.org/10.1145/3046680

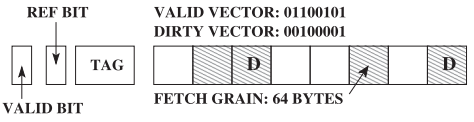

Fig. 1. A 512-byte sector along with its tag and states.

2012; El-Nacouzi et al. 2013; Hameed et al. 2013], coarse-grain (e.g., 512 bytes to 4KB) [Jiang et al. 2010; Jevdjic et al. 2013, 2014; Lee et al. 2015; Jang et al. 2016], or mixed-grain [Gulur et al. 2014] allocation units (referred to as the DRAM cache block size). There are other studies that explore a range of allocation units [Zhao et al. 2007] and configurable block sizes [Madan et al. 2009]. The studies assuming fine-grain or conventional allocation units focus their efforts on managing the latency and bandwidth of accessing the large tag store. Large allocation units significantly reduce the tag store size. The designs that exercise large allocation units fetch either the entire large block or only the useful portions of the large block on a cache miss. The designs in the second category aim to optimize the main memory bandwidth requirement and are usually referred to as sectored or sub-blocked caches. The allocation unit in these designs is referred to as a sector. Each sector is composed of a number of contiguous conventionally-sized cache blocks. The amount of data fetched from main memory on a demand miss is usually a cache block, the size of which is assumed to be 64bytes in this study. Figure 1 shows a 512-byte sector composed of eight 64-byte cache blocks, four of which are valid/occupied. Two of the occupied blocks are dirty (marked "D"). The sector tag, the reference (REF) bit needed by the not-recently-used (NRU) replacement policy, the sector-wide valid bit, and the valid and dirty vectors are also shown. An n-way sectored cache would have n such sectors in each cache set. The REF bit of a way is set on an access. When all the REF bits are marked one in a set, all the bits except the one corresponding to the way currently being accessed are reset. The replacement policy victimizes the way with the smallest physical way id such that its REF bit is reset. We do not consider the least-recently-used (LRU) replacement algorithm in this study because of its significantly larger replacement state overhead in terms of storage as well as access/update bandwidth compared to the NRU replacement algorithm.

The Unison cache is a recently proposed sectored DRAM cache design exercising a small associativity [Jevdjic et al. 2014]. The tag/state and other metadata of a set are co-located with the sectors that constitute the set. On a lookup, the metadata necessary for deciding hit/miss in the target set are read out. To avoid tag access serialization, a way predictor is looked up and the requested data block is read out from the sector in the predicted way. On a misprediction, the data block is fetched from either the main memory or the correct DRAM cache way. On a sector miss, the Unison cache consults a sector footprint predictor employing a spatial memory streaming prefetcher [Somogyi et al. 2006] to prefetch the blocks within the sector that are likely to be accessed during the sector's residency in the cache.

The early proposals of the sectored cache were motivated by the reduction in the SRAM storage required for tag/state [Liptay 1968; Goodman 1983; Hill and Smith 1984; Przybylski 1990; Moore 1993; Windheiser et al. 1993; Tremblay and O'Connor 1996; Rothman and Smith 2000]. However, the recent sectored DRAM cache designs, such as the Unison cache, allocate tag/state in DRAM. As a result, the savings in the tag/state storage achieved by the sectored DRAM cache designs compared to the designs that use small allocation units may no longer serve as a significant motivation for designing sectored DRAM caches. However, the small metadata working set of sectored DRAM caches and the large data coverage (one sector worth of data) per metadata element enable space- and bandwidth-efficient design of SRAM structures

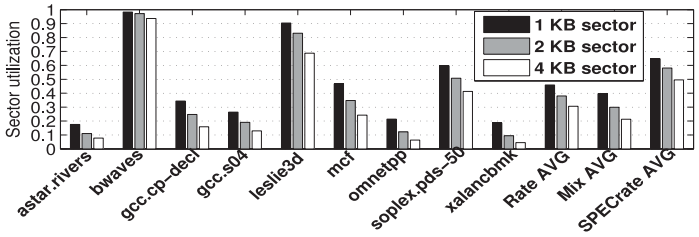

Fig. 2. Sector utilization in a 128MB 4-way Unison cache.

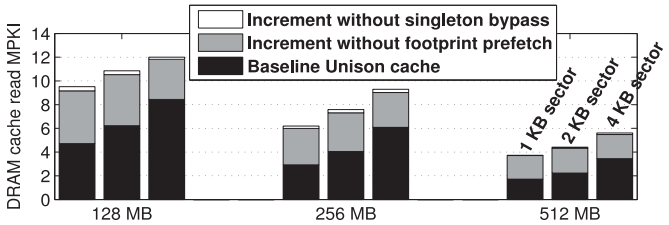

Fig. 3. DRAM cache read MPKI. Each group of bars corresponds to one DRAM cache capacity point and the bars in a group correspond to 1KB, 2KB, and 4KB sector sizes.

for metadata caching. We quantitatively show the importance of metadata caching while designing an efficient sectored DRAM cache baseline for this study in Section 3. We will also revisit this aspect in Section 5 while comparing our proposal with the DRAM caches exercising small allocation units.

One major drawback of the sectored caches is that they often suffer from poor performance due to low space utilization arising from the unoccupied cache blocks in a sector. For example, in Figure 1, the sector is only 50% utilized. The Unison cache addresses only a special case of this problem by not allocating the sectors that are predicted to have a single-block footprint (this optimization will be referred to as singleton sector bypass). Figure 2 shows the sector utilization of nine SPEC CPU 2006 workloads in a 128MB 4-way Unison cache when 16 copies of each workload (16-way rate mode) are run on a simulated 16-core system with a four-level cache hierarchy.[1] The DRAM cache is the last-level (L4) cache.[2] The sector utilization data are computed by taking a snapshot of the utilization of the sectors resident in the L4 cache every 500K L4 cache read lookups and averaging the utilization values over all the snapshots. The rate average group of bars (Rate AVG) shows that, on average, a 1KB sector is utilized 46% and a 4KB sector is utilized only 31% for these nine homogeneous 16-way multi-programmed workloads. The average sector utilization for a bigger set of 30 16-way multi-programmed workloads prepared by mixing the nine workloads is shown in the mix average group of bars (Mix AVG). This average shows that a 1KB sector is 40% occupied and a 4KB sector is only 21% utilized, on average. The rightmost group of bars (SPECrate AVG) shows the average sector utilization for a larger set of 28 SPEC CPU 2006 workloads when each of these is executed in 16-way rate mode. This average shows that a 1KB sector is 65% utilized and a 4KB sector is only 50% utilized.

Figure 3 further quantifies the L4 cache read misses per kilo instructions (MPKI) as a function of the L4 cache capacity and the sector size.[3] The results are averaged over

[1]We explore a larger set of 28 SPEC CPU 2006 workloads in Section 5.

[2]Simulation methodology is discussed in Section 2.

[3]In this study, we focus only on read MPKI, since read MPKI correlates well with performance. Write MPKI is usually very low and has little relation with performance.

30 16-way multi-programmed workloads prepared by mixing the nine SPEC CPU 2006 workloads. The lower section of a bar corresponds to the average read MPKI of the Unison cache. The middle portion corresponds to the increment in average read MPKI when sector footprint prefetching is disabled. The upper portion corresponds to the additional increment in average read MPKI when singleton sector bypass is disabled. Even though Unison cache's footprint prefetching is able to save a significant volume of L4 cache read misses, across the board, with or without this optimization, the L4 cache read MPKI steadily increases with increasing sector size. For example, as the sector size increases from 1KB to 4KB in a 128MB Unison cache, the read MPKI increases from 4.7 to 8.4.

In this study, we focus our effort on improving the sector utilization and read MPKI of the sectored DRAM caches. Before embarking on our proposal, we devote careful attention to designing an optimized version of the Unison cache and use it as the baseline for our study (Section 3). We begin our exploration of how to improve sector utilization by understanding the shortcomings of the decoupled sectored (DS) cache, which is one of the few notable efforts to improve the performance of SRAM sectored caches [Seznec 1994]. In Sections 4.1 and 4.2, we design and evaluate a DRAM cache architecture employing the idea of the DS cache. Sections 4.3, 4.4, and 4.5 present the crux of our proposal, which builds on top of the DS DRAM cache. We propose a novel sectored cache organization that systematically incorporates flexibility and locality information in the sector allocation and sector replacement algorithms. The simulation results show that our DRAM cache proposal significantly improves the sector utilization and outperforms the baseline (Sections 4.6 and 5). A summary of our contributions is presented in the following.

—We begin our study by designing a DRAM cache employing the DS organization, which was originally proposed for improving the space utilization in sectored SRAM caches. This design brings out two major shortcomings of the DS organization: (a) high metadata overhead and (b) thrashing within a physical sector frame.
—We simultaneously address both the shortcomings of the DS organization by proposing a sectored DRAM cache design that employs a novel allocation and replacement unit called micro-sector. The proposed Micro-sector cache incorporates flexibility in mapping data within a sector, thereby significantly improving the sector utilization.
—We further augment the Micro-sector cache with a novel sector replacement algorithm that incorporates spatial locality of the sectors in the replacement decision.
—We present a detailed evaluation of the proposed Micro-sector cache for different sector sizes as the DRAM cache size is varied from 128MB to 1GB.

1.1. Related Work

The first study on DRAM caches with conventional small block sizes (e.g., 64 bytes) proposed to co-locate the tags and the data blocks of a highly associative set in a single DRAM page [Loh and Hill 2011]. This proposal suffers from a relatively large DRAM cache hit latency due to the compound access required to read out all the tags of a set and then the data, if a hit is detected. Also, spatial streaming through consecutive sets requires opening a new DRAM page for each set. This proposal incorporates a multi-megabyte SRAM structure (called MissMap) to detect DRAM cache misses quickly. A subsequent proposal replaces the MissMap structure with a DRAM cache hit/miss predictor and dynamically manages the congestion at the DRAM cache by forcing a subset of the DRAM cache hits to fetch data from the main memory [Sim et al. 2012]. Further studies have explored different designs of the DRAM cache hit/miss predictors [El-Nacouzi et al. 2013]. The state-of-the-art direct-mapped Alloy cache significantly shortens the critical path of a hit by maintaining the tag and data (TAD) of a

cache block together and reading out a TAD through a single CAS operation [Qureshi and Loh 2012]. This proposal also improves row buffer locality by contiguously allocating the TADs from consecutive sets. This design incorporates simple hit/miss predictors to speed up miss handling. However, the Alloy cache requires a custom burst length of at least five for reading out a TAD through a single CAS in a system with 128-bit DDR channels. A significant fraction of the DRAM cache bandwidth is spent in communicating the tag with each transaction. The bandwidth-efficient architecture (BEAR) for DRAM caches explores a few optimizations to improve the Alloy cache [Chou et al. 2015]. Different functions for mapping the cache sets into the DRAM pages with the goal of optimizing hit latency as well as hit rate have also been proposed [Hameed et al. 2013]. On-die SRAM storage for the most recently used tags [Meza et al. 2012; Huang and Nagarajan 2014] and space-efficient storage of tags exploiting spatial locality in access stream and metadata compression [Franey and Lipasti 2015] for DRAM caches with small allocation units have been explored. In contrast, our proposal deals with the DRAM caches with large allocation units called sectors and explores optimizations to improve the space-efficiency of storing DRAM cache data. The storage needed for the metadata in the sectored DRAM caches is already small, unlike the DRAM caches with small allocation units.

