2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

Efficient Caching with A Tag-enhanced DRAM

Maryam Babaie
University of California, Davis
mbabaie@ucdavis.edu

Ayaz Akram†
Samsung Electronics
ayaz.akram@samsung.com

Wendy Elsasser
Rambus Labs, Rambus Inc.
welsasser@rambus.com

Brent Haukness
Rambus Labs, Rambus Inc.
haukness@rambus.com

Michael R. Miller
Rambus Labs, Rambus Inc.
michaelm@rambus.com

Taeksang Song‡
Samsung Electronics
taeksang.song@samsung.com

Thomas Vogelsang
Rambus Labs, Rambus Inc.
tvogelsang@rambus.com

Steven C. Woo
Rambus Labs, Rambus Inc.
swoo@rambus.com

Jason Lowe-Power
University of California, Davis
jlowepower@ucdavis.edu

***Abstract*—As SRAM-based caches are hitting a scaling wall, manufacturers are integrating DRAM-based caches into system designs to continue increasing cache sizes. While DRAM caches can improve the performance of memory systems, existing DRAM cache designs suffer from high miss penalties, wasted data movement, and interference between misses and demands. In this paper, we propose TDRAM, a novel DRAM microarchitecture tailored for caching. TDRAM enhances existing DRAM, such as HBM3, by adding small, low-latency mats to store tags and metadata on the same die as the data mats. These mats enable tag and data access in lockstep, in-DRAM tag comparison, and conditional data response based on the comparison result (reducing wasted data transfers), akin to SRAM cache mechanisms. TDRAM further optimizes hit and miss latencies through opportunistic early tag probing. Moreover, TDRAM introduces a flush buffer to store conflicting dirty data on write misses, eliminating data bus turnaround delays on write demands. We evaluate TDRAM in a full-system simulation using a set of HPC workloads with large memory footprints, showing that TDRAM, on average, provides 2.65× faster tag checks, 1.23× speedup, and 21% less energy consumption compared to state-of-the-art commercial and research designs.**

I. Introduction

Today's high-performance computers leverage heterogeneous memory systems, combining high-performance memories such as HBM with high-capacity, lower-performance ones to meet the memory demands of tasks like machine learning and artificial intelligence. Interconnect technologies like Compute Express Link (CXL) further enhance memory heterogeneity by integrating local and remote memory pools. Intel's Sapphire Rapids CPU exemplifies this approach by utilizing HBM DRAMs as cache [24], [62], effectively addressing the scaling limitations of SRAM [3]. The expanded cache capacity and enhanced bandwidth of HBMs offer the potential for improved data locality without programmer intervention.

However, the potential benefits of DRAM caches have not borne fruit. Previous studies of DRAM caches have shown that using standard DRAM devices as a cache can slow down applications with large memory footprints and high miss rates [37], [59]. Current designs of DRAM caches, such as Intel's Cascade Lake [6], [37], store tags and metadata together with the cache line data in the same DRAM. Storing tags and data together reduces hit time for read demands [58]; however, it significantly increases miss penalties, as a separate DRAM read is necessary to retrieve tag and metadata for hit/miss determination and status information, causing contention with read demands. Additionally, *all write requests, including those hitting in the cache, require a DRAM read* to fetch both tag and data to ensure dirty data is not overwritten. This exacerbates contention and leads to expensive turnaround bubbles on the data bus [17]. These extra accesses for read misses and write demands increase: (i) service time of missed demands, (ii) contention in the read buffer, which extends queue occupancy time, and (iii) wasted data movement and energy consumption. Many of today's applications have high miss rates in DRAM caches, causing these issues to negatively impact workload performance. Since SRAM caches cannot scale to the capacities required by today's applications, it becomes imperative to enhance existing DRAM cache designs to address these challenges.

In this work, we introduce TDRAM (**T**ag-enhanced **DRAM**), a DRAM microarchitecture specifically tailored for caching. TDRAM enhances existing DRAM, such as HBM3, by adding a set of small, low-latency mats to store tags and metadata on the same die as the data mats. These tag mats enable faster access by reducing wordline and bitline lengths compared to the data mats. The additional on-die storage is sufficiently large to accommodate the tag and metadata for all DRAM cache lines; thus, the tag store scales with the data capacity. By placing the tags in separate fast mats, TDRAM enables rapid on-die tag checking, which reduces the latency for demand misses, mitigates contention on the DRAM read queue, and decreases wasted data transfers (and thus energy).

TDRAM extends HBM3's interface in three ways: (1) It adds a unidirectional hit-miss (HM) bus to transfer the tag check result and metadata to the controller. (2) It introduces

† Work completed while at *University of California, Davis.*
‡ Work completed while at *Rambus Labs, Rambus Inc.*

TABLE I: Comparison of TDRAM with Related Work

Tag storage maintained on:	On the processor die		Off the processor die				
			Tag & data in the same row	Tag & data in separate storage			
Tag storage type	SRAM	eDRAM	DRAM	RRAM	DRAM		
Examples	[32,40, 52,73]	eTag [69]	[22,25,27,28,30,33,34,39,48,49, 53,54,56,58,63,67,70]	R-Cache [23]	[35]	NDC [60]	**TDRAM**
Tag check	Before MC [1]	Before MC	In MC	In RRAM	In DRAM	In DRAM	**In DRAM**
Processor die area	High [2]	High	Low	Low	Low	Low	**Low**
No Extra HW	✓	×	×	×	×	✓	✓
Tags scale with data	×	×	✓	✓	✓	✓	✓
Cond.[3] column operation	×	×	×	×	×	×	✓
Low hit/miss latency	✓	✓	×	×	×	✓	✓

Notes: [1] MC: memory controller, [2] Some prior work propose 3D-based solutions, [3]Cond.: Conditional, denotes if DRAM data banks column operation is conditional to the tag check result.

two new commands: *ActRd* and *ActWr*, which access both tag and data mats in lockstep. These commands check the tag for the block and only send data to the controller when it is needed. (3) It adds a *flush buffer* to store conflicting dirty data from write misses, which eliminates costly turnaround delays on the data bus and immediate cache line data transfer to the controller. Compared to HBM3, this new design has 8.24% total die area and 192 extra pins (10% increase) overhead.

TDRAM further improves performance by implementing an *early tag probing* mechanism, which opportunistically performs tag checks (without data access) in otherwise unused command and HM bus slots. Tag probing returns early hit-miss and status indication of a memory demand, allowing certain operations (e.g., main memory access for read demand misses) to start earlier. This mechanism also reduces the request's queue occupancy time by removing misses from the queue early, allowing other demands to proceed with fewer stalls.

TDRAM is orthogonal to many prior works focusing on improving DRAM caches performance by adding predictors, prefetchers, tag caches, modifying coherence protocols, bypass policies, and other application-specific mechanisms [22], [23], [28], [35], [69]. TDRAM is designed so these techniques can be applied to further improve caching performance. Overall, TDRAM enables a perfectly scalable HBM-based cache with a cohesive caching paradigm akin to processors' SRAM caches.

