A Survey Of Techniques for Architecting DRAM Caches

Article *in* IEEE Transactions on Parallel and Distributed Systems · January 2015

DOI: 10.1109/TPDS.2015.2461155

Sparsh Mittal
Indian Institute of Technology Roorkee
133 PUBLICATIONS **2,050** CITATIONS
SEE PROFILE

Jeffrey Vetter
University of Tennessee
258 PUBLICATIONS **7,010** CITATIONS
SEE PROFILE

A Survey Of Techniques for Architecting DRAM Caches

Sparsh Mittal, *Member, IEEE,* and Jeffrey S. Vetter, *Senior Member, IEEE*

Abstract—Recent trends of increasing core-count and memory/bandwidth-wall have led to major overhauls in chip architecture. In face of increasing cache capacity demands, researchers have now explored DRAM, which was conventionally considered synonymous to main memory, for designing large last level caches. Efficient integration of DRAM caches in mainstream computing systems, however, also presents several challenges and several recent techniques have been proposed to address them. In this paper, we present a survey of techniques for architecting DRAM caches. Also, by classifying these techniques across several dimensions, we underscore their similarities and differences. We believe that this paper will be very helpful to researchers for gaining insights into the potential, tradeoffs and challenges of DRAM caches.

Index Terms—Review, classification, last level cache, die-stacking, 3D, stacked DRAM, bandwidth wall, extreme-scale system, architectural techniques.

1 Introduction

As the quest of ongoing process technology scaling meets the formidable challenges of power- and bandwidth-wall, architecture of modern processors is already witnessing dramatic shifts. As the cache capacity demands of modern applications escalate due to increasing number of on-chip cores, the conventional wisdom of using SRAM for designing caches has been subjected to critical scrutiny. The low density coupled with high leakage power consumption of SRAM make use of large SRAM caches infeasible and hence, the size of largest SRAM cache in modern processors ranges in merely few tens of megabytes (e.g. 37.5MB SRAM LLC in Ivytown processor [1]). By comparison, modern scale-out workloads (e.g. business analytics, web search etc.) demand hundreds of megabytes of last level cache [2, 3]. Large caches are also required for avoiding bandwidth wall and if such caches are designed using SRAM, they may occupy 90% of the chip-area in future process technology generations [4]. This severely limits the number of cores that can be placed on-chip and thus restricts multicore scaling [5]. Further, given the high leakage power of SRAM, large SRAM caches may necessitate expensive cooling solutions (e.g. liquid cooling), which increase the cost and complexity of chip significantly. These factors and trends have motivated the researchers to explore the alternatives of SRAM for designing caches.

Although DRAM provides nearly 8× [6, 7] density advantage compared to SRAM, its relatively long latency had conventionally restricted its use to designing main memories only. However, recent advancements in die-stacking have led to significant improvement in these properties [6]. By virtue of improving integration density and eliminating wires, stacked DRAM can provide between 8× [8] to 15× [9] more bandwidth than off-chip DRAM while incurring only half [10, 11] or one-third [12] the latency.

These features have motivated the development and release of high performance stacked DRAM from several leading vendors [13–17] and as such, stacked DRAM is poised to be integrated in the memory hierarchy of both CPUs and GPUs [9, 18, 19]. In fact, Intel's 14nm Knights Landing processor is said to use a 16GB stacked DRAM last level cache with 500GB/s bandwidth [9]. Although non-volatile memories (NVMs) such as STTRAM (spin transfer torque RAM) and ReRAM (resistive RAM) have also been considered for designing last level caches (LLCs) [20], they have yet to reach the level of fabrication maturity and commercial viability that may justify their adoption in mainstream computing systems. Clearly, at least for the foreseeable future, DRAM will be a promising technology for mitigating power/bandwidth barriers by enabling design of large caches.

Fully leveraging the potential of DRAM caches, however, requires addressing several challenges and making careful choice of design parameters such as block size, tag store architecture, 3D partitioning strategy, thermal management issues etc. In fact, due to the marked difference between SRAM and DRAM, blindly porting even well-known SRAM cache optimizations (e.g. minimizing miss-rate, using high associativity and better replacement policies) to DRAM

- *S. Mittal is with the Future Technologies Group, Oak Ridge National Laboratory, Oak Ridge, TN 37830. E-mail: mittals@ornl.gov.*
- *J. S. Vetter is with the Future Technologies Group, Oak Ridge National Laboratory, Oak Ridge, TN 37830, and with the Georgia Institute of Technology, Atlanta, GA 30332. E-mail: vetter@ornl.gov.*

Support for this work was provided by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research.

cache may degrade its performance [8]. Clearly, novel techniques are required to fully realize the potential of DRAM caches.

In this paper, we survey techniques proposed for architecting DRAM caches. We classify these techniques based on key parameters to highlight their similarities and differences. We focus on architecture and system-level techniques and not on device-level techniques. Since the evaluation approach and workloads used in different works vary, we only focus on their key research insights and typically do not discuss their quantitative results. We hope that by presenting a unified view of frontiers of DRAM cache research, this paper will provide clear directions for future research and development in this area. In this paper, DRAM cache generally refers to stacked DRAM cache, since the higher latency and lower bandwidth of off-chip (commodity) DRAM make them unsuitable for cache design.

The rest of the paper is organized as follows. Section 2 provides a brief background on DRAM caches and tradeoffs involved in their design. Section 3 first provides an overview and classification of DRAM cache management techniques. It then reviews several techniques for managing DRAM caches. Finally, Section 4 concludes this paper with a discussion of future challenges.

2 Background

We now discuss the motivation behind use of DRAM cache, along with the challenges associated in their use. We refer the reader to prior work for more details on DRAM architecture [21–24] and die-stacking technology [2, 6, 25].

2.1 Promises of DRAM Caches

While DRAM has been traditionally considered synonymous to main memory, several recent trends present compelling reasons to consider its use for designing caches.

Increasing memory pressure and high density of DRAM: With increasing number of cores, the pressure on memory hierarchy is increasing. The poor pin-count scalability leads to memory bandwidth wall, which needs to be offset using effective approaches such as large-sized caches. As mentioned before, low density and high leakage energy of SRAM present a major challenge in its use for designing large caches. Although architectural techniques can partially offset the energy challenges of SRAM [26], they cannot bridge the magnitude order gap compared to the energy efficiency and capacity requirements placed on extreme-scale memory systems.

Looking at device-level characteristic, we find that SRAM has 6T (6 transistor) structure with a cell size of 120-200F^2 (F shows the feature size), whereas, DRAM has 1T1C (one transistor and one capacitor) structure with a cell size of 6-10F^2 [20, 23]. Clearly, DRAM provides much higher density than SRAM. Hence, DRAM technology can allow integration of giga-byte size caches and thus, presents as a promising memory technology to overcome memory and bandwidth walls. The density advantage provided by DRAM can be leveraged to place more cores on the chip which paves the way for multicore scaling to continue [4, 5]. These factors have motivated the researchers to use DRAM even as an L4 cache [27–31] and as a cache for NVM main memory [32].

Benefits of die-stacking: As mentioned before, die-stacking provides significant latency and bandwidth benefits compared to conventional off-chip memory [6, 25, 33]. In terms of energy per bit, the TSV (through silicon via)-based interface provides an order of magnitude improvement over low-power DRAM interfaces (LPDDR2) and two orders of magnitude improvement over DDR3 interface [2]. By sharing the peripheral circuitry over the stack, die-stacked DRAM can also improve cost efficiency. Further, while stacking multiple processor chips may create thermal issues, stacking memory on top of processor cores results in only a small temperature increase (less than 10° for 8 additional layers of memory [25]).

The bandwidth provided by off-chip main memory is limited due to pin-count limitations. Die-stacking circumvents the pin limitations and thus provides low latency interconnects and very high bandwidth. At the same time, the capacity provided by stacked DRAM is significantly smaller than the off-chip DRAM [3, 10, 34], for example, in Intel's Knights Landing system, the capacity of stacked DRAM and off-chip DRAM are 16GB and 384GB, respectively [9]. Also, the bandwidth to stacked DRAM is restricted by the parallelism in the stacked DRAM itself and not by the interface. These facts make the stacked DRAM a better fit as a cache rather than as main memory.

