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

Mainak Chaudhuri[†] Mukesh Agrawal[‡] Jayesh Gaur[♯] Sreenivas Subramoney[♯]

† Indian Institute of Technology, Kanpur 208016, INDIA
‡ Intel Architecture Group, Hilsboro, OR 97124, USA
♯ Intel Microarchitecture Research Lab, Bangalore 560103, INDIA

Abstract

DRAM caches are increasingly becoming the preferred choice for architecting high-capacity last-level caches in high-end computing systems. Recent research proposals on DRAM cache architectures with conventional allocation units (64 or 128 bytes) as well as large allocation units (512 bytes to 4 KB) have explored ways to minimize the space/latency impact of the tag store and maximize the effective utilization of the main memory bandwidth. In this paper, 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 requested portions of an allocated sector. The recently proposed Unison cache design can be efficiently used in a sectored cache to improve the hit rate and the overall performance by prefetching the necessary portions of an allocated sector with the help of a sector footprint predictor. However, the sectored cache designs suffer from poor utilization of the cache space, since a large sector is always allocated even if the sector utilization is low. The Unison cache design addresses only a special case of this problem by not allocating the sectors that have only one actively used block.

In this paper, 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 thirty sixteen-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 1 KB, 2 KB, and 4 KB sectors at 128 MB 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 1 KB, 2 KB, and 4 KB sectors at 128 MB capacity. We explore the design space of our proposal in terms of DRAM cache capacity ranging from 128 MB to 1 GB and sector size ranging from 1 KB to 4 KB.*

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 [1, 12, 13, 19, 38]. 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) [3, 7, 23, 25, 29, 35], coarse-grain (e.g., 512 bytes to 4 KB) [14, 16, 17, 18, 21], or mixed-grain [6] allocation units (referred to as the DRAM cache block size). There are other studies that explore a range of allocation units [45] and configurable block sizes ranging from 64 bytes to 512 bytes [24].

The studies assuming fine-grain or conventional allocation units focus their efforts on managing the large tag store and minimizing the impact of the serialization latency introduced by tag lookup. The state-of-the-art design in this direction proposes to organize the DRAM cache as a direct-mapped cache with tag and data (TAD) of a cache block fused and maintained together in the DRAM array [29]. This design is referred to as the Alloy cache. On a lookup, one TAD unit is read out of the DRAM array and the hit/miss detection is done by examining the tag. The proposal further incorporates simple hit/miss predictors to initiate the main memory lookup early in the case of a predicted DRAM cache miss. This design enjoys fast hits due to the absence of tag serialization latency and can initiate miss handling early with good miss prediction accuracy.

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 demanded portions of the large block on a cache miss. The proposals in the first category can waste a significant amount of main memory bandwidth if a sizable portion of the large block brought from main memory does not get used before getting evicted due to lack of spatial locality. As a result, these proposals focus their efforts to minimize this wastage [6, 18]. The designs in the second category are usually referred to as sectored or sub-blocked caches. The design of the sectored DRAM caches is the central topic of this study. The allocation unit in these designs is referred to as a sector ranging in size from 512 bytes to 4 KB. 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 64 bytes in this study. Therefore, the sectored cache proposals can simultaneously optimize the main memory bandwidth requirement and the tag store size [5, 8, 14, 16, 17, 21, 22, 27, 28, 32, 40, 43, 45].

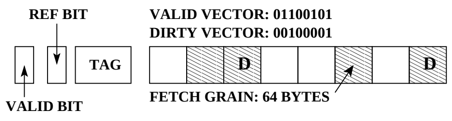

Figure 1. A 512-byte sector along with tag, state vectors, REF bit, and valid bit.

In a sectored cache, only one tag needs to be maintained per sector along with a state vector to record the states of the constituent cache blocks (e.g., valid/occupied and dirty). Figure 1 shows a 512-byte sector composed of eight 64-byte cache blocks, four of which are 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 (required for sector replacement), the sector-wide valid bit, and the valid and dirty vectors are also shown. An n-way set-associative 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 DRAM cache design that exercises large allocation units [16]. As far as the cache and metadata organization is concerned, the Unison cache is effectively a sectored cache. In this study, we will refer to its allocation unit as a sector.[1] It is a set-associative cache with a small number of ways. The tag/state and other metadata of a set are co-located with the sectors that constitute the set. A DRAM row is designed to accommodate a small number of consecutive sets. On a lookup, the metadata necessary for deciding hit/miss in the target set are read out. To avoid serialization due to tag access, a way predictor is looked up and the requested data block is read out from the sector in the predicted way. If the prediction turns out to be correct, the fetched block is returned to the inner levels of the cache hierarchy. 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 allocates a full sector and consults a sector footprint predictor, which returns a bitvector indicating the blocks within the sector that are likely to be accessed during the sector's residency in the cache. These blocks are also fetched along with the demanded block. This prefetching mechanism employs a spatial memory streaming prefetcher [37] and significantly improves the DRAM cache hit rate while controlling the main memory bandwidth wastage.

[1] The original proposal refers to the allocation unit of the Unison cache as a page. Since the DRAM page size can be different from the allocation unit of the Unison cache, to avoid any confusion we refer to the allocation unit as a sector.

The early proposals of the sectored cache were motivated by the reduction in the SRAM storage required for tag/state. 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 for metadata caching. Such an SRAM structure will be referred to as a tag cache. While designing an efficient sectored DRAM cache baseline for this study in Section 3 we quantitatively show the importance of a tag cache in saving metadata bandwidth and improving the performance of the DRAM cache. We will also revisit this aspect in Section 5 while comparing our proposal with the DRAM caches exercising small allocation units.

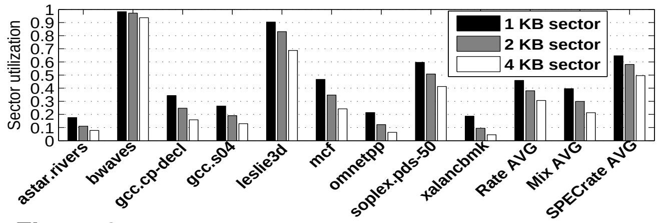

Figure 2. Sector utilization in a 128 MB 4-way Unison cache.

While sectored caches offer an attractive design choice for architecting DRAM caches, 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. Low sector utilization reduces the effective cache capacity. 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). To understand the severity of the problem of sector under-utilization, we selected 28 application-input combinations from the SPEC CPU2006 suite spanning 24 different applications.[2] We sample nine of these 28 application-input combinations and use them for most of this study. We show detailed results on the entire set of 28 application-input combinations toward the end of this article. Figure 2 shows the sector utilization of these nine SPEC CPU2006 workloads in a 128 MB 4-way Unison cache when the workloads are run in sixteen-way rate mode. Sixteen copies of each workload are executed on a simulated sixteen-core system with a four-level cache hierarchy, the last two levels being shared across all the cores. The DRAM cache is the last-level (L4) cache.[3] 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

[2] Our simulation environment cannot run gemsFDTD and tonto correctly. Additionally, we exclude calculix, namd, and povray due to their performance insensitivity toward large DRAM caches.

[3] Simulation methodology is discussed in Section 2.

the utilization values over all the snapshots. The rate average group (Rate AVG) of bars shows that, on average, a 1 KB sector is utilized 46% and a 4 KB sector is utilized only 31% for these nine homogeneous sixteen-way multi-programmed workloads.[4] The average sector utilization for a bigger set of thirty sixteen-way multi-programmed workloads prepared by mixing the nine SPEC CPU2006 workloads is shown in the mix average group (Mix AVG) of bars. This average shows that a 1 KB sector is 40% occupied and a 4 KB sector is only 21% utilized, on average. The rightmost group (SPECrate AVG) of bars shows the average sector utilization for all the 28 application-input combinations when each of these is executed in sixteen-way rate mode. This average shows that a 1 KB sector is 65% utilized and a 4 KB sector is 50% utilized. Overall, we conclude from the data shown in Figure 2 that significant portions of a sector are not used and this space inefficiency increases quickly with growing sector sizes.

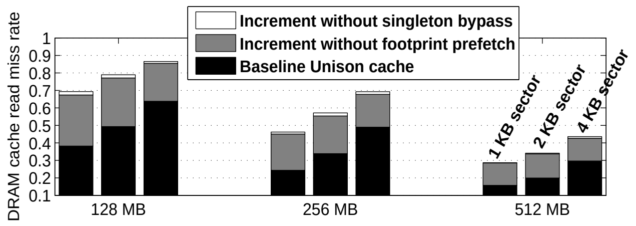

Figure 3. DRAM cache read miss rate with varying capacity and sector size. Each group of bars corresponds to one DRAM cache capacity point and the bars in a group correspond to 1 KB, 2 KB, and 4 KB sector sizes.

Figure 3 further compares the L4 cache read miss rate as a function of the L4 cache capacity and the sector size.[5] The results are averaged over thirty sixteen-way multi-programmed workloads prepared by mixing the nine SPEC CPU2006 workloads. Each group of bars corresponds to one capacity point and the bars in a group correspond to 1 KB, 2 KB, and 4 KB sector sizes respectively from left to right. Each bar has three sections. The lower section corresponds to the average read miss rate of the Unison cache. The middle portion corresponds to the increment in average read miss rate when sector footprint prefetching is disabled. The upper portion corresponds to the additional increment in average read miss rate when singleton sector bypass is disabled. While footprint prefetching is highly effective in bringing down the read miss rate of the sectored cache, singleton sector bypass offers small benefits. Overall, even though Unison cache's footprint prefetching and singleton bypass optimizations are able to save a significant volume of L4 cache misses, across the board, with or without these optimizations, the L4 cache read miss rate steadily increases with increasing sector size. For example, as the sector size increases from 1 KB to 4 KB in a 128 MB Unison cache, the read miss rate increases from 38% to 64%.

[4] The actual sector sizes used are 960, 1920, and 4032 bytes. This is further clarified in Sections 3 and 4.

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

In this study, we focus our effort on improving the sector utilization and read miss rate of the sectored DRAM caches, particularly for the designs exercising large sectors. 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 by understanding the shortcomings of the decoupled sectored cache (Sections 4.1 and 4.2), which is one of the few notable efforts to improve the performance of sectored caches [33]. We propose a novel sectored cache organization that systematically incorporates flexibility and locality information in the sector allocation and sector replacement algorithms leading to the design of a locality-aware sectored cache (Sections 4.3, 4.4, and 4.5). The simulation results show that our DRAM cache proposal significantly improves the sector utilization and outperforms the baseline Unison cache and other DRAM cache designs for different sector sizes in the capacity range 128 MB to 1 GB (Sections 4.6, 5.1, 5.2, 5.3, and 5.4).[6] A summary of our contributions is presented in the following.

- We begin our study by designing a DRAM cache employing the decoupled sectored organization, which was originally proposed for improving the space utilization in sectored SRAM caches. This design brings out two major shortcomings of the decoupled sectored organization: a) high metadata overhead and b) thrashing within a physical sector frame.

- We simultaneously address both the shortcomings of the decoupled sectored 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 128 MB to 1 GB.

1.1. Related Work

DRAM cache proposals have used small as well as large allocation units. 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

[6] Most of the analyses in this article are based on the results for 128 MB to 512 MB DRAM caches. We present the results for a 1 GB cache in Sections 5.2, 5.3, and 5.4.

associative set in a single DRAM page [23]. A DRAM cache lookup reads out the entire page into the row buffer and examines the tags while reserving the row buffer contents for a possible subsequent read for obtaining the data in the case of a hit. To detect DRAM cache misses quickly without looking up the DRAM array, the proposal maintains an SRAM array of bitvectors (called MissMap), where each vector tracks the presence of the DRAM cache blocks in a coarse-grain contiguous region. This proposal suffers from relatively large DRAM cache hit latencies primarily due to the compound access to get the tag and the data. Also, accesses to consecutive sets (seen in applications with good spatial locality) require opening a new DRAM page for each such access to a new set leading to poor row buffer hit rate even for streaming access patterns.

A subsequent proposal replaces the multi-megabyte MissMap structure with a DRAM cache hit/miss predictor and dynamically manages the congestion at the DRAM cache by dispatching some of the requests to the main memory even if these could hit in the DRAM cache [35]. Further studies have explored different designs of the DRAM cache hit/miss predictors by combining page-grain and block-grain residency trackers for the DRAM cache contents [3].

The direct-mapped Alloy cache significantly shortens the critical path of a hit and improves row buffer locality by contiguously allocating the TADs from consecutive sets [29]. However, the Alloy cache either requires custom burst length for transferring a TAD or wastes bandwidth by using the next higher burst length made available by the interface. The recently proposed bandwidth-efficient architecture (BEAR) for DRAM caches explores a few optimizations to improve the bandwidth utilization of the DRAM caches exercising 64-byte blocks [2]. Studies exploring DRAM caches with conventional small block sizes have also proposed different functions for mapping the cache sets into the DRAM pages with the goal of optimizing hit latency as well as hit rate [7]. Small on-die SRAM storage for a subset of the most recently used tags has also been proposed [25]. A DRAM cache architecture exercising 64-byte blocks and an SRAM tag cache working in conjunction with a tag prefetcher has been explored [9]. A recent proposal explores space-efficient ways of storing tags of DRAM caches exercising 64-byte blocks in the on-die last-level SRAM cache [4]. This proposal exploits spatial locality in access stream and metadata compression to arrive at an efficient metadata store called Tag Tables for DRAM caches exercising 64-byte blocks. 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, but require mechanisms to avoid wasting the main memory bandwidth. Caching only hot DRAM pages with the help of a small filter cache that identifies the most frequently accessed pages has been explored [18]. The more recent Bimodal cache proposal dynamically selects between large and small blocks based on the utilization of the large blocks [6]. This adaptation is enabled by dynamically varying the associativity of a set so that at any point in time a number of big and small ways constitute a set while keeping the set size constant. The recently proposed tagless fully-associative DRAM cache design stores the location of a 4 KB (or the OS page size) DRAM cache block in the page table entry of the corresponding OS page and caches this information in an on-die specially designed hardware-managed extension of the TLB [21, 14]. While this design eliminates the need for maintaining DRAM cache tags, it requires OS modifications for appropriately extending the page table entries. Also, the block size (or the allocation unit) of the DRAM cache gets tied to the OS page size (typically at least 4 KB) leading to significant space wastage as already indicated in Figure 2. Our proposal is orthogonal to this design and can significantly improve the performance of the designs with such large DRAM cache block sizes.

The DRAM cache architecture that fetches only the necessary/demanded portions of a large allocation unit is usually referred to as a sectored cache. The 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 [17]. However, since this design maintains the cache metadata in SRAM, it does not scale to large capacities. The more recent Unison cache proposal employs the Footprint cache's predictor while co-locating the metadata of a set with the sectors of the set in a DRAM row [16]. The associativity of the Unison cache is kept small so that the metadata needed to decide hit/miss in a set can be read out with a single CAS command (at most 64 bytes with a burst length of four and 128-bit channels). This decision also allows one DRAM row to hold a few consecutive sets, thereby improving the row buffer locality. To avoid the serialization due to metadata lookup, the Unison cache employs a way predictor to start accessing the data from the predicted way while the metadata is fetched.

A sectored DRAM cache with an off-die tag store and an on-die tag cache has been explored in [44]. A design space exploration of DRAM caches with a range of block sizes as well as sector sizes has been presented in [45].

The decoupled sectored cache aims at improving the sector utilization by allowing multiple sectors to share a group of data-associative physical sector frames [33]. More specifically, 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 improving the overall utilization of the cache space. We further explore decoupled sectored caches in the context of DRAM caches in Sections 4.1 and 4.2.

Another proposal for improving sector utilization 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 [31]. Each cache block can be dynamically assigned to any sector within a set based on the need. The cache block pool size of each set is under-provisioned to avoid space wastage.

Our proposal first appropriately moulds the decoupled sectored cache design into an efficient DRAM cache design. Our exploration begins with a thorough understanding of the performance and drawbacks of this decoupled sectored DRAM cache. Based on this understanding, we propose the Micro-sector cache, which systematically incorporates locality-aware policies in sector allocation and sector replacement algorithms.

2. Simulation Framework

We use the Multi2Sim simulator [41] to model sixteen dynamically scheduled out-of-order issue x86 cores clocked at 4 GHz. Each core has 32 KB 8-way instruction and data L1 caches and a 256 KB 8-way unified L2 cache. The L1 and L2 cache access latencies are two and five cycles, respectively. We model an inclusive 16 MB 16-way shared L3 cache. The L3 cache is 16-way banked with seven-cycle bank lookup latency. The L1, L2, and L3 caches have 64-byte blocks. The cores and the L3 cache banks are arranged over a 4×4 mesh interconnect with single-cycle hop time. Each interconnect hop has a core, its L1 and L2 caches, and one L3 cache bank. We place two single-channel DDR3-1600 memory controllers at a pair of opposite corners of the mesh. The main memory DRAM array is modeled using DRAMSim2 [30]. The DRAM timing parameters correspond to 11-11-11-28. We model a channel width of 64 bits, two ranks per channel (Micron TwinDie configuration), eight banks per rank, 1 KB row buffer per bank per device with an open page policy, x8 devices, and a burst length of eight. Each bank within each rank is equipped with a read and a write command queue, each of depth sixteen. Each DRAM bank schedules the commands following the FR-FCFS policy with reads served before writes within an open row. Read-activates are given priority over write-activates when deciding which row to open in a closed bank. The aggregate main memory DRAM bandwidth is 25.6 GB/s.

The DRAM cache is the last level of our simulated four-level cache hierarchy. It is non-inclusive with respect to the L3 cache (an L4 cache eviction does not send invalidation to the L3 cache) and shared by all cores. The L3 cache dirty evictions that miss in the L4 cache are allocated in the L4 cache. All sectored L4 cache configurations are four-way set-associative. The DRAMSim2 module used to model the L4 DRAM cache is configured to have a frequency of 1.6 GHz (effective DDR transfer rate of 3.2 GT/s) and 11-11-11-28 timing parameters. We model four 128-bit channels, one rank per channel, sixteen banks per rank, and a burst length of four. The aggregate bandwidth offered by the DRAM cache is 204.8 GB/s. Compared to the main memory DRAM, the DRAM cache has eight times the bandwidth and half the latency. Each bank within each rank is equipped with a read and a write command queue, each of depth 32. Each DRAM cache bank schedules the commands following the FR-FCFS policy with reads served before writes within an open row. Read-activates are given priority over write-activates when deciding which row to open in a closed bank.