We have extensively modeled TDRAM in the gem5 simulator [51] for a full-system cycle-level timing analysis. Our evaluations using scientific and graph analytics applications with large memory footprints have shown TDRAM provides 2.65× faster tag check, 1.23× speedup, and at least 21% less energy consumption, compared to the commercial and research designs such as Intel's Cascade Lake, Alloy, BEAR [28], and NDC [60].

In this paper, we make the following contributions:

- We introduce TDRAM, a new DRAM for caching with in-DRAM tag management, for scalable HBM3-based caching.
- We extend HBM3's interface with a unidirectional *Hit-Miss bus* to transfer tag check results and metadata from DRAM to the controller, decoupling it from data transfer.
- We optimize both read and write operations to access separate tag and data banks in lockstep. The protocol *selectively streams data* to the controller only when necessary, based on tag comparison, reducing bandwidth bloat.
- We add a *flush buffer* to hold conflicting dirty data from write misses which eliminates both the costly data bus turnaround delays and immediate cache line data transfer to the controller for write requests. TDRAM opportunistically sends them to the controller when the data bus is idle.
- We propose *opportunistic early tag probing* in unused HM and command bus slots to optimize miss latencies.
- In evaluations, we demonstrate DRAM caching using existing designs cause **slowdown** while TDRAM provides 1.11× **speedup**. We show TDRAM reduces energy consumption by geo-mean of 12–21%.

II. Background and Motivation

A. Tag Management in Existing DRAM Caches

Numerous studies have investigated the management of tag and metadata (referred to collectively as *tag*) in hardware-managed DRAM caches and Table I compares some of them to TDRAM. Also, DRAM cache products, like Intel's Xeon series, offering gigabytes of DRAM cache, are available in the market. In terms of storage size, a 64 GiB block-based DRAM cache requires 3 GiB of storage for 3B tag per 64B blocks. This is far beyond the cache sizes in high-end CPUs by AMD (384 MiB in EPYC 9654P [1]) and Intel (105 MiB in Xeon Platinum 8468H [5]) today. While SRAM caches are hitting a scaling wall, tags-in-SRAM solutions (e.g., on processor die) will add to the area overhead and price [32], [40], [52], [73]. Moreover, solutions that put tags on the processor die, e.g., eTag [69], severely limits scalability of DRAM cache capacity by tying it to the tag capacity the processor chip can provide. Thus, solutions that enable highly scalable DRAM caching could be more effective when SRAM caches face limitations in scaling.

Previous studies suggest storing tags in the same cache line that data resides in [22], [25], [27], [28], [30], [33], [34], [39], [48], [49], [53], [54], [56], [58], [63], [67], [70]. For instance, in Alloy cache, instead of accessing a 64B block, 72B (plus 8B ignored) must be accessed, causing misalignment in column layout within DRAM rows and leaving unused bits that causes scalability overhead. In Intel's Xeon Series (e.g., Cascade Lake), tags are stored in the unused bits of ECC in

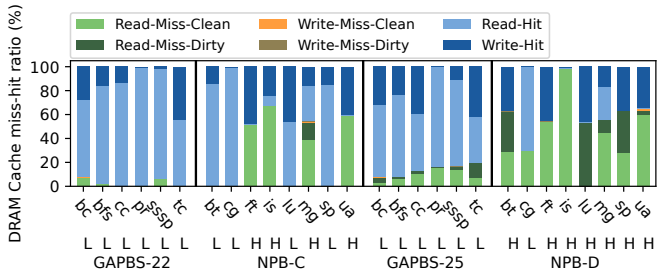

Fig. 1: The breakdown of hit and miss ratios of DRAM cache. The letters show high or low miss ratio.

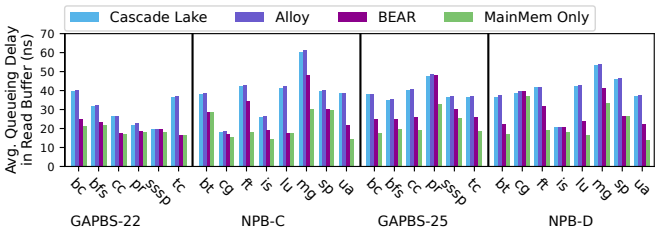

Fig. 2: The average queueing delay in the read buffer, in Intel's Cascade Lake, Alloy, and BEAR DRAM caches, compared to the system without a DRAM cache.

commodity DRAM devices [37]. However, ECC bits are not designed for this purpose. These designs depend on a DRAM read to access the tag, which can create a serialization of tag and data access (e.g., in write-hits), increasing bandwidth bloat.

Some prior work proposed to store tags in separate storage on DRAM die. R-Cache uses resistive RAM for tags [23]. Since tag access is on the critical path (i.e., data access in DRAM cache depends on the tag comparison result), the tag read and update latencies must be minimized. Resistive RAM cannot provide the required speed. Moreover, the tag and metadata are subject to frequent updates, which can wear out resistive RAM quickly. Other works suggested DRAM-based tag storage [35], [69]. These works optimize tag management and data layout in DRAM rows for set-associative caches that require multiple tag comparisons and activate tag and data regions in parallel. However, they have to delay the start of column operations till tag comparison logic finds the corresponding column, which in fact internally ties the data access to the tag access. Besides, they lack an efficient way to handle write misses to dirty cache lines, requiring a data read before write for correctness. These solutions depend on speculative mechanisms (e.g., predictors and DRAM bypassing with application-specific designs [35]), or need deep cache coherence protocol changes. For example, BEAR cache needs DRAM changes to support 80B accesses (like Alloy cache) and requires: (i) DRAM to send eviction messages to the LLC, (ii) LLC to send a DRAM cache existence indicator to the DRAM [28], [50]. This approach ties the designs of on-chip caches, memory controllers, and DRAM together, complicating industrial adoption.

B. Opportunities to Improve DRAM Cache Designs

Tag check latency is always on the critical path of servicing memory demands, affecting hit/miss latencies. Previous designs storing tags in DRAM cache lines [28], [41], [58] or in ECC bits of DRAM (e.g., Intel's Cascade Lake) require a DRAM read to retrieve the tag and data simultaneously, aiming to parallelize access and improve hit latency. Our experiments reveal their inefficiencies.

Using the gem5 simulator [51], we modeled $\frac{1}{8}$ of the Intel's Xeon Max series [24], rounded up to 64 cores and 64 GiB of HBM (as DRAM cache). We implemented BEAR, Alloy and Intel Cascade Lake DRAM cache, implementing a direct-mapped insert-on-miss cache storing tags in DRAM. We executed 28 HPC multithreaded workloads.Their memory footprints are 0.1–80 GiB, and the DRAM cache size is 8 GiB. In a full-system simulation, we employed the LoopPoint technique [61].Notably, our methodology differs from previous works, uncovering pathological pathways not observed in prior studies. More details on our methodology are in §IV.