DRAM cache architectures with large allocation units fetch either a large block or the necessary/demanded portions of a large block. The former design can lower the volume of misses due to the inherent prefetching effect enabled by the large blocks [Jiang et al. 2010; Gulur et al. 2014; Lee et al. 2015]. To avoid wasting the main memory bandwidth in these designs, identification and caching of only hot DRAM pages [Jiang et al. 2010] and dynamic selection between large and small blocks [Gulur et al. 2014] have been explored. The sectored DRAM caches fetch only the necessary/demanded portions of a large allocation unit. Sectored cache designs with an on-die tag cache have been explored [Zhang et al. 2004]. The impact of different sector sizes has been documented [Zhao et al. 2007]. The more recent Footprint cache proposal explores the possibility of predicting and fetching only those 64-byte blocks within a sector that are likely to be used during the sector's residency in the DRAM cache [Jevdjic et al. 2013]. The Unison cache proposal employs the Footprint cache's predictor while co-locating the metadata and the sectors of a set [Jevdjic et al. 2014]. The tagless DRAM cache design with OS page-sized blocks [Lee et al. 2015] has been recently extended with the Footprint cache's predictor to save main memory bandwidth [Jang et al. 2016]. This design requires OS modifications for maintaining the sector locations in the appropriately extended page table entries. Our proposal can significantly improve the space utilization of such a DRAM cache employing OS page-sized (i.e., at least 4KB) sectors.

The DS cache, proposed in the context of SRAM caches, aims at improving the sector utilization by allowing multiple sectors to share a group of data-associative physical sector frames [Seznec 1994]. The cache blocks at a particular position k of N sectors could be allocated to the cache block positions k of A_d data-associative physical sector frames within a set (set-associativity is an integral multiple of A_d). As a result, one physical sector frame can be filled by cache blocks from multiple different sectors.

Another proposal for improving sector utilization in sectored SRAM caches organizes all the cache blocks (called sub-sectors in the proposal) of all the sectors in a set to form a pool of cache blocks [Rothman and Smith 1999]. Each cache block can be dynamically assigned to any physical sector way within a set based on the need.

2. SIMULATION FRAMEWORK

We use the Multi2Sim simulator [Ubal et al. 2012] to model 16 dynamically scheduled out-of-order issue x86 cores clocked at 4GHz. The main memory DRAM array and the

Table I. Simulation Environment

On-die cache hierarchy and interconnect
Per-core caches: iL1, dL1: 32KB, 8-way, 2 cycles; Unified L2: 256KB, 8-way, 5 cycles; Shared L3 cache: 16MB, 16-way, 16 banks, 7 cycles per bank; 64-byte blocks, LRU; Interconnect: 4 × 4 mesh, single-cycle hop; Each hop: a core, its L1 and L2 caches, one L3 cache bank.
Main memory
Memory controllers: two single-channel DDR3-1600 at two opposite corners of mesh; FR-FCFS, reads served before writes in an open row, read-activates have priority over write-activates; 11-11-11-28, 64-bit channels, 2 ranks/channel, 8 banks/rank, 1KB row/bank/device, BL = 8, open-page policy; Command queues per bank: 16-entry read, 16-entry write.
DRAM cache (Non-inclusive L4 cache)
Allocates write misses, no back-invalidation on eviction; 1.6GHz (effective rate of 3.2 GT/s); FR-FCFS, reads served before writes in an open row, read-activates have priority over write-activates; 11-11-11-28, four 128-bit DDR channels, 1 rank/channel, 16 banks/rank, open-page policy, BL = 4, 8KB row/bank/device [Jevdjic et al. 2014] (see discussion in Section 3); Command queues per bank: 32-entry read, 32-entry write.

Table II. Workload Mixes

MIX1–MIX9: one app. × 16 copies of each
astar.rivers, bwaves, gcc.cp-decl, gcc.s04, leslie3d, mcf, omnetpp, soplex.pds-50, xalancbmk
MIX10–MIX20: two apps × eight copies of each
(astar.rivers, bwaves), (astar.rivers, gcc.cp-decl), (astar.rivers, gcc.s04), (astar.rivers, leslie3d), (gcc.cp-decl, omnetpp), (gcc.cp-decl, xalancbmk), (gcc.s04, omnetpp), (mcf, omnetpp), (mcf, xalancbmk), (omnetpp, soplex.pds-50), (omnetpp, xalancbmk)
MIX21–MIX28: four apps × four copies of each
(astar.rivers, gcc.cp-decl, mcf, omnetpp), (gcc.cp-decl, gcc.s04, omnetpp, xalancbmk), (gcc.s04, gcc.cp-decl, mcf, soplex.pds-50), (gcc.cp-decl, leslie3d, omnetpp, xalancbmk), (mcf, omnetpp, xalancbmk, gcc.cp-decl), (gcc.s04, gcc.cp-decl, omnetpp, soplex.pds-50), (gcc.cp-decl, gcc.s04, soplex.pds-50, xalancbmk), (gcc.cp-decl, omnetpp, soplex.pds-50, xalancbmk)
MIX29–MIX30: eight apps × two copies of each
(all excluding bwaves), (all excluding gcc.s04)

DRAM cache array are modeled using DRAMSim2 [Rosenfeld et al. 2011]. Additional details appear in Table I. The aggregate bandwidths of the main memory and the DRAM cache are 25.6GB/s and 204.8GB/s, respectively.

We select 28 application-input combinations from the SPEC CPU 2006 suite spanning 24 different applications for this study. From these 28 combinations, we sample nine combinations representing a wide range of sector utilization. These were shown in Figure 2.[4] The workload set used for most of the execution-driven simulation studies in this article consists of 30 16-way multi-programmed mixes prepared from these nine application-input combinations. The mixes are prepared by drawing 1, 2, 4, or 8 different application-input combinations from the set of nine and replicating 16, 8, 4, or 2 copies of each of the drawn applications to fill up the 16 slots. Table II details these mixes. All applications use the ref input sets. If the application has multiple ref inputs, we mention the input(s) used with the application name (e.g., gcc, astar, and soplex). The L3 cache read MPKI of these mixes varies from 1.7 to 34.9 with an average of 13.3. Each workload mix commits at least eight billion dynamic

[4]We evaluate our proposal on the complete set of 28 application-input combinations in Section 5.

instructions (a representative segment of 500 million dynamic instructions for each thread [Sherwood et al. 2002]). Threads completing early continue to run past their representative segments. The weighted speedup results are measured based on the statistics collected during the first 500 million retired instructions of each thread. Before commencing the detailed simulation of the representative segment of 500 million dynamic instructions, each thread in a mix runs in a functional mode for 1.5 billion instructions to warm up the cache hierarchy (i.e., 24 billion warm-up instructions per mix).

3. BASELINE SECTORED DRAM CACHE ARCHITECTURE

In this section, we first outline the tag and data layout of our baseline sectored DRAM cache design. Next, we discuss a few optimizations that we incorporate in the baseline design. We, however, note that our contributions for improving the sector utilization are generic in nature and not tied to any specific baseline implementation.

3.1. Tag/Data Organization

We would like to place an OS page (4KB) worth of sequential data in one DRAM cache row so that a stream (maximum size limited by OS page) can enjoy row hits in all accesses after the row is opened. Therefore, $4/Y$ consecutive sets must be allocated in the same DRAM cache row where the sector size is Y KB. With an associativity of four (as in the Unison cache), such an organization requires a row size of 16KB. We use 8KB row buffers per bank per device with an open page policy, as in the Unison cache. An aggregate row buffer size of 16KB in a rank can be designed by having either two devices per rank with 64-bit output per device or one device per rank with support for two 64-bit pseudo-channels per channel as outlined in the second generation high bandwidth memory (HBM2) implementation from SK Hynix [Tran and Ahn 2014]. Although multiple devices per rank can lead to several design issues [Sim et al. 2013], it may not be difficult to support two devices in a rank.

As in the Unison cache design, we borrow 64 bytes from each sector for storing the metadata. In other words, the 1KB, 2KB, and 4KB sectors get reduced in size to 960, 1,984, and 4,032 bytes. However, we will continue to refer to them as 1KB, 2KB, and 4KB sectors. The metadata of an entire set is allocated before storing the first data way in a DRAM cache row. The DRAM cache row organization is shown in Figure 4 (an open row contains 16KB worth of data).[5] Since each DRAM cache lookup needs to access the metadata of the target set, we would like to limit the size of the critical metadata per set to 64 bytes or 128 bits per way for a four-way cache. This allows us to read out the set's critical metadata with a single CAS command (128-bit channel $\times$ burst length of four). The critical metadata of a set is the portion of the set's metadata that is needed for hit/miss detection. Each way of the Unison cache needs to maintain a tag (23 to 20 bits for 128MB to 1GB capacity assuming a 48-bit physical address), a tag valid bit, a valid vector, a dirty vector, a REF bit, a program counter, the offset of the address that allocated the sector, and the id of the allocating core. The first four fields are needed for hit/miss detection (hence, critical) and the last three fields are needed for indexing into the sector footprint prediction table at the time of updating the demanded footprint of a replaced sector (these are not needed for hit/miss detection and, hence, not critical). Both valid and dirty vectors are part of the critical metadata because the Unison cache uses these vectors to encode four states of a 64-byte block: invalid,

[5]We borrow the 8KB row buffer size per device from the Unison cache proposal for the ease of comparison. However, the JEDEC HBM standard supports only 2KB and 4KB row buffer sizes per device [JEDEC 2015]. These standard row buffer sizes can be seamlessly adopted by our proposal by allocating the metadata of a set and the data ways to different rows.

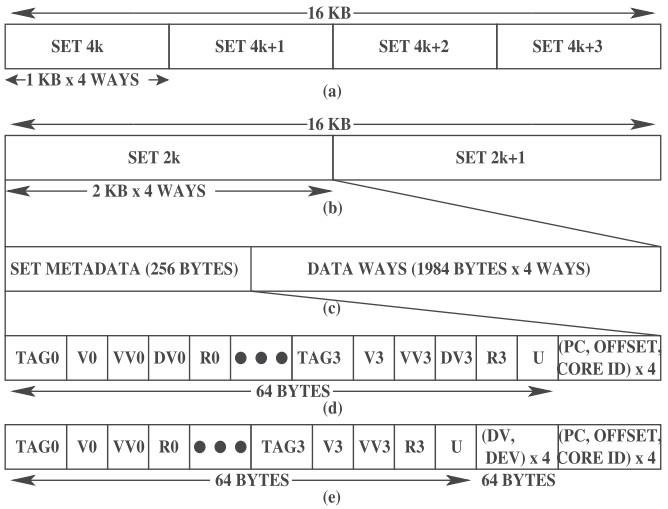

Fig. 4. (a) DRAM cache row layout when using 1KB sectors. (b) DRAM cache row layout when using 2KB sectors. (c) Set layout when using 2KB sectors. (d) Set metadata layout when using 2KB sectors. (e) Set metadata layout when using 4KB sectors. TAG0=Tag of way0, V0=Valid bit of way0, VV0=Valid vector of way0, DV0=Dirty vector of way0, R0=REF bit of way0, DEV=Demand vector, U=Unused.

valid and not demanded, valid and demanded and not dirty, valid and demanded and dirty. For sector sizes of 960 and 1,984bytes, the critical fields of a way can be stored within 128 bits. For 4,032-byte sectors, to be able to accommodate the critical metadata of a way within 128 bits, we decouple the dirty vector from the critical metadata by maintaining a separate demand vector (which is not critical) to track the demanded blocks in the sector (Figure 4(e)). Finally, consecutive logical rows are distributed across the DRAM cache channels. The rows mapped to a channel are distributed across the banks of the rank in that channel.