Table 1. Workload mixes

MIX1–MIX9: one app. × sixteen 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
(astar.rivers, gcc.cp-decl, gcc.s04, leslie3d, mcf, omnetpp, soplex.pds-50, xalancbmk) (astar.rivers, bwaves, gcc.cp-decl, leslie3d, mcf, omnetpp, soplex.pds-50, xalancbmk)

We select 28 application-input combinations from the SPEC CPU2006 suite spanning 24 different applications for this study.[7] From these 28 combinations, we sample nine combinations, which we showed in Figure 2. The workload set used for most of the execution-driven simulation studies in this article consists of thirty sixteen-way multi-programmed mixes prepared from these nine SPEC CPU2006 application-input combinations.[8] The mixes are prepared by drawing one, two, four, or eight different application-input combinations from the set of nine and replicating sixteen, eight, four, or two copies of each of the drawn applications to fill up the sixteen slots. Table 1 details the thirty multi-programmed mixes used in this study. 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 (read misses per kilo instructions) of these mixes varies from 1.7 to 34.9 with an average of 13.3.

[7] The selected 28 application-input combinations (if an application has multiple inputs, the chosen combination is shown as application-name.input-name): astar.rivers, bwaves, bzip2.combined, cactusADM, dealII, gamess.triazolium, gcc.cp-decl, gcc.s04, gobmk.nngs, gobmk.score2, gobmk.trevorc, gromacs, h264ref.foreman_main, hmmer.nph3, lbm, leslie3d, libquantum, mcf, milc, omnetpp, perlbench.checkspam, sjeng, soplex.pds-50, soplex.ref, sphinx3, wrf, xalancbmk, zeusmp.

[8] We evaluate our proposal on the full set of 28 application-input combinations in Section 5.4.

The workload mixes are run on the simulated sixteen-core system with each workload thread pinned to one core. Each workload mix commits at least eight billion dynamic instructions (a representative segment of 500 million dynamic instructions for each thread [34]). Threads completing early continue to run past their representative segments. The performance results are reported based on the statistics collected during the first 500 million retired instructions of each thread. We report speedup as the weighted speedup for a certain L4 cache optimization over the weighted speedup of the baseline. The weighted speedup of a workload mix for a configuration is defined as $\sum_i (IPC_i/SIPC_i)$ where IPC_i is the average instructions retired per cycle (IPC) for thread i with $0 \leq i \leq 15$ when the mix is run. $SIPC_i$ is the average IPC for thread i when this thread is allowed to run standalone.

3. Baseline Tag/Data Organization

The baseline Unison cache co-locates the metadata of a set with the sectors that constitute the set. We would like to organize the baseline DRAM cache so that when a core is sequentially streaming through some data, we enjoy row hits in all accesses after the row is opened. The size of the stream is limited by the OS page size, which is 4 KB in our model. Therefore, we would like to place 4 KB of sequential data in one row of the DRAM cache. If the DRAM cache sector size is 1 KB, we must have four consecutive sets allocated in the same DRAM cache row. Similarly, for 2 KB sectors, we need to have two consecutive sets allocated in the same DRAM cache row. As in the Unison cache proposal, we organize the baseline DRAM cache to have an associativity of four. Therefore, to be able to accommodate four, two, or one set for 1 KB, 2 KB, and 4 KB sectors in a row, we must have a row size of 16 KB. As in the Unison cache proposal, we use 8 KB row buffers per bank per device with an open page policy. There are two possible ways of designing an aggregate row buffer size of 16 KB in a rank. First, we can have two devices per rank with 64-bit output per device. Although multiple devices per rank can lead to several design issues [36], we believe that it is possible to efficiently support a small number (such as two) of devices in a rank. Second, we can have one device per rank and support two 64-bit pseudo channels per channel as outlined in the second generation high bandwidth memory (HBM2) implementation from SK Hynix [39]. Each pseudo channel has an independent bank array and we model an 8 KB row buffer per bank.[9] As a result, when a row is activated in a particular bank in a channel, the two corresponding 8 KB rows are opened in the two pseudo channels. This is functionally equivalent to having two devices per rank. In summary, 16 KB worth of data is opened when a row is activated in a bank within a DRAM cache channel (we simulate single rank per channel).

[9] We note that the pseudo channel concept is introduced in the HBM2 implementation from SK Hynix to reduce the row activation power per bank per pseudo channel by halving the per-bank row size. However, in this study, we use the pseudo channel concept to double the effective row size.

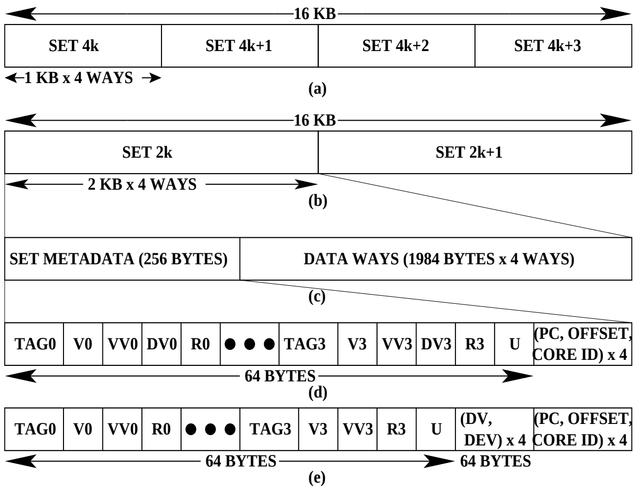
16 KB
SET 4k
SET 4k+1
SET 4k+2
SET 4k+3
1 KB x 4 WAYS
(a)
16 KB
SET 2k
SET 2k+1
2 KB x 4 WAYS
(b)
SET METADATA (256 BYTES)
DATA WAYS (1984 BYTES x 4 WAYS)
(c)
TAG0 V0 VV0 DV0 R0 ● ● ● TAG3 V3 VV3 DV3 R3 U (PC, OFFSET, CORE ID) x 4
64 BYTES
(d)
TAG0 V0 VV0 R0 ● ● ● TAG3 V3 VV3 R3 U (DV, DEV) x 4 (PC, OFFSET, CORE ID) x 4
64 BYTES
64 BYTES
(e)

Figure 4. (a) DRAM cache row layout when using 1 KB sectors. (b) DRAM cache row layout when using 2 KB sectors. (c) Set layout when using 2 KB sectors. (d) Set metadata layout when using 2 KB sectors. (e) Set metadata layout when using 4 KB 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.

Having decided the high-level organization of a DRAM cache row, we turn to understand how the metadata of a set is laid out. As in the Unison cache design, we borrow 64 bytes from each sector for storing the metadata. In other words, the 1 KB, 2 KB, and 4 KB sectors get reduced in size to 960, 1984, and 4032 bytes. The implication of these sector sizes on set indexing has been discussed in the Unison cache proposal. We will refine these sector sizes further in Section 4. However, we will continue to refer to them as 1 KB, 2 KB, and 4 KB 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 16 KB worth of data).[10] 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 128 MB to 1 GB capacity and 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 and the last three fields are needed for indexing into the

[10] While we borrow the 8 KB row buffer size and the basic row organization from the Unison cache proposal for the ease of comparison and extension, we would like to note that the JEDEC HBM standard does not support an 8 KB row buffer size [15]. To adopt a standard 2 KB or 4 KB row buffer size (as supported by the JEDEC HBM standard), it is necessary to allocate the metadata and the data of a set in different rows (possibly belonging to different banks so that the set metadata and the demanded data block can be fetched concurrently, if needed). Such a re-organization would not affect our proposal in any way, as our proposal's central contributions are orthogonal to this re-organization.

sector footprint prediction table at the time of updating the demanded footprint of a replaced sector. The Unison cache uses the valid and the dirty vectors to jointly encode four states of a 64-byte block within a sector: invalid, valid and not demanded, valid and demanded and not dirty, valid and demanded and dirty. This is the reason why it needs both valid and dirty vectors for deciding hit/miss. For sector sizes of 960 and 1984 bytes, the first four critical fields of a way can be comfortably stored within 128 bits. For 4032-byte sectors, each of the valid and dirty vectors is of length 63 bits. For this sector size, we do not use the dirty vector and the valid vector to jointly encode four states. Instead, we use the valid vector for recording only the valid blocks in a sector and the dirty vector for recording the dirty blocks in a sector. We maintain a separate demand vector to track the demanded blocks in the sector. As a result, the critical metadata now excludes the dirty vector and the remaining critical fields of a way can be accommodated within 128 bits. This layout is shown in Figure 4(e).

The address decoding for the DRAM cache is done as follows. Consecutive rows are interleaved across the DRAM cache channels. Since we have only one rank per channel, the rows mapped to a channel are interleaved across the banks of the rank on that channel. In other words, given a target <setid, wayid> tuple, it is converted to the cache address (setid$\times$associativity + wayid)$\times$waysize. The row number, bank number, channel number, and the offset within a row are extracted in that order from the most significant side to the least significant side of this cache address. The offset within a row corresponds to a 16 KB row size. This offset is further used to extract the column offsets for the CAS operations to fetch the set metadata and the required data block.

3.1. 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. To reduce the serialization due to metadata lookup, it employs a way predictor and initiates a data fetch from the predicted way while the critical metadata is fetched. However, since the metadata access and the speculative data access are sent to the same bank within the same rank, some amount of serialization cannot be avoided. This serialization delay can further increase under high load when the read command queue of the corresponding bank has only one slot available for enqueuing the metadata request and the speculative data access request can be enqueued at a later point in time when the read command queue of the corresponding bank has a free slot. Additional serialization delay can get introduced due to the read-modify-write operations to the metadata. Since every read lookup must update the set metadata to maintain the REF bits, valid vectors, and dirty vectors correctly, the metadata read of one request cannot be enqueued for scheduling until the metadata write command of the previous request to the same set has been enqueued. This ordering requirement arises from the fact that each read-modify-write to the metadata of a set must be atomic with respect to the other accesses to the same set metadata. However, it is enough to delay enqueuing a metadata read command to a set until the point when the latest metadata write command to the set has been enqueued in the write command queue. Subsequently, a write-to-read forwarding logic between the write and the read command queues of a bank preserves the correctness. This metadata write to read ordering requirement within a set can elongate the critical path of an access. In a sectored cache, this could be a common case because when a sector is streamed through, all the accesses go to the same L4 cache set.

To optimize the critical path, we incorporate a small SRAM cache of recently accessed tags, which we will refer to as a tag cache. Tag caches were first explored with the goal of reducing the on-die tag area [42]. The design of an on-die SRAM cache for holding the metadata of a stretch of consecutive sets of a direct-mapped DRAM cache exercising conventional small allocation units has been explored in a design called TIMBER [25]. Another proposal for including a tag cache to accelerate DRAM cache accesses has been explored in the context of a design called ATCache, which uses conventional small allocation units [9]. Due to the use of small allocation units (64 bytes), achieving good tag cache hit rate in the ATCache design requires prefetching into the tag cache. In our design, each entry of the tag cache holds the critical metadata of a way necessary for determining hit/miss (tag, tag valid bit, valid vector, and dirty vector). Since a tag covers one sector worth of data, we can achieve reasonable tag cache hit rate with a small tag cache and without tag cache prefetching.

An L3 cache miss looks up the tag cache and the way predictor in parallel. A tag cache hit provides us with the data way and obviates the need to wait for the metadata read to complete. Note that in the case of a read request, we still need to read out the first 64 bytes of the set metadata for deciding if the REF bits of all the ways require an update as a result of this access. At this time, if necessary, we also update the valid vector of the accessed way and the dirty vector of the accessed way if it is used to encode the demand vector (e.g., for designs with 1 KB and 2 KB sectors). However, the metadata read-modify-write is now completely off the critical path. We also keep the valid and dirty vectors of the tag cache entry always up-to-date because these are needed for determining hit/miss.

An L3 cache dirty eviction also looks up the tag cache and a tag cache hit provides us with the data way to write to. In this case, since the REF bits of the set are not updated, we avoid the metadata read-modify-write completely by updating the valid and dirty vectors in the tag cache entry.

If an L4 cache read misses in the tag cache, the way predictor's outcome is used for speculatively accessing the data way and the access path follows the original Unison cache proposal. If an L4 cache write misses in the tag cache, the write must be delayed until the critical metadata is read out.

When a tag is replaced from the L4 cache, its entry is looked up in the tag cache and the tag cache entry contents are taken into account to decide the set of dirty blocks in the replaced sector and the demand vector of the replaced sector needed for updating the footprint prediction table. At this time the tag cache entry is invalidated. On a tag cache replacement, if the dirty vector has been modified due to L4 cache write(s), the set metadata is read out and the dirty vector is updated. For architectures using 4 KB sectors, the demand vector is also updated at this time by decoding it from the valid and dirty vectors stored in the replaced tag cache entry. Allocating the dirty vector and the demand vector together in a 64-byte aligned portion of the metadata helps us update both these two vectors together through one CAS command (shown in Figure 4(e)). Finally, the replaced tag cache entry is used to update the way predictor so that a subsequent access to this tag can get a correct way prediction. It is important to note that while a tag is resident in the tag cache, the way predictor has no role to play. Therefore, to maximize the overall coverage of the tag cache and the way predictor taken together, the way predictor is updated with a new entry only when an entry is replaced from the tag cache. The way predictor continues to be a tagless direct-mapped structure employing address-based hashing for lookup.

We use a 32K-entry tag cache for the designs exercising 1 KB and 2 KB sectors, while for the designs exercising 4 KB sectors we use a 16K-entry tag cache. The tag cache is designed to have eight ways with a lookup latency of five cycles. It exercises NRU replacement. It is indexed using the lower bits of the L4 cache set index. The remaining upper bits of the physical address are used as the tag of the tag cache entry. Each tag cache entry holds a slightly extended L4 cache tag (extended by the upper bits of the L4 cache set index that are not used in the tag cache index), the dirty vector, the valid vector, and the much-needed DRAM cache way-id of the data corresponding to the cached tag entry. Additionally, a tag cache entry has its own valid bit, dirty bit, and REF bit needed by the NRU replacement policy.

The tag cache eliminates the REF bit update latency from the critical path. However, the REF bit updates continue to consume precious DRAM cache bandwidth. A significant portion of this bandwidth can be saved by incorporating a small SRAM cache for holding the REF bits of the recently accessed sets. In a sectored design, each entry of such a cache can exhibit significant utility particularly for streaming access patterns. We will refer to this structure as the NRU state cache. It is important to note that the NRU state cache access is not on the critical path of DRAM cache access. Its primary purpose is to absorb the REF bit update bandwidth and has no direct effect on the L4 cache hit/miss latency. In all our designs, 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 remaining upper bits of the L4 cache set index are used as the tag of the NRU state cache entry. In addition to these tag bits, each entry has a valid bit, a dirty bit, and an n-bit vector representing the n REF bits of an n-way DRAM cache set. In our baseline design, the value of n is four, since we explore four-way L4 caches.[11] The NRU state cache is accessed on every L4 cache read lookup and if the access changes the REF bits of the target L4 cache set, the corresponding NRU state cache entry is marked dirty. When a dirty entry is replaced from the NRU state cache, the corresponding L4 cache set metadata is read out and the REF state is updated. When a victim needs to be selected for an L4 cache set, the NRU state cache is looked up to read out the latest REF state of the target L4 cache set.

In addition to the tag cache and the NRU state cache, we incorporate the MAP-I hit/miss predictor to speed up L4 cache misses [29]. The tag cache, the way predictor, and the hit/miss predictor are looked up in parallel on an L3 cache read miss. A tag cache hit provides the target L4 cache way that needs to be accessed. 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. The hit/miss predictor uses a bank of 256 3-bit saturating counters per core [29]. For sixteen cores, the total storage is 1.5 KB for 4K counters.

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) so that a subsequent access to this tag can get a correct way prediction. Since the way predictor has no role to play when a tag is resident in the tag cache, the way predictor is updated with a new entry only when an entry is replaced from the tag cache. The tagless direct-mapped way predictor employs address-based hashing for lookup.

[11] 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 miss rate marginally.

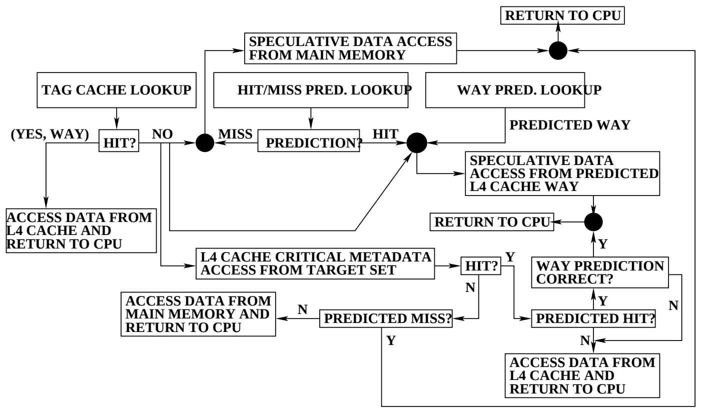
RETURN TO CPU
SPECULATIVE DATA ACCESS FROM MAIN MEMORY
TAG CACHE LOOKUP
HIT/MISS PRED. LOOKUP
WAY PRED. LOOKUP
PREDICTED WAY
(YES, WAY)
HIT?
NO
MISS
PREDICTION?
HIT
SPECULATIVE DATA ACCESS FROM PREDICTED L4 CACHE WAY
ACCESS DATA FROM L4 CACHE AND RETURN TO CPU
RETURN TO CPU
Y
L4 CACHE CRITICAL METADATA ACCESS FROM TARGET SET
HIT?
Y
WAY PREDICTION CORRECT?
N
Y
N
ACCESS DATA FROM MAIN MEMORY AND RETURN TO CPU
N
PREDICTED MISS?
PREDICTED HIT?
Y
N
ACCESS DATA FROM L4 CACHE AND RETURN TO CPU

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

Table 2. SRAM overhead of baseline L4 DRAM cache

Structure	Number of entries, Size, Latency at 4 GHz (for 22 nm nodes)			On critical path?
	1 KB sector	2 KB sector	4 KB 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 pred.	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	Yes
Total size	418.5KB	572.5KB	570.5KB	

Table 2 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 overhead calculation assumes a 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 1 KB to 4 KB. The footprint table is used to store the already learned sector footprints, while the singleton table is used to store the recently observed singleton sectors' tags so that any singleton misprediction can be corrected. The footprint table is organized to have 4K sets and four ways and the singleton table is organized to have 128 sets and four ways. Both tables exercise NRU replacement.[12] The size of the tag associated with an NRU state cache entry is calculated assuming a 1 GB 4-way L4 cache (the tag size of an NRU state cache entry depends on the number of L4 cache sets). For smaller L4 caches, the NRU state cache overhead is smaller. For example, a 256 MB 4-way L4 cache with 4 KB sectors needs less than 32K tagless entries in the NRU state cache, since the L4 cache has less than 32K sets. Overall, the total SRAM overhead of the baseline L4 DRAM cache varies from 418.5 KB to 572.5 KB, 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,

[12] The sizes of the footprint and singleton tables are borrowed and extrapolated from the Unison cache proposal [16].