*1) **DRAM Cache's Increased Hit Latency**:* In DRAM caches employing standard DRAM devices with tags stored within the device, the cache hit latency for LLC read misses is equivalent to DRAM read latency. For LLC writebacks (i.e., evicting dirty data from LLC), the hit latency comprises a DRAM read latency (to retrieve the tag) followed by a DRAM write (to write incoming data into the cache). Previous efforts to parallelize or decouple tag and data accesses for each memory request face challenges with LLC writebacks, including those hitting the DRAM cache [35], [58], [69], because the controller must complete the DRAM read for tag comparison before initiating the incoming data write into the cache to avoid overwriting any conflicting dirty data. Commodity DRAMs necessitate reading the entire cache line data to fetch the tag, irrespective of incoming write data size, keeping the DRAM read on the critical path for write demand and leading to access amplification (bandwidth bloat). Prior research reported that DRAM caches' access amplification can reach up to 5 accesses [37]. Figure 1 illustrates the miss ratio percentage of the DRAM cache and its breakdown in our experiments. As shown in dark blue, many workloads show significant LLC writebacks hitting the DRAM cache, impacting its hit latency. BEAR proposes to change the LLC structure, on-chip cache coherence interface, and DRAM controller to exchange information on LLC writeback existence in DRAM cache and DRAM cache evictions to avoid tag check on LLC writebacks that will hit on DRAM cache [28]. However, these are deep and complicated changes at multiple independent parts of the system and not easily adoptable by the industry.

*2) **DRAM Cache's Increased Miss Latency**:* In current DRAM caches, all read and write requests (i.e., LLC's read misses and writebacks) must undergo a DRAM read to fetch the tag. The controller handles these DRAM reads, including those for LLC writebacks, in the same read buffer. This

arrangement heightens contention in the buffer, increasing the queueing delay of all demands.

Figure 2 shows the average queueing delay of DRAM reads in existing DRAM caches, compared to a system solely equipped with main memory (no DRAM cache). As shown, the bars are higher in the DRAM cache systems compared to the system without a DRAM cache. Specifically in Cascade Lake and Alloy, because every read and write demand has to start by reading a tag in DRAM cache (even with predictors), it increases contention in the read buffer and bank conflicts when the tag reads occur. In BEAR cache, the LLC writebacks that hit in DRAM cache bypass the tag check step, reducing the queueing delay for these demands. However, the read-misses still have to go through the tag check process. This extended latency directly impacts the tag check latency for all read demands that miss in the DRAM cache, leading to a delay in fetching the missing line from the main memory. This latency is crucial for the LLC read misses, as the CPU observes their latency, which directly affects the overall system throughput. As Figure 1 shows, the number of read misses (in dark/light green) in the DRAM cache is significant. Thus, it is important to optimize the miss latency of DRAM caches for read demands.

3) ***Increased Bandwidth Bloat and Energy Consumption****:* The data fetched by the controller during tag check benefits only read demands hitting the cache or demands missing to a dirty cache line. *In cases of read/write misses to a clean line (in Cascade Lake, Alloy, BEAR) and write hits (in Cascade Lake and Alloy), the controller discards the read data immediately after tag comparison, serving no purpose.* In such cases, existing DRAM cache designs introduce data movement overheads by: (i) keeping DRAM's command bus and banks busy, and (ii) occupying data bus for unnecessary data transfers. These extra communications between the DRAM and the controller result in bandwidth bloat, wasting energy. This inefficiency worsens as the miss ratio rises. Figure 3 quantifies the relative amount of wasted data movement during the tag check process. In many applications (e.g., ft, is, mg, ua) the wasted data movement is significant. Note that Alloy and BEAR caches have 80B (64B data, 8B tag and 8B ignored) access granularity for every 64B memory demand, which increases the unuseful data movement.

Moreover, in cases where the read data is a dirty line, it is unnecessarily part of the critical path of servicing a demand. A thoughtful design could put such accesses off the critical path while ensuring correctness. Figure 1 illustrates that the memory demands not using the read data in tag access (i.e., write-hits, read-miss-cleans, write-miss-cleans) are common. Notably, write demands that miss to a dirty line in DRAM cache are very rare, indicating an opportunity to cautiously eliminate data reads in tag checks on write demands.

4) ***Goals****:* Based on our preliminary analysis, we have the following goals when constructing a cache-optimized DRAM architecture:

1. Reduce the hit latency by optimizing the tag check mechanism and write-hits, and execute tag check entirely within DRAM; 2. Reduce the miss latency, specifically for reads by reducing tag check and queueing delays; 3. Reduce the wasted data movement on write-hits, read-miss-cleans, and write-miss-cleans to save energy; and 4. Support write-miss-dirty (i.e., we cannot simply overwrite on writes) for correctness not necessarily performance since they are uncommon.

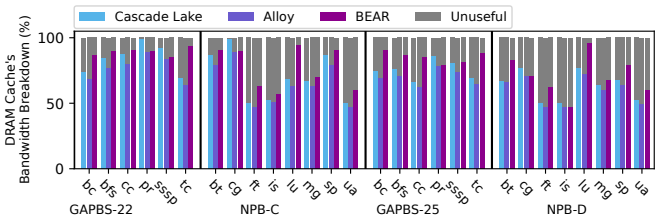

Fig. 3: Intel's Cascade Lake, Alloy, and BEAR DRAM caches' bandwidth, broken to useful and unuseful data movement, normalized to total bandwidth. In Cascade Lake and Alloy all read/write miss-cleans and write-hits, after tag comparison (which also retrieves data) the controller immediately discards the data (serving no purpose), shown as unuseful. BEAR eliminates the unuseful data movement for write-hits only. Alloy and BEAR have a longer DRAM burst than Cascade Lake, which increases the unuseful data movement.

III. Tag-enhanced DRAM Design

In this section, we describe the details of TDRAM, a new DRAM crafted for caching. TDRAM is designed in the same vein as other custom DRAMs such as RLDRAM [9]. Given the slowdown in SRAM scaling, industry is already integrating DRAMs as caches [17], [24], [64] and announced future DRAM devices specialized for caching [15]. TDRAM is a novel microarchitecture in this track.

A. HBM3 as the Basis of TDRAM

We choose HBM3 as the basis of TDRAM since the latest products (e.g., Intel's Sapphire Rapids) have used HBM as a cache for a backing store (DDR5) and HBM3 is the latest version of this technology. However, what we propose for TDRAM is perfectly applicable for other technologies, since we separated the data storage from tag and keep their underlying technology intact. When writing this paper, we envisioned TDRAM as an optional extension to current DRAM designs and we try to keep the protocol as similar to DDR as possible. Redesigning DRAM from the ground up offers full optimization but introduces challenges: (i) Established memory standards have extensive ecosystem support, including supply chains, software stacks, and architecture compatibility. Starting from scratch would need creating this ecosystem from scratch too, adding complexity and cost. (ii) An incremental approach allows for controlled risk and targeted enhancements, making it a lower-cost, lower-risk path to innovation.