3.2. Optimizing Critical Access Path

The Unison cache needs to look up the metadata of the target set on every DRAM cache access to determine hit/miss and optionally update the NRU replacement states (the REF bits). With the help of a way predictor, the Unison cache can issue the data read CAS from the predicted way immediately following the metadata read CAS from the target set. The data fetched from the predicted way can be returned to the CPU after verifying the correctness of the prediction from the fetched metadata. To reduce the DRAM cache bandwidth consumption and queuing delays arising from a large volume of critical metadata and NRU replacement state reads/writes, our baseline design includes two small SRAM caches, namely, a tag cache [Wang et al. 1995; Meza et al. 2012; Gulur et al. 2014; Huang and Nagarajan 2014; Franey and Lipasti 2015; Chou et al. 2015] and an NRU state cache. The tag cache holds the critical metadata of the recently accessed DRAM cache ways. A tag cache hit provides the target way of the DRAM cache access and obviates the need to issue the metadata CAS command. On a tag cache miss, the critical metadata of the target DRAM cache set must be fetched. However, the tag cache entry that is allocated at this time stores the critical metadata of only the accessed DRAM cache way. Not storing the critical metadata of an entire DRAM cache set in a tag cache entry improves tag cache space efficiency and coverage because all the four ways of a DRAM cache set may not get used at the same time. Since a tag cache entry covers one sector worth of data, we can achieve a reasonable tag cache hit rate with a small tag cache. The NRU state cache holds the NRU states (four

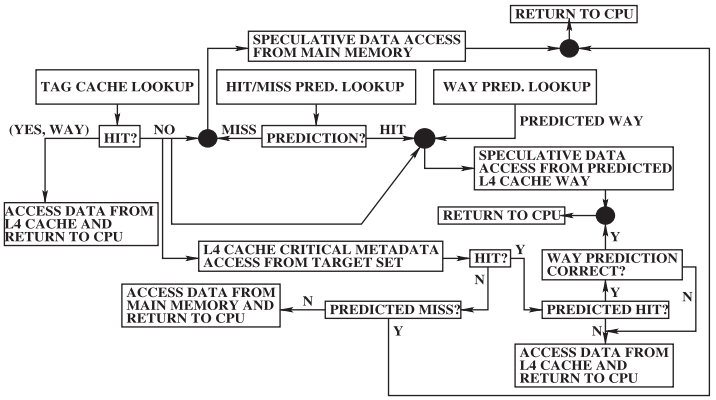

Fig. 5. Flow of a read access in the baseline L4 cache.

REF bits per set in the baseline four-way DRAM cache) of the recently accessed DRAM cache sets.[6] These two SRAM caches can improve the DRAM cache average hit latency by eliminating a large fraction of critical metadata and NRU state reads/writes. The resulting reduction in the total number of CAS commands leads to improvement in DRAM cache bandwidth consumption and drop in the overall queuing delays.

To speed up miss handling, the baseline DRAM cache incorporates the MAP-I hit/miss predictor [Qureshi and Loh 2012] and prioritization of the demand requests over the footprint prefetch requests. On an L3 cache read miss, the tag cache, the way predictor, and the hit/miss predictor are looked up in parallel, the critical path being determined by the structure having the longest latency (usually the tag cache). A tag cache hit provides the target L4 cache way. On a tag cache miss, a metadata read is enqueued in the L4 DRAM cache. Additionally, if the hit/miss predictor predicts a miss, a demand request is sent to the appropriate main memory controller. On the other hand, if the hit/miss predictor predicts a hit in the case of a tag cache miss, the outcome of the way predictor is used to initiate a speculative data access to the L4 DRAM cache. Figure 5 summarizes the flow of a read access in the baseline L4 cache. The correctness of a speculatively accessed data block must be verified with the help of metadata before returning the block to the CPU, as depicted in Figure 5. As a result, even if a speculative data access completes early, it must wait for the critical metadata fetch to complete. While the critical path of read accesses to the L4 cache can be shortened with the help of the hit/miss predictor and the way predictor, the write accesses that miss in the tag cache cannot make use of them and must be delayed until the critical metadata is read out. When a tag is replaced from the L4 cache, the tag cache entry, if present, is invalidated. On a tag cache entry replacement, if the entry is dirty, the corresponding set metadata is read out and updated. The replaced tag cache entry is also used to update the way predictor (this is the only time the way predictor is updated). The tagless direct-mapped way predictor employs address-based hashing for lookup.

Table III summarizes the SRAM overhead of the baseline L4 DRAM cache along with the latency through each of the SRAM structures and whether a structure is on the critical path of L4 cache lookup. The storage overhead calculation assumes a

[6]The LRU replacement policy would have required eight replacement state bits per set halving the coverage of the NRU state cache for a given storage investment. Our experiments with the LRU replacement policy show that this loss in coverage leads to significantly increased DRAM cache bandwidth consumption toward replacement state lookup and update. On the other hand, due to a significantly filtered temporal locality seen by the DRAM cache, the LRU replacement policy improves the MPKI marginally.

Table III. SRAM Overhead of Baseline L4 DRAM Cache

Structure	Number of entries, size, latency at 4GHz (for 22nm nodes)			On critical path?
	1KB sector	2KB sector	4KB sector	
Footprint table	16K, 110KB, 2 cycles	16K, 140KB, 2 cycles	16K, 206KB, 3 cycles	No
Singleton table	512, 3KB, 1 cycle	512, 3KB, 1 cycle	512, 3KB, 1 cycle	No
Way predictor	32K, 8KB, 1 cycle	32K, 8KB, 1 cycle	32K, 8KB, 1 cycle	Yes
Tag cache	32K, 256KB, 4 cycles	32K, 384KB, 4 cycles	16K, 320KB, 4 cycles	Yes
NRU state cache	32K, 40KB, 1 cycle	32K, 36KB, 1 cycle	32K, 32KB, 1 cycle	No
Hit/Miss predictor	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	Yes
Total size	418.5KB	572.5KB	570.5KB	

48-bit physical address. There are six SRAM structures that assist the L4 cache. The overhead of each of these is listed as the sector size varies from 1KB to 4KB. The tag cache is designed to have 32K entries (for 1KB and 2KB sectors) or 16K entries (for 4 KB sectors) and eight ways with a lookup latency of four cycles. It is indexed using the lower bits of the L4 cache set index. We use a direct-mapped NRU state cache with at most 32K entries. The NRU state cache is indexed using the lower bits of the L4 cache set index. The size of the tag associated with an NRU state cache entry is calculated assuming a 1GB 4-way L4 cache. The footprint table stores the already learned sector footprints. The singleton table stores the recently observed singleton sectors' tags so that any singleton misprediction can be corrected. The singleton table is updated with the sector tag of an L4 cache read miss when such a sector is predicted to be singleton and not allocated in the L4 cache. The hit/miss predictor uses a bank of 256 3-bit saturating counters per core, indexed using a program counter-based hash [Qureshi and Loh 2012]. Overall, the total SRAM overhead of the baseline L4 DRAM cache varies from 418.5KB to 572.5KB, which is around half of one L3 cache way. The tag cache, the way predictor, and the hit/miss predictor are looked up in parallel, and for our configuration, the tag cache latency is the dominant one determining the additional latency on the critical path. However, this additional latency of four cycles constitutes a small fraction of the overall L4 cache hit latency. Further, the savings in the average L4 cache hit latency achieved by the tag cache significantly exceed the loss due to tag cache lookup latency.

Figure 6 quantifies the benefits of the tag cache (TC), NRU state cache (NSC), MAP-I hit/miss predictor (HMP), and demand request prioritization (DP) for three capacity points and three sector sizes for each capacity point. All results are averaged over 30 multi-programmed mixes. As the L4 cache capacity is scaled up, the DRAM cache requires more bandwidth to serve an increased volume of hits. This makes saving the metadata bandwidth more important at bigger capacity points. The tag cache and the NRU state cache free-up precious L4 cache bandwidth that can now be used to serve hits more quickly. Therefore, as the L4 cache capacity increases, the performance benefits of the tag cache and the NRU state cache increase significantly. For 1KB, 2KB, and 4KB sector sizes, the tag cache experiences miss ratios of 31%, 23%, and 21%, respectively. Referring back to Table III, we observe that the maximum achievable data coverage of the tag cache for 1KB, 2KB, and 4KB sectors is 32MB, 64MB, and 64MB, respectively. However, the actual data coverage observed at runtime depends on the sector utilization of the applications. As a result, the tag cache miss rate is a function of the number of tag cache entries, the sector size, and the sector utilization. The NRU

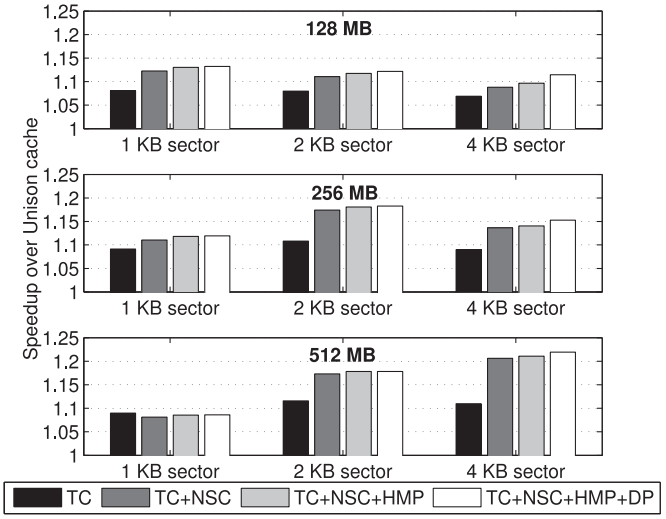

Fig. 6. Performance improvement over Unison cache due to introduction of tag cache (TC), NRU state cache (NSC), MAP-I hit/miss predictor (HMP), and demand prioritization (DP).

state cache miss rate increases with increasing L4 cache capacity and decreasing sector size. At 512MB with 1KB sectors, the NRU state cache miss ratio is 27%. A high NRU state cache miss rate leads to a lot of dirty NRU state replacements causing write-induced interference in the L4 cache. Larger sectors lead to more efficient designs of the tag cache and the NRU state cache, making the 4KB sector size an attractive design point. The hit/miss predictor offers about a percentage improvement in performance. Demand prioritization adds 1%–2% of performance improvement in the 4KB sectored configurations. At this sector size, the volume of prefetch requests injected in a burst increases significantly, making demand prioritization important.

Finally, we conduct experiments to validate that a four-way baseline outperforms a simpler direct-mapped baseline, and the NRU replacement policy outperforms a simpler random replacement policy, particularly at 2KB and 4KB sector sizes, where the cache management algorithms are very important. Henceforth, the four-way Unison cache with NRU replacement, tag cache, NRU state cache, MAP-I hit/miss predictor, and demand prioritization will be used as the baseline. Our proposal discussed in the next section is incorporated on top of this baseline.

4. IMPROVING SECTOR UTILIZATION

We discuss the details of our proposal in this section. We begin our exploration by understanding the DS cache architecture, one of the prominent efforts toward improving the space utilization of SRAM-based sectored caches.

4.1. Background: DS Cache

The DS cache improves sector utilization by multiplexing multiple sectors onto the same physical sector frame. The cache blocks at position k of all the multiplexed sectors (say, N in number) can compete for the cache block position k within a physical sector frame. Each cache block position of the physical sector frame needs to maintain $\log_2(N)$ bits, indicating which one of the multiplexed sectors it belongs to, where N is the degree of multiplexing. Additionally, each physical sector needs to maintain N sector tags and the associated REF and valid bits. Figure 7 shows a 512-byte physical sector with two sectors multiplexed onto it. Each bit position of the membership vector

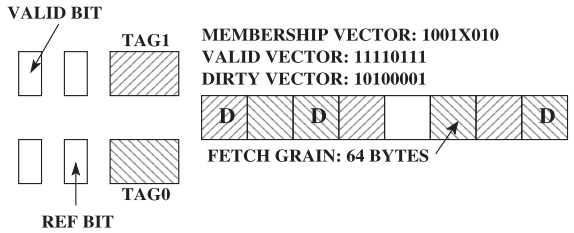

Fig. 7. A two-way multiplexed 512-byte sector along with tags, state vectors, REF bits, and valid bits.

indicates the sector the cache block in that position belongs to. The following analysis of DS DRAM cache assumes a data-associativity (not to be confused with set-associativity) of one to simplify the design (data-associativity was introduced in Section 1.1). In Section 4.5, we will explain data-associativity in more detail and consider designs with higher data-associativity.