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.

In addition to the tag cache, the NRU state cache, and the hit/miss predictor, we incorporate a fourth optimization in our baseline Unison cache design. Since the Unison cache controller injects footprint prefetch requests along with demand requests into the main memory controllers, it is important to ensure that the demand requests do not get delayed due to the burst of footprint prefetch requests. We prioritize the demand requests over prefetch requests in the main memory controllers as follows. Within an open row, demand reads are scheduled before prefetch reads. If there is no request (demand or prefetch) to an open row, the bank is precharged and the oldest demand read or the oldest prefetch read (in that priority order) issues an activate command to its target row. It is important to note that our goal was not to design an optimized main memory access scheduler that adaptively prioritizes demand and prefetch requests. Instead, we incorporate a simple intuitive policy that can offer some benefit in the presence of bursts of large sector footprint prefetches. For a more comprehensive treatment on prefetch-aware main memory access scheduling, the readers can refer to studies devoted to that problem [20].

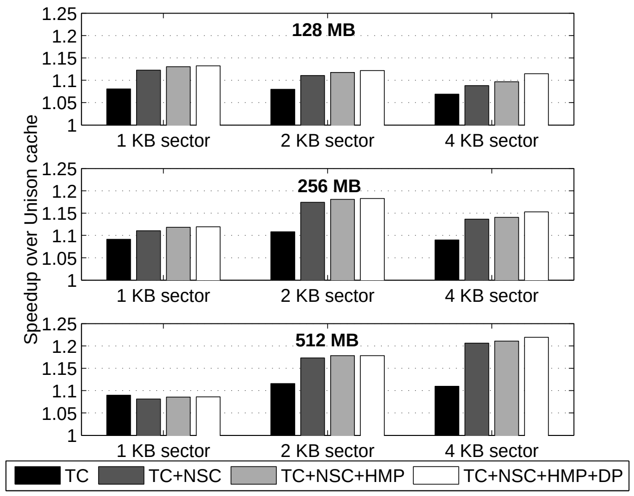

Figure 6. Performance improvement over Unison cache due to introduction of tag cache (TC), NRU state cache (NSC), MAP-I hit/miss predictor (HMP), and demand prioritization (DP).

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). Each panel shows the speedup over the Unison cache for a particular capacity point (top panel: 128 MB, middle panel: 256 MB, bottom panel: 512 MB). For each capacity point, we

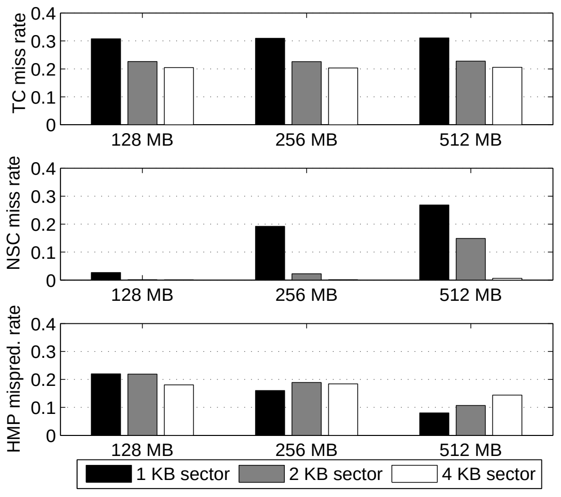

Figure 7. Upper panel: tag cache (TC) miss rate; Middle panel: NRU state cache (NSC) miss rate; Bottom panel: MAP-I hit/miss predictor (HMP) misprediction rate.

show three groups of bars corresponding to the speedup achieved for three sector sizes. Within each group, from left to right, we show four bars quantifying the speedup achieved as we gradually introduce the tag cache, the NRU state cache, the MAP-I hit/miss predictor, and demand prioritization. Figure 7 shows the tag cache miss rate (top panel), NRU state cache miss rate (middle panel), and misprediction rate of the hit/miss predictor (bottom panel). Within each panel, each group of bars represents one L4 cache capacity point and within each group, the bars from left to right correspond to 1 KB, 2 KB, and 4 KB sectored configurations. In both Figures 6 and 7, all results are averaged over thirty multi-programmed mixes.

The tag cache miss rate remains unaffected by the L4 cache capacity, as expected. As the sector size increases, each tag cache entry covers more data leading to lower tag cache miss rate. Even though the tag cache capacity for a 4 KB sectored configuration is half of that for a 2 KB sectored configuration, there is a small improvement in the tag cache miss rate when going from 2 KB sectors to 4 KB sectors. This is primarily because with one tag per 4 KB data, a fraction of the tag cache conflicts that were observed with 2 KB sectors do not arise. 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 even more important. Since the tag cache can save a significant fraction of the metadata bandwidth, its importance increases with increasing DRAM cache capacity. This trend is visible for all three sector sizes as the L4 capacity is scaled up. At 1 KB sector size, this trend is less pronounced due to a 31% tag cache miss rate. Overall, addition of the tag cache always improves performance for the sector sizes and L4 cache capacities considered in this study.

The NRU state cache miss rate slowly increases with L4 cache capacity for a given sector size. This is expected because the number of L4 cache sets increases with increasing L4 cache capacity for a fixed sector size leading to more capacity/conflict misses in the fixed-sized NRU state cache. On the other hand, for a fixed L4 cache capacity, the NRU state cache miss rate decreases with increasing sector size, since the number of L4 cache sets goes down. Overall, the addition of the NRU state cache offers significant performance improvement across the board. With increasing L4 cache capacity, the performance benefits of the NRU state cache increase significantly. This is primarily because the NRU state cache frees up L4 cache bandwidth that can now be used to serve hits more quickly and the volume of L4 cache hits increases with increasing L4 cache capacity. However, at 512 MB with 1 KB sectors, the addition of the NRU state cache actually hurts performance. This is because a 27% NRU state cache miss rate for this configuration leads to a significant volume of dirty replacements from the NRU state cache. These replacements inject writes to the L4 cache causing disturbance to the row locality. Without an NRU state cache, the number of reads and writes to the L4 cache set metadata is more, but such a write gets scheduled before the corresponding row gets closed (note that the row was opened by the set metadata read request). Larger sectors lead to more efficient designs of the tag cache and the NRU state cache making the 4 KB sector size an attractive design point.

The hit/miss predictor offers reasonable prediction accuracy with the minimum accuracy being 78%. With increasing L4 cache capacity, the prediction accuracy improves, in general. This is because the outcomes of the L4 cache lookups get increasingly biased toward hits as the L4 cache capacity increases making it easier for the predictor to offer a correct hint. Across the board, the hit/miss predictor offers about a percentage improvement in performance.

Demand prioritization is particularly helpful at larger sector sizes because the volume of prefetch requests injected in a burst increases significantly with growing sector sizes. This optimization adds 1%-2% of performance improvement in the 4 KB sectored configurations.

Overall, all the four optimizations help improve the performance of the Unison cache, the tag cache and the NRU state cache being the most important additions. Henceforth, the Unison cache with all four optimizations will be used as the baseline for this study and all the optimizations considered in the next section for improving the sector utilization are incorporated on top of this baseline. We, however, note that none of these optimizations are necessary for our proposal to work; we implement these optimizations in the L4 cache to arrive at a better baseline to start with.

To complete the design of the baseline, we need to validate two fundamental design decisions that we have

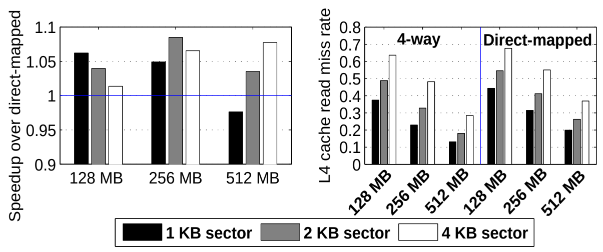

Figure 8. Comparison between four-way set-associative and direct-mapped baselines.

inherited from the Unison cache. First, we need to confirm that for our workload set, a set-associative organization is better than a direct-mapped organization. Second, we need to justify the choice of the NRU replacement policy over a simpler random replacement policy. Figure 8 quantitatively compares a four-way set-associative baseline with a direct-mapped baseline. Both designs exercise tag caches, hit/miss predictors, and demand prioritization. The four-way set-associative baseline exercises the NRU replacement policy. The direct-mapped organization enjoys two advantages. First, it does not need to rely on a way predictor if there is a miss in the tag cache and the hit/miss predictor predicts a hit. Second, it does not need an NRU state cache, thereby eliminating the L4 cache bandwidth consumption arising from the NRU state cache misses and dirty replacements. The left panel of Figure 8 shows the speedup achieved by the four-way set-associative baseline over the direct-mapped baseline. Each of the three groups of bars represents one L4 cache capacity point and each bar within a group represents a sector size. The right panel of Figure 8 shows the L4 cache read miss rate experienced by the two baselines. At all design points, the four-way baseline has a lower read miss rate than the direct-mapped baseline. Except for the 512 MB design with 1 KB sector, the four-way baseline outperforms the direct-mapped baseline. For the 512 MB design with 1 KB sector, the four-way baseline performs 2% worse compared to the direct-mapped baseline. This is due to two reasons. First, at this design point, the L4 cache hit rate is very high and the four-way baseline relies on the way predictor to offer fast hits. However, the way predictor is not accurate all the time. The direct-mapped baseline always fetches the correct data in the cases of L4 cache hits. Second, the NRU state cache miss rate is 27% at this design point causing reasonable bandwidth wastage in the four-way baseline. Nonetheless, since our proposal for improving sector utilization is more effective at larger sector sizes, we would like to use the four-way baseline, which always outperforms the direct-mapped baseline at larger sector sizes. For a fair evaluation, it is important to use the baseline that offers better performance at the design points that are critical for our proposal.

Having decided to use a four-way set-associative baseline, Figure 9 shows a comparison between the NRU and the random replacement policies. The left panel shows the speedup achieved by the NRU policy over the random policy. The right panel shows the L4 cache read miss rates for these two policies. Across the board, the NRU

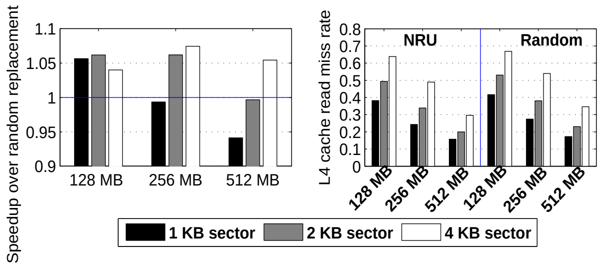

Figure 9. Comparison between NRU and random replacement policies.

policy exhibits better read miss rates than the random policy at all design points. Except for the design points where the NRU state cache miss rate is more than 10% (see Figure 7), the NRU policy outperforms the random policy. Since for larger sector sizes (particularly at 4 KB) the NRU replacement policy performs better and we are more interested in improving the performance of the designs with larger sectors, we choose to use the NRU replacement policy in our baseline four-way L4 cache. However, we note that the NRU state cache may have to be scaled up for the NRU policy to remain competitive as the L4 cache capacity is scaled up. From Figures 8 and 9 we observe that the four-way set-associative baseline with NRU replacement policy remains competitive as long as the number of NRU state cache entries is at least half of the number of L4 cache sets. Fortunately, the scaling cost of the NRU state cache is low for an L4 cache with low associativity (e.g., four). For example, a 16 GB 4-way L4 cache with 4 KB sectors has 1M sets and an NRU state cache with 1M (512K) entries requires 768 KB (448 KB) storage, which is less than the size of one L3 cache way. Nonetheless, in this study, we fix the size of the NRU state cache at 32K entries for all configurations, as already stated. Having decided the baseline design, we now turn to the crux of our proposal on improving the sector utilization.

4. Improving Sector Utilization

We discuss the details of our proposal in this section. We begin our exploration by understanding the decoupled sectored cache architecture, one of the prominent efforts toward improving the space utilization of SRAM-based sectored caches. We use this understanding to develop an efficient sectored DRAM cache architecture.

4.1. Background: Decoupled Sectored Cache

The decoupled sectored 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. The following analysis of decoupled sectored 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 higher data-associativity in the context of our proposal.

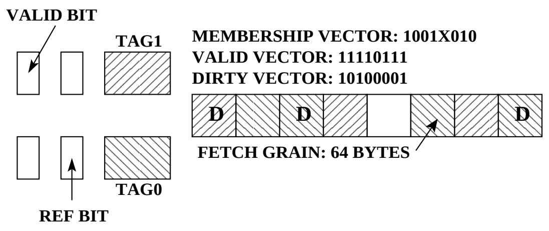

Figure 10. A two-way multiplexed 512-byte sector along with tags, state vectors, REF bits, and valid bits.

Figure 10 shows a 512-byte physical sector with two sectors multiplexed onto it. Each bit position of the membership vector indicates the sector the cache block in that position belongs to. In the next section, we explore how we mould the idea of decoupled sectored cache to design a DRAM cache. This will serve as the starting point of our proposal.

4.2. A Decoupled Sectored DRAM Cache

In this section, we design and evaluate a DRAM cache architecture that is derived from the idea of decoupled sectored cache. In the following, we discuss the implications on the organization of the set metadata and the assisting SRAM structures. We also propose the hierarchical NRU replacement policy suitable for the decoupled sectored DRAM cache.

4.2.1. Implications on Set Metadata Organization

The degree of multiplexing in the decoupled sectored cache directly impacts the critical metadata size per set. For a multiplexing degree of N, the critical metadata in a four-way set needs to maintain $4(N-1)$ additional tags, $4(N-1)$ additional valid bits, and $4S\log_2(N)$ membership vector bits, where S is the number of blocks in a sector. Since we want to restrict the critical metadata size per set to 64 bytes (the critical metadata can be read out with a single CAS command) and the smallest sector size (960 bytes) has S equal to fifteen, 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 256 MB. However, we restrict our design to only 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 4032-byte sector, a degree of multiplexing of even two requires two CAS commands to fetch the critical metadata of the set. This can introduce significant serialization in the case of a tag cache miss. Fortunately, the tag cache miss rate for a 4032-byte sectored organization is reasonably low (about 20% as shown in Figure 7). In the rest of the paper, we will consider decoupled sectored DRAM caches with a degree of multiplexing two only.

The non-critical part of the metadata also needs to be extended by the following additional fields: REF bit, demand vector, program counter, offset value, and core id corresponding to the second tag in each way. We continue to accommodate the REF states of a set within the first 64 bytes of set metadata. The entire metadata of a way still comfortably fits within 64 bytes, which we have borrowed from the sector in each way.

4.2.2. Implications on SRAM Structures

Among the SRAM structures, the tag cache and the NRU state cache entries require expansion in the decoupled sectored cache. 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. Let us suppose that an access to the sector corresponding to tag A replaces a block belonging to tag B. In this process, the tag cache entry corresponding to tag A would be looked up, since the access is to that tag. 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. The following steps are followed to achieve this. As a part of the DRAM cache lookup, the tag cache is looked up for tag A (this is the parent request). Appropriate actions are taken and the partner tag B is read out. A second lookup is made to the same tag cache set for tag B and on a hit, tag B's entry is updated with the new membership, valid, and dirty vectors.

Although the decoupled sectored cache has roughly double the number of tags resident in the L4 cache compared to the baseline Unison cache at any point in time, this does not have any influence on the tag cache miss rate because the tag cache continues to see the same sequence of accesses. However, since the volume of hits offered by the decoupled sectored cache is expected to improve compared to the baseline, the fraction of L4 cache hits that can be covered by the tag cache is expected to drop.

Each entry of the NRU state cache needs to be widened to include the REF bits of the partner tags. For a four-way decoupled sectored cache with degree of multiplexing equal to two, each NRU state cache entry requires four additional bits.

4.2.3. Hierarchical NRU Replacement Policy

The baseline DRAM cache uses a global NRU replacement policy, which selects the global NRU sector for replacement. We devise a new hierarchical sector replacement policy for the decoupled sectored DRAM cache. Let us refer to the two sectors multiplexed (or paired) onto a physical sector frame as partners of each other. The global NRU sector is a poor choice as a victim if its partner sector is currently actively accessed (e.g., the global MRU sector). In such a situation, the newly filled sector and the actively accessed partner sector can thrash each other. Ideally, we would like to victimize a sector whose partner sector is not actively accessed. 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 actively accessed in the recent past. It is important to note that the number of REF bits needed per set for this policy is same as that in the global NRU policy (i.e., twice the associativity of a set for a degree of multiplexing of two).

Once the replacement candidate is selected, all cache blocks belonging to the victim sector are replaced from the physical sector of the way containing the victim. If the requested cache block is at position k of the new sector, it is filled into position k of the physical sector. This may require replacing the block at position k of the physical sector, if that position is still occupied. Similarly, when a block is filled for an already allocated sector (as a result of a block miss or a footprint predicted fill), the block occupying the corresponding position in the physical sector may have to be replaced.

4.2.4. Performance Analysis of the Decoupled Sectored DRAM Cache

Figure 11 summarizes the performance of the decoupled sectored cache design incorporated in the baseline Unison cache design at 128 MB capacity. The results are averaged over thirty multi-programmed mixes. The left panel compares the L4 cache read miss rates of the baseline Unison cache and the decoupled sectored cache. The middle panel compares the L4 cache sector utilization and the right panel shows the performance speedup achieved by the decoupled sectored cache over the baseline. Within each panel, each group of bars corresponds to a sector size noted on the horizontal axis. Across the board, the decoupled sectored cache is able to significantly improve the L4 cache read miss rate (e.g., 64% to 56% for 4 KB sectors) and the L4 cache sector utilization (e.g., 21% to 30% for 4 KB sectors). These improvements lead to 6%, 8%, and 8% speedup for 1 KB, 2 KB, and 4 KB sectors, respectively.

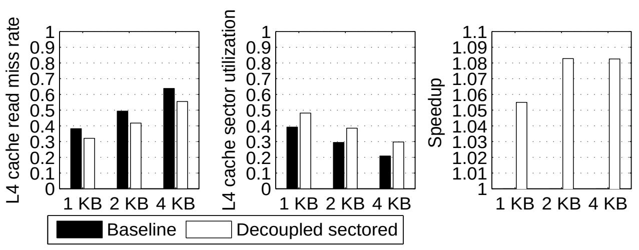

Figure 11. Effectiveness of the decoupled sectored DRAM cache at 128 MB capacity.