HBM3 DRAMs stack multiple DRAM die into a single package, supporting up to 64 GiB capacity using 12 to 16-high stacks [47]. The peak bandwidth is 1TB/s when running

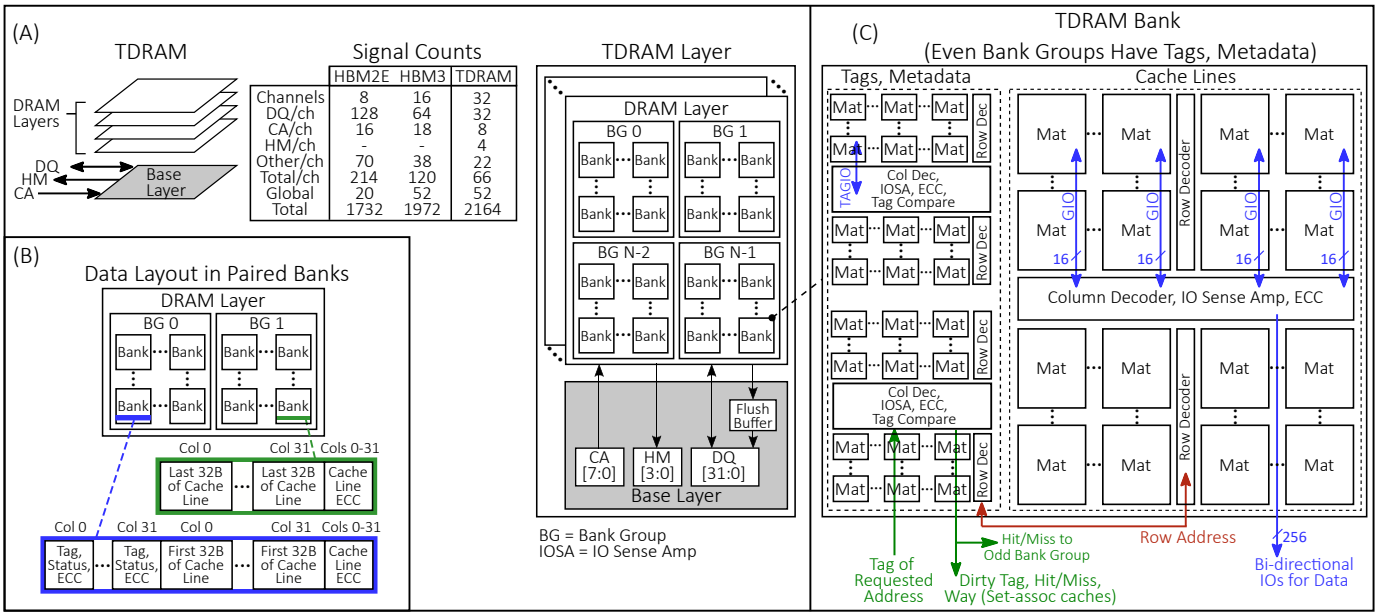

Fig. 4: TDRAM's architecture and bank organization. The logical rows and columns in the tag bank match those in the data bank, but the tag portion is divided into smaller sections, resulting in more tag mats than data mats in a bank.

at 8 Gbps across 16 independent channels with 64b data (DQ) and 10b Row command (R) and 8b Column command (C) buses. Each DQ channel is split into two 32-bit pseudo-channels (PCs) that share the same R and C buses, with each PC providing 32B access granularity [7], [12]–[14]. The wires connecting the high pin count interface between the DRAM and host (1024 DQs, 288 command/address (CA) buses, and more than 650 pins for additional channel and global functions), are implemented in TSMC's InFO or silicon (e.g., a silicon interposer) to support the high pin and trace densities required for this technology.

Regarding the protocol, a command decoder receives commands and addresses from the memory controller over a CA bus. When reading data from DRAM, an activate command provides a row address to move all bits in one row of a bank to sense amplifiers (or sense amps). A separate read command provides a column address to select a subset of the bits from the sense amps to be returned across the DQ bus. Write commands work similarly, providing data to be written.

B. TDRAM's Interface

TDRAM leverages the HBM3 interface and introduces three changes as shown in Figure 4A: (i) the R and C buses are merged into a single CA bus (i.e., like DDR DRAMs), (ii) each of the 32 PCs is converted to an independent channel with its own 8b CA bus and 32b DQ bus, and (iii) a 4b unidirectional Hit-Miss (HM) bus is added to each channel. Converting PCs to independent channels simplifies memory controller design, as each PC already has its own memory controller [11] and command/address arbitration for the shared R and C buses in HBM3 can be removed. The CA bus runs at the same speed as the DQ bus. Data transfers are protected by ECC and redundancy as is done in HBM3.

TDRAM uses the HM bus to communicate the result of tag comparison (hit/miss), status information (valid, dirty, etc.), and tag of dirty data (for main memory writeback), to the host. The HM bus runs at full data rate. Data packets are much longer than the HM bus occupancy for a single transaction. Thus, the tags and metadata can be transferred over HM bus in a number of beats (e.g., 6 for 3B metadata) without bandwidth becoming an issue. Each channel has 22 additional signals (clocks, strobes, ECC, etc.). The DRAM has 52 additional global signals (reset, IEEE1500, etc.) for a total of 2164 signals, a 9.7% increase over HBM3. The signal counts table in Figure 4A compares TDRAM's signals overhead to HBM3's. The HBM3 package has 320 unused bump sites in the area for address and data signals [7], enough to accommodate the additional 192 signals (2b CA + 4b HM = 6, per 32-bit channel) in TDRAM, allowing it to use a similar package.

C. TDRAM's Internal Architecture

*1) **Data Storage and Access Granularity**:* TDRAM uses the standard bank microarchitecture of HBM3 for data storage. CPUs from Intel and AMD operate on 64B cache lines, but HBMs are designed to provide 32B granularity, TDRAM pairs banks in different bank groups and staggers accesses to them to achieve 64B granularity at lower latencies. Figure 4B shows the layout of these paired banks. The controller views the paired banks as one logical bank and schedules accesses accordingly. To simplify the management of paired banks, the controller issues a single command (e.g., activate, read, etc.) and the logic on the base die replicates it, staggering it in time across the bank pair. Pairing banks across bank groups simplifies the controller management since the design eliminates the back-to-back accesses to the same bank group.