Maintaining software transparency: In comparison to using DRAM as a main memory, the benefit of using DRAM as a cache is that the software (both application and OS) need not be modified to reap the benefits of this cache. Thus, legacy software can be benefited and dependence on OS vendors is not required.

2.2 Factors and tradeoffs in designing DRAM caches

Despite their promises, DRAM caches also present several challenges that need to be addressed for ensuring their efficient integration in product systems.

Performance challenges: For applications with poor locality, addition of DRAM cache may not reduce the miss-rate sufficiently. In fact, since their latency is only 50% lower than that of main memory, DRAM caches may significantly increase the latency of misses. There is also a tradeoff between capacity (or miss-rate) and latency and hence, some techniques aim to optimize latency [8], while others optimize miss-rate [29] at the expense of other parameter. DRAM also requires refresh which consume significant amount of energy and reduces device availability [24]. Due to these tradeoffs, increasing the size of DRAM beyond a limit may not be feasible.

Architecture and metadata management: Due to their large capacity, DRAM caches also require a large amount of metadata, such as tags, dirty, valid bits etc. (we henceforth use the words 'tag' and 'metadata' synonymously) and tag lookups lie on the critical path of all requests. This makes the choice of cache line granularity and storage location of tags (e.g. in SRAM or DRAM) key design issues for DRAM caches, which are also interrelated. At two extremes are fine-grained (also called block-based, having 64B granularity) and coarse-grained (also called page-based with granularity of few KBs) designs which lead to vastly different tradeoffs. Assuming 6B tag overhead per cache line, a 1GB DRAM cache will require 96MB and 3MB of tag storage for fine-grained (64B line) and coarse-grained (2KB line) designs, respectively.

It is clear that the fine-grained organizations incur prohibitively high tag storage overhead, which may be much greater than the size of last level SRAM cache itself. This forces the designers to store tags in DRAM itself, which leads to increased hit latency. Serial tag-data access and use of set-associative caches further increase this latency and hence, parallel tag-data access and use of direct-mapped caches have been proposed (see Table 1), although these design choices have their own limitations [26]. Access to DRAM for updating replacement information reduces DRAM availability and hence, some works use random replacement policy instead of LRU (least recently used) policy since the former does not require update [8]. Several techniques use intelligent predictors to bypass access to DRAM cache for requests which are predicted to miss and their efficacy depends on the accuracy of prediction and proper management of dirty data [35]. Fine-grained organizations are also less effective in exploiting spatial locality and hence, incur higher miss rates, which incurs the penalty of off-chip access. The advantage of fine-grained organization is that they use off-chip bandwidth and cache capacity efficiently.

Coarse-grained organizations incur much smaller tag overhead which allows the tag to be stored in SRAM, thus leading to fast lookups. They rely on exploiting spatial locality and hence, they may see higher hit ratio compared to fine-grained caches, although their miss-rate reduction may not be sufficient to reduce the bandwidth wastage. The limitation of coarse-grained approach is that it leads to bandwidth wastage and queue contention leading to back-pressure delays in the cache hierarchy (writing back only dirty subblocks can partially reduce the memory traffic [22]). Large cache lines lead to under-utilization of cache capacity and also lead to false-sharing in multi-threaded applications [29]. Also, the cacheline granularity of coarse-grained designs is limited by DRAM page size, since use of larger granularity will necessitate mapping the data across more than one physical rows which leads to timing and energy inefficiencies. Further, with rising DRAM cache capacity, even coarse-grained organizations incur increasing amount of tag overhead which makes storing their tag in SRAM infeasible.

Fabrication and reliability challenges: Due to unique design of DRAM, the cache controller in DRAM caches need to also perform device-level tasks such as managing refresh, issuing row activation and precharge commands, fulfilling DRAM device timing constraints, etc. These may lead to significant complexity and may require redesign of the cache control logic [3]. Further, compared to conventional 2D off-chip memories, die-stacked memories demand much higher reliability, since a fault in die-stacked memory may not be easily serviceable and hence, may require discarding the entire package including the functioning processor layer [36]. Also, designing memory stacks with very large number of layers presents challenges of power delivery and cooling, yield, testability etc. [25].

Challenges in hybrid caches: Some researchers have proposed SRAM/DRAM hybrid caches (refer Table 1). Since the latency of SRAM is less than 10ns [33, 37], while that of DRAM is 60-80ns [34, 37], the large difference between the latency of SRAM and DRAM may make the hit latency different for different accesses which makes the scheduling of dependent instructions difficult. For this reason, the data migration schemes as used in SRAM/STTRAM or SRAM/ReRAM hybrid caches [38] may not be feasible in SRAM/DRAM hybrid caches.

3 DRAM Cache Management Techniques

Table 1 provides an overview and classification of DRAM cache management techniques discussed in this survey. This table classifies the research techniques based on their optimization objective, their key feature and the cache architecture used by them. This classification is expected to be useful for CAD-designers, researchers and technical marketing-professionals.

We now briefly discuss several DRAM cache management techniques. For sake of convenience, we roughly organize them in several categories.

3.1 Comparative evaluation of die-stacked DRAM

In this subsection, we discuss research works that explore the architecture of DRAM caches and/or compare them with SRAM caches.

TABLE 1
A classification of research works

Classification	References
Study/optimization objective	
Performance	[3, 4, 6–8, 10–12, 21, 27–32, 35, 39–58]
Energy Saving	[2, 6, 7, 32, 43, 47, 49, 53, 54, 56, 57, 59–61]
Providing inter-core isolation	[42, 44, 45]
Area saving for integrating more cores	[2, 4, 5, 62]
Refresh overhead management	[59, 61]
Dynamically optimizing bandwidth usage	[8, 12, 21, 32, 34, 35, 53, 58]
Thermal management	[2, 6, 60]
Reliability/resilience	[36, 60]
Mitigating process variation	[50]
Comparison with SRAM caches	[4, 6, 7, 32, 42, 47, 56, 57, 59, 62]
Comparison with NVM caches	[56, 57]
Cache architecture	
SRAM+DRAM hybrid cache	[43, 44, 49]
Direct-mapped cache	[3, 8, 32, 36, 48, 58]
Parallel tag-data access	[8, 21, 36, 51, 58]
Metadata on DRAM	[3, 8, 10, 11, 21, 29, 31, 32, 36, 39, 41, 44, 46, 48, 50–52, 54, 55, 58]
Metadata on SRAM	[6, 12, 39, 43, 49, 53, 61]
Fine-grain	[3, 8, 10, 29, 31, 36, 39, 41, 44, 48, 49, 52, 54, 58]
Coarse (large than block)-grain	[6, 12, 28, 50, 51, 53, 55]
Adaptive granularity	[21, 32]
Key feature of technique	
Steering accesses to DRAM cache or main memory	[22, 35]
Stacked DRAM as cache or OS-managed main memory	[3, 10, 30, 48]
Cache reconfiguration	[43, 49, 54]
Cache mapping scheme	[44–46]
Use of additional structures/approaches	
Hit/miss or location predictor or tag cache	[8, 10, 21, 21, 27, 31, 32, 35, 39, 51, 52, 58]
Predictor for selecting useful/hot blocks	[12, 51, 53]
Prefetching	[3, 39, 48, 52]
Dynamic voltage/frequency scaling	[5, 6, 60]

Black et al. [6] evaluate different SRAM and DRAM 2D/3D cache designs from performance and thermal perspectives. Their baseline is a dual-core processor with 4MB SRAM cache. For the second design, they stack an 8MB SRAM cache directly above the processor die, assuming that the area of 8MB stacked L2 is nearly the same as that of the baseline die. Assuming the DRAM density to be 8× that of SRAM density, in the third design, 4MB SRAM L2 is replaced with a 32MB stacked DRAM L2 and the area saved on processor die is used for storing tags for DRAM cache. In the fourth design, 64MB DRAM cache is stacked on top and its tags are stored in 4MB L2 on processor die. They show that increased cache capacity enabled by die-stacked DRAM cache helps in improving the performance and energy efficiency and reducing the off-chip bandwidth. They also note that compared to baseline, none of the stacked cache designs impact the thermal profile significantly, although stacking SRAM results in higher thermal increase due to the higher power density of SRAM compared to DRAM.