Although the performance improvement at 4 KB sector size is expected to be more than at 2 KB sector size, the former design point suffers from an increase in the L4 cache read hit latency due to two CAS commands in the case of a tag cache miss. This is quantified in Figure 12, which shows the average read hit, read miss, and total read latency of the decoupled sectored DRAM cache normalized to the corresponding latencies in the baseline at 128 MB capacity. We include the partial hits also in the group of hits.[13] For 1 KB and 2 KB sector sizes, the decoupled sectored DRAM cache has 3% higher read hit latency than the baseline. This small increase in the average read hit latency is related to the slight loss in row buffer locality caused by the reads of evicted dirty blocks. In the baseline design, all dirty blocks in an evicted sector are read out from the L4 cache at once for writing back to main memory. However, in the decoupled sectored cache, there are two types of evictions, namely, sector eviction and individual block eviction. While the first type of eviction is handled in the same way as the baseline design, the second type of eviction arises when a filling block of an already allocated sector conflicts with the block at the same position of the other multiplexed sector. This type of single-block eviction disrupts the row buffer locality if the evicted block is dirty. Nonetheless, this small increase in the average read hit latency is more than compensated by converting a large number L4 cache read misses to L4 cache read hits. However, at 4 KB sector size, the average hit latency of the decoupled sectored cache is 7% higher than the baseline. A tag cache miss requires two CAS commands to fetch the set metadata at this sector size. This increased hit latency prevents the decoupled sectored cache from realizing its full potential at 4 KB sector size. The decoupled sectored cache enjoys a 7-8% improvement in the average L4 cache read miss latency. This improvement comes from reduced congestion and queuing at the main memory controllers due to a lower volume of L4 cache read misses. Finally, the overall average L3 cache miss latency improves by 8-11% for the decoupled sectored cache compared to the baseline.

Although the two-way multiplexed decoupled sectored cache is able to improve the baseline significantly, it is important to understand possible avenues for further improvement. We have already pointed out that the critical

[13] An L4 cache access is said to experience a partial hit if a prefetch has already been issued for the requested block and the prefetch fill is pending. The partial hit latency is the time from the arrival of the L3 cache miss request at the L4 cache to the arrival of the prefetch response at the L4 cache. Partial hits constitute only about 2% of all L4 cache hits.

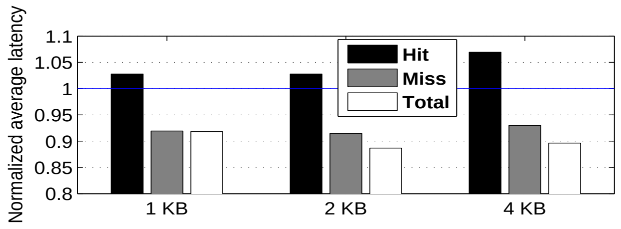

Figure 12. Read hit, read miss, and overall average read latency of the decoupled sectored DRAM cache at 128 MB capacity.

metadata overhead of the decoupled sectored cache does not scale favorably and requires more than one DRAM cache row access at 4 KB sector size. An additional problem arises due to the fact that 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 reasonably high sector utilization leading to high volume of thrashing. 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 at the time a cache block from sector j is filled, where $i \neq j$.

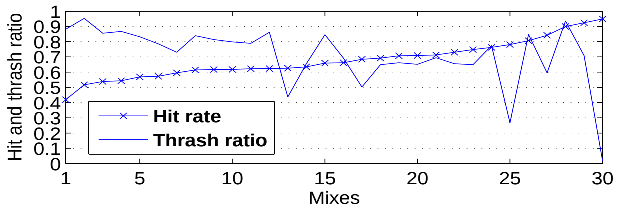

Figure 13. Correlation between thrash ratio and hit rate of the decoupled sectored cache for 128 MB capacity and 1 KB sector.

Figure 13 shows the thrash count and the number of L4 cache read hits, both normalized to the number of L4 cache read lookups for a 128 MB decoupled sectored cache with 1 KB sectors. The thirty 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, in general, increases as the read hit rate falls, except at very high read hit rate values.

These data suggest that achieving higher read hit rates requires mechanisms that can lower the thrash count. In general, there are two ways to lower the thrash count in a decoupled sectored cache. First, one can design more flexible mechanisms for allocating the cache blocks of the multiplexed sectors in the physical sector frame. Second, one can design sector replacement algorithms that can control which pair of sectors is allowed to multiplex on a physical sector frame. Ideally, we would like to pair sectors that are mostly complementary in terms of spatial locality or sector utilization. Our contributions along these two dimensions are discussed next (Sections 4.3 and 4.4).

4.3. Flexible Allocation with Micro-sectors

The maximum amount of flexibility can be incorporated in the decoupled sectored 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. Incorporating this kind of flexibility in cache block allocation requires additional $\lceil \log_2(S) \rceil$ bits to be maintained with each cache block position of the physical sector frame, where S is the number of cache blocks in a sector. These bits indicate the actual position of the cache block within its sector. However, this imposes too high an overhead on the metadata storage. In the following, we develop an implementable, yet reasonably flexible, cache block mapping scheme that offers a range of options trading hit rate for state storage.

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 1 KB 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 in the mapping scheme. Micro-sectors introduce the notion of allocation block to Goodman's nomenclature of transfer block and address block [5].

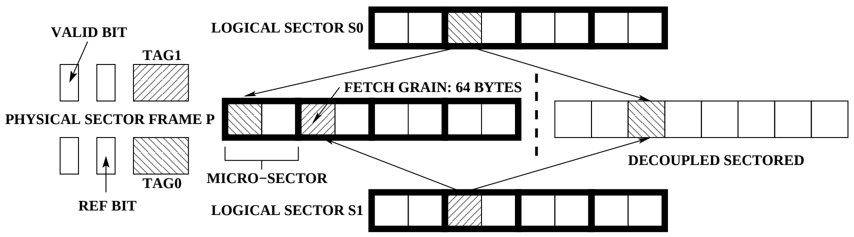

Figure 14. An example showing the advantage of micro-sectors over the decoupled sectored cache.

As an example, consider a decoupled sectored cache with 512-byte sectors and 128-byte micro-sectors. There are four possible micro-sector mapping positions in each physical sector frame. Each sector has eight cache blocks numbered 0 to 7 and four micro-sectors. Figure 14 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 (containing cache blocks 2 and 3 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. This block belongs to the second micro-sector of S1 (containing cache blocks 2 and 3 of S1). 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 14 shows that these two accesses would have conflicted in the decoupled sectored cache without micro-sectors. In general, a filling cache block first finds out if its micro-sector has already been allocated. If yes, it just fills at the appropriate offset within the micro-sector. For instance, in the above example, a subsequent request to cache block 3 of sector S0 would not allocate a new micro-sector, but fill the cache block at the second cache block position within the first micro-sector of P. If a filling cache block needs to allocate a new micro-sector, it first looks for an unoccupied micro-sector in the physical sector frame; if none exists, a micro-sector replacement needs to be carried out.

The first obvious choice for micro-sector replacement is the NRU micro-sector among all the micro-sectors of the physical sector frame. This algorithm, however, degrades hit rate severely 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 a poor choice. 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), which is not currently filling, we opted to replace the micro-sector that has the lowest number of valid blocks. This decision is based on the observation that the utility of a micro-sector directly depends on the number of valid blocks it has.

The micro-sector-based design continues to include two tags, their REF and valid bits, and the valid vector and the dirty vector per physical way. Instead of a membership bit per cache block, now we need a membership bit per micro-sector of the physical sector frame. This bit indicates which of the two sectors a particular physical micro-sector belongs to. Additionally, each physical micro-sector needs to maintain $\lceil \log_2(\mu) \rceil$ bits indicating its actual position within its parent sector, where μ is the number of micro-sectors in a sector. In summary, assuming S cache blocks per sector, the S-bit membership vector of the decoupled sectored cache gets replaced by $\mu + \mu\lceil \log_2(\mu) \rceil$ bits of micro-sector membership vector and micro-sector location vector. Therefore, 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 decoupled sectored cache. 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.

Figure 15 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 S_0 and S_1. Let the four micro-sectors in each sector

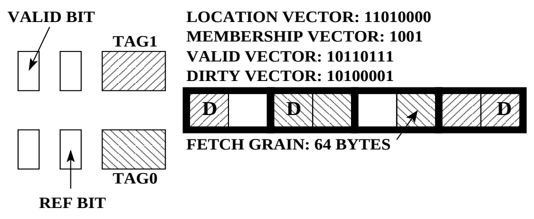

Figure 15. A two-way multiplexed 512-byte sector implementing 128-byte micro-sectors.

be numbered zero to three. The membership vector indicates that the first and the last micro-sectors belong to S_1, while the middle two micro-sectors belong to S_0. The location vector is interpreted as follows. The first two bits 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. In this example, the first physical micro-sector is micro-sector number three in S_1, the second physical micro-sector is micro-sector number one in S_0, etc.. The hit test logic proceeds by first locating the target sector tag (traditional sector tag comparison) and then examining the membership and position vectors of the micro-sectors and the valid bits of the cache blocks residing in the physical sector hosting the target sector.

We now discuss the design changes required for incorporating micro-sectors in the baseline Unison cache. The baseline Unison cache has sector sizes 960, 1984, 4032 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 1920 bytes in the place of 1984 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 1920-byte sectors, we use six micro-sectors each of size 320 bytes. In this case, for DRAM cache capacities more than 256 MB, 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, six bits of membership vector, eighteen bits of location vector, thirty bits each for valid and dirty vectors). For 128 MB and 256 MB capacity, we store only 20 bits of each of the two tags along with an OR of the higher three or two bits of each of the tags in the critical part of the metadata. Only if at least one of the higher two or three bits of the accessed tag is one, more than one CAS command would be needed to determine hit/miss in the case of a tag cache miss.[14] This happens only if the amount of the installed physical memory exceeds 64 TB or 32 TB when the DRAM cache capacity is 256 MB or 128 MB, respectively. We believe that this is an unlikely situation, given today's typical physical memory sizes. For 4032-byte sectors, we use seven micro-sectors each of

[14] 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.

size 576 bytes. In this case, within the critical metadata per way, we can store sixteen 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, seven bits of membership vector, 21 bits of location vector, and 63 bits of valid vector. Only if a physical address lies beyond 2 TB, 4 TB, 8 TB, or 16 TB respectively for DRAM cache capacities 128 MB, 256 MB, 512 MB, or 1 GB, more than one CAS command may be needed to decide hit/miss. In summary, for all sector sizes under consideration in this study, the Micro-sector cache rarely requires more than one CAS to fetch the set metadata on a tag cache miss.

In addition to the changes in the metadata layout, each tag cache entry needs to accommodate the location and the membership vectors. It can be seen that these two vectors together are significantly less in size compared to the membership vector of the decoupled sectored cache for 1920-byte and 4032-byte sectors. Each way predictor entry also needs to be augmented because predicting the way of a requested tag is not enough for fetching the requested block. Each way predictor entry now contains a membership bit indicating which of the two tags in the predicted way this entry corresponds to. Additionally, it stores the micro-sector membership and micro-sector location vectors to compute the physical offset of the requested block. It is important to note that the way predictor is still a tagless direct-mapped array and as a result, the way and the offset obtained from a way predictor lookup are just predictions and must be verified, as usual.

Table 3. SRAM overhead of Micro-sector cache

Structure	Number of entries, Size, Latency at 4 GHz (for 22 nm nodes)			On critical path?
	1 KB sector	2 KB sector	4 KB 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 pred.	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	4K, 1.5KB, 1 cycle	Yes
Total size	710.5KB	876.5KB	794.5KB	

Table 3 summarizes the SRAM overhead of the L4 Micro-sector cache. Overall, our proposal's total SRAM overhead varies from 710.5 KB to 876.5 KB, which is less than one L3 cache way. Referring back to Table 2, we see that our proposal requires at most 300 KB additional SRAM storage compared to the baseline and most of this additional SRAM storage is devoted to the tag cache. As shown in Table 3, 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 4 GHz in our configuration) between the completion of these two fetch operations. 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

Micro-sectoring enables flexible allocation of the sectors multiplexed onto a physical sector frame. However, poor pairing of sectors that multiplex onto a physical sector frame can prevent micro-sectoring from realizing its full potential. The hierarchical NRU replacement policy takes the recent activities into account while selecting a victim. While recent activities are good indicators of the current hotness of the partner sector of the victim, it is also important to ensure that the partner sector of the victim and the newly allocated sector have complementary spatial locality so that they do not thrash each other. Since sector utilization can serve as a reasonably good indicator of spatial locality, we take into account the sector utilization of the resident sectors in a set ($2n$ sectors in an n-way set) in addition to the REF bits for selecting the victim sector and propose a spatial locality-aware sector replacement algorithm.

We discuss the 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 can also replace from the next NRU way. In other words, the victim is always selected from the top two NRU ways. If the new sector's predicted utilization is low (at most half), it is allowed to probe one extra way deeper into the access recency stack. Doing so offers more replacement options without introducing too much thrashing because the predicted utilization of the new sector is at most half.

The algorithm first orders the four physical sector ways in a set from NRU to non-NRU. Note that with a single REF bit per physical sector way, we can order the ways with REF bits reset first and then those with REF bits set (the ways with REF bits set are more recently used than the remaining ways). 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 decoupled sectored cache proposal introduced another orthogonal dimension called data-associativity for mapping cache blocks to physical sectors. If the data-associativity is A_d and the set-associativity of the cache 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. The implication of this additional mapping flexibility is that each element of the membership vector gets extended by $\log_2(A_d)$ bits. These additional bits are used to locate a cache block's physical data way, which can now be different from the block's tag way. Since the metadata overhead of the decoupled sectored cache with data-associativity of one is already high, we consider the possibility of introducing higher data-associativity in our Micro-sector cache design. We will assume a data-associativity of four in the following discussion. This implies that in a four-way set-associative cache, a micro-sector can be mapped to any micro-sector position in any of the four physical ways.

Figure 16 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 sector can be allocated to any of the ways belonging to its data-associative

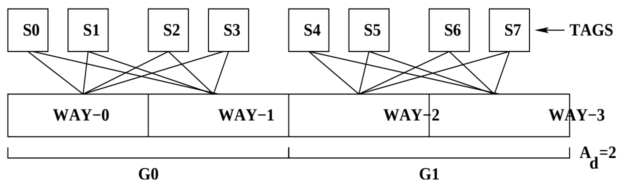

Figure 16. A two-way multiplexed four-way set with data-associativity (DA) of two.

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, we have five micro-sectors and as a result, the membership vector length increases by only ten bits. This can be comfortably accommodated within 128-bit critical metadata per way. This, however, is not true for bigger sector sizes. Without any optimization, the 1920-byte and 4032-byte sectored designs would require two CAS commands for fetching the set metadata on a tag cache miss. For 1920-byte sectored designs, we maintain a separate demand vector (as in the baseline 4032-byte sectored design) and disassociate the dirty vector from the critical metadata. The valid and the dirty vectors separately keep track of the valid and dirty blocks in a sector. This re-organization allows us to fetch the entire set metadata on a tag cache miss through a single CAS command for four-way data-associative designs with 1920-byte sectors. The penalty of this re-organization is that on replacing a dirty tag cache entry, we need to execute two read CAS operations and two write CAS operations for updating the set metadata. However, the reduction in the critical path latency on a tag cache miss outweighs this disadvantage. Unfortunately, for a 4032-byte sectored design, there is no option but to use two CAS commands for fetching the set metadata on a tag cache miss.

The way predictor and the tag cache organizations pose bigger problems. Since the micro-sectors of a sector can now be scattered over multiple different data ways, the way predictor must be designed to offer micro-sector-specific predictions. In other words, each way predictor entry can no longer correspond to a sector, but must correspond to a micro-sector. This naturally puts more pressure on the way predictor if the number of entries in it is left unchanged as the baseline. Fortunately, streaming through a micro-sector is short-lived and the same-size way predictor can still offer reasonable benefits, particularly at 1 KB sector size where we have five micro-sectors per sector. At 2 KB and 4 KB sector sizes, the number of micro-sectors per sector increases to six and seven, respectively. Additionally, each way predictor entry needs to be widened to accommodate a longer membership vector.

The tag cache entry containing the critical metadata of the accessed tag way is no longer sufficient for determining hit/miss because the membership vectors and the location vectors of all the four data-associative ways are needed for this purpose. 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 cache, for fairness of evaluation, we size the set-tag cache to have only 8K entries for 960-byte and 1920-byte sector sizes and 4K entries for 4032-byte sector size (one-fourth of the baseline tag cache). The set-tag cache is organized to have the same associativity as the tag cache. The set-tag cache is looked up using the lower bits of the L4 cache set index and the remaining higher set index bits are used as the tag of the set-tag cache entries. When a new tag is allocated to an L4 cache way, the corresponding set-tag cache entry is also updated to reflect the new constitution of the set. The set-tag cache employs the NRU replacement policy and each set-tag cache entry has its own valid, dirty, and REF bits. On replacing a dirty set-tag cache entry, the entire set metadata of the replaced entry is read out and updated in the L4 cache.

4.6. Performance of Micro-sector Cache

In this section, we evaluate the performance of the Micro-sector cache at 128 MB capacity. 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.

4.6.1. Effectiveness of Micro-sectors

Figure 17 quantifies the benefits of micro-sector allocation for a 128 MB L4 cache when this optimization is incorporated in the baseline DRAM cache design. The left panel compares the L4 cache read miss rate, the middle panel compares the sector utilization, and the rightmost panel shows the speedup over baseline. Within each panel, each group of bars corresponds to a sector size noted on the horizontal axis. Each group contains four bars representing the baseline, decoupled sectored cache (DS), 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 as discussed in Section 4.3 (Micro-sectorPopR). The rightmost panel does not show the baseline bar because it is at 1.0. All results are averaged over thirty multi-programmed workloads.

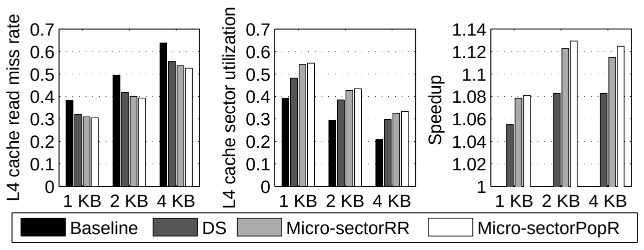

Figure 17. Effectiveness of micro-sectors at 128 MB capacity.