*2) **On-Die Tag Storage**:* TDRAM stores tags, metadata, and their ECC in separate mats on the same die as the data mats. TDRAM uses a set of small low-latency mats to provide fast tag access. These low-latency mats allow parallel tag and data lookup, with hit/miss determined before the data becomes available in the data mats. The tag mats are placed at the edge of each bank (Figure 4C). Latency for the tag mats decreases because: (1) the tag storage size is much smaller than the data storage size (3B tag per 64B cache line); (2) we use more tag mats than data mats, further reducing the size and latency of tag access. While the logical number of rows and columns in the tag bank matches that of the corresponding data bank, the tag portion is divided into smaller sections to improve the latency. Thus, there are more mats in the tag banks compared to the data banks (Figure 4C), leading to much lower latency. Due to their smaller size, these mats have shorter wordline and bitline lengths than the data mats which improves latency as shown by prior research [65]. Our design scales the tag mats by $\frac{1}{2}$ in each direction, reducing the wordline delay time and bitline charge sharing completion time. We take this ratio as a starting point based on the prior work [65]. This choice is technology-dependent and not integral to our design. We increase the number of centralized decoder and IOSAs to further improve the latency. The tag mats will have the same refresh rate as the data mats and are refreshed in parallel with data mats. This does not add performance overhead; however, it adds to the energy consumption that we take it into account in §V-C.

Alternatively, the tag arrays can be implemented on a separate die within the TDRAM stack. However, an advantage of placing tags on the same die as the cache line data is that tag storage scales with data storage. For the remainder of this paper we assume the tags are on the same die as the data.

*3) **Tag/Metadata Access and Tag Comparison**:* We add two new DRAM commands to the HBM3 command set: *activate-read (ActRd)* and *activate-write (ActWr)*. When a ActRd or ActWr command is issued to a bank, the tag mats are activated in parallel with the data mats. To avoid sending tags and metadata back to the controller, TDRAM uses on-die tag comparators implemented in the IOSA area of the tag mats to determine hit/miss status of an access. Then, the HM result is routed to the periphery of the chip for output on the HM pins. The HM result is also sent to the column decoders of the data mats where it is used to gate the column decode logic (like SALP [43]). If the tag comparison results in a hit or in a miss to a dirty cache line for read demands, the data is transferred through the DQ bus. If the tag comparison results in a miss to a clean cache line, the column decode does not happen and no data is transferred on the DQ bus, saving energy.

To improve reliability, TDRAM has separate ECCs for tag and data. ECCs for tags are analyzed and corrected if needed by on-DRAM-die circuitry as in the baseline HBM3 [7]. There are fewer bits in the tags and metadata (3B), so it can use a different algorithm than the data. For instance, it can use symbol-based Reed-Solomon encoding. For a 1PB address space, a direct-mapped TDRAM has 14-bits tag + Valid + Dirty = 16 bits which leaves 8 bits ECC to cover the 16 bits.

TDRAM uses the same tag-write mechanism as is used during DRAM writes. Like existing HBMs, there is a state machine on the base die which initializes ECC, metadata, etc. TDRAM extends the logic on base layer built-in self-test (BIST) block to also initialize the tags to zero at startup. Figures 5, 6, and 7 show the timing transactions of read and write operations in TDRAM and are discussed in §III-D.

*4) **Tag Mats Timing Values**:* TDRAM architecture minimizes the tags access latencies using small low-latency mats as discussed in §III-C2. Given the proprietary nature of DRAM timings (especially for low-volume devices like HBM), we choose to rely on publicly available data closely matching our design. For our evaluations we use timing parameters for the tag mats that are based on RLDRAM. The RLDRAM spec values (e.g., tRL=15ns and tRC=8ns) match, or are more optimistic than, our values (e.g., tRCD_TAG+tHM=15ns and tRC_TAG=12ns). Furthermore, these values and internal TDRAM timings were also correlated with prior work analyzing the use of smaller mats [65] to set a strict upper bound on the latencies. In fact, we expect in modern technology the timing parameters would be even faster. However, we do our analysis with conservative numbers; thus, we base our timing parameters on public data sheets of RLDRAM which similarly has smaller mats than normal DRAM. Table III shows a list of these timing values.

Specifically, we explain the tRL_core and tHM_int values. tHM_int = tCCD_L+tHM_detect (which is a fast equal comparison). Address comparisons are already done in DRAMs today to quickly determine if every row or column address that the DRAM receives is a repaired row or column. We set tHM_detect to 0.5ns (one 2GHz clock cycle) for the fast equal comparison based on discussions with DRAM designers. The use of tHM_int depends on tRCD since a read operation cannot occur until tRCD is met. In our design, tRCD = 12ns which is longer than tRCD_TAG+tHM_int=10ns, effectively hiding the tag access and hit/miss detection latency. tHM_int was also correlated with prior work [65], which breaks ACT-to-data delay into: 47%-sensing, 26%-address-decode, 20%-MUXing (transfer+rate-conversion), and 7%-IO. The column decode is done in parallel to sensing (tRCD) with our ActRd/ActWr commands. Finally, the I/O delay is not relevant to internal timing. A portion of the MUXing delay, the delay to move data out of the IOSA, is relevant to internal timing which was optimized in our design with smaller mats. Based on RLDRAM3, the full delay from a read command to the start of data on the DQ bus is 15ns, roughly half the latency of normal DRAM; thus, supporting our tHM_int=2.5ns. On the other hand, to ensure dirty data is not overwritten in the SA, tRL_CORE (used in write operations as illustrated in Figure 6) needs to be less than or equal to intRD-to-WR_data_Delay+tBURST/2=9ns. We performed our evaluation using tRL_CORE=tCCD_L=2ns. For both tHM_int and tRL_CORE parameters, other delays (e.g., tWL and tWR in Figure 6) dominate the latency as shown in the figure.

TABLE II: TDRAM's cache operations on different accesses.

Cache Access	CMD	DQ Activity	HM Bus	Later Actions
Read hit to clean	ActRd	Hit Data	Hit	None
Read hit to dirty		Hit Data	Hit	None
Read to invalid		None	Miss	Read main mem & fill
Read miss to clean		None	Miss	Read main mem & fill
Read miss to dirty		Dirty Data	Miss, Dirty Tag	Read main mem & fill Writeback dirty data
Write to invalid	ActWr	Wr Data	Miss	None
Write miss to clean		Wr Data	Miss	None
Write miss to dirty		Wr data	Miss, Dirty Tag	Dirty data to flush buffer
Write hit to clean		Wr Data	Hit	None
Write hit to dirty		Wr Data	Hit	None

*5) **Tag Storage Area Overhead**:* A 64 GiB direct-mapped cache can support 1 PB address space using a 14-bit tag. We assume 3B of tag and metadata for each 64B cache line. Tags are stored only in the even-numbered bank group of the pair, and the result of the tag comparison is communicated to the other (odd-numbered) bank group through an internal bus.

We estimate the die size impact of tag storage, on-die comparison, and control logic compared to a baseline HBM3 memory as follows. HBM3 stores an additional 6B of information (2B metadata and 4B parity) for every 32B of data (i.e., total column size is 38B) across 19 mats as shown by Park et al. [57]. The HBM3 die photo shows that banks (including mats, BLSAs, and Sub-WL drivers) occupy about 66% of die area. The remaining 34% of die area includes shared resources like through silicon vias (TSVs), IOSAs, per-bank group ECC, and column decoders.