Sun et al. [41] present a design for multicore systems where 3D DRAM is used to implement a private L2 cache for each core and main memory shared by all the cores. To architect 3D DRAM without requiring stringent constraints on TSV fabrication, they propose a coarse-grained 3D partitioning strategy. Instead of splitting individual memory sub-array across several layers, they propose retaining individual memory sub-array is a single layer. They show that as the number of DRAM dies and the L2 cache capacity increase, the 3D DRAM L2 cache may have comparable or even smaller latency than the 2D SRAM L2 cache. To reduce the latency of DRAM further, they propose reducing the size of each individual DRAM subarray and using low threshold-voltage transistors in peripheral circuits and H-tree buffers. They show that compared to the baseline using 3D DRAM as main memory, their approach of using 3D DRAM for both cache and main memory improves the performance significantly.

Xu et al. [7] study different NoC (network-on-chip) designs with SRAM and DRAM LLCs to evaluate the effect of DRAM cache in terms of access latency, cache size, area and power consumption. Compared to 2D SRAM LLC, both 2D DRAM LLC and 3D DRAM LLC reduce miss-rate which also reduces the total amount of router/link activities. However, compared to 2D DRAM LLC, 3D DRAM LLC shows lower latency due to shorter wire length but higher power consumption due to increased number of routers and links and the increased complexity of routers.

3.2 Techniques for managing metadata

Zhao et al. [39] evaluate latency/area/bandwidth tradeoffs of multiple DRAM cache designs, viz. 1) using tags-in-DRAM, 2) using tags-in-SRAM, 3) sectored cache [63] with tags in SRAM 4) using ‘partial tags’ in SRAM, 5) a combination of sectored cache and partial tags. For tags-in-SRAM approach, the tags are stored in part of last-level SRAM cache, which reduces its capacity for data storage. A sectored DRAM cache uses large granularity block, which is composed of multiple sub-blocks and the block address. Each sub-block consists of small cache block and its state. A sectored DRAM cache reduces tag overhead by using large granularity block and also reduces cache traffic by only bringing the sub-block and not the entire sector on each miss event. To offset its high miss-rate, prefetching is also used. The partial-tag scheme maintains few least significant bits of tags (i.e. partial tag) in SRAM and full tag directory in DRAM. By consulting partial tags, prediction of hit/miss can be made to avoid DRAM access on a predicted miss. They show that combining sectored cache with partial tags provides the highest performance improvement while also reducing the tag overhead.

Jiang et al. [12] present filter-based techniques to avoid bandwidth-wastage in page-based DRAM caches. Their first technique uses a filter cache that profiles pages and selects the hot (i.e. most frequently accessed) pages which are allocated in DRAM cache. By not allocating pages with low spatial locality in DRAM cache, their technique avoids bandwidth wastage. The limitation of this technique is that when a victim page is evicted from filter cache, its access history is lost and on a later access, this page is considered as new and hence, many pages get evicted before reaching the threshold for being considered as hot. To avoid this, the second technique stores the counter of the page evicted from filter cache into memory and restores it later when the page is accessed. Thus, the second technique improves the accuracy of hot-page identification which allows for reducing the size of filter cache. Their third technique adapts to the intra- and inter-application variation by dynamically activating or deactivating the filter cache based on whether an application or its phase does or does not (respectively) saturate the memory bandwidth. When the filter cache is deactivated, all pages are allocated in the DRAM cache.

Loh et al. [29] propose a technique to enable use of conventional block size (64B) in DRAM cache. They store tags in DRAM and schedule tag and data accesses as a compound access which ensures that data access always results in a row buffer hit. For this, a 2KB DRAM row which can store up to thirty-two 64B data blocks is reorganized such that it stores twenty-nine 64B data blocks along with 6B tag for each of those data blocks (18B are left unused). Thus, at the cost of DRAM capacity, their technique provides scalable tag management approach by storing the tag in DRAM. On a DRAM cache miss, the overhead of DRAM access is incurred and to avoid this, they maintain a structure called MissMap which tracks the contents of DRAM cache. When MissMap determines a cache miss, access to the cache is skipped. The limitation of their approach is the large size of MissMap due to which storing it in last level SRAM cache is unattractive. Note that a few other works also use MissMap for DRAM cache management [11, 44–46].

Sim et al. [35] note that the key idea used in MissMap design of maintaining *precise* information about DRAM cache contents is overly conservative. To avoid the overhead of MissMap, they use low-cost hit/miss predictor which helps in *speculating* whether a request can be served by the DRAM cache or main memory. They also use a load-balancing scheme which redirects memory requests to either the die-stacked DRAM cache or off-chip main memory based on the instantaneous queuing delays at the two memories. This helps in improving overall system bandwidth by utilizing the otherwise idle off-chip bandwidth when the DRAM cache is servicing a burst of cache hits. The limitation of these techniques is that in presence of dirty data, they may lead to incorrect execution since they may access the stale value in main memory. To address this, they use a hybrid write policy that operates the majority of DRAM cache in write-through mode and enables write-back only for a few hot pages. This ensures that the DRAM cache is mostly clean and the requests to only the dirty pages need to go to DRAM cache. This obviates the need of waiting on predictor verification and allows more opportunities to redirect requests to off-chip memory. Their technique is useful when latency and storage overheads of MissMap are unacceptable and workloads are write-unintensive but benefit from larger system bandwidth achieved by their load-balancing approach.

To avoid the overhead of MissMap, El-Nacouzi et al. [31] propose a predictor for estimating hit/miss in a DRAM cache. They note that due to spatial locality, first access to a page will be followed by access to other blocks in the same page. Also, at any time, only few pages will experience misses and these misses are likely to happen soon after the first miss. Hence, tracking a small number of pages and their blocks is likely to provide reasonable prediction accuracy. Based on this, their technique uses two filters viz. coarse and fine-grain. The coarse-grain filter tracks a superset of pages that have cached blocks in DRAM cache and the fine-grain filter tracks all blocks in selected pages. Their technique first checks coarse-grain filter and if the page is predicted to be cached, fine-grain filter is checked to see if the block is predicted to be cached. If any of the filters predicts a miss, access to DRAM cache is bypassed.

Qureshi et al. [8] note that since DRAM caches are slower than conventional caches, optimizations that degrade the already high hit latency (e.g. use of MissMap [29] and tag-data serialization) may reduce the performance even if they provide a small improvement in hit rate. Hence, DRAM cache should be optimized first for latency and then for hit rate. They propose using a direct-mapped DRAM cache instead of set-associative cache, which also improves the row-buffer hit to further reduce the access latency. Furthermore, to avoid tag serialization latency, both data and tag are retrieved together on each access. They also use a memory access predictor to estimate whether the data is present in DRAM cache. If so, the DRAM cache is accessed before accessing main memory to save bandwidth. If the data are unlikely to be present in DRAM cache, both the DRAM cache and main memory are accessed in parallel to reduce the latency by avoiding cache miss penalty.

To bring together the best of both block-based and page-based cache designs, Jevdjic et al. [53] present a DRAM cache design which allocates data at the granularity of pages, but fetches only those blocks within a page that will be touched during the page's residency in the cache. Their technique also identifies pages that do not show spatial or temporal locality and does not allocate such pages in the cache. For this, their technique uses a predictor which identifies the footprint of the pages based on the observation that there is a high correlation between data and the code that accesses those data. Based on this, the first instruction that accesses a page provides hint about the data that the page contains and by monitoring and remembering which blocks the code further accesses, a prediction can be made about which blocks will be demanded when another page is accessed by the same piece of code. They show that their technique achieves high hit ratio, small lookup latency and low tag array overhead while also eliminating the off-chip traffic overhead of page-based designs.