The left and middle panels show that the L4 cache read miss rate and sector utilization improve due to the flexibility offered by micro-sector placement. The rightmost panel shows that micro-sector placement is able to offer reasonable improvement in performance. 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 1 KB, 2 KB, and 4 KB sector sizes, respectively. The corresponding speedup figures for the decoupled sectored cache are 6%, 8%, and 8%, respectively.

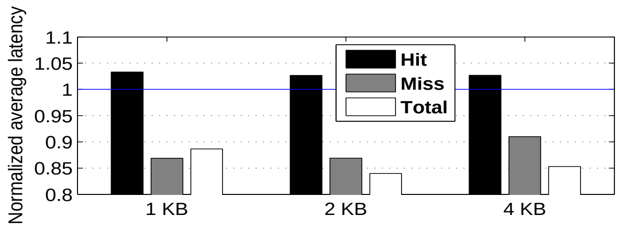

Figure 18. Read hit, read miss, and overall average read latency of the DRAM cache exercising micro-sectors at 128 MB capacity.

Figure 18 quantifies the average L4 cache read hit, miss, and total latency after the introduction of micro-sectors normalized to the baseline. While the average read hit latency increases by 3% for all sector sizes, the savings in the miss latency vary from 9% to 13% and the savings in the total latency vary from 11% to 16%. The decoupled sectored cache suffered from a 7% increase in the average read hit latency for 4 KB sector size (see Figure 12). Micro-sectoring helps resolve this problem.

4.6.2. Effectiveness of Locality-aware Sector Tag Allocation

Figure 19 quantifies the benefits of the locality-aware sector tag allocation algorithm for a 128 MB L4 cache. The left panel compares the L4 cache read miss rate, the middle panel compares the sector utilization, and the rightmost panel shows the speedup over baseline. Within each panel, each group of bars corresponds to a sector size noted on the horizontal axis. Each group contains four bars representing the baseline, the baseline with micro-sectors exercising the valid block population-aware micro-sector replacement (Micro-sectorPopR), the baseline with micro-sectors exercising the valid block population-aware micro-sector replacement and locality-aware sector tag replacement (Micro-sectorPopR+LAR), and the baseline with a double-sized tag cache (Baseline+2xTC). The configuration in the last bar compensates for the additional SRAM needed by the micro-sectored designs. The rightmost panel does not show the baseline bar because it is at 1.0. All results are averaged over thirty multi-programmed workloads.

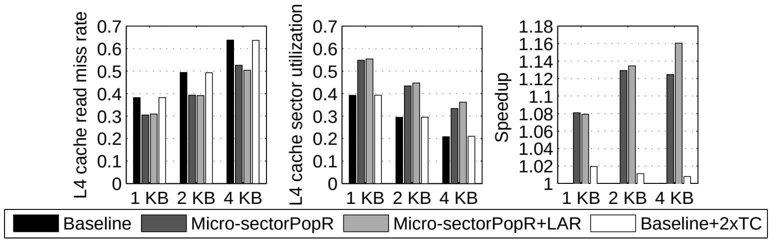

Figure 19. Effectiveness of locality-aware replacement at 128 MB capacity.

The results show that the locality-aware sector tag allocation algorithm is very effective at 4 KB sector size (miss rate decreases by 4% and sector utilization improves by 9% compared to Micro-sectorPopR). It shows small benefits at 2 KB sector size. Overall, the Micro-sectorPopR+LAR configuration improves performance by 8%, 14%, and 16% over the baseline for 1 KB, 2 KB, and 4 KB 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 miss rate and sector utilization unaffected. The rightmost panel shows that the Baseline+2xTC configuration improves performance by only 1-2% over the baseline.

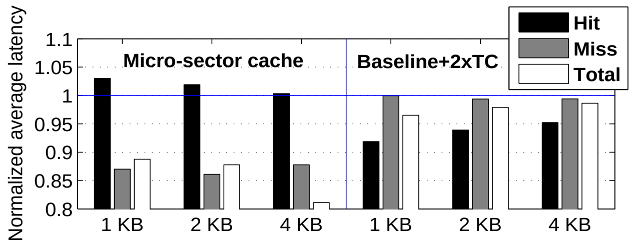

Figure 20. Normalized read hit, read miss, and overall average read latency of the Micro-sector cache and the baseline with double-capacity tag cache at 128 MB capacity.

Figure 20 shows the average read hit, miss, and total latency for the Micro-sector cache and the baseline with a double-sized tag cache. The savings in the overall average L4 cache read access latency enjoyed by the Micro-sector cache ranges from 11% to 19%. While the Baseline+2xTC configuration reduces the average read hit latency by 5-8%, the savings in the overall L4 cache access latency are small (1-3% only).

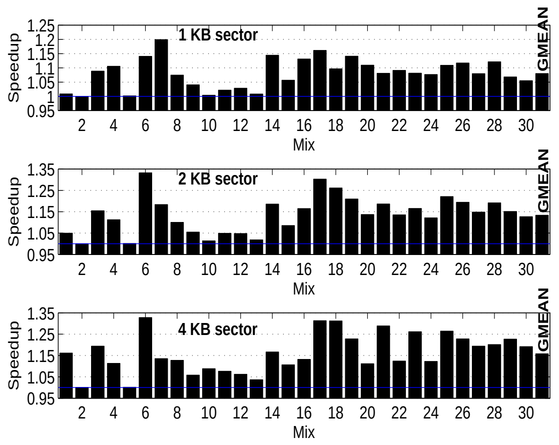

Figure 21. Speedup of the mixes at 128 MB capacity for 1 KB (top panel), 2 KB (mid panel), and 4 KB (bottom panel) sector sizes.

The performance improvement enjoyed by the Micro-sector cache arises from better sector utilization leading to improved read miss rate. The read 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 volume of both types of misses through better sector allocation and better micro-sector mapping leading to lowered thrashing. The sector read misses per kilo instruction (MPKI) for the baseline 128 MB DRAM cache is 2.1, 1.9, and 1.8 for 1 KB, 2 KB, and 4 KB 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 128 MB DRAM cache is 2.6, 4.3, and 6.6 for 1 KB, 2 KB, and 4 KB sector sizes, respectively. For the Micro-sector cache, the corresponding MPKI values are 2.5, 3.7, and 5.3.

Figure 21 details the speedup achieved by the Micro-sector cache for each mix listed in Table 1 at 128 MB capacity. Note the different scales of the y-axis for different panels. For 1 KB sector size (top panel), the maximum gain achieved by a mix is 20% and no mix loses in performance. For 2 KB and 4 KB sector sizes, the maximum gain is 33%. For all three sector sizes, several mixes gain more than 10%. On average, the Micro-sector cache improves performance by 8%, 14%, and 16% over the baseline for 1 KB, 2 KB, and 4 KB sector sizes, respectively.

4.6.3. Impact of Data-associativity

Figure 22 summarizes the effectiveness of the Micro-sector cache with a data-associativity of four (Micro-sector cache + DA4) at 128 MB capacity. The left panel shows that data-associativity can further reduce the L4 cache read miss rate. The middle panel shows that data-associativity can significantly improve sector utilization. The rightmost panel, however, shows that data-associativity of four fails to add any performance benefit to the Micro-sector cache. In fact at 1 KB and 4 KB sector sizes, the design with data-associativity of four performs significantly worse than the Micro-sector cache. Only at 2 KB sector size, the design comes close to the Micro-sector cache in performance.

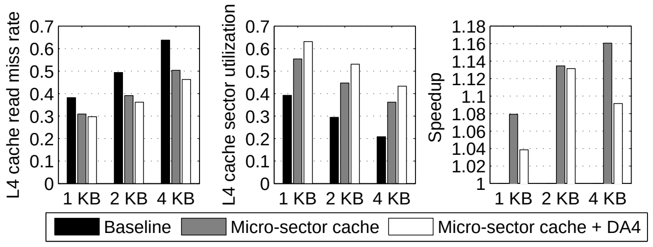

Figure 22. Effectiveness of data-associativity at 128 MB capacity.

At 1 KB sector size, the primary reason for the poor performance of the design with data-associativity of four is much higher miss rate of the set-tag cache compared to the tag cache. The set-tag cache miss rates are 40%, 32%, and 31% at 1 KB, 2 KB, and 4 KB sector sizes, while the tag cache miss rates are 31%, 23%, and 21%, respectively. At 2 KB sector size, the negative effect of higher set-tag cache miss rate is almost compensated by the improved L4 cache read miss rate. At 4 KB sector size, a set-tag cache miss requires two CAS commands to fetch the critical metadata in the design with data-associativity of four. This, when compounded with a higher set-tag cache miss rate, leads to a significant performance loss compared to the Micro-sector cache. Overall, the Micro-sector cache improves baseline performance by 8%, 14%, and 16% at 1 KB, 2 KB, and 4 KB sector sizes, while the design with data-associativity of four offers 4%, 13%, and 9% performance benefits, respectively.

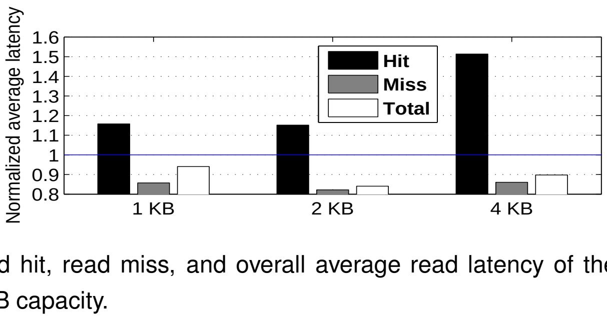

Figure 23. Normalized read hit, read miss, and overall average read latency of the Micro-sector cache with data-associativity of four at 128 MB capacity.

Figure 23 quantifies the average L4 cache read hit, miss, and total latency normalized to the baseline for the

Micro-sector cache with data-associativity of four. As expected, the average hit latency increases by 16%, 15%, and 51% respectively for 1 KB, 2 KB, and 4 KB sector sizes. Since this design is able to reasonably improve the L4 cache read miss rate (see left panel of Figure 22), the reduced level of congestion at the main memory controllers helps improve the average read miss latency by 14%, 18%, and 14% at 1 KB, 2 KB, and 4 KB sector sizes, respectively. Overall, the average L4 cache read access latency reduces by 6%, 16%, and 10% compared to the baseline at the three sector sizes. However, referring back to Figure 20, we see that the savings in the average L4 cache read access latency enjoyed by the Micro-sector cache are much higher at 1 KB and 4 KB sector sizes. In the rest of this study, we will consider data-associativity of only one, which is the default data-associativity for the Micro-sector cache.

5. Detailed Performance Analysis of Micro-sector Cache

In this section, we present a detailed analysis of the Micro-sector cache. We begin by evaluating the Micro-sector cache in the context of bigger capacities (Section 5.1) and larger core-count systems (Section 5.2). In Section 5.3, we present a quantitative comparison of the Micro-sector cache with the Bimodal cache, which also exercises large allocation units. Section 5.4 evaluates the Micro-sector cache on the entire set of the selected 28 application-input combinations spanning 24 different SPEC CPU2006 applications, as was discussed in Section 1. We close this study by comparing the Micro-sector cache with the state-of-the-art designs that exercise fine-grain allocation units (Section 5.4.1).

5.1. Sensitivity to L4 Cache Capacity

Figure 24 summarizes the average L4 cache read miss rate (left panel) and sector utilization (right panel) at 256 MB and 512 MB capacity for the baseline and our Micro-sector cache proposal. The results are averaged over thirty mixes. Each group of bars corresponds to a sector size. Across the board, our proposal continues to save L4 cache misses resulting from significantly improved sector utilization. At 256 MB and 512 MB with 4 KB sectors, our proposal has L4 cache read miss rates of 34% and 20% respectively, while the baseline read miss rates are 49% and 30%.

Figures 25 and 26 show the detailed speedup achieved by our proposal on the thirty mixes at 256 MB and 512 MB capacity, respectively. The last bar in each panel shows the average speedup. The speedup figures are particularly impressive for 4 KB sector size. At 256 MB with 4 KB sectors, our proposal achieves an average

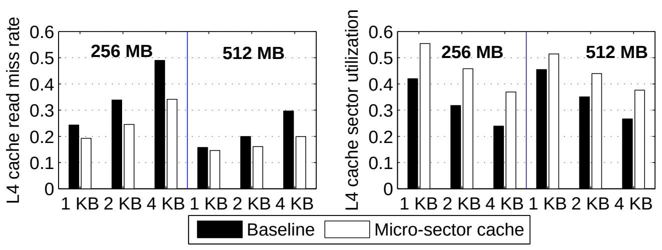

Figure 24. Effectiveness of Micro-sector cache at 256 MB and 512 MB.

speedup of 16%. More than two-third of the mixes gain more than 10%. At 512 MB with 4 KB sectors, the average speedup is 8% and the maximum speedup is 30%.

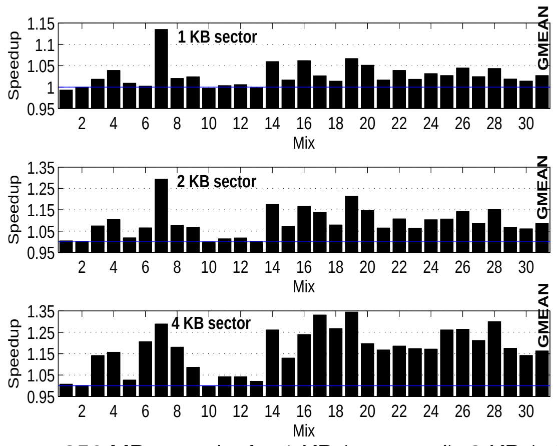

Figure 25. Speedup of the mixes at 256 MB capacity for 1 KB (top panel), 2 KB (mid panel), and 4 KB (bottom panel) sector sizes.

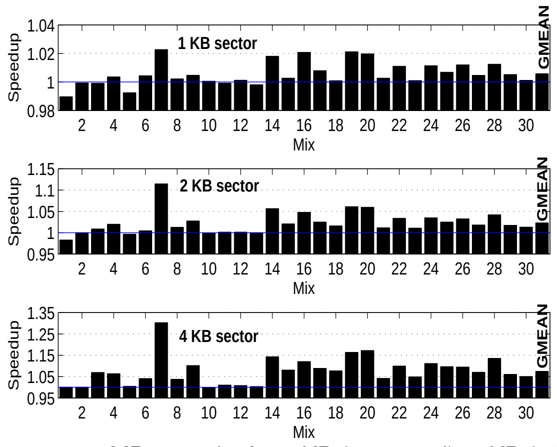

Figure 26. Speedup of the mixes at 512 MB capacity for 1 KB (top panel), 2 KB (mid panel), and 4 KB (bottom panel) sector sizes.

Figure 27 shows the average hit, miss, and total L4 cache access latency for the Micro-sector cache normalized to the baseline. While the average hit latency of the Micro-sector cache is close to that of the baseline, the savings in the average miss and total latency enjoyed by the Micro-sector cache are significant. The improvement in the average miss latency of the Micro-sector cache is a result of reduced volume of L4 cache misses leading to reduced congestion in the main memory. The average total access latency of the Micro-sector cache improves due to a compound effect of increased volume of hits and reduced miss latency.

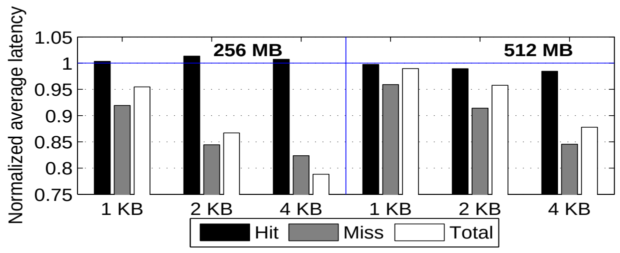

Figure 27. Normalized read hit, read miss, and overall average read latency of the Micro-sector cache at 256 MB and 512 MB capacity.

5.2. Scaling to Larger Core-count Systems

This section evaluates our proposal on a 32-core system with 512 MB and 1 GB L4 caches. We scale up the number of cores in our simulated machine to 32 and the shared L3 cache to 32 MB. The hit/miss predictor of the L4 cache is scaled up to support 32 cores. All other configuration parameters are left unchanged. We continue to simulate an on-die two-dimensional mesh interconnect. The number of copies of each application in each workload mix listed in Table 1 is doubled to build the set of thirty 32-way multi-programmed mixes (i.e., a 32-way mix is equivalent to two copies of a 16-way mix).

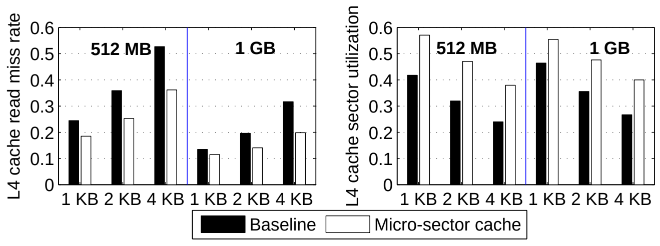

Figure 28. Effectiveness of Micro-sector cache at 512 MB and 1 GB with 32 cores.

Figure 28 summarizes the average L4 cache read miss rate (left panel) and sector utilization (right panel) at 512 MB and 1 GB capacity for the baseline and our Micro-sector cache proposal. The results are averaged over thirty mixes. Each group of bars corresponds to a sector size. The Micro-sector cache continues to improve sector utilization across the board and enjoys significant savings in the volume of the L4 cache misses. At 512 MB and 1 GB with 4 KB sectors, our proposal has L4 cache read miss rates of 36% and 20% respectively, while the baseline read miss rates are 53% and 32%. The miss rate improvements achieved by the Micro-sector cache at 1 KB and 2 KB sector sizes are also reasonable.

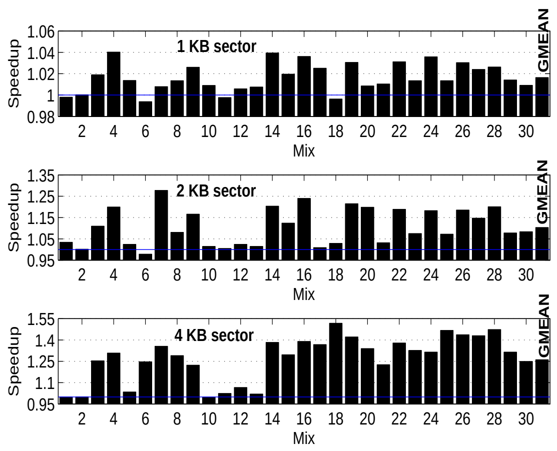

Figure 29. Speedup of the mixes at 512 MB capacity for 1 KB (top panel), 2 KB (mid panel), and 4 KB (bottom panel) sector sizes and 32 cores.