4.2. A DS DRAM Cache

In this section, we design and evaluate a DRAM cache architecture that is derived from the idea of a DS cache. We also propose a hierarchical NRU replacement policy suitable for the DS DRAM cache.

4.2.1. Implications on Set Metadata. For a multiplexing degree of N, the critical metadata in a four-way set needs to maintain $4N$ tags (each of length 23 to 20 bits for DRAM cache capacities 128MB to 1GB), $4N$ tag valid bits, $4S\lceil \log_2(N) \rceil$ membership vector bits (where S is the number of blocks in a sector), four S-bit valid vectors, and four S-bit dirty vectors. Since we want to restrict the critical metadata size per set to 64 bytes and the smallest sector size (960 bytes) has S equal to 15, degree of multiplexing bigger than three is not possible. A degree of multiplexing equal to three is possible only for the smallest sector size and cache capacities more than 256MB. However, we restrict our design to only a power-of-two degree of multiplexing to keep the implementation simple. This leaves us with the only choice of degree of multiplexing equal to two. For a 4,032-byte sector ($S = 63$), a degree of multiplexing of even two requires two CAS commands to fetch the critical metadata (which exceeds 64 bytes in size) in the case of a tag cache miss. In the rest of this article, we consider a degree of multiplexing two only. The non-critical part of the metadata needs to be extended by the REF bit, demand vector, program counter, offset value, and core id corresponding to the second tag in each way. The entire metadata of a way comfortably fits within 64 bytes, which we have borrowed from each sector.

4.2.2. Implications on SRAM Structures. Each tag cache entry needs to be widened to include the second tag (will be referred to as the partner tag) of a way and the membership vector. To understand why the second tag needs to be maintained in a tag cache entry, let us consider an L4 cache way with two tags A and B multiplexed on it. Let us suppose that an access to the sector corresponding to tag A replaces a block belonging to tag B. Since the membership vector, valid vector, and the dirty vector of the way can get modified due to this replacement, these entities in the tag cache entry for tag B must also be updated. This is achieved by maintaining tag B in the tag cache entry of tag A. The tag cache is looked up for tag A (this is the parent request) and the entire entry (including partner tag B) is read out. Now, a second lookup can be executed to the same tag cache set for tag B and on a hit, tag B's entry is updated. Each entry of the NRU state cache needs to be widened to include the REF bits of the partner tags (four additional bits in a four-way cache).

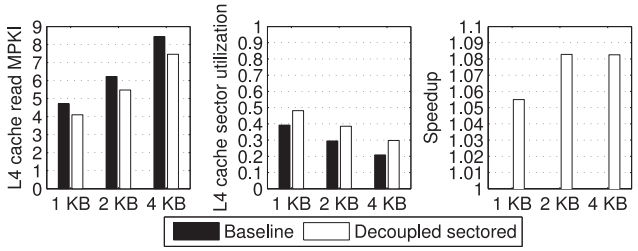

Fig. 8. Effectiveness of the DS DRAM cache at 128MB capacity.

4.2.3. Hierarchical NRU Replacement Policy. We devise a new sector replacement policy for the DS DRAM cache without requiring any additional replacement states. Let us refer to the two sectors multiplexed (or paired) onto a physical sector frame as partners of each other. The NRU sector in a set is a poor choice as a victim if its partner sector is currently actively accessed (e.g., the MRU sector). In such a situation, the newly filled sector and the actively accessed partner sector can thrash each other. Based on this observation, we propose the hierarchical NRU replacement policy, which first locates the NRU physical way and within the NRU physical way, it replaces the NRU tag. This policy guarantees that the partner sector of the newly filled sector has not been accessed in the recent past.

4.2.4. Performance Analysis of the DS DRAM Cache. Figure 8 summarizes the performance of the decoupled sectored DRAM cache at 128MB capacity. The results are averaged over 30 multi-programmed mixes. The sector sizes are noted on the horizontal axis. The left and the middle panels show that the decoupled sectored cache is able to significantly improve the L4 cache read MPKI (e.g., 8.4 to 7.5 for 4KB sectors) and the L4 cache sector utilization (e.g., 21% to 30% for 4KB sectors). These improvements lead to 6%, 8%, and 8% speedup for 1KB, 2KB, and 4KB sectors, respectively (right panel). The performance improvement flattens out at the 4KB sector size because the 4KB sectored design suffers from an average 7% increase in the L4 cache read hit latency. Two CAS commands are required to fetch the set metadata in the case of a tag cache miss at the 4KB sector size.

The two-way multiplexed DS cache suffers from two performance pathologies. First, the critical metadata overhead does not scale favorably and requires more than one CAS at the 4KB sector size. Second, each cache block position of a physical sector is contended by two cache blocks from the two multiplexed sectors. The likelihood of this contention is high if both of the multiplexed sectors have high sector utilization. For example, in Figure 7, if the sector corresponding to TAG0 fills its block at position zero, this will lead to a contention with the block at position zero of the other sector. To quantify this phenomenon, we maintain a thrash count, which is incremented whenever in a physical sector frame, a cache block belonging to sector i is evicted by a cache block from sector j, where $i \neq j$.

Figure 9 shows the thrash count and the number of L4 cache read hits, both normalized to the number of L4 cache read lookups for a 128MB DS cache with 1KB sectors. The 30 multi-programmed mixes are sorted by their L4 cache read hit rates. As expected, there is a visible inverse correlation between thrash ratio and read hit rate. The thrash ratio increases as the read hit rate falls, except at very high read hit rates.

The crux of our proposal on improving sector utilization revolves around two techniques that lower the thrash count. First, we design more flexible mechanisms for allocating the cache blocks of the multiplexed sectors (Section 4.3). Second, we design

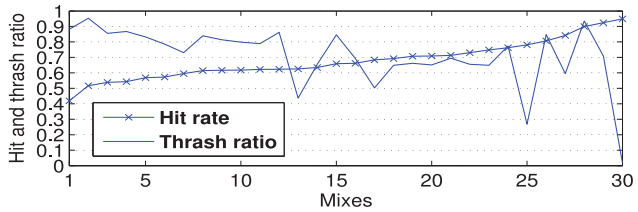

Fig. 9. Correlation between thrash ratio and hit rate at 128MB with a 1KB sector.

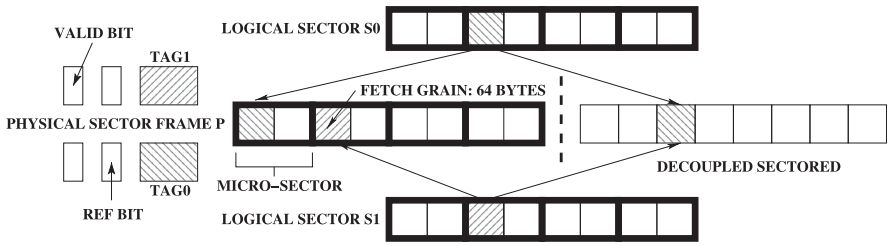

Fig. 10. An example showing the advantage of micro-sectors over the DS cache.

sector replacement algorithms that can control which pair of sectors is allowed to multiplex (Section 4.4).

4.3. Flexible Allocation with Micro-Sectors

Maximum flexibility can be incorporated in the DS cache's cache block placement algorithm by allowing a cache block of a multiplexed sector to fill in any cache block position of the host physical sector frame. However, such a design imposes too high an overhead on the metadata storage. In the following, we develop an implementable, yet reasonably flexible, cache block mapping scheme.

We introduce, within each sector, a coarse-grain allocation unit called micro-sector. A micro-sector is a contiguous region of a sector comprised of one or more consecutive cache blocks. For example, a 1KB sector has four 256-byte micro-sectors; the first four cache blocks form the first micro-sector, the next four cache blocks form the second micro-sector, and so on. We propose that when a cache block belonging to one of the multiplexed sectors is filled into the host physical sector frame, a full micro-sector be reserved for that sector. However, this micro-sector can be allocated in any of the micro-sectors of the physical sector frame, thereby offering significant flexibility. The micro-sector can be seen as the allocation block extending Goodman's nomenclature of transfer and address blocks [Goodman 1983].

As an example, consider a DS cache with 512-byte sectors and 128-byte micro-sectors. Each sector has eight cache blocks numbered 0 to 7 and four micro-sectors. Figure 10 shows two logical sectors S0 and S1 and a physical sector frame P. The micro-sectors are shown using bold lines. The proposed solution boils down to mapping the logical micro-sectors to the micro-sectors of the physical sector frame. Suppose that cache block number 2 of sector S0 is requested. This block belongs to the second micro-sector of S0. At this point, TAG0 is allocated for S0 and the first micro-sector of P is allocated to the second micro-sector of S0. Later, cache block number 2 of sector S1 is requested. At this point, TAG1 is allocated for S1 and the second micro-sector of P is allocated to the second micro-sector of S1. The right side of Figure 10 shows that these two accesses would have conflicted in the DS cache without micro-sectors.

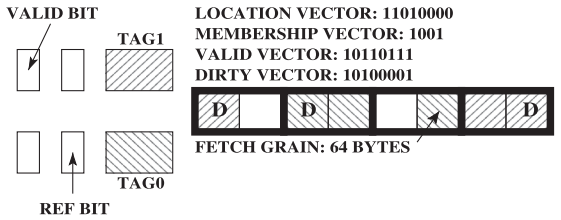

Fig. 11. A two-way multiplexed 512-byte sector implementing 128-byte micro-sectors.

A filling cache block invokes a micro-sector replacement policy when a new micro-sector needs to be allocated and there is no unoccupied micro-sector in the physical sector frame. The NRU micro-sector replacement policy is a poor choice because evicting a micro-sector (even if it is the NRU micro-sector) that belongs to the sector which is currently filling the cache block is found to degrade hit rate. The micro-sectors belonging to the filling sector have a high likelihood of being accessed soon. Having decided that the victim micro-sector must be chosen from the other sector (will be referred to as the partner sector), we explore two policies, namely, a random micro-sector replacement policy and a policy that replaces the micro-sector that has the lowest population of valid blocks.

The micro-sector-based design needs a membership bit per micro-sector of the physical sector frame. This bit indicates to which of the two sectors a particular physical micro-sector belongs. Additionally, each physical micro-sector needs to maintain $\lceil \log_2(\mu) \rceil$ location bits indicating its actual position within its parent sector, where μ is the number of micro-sectors in a sector. Assuming S cache blocks per sector, if μ is chosen such that $\mu + \mu\lceil \log_2(\mu) \rceil < S$ with $2 \leq \mu \leq S$, the overall metadata storage can be less than the DS cache's membership vector length. Additionally, while choosing μ, it is important to keep in mind that smaller micro-sectors (i.e., larger values of μ) offer higher flexibility in allocation and lower likelihood of sector under-utilization. Therefore, μ should be maximized while constraining the critical metadata fetch to a single CAS. Figure 11 shows a 512-byte sector implementing 128-byte micro-sectors. The four micro-sectors are shown with bold lines. Let TAG0 and TAG1 correspond to sectors S0 and S1. Let the four micro-sectors in each sector be numbered zero to three. The membership vector indicates that the first and the last micro-sectors belong to S1, while the middle two micro-sectors belong to S0. The first two bits of the location vector indicate the position of the first physical micro-sector within its parent sector, the next two bits indicate the position of the second physical micro-sector within its parent sector, and so on.