We use four smaller tag mats per data mat to reduce the row cycle time by reducing the bit line and word line lengths. Son et al. show that the overhead of changing the aspect ratio by a factor of 4 is 19% [65]; however, we estimate a more pessimistic 24.3% when we scale by 1/2 in each dimension, based on our discussions with DRAM designers. Additionally, we only need tag mats in the even banks, further reducing the overhead. Thus, even banks require 24.3% additional area for the tags and the data banks occupy 66% of the die. So the overall impact on die size is 24.3% × 0.5 (only even banks) × 0.66 (area for banks) = 8.02%. We also add additional area for wire routing (e.g., to route hit/miss signals from the even bank to the odd bank), resulting in 8.24% die area impact.

D. Protocol

TDRAM's protocol is similar to traditional DRAM's, with modifications to minimize tags latency and bandwidth bloat. TDRAM has combined *ActRd* and *ActWr* commands that activate a row and read/write a column at both tag and data banks with an auto-precharge for close-page policy. These commands include the row and column addresses, bank group, bank, and tag address needed to determine cache hit/miss. Internal state machines in TDRAM handle sequencing and timing of the activate and column operations to the banks and SAs. Read/Write data appears at fixed offsets on the DQ bus from these commands, similar to modern DRAMs.

Having a single command to access tag and data banks reduces command amplification and saves energy [4], [8], [9]. Moreover, it simplifies the memory controller since the tag and data banks have the same number of rows and columns.

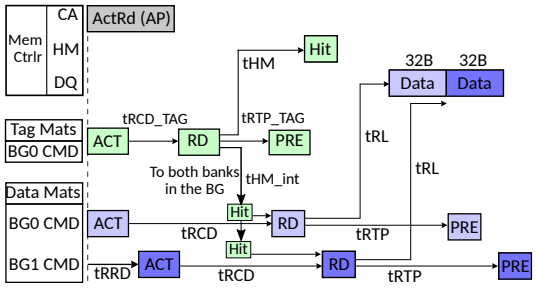

Fig. 5: Timing transactions of a read operation in TDRAM. The timing is the same for a read miss dirty.

A single address is decoded for both tag and data, allowing their banks to be activated by a single command in lockstep. TDRAM's controller can adopt any scheduling policy such as first-ready first-come first-serve (FR-FCFS). Table II shows the operations the cache performs per access.

*1) **Read Operations**:* The low-latency tag mats allow hit-miss determination to occur before cache line data is available. Figure 5 shows the timing transaction of commands involved in read operations of TDRAM. For reads, the HM response will precede the DQ bus transfer, allowing a *conditional response* based on the hit/miss result: (1) on a *read-hit*, cache line data is returned to the controller. (2) On a *read-miss-clean*, no read command is issued and no cache line data is returned to the controller. The unused DQ slot can be used to transfer data from the flush buffer (§III-D2) to the controller. (3) On a *read-miss-dirty*, the dirty data is returned to the controller with the same DQ bus timings used to return data on a cache hit. The dirty tag is returned on the HM bus along with a dirty-miss indication. When the controller receives miss indicator for read requests on the HM bus, it can initiate a backing store read to fetch the data needed (for the cache line fill and LLC response) before dirty data (if any) arrives at the controller. Early tag probing optimizes this further (§III-E).

*2) **Write Operations**:* Writes must avoid overwriting a dirty cache line with the new data on a write-miss – a rare occurrence but one that needs to be handled correctly. All existing DRAM caches have to either serialize the cache line data read (sending it back to the controller) and the incoming write data [58], or rely on complex coherence protocol modifications [28], [50] and application-specific DRAM bypass techniques [35]. TDRAM avoids these inefficiencies by adding a flush buffer that enables a general-purpose approach for writes. The flush buffer is shared among all banks, and operates similar to how a write buffer in a controller stores data to be written to the DRAMs. The flush buffer (along with additional logic to support caching) is placed on the existing base layer which already contains logic to support HBM protocol, etc. The base layer is not area limited, thus can support the needed buffer and logic.

TDRAM issues an ActWr command that initiates an internal tag and data access. Once the tag comparison result arrives to the data banks, in case of hit and miss-clean, only an internal

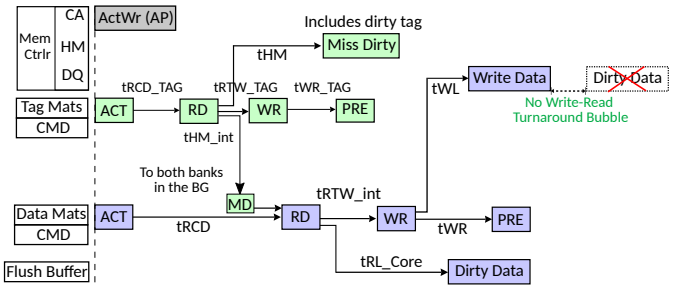

Fig. 6: Timing transactions of write operations in TDRAM.

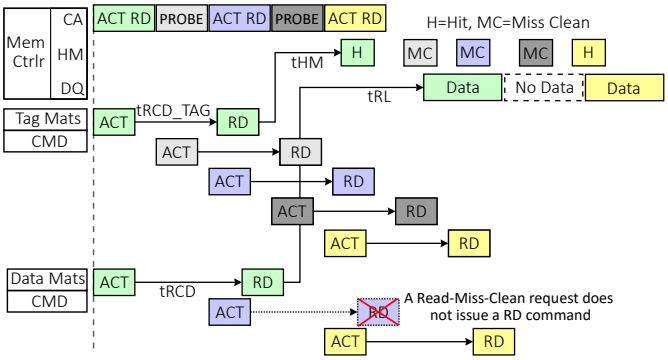

Fig. 7: The timing transaction of early tag probing. For readability, the tag results from tag to data mats are left out.

write command is issued. If the tag check indicates a miss-dirty, an internal read command followed by an internal write command are issued. Figure 6 shows the sequence of these commands. TDRAM places the dirty data into the flush buffer and then writes the new data to the DRAM. The flush buffer needs to be sized large enough such that the controller does not need to interrupt a sequence of cache writes for the sole purpose of emptying a full flush buffer, which would require insertion of a full DQ bus turnaround from write to read and then back to write direction. Since write-miss-dirty is expected to be a relatively rare event, the flush buffer can be sized modestly (e.g., 16 entries, §V-E) to eliminate virtually any need to require a forced emptying of the flush buffer. There will be a small read-to-write turnaround internally to support moving the dirty data from the DRAM bank to the flush buffer, but the much larger turnaround on the DQ bus to send the data to the controller, can be avoided. A sequence of cache writes from the controller would not experience any delay on the DQ bus due to the write-miss-dirty. Direct RDRAM uses a similar approach implementing a Write Buffer and a Write/Retire mechanism to minimize turnarounds due to resource conflicts in the DRAM core [2]. Next, we explain how TDRAM opportunistically unloads the flush buffer.