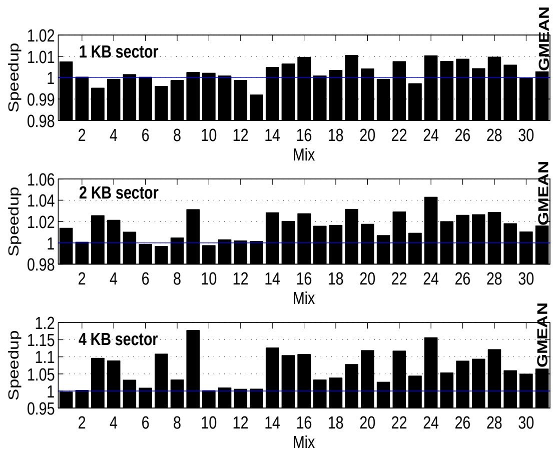

Figure 30. Speedup of the mixes at 1 GB capacity for 1 KB (top panel), 2 KB (mid panel), and 4 KB (bottom panel) sector sizes and 32 cores.

Figures 29 and 30 show the detailed speedup achieved by our proposal on the thirty mixes at 512 MB and 1 GB capacity, respectively for a 32-core system. The last bar in each panel shows the average speedup. At 512 MB with 4 KB sectors, our proposal achieves an average speedup of 26%. It is important to note that a 32-core system can demand very high bandwidth from the main memory system if the L4 cache miss rate is high. This can lead to significant congestion and queuing in a dual-channel main memory system. We have already noted that at 512 MB with 4 KB sectors, the baseline L4 cache miss rate is 53%. The Micro-sector cache brings this miss rate down to 36%. This large saving in the volume of misses results in a 26% performance speedup. At 1 GB with 4 KB sectors, the average speedup is 7% and the maximum speedup is 18%.

5.3. Comparison with Bimodal Cache

The Bimodal cache adapts between big and small blocks depending on the utilization of the big blocks [6]. It varies the associativity of a fixed-sized set to achieve this. We evaluate a Bimodal cache with a set size of 4 KB. Each set can be independently and dynamically configured to have either four or nineteen ways. In the first configuration, each of the four ways has a big 1 KB block, which is entirely fetched on a miss. In the second configuration, each set has three big ways each of size 1 KB and sixteen small ways each of size 64 bytes. To be able to use both the configurations, the set metadata must be sized to accommodate nineteen ways. This requires two CAS operations to access the set metadata, as in the original Bimodal cache proposal. Allowing configurations to have a larger number of small ways 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 in functionality) with 64K entries (double compared to the largest tag cache of the Micro-sector cache), respectively. A way locator miss requires two CAS operations to fetch the entire set metadata. We also note that the Bimodal cache does not maintain NRU replacement states for the ways within a set. At the time of replacement, if a way locator access can identify a couple of top MRU ways within the target set, a random way other than these MRU ways is victimized; if the MRU ways cannot be identified, a random way from the set is victimized.

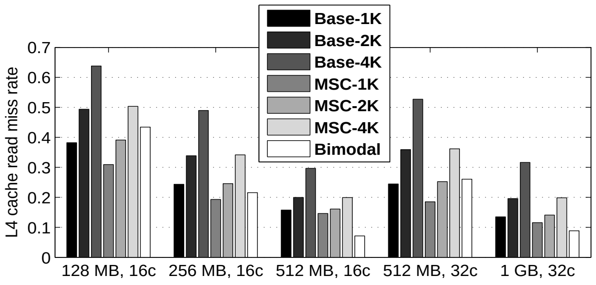

Figure 31. Comparison of L4 cache read miss rate.

Figure 31 compares the L4 cache read miss rate between the baseline, the Micro-sector cache, and the Bimodal cache. The results are averaged over thirty mixes. Each group of bars corresponds to one L4 cache capacity and core count combination (16c refers to a sixteen-core system and 32c refers to a 32-core system). Within each group from left to right, we show the baseline for 1 KB, 2 KB, and 4 KB sector sizes (Base-1K, Base-2K, and Base-4K), the Micro-sector cache for 1 KB, 2 KB, and 4 KB sector sizes (MSC-1K, MSC-2K, and MSC-4K), and the Bimodal cache. Across the board, the Bimodal cache shows reasonably low read miss rate. In general, we expect the Bimodal cache to have a read miss rate that is lower than the read miss rate of Base-1K because the big ways of the Bimodal cache are 1 KB in size. Additionally, the Bimodal cache is expected to enjoy benefits arising from the dynamically configured small ways. Only for 128 MB capacity with sixteen cores and 512 MB capacity with 32 cores, the Bimodal cache has a read miss rate that is higher than the read miss rate of Base-1K. For these two configurations, the departure from the strict NRU replacement hurts the Bimodal cache. The Bimodal cache's read miss rate also compares favorably with the Micro-sector cache. At 512 MB with sixteen cores and 1 GB with 32 cores, the Bimodal cache offers better read miss rates than the Micro-sector cache with all three sector sizes. In the remaining three configurations, the Micro-sector cache for some of the sector sizes enjoys better read miss rates compared to the Bimodal cache.

Despite the promising read miss rates of the Bimodal cache, Figure 32 shows that the Bimodal cache fails to offer good performance. This figure summarizes the performance speedup achieved by the baseline and the Micro-sector cache compared to the Bimodal cache. The results are averaged over thirty mixes. Only at 1 GB with 32 cores, the baseline and the Micro-sector cache with 1 KB sectors (Base-1K and MSC-1K) deliver 2% worse performance compared to the Bimodal cache. The high NRU state cache miss rate causing bandwidth wastage in this configuration hurts the baseline and the Micro-sector cache. At 128 MB with 16 cores, the baseline with 1 KB sectors delivers the same level of performance as the Bimodal cache. For all other configurations, the baseline and the Micro-sector cache outperform the Bimodal cache by wide margins (by 2% to 33%).

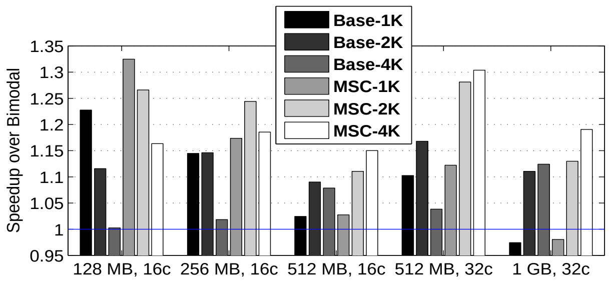

Figure 32. Speedup normalized to Bimodal cache.

To understand the reason behind the poor performance of the Bimodal cache, Figures 33 and 34 quantify respectively the average hit latency and the average miss latency of the baseline normalized to the Bimodal cache. Each group of bars corresponds to one combination of L4 cache capacity and core count. Within each group, three sector sizes are shown. In most of the configurations, the baseline enjoys a better average hit latency than the Bimodal cache. The average hit latency of the Bimodal cache suffers due to two CAS operations required for fetching the set metadata on a way locator miss. For a few configurations, the baseline hit latency is worse than that of the Bimodal cache due to the bandwidth demand arising from the high volume of NRU state cache misses in the baseline.

Turning to Figure 34, we observe that across the board, the average miss latency of the baseline is far better than that of the Bimodal cache. Compared to the baseline, the Bimodal cache demands much higher bandwidth

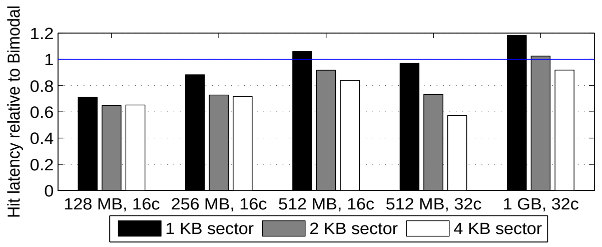

Figure 33. Average L4 cache hit latency of baseline normalized to Bimodal cache.

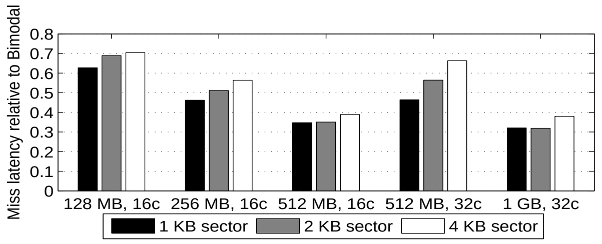

Figure 34. Average L4 cache miss latency of baseline normalized to Bimodal cache.

from the main memory system when fetching a big way. The baseline employs a footprint prefetcher and fetches the portions of a sector that are likely to get used. The Bimodal cache fetches the entire big block. Unfortunately, much of this additional bandwidth is wasted because parts of the fetched big block are not used, especially for the applications with low sector utilization. Although the Bimodal cache detects this problem and dynamically learns to adapt the block size for a limited number of ways, the adaptation is bimodal in nature leading to significant wastage. The extra bandwidth pressure on the main memory system gets translated to high queuing delays leading to significantly larger average miss latency in the Bimodal cache compared to the baseline. In summary, the Bimodal cache offers promising miss rates, but fails to deliver good performance due to large hit and miss latency.

5.4. Performance on Larger Set of Workloads

This section analyzes the performance of our proposal on a larger selection of workloads (28 application-input combinations) spanning 24 different SPEC CPU 2006 applications. All evaluations are carried out by executing each workload in rate mode meaning that on a simulated n-core machine, n copies of the workload are executed. We partition the 28 workloads into two groups based on the sector utilization for the 128 MB baseline. Figures 35 and 36 quantify the sector utilization of the workloads with high and low utilization respectively in the 128 MB baseline. The thirteen workloads in the high utilization category exhibit an average sector utilization of 92%, 89%, and 80% respectively for 1 KB, 2 KB, and 4 KB sector sizes at 128 MB capacity (Figure 35). None of these workloads exhibit a sector utilization of less than 60% even with 4 KB sectors (this is the criterion used to classify the workloads into two groups). On the other hand, the fifteen workloads in the low utilization category exhibit an average sector utilization of 41%, 32%, and 24% respectively for 1 KB, 2 KB, and 4 KB sector sizes at 128 MB capacity (Figure 36).

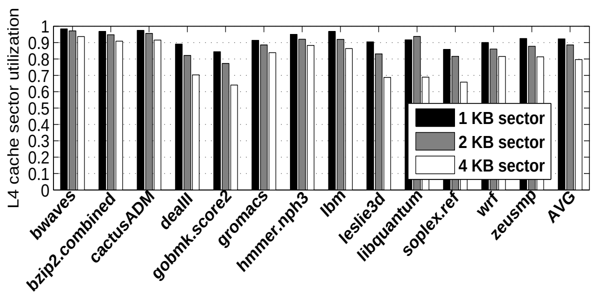

Figure 35. Sector utilization for the workloads with high utilization in 128 MB baseline.

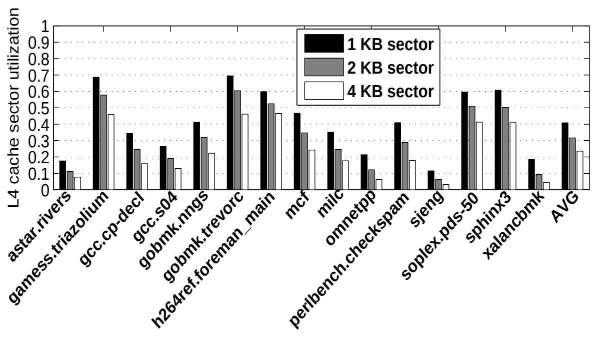

Figure 36. Sector utilization for the workloads with low utilization in 128 MB baseline.

Figures 37 and 38 compare the average sector utilization exhibited by the baseline and the Micro-sector cache for the high and low sector utilization groups, respectively. The results are shown for five different combinations of L4 cache capacity and core count (16c refers to a sixteen-core system and 32c refers to a 32-core system). Each group of bars represents a sector size noted on the horizontal axis. As expected, for the high utilization group, the baseline and the Micro-sector cache achieve similar sector utilization. For the low utilization group, the Micro-sector cache improves the sector utilization by moderate to significant amounts across the board.

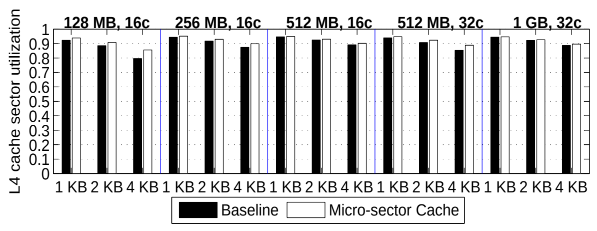

Figure 37. Comparison of sector utilization for the workloads with high utilization.

Figures 39 and 40 quantify the performance speedup achieved by the Micro-sector cache over the baseline for the workload groups with high and low sector utilization, respectively. For the workloads with high sector utilization, the Micro-sector cache and the baseline deliver similar levels of performance, as expected. The Micro-sector cache exhibits a performance speedup of at most 1.5%. For the workloads with low sector utilization, the

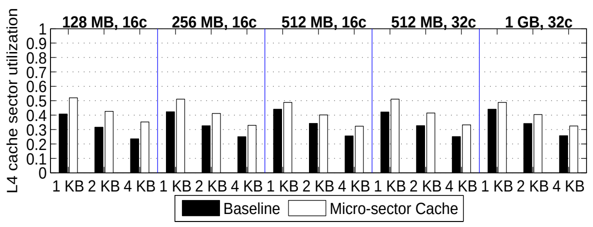

Figure 38. Comparison of sector utilization for the workloads with low utilization.

Micro-sector cache offers reasonable performance benefits across the board. The maximum gain enjoyed by the Micro-sector cache is 11% observed in a 32-core system with a 512 MB L4 cache.

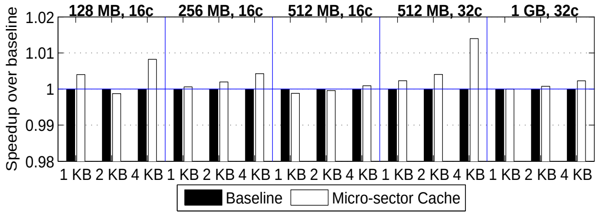

Figure 39. Effectiveness of the Micro-sector cache for the workloads with high sector utilization.

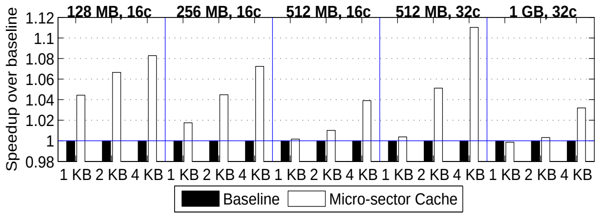

Figure 40. Effectiveness of the Micro-sector cache for the workloads with low sector utilization.

To understand how the Bimodal cache performs on this wider set of workloads, Figures 41 and 42 show the performance of the baseline (Base-1K, Base-2K, Base-4K) and the Micro-sector cache (MSC-1K, MSC-2K, MSC-4K) relative to the Bimodal cache for the workload groups with high and low sector utilization, respectively. Each group of bars corresponds to one combination of L4 cache capacity and core count. In general, the Bimodal cache delivers worse performance compared to the baseline and the Micro-sector cache. We have already analyzed the reasons for poor performance of the Bimodal cache in Section 5.3. Only in a few configurations, the baseline performs worse than the Bimodal cache by a few percentages (at most 4%). The Micro-sector cache performs worse than the Bimodal cache only in one configuration, namely, 128 MB with sixteen cores for workloads with high sector utilization; the performance gap in this case is only 2%. In all these cases where the baseline or the Micro-sector cache performs worse than the Bimodal cache, the sector size is 4 KB and the primary reason for under-performance of the baseline or the Micro-sector cache is worse utilization of the L4 cache space compared to the Bimodal cache.

Figures 43 and 44 quantify the L4 cache read miss rates of the baseline (Base-1K, Base-2K, Base-4K), the

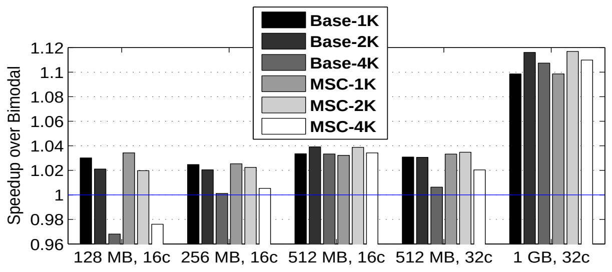

Figure 41. Speedup relative to Bimodal cache for the workloads with high sector utilization.

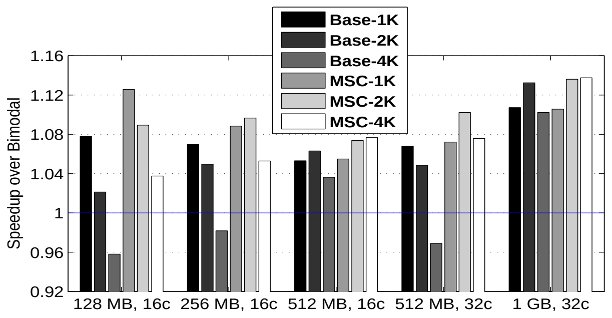

Figure 42. Speedup relative to Bimodal cache for the workloads with low sector utilization.

Micro-sector cache (MSC-1K, MSC-2K, MSC-4K), and the Bimodal cache for the workload groups with high and low sector utilization, respectively. Each group of bars corresponds to one combination of L4 cache capacity and core count. The Bimodal cache offers the best L4 cache read miss rate across the board, but fails to offer good performance due to high average hit and miss latency (as discussed in Section 5.3). For the workloads with high sector utilization, the L4 cache read miss rates of the baseline and the Micro-sector cache are similar, as expected. For the workloads with low sector utilization, the Micro-sector cache offers reasonable savings in the L4 cache read miss rate compared to the baseline; in several cases, the reduction in the L4 cache read miss rates from the baseline to the Micro-sector cache reaches 10%.

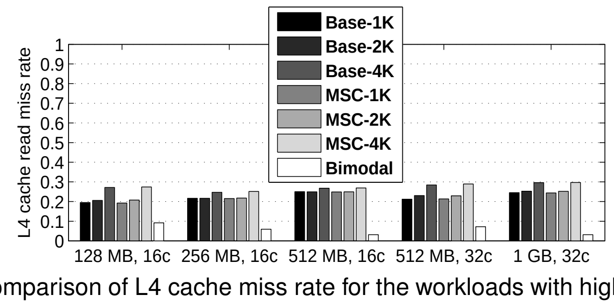

Figure 43. Comparison of L4 cache miss rate for the workloads with high sector utilization.