4.3.1. Implications on Set Metadata. The baseline has sector sizes 960, 1,984, 4,032 bytes corresponding to 15, 31, and 63 cache blocks per sector. Except 31, the other two can be factored to define the possible micro-sector sizes. To be able to define a micro-sector size, we use a sector size of 1,920 bytes in the place of 1,984 bytes. For 960-byte sectors, we use five micro-sectors, each of size 192 bytes. In this case, the micro-sector membership and location vectors (5 + 15 bits) can be comfortably accommodated in the 128-bit critical metadata per way. For 1,920-byte sectors, we use six micro-sectors, each of size 320 bytes. In this case, for DRAM cache capacities more than 256MB, the critical metadata storage per way can be accommodated within 128 bits (two tag valid bits, two tags each of size at most 21 bits, 6 bits of membership vector, 18 bits of location vector, 30 bits each for valid and dirty vectors). For 128MB and 256MB capacity, we store only the lower 20 bits of each of the two tags along with an OR of the left-out higher bits of each of the tags in the critical part of the metadata. Only if at least one

Table IV. SRAM Overhead of Micro-sector Cache

Structure	Number of entries, size, latency at 4GHz (for 22nm nodes)			On critical path?
	1KB sector	2KB sector	4KB sector	
Footprint table	16K, 110KB, 2 cycles	16K, 140KB, 2 cycles	16K, 206KB, 3 cycles	No
Singleton table	512, 3KB, 1 cycle	512, 3KB, 1 cycle	512, 3KB, 1 cycle	No
Way predictor	32K, 92KB, 2 cycles	32K, 104KB, 2 cycles	32K, 120KB, 2 cycles	Yes
Tag cache	32K, 448KB, 6 cycles	32K, 576KB, 6 cycles	16K, 416KB, 6 cycles	Yes
NRU state cache	32K, 56KB, 1 cycle	32K, 52KB, 1 cycle	32K, 48KB, 1 cycle	No
Hit/Miss predictor	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	Yes
Total size	710.5KB	876.5KB	794.5KB	

of the higher bits of the accessed tag is one, more than one CAS command is needed to determine hit/miss in the case of a tag cache miss.[7] This happens only if an accessed address exceeds 64TB or 32TB when the DRAM cache capacity is 256MB or 128MB, respectively. For 4,032-byte sectors, we use seven micro-sectors each of size 576 bytes. In this case, within the critical metadata per way, we can store 16 lower bits of each of the two tags along with an OR of the higher bits of each of the two tags, two tag valid bits, 7 bits of membership vector, 21 bits of location vector, and 63 bits of valid vector. Only if a physical address lies beyond 2TB, 4TB, 8TB, or 16TB, respectively, for DRAM cache capacities 128MB, 256MB, 512MB, or 1GB, more than one CAS command is needed to decide hit/miss. In summary, the Micro-sector cache rarely requires more than one CAS to fetch the set metadata on a tag cache miss.

4.3.2. Implications on SRAM Structures. Each tag cache entry needs to accommodate the location and the membership vectors. Each way predictor entry also needs to be augmented to include a membership bit indicating which of the two tags in the predicted way this entry corresponds to. Each way predictor entry also stores the membership and location vectors to compute the physical offset of the requested block.

Table IV summarizes the SRAM overhead of the L4 Micro-sector cache. Overall, our proposal's total SRAM overhead varies from 710.5KB to 876.5KB, which is less than one L3 cache way. Referring back to Table III, we see that our proposal requires at most 300KB of additional SRAM storage compared to the baseline, and most of this additional SRAM storage is devoted to the tag cache. As shown in Table IV, for the Micro-sector cache, we increase the tag cache latency by two additional cycles, compared to the baseline, for two reasons. First, one extra cycle accounts for the larger size of the tag cache. Second, another extra cycle accounts for the time to decode the block offset within a sector from the valid vector, membership vector, and location vector. Further, to compensate for the additional SRAM storage of the Micro-sector cache, we will also show the results for a baseline that has double the number of tag cache entries and four-cycle lookup latency.

On a tag cache miss, as in the baseline, a metadata fetch is queued up in the DRAM cache followed by a speculative data fetch from the predicted way if the hit/miss predictor indicates a possible hit. There are at least BL/2 cycles (equivalent to five cycles at 4GHz in our configuration) between the completion of these two fetch operations.

[7]We assume that the two CAS commands are issued back-to-back. However, the second CAS command would be wasted if the metadata fetched by the first CAS command is enough to flag a DRAM cache miss.

This is enough time to carry out the tag comparison and decoding of the block offset within the requested sector so that the correctness of the predicted data fetch can be verified in time.

4.4. Locality-Aware Sector Tag Allocation

Our locality-aware sector tag allocation algorithm ensures that the partner sector of a victim sector and the newly allocated sector have complementary spatial locality so that they do not thrash each other. Since sector utilization can serve as a good indicator of spatial locality, our algorithm takes into account the sector utilization of the resident sectors in a set in addition to the REF bits for selecting the victim sector.

We discuss the replacement algorithm in the context of a four-way cache, but it can be extended to work with arbitrary associativity. The central idea of the algorithm is that if the new sector has a high predicted utilization (more than half), it is allowed to replace only from the NRU way; otherwise, it is allowed to probe one extra way deeper into the NRU stack, looking for more replacement options without introducing too much thrashing. The algorithm first orders the four physical sector ways in a set from NRU to non-NRU (the ways with REF bits reset first and then those with REF bits set). The algorithm uses the predicted footprint (offered by the footprint prediction table) of the new sector to estimate its utilization. If the predicted utilization is more than half, the algorithm falls back to the hierarchical NRU replacement policy; otherwise, it examines the current occupancy of the non-NRU tag in each of the first two physical ways in the NRU to non-NRU order. Let us suppose that these two physical ways are denoted by $W1$ and $W2$. The NRU and the non-NRU sectors in way $W1$ are denoted by S_{W1}^{NRU} and S_{W1}^{nNRU}, respectively. Similarly, we denote the sectors in way $W2$ as S_{W2}^{NRU} and S_{W2}^{nNRU}. The algorithm examines the current occupancy of S_{W1}^{nNRU} and S_{W2}^{nNRU}. Let us suppose that the one among these two sectors that has the minimum current occupancy be resident in way $W \in \{W1, W2\}$. The algorithm picks the NRU tag of way W for victimization. In other words, if $W = W1$, the victim is S_{W1}^{NRU}. Similarly, if $W = W2$, the victim is S_{W2}^{NRU}. The algorithm victimizes the NRU sector in a physical way such that the current occupancy of the victim's partner sector is minimized among the available options.

The sector replacement and the micro-sector replacement/allocation algorithms are orthogonal. On a sector replacement, all micro-sectors belonging to the replaced sector must be evicted. The newly allocated sector will allocate the micro-sector containing the demanded block as well as any other micro-sectors that are prefetched as part of the sector footprint prefetching mechanism. The allocation of the micro-sectors employs the already discussed micro-sector replacement and allocation algorithms.

4.5. Data-Associativity in Micro-Sector Mapping

The DS cache proposal introduced an orthogonal dimension called data-associativity for mapping cache blocks to physical sectors. If the data-associativity is A_d and the set-associativity is nA_d, the idea is to divide the physical ways into n "data-associative" groups, where each group contains a contiguous chunk of A_d physical ways. Suppose that a cache block B is at position k of its sector S and the tag of sector S is allocated in physical way w. The cache block B can compete for cache block position k in each of the physical ways within the data-associative group containing way w. Each block within a physical sector of the DS cache maintains additional $\log_2(A_d)$ bits to identify the block's tag way, which can now be different from the block's data way.

Figure 12 shows a four-way set with a data-associativity of two ($A_d = 2$). The two data-associative groups are shown as G0 and G1. Assuming a degree of multiplexing of two, each way has two sector tags multiplexed on it. However, the data blocks of a

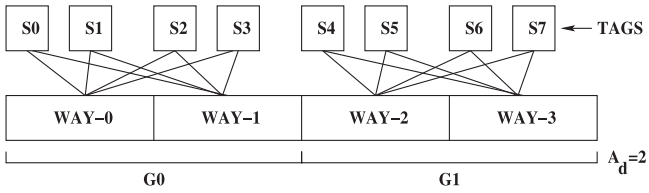

Fig. 12. A two-way multiplexed four-way set with data-associativity of two.

sector can be allocated to any of the ways belonging to its data-associative group. For example, sectors S0 and S1 have their tags mapped to way-0, but a block at position k of any of these sectors can use the position k of way-0 or way-1. Similarly, sectors S2 and S3 have their tags mapped to way-1, but a block at position k of any of these sectors can use the position k of way-0 or way-1. The lines connecting the tags with the ways show all possible ways that a data block belonging to a particular tag can map to. Clearly, such an arrangement can further improve the utilization of the cache ways. We consider the possibility of introducing a data-associativity of four in our Micro-sector cache design. This effectively extends the concept of data-associativity from blocks to micro-sectors. A design with $A_d = 4$ implies that in a four-way cache, a logical micro-sector can be mapped to any physical micro-sector position in the entire set. Each physical micro-sector maintains two bits to identify the micro-sector's tag way. These bits form a data-associativity vector for each physical sector way.

For 960-byte sector size, the ten-bit data-associativity vector can be comfortably accommodated within 128-bit critical metadata per way. For 1,920-byte sector size, we maintain a separate demand vector and disassociate the dirty vector from the critical metadata to accommodate the twelve-bit data-associativity vector within the critical metadata. For a 4,032-byte sectored design, there is no option but to use two CAS commands for fetching the set metadata on a tag cache miss. Since the micro-sectors of a logical sector can now be scattered over different ways of a set, each way predictor entry must offer micro-sector-specific (as opposed to sector-specific) predictions. Determining hit/miss requires access to the membership vectors, the location vectors, and the data-associativity vectors of all the four data-associative ways. To handle this, we replace the tag cache by a set-tag cache, where each entry holds the critical metadata of an entire set. Since each set-tag cache entry is roughly four times larger than a tag cache entry in a four-way set-associative L4 cache, for fairness of evaluation, we size the set-tag cache to have one-fourth of the baseline tag cache entries.

4.6. Performance of Micro-Sector Cache

We show the benefit of introducing micro-sectors in Section 4.6.1. Section 4.6.2 evaluates the locality-aware sector tag allocation algorithm. Section 4.6.3 explores the impact of data-associativity. The evaluations in Sections 4.6.1, 4.6.2, and 4.6.3 are done for 128MB capacity. Section 4.6.4 evaluates our proposal for larger capacity points and larger core counts.

4.6.1. Effectiveness of Micro-Sectors. Figure 13 quantifies the benefits of micro-sector allocation. The sector sizes are noted on the horizontal axis. Each sector size shows four bars representing the baseline (the optimized Unison cache), DS cache, baseline with micro-sectors exercising random micro-sector replacement policy (Micro-sectorRR), and baseline with micro-sectors exercising the valid block population-aware micro-sector replacement (Micro-sectorPopR). Results are averaged over 30 mixes (Table II).

The left and middle panels, respectively, show that the L4 cache read MPKI and sector utilization improve due to the flexibility offered by micro-sector placement. The

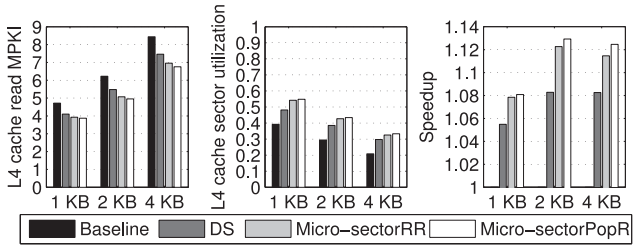

Fig. 13. Effectiveness of micro-sectors at 128MB capacity.