Jevdjic et al. [51] aim to bring the best of two previous approaches [8, 53] together. They use page-based cache allocation to achieve high hit rate and low tag overhead, while estimating and fetching only the useful blocks within each page to minimize bandwidth wastage. They use set-associative cache along with a way-predictor which avoids fetching all the ways in parallel. This avoids the need of direct-mapped cache which induces conflict misses. Also, they maintain single tag per page to simplify footprint tracking and the tag read is overlapped with data read to avoid tag serialization latency.

Hameed et al. [46] propose a DRAM cache architecture where each DRAM row comprises four cache sets, consisting of 1 tag block and 7 data blocks (i.e. 7-way set associativity). The DRAM cache set number within a row is determined by the two least significant bits of the memory block address. By virtue of looking up smaller number of tags than the miss-rate optimized design and having higher row-buffer hit rate, their technique achieves lower DRAM hit latency. Also, compared to the direct-mapped DRAM cache, it provides lower miss-rate by using a 7-way associative cache. Since non-uniform distribution of accesses to different cache sets may increase conflict misses, they propose a set-mapping policy which assigns the DRAM row number to each core in a round robin manner. This leads to uniform distribution of accesses to different sets of DRAM cache which reduces the miss-rate.

3.3 Using tag cache

Huang et al. [52] note that although the tags-in-SRAM approach incurs higher area overhead than the tags-in-DRAM approach, its performance is superior. To bring the best of both together, their technique maintains the tags in DRAM, but also caches a small fraction of tags in a dedicated SRAM cache, called aggressive tag-cache (ATCache). ATCache leverages temporal locality by caching the tags of recently accessed DRAM sets only and spatial locality by prefetching the tags of adjacent DRAM cache sets only when such locality amongst the sets has been confirmed. They show that due to the small access latency of ATCache, their technique achieves much better performance than tags-in-DRAM approach. ATCache incurs only a fraction of area overhead of all the tags-in-SRAM approach but has disadvantage of *duplicating (due to caching) some tags* in SRAM which are also in DRAM cache and this overhead grows with increasing size of DRAM cache. Hence, their technique and similar techniques (e.g. [27, 32]) which cache the tags on-chip, are useful when resultant waste of on-chip space and higher complexity of tag management are acceptable for achieving performance comparable to that of having all the tags-in-SRAM.

Hameed et al. [27] present a technique to address the high tag latency of large DRAM caches. For a system with large L3 SRAM cache and L4 DRAM cache, they propose small SRAM structures named SRAM tag-cache and DRAM tag-cache, which hold the tags of the sets that were recently accessed in L3 and L4, respectively. These tag-caches quickly identify hit/miss for large caches and thus reduce the tag lookup latency. For applications with limited spatial locality, the hit-rate in tag-cache remains small. To avoid this, their technique identifies the number of useless insertions into tag-caches based on the behavior in recent past and restricts such insertions for increasing their hit rates.

3.4 Use of adaptive data granularity

Meza et al. [32] store the metadata in DRAM in the same row as their data to avoid row buffer miss on DRAM cache hits. To further reduce the metadata lookup latency, they cache the metadata for the most recently accessed DRAM rows in a small on-chip buffer (cache). Due to data access locality, the metadata for hot data are likely to be cached on-chip and hence, they can be accessed with the same latency as the SRAM tag store. Compared to a "full" SRAM tag store, the storage overhead of their "hot" SRAM tag buffer is three orders of magnitude smaller. To achieve a balance between locality and bandwidth consumption, they also propose dynamically adjusting the migration granularity. By testing with different granularities, e.g. 128B, 256B, 512B, 1KB, 2KB, 4KB and no migration, their technique decides the right migration granularity for a thread. This improves the DRAM cache utilization and reduces bandwidth contention at the cost of additional complexity of adaptive migration granularity.

Gulur et al. [21] present a technique which organizes the data in bi-modal fashion, such that the blocks with high spatial locality are organized as large blocks and those with limited spatial locality as small blocks. By dynamically selecting the right granularity of storage for different blocks at runtime, their technique uses the DRAM cache capacity effectively and also reduces the off-chip memory bandwidth consumption. Each set can hold X big (512B) and Y small (64B) blocks. Denoting the state of the set as (X,Y), a 2KB set can have states (4, 0), (3, 8), (2, 16). Thus, the set-associativity (=X+Y) can vary dynamically on per-set basis, but the number of sets is fixed. By tracking the utilization of 64B sub-blocks in sampled sets, their technique decides whether the sampled way should be classified as a big or a small block. Unlike the scheme of Loh et al. [29] where the metadata are interleaved with data on the same DRAM rows, they store the metadata for data banks belonging to one channel onto a bank of another channel which enables packing more metadata entries per page increasing the row-buffer hit rate. At the same time, it allows concurrent access of metadata and data due to high internal bandwidth of stacked DRAM. To further reduce the cache hit latency, their technique uses a small SRAM based way locator which caches the way IDs of the most recent accesses to DRAM cache sets. On a hit to way locator, access to DRAM for metadata is avoided. They show that their technique improves average access latency over both tags-in-DRAM and tags-in-SRAM approaches. The limitation of their technique comes from the additional complexity of organizing the data in bi-modal manner.

A comparison of bandwidth-saving approaches: It is noteworthy that Jiang et al. [12] optimize bandwidth by using *adaptive page allocation* in DRAM, which is different from Meza et al. [32] and Gulur et al. [21] who use *adaptive fetch granularity* to optimize bandwidth. Sim et al. [35] and Gulur et al. [22] (refer Section 3.13) optimize bandwidth by utilizing otherwise idle off-chip bandwidth also, in addition to bandwidth of die-stacked DRAM cache. Qureshi et al. [8] optimize bandwidth by reading tag and data together (i.e. avoiding tag-data serialization) and using a direct-mapped cache that avoids transferring multiple tags as in a set-associative cache. Jevdjic et al. [53] use page-based cache design but still save bandwidth by fetching selected blocks from the page that will be accessed while the page resides in the cache.

3.5 Exploring two-level memory

Chou et al. [10] propose an approach to use the stacked DRAM both as part of main memory to increase its capacity and as cache to capture data-locality and perform data management at fine-granularity. This approach is especially useful for large-sized stacked memories, such as 4GB stacked memory and 12GB main memory. Their technique stores recently accessed data lines in stacked DRAM and swaps the victim lines to main memory. Since this dynamically changes the physical location of a line, they also use a line location table (LLT) which tracks the physical location of all data lines. The LLT acts similar to a tag directory for conventional caches, except that LLT also identifies the location of line in main memory, in case it is not found in stacked DRAM. LLT is co-located with data in stacked DRAM to avoid SRAM storage overhead. For the lines not stored in stacked DRAM, LLT lookup and memory access become serialized and to avoid this, they use a line location predictor. If this predictor estimates the line to be in main memory, it is accessed from main memory in parallel with the LLT access. They show that their technique provides larger improvement than that offered by using stacked DRAM either as a cache or as a part of main memory.

Meswani et al. [48] note that given the limited capability provided by die-stacked DRAM, the off-chip memory is still required to satisfy the memory requirement of large-scale applications. This leads to a two-level memory (TLM) design which demands co-design of memory architecture and software applications. To manage such TLM, they study hardware-based, OS-based and programmer-driven approaches and compare their relative merits. The hardware approach uses stacked DRAM as last level cache. The OS-based approach uses this as main memory and the off-chip memory is used as the conventional swap-device. In programmer-driven approach, the information about existence of multiple memories is exposed to the programmer who can decide about allocating or pinning different data objects on different memories. Their study using exascale proxy applications shows that the hardware cache achieves high hit rate and consumes small bandwidth. The programmer-managed TLM also consumes small bandwidth but has a low hit rate while the OS-based approach wastes bandwidth due to use of page-granularity allocation.