Unloading the Flush Buffer: TDRAM transmits the dirty data in the flush buffer to the controller opportunistically or on-demand, as follows: (i) when the DQ bus is idle, such as during *refresh* operations, (ii) in *read-miss-clean* accesses in which DQ is in read-state and is not used for data transfer, and (iii) if the flush buffer becomes full, the controller sends explicit read from flush buffer commands, transmitting multiple entries as a group to amortize any bus turnarounds. The controller has a global knowledge of the addresses in the flush buffer. If the DRAM cache receives a read request to any of the addresses in the flush buffer, the controller will get the data from the buffer. In case of a write demand to an address in the flush buffer, the incoming write demand will proceed into the DRAM cache and the controller removes the older data from the flush buffer. Our analysis (§V-E) has shown if we assume 16 entries for the buffer, transferring during read-miss-cleans and refresh cycles, prevents its overflow.

E. Early Tag Probing Optimization

Early tag probing tries to expedite the tag check of a demand in periods that tag-related resources are free while the data-related resoures are busy. Thus, it accesses tag banks without accessing the data banks, and returns the result to the controller on the HM bus. TDRAM's HM and command buses have unused bandwidth, because: (i) tag banks are faster than data banks; therefore, the busy time of tag banks is shorter than that of data banks, and (ii) the size of the packets transferred on HM bus (3B) is much smaller than DQ bus (64B), while both buses work at the same frequency. We use this unused bandwidth for early tag probing, in which the controller can query the status of a cache line and get an *earlier hit/miss determination* so that following actions (e.g., read from main memory for read demand misses) can begin earlier. As shown in Figure 7, in a set of pipelined read transactions, while the data bus is fully occupied by back-to-back data transfers, the CA bus and HM bus are not. Tag probing fills in the unused CA bus cycles with probing commands to perform a tag access and comparison.

1) ***Probing Mechanism:*** Tag probing accesses only the tag, transmitting results via HM bus without accessing cache line data. Figure 7 distinguishes MAIN slot commands, accessing tag and data via both HM and DQ buses, from PROBE slot commands, which solely access tags and the HM bus. While TDRAM without probing accelerates the tag check through fast tag bank access, the probing mechanism aims to reduce the tag check latency by *minimizing the queue occupancy time of the requests waiting to be scheduled for tag access.* The early tag probing lowers the contention in the read buffer, requiring fewer entries and reducing the average queueing delay. E.g., if the probing indicates a miss-clean for a read demand, the request can be removed from the read queue upon arrival of the tag check result to the controller on HM bus. Moreover, it reduces the miss latency. E.g., if a tag probe of a read demand results in a miss, the main memory access starts earlier than if the system waited for the MAIN slot for tag check, effectively removing the cache line data access from the critical path. A future MAIN slot can then be used for the cache line fill.

2) ***Selection Policy:*** Once the controller finds a PROBE slot, amongst all tag check requests that can be issued at that time (i.e., no bank conflict), it picks the ***youngest*** request to minimize the average queueing delay in the controller.

Even though the write packets can also use probing, TDRAM focuses on using these slots for read requests to reduce potential bank conflicts induced by early tag probing. Our analysis has shown that the probing-induced bank conflicts are not common (less than 1% of total demands).

IV. Evaluation Methodology

A. Modeled System for Evaluation

Many of the previous DRAM cache studies [28], [41], [42], [58], [71] rely on trace-based or functional-first simulators, which might not faithfully simulate the behavior of applications that take different paths depending on I/O or thread timings [29]. In contrast, we use an execute-in-execute full-system simulator *gem5*. Notably, prior DRAM cache research often omits full-system simulations, failing to capture OS effects. Bin et al. demonstrated that OS kernel bottlenecks can degrade memory access latency in DRAM caches [31]. We extended *gem5*'s memory system and implemented TDRAM device and DRAM cache controller [18], [19]. This device uses the timing parameters listed in Table III. §III-C4 explains the details of timing values setup for the tags in TDRAM.

We have integrated alternative DRAM cache designs into *gem5* to assess the performance of TDRAM cache: **Cascade Lake**: our evaluation baseline, a state-of-the-art commercial DRAM cache in Intel's Cascade Lake. This is a block-granule direct-mapped insert-on-miss cache storing tag and metadata in DRAM. **Alloy**: designed to reduce hit latency [58]. We chose Alloy since it has the most similar design principles to TDRAM. Alloy's 80B burst size is modeled with increased timing parameters (e.g., tBURST, etc.). **BEAR**: designed to reduce bandwidth bloat [28]. **NDC**: Native DRAM Cache is a recent proposal to provide a scalable DRAM cache while reducing the data movement due to caching [60]. Unlike TDRAM, it does not have an early tag probing mechanism. **TDRAM**: our proposed work. **Ideal**: an ideal cache which knows hit/miss and metadata status in zero latency. This sets an upper-bound for Tags-in-SRAM designs.

For a fair comparison between designs, *we use the same timing parameters for modeling the DRAM cache device unless a parameter does not apply to a DRAM cache (e.g., tRCD_TAG in Table III is only used for TDRAM).* We modeled $\frac{1}{8}$ of a target system similar to Intel's Xeon Max series [24], rounded up to 64 cores and 64 GiB of HBM (as DRAM cache) as shown in Figure 8. Table III shows the detailed parameters of the modeled system.

B. Benchmarks

Many past studies use copies of benchmarks across multiple cores, neglecting inter-thread dependencies in real-world workloads. In contrast, we leverage multithreaded HPC workloads to fully utilize simulated cores, enhancing realism. We use class C and D of NPB [20] and large synthetic graphs for the GAPBS [21] with inputs 22 and 25. The performance of same workload at different classes or inputs must not be compared together, as the workload has different execution phases in different cases. They should be seen as 28 separate

TABLE III: System Configurations

Processors		**On-chip Caches**	
Number of cores	8	Private Inst.	32 KB
Frequency	5 GHz	Private Data	512 KB
		Shared LLC	8 MB
DRAM Cache Controller		**DRAM Cache Device (TDRAM)**	
Read & Write Buffers	64 entries each	Capacity	8 GiB (8 channels)
Writeback Buffer	64 entries	Peak BW	32 GiB/s per channel
Conflicting Request Buffer	32 entries		
Sched. Policy	FR-FCFS		
Main Memory (DDR5)			
Capacity		128 GiB (2 channels)	
Peak BW		32 GiB/s per channel	
Read/Write Buffer		64 entries each	
Timing Parameters (ns) (same for all evaluated DRAM cache designs)			
Clk=2 GHz, data rate = 8Gbps, close page, RoCoRaBaCh, tBURST = 2, tRCD = 12, tRCD_WR = 6, tCCD_L = 2, tRP = 14, tRAS = 28, tCL = 18, tCWL = 7, tRRD = 2, tXAW = 16, tRL_core = 2, tRTW_int = 1, For Tag Banks in TDRAM only: tHM = 7.5, tHM_int=2.5, tRCD_TAG = 7.5, tRTP_TAG = 2.5, tRRD_TAG = 2, tWR_TAG = 1, tRTW_TAG = 1, tRC_TAG = 12			