5.4.1. Comparison with Fine-grain Allocation

In this section, we present a quantitative comparison between the Micro-sector cache and the state-of-the-art DRAM cache designs exercising fine-grain allocation units (e.g., 64 bytes). The state-of-the-art fine-grain designs

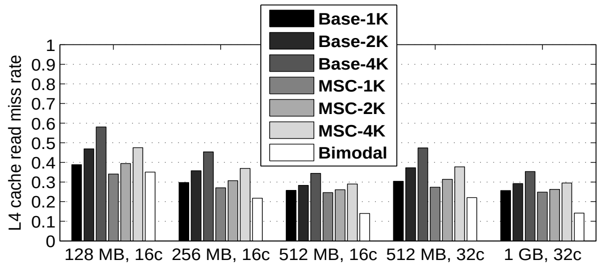

Figure 44. Comparison of L4 cache miss rate for the workloads with low sector utilization.

considered in this section are based on the direct-mapped Alloy cache [29], since this design has been shown to outperform the other existing fine-grain designs. The advantage of the fine-grain designs is that they do not suffer from any space wastage due to internal fragmentation within the allocation unit.

We consider three different designs of fine-grain DRAM caches. All these designs exercise 64-byte blocks. The first design is the Alloy cache, which organizes a DRAM cache row such that the metadata and the data block of a direct-mapped set sit together side-by-side. The Alloy cache assumes a custom-designed burst length of five to fetch the co-located metadata and data of a set through a single CAS operation. Since for every L4 cache access the metadata and the data must be fetched, the Alloy cache has an inherently higher bandwidth demand than an architecture that does not need to fetch the metadata on every access with the help of some auxiliary structures (e.g., the tag cache of the Micro-sector cache). However, the assumption of a burst length of five offers a significant advantage to the Alloy cache in terms of hit latency due to requirement of a single CAS operation to fetch the demanded metadata and the data. The Alloy cache also employs the MAP-I hit/miss predictor to accelerate L4 cache misses.

The second design implements a subsequent proposal named BEAR (bandwidth-efficient architecture) [2] that attempts to address some of the bandwidth issues of the Alloy cache. The BEAR proposal employs three optimizations. First, it does selective L4 cache bypass on misses with a probability of 0.9 to save fill write bandwidth provided the loss in miss rate is at most 1/16. Second, it incorporates a small SRAM vector to save metadata read bandwidth on L4 cache writes. Third, it integrates a small fully-associative tag cache with each L4 cache bank to save a portion of the matadata read bandwidth. This tag cache is referred to as the neighboring tag cache (NTC) signifying that an entry in this structure holds the <set id, metadata> tuple of the set that is next to the currently accessed set in an L4 cache bank. Our design incorporates a 16- and 32-entry fully-associative NTC per L4 cache bank per channel for 16- and 32-core systems, respectively so that each core can get one NTC entry, on average.

The Alloy cache and the BEAR optimizations applied to the Alloy cache assume a non-standard burst length of five. The JEDEC HBM standard supports burst lengths of only two and four [15]. Our third design adopts the Alloy cache to a system that supports a burst length of four like all our sectored cache designs. 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 large increase in the average hit latency. For this purpose, we design a tag cache that is similar in spirit to TIMBER [25]. We re-organize the Alloy cache row slightly 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. As a result, a metadata fetch brings 64-byte worth of metadata from a sequential stretch of sets. We assume that one metadata is four bytes in size; one metadata fetch brings the metadata of sixteen contiguous sets. We widen each NTC entry from one metadata to sixteen metadata so that the fetched group of metadata can be cached for future use.[15] This is particularly useful for streaming applications. On an NTC miss, this design needs to fetch the metadata of the target set through one CAS operation. If an L4 cache hit is detected from the fetched metadata, another CAS operation is issued (to the same row) to fetch the data. On an NTC hit, only one CAS operation is required to fetch the data. Additionally, we incorporate an optimization where on an NTC miss, two CAS operations (for metadata and data) are issued back-to-back one after another if the MAP-I predictor indicates an L4 cache hit. This optimization is justified by the high MAP-I prediction accuracy. This optimization helps improve the average hit latency if the NTC miss rate is high. Additionally, this design incorporates all BEAR optimizations. We will refer to this design as BEAR+BL4. We note that compared to the Alloy cache, this design may be considered commercially more viable in today's context due to its use of burst length four conforming to the JEDEC HBM standard and supported by the SK Hynix parts [39].

In all these three designs, we incorporate a spatial memory streaming (SMS) prefetcher [37], which has similar hardware requirements as the footprint prefetcher of the baseline and the Micro-sector cache designs. There are three notable differences in how the SMS prefetcher is used in the Micro-sector cache and the designs with fine-grain allocation units. First, each way of the Micro-sector cache reserves metadata space for recording the footprint of the sector(s) allocated in that way. This is not possible in the fine-grain designs. Separate SRAM tables (referred to as accumulation tables) are required for this purpose. Our design models a sixteen-entry fully-associative accumulation table per core. Each entry maintains a bitvector recording the footprint of a spatial region. To filter out the regions with a single accessed block, a sixteen-entry fully-associative filter table is used per core. These regions never enter the accumulation table. Second, the Micro-sector cache trains the prefetcher with the footprint of a sector when the sector is evicted from the L4 cache. The fine-grain designs train the prefetcher with

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

the footprint of a spatial region when either the corresponding entry is evicted from the accumulation table or the first L4 cache eviction (of any block) from the region takes place. Both designs employ a footprint table having the same organization (16K entries) to store the already learned footprints. Third, a sector miss triggers a footprint prefetch request in the Micro-sector cache. The prefetch requests need not look up the L4 cache before getting issued to the main memory system because the entire sector is absent from the L4 cache. In the fine-grain designs, the first L4 cache miss to a region triggers a region prefetch. Each prefetch request to a 64-byte block within the region needs to look up the L4 cache before getting issued to the main memory system. This is necessary because it is not clear which blocks of the region are currently not in the L4 cache. This prefetch lookup has significant bandwidth implications on the fine-grain designs; very large spatial regions are likely to hurt performance (e.g., a 4 KB spatial region in the worst case may require 64 prefetch lookups if the region has a perfect streaming behavior).

To determine the best spatial region size for the SMS prefetcher, we evaluate the performance of each of the three fine-grain designs with region sizes ranging from two blocks to 64 blocks (region sizes from 128 bytes to 4 KB). For Alloy cache and BEAR, we find that sixteen-block regions (1 KB size) yield the best performance. For BEAR+BL4, the best region size is 512 bytes (eight 64-byte blocks). The best region size of the BEAR+BL4 design is expected to be smaller than the other two designs because the BEAR+BL4 design injects far 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 lookup requests low for best performance. These two SMS prefetchers will be referred to as SMS16 and SMS8 indicating the associated spatial region sizes in terms of the number of 64-byte blocks. The prefetch requests in the fine-grain designs continue to employ the MAP-I predictor to accelerate the prefetches. Since the MAP-I predictor relies on the program counter of the access to offer a prediction, we associate each prefetch request with the program counter of the corresponding trigger demand access to the spatial region. We have verified that this is mostly accurate, particularly for the applications with good spatial streaming behavior. Finally, referring back to Table 3, we recall that the Micro-sector cache requires about 700-900 KB additional SRAM storage. To ensure a fair evaluation, we evaluate all fine-grain L4 cache designs with an L3 cache provisioned with one additional way and no change in the access latency of an L3 cache bank. One additional L3 cache way amounts to 1 MB for sixteen-core systems and 2 MB for 32-core systems.

Figures 45 and 46 compare the L4 cache demand read miss rates of the Micro-sector cache (MSC-1K, MSC-2K, MSC-4K corresponding to three different sector sizes), the Alloy cache, the Alloy cache with SMS16 prefetcher, BEAR, BEAR with SMS16 prefetcher, BEAR+BL4, and BEAR+BL4 with SMS8 prefetcher (from left to right

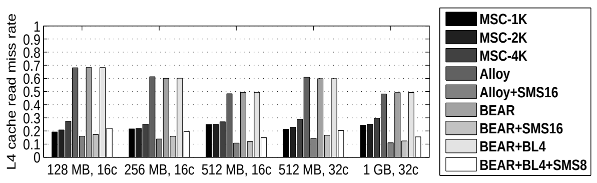

Figure 45. Comparison of L4 cache demand read miss rate for the workloads with high sector utilization.

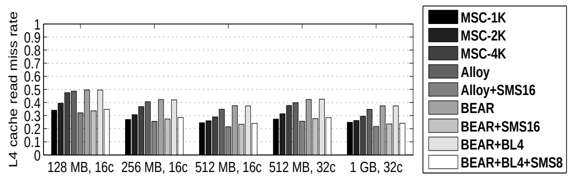

Figure 46. Comparison of L4 cache demand read miss rate for the workloads with low sector utilization.

within each group of bars). Each group of bars represents one combination of L4 cache capacity and core count. We make four important observations from these results. First, as expected, the SMS prefetcher is more effective for the workloads with high sector utilization (Figure 45) compared to those with low sector utilization (Figure 46). Nonetheless, in all three fine-grain designs, the savings in the L4 cache demand read misses achieved by the SMS prefetcher are remarkable for both groups of workloads. Second, among the three fine-grain designs with the SMS prefetcher, the Alloy cache has the best miss rate. The BEAR proposal loses slightly in miss rate due to selective fill bypass. The BEAR+BL4 design loses further in miss rate due to smaller spatial regions for the prefetcher. Third, the demand read miss rate experienced by the Micro-sector cache for the high sector utilization workload group is higher than the fine-grain designs with the SMS prefetcher. Some of the high sector utilization workloads experience very low volume of sector evictions. This hampers the training of the prefetcher because the footprint is trained only on sector eviction. The end-result is that these applications suffer from more compulsory demand misses in the Micro-sector cache compared to the fine-grain designs, which train the footprint as soon as the L4 cache experiences the first eviction in a spatial region. However, these applications are not particularly sensitive to the L4 cache organizations, as will be evident from the performance results. Table 4 shows the L4 cache demand read miss rates of the individual workloads in the high sector utilization group for the 128 MB sixteen-core configuration. The other configurations show similar trends. Only for dealII and gromacs, the fine-grain designs with prefetching enabled enjoy much better demand read miss rates compared to the Micro-sector cache. However, these two workloads have very few L4 cache accesses making them insensitive to L4 cache optimizations.. For several workloads, the Micro-sector cache delivers better demand read miss rates than the fine-grain designs due to higher associativity of the Micro-sector cache. Fourth, it is very encouraging to note that the demand read miss rate experienced by the Micro-sector cache for the low sector utilization workload group comes surprisingly close to that of the fine-grain designs employing the SMS prefetcher when the sector size is

1 KB. The largest difference in the demand read miss rate between the Micro-sector cache with 1 KB sectors and the Alloy+SMS16 is 3% observed in the 512 MB 16-core and the 1 GB 32-core systems. The demand read miss rate of the Micro-sector cache with 1 KB sectors is within 1% of that of BEAR+SMS8. In several cases, the Micro-sector cache with 1 KB sectors offers better demand read miss rate than the BEAR+BL4+SMS8 design. Table 5 shows the L4 cache demand read miss rates of the individual workloads in the low sector utilization group for the 128 MB sixteen-core configuration.

Table 4. L4 cache demand read miss rates of the high sector utilization workloads for the 128 MB sixteen-core config.

Workloads	MSC-1K	MSC-2K	MSC-4K	Alloy+SMS16	BEAR+SMS16	BEAR+BL4 +SMS8
bwaves	0.10	0.07	0.08	0.09	0.10	0.17
bzip2.combined	0.08	0.08	0.08	0.02	0.03	0.03
cactusADM	0.22	0.30	0.45	0.33	0.34	0.34
dealII	0.30	0.35	0.41	0.10	0.10	0.15
gobmk.score2	0.33	0.36	0.46	0.38	0.33	0.45
gromacs	0.50	0.51	0.51	0.13	0.16	0.17
hmmer.nph3	0.04	0.04	0.04	0.01	0.01	0.01
lbm	0.07	0.08	0.10	0.10	0.10	0.15
leslie3d	0.16	0.21	0.37	0.16	0.23	0.27
libquantum	0.13	0.13	0.23	0.19	0.20	0.33
soplex.ref	0.19	0.20	0.29	0.19	0.21	0.26
wrf	0.20	0.21	0.31	0.12	0.19	0.22
zeusmp	0.18	0.17	0.23	0.27	0.28	0.29
Average	0.19	0.21	0.27	0.16	0.17	0.22

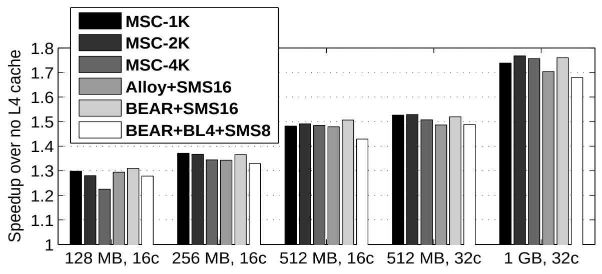

Figure 47. Speedup relative to no L4 cache for the workloads with high sector utilization.

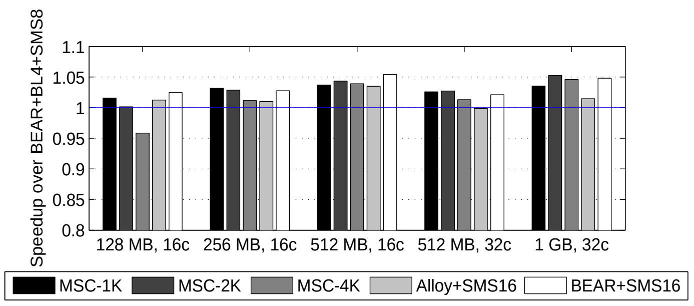

Figure 48. Speedup relative to BEAR+BL4+SMS8 for the workloads with high sector utilization.

Table 5. L4 cache demand read miss rates of the low sector utilization workloads for the 128 MB sixteen-core config.

Workloads	MSC-1K	MSC-2K	MSC-4K	Alloy+SMS16	BEAR+SMS16	BEAR+BL4 +SMS8
astar.rivers	0.05	0.06	0.13	0.07	0.08	0.08
gamess.triazolium	0.56	0.55	0.56	0.41	0.43	0.38
gcc.cp-decl	0.32	0.41	0.54	0.30	0.34	0.34
gcc.s04	0.45	0.56	0.66	0.36	0.38	0.37
gobmk.nngs	0.42	0.44	0.49	0.36	0.39	0.38
gobmk.trevorc	0.31	0.36	0.48	0.38	0.34	0.44
h264ref.foreman_main	0.13	0.13	0.13	0.08	0.12	0.11
mcf	0.25	0.34	0.44	0.34	0.35	0.35
milc	0.32	0.34	0.42	0.42	0.41	0.52
omnetpp	0.46	0.58	0.71	0.39	0.40	0.41
perlbench.checkspam	0.40	0.46	0.56	0.37	0.44	0.42
sjeng	0.80	0.82	0.84	0.73	0.74	0.75
soplex.pds-50	0.25	0.30	0.41	0.26	0.27	0.27
sphinx3	0.02	0.03	0.05	0.08	0.10	0.13
xalancbmk	0.37	0.55	0.72	0.28	0.29	0.29
Average	0.34	0.39	0.48	0.32	0.34	0.35

Figure 47 presents the performance of the Micro-sector cache (MSC-1K, MSC-2K, MSC-4K corresponding to three different sector sizes), the Alloy cache with the SMS16 prefetcher, BEAR with the SMS16 prefetcher, and BEAR+BL4 with the SMS8 prefetcher for the workload group with high sector utilization. All results are normalized to a similar system without the L4 cache. To further zoom in on the performance differences between the Micro-sector cache and the fine-grain designs, Figure 48 shows the performance speedup relative to BEAR+BL4+SMS8. Across the board, the Micro-sector cache either matches the performance of or outperforms the fine-grain designs. This is expected for this group of workloads because the Micro-sector cache does not suffer from space wastage due to problems related to sector utilization. At 1 GB capacity, the best Micro-sector cache design (2 KB 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 1 GB. 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 to transfer a TAD and two of these cycles transfer data. As a result, one-third of the bandwidth is spent in metadata reads/writes. As the L4 cache capacity is scaled up and the volume of hits increases, 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. 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 1 GB. 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 4 KB 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 2 KB and 4 KB sectors, the BEAR+BL4 design enjoys less number of tag cache hits per metadata CAS. The BEAR+BL4 design fetches sixteen spatially contiguous tags through one metadata CAS. As a result, while streaming through a 1920-byte or 4032-byte spatial region, the BEAR+BL4 design needs two or four metadata CAS commands. On the other hand, the Micro-sector cache with 1920-byte or 4032-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 compared to the Micro-sector cache. We note that metadata prefetching can improve the tag cache miss latency in BEAR+BL4, but will not improve the metadata bandwidth bloat in the L4 cache.

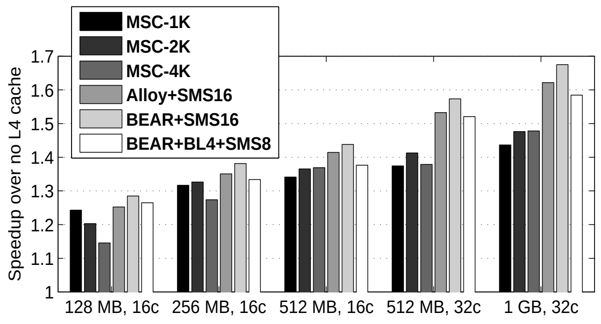

Figure 49. Speedup relative to no L4 cache for the workloads with low sector utilization.

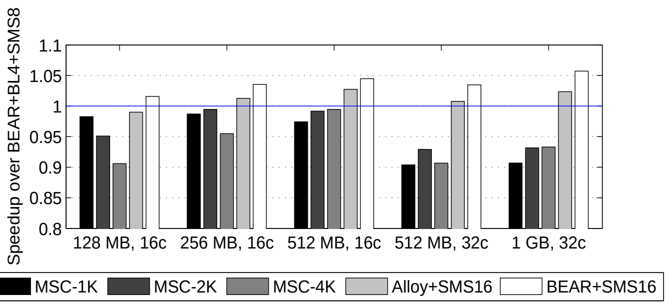

Figure 50. Speedup relative to BEAR+BL4+SMS8 for the workloads with low sector utilization.