3.6 Addressing issues related to multicore processors

Hameed et al. [45] note that application-unaware bank-mapping policies in large DRAM caches can lead to unfair slowdown of some applications. Towards this, their technique tracks the run-time miss rate information of concurrently executing applications to detect thrashing applications. Also, the cache banks are organized into two regions namely "private region" and "shared region". Since thrashing applications may evict useful cache lines of other applications, their technique aims to confine the memory segments from thrashing applications into a single cache bank. This reduces the inter-core cache contention and provides performance isolation between thrashing and non-thrashing applications. Further, to improve the utilization of cache, the shared region is allocated to non-thrashing applications which provides cooperative cache sharing.

Loh [42] propose a cache organization scheme for DRAM cache to improve its performance. Each set is organized as multi-level logical queues, where a single cache line constitutes the queue entry. All cache lines are initially inserted into the first-level queue. After Q insertions to a queue of size Q ($< W$, where W is the set-associativity), the original line leaves this queue. If it has not received any hit during its residency in *this* queue, it is evicted, otherwise, it is inserted into next (second) level queue (and so on). The last queue uses LRU replacement policy. Using this organization, a line which sees no reuse is evicted after Q insertions only, while in conventional LRU replacement policy, it will be evicted after W insertions. This improves the efficiency of cache by ensuring quick eviction of dead lines. The lines seeing burst-accesses satisfy those requests while residing in first-level queue and after the burst has ended, these lines are evicted from second-level queue. Thus, the key idea behind using multi-level queues is that they act as filters for cache access patterns with limited reuse. In multicore processors, each core has its own dedicated first-level queue and the second-level queue can be shared. By virtue of isolating traffic from different cores into different first-level queues, their organization also provides performance isolation to those cores, since the cores with streaming behavior cannot replace lines of other cores. The increased complexity in replacement logic and design of queues are limitations of this technique.

The difference between the technique of Hameed et al. [45] and Loh [42] is that for providing inter-core isolation, the former uses a *bank mapping policy* that explicitly limits the cache quota of thrashing applications, while the latter uses a *cache replacement policy* that indirectly accelerates eviction of lines from thrashing applications or those with little reuse. Thus, the former technique uses 'strict allocation', while the latter uses 'soft allocation'.

3.7 DRAM caches for enabling multicore scaling

Rogers et al. [4] study how multicore scaling may be restricted due to bandwidth-wall problem and to what extent can the bandwidth conservation techniques mitigate this challenge. They evaluate several techniques such as use of smaller sized cores to enable larger sized caches, 3D stacked caches, cache compression and link compression and DRAM cache. Under proportional scaling, the number of cores on the chip should reach 16 ($= 2^4$) in four technology generations, however, bandwidth wall limits it to only 11 cores (assuming SRAM L2 caches). They show that on assuming the DRAM density to be 4X and 8X compared to that SRAM, the number of supportable cores reaches 16 and 18, respectively. Although their model makes some simplifying assumptions (e.g. ignoring refresh overhead etc.), their results confirm that use of DRAM caches is an effective approach to filter extra memory traffic generated by large number of cores for enabling multicore scaling. They also show that by combining use of DRAM cache with other bandwidth conservation approaches (mentioned above), the bandwidth-wall issue can be delayed by several technology generations, thus allowing the multicore scaling to continue.

Hardavellas et al. [5] note that due to power-wall and off-chip bandwidth-wall, an increase in core-count does not translate into corresponding performance improvements. They show that power-wall can be partially overcome by using customized energy-efficient heterogeneous multicores and for overcoming the bandwidth-wall, large 3D-stacked DRAM caches can be used. The headroom provided by mitigation of bandwidth-wall allows for adding more cores on the chip for performance scaling, although in such case, the increased energy consumption of network subsystem becomes the new bottleneck. In other words, the benefits of using DRAM cache are bounded.

Pan et al. [62] study the use of die-stacked DRAM cache in VLIW (very large instruction word) processors. They note that die-stacking enables the latency of a 3D DRAM cache to approach that of a 2D SRAM cache. Using this, they propose replacing the 2D SRAM cache by 3D DRAM cache and allocating the saved area to additional clusters for improving parallelism and performance. They show that a four cluster system with 3D DRAM L2 cache and 3D DRAM main memory occupies similar logic die area as a two cluster system with 2D SRAM L2 cache and 3D DRAM main memory and the former also provides better performance.

Thus, while Rogers et al. [4], Hardavellas et al. [5] and Pan et al. [62] use different models or evaluation platforms, they all provide the same conclusion that DRAM caches can be effective in allowing multicore scaling to continue.

3.8 Refresh management techniques

Jaksic et al. [59] present a DRAM-based L1 and L2 coherent cache design. To reduce the refresh overhead, the refresh operation for a block are decided based on its coherence state. For example, for MESI (modified, exclusive, shared, invalid) coherence protocol, they show that an invalid line need not be refreshed, a line in shared or exclusive state transitions to invalid if not refreshed but need not be written back and a line in modified state needs to be invalidated and written back if it is not refreshed. Based on the combination of the line states that can be refreshed or not, they propose multiple refresh policies ranging from one where no line is refreshed to one where modified, shared and exclusive lines are refreshed. By choosing a suitable refresh policy, a tradeoff between performance loss and energy saving can be achieved. They also show that compared to a design with SRAM-based caches, the design with DRAM-based cache provides significant energy saving with only small performance loss.

Ghosh et al. [61] present a technique to reduce the refresh requirement of a 3D DRAM cache. They note that from the perspective of data retention, an access to a memory row performs an operation equivalent to a regular refresh and hence, the conventional refresh mechanisms which periodically refresh all memory rows for retaining data lead to wastage of energy. Based on this, their technique maintains counters for each row in the memory module and avoids refreshing a memory row if it has recently seen a read/write access.

The difference between the technique of Jaksic et al. [59] and Ghosh et al. [61] is that former uses properties of cache coherence to skip refresh operations, while the latter uses equivalence of access operation to refresh operation from refresh perspective, to skip refresh operation to a row that has just seen an access operation.

3.9 Thermal management techniques

Milojevic et al. [2] study a server-on-chip architecture targeted for the datacenter market. The bottom layer of the chip has many-core compute engine and the top layer has wide I/O DRAMs (similar to [13]) which are used as LLC shared by all cores. To optimize the total chip throughput, the bottom layer allocates minimal amount of area to second-level cache, which is just enough to capture the instruction and data working sets of the data-intensive commercial workloads. A larger fraction of area is dedicated to many cores. Their evaluation using CPU-centric workloads (e.g. SPECInt and Dhrystone) shows that temperature in the server-on-chip (logic+DRAM) lies in the range of 175-200°C which exceeds the reliable operating bounds. By comparison, cloud-workloads consume less power in the processing cores due to their memory-bound nature and hence, an evaluation using cloud workloads shows reduced power density at hotspots and temperatures that are within operating bounds of stacked DRAM. Thus, their study shows that a server-on-chip system is feasible even with a low-cost cooling option which provides opportunity for cost and energy savings in datacenters.

Yun et al. [60] note that the high integration density of 3D chips may lead to high operating temperature, which increases leakage power consumption and error-rate. Towards this, they present a DVFS scheme for 3D-stacked DRAM cache where voltage/frequency of each cache bank or each group of cache banks can be adjusted, based on the error-rate (due to retention and sensing failure), cache access rate and temperature-induced power consumption. Their technique monitors the cache access rate and temperature of cache zones in each time-interval at runtime. For applications with low frequency of accesses, low voltage and frequency are used to reduce power consumption with small performance loss and vice versa. Since the error-rate increases with decreasing supply voltage, the minimum value of voltage is also determined by the error-rate constraint.