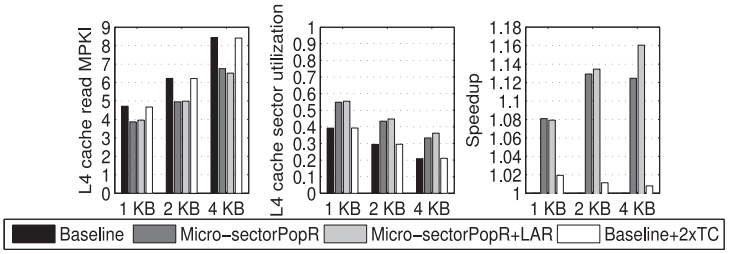

Fig. 14. Effectiveness of locality-aware replacement at 128MB capacity.

rightmost panel shows that micro-sector placement is able to offer reasonable improvement in performance (baseline is at 1.0). Since the valid block population-aware micro-sector replacement policy is more effective than the random replacement policy, we will use the former in the rest of the study. Overall, introduction of micro-sectors improves the baseline performance by 8%, 13%, and 13% for 1KB, 2KB, and 4KB sector sizes, respectively. The corresponding speedup figures for the DS cache are 6%, 8%, and 8%, respectively.

4.6.2. Effectiveness of Locality-Aware Sector Allocation. Figure 14 quantifies the benefits of the locality-aware sector tag allocation algorithm. We show results for the baseline, the baseline with micro-sectors exercising the valid block population-aware micro-sector replacement (Micro-sectorPopR), Micro-sectorPopR with locality-aware sector tag replacement (Micro-sectorPopR+LAR), and the baseline with a double-sized tag cache (Baseline+2xTC). The Baseline+2xTC configuration compensates for the additional SRAM needed by the micro-sectored designs. All results are averaged over 30 mixes.

The rightmost panel shows that the locality-aware sector tag allocation algorithm is very effective at the 4KB sector size (MPKI decreases by 4% and sector utilization improves by 9% compared to Micro-sectorPopR). It brings small benefits at the 2KB sector size also. Overall, the Micro-sectorPopR+LAR configuration improves performance by 8%, 14%, and 16% over the baseline for 1KB, 2KB, and 4KB sector sizes, respectively. Henceforth, the Micro-sectorPopR+LAR configuration will be referred to as the Micro-sector cache. Provisioning the baseline with a double-sized tag cache can only improve the average hit latency, while leaving the L4 cache read MPKI (leftmost panel) and sector utilization (middle panel) unaffected. The rightmost panel shows that the Baseline+2xTC configuration offers 1% to 2% speedup.

The performance improvement enjoyed by the Micro-sector cache arises from better sector utilization leading to improved read MPKI. The misses in a sectored cache can be classified into sector misses (the sector containing the requested block is not found in the cache) and block misses (the sector containing the requested block is present in the cache, but the block is not present). The Micro-sector cache reduces the

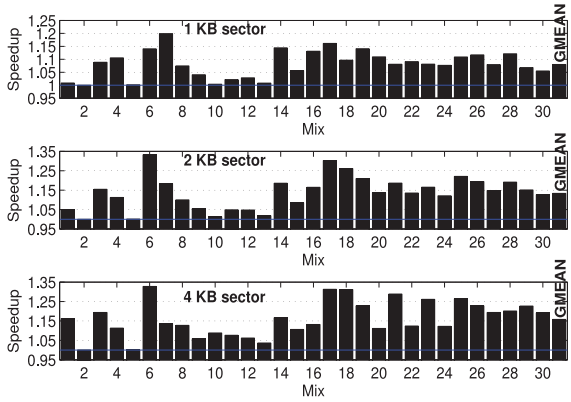

Fig. 15. Speedup at 128MB for 1KB (top panel), 2KB (mid panel), and 4KB (bottom panel) sector sizes.

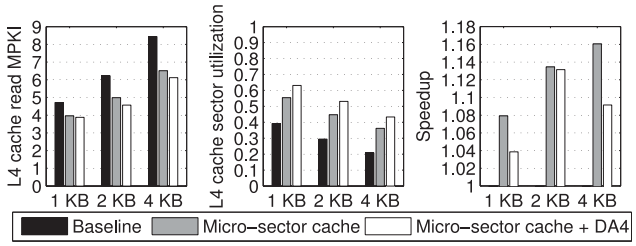

Fig. 16. Effectiveness of data-associativity at 128MB.

volume of both types of misses through better sector allocation and better micro-sector mapping, leading to lowered thrashing. The sector read MPKI for the baseline 128MB DRAM cache is 2.1, 1.9, and 1.8 for 1KB, 2KB, and 4KB sector sizes, respectively. For the Micro-sector cache, the corresponding MPKI values are 1.4, 1.3, and 1.2. With increasing sector size, the number of sectors decreases, leading to a gradual drop in sector read MPKI. The block read MPKI for the baseline 128MB DRAM cache is 2.6, 4.3, and 6.6 for 1KB, 2KB, and 4KB sector sizes, respectively. For the Micro-sector cache, the corresponding MPKI values are 2.5, 3.7, and 5.3.

Figure 15 details the speedup achieved by the Micro-sector cache for each mix listed in Table II at 128MB capacity. For the 1KB sector size (top panel), the maximum gain achieved by a mix is 20% and no mix loses in performance. For 2KB and 4KB sector sizes, the maximum gain is 33%. For all three sector sizes, several mixes gain more than 10%.

4.6.3. Impact of Data-Associativity. Figure 16 summarizes the effectiveness of the Micro-sector cache with a data-associativity of four (Micro-sector cache + DA4). The results are averaged over 30 mixes. The left panel shows that DA4 can further reduce the L4 cache read MPKI. The middle panel shows that DA4 can significantly improve sector utilization. The rightmost panel, however, shows that DA4 fails to add any performance benefit. There are two reasons for this. First, the set-tag cache has much lower effective coverage than the tag cache because all the four ways in an L4 cache set may not get used during the set's residency in the set-tag cache. This leads to under-utilization of the space devoted to a set-tag cache entry, which is made large enough to hold the critical metadata of a complete set of the L4 cache. As a result, the set-tag cache miss ratio (40%, 32%, and 31% at 1KB, 2KB, and 4KB sector sizes) is much higher than the

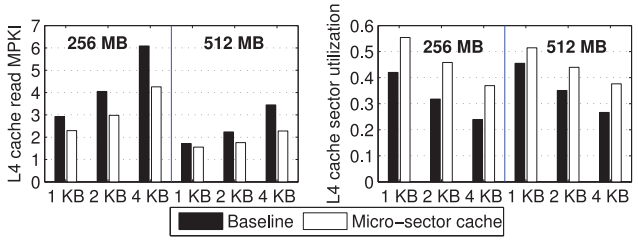

Fig. 17. Effectiveness of the Micro-sector cache at 256MB and 512MB.

Table V. Speedup Achieved by the Micro-Sector Cache

Size	1KB sector		2KB sector		4KB sector	
	Avg.	Max.	Avg.	Max.	Avg.	Max.
256MB	1.03	1.13	1.09	1.29	1.16	1.34
512MB	1.01	1.02	1.02	1.12	1.08	1.30

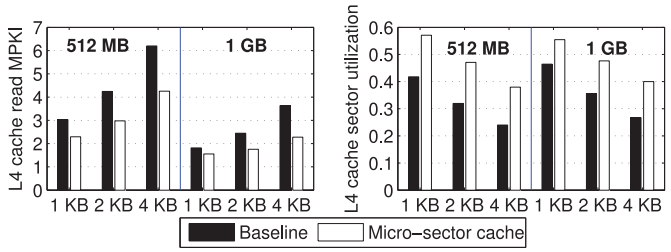

Fig. 18. Effectiveness of Micro-sector cache at 512MB and 1GB with 32 cores.

tag cache miss ratio (31%, 23%, and 21% at 1KB, 2KB, and 4KB sector sizes). Second, at the 4KB sector size, two CAS commands are needed to fetch the metadata on a set-tag cache miss. At the 2KB sector size, the improvement in the L4 cache read MPKI nearly compensates the loss due to the higher set-tag cache miss rate. Henceforth, we will consider only the default data-associativity of one.

4.6.4. Larger L4 Caches and Larger Core-Count. Figure 17 summarizes the average (over 30 mixes) L4 cache read MPKI (left panel) and sector utilization (right panel) at 256MB and 512MB capacity. With 4KB sectors, our proposal has L4 cache read MPKI of 4.3 and 2.3, respectively, at the 256MB and 512MB capacity, while the baseline has read MPKI of 6.1 and 3.5. Table V lists the average (geometric mean) and maximum speedup achieved by the Micro-sector cache for the 30 mixes. The speedup is particularly impressive for the 4KB sector size.

To evaluate our proposal on a 32-core system, we scale up the shared L3 cache to 32MB, leaving the associativity unchanged. The number of copies of each application in each workload mix listed in Table II is doubled to build the set of 30 32-way multi-programmed mixes. Figure 18 summarizes the average (over 30 mixes) L4 cache read MPKI (left panel) and sector utilization (right panel) at the 512MB and 1GB capacity. The Micro-sector cache improves sector utilization across the board and enjoys significant savings in the volume of the L4 cache read misses. At 512MB and 1GB with 4KB sectors, our proposal has L4 cache read MPKI of 4.4 and 2.3, respectively, while the baseline has read MPKI of 6.2 and 3.6. The read MPKI improvements achieved by the Micro-sector cache at 1KB and 2KB sector sizes are also reasonable. Table VI lists the average (geometric mean) and maximum speedup achieved by the Micro-sector cache for the 30 mixes at 512MB and 1GB. For a 4KB sector size, the Micro-sector

Table VI. Speedup Achieved by the Micro-Sector Cache in 32-Core Systems

Size	1KB sector		2KB sector		4KB sector	
	Avg.	Max.	Avg.	Max.	Avg.	Max.
512 MB	1.02	1.04	1.10	1.28	1.26	1.52
1 GB	1.00	1.01	1.02	1.04	1.07	1.18

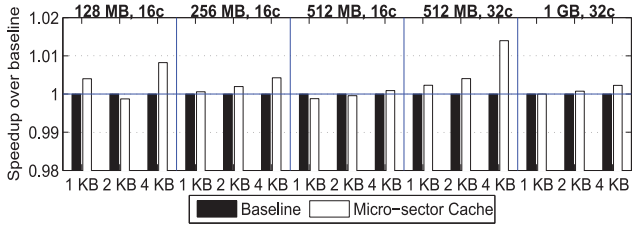

Fig. 19. Effectiveness of the Micro-sector cache for the workloads with high sector utilization.

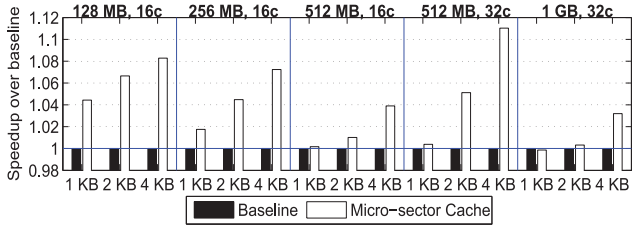

Fig. 20. Effectiveness of the Micro-sector cache for the workloads with low sector utilization.

cache offers excellent speedup at both capacity points (26% and 7% on average). At 512MB capacity, even a 2KB sectored design enjoys 10% speedup.

5. EVALUATION ON LARGER SET OF WORKLOADS

In this section, we evaluate the Micro-sector cache in the context of a larger set of workloads. We also compare the Micro-sector cache against the state-of-the-art designs that exercise fine-grain allocation units (Section 5.1) and the Bimodal cache (Section 5.2).