Figure 49 quantifies the performance of the Micro-sector cache (MSC-1K, MSC-2K, MSC-4K corresponding to three different sector sizes), the Alloy cache with the SMS16 prefetcher, BEAR with the SMS16 prefetcher, and BEAR+BL4 with the SMS8 prefetcher for the workload group with low sector utilization. All results are normalized to a similar system without the L4 cache. Figure 50 further summarizes the performance speedup relative to BEAR+BL4+SMS8. The following two observations bring out the salient points of these results. First, we notice that except for 128 MB and 256 MB capacities, the Micro-sector cache does not lose performance as the sector size is increased. In fact, in some cases, the performance improves as the sector size increases. This is surprising, given that the L4 cache read miss rate increases steadily with increasing sector size. The primary reason for this counter-intuitive performance trend is that with the increasing sector size, the tag cache and the NRU state cache miss rates decrease gradually (Figure 7) leading to improved average hit latency. Since the influence of hit latency on the end-performance increases with increasing L4 cache capacity (due to increasing hit rate), this effect is more prominent at 1 GB capacity where the 2 KB and 4 KB sector sizes outperform the 1 KB sector size by a reasonable margin. Second, we observe that the fine-grain designs outperform the Micro-sector cache by varying margins across the board. The margin is largest at 512 MB and 1 GB capacities with 32 cores. For both 512 MB and 1 GB capacity with 32 cores, the best Micro-sector cache design (2 KB sector size for 512 MB and 2 KB or 4 KB sector size for 1 GB) performs 7% worse than BEAR+BL4+SMS8. For 256 MB and 512 MB capacities with sixteen cores, the best Micro-sector cache design (2 KB sector size for 256 MB and 2 KB or 4 KB sector size for 512 MB) matches the performance of BEAR+BL4+SMS8 and performs within 5% of BEAR+SMS16. For 128 MB capacity, the best Micro-sector cache design (1 KB sector size) performs within 2% of BEAR+BL4+SMS8 and within 3% of BEAR+SMS16.

The performance difference between the Micro-sector cache and the fine-grain designs arises due to a combination of various factors. For higher capacities of the L4 cache, the NRU state cache misses consume a significant amount of Micro-sector cache bandwidth, as already discussed in Section 3.1. On top of that, the BEAR+BL4+SMS8 design enjoys an advantage in the L4 cache demand read miss rate (Figure 46). The Alloy cache and the BEAR designs exercising a custom non-standard burst length of five enjoy the additional advantage of better average hit latency because these designs can fetch the data block and the metadata through a single CAS operation to satisfy an L4 cache hit. The Micro-sector cache, on average, requires more than one CAS operation to satisfy an L4 cache hit because a tag cache miss needs to issue two CAS operations due to adherence to the standard burst length of four. The tag cache miss rate can be reasonably high for the workloads with low sector utilization because the utility of one tag cache entry is relatively low. Table 6 shows the average tag cache miss rates for the high and low sector utilization workload groups in sixteen-core (High-16c and Low-16c) and 32-core (High-32c and Low-32c) systems equipped with the Micro-sector cache as the sector size is varied (recall that the tag cache miss rate does not depend on the L4 cache capacity). The workloads with low sector utilization have reasonably high tag cache miss rates. Further, these workloads do not have a high bandwidth demand (unlike the workloads with high sector utilization) and as a result, the metadata bandwidth penalty of the Alloy cache does not have a negative influence on the end-performance. Overall, the Micro-sector cache, while offering a scalable path to large-capacity DRAM caches, delivers performance that is better than or similar to the fine-grain designs for the high sector utilization workload group and experiences performance losses within acceptable margins compared to the fine-grain designs for the low sector utilization workload group.

Table 6. Tag cache miss rate.

Sector size	High-16c	Low-16c	High-32c	Low-32c
1 KB	0.09	0.31	0.11	0.36
2 KB	0.04	0.23	0.05	0.28
4 KB	0.03	0.21	0.04	0.27

5.4.1.1. Sensitivity to the NRU State Cache Capacity. We have already pointed out that the tag cache and the NRU state cache play an important role in the overall performance of our proposal. Referring back to Table 3, we see that scaling the tag cache of the Micro-sector cache any further can introduce a significant SRAM storage overhead. On the other hand, the NRU state cache can be scaled up in size with small additional storage. For example, a 128K-entry direct-mapped NRU state cache requires 192 KB, 176 KB, and 160 KB of storage when operating with a 1 GB Micro-sector cache having sector sizes 1 KB, 2 KB, and 4 KB, respectively. As a result, such an NRU state cache can be comfortably accommodated within a total SRAM storage overhead of 1 MB (see Table 3). We note that 1 MB SRAM storage is equivalent to one L3 cache way in a sixteen-core system and half of one L3 cache way in a 32-core system (recall that all fine-grain designs are provisioned with one additional L3 cache way). In the following, we explore how the end-performance of the Micro-sector cache gets influenced when the NRU state cache is scaled up and show that most of the performance gap between the Micro-sector cache and the fine-grain designs can be bridged when the Micro-sector cache is equipped with a scaled up direct-mapped NRU state cache.

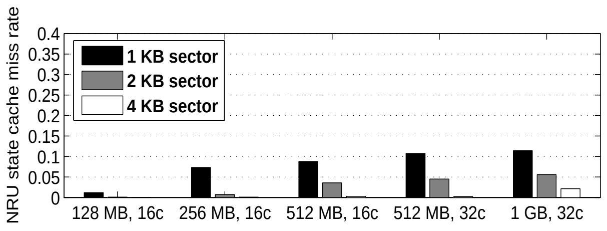

Figure 51. Miss rate of a 32K-entry direct-mapped NRU state cache for the workloads with high sector utilization.

Figures 51 and 52 show the miss rates of a 32K-entry direct-mapped NRU state cache (default size used so far as noted in Table 3) for the workload groups with high and low sector utilization, respectively as the Micro-sector cache capacity and the core count are varied. As expected, the NRU state cache miss rate for the workloads with high sector utilization is within acceptable limits. The maximum miss rate is slightly over 10% (seen at 512 MB

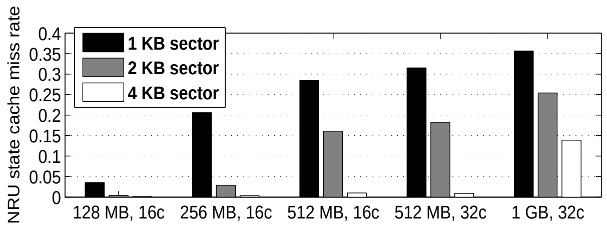

Figure 52. Miss rate of a 32K-entry direct-mapped NRU state cache for the workloads with low sector utilization.

and 1 GB capacities with 1 KB sector size in a 32-core system). On the other hand, for the workloads with low sector utilization, the NRU state cache miss rate increases rapidly with increasing Micro-sector cache capacity for a fixed sector size. The maximum miss rate is 36% experienced at 1 GB capacity with 1 KB sector size in a 32-core system. Since each NRU state cache entry has lower utility when the sector utilization is low compared to when the sector utilization is high, the workloads with low sector utilization struggle to keep their NRU state working set in a 32K-entry NRU state cache.

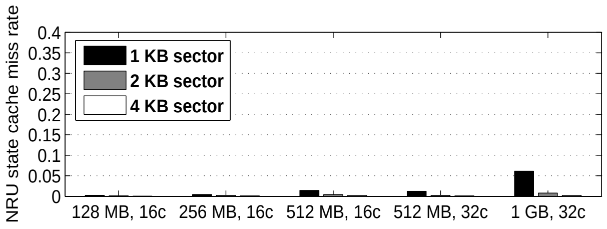

Figure 53. Miss rate of a 128K-entry direct-mapped NRU state cache for the workloads with high sector utilization.

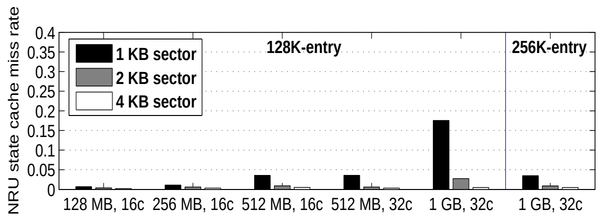

Figure 54. Miss rate of a 128K-entry direct-mapped NRU state cache for the workloads with low sector utilization. The rightmost group of bars shows the miss rate of a 256K-entry direct-mapped NRU state cache associated with a 1 GB Micro-sector cache in a 32-core system.

Figures 53 and 54 show the miss rates of a 128K-entry direct-mapped NRU state cache for the workload groups with high and low sector utilization, respectively as the Micro-sector cache capacity and the core count are varied. For the workload group with low sector utilization, we also show the miss rate of a 256K-entry direct-mapped NRU state cache when associated with a 1 GB Micro-sector cache in a 32-core system (the rightmost group of bars in Figure 54). With a 128K-entry NRU state cache, the average miss rate is well below 10% in all cases except 1 GB Micro-sector cache with 1 KB sector size for the low sector utilization workload group. However, at 1 GB capacity with 1 KB sector size, a 256K-entry Micro-sector cache brings down the miss rate to about 4%. We note that a 256K-entry direct-mapped NRU state cache requires 352 KB of storage when operating with a 1 GB Micro-sector cache having 1 KB sector size. Therefore, inclusion of such an NRU state cache in a 1 GB Micro-sector cache with 1 KB sector size requires at most 1006.5 KB of total SRAM storage overhead (see Table 3), which is still under 1 MB.

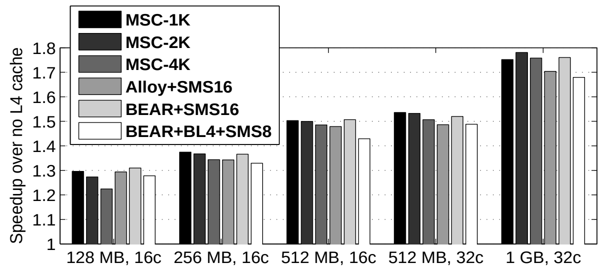

Figure 55. Speedup relative to no L4 cache for the workloads with high sector utilization. The Micro-sector cache is equipped with a 128K-entry direct-mapped NRU state cache.

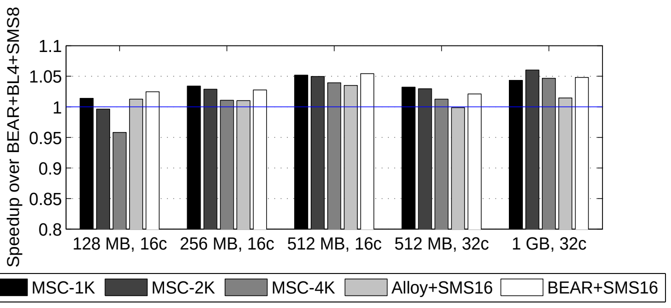

Figure 56. Speedup relative to BEAR+BL4+SMS8 for the workloads with high sector utilization. The Micro-sector cache is equipped with a 128K-entry direct-mapped NRU state cache.

Figure 55 quantifies the performance of the Micro-sector cache (MSC-1K, MSC-2K, MSC-4K corresponding to three different sector sizes), the Alloy cache with the SMS16 prefetcher, BEAR with the SMS16 prefetcher, and BEAR+BL4 with the SMS8 prefetcher for the workload group with high sector utilization when the Micro-sector cache is provisioned with a 128K-entry direct-mapped NRU state cache. All results are normalized to a similar system without the L4 cache. Figure 56 further summarizes the performance speedup relative to BEAR+BL4+SMS8. Only at 128 MB capacity, the best Micro-sector cache design (1 KB sector size) delivers performance that is 1% worse than the best fine-grain design, namely, BEAR+SMS16. At each of the remaining four <capacity, core count> configurations, there is at least one Micro-sector cache sector size that either matches the performance of or outperforms BEAR+SMS16.

Figure 57 quantifies the performance of the Micro-sector cache, the Alloy cache with the SMS16 prefetcher, BEAR with the SMS16 prefetcher, and BEAR+BL4 with the SMS8 prefetcher for the workload group with low sector utilization when the Micro-sector cache is provisioned with a 128K-entry direct-mapped NRU state cache.

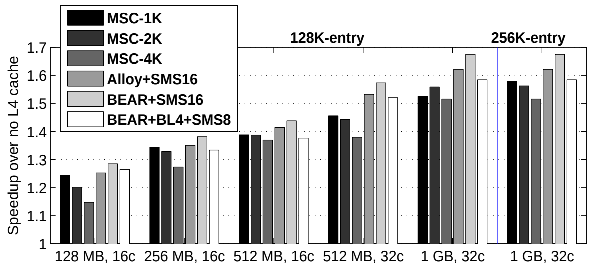

Figure 57. Speedup relative to no L4 cache for the workloads with low sector utilization. The Micro-sector cache is equipped with a 128K-entry direct-mapped NRU state cache in all cases except the rightmost group of bars, where the NRU state cache has 256K entries.

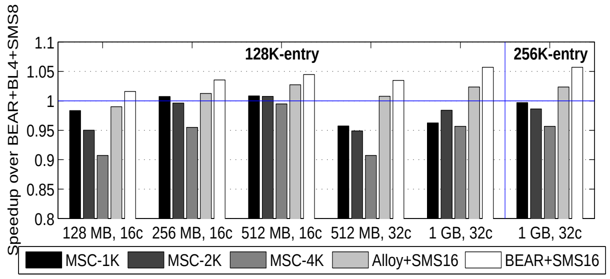

Figure 58. Speedup relative to BEAR+BL4+SMS8 for the workloads with low sector utilization. The Micro-sector cache is equipped with a 128K-entry direct-mapped NRU state cache in all cases except the rightmost group of bars, where the NRU state cache has 256K entries.

We also show the performance of a 1 GB Micro-sector cache provisioned with a 256K-entry direct-mapped NRU state cache (the rightmost group of bars). All results are normalized to a similar system without the L4 cache. Figure 58 further summarizes the performance speedup relative to BEAR+BL4+SMS8. With a 128K-entry NRU state cache, the best Micro-sector cache design outperforms BEAR+BL4+SMS8 at 256 MB and 512 MB capacities in a sixteen-core system. At 128 MB capacity in a sixteen-core system, the best Micro-sector cache design performs within 2% of BEAR+BL4+SMS8. At 512 MB and 1 GB capacities in a 32-core system, the best Micro-sector cache design performs respectively within 4% and 2% of BEAR+BL4+SMS8. However, with a 256K-entry NRU state cache, the best 1 GB Micro-sector cache (1 KB sector size) delivers the same level of performance as BEAR+BL4+SMS8. Overall, scaling the NRU state cache up while staying within an SRAM storage overhead of 1 MB allows the Micro-sector cache to perform reasonably close to the fine-grain designs.

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

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 [10] (distributed with McPAT [11]) to evaluate the dynamic and leakage energy in the SRAM parts for 22 nm 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 [26] to the extent possible[16] and includes background, activation/precharge, CAS, and refresh power.

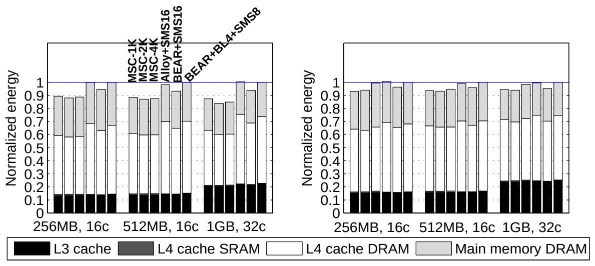

Figure 59. Energy normalized to BEAR+BL4+SMS8 for the workloads with high (left panel) and low (right panel) sector utilization. The 1 GB configuration has 32 MB L3 cache, while others have 16 MB L3 cache.

Figure 59 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 1 KB and 2 KB 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 2 KB sectors expends 5-7% less energy than BEAR+SMS16. At 1 GB capacity, the Micro-sector cache with 2 KB sectors is 6% better in terms of energy expense and 3% better in terms of energy-delay product compared to BEAR+SMS16.

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

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 128 MB and 1 GB. We also present a detailed performance comparison of the Micro-sector cache with the Bimodal cache and a range of designs exercising fine-grain allocation units. Our results show that the Micro-sector cache outperforms the Bimodal cache by large margins and performs within acceptable margins of the fine-grain designs while expending less energy. Overall, the Micro-sector cache proposal offers a scalable DRAM cache design with good performance and energy characteristics and acceptable metadata overhead as the DRAM cache grows in capacity.

References

[1] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen. IBM POWER7 Systems. In *IBM Journal of Research and Development*, **55**(3): 2:1–2:13, May/June 2011.

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

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

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

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

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

[7] F. Hameed, L. Bauer, and J. Henkel. 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*, pages 1–10, September/October 2013.

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

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

[10] HP Labs. 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/.

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

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

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

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

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

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

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

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

[19] 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. Haswell: A Family of IA 22 nm Processors. In *International Solid-State Circuits Conference*, pages 112–113, February 2014.

[20] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt. Prefetch-aware DRAM Controllers. In *Proceedings of the 41st Annual International Symposium on Microarchitecture*, pages 200–209, November 2008.

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

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

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

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

[25] J. Meza, J. Chang, H-B. Yoon, O. Mutlu, and P. Ranganathan. Enabling Efficient and Scalable Hybrid Memories using Fine-Granularity DRAM Cache Management. In *IEEE Computer Architecture Letters*, pages 61–64, July-December 2012.

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

[27] C. R. Moore. The PowerPC 601 Microprocessor. In *Proceedings of the IEEE COMPCON*, pages 109–116, February 1993.

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

[29] M. K. Qureshi and G. H. Loh. 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*, pages 235–246, December 2012.

[30] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. In *IEEE Computer Architecture Letters*, **10**(1): 16–19, January-June 2011.

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

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

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

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

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

[36] J. Sim et al. Resilient Die-stacked DRAM Caches. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 416–427, June 2013.

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

[38] J. Stuecheli. Next Generation POWER microprocessor. In *Hot Chips*, August 2013.

[39] K. Tran and J. Ahn. HBM: Memory Solution for High Performance Processors. In *MemCon*, October 2014.

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

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

[42] H. Wang, T. Sun, and Q. Yang. 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*, pages 381–390, June 1995.

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

[44] Z. Zhang, Z. Zhu, and X. Zhang. Design and Optimization of Large Size and Low Overhead Off-Chip Caches. In *IEEE Transactions on Computers*, **53**(7): 843–855, July 2004.

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