Thus, Milojevic et al. [2] study and compare temperature profile for CPU-centric and cloud workloads, while Yun et al. [60] present a DVFS-based technique for thermal management.

3.10 Ensuring resilience to soft-errors and process-variation

Given the high reliability requirement of die-stacked DRAMs, Sim et al. [36] propose a technique which provides protection at both fine-grain (e.g. single-bit faults) and coarse-grain (e.g. for row, bank and channel-faults) levels. They note that in a die-stacked DRAM LLC, adding an extra chip in the stack for storing ECC information may not be practical since the overhead of ECC becomes very large and hence, they utilize only non-ECC stacked DRAM chips. Instead of using SECDED (single error correction, double error detection) ECC, they use SEC and CRC (cyclic redundancy check). CRCs can detect multi-bit errors, regardless of whether these errors are clustered and thus, they greatly improve the error-detection capability and reduce the silent data corruption (SDC) rates [64]. They also note that on an error in clean data in cache, merely detecting the error is sufficient since the correct copy can be retrieved form the main memory. By comparison, for the dirty data, both error correction and detection are required, since the modified copy in the cache is the only valid copy [64]. Based on this, their technique relies on main memory to correct errors in clean blocks in DRAM cache and uses duplication of dirty blocks in other banks to provide error-correction for them. To avoid the capacity overhead of duplication, they propose duplicating only critical applications or memory regions and writing dirty blocks from the DRAM cache to main memory.

Zhao et al. [50] note that process variation can lead to non-uniform access times in different subbanks in a DRAM LLC. They propose techniques which work by migrating data from slow subbanks to fast subbanks to reduce the average access time. One technique always migrates data to fastest subbank. Since this may cause contention in fastest subbank, a second technique divides the subbanks in a rank into several tiers where each tier has several subbanks of similar speed. The data are migrated from a slow tier to the fastest tier where the data are uniformly distributed to its subbanks. Since DRAM cache provides large capacity, the working set of several applications fits in the fast subbanks which makes their technique effective.

3.11 Cache reconfiguration techniques

Chang et al. [54] note that die-stacked DRAM caches provide large capacity, however, several applications and application phases may not benefit from this capacity due to their small working set size. Hence, the unused cache portions consume power due to refresh overhead and leakage in the peripheral circuitry. They present a cache reconfiguration technique for such caches. Conventional reconfiguration approaches (e.g. [26, 37]) remap the data stored in sets of both powered-on and powered-down banks, which leads to large overhead. Their technique works using consistent hashing algorithm [54] whereby a machine failure (or analogously, powering-down a DRAM bank) results in remapping only the data mapped to that machine (or DRAM bank). Further, the remapping is done in a load-balanced manner which does not turn the powered-on banks into hotspots leading to increased cache access latencies, and the data already mapped to the powered-on banks remain where they were. While previous techniques e.g. [37] allow reconfiguring the cache to power-of-two set-counts only, the technique of Chang et al. [54] does not impose such restriction.

3.12 SRAM/DRAM Hybrid Caches

Madan et al. [49] present a 3D hybrid SRAM/DRAM cache design that aims to bring together the low-latency advantage of SRAM and high density advantage of DRAM. They assume a stacked processor with three layers, where bottom layer contains 16 cores, the middle layer contains 16 SRAM banks and the top layer contains 16 DRAM banks. Both SRAM and DRAM together constitute the L2 cache. Based on offline analysis, they first identify pages private to each core and those shared by multiple cores. Afterwards, using OS page coloring, they place private pages in the SRAM bank directly above the core and shared pages in one of the central SRAM banks. Since the applications running on different cores present different L2 cache demand, spilling the pages of an application in adjacent banks can help in adjusting the cache quota of each core, however, this also increases the access latency and pressure on the inter-bank network. To avoid this, their technique provisions spilling the additional pages into the third dimension, namely the DRAM bank directly above the SRAM cache bank. Since the density of DRAM is 8× compared to that of SRAM, activation of a DRAM bank increases the cache capacity from 1MB to 9MB. Further, by attempting to serve most of the requests from SRAM bank, their technique keeps the average cache access latency small. As for cache reconfiguration approach, Madan et al. use selective-way approach while Chang et al. [54] use selective-set approach [26].

Inoue et al. [43] present an SRAM/DRAM hybrid cache architecture. They assume a processor designed with small SRAM L2 cache in same layer as the core layer (note the difference with the design of Madan et al. [49] where SRAM cache is in a separate layer) and a large die-stacked DRAM L2 cache. Using profiling, the cache requirement of an application is estimated and based on it, the cache is configured to work in one of the following two modes before the application execution. When the cache requirement is small, the DRAM cache is power-gated and only the SRAM cache is used, thus, access latency is kept small. For applications with large cache requirement, both SRAM and DRAM are kept active and the SRAM is used to store the tags for data stored in DRAM cache. In this mode, the capacity becomes higher, at the cost of higher access latency of DRAM. Thus, at a time, either SRAM or DRAM, but not both, store the data, which makes their technique different from that of Madan et al. [49], where both SRAM and DRAM may simultaneously store the data.

Hameed et al. [44] propose a hybrid SRAM-DRAM LLC (L3) which avoids data duplication compared to a hierarchy using L3 SRAM and L4 DRAM caches. They note that in a shared LLC, unnecessary fill requests from thrashing applications can delay the critical read/write accesses from non-thrashing applications, which creates inter-core DRAM interference. To address this, they propose a policy to dynamically decide whether a line brought from main memory is placed in SRAM and DRAM part of LLC or only in SRAM part. Their technique detects whether an application is thrashing based on the reuse distance and for such applications, the line is placed in DRAM cache with low probability. This avoids unnecessary insertions into DRAM. Unlike Inoue et al. [43], they do not power-gate the DRAM for saving energy, but instead focus on reducing inter-core interference in DRAM for improving performance.

3.13 Performance model for DRAM caches

Gulur et al. [22] present an analytical performance model of the DRAM Cache for both tags-in-SRAM and tags-in-DRAM organizations. Their model accounts for key parameters such as DRAM Cache and off-chip memory timing values, cache block size, hit rate of tag cache/predictor and application characteristics. Their model estimates the average access latency and request arrival rate at DRAM cache. Based on their model, they show that for the tags-in-DRAM design to outperform the tags-in-SRAM design, the hit rate of auxiliary tag cache/predictor needs to be very high. They also note that when DRAM cache hit rate is high, the large traffic at DRAM cache leads to contention and queuing delays. For such cases, some cache traffic can be diverted to idle main memory to achieve load-balancing. Using their model, they also present a load-balancing scheme which minimizes the average latency by identifying the optimal fraction of accesses to divert to main memory.

3.14 Comparison with NVM caches

Dong et al. [56] compare STT-RAM cache with SRAM and DRAM caches in terms of area, performance, and energy. NVMs such as STT-RAM consume near-zero leakage power, however, their write latency and energy are significantly higher than that of SRAM and even DRAM [20]. The density of STT-RAM is close to DRAM and is 4X compared to SRAM. Using architectural simulations they show that due to the requirement of refresh operations, DRAM consumes significantly more power than STT-RAM, while their performance values are nearly similar.

While byte-addressable NVMs may show superior energy efficiency compared to DRAM, it is also noteworthy that in terms of commercial viability and maturity, DRAM is superior to these NVMs. Further, NVMs also present reliability issue [64], for example, the resistance of a PCM cell increases over time, due to which an MLC (multi-level cell) PCM can start representing different value than the one originally stored. Due to this, soft-error rate in a 4-level PCM may be 10^6 times higher than that in DRAM [65].

Furthermore, the write endurance of NVMs is orders of magnitude smaller than that of DRAM. For example, while the write endurance of DRAM is 10^{16}, this value for PCM and ReRAM is 10^8 and 10^{11}, respectively [20, 38, 66]. For STT-RAM, this value is expected to be 10^{15}, although best results so far show only 4×10^{12} [20]. This not only limits the lifetime of NVM devices, but also presents a crucial security vulnerability from both malicious attackers and greedy-users who may make the system fail by write-attacks just before the warranty period to get a new system. Unlike NVMs, DRAM does not face such write-endurance issues.