For the studies discussed in this section, we select 28 application-input combinations spanning 24 different SPEC CPU 2006 applications. All evaluations are carried out by executing each of these 28 workloads in rate mode. We partition the 28 workloads into two groups based on the sector utilization observed in the 128MB baseline. The high utilization group has 13 workloads: bwaves, bzip2.combined, cactusADM, dealII, gobmk.score2, gromacs, hmmer.nph3, lbm, leslie3d, libquantum, soplex.ref, wrf, and zeusmp. This group exhibits an average sector utilization of 92%, 89%, and 80%, respectively, for 1KB, 2KB, and 4KB sector sizes at 128MB capacity. The low utilization group has 15 workloads: astar.rivers, gamess.triazolium, gcc.cpdecl, gcc.s04, gobmk.nngs, gobmk.trevorc, h264ref.foreman_main, mcf, milc, omnetpp, perlbench.checkspam, sjeng, soplex.pds-50, sphinx3, and xalancbmk. This group exhibits an average sector utilization of 41%, 32%, and 24%, respectively, for 1KB, 2KB, and 4KB sector sizes at 128MB.

Figures 19 and 20 quantify the performance speedup achieved by the Micro-sector cache over the baseline for the workload groups with high and low sector utilization, respectively. We show the results for five different design points, where each design point is represented by the <capacity, core count> tuple (16c and 32c refer to 16-core and 32-core systems, respectively). The sector sizes are noted on the horizontal axis.

For the workloads with high sector utilization, the Micro-sector cache and the baseline deliver similar levels of performance, as expected (Figure 19). For the workloads with low sector utilization, the Micro-sector cache offers reasonable performance benefits (Figure 20). The maximum average gain enjoyed by the Micro-sector cache is 11%, observed in a 32-core system with a 512MB L4 cache.

5.1. Comparison with Fine-Grain Allocation

The state-of-the-art designs with 64-byte allocation units considered in this section are based on the direct-mapped Alloy cache [Qureshi and Loh 2012], since this design has been shown to outperform the other existing fine-grain designs. The Alloy cache and the BEAR optimizations [Chou et al. 2015] assume a non-standard burst length of five. The JEDEC HBM standard supports burst lengths of only two and four [JEDEC 2015]. We consider a new fine-grain design that adopts BEAR to a burst length of four. With a burst length of four, the data and metadata fetch would require separate CAS commands. As a result, it is important to incorporate efficient tag caching in this design to avoid a large increase in the average hit latency. We design a tag cache that is similar in spirit to TIMBER [Meza et al. 2012]. We re-organize the Alloy cache row so that the metadata of all the sets mapped to a row are allocated together at the end of a row after the data blocks. A metadata fetch brings 64 bytes worth of metadata from a sequential stretch of 16 sets (one metadata is assumed to occupy 4 bytes). We widen each neighboring tag cache (NTC) entry of BEAR from one metadata to 16 metadata so that the fetched group of metadata can be cached.[8] Each L4 cache bank is equipped with one fully-associative NTC having 16 and 32 such wide entries for 16- and 32-core systems, respectively. On an NTC miss, two CAS operations (for metadata and data) are issued back-to-back if the MAP-I predictor speculates an L4 cache hit; if the predictor speculates a miss, only the metadata request is queued. Later, if this speculation turns out to be wrong, a data CAS request is queued. On an NTC hit, only one CAS operation is required for data fetch. We will refer to this new fine-grain design as BEAR+BL4.

In Alloy cache, BEAR, and BEAR+BL4, we incorporate a spatial memory streaming (SMS) prefetcher [Somogyi et al. 2006], which has similar hardware requirements as the footprint prefetcher of the Micro-sector cache. In addition to an equally-sized footprint table (16K entries), the fine-grain designs use a 16-entry accumulation table and a 16-entry filter table per core for learning the footprint vectors of the spatial regions [Somogyi et al. 2006]. The footprint of a spatial region is allocated in the footprint table when either the corresponding entry is evicted from the accumulation table or a block belonging to the corresponding region is evicted from the DRAM cache. Before injecting every prefetch request, the fine-grain designs require a bandwidth-consuming L4 cache lookup to find out if the block to be prefetched is already resident in the cache (in a sectored cache, these lookups are unnecessary because a sector miss guarantees the absence of all blocks to be prefetched in that sector). So, the optimal spatial region size must strike a balance between L4 cache bandwidth and prefetch coverage. To determine the best spatial region size, we evaluate the performance of the fine-grain designs with region sizes ranging from two blocks to 64 blocks (region sizes from 128 bytes to 4KB). For Alloy cache and BEAR, 16-block regions (1KB size; will be referred to as SMS16) yield the best performance. For BEAR+BL4, the best region size is 512 bytes (eight 64-byte blocks; will be referred to as SMS8). The best region size for the BEAR+BL4 design is expected to be smaller than the other two designs because the BEAR+BL4 design injects more CAS operations into the L4 cache, leading to more queuing. As a result, it is necessary to keep the bandwidth demand of the prefetch

[8]In the absence of a divider, the tag length may have to be bigger when the number of sets is not a power of two, leading to a lowered metadata coverage per metadata fetch.

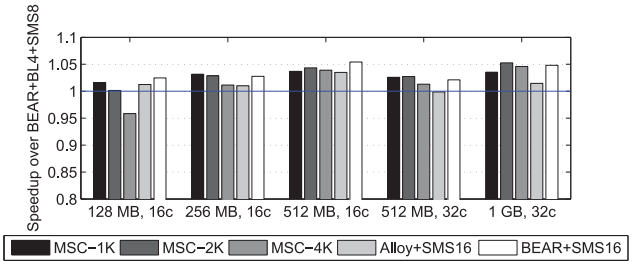

Fig. 21. Speedup relative to BEAR+BL4+SMS8 for the workloads with high sector utilization.

lookup requests low for best performance in BEAR+BL4. Finally, to compensate for the 700–900KB additional SRAM storage of the Micro-sector cache, we evaluate all fine-grain L4 cache designs with an L3 cache provisioned with one additional way, but we do not charge any additional latency for L3 cache access.

5.1.1. Performance Analysis. Figure 21 presents the performance of the Micro-sector cache (MSC-1K, MSC-2K, and MSC-4K corresponding to three different sector sizes), the Alloy cache with the SMS16 prefetcher (Alloy+SMS16), and BEAR with the SMS16 prefetcher (BEAR+SMS16) for the workload group with high sector utilization. The results are normalized with respect to the performance of BEAR+BL4 equipped with the SMS8 prefetcher (BEAR+BL4+SMS8). The Micro-sector cache either matches the performance of or outperforms the fine-grain designs. This is expected because the Micro-sector cache enjoys excellent sector utilization in these workloads. At 1GB capacity, the best Micro-sector cache design (2KB sector size) outperforms the Alloy cache by 4% and BEAR+BL4 by 5% on average. We found that for the highly bandwidth-sensitive applications, the best Micro-sector cache design outperforms the Alloy cache by at least 10% at 1GB. These include cactusADM (10% better), gobmk.score2 (10% better), lbm (20% better), and libquantum (16% better). Recall that the Alloy cache uses a burst length of five spread over three channel cycles (corresponding to tCCD=3) to transfer a TAD and two of these cycles transfer data. As a result, one-third of the bandwidth is spent in non-data transfers, part of which is metadata. As the L4 cache capacity is scaled up, more L4 cache bandwidth is demanded for serving the larger volume of hits. The metadata traffic starts affecting the hit latency of the Alloy cache as the cache grows in size. BEAR is able to address some of the bandwidth issues of the Alloy cache and delivers performance similar to the best Micro-sector cache at 1GB. However, we found that the Micro-sector cache continues to perform better for several workloads, with lbm showing the largest performance gap of 16% at 4KB sector size.

The BEAR+BL4 design uses the JEDEC-compliant standard burst length of four. It incorporates a tag cache to eliminate a large fraction of the metadata traffic. However, compared to BEAR, this design has a larger average L4 cache hit latency because a tag cache miss requires two CAS commands (metadata and data) to satisfy an L4 cache hit. Compared to the Micro-sector cache designs with 2KB and 4KB sectors, the BEAR+BL4 design suffers from more number of tag cache misses per metadata CAS. The BEAR+BL4 design fetches 16 spatially contiguous tags through one metadata CAS. As a result, while streaming through a 1,920-byte or 4,032-byte region, the BEAR+BL4 design needs two or four metadata CAS commands. On the other hand, the Micro-sector cache with 1,920-byte or 4,032-byte sectors would require one metadata CAS command in these cases. These additional metadata CAS commands in the BEAR+BL4 design lead to higher queuing delays resulting in a higher average L4 cache hit latency. We note that metadata prefetching can improve the tag cache miss latency in BEAR+BL4, but will not improve the metadata bandwidth bloat.

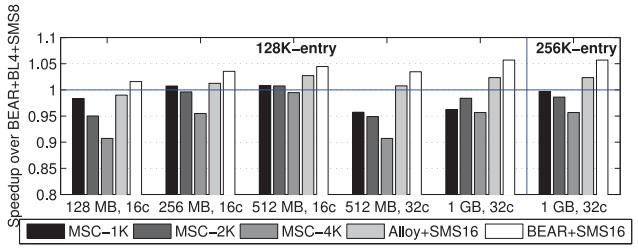

Fig. 22. Speedup relative to BEAR+BL4+SMS8 for the workloads with low sector utilization.

Preliminary evaluation on the applications with low sector utilization showed that the Micro-sector cache delivers performance close to the fine-grain designs at 128MB and 256MB. However, it lags behind by more than 5% at 512MB and 1GB capacities. One reason for this performance gap is the high NRU state cache miss rate in the Micro-sector cache at large capacity points. Fortunately, a 128K-entry direct-mapped NRU state cache can be comfortably accommodated within a total SRAM budget of 1MB (see Table IV). Even a 256K-entry direct-mapped NRU state cache can fit within a total SRAM budget of 1MB when operating with a 1GB Micro-sector cache having a sector size of 1KB.

Figure 22 quantifies the performance of the Micro-sector cache and the fine-grain designs relative to BEAR+BL4+SMS8 for the workload group with low sector utilization. The Micro-sector cache is provisioned with a 128K-entry direct-mapped NRU state cache. We also show the performance of a 1GB Micro-sector cache provisioned with a 256K-entry direct-mapped NRU state cache (the rightmost group of bars). With a 128K-entry NRU state cache, the best Micro-sector cache design performs within 4% and 2% of BEAR+BL4 at 512MB and 1GB capacities in a 32-core system. With a 256K-entry NRU state cache, the best 1GB Micro-sector cache (1KB sector size) delivers the same level of performance as BEAR+BL4. For the remaining configurations, the best Micro-sector cache design either outperforms or performs within a percentage of BEAR+BL4. The residual performance gap between the Micro-sector cache and the BEAR+BL4 design arises primarily due to the higher L4 cache read miss rates in the former. The Alloy cache and BEAR enjoy additional benefits in the average L4 cache read hit latency due to the use of custom burst length. The metadata bandwidth bloat in these designs is not a problem for the workloads with low sector utilization, since these workloads do not demand much bandwidth from the memory system.

In summary, when averaged over the entire set of 28 workloads, we observe that the 1GB Micro-sector cache using 2KB sectors and a 256K-entry direct-mapped NRU state cache outperforms the 1GB Alloy cache. For 256MB and 512MB capacity points with 16 cores, the Micro-sector cache having 2KB sector size delivers similar performance as the Alloy cache. In the following, we explore the energy-efficiency of these three capacity points.