4 Future Challenges and Conclusion

While DRAM caches may provide a short-term solution to power issues, even they fall far short when measured against the goal of Exascale computing which seeks 10^{18} operations/second computation capability within a power budget of 20MW [37]. To accomplish this goal, solutions spanning across the whole software-stack are required. For example, at device-level, novel array organization can reduce refresh requirement and DRAM access energy [67]. At architecture level, multiple techniques such as data compression, rank subsetting, access scheduling etc. [24] can be synergistically integrated to bring the best of them together. Similarly, compiler and OS techniques can be used to perform data placement in a manner that improves hit rate by co-locating frequently used data on the same page [30].

With feature size scaling, the effect of process variation (which affects DRAM cell retention period) and susceptibility of DRAM cells to soft-errors also increase [64]. While initial studies on DRAM cache have mainly focused on architectural design and performance/energy issues, moving forward, a thorough evaluation of other issues such as resilience, thermal management etc. will be definitely required to ensure reliable operation of these DRAM caches.

Architectural studies performed using simulators or real-systems are crucial for evaluating design innovations proposed for DRAM caches. Due to their large size, DRAM caches require large evaluation window for performing representative studies. However, due to the slow speed of existing simulators, along with the requirement of storing huge workload traces, full design-space exploration of large caches may not be feasible. Further, due to the emerging nature of stacked DRAM caches, their real-world prototypes may not be widely available or may not be economically viable. Clearly, improvements in manufacturing and commercial viability of stacked DRAM, along with development of extremely-fast simulation infrastructures will be very helpful in boosting further research on DRAM caches.

Memory system plays a crucial role in determining performance of high-end computing systems and given the limitations of other memory technologies (e.g. SRAM and NVMs), DRAM promises to be the most suitable candidate for designing multi-gigabyte caches. In this paper, we reviewed several techniques for managing DRAM caches. To underscore their similarities and differences, we classified them based on several key parameters. We also briefly discussed the challenges that lie ahead in this area. It is hoped that this paper will be useful for chip-designers, computer-architects and system-researchers and will motivate further research in this area.

References

[1] S. Rusu, H. Muljono, D. Ayers, S. Tam, W. Chen, A. Martin, S. Li, S. Vora, R. Varada, and E. Wang, "Ivytown: A 22nm 15-core enterprise Xeon® processor family," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2014, pp. 102–103.

[2] D. Milojevic, S. Idgunji, D. Jevdjic, E. Ozer, P. Lotfi-Kamran, A. Panteli, A. Prodromou, C. Nicopoulos, D. Hardy, B. Falsari *et al.*, "Thermal characterization of cloud workloads on a power-efficient server-on-chip," in *International Conference on Computer Design (ICCD)*, 2012, pp. 175–182.

[3] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. Loh, "Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories," *HPCA*, 2015.

[4] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for CMP scaling," in *International Symposium on Computer Architecture (ISCA)*, 2009, pp. 371–382.

[5] N. Hardavellas, M. Ferdman, A. Ailamaki, and B. Falsafi, "Power scaling: the ultimate obstacle to 1K-core chips," Northwestern University, Tech. Rep. NWU-EECS-10-05, 2010.

[6] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCauley, P. Morrow, D. W. Nelson, D. Pantuso

et al., “Die stacking (3D) microarchitecture,” in *International Symposium on Microarchitecture*, 2006, pp. 469–479.

[7] T. C. Xu, P. Liljeberg, and H. Tenhunen, “Exploring DRAM last level cache for 3D network-on-chip architecture,” in *Advanced Materials Research*, vol. 403, 2012, pp. 4009–4018.

[8] M. K. Qureshi and G. H. Loh, “Fundamental latency tradeoff in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design,” in *International Symposium on Microarchitecture*, 2012, pp. 235–246.

[9] “Intel Xeon Phi Knights Landing Processors to Feature Onboard Stacked DRAM Supercharged Hybrid Memory Cube (HMC) upto 16GB,” http://wccftech.com/intel-xeon-phi-knights-landing-processors-stacked-dram-hmc-16gb/, 2014.

[10] C. C. Chou, A. Jaleel, and M. K. Qureshi, “CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache,” in *International Symposium on Microarchitecture (MICRO)*, 2014, pp. 1–12.

[11] S. Yin, J. Li, L. Liu, S. Wei, and Y. Guo, “Cooperatively managing dynamic writeback and insertion policies in a last-level DRAM cache,” in *Design, Automation & Test in Europe (DATE)*, 2015, pp. 187–192.

[12] 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 *International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[13] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H.-R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S.-K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, K. Kyung, J.-S. Choi, and Y.-H. Jun, “A 1.2V 12.8GB/s 2Gb mobile Wide-I/O DRAM with 4X128 I/Os using TSV-based stacking,” in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2011, pp. 496–498.

[14] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J. Ahn, D. Kwon, J. H. Kim, J.-W. Lee *et al.*, “8Gb 3D DDR3 DRAM using through-silicon-via technology,” in *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, 2009, pp. 130–131.

[15] Tezzaron Semiconductor, “Octopus 8-Port DRAM for Die-Stack Applications,” www.tachyonsemi.com/memory/datasheets/TSC10080x_0_1.pdf, 2010.

[16] J. Jeddeloh and B. Keeth, “Hybrid memory cube new DRAM architecture increases density and performance,” in *Symposium on VLSI Technology (VLSIT)*, 2012, pp. 87–88.

[17] D. U. Lee, K. W. Kim, K. W. Kim, K. S. Lee, S. J. Byeon, J. H. Kim, J. H. Cho, J. Lee, and J. H. Chun, “A 1.2 V 8 Gb 8-Channel 128 GB/s High-Bandwidth Memory (HBM) Stacked DRAM With Effective I/O Test Circuits,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 191–203, 2015.

[18] http://www.pcper.com/reviews/General-Tech/High-Bandwidth-Memory-HBM-Architecture-AMD-Plans-Future-GPUs, 2015.

[19] NVIDIA, http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/, 2014.

[20] S. Mittal, J. S. Vetter, and D. Li, “A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-volatile On-chip Caches,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2015.

[21] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, “Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth,” in *International Symposium on Microarchitecture (MICRO)*, 2014, pp. 38–50.

[22] N. Gulur, M. Mehendale, and R. Govindarajan, “A Comprehensive Analytical Performance Model of DRAM Caches,” in *International Conference on Performance Engineering*, 2015, pp. 157–168.

[23] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, “CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory,” in *Conference on Design, Automation and Test in Europe*, 2012, pp. 33–38.

[24] S. Mittal, “A Survey of Architectural Techniques For DRAM Power Management,” *International Journal of High Performance Systems Architecture*, vol. 4, no. 2, pp. 110–119, 2012.

[25] G. H. Loh, “3D-stacked memory architectures for multi-core processors,” in *International Symposium on Computer Architecture (ISCA)*, 2008, pp. 453–464.

[26] S. Mittal, “A survey of architectural techniques for improving cache power efficiency,” *Elsevier Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33–43, March 2014.

[27] F. Hameed, L. Bauer, and J. Henkel, “Reducing latency in an SRAM/DRAM cache hierarchy via a novel Tag-Cache architecture,” in *Design Automation Conference (DAC)*, 2014, pp. 1–6.

[28] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, “Simple but effective heterogeneous main memory with on-chip memory controller support,” in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[29] G. H. Loh and M. D. Hill, “Efficiently enabling conventional block sizes for very large die-stacked DRAM caches,” in *International Symposium on Microarchitecture*, 2011, pp. 454–464.

[30] G. H. Loh, N. Jayasena, K. McGrath, M. OConnor, S. Reinhardt, and J. Chung, “Challenges in heterogeneous die-stacked and off-chip memory systems,” in *In Proc. of 3rd Workshop on SoCs, Heterogeneity, and Workloads (SHAW)*, 2012.