5.1.2. Energy Analysis. We compare the designs in terms of energy expended in the L3 cache, the L4 cache SRAM, the L4 cache DRAM, and the main memory DRAM. We use CACTI [HP Labs 2009a] (distributed with McPAT [HP Labs 2009b]) to evaluate the dynamic and leakage energy in the SRAM parts for 22nm technology. We appropriately upgrade the DRAM power model integrated with DRAMSim2 to evaluate the energy expended in the DRAM parts. The DRAM power model has been validated against the Micron power calculator [Micro Technology Inc. 2007] to the

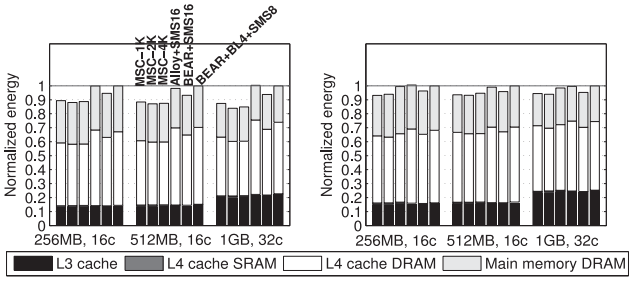

Fig. 23. Energy normalized to BEAR+BL4+SMS8 for the workloads with high (left panel) and low (right panel) sector utilization. The 1GB configuration has 32MB L3 cache, while others have 16MB L3 cache.

extent possible[9] and includes background, activation/precharge, CAS, and refresh power.

Figure 23 evaluates the energy expended by the Micro-sector cache and the fine-grain designs normalized to the energy expended by BEAR+BL4+SMS8. As expected, across the board, the major portion of the energy is expended in the L4 cache DRAM. The L4 cache SRAM has negligible energy consumption. For the workloads with high sector utilization (left panel), the Micro-sector cache consistently expends less energy compared to all the fine-grain designs. The Alloy cache and the BEAR designs consume more energy per CAS due to a higher burst length (spread over three channel cycles) compared to the Micro-sector cache design (two channel cycles). For the workloads with low sector utilization (right panel), the Micro-sector cache with 1KB and 2KB sector sizes is more energy-efficient than the fine-grain designs. However, the difference in energy consumption between the Micro-sector cache and the fine-grain designs is less pronounced in these workloads than in those with high sector utilization. This is primarily because the Alloy cache and the BEAR designs are able to mostly compensate the energy loss in longer burst length with a much smaller number of CAS operations compared to the Micro-sector cache, which suffers from a reasonably high tag cache miss ratio for the workloads with low sector utilization, requiring a larger volume of metadata CAS operations. Overall, when averaged over the entire set of 28 workloads, we observe that the Micro-sector cache is consistently more energy-efficient than the fine-grain designs. In particular, the Micro-sector cache with 2KB sectors expends 5% to 7% less energy than BEAR+SMS16. At 1GB capacity, the Micro-sector cache with 2KB sectors is 6% better in terms of energy expense and 3% better in terms of energy-delay product compared to BEAR+SMS16.

5.2. Comparison with Bimodal Cache

The Bimodal cache adapts between big and small blocks depending on the utilization of the big blocks [Gulur et al. 2014]. It varies the associativity of a fixed-sized set to achieve this. We evaluate a Bimodal cache with a set size of 4KB. Each set can be independently and dynamically configured to have either 4 or 19 ways. In the first configuration, each of the four ways has a big 1KB block, which is entirely fetched on a miss. In the second configuration, each set has three big ways, each of size 1KB, and 16 small ways, each of size 64 bytes. To be able to use both the configurations, the set metadata must be sized to accommodate 19 ways. This requires two CAS operations to access the set metadata. Allowing configurations to have a larger number of small ways

[9]The HBM power model is approximate and is mostly based on DDR3 power model with some parameters appropriately scaled.

results in a prohibitively large overhead of set metadata access. The Bimodal cache exercises a block size predictor with 256K entries and a way locator (similar to a tag cache) with 64K entries, respectively. A way locator miss requires two CAS operations to fetch the entire set metadata.

The Micro-sector cache outperforms the Bimodal cache at all configuration points for both high-utilization and low-utilization workloads. For the high-utilization workloads, the performance gains range from 2% to 11%. For the low-utilization workloads, the performance gains range from 8% to 14%. Although the Bimodal cache is able to enjoy lower L4 cache read miss rates compared to the Micro-sector cache, it suffers from two serious performance pathologies. First, it experiences higher average L4 cache miss latency due to the wasted main memory bandwidth and queuing resulting from the big block fetches (even under low utilization, three ways in a set must be big). Second, it suffers from higher average L4 cache hit latency due to the necessity of two CAS commands for fetching the metadata on a way locator miss. The second problem tends to be the dominating reason for the performance loss, as the L4 cache capacity is scaled up, leading to an increased volume of L4 cache hits.

6. SUMMARY

We have presented the Micro-sector cache, an efficient design for improving the space utilization of sectored DRAM caches. Our proposal first blends the ideas from the decoupled sectored cache design into the state-of-the-art DRAM cache designs, exercising large allocation units. Next, we improve this design with two optimizations. First, we introduce micro-sectors enabling flexible allocation of cache blocks in a sector. Second, we propose hierarchical NRU and locality-aware algorithms for improving the quality of sector replacement. Our proposed design, when incorporated in an optimized Unison cache baseline, improves the baseline performance by significant margins, particularly for large sector sizes as the DRAM cache capacity is varied between 128MB and 1GB.

REFERENCES

R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen. 2011. IBM POWER7 systems. *IBM Journal of Research and Development* 55, 3, 2:1–2:13.

C.-C. Chou, A. Jaleel, M. K. Qureshi. 2015. BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 198–210.

M. El-Nacouzi, I. Atta, M. Papadopoulou, J. Zebchuk, N. Enright-Jerger, and A. Moshovos. 2013. A Dual Grain Hit-miss Detector for Large Die-stacked DRAM Caches. In *Proceedings of the Conference on Design, Automation and Test in Europe*. 89–92.

S. Franey and M. Lipasti. 2015. Tag Tables. In *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture*. 514–525.

J. R. Goodman. 1983. Using Cache Memory to Reduce Processor-Memory Traffic. In *Proceedings of the 10th Annual International Symposium on Computer Architecture*. 124–131.

N. D. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan. 2014. Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth. In *Proceedings of the 47th Annual International Symposium on Microarchitecture*. 38–50.

F. Hameed, L. Bauer, and J. Henkel. 2013. Simultaneously Optimizing DRAM Cache Hit Latency and Miss Rate via Novel Set Mapping Policies. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. 1–10.

M. D. Hill and A. J. Smith. 1984. Experimental Evaluation of On-chip Microprocessor Cache Memories. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*. 158–166.

HP Labs. 2009a. CACTI: An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model. Available at http://www.hpl.hp.com/research/cacti/.

HP Labs. 2009b. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. Available at http://www.hpl.hp.com/research/mcpat/.

C.-C. Huang and V. Nagarajan. 2014. ATCache: Reducing DRAM Cache Latency via a Small SRAM Tag Cache. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 51–60.

IBM Corporation. 2012. IBM POWER Systems. Available at http://www-05.ibm.com/cz/events/febannouncement2012/pdf/power_architecture.pdf.

Intel Corporation. 2013. Crystalwell products. Available at http://ark.intel.com/products/codename/51802/Crystal-Well.

H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee. 2016. Efficient Footprint Caching for Tagless DRAM Caches. In *Proceedings of the 22nd International Conference on High-Performance Computer Architecture*. 237–248.

JEDEC. 2015. High Bandwidth Memory (HBM) DRAM. *Standard Documents JESD235A*, November 2015. Available at https://www.jedec.org/standards-documents/docs/jesd235.

D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *Proceedings of the 47th Annual International Symposium on Microarchitecture*. 25–37.

D. Jevdjic, S. Volos, and B. Falsafi. 2013. Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache. In *Proceedings of the 40th Annual International Symposium on Computer Architecture.* 404–415.

X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian. 2010. CHOP: Adaptive Filter-based DRAM Caching for CMP Server Platforms. In *Proceedings of the 16th International Conference on High-Performance Computer Architecture*.

N. Kurd, M. Chowdhury, E. Burton, T. P. Thomas, C. Mozak, B. Boswell, M. Lal, A. Deval, J. Douglas, M. Elassal, A. Nalamalpu, T. M. Wilson, M. Merten, S. Chennupaty, W. Gomes, and R. Kumar. 2014. Haswell: A Family of IA 22 nm Processors. In *International Solid-State Circuits Conference.* 112–113.

Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, J. W. Lee. 2015. A Fully Associative, Tagless DRAM Cache. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 211–222.

J. S. Liptay. 1968. Structural Aspects of the System/360 Model 85, Part II: The Cache. *IBM Systems Journal* 7, 1, 15–21.

G. H. Loh and M. D. Hill. 2011. Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches. In *Proceedings of the 44th Annual International Symposium on Microarchitecture.* 454–464.

N. Madan, L. Zhao, N. Muralimanohar, A. N. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell. 2009. Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy. In *Proceedings of the 15th International Conference on High-Performance Computer Architecture*. 262–274.

J. Meza, J. Chang, H-B. Yoon, O. Mutlu, and P. Ranganathan. 2012. Enabling Efficient and Scalable Hybrid Memories using Fine-Granularity DRAM Cache Management. *IEEE Computer Architecture Letters* 11, 2, 61–64.

Micron Technology Inc. 2007. DDR3 SDRAM System-Power Calculator. Available at https://www.micron.com/~media/documents/products/power-calculator/ddr3_power_calc.xlsm?la=en.

C. R. Moore. 1993. The PowerPC 601 Microprocessor. In *Proceedings of the IEEE COMPCON*. 109–116.

S. A. Przybylski. 1990. The Performance Impact of Block Sizes and Fetch Strategies. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*. 160–169.

M. K. Qureshi and G. H. Loh. 2012. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *Proceedings of the 45th Annual International Symposium on Microarchitecture*. 235–246.

P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* 10, 1, 16–19.

J. B. Rothman and A. J. Smith. 1999. The Pool of Subsectors Cache Design. In *Proceedings of the International Conference on Supercomputing.* 31–42.

J. B. Rothman and A. J. Smith. 2000. Sector Cache Design and Performance. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 124–133.

A. Seznec. 1994. Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost and Low Miss Ratio. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*. 384–393,

T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*. 45–57.

J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi. 2012. A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch. In *Proceedings of the 45th Annual International Symposium on Microarchitecture*. 247–257.

J. Sim, G. H. Loh, V. Sridharan, and M. O'Connor. 2013. Resilient Die-stacked DRAM Caches. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 416–427.

S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. 2006. Spatial Memory Streaming. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture.* 252–263.

J. Stuecheli. 2013. Next Generation POWER Microprocessor. In *Hot Chips*.

K. Tran and J. Ahn. 2014. HBM: Memory Solution for High Performance Processors. In *MemCon.*

M. Tremblay and J. M. O'Connor. 1996. UltraSparc I: A Four-issue Processor Supporting Multimedia. *IEEE Micro* 16, 2, 42–50, April 1996.

R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. 2012. Multi2Sim: A Simulation Framework for CPU-GPU Computing. In *Proceedings of the 21st International Conference on Parallel Architecture and Compilation Techniques*. 335–344.

H. Wang, T. Sun, and Q. Yang. 1995. CAT - Caching Address Tags: A Technique for Reducing Area Cost of On-Chip Caches. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*. 381–390.

D. Windheiser, E. L. Boyd, E. Hao, S. G. Abraham, and E. S. Davidson. 1993. KSR1 Multiprocessor: Analysis of Latency Hiding Techniques in a Sparse Solver. In *Proceedings of the 7th International Parallel Processing Symposium*. 454–461.

Z. Zhang, Z. Zhu, and X. Zhang. 2004. Design and Optimization of Large Size and Low Overhead Off-Chip Caches. *IEEE Transactions on Computers* 53, 7, 843–855.

L. Zhao, R. Iyer, R. Illikkal, and D. Newell. 2007. Exploring DRAM Cache Architectures for CMP Server Platforms. In *Proceedings of the 25th International Conference on Computer Design.* 55–62.

Received June 2016; revised December 2016; accepted January 2017