[31] M. El-Nacouzi, I. Atta, M. Papadopoulou, J. Zebchuk, N. E. Jerger, and A. Moshovos, “A dual grain hit-miss detector for large die-stacked DRAM caches,” in *Design, Automation and Test in Europe*, 2013, pp. 89–92.

[32] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, “Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management,” *Computer Architecture Letters*, vol. 11, no. 2, pp. 61–64, 2012.

[33] M. Poremba *et al.*, “DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches,” in *Design Automation and Test in Europe*. IEEE, 2015, pp. 1543 – 1546.

[34] C. Chou, A. Jaleel, and M. K. Qureshi, “BATMAN: Maximizing Bandwidth Utilization of Hybrid Memory Systems,” Georgia Tech, Tech. Rep. TR-CARET-2015-01, 2015.

[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 *International Symposium on Microarchitecture (MICRO)*, 2012, pp. 247–257.

[36] J. Sim, G. H. Loh, V. Sridharan, and M. O’Connor, “Resilient die-stacked DRAM caches,” in *International Symposium on Computer Architecture*, 2013, pp. 416–427.

[37] S. Mittal and Z. Zhang, “EnCache: A Dynamic Profiling Based Reconfiguration Technique for Improving Cache Energy Efficiency,” *Journal of Circuits, Systems, and Computers*, vol. 23, no. 10, 2014.

[38] S. Mittal and J. S. Vetter, “AYUSH: A Technique for Extending Lifetime of SRAM-NVM Hybrid Caches,” *IEEE Computer Architecture Letters (CAL)*, 2015.

[39] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, “Exploring DRAM cache architectures for CMP server platforms,” in *International Conference on Computer Design*, 2007, pp. 55–62.

[40] Y. Chen, E. Li, C. Kim, and Z. Tang, “Efficient shared cache management through sharing-aware replacement and streaming-aware insertion policy,” in *IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–11.

[41] H. Sun, J. Liu, R. Anigundi, N. Zheng, J. Lu, R. Ken, and T. Zhang, “Design of 3D DRAM and Its Application in 3D Integrated Multi-Core Computing Systems,” *IEEE Design and Test of Computers*, 2013.

[42] G. H. Loh, “Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy,” in *International Symposium on Microarchitecture*, 2009, pp. 201–212.

[43] K. Inoue, S. Hashiguchi, S. Ueno, N. Fukumoto, and K. Murakami, “3D implemented SRAM/DRAM hybrid cache architecture for high-performance and low power consumption,” in *International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2011, pp. 1–4.

[44] F. Hameed, L. Bauer, and J. Henkel, “Adaptive cache management for a combined SRAM and DRAM cache hierarchy for multi-cores,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 77–82.

[45] F. Hameed, L. Bauer, and J. Henkel, “Reducing inter-core cache contention with an adaptive bank mapping policy in DRAM cache,” in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, 2013, pp. 1–8.

[46] F. Hameed, L. Bauer, and J. Henkel, “Simultaneously optimizing dram cache hit latency and miss rate via novel set mapping policies,” in *International Conference on Compilers, Architectures*

and Synthesis for Embedded Systems, 2013, p. 11.

[47] N. M. Tshibangu, P. D. Franzon, E. Rotenberg, and W. R. Davis, "Design of controller for L2 cache mapped in Tezzaron stacked DRAM," in *IEEE International 3D Systems Integration Conference (3DIC)*, 2013, pp. 1–4.

[48] M. R. Meswani, G. H. Loh, S. Blagodurov, D. Roberts, J. Slice, and M. Ignatowski, "Toward efficient programmer-managed two-level memory hierarchies in exascale computers," in *Hardware-Software Co-Design for High Performance Computing (Co-HPC)*, 2014, pp. 9–16.

[49] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing communication and capacity in a 3D stacked reconfigurable cache hierarchy," in *International Symposium on High Performance Computer Architecture*, 2009, pp. 262–274.

[50] B. Zhao, Y. Du, Y. Zhang, and J. Yang, "Variation-tolerant non-uniform 3D cache management in die stacked multicore processor," in *IEEE/ACM international Symposium on Microarchitecture*, 2009, pp. 222–231.

[51] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *International Symposium on Microarchitecture (MICRO)*, 2014, pp. 25–37.

[52] C.-C. Huang and V. Nagarajan, "ATCache: reducing DRAM cache latency via a small SRAM tag cache," in *International conference on Parallel architectures and compilation*, 2014, pp. 51–60.

[53] 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 *International Symposium on Computer Architecture*, 2013, pp. 404–415.

[54] K. Chang, G. H. Loh, M. Thottethodi, Y. Eckert, M. OConnor, L. Subramanian, and O. Mutlu, "Enabling Efficient Dynamic Resizing of Large DRAM Caches via A Hardware Consistent Hashing Mechanism," Carnegie Mellon University, Tech. Rep. 2013-001, 2013.

[55] K. H. Park, S. K. Park, H. Seok, W. Hwang, D.-J. Shin, J. H. Choi, and K.-W. Park, "Efficient memory management of a hierarchical and a hybrid main memory for MN-MATE platform," in *International Workshop on Programming Models and Applications for Multicores and Manycores*, 2012, pp. 83–92.

[56] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement," in *Design Automation Conference*, 2008, pp. 554–559.

[57] S. Lee, J. Jung, and C.-M. Kyung, "Hybrid cache architecture replacing SRAM cache with future memory technology," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 2481–2484.

[58] C. Chou, A. Jaleel, and M. K. Qureshi, "BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches," in *International Symposium on Computer Architecture (ISCA)*, 2015.

[59] Z. Jaksic and R. Canal, "DRAM-based coherent caches and how to take advantage of the coherence protocol to reduce the refresh energy," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2014, pp. 1–4.

[60] W. Yun, J. Jung, K. Kang, and C.-M. Kyung, "Temperature-aware energy minimization of 3D-stacked L2 DRAM cache through DVFS," in *International SoC Design Conference (ISOCC)*, 2012, pp. 475–478.

[61] M. Ghosh and H.-H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D Die-Stacked DRAMs," in *International Symposium on Microarchitecture*, 2007, pp. 134–145.

[62] Y. Pan and T. Zhang, "Improving VLIW processor performance using three-dimensional (3d) DRAM stacking," in *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2009, pp. 38–45.

[63] C. Anderson and J.-L. Baer, "Design and evaluation of a subblock cache coherence protocol for bus-based multiprocessors," Univ. of Washington, Tech. Rep. 94-05-02, 1994.

[64] S. Mittal and J. Vetter, "A Survey of Techniques for Modeling and Improving Reliability of Computing Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2015.

[65] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," in *International Symposium on Computer Architecture*, 2013, pp. 440–451.

[66] S. Mittal, "A survey of power management techniques for phase change memory," *International Journal of Computer Aided Engineering and Technology (IJCAET)*, 2014.

[67] B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H. M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw, "Exploring DRAM organizations for energy-efficient and resilient exascale memories," in *Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, 2013, p. 23.

Sparsh Mittal received the B.Tech. degree in electronics and communications engineering from IIT, Roorkee, India and the Ph.D. degree in computer engineering from Iowa State University, USA. He is currently working as a Post-Doctoral Research Associate at ORNL. His research interests include cache and main memory systems, resilience and energy efficiency, non-volatile memory and GPU architectures.

Jeffrey S. Vetter , Ph.D., holds a joint appointment between ORNL and Georgia Institute of Technology (GT). At ORNL, he is a Distinguished R&D Staff Member, and the founding group leader of the Future Technologies Group. At GT, he is a Joint Professor in the Computational Science and Engineering School, the Project Director for the NSF Track 2D Experimental Computing Facility for large scale heterogeneous computing using graphics processors, and the Director of the NVIDIA CUDA Center of Excellence. His research interests include massively multithreaded processors, memory architecture for extreme-scale systems and heterogeneous multicore processors.