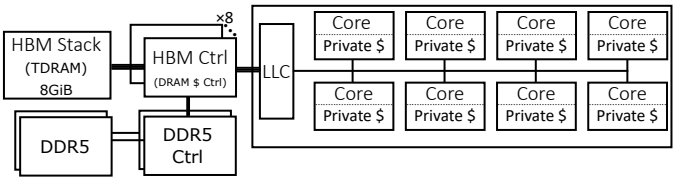

Fig. 8: The target system for evaluation. We model $\frac{1}{8}$ of a target system similar to Intel's Xeon Max series.

workloads. We employ LoopPoint, a sampling technique for multithreaded applications, tracking work progress via global loop instruction counts [16], [61]. Thus, we ensure the execution phases remain the same across different designs. Using a checkpoint for each application, we ensure that all runs start at the same system state (e.g., warmed-up SRAM and DRAM caches) for a fair comparison across different configurations.

The memory footprints of the workloads are 0.1–80 GiB, giving different miss ratios in the 8 GiB DRAM cache (Figure 1). We grouped our applications based on their miss ratios: (i) below 30% are low miss ratio, and (ii) above 50% are high miss ratio. There are no workloads in middle range.

V. Results and Discussion

A. Impact of Optimizing Tag Check Mechanism

Figure 9 compares the average tag check latency of TDRAM to Intel's Cascade Lake, Alloy, BEAR, and NDC caches. Tag check latency is the time from when the controller issues a tag read request to when the result is ready at the controller. The reported numbers are measured in the controller during simulation and include the queue occupancy time, DRAM cache tag access time, tag compare latency, bus latency, etc. The tag access time in Cascade Lake, Alloy, and BEAR designs consist a read from cache line data, while for NDC and TDRAM is an access to the separate tag banks in the DRAM cache. All designs use the same timing parameters (Table III) for cache line data access. NDC uses tag timing parameters discussed in the work [60] and TDRAM uses validated timings for tag storage discussed in §III-C4. TDRAM achieves faster hit/miss indication across all applications compared to all designs by parallelizing tag and data access and employing conditional data response. TDRAM also incorporates early

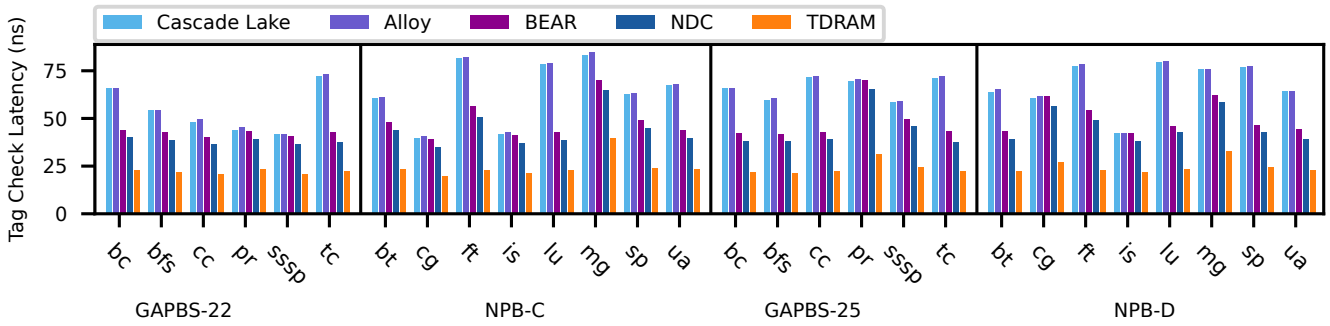

Fig. 9: Tag check latency. Lower is better. TDRAM is 2.6× , 2.65× , 2×, and 1.82× faster than Cascade Lake, Alloy, BEAR, and NDC respectively.

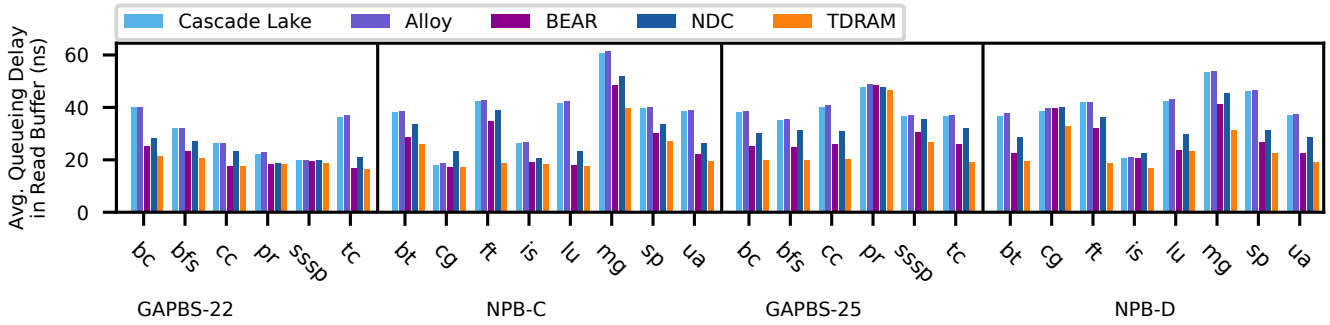

Fig. 10: Average queueing delay in read buffer. Lower is better. TDRAM's queueing delay is shorter than all the prior designs, leading to reduced tag check latency.

tag probing, further expedites this process by opportunistically performing tag checks. On a geo-mean, TDRAM's tag check is 2.61×, 2.65×, 2×, and 1.82× faster than Cascade Lake, Alloy, BEAR, and NDC, respectively. We also analyzed the tag check latency for TDRAM without early tag probing which had a result similar to NDC, hence it's omitted in the figure.

Tag check latency is on the critical path of the hit and miss latencies. For read demands that miss on DRAM cache, this latency directly impacts the LLC miss penalty, thereby affecting CPU throughput. *Improving the tag check latency accelerates the fetch of missing line from the main memory (response to the LLC), thus, reduces LLC miss penalty.* Figure 9 shows how much faster this main memory read can be issued in TDRAM.

In Cascade Lake and Alloy, the controller issues a DRAM read for tag check, placing them in the read buffer. I.e., all read and write demands compete in the same queue for DRAM read access in their tag check process. BEAR cache has the same policy but bypasses tag checks for write-hits only. NDC cache has a separate tag bank, but the tag check latency is higher than TDRAM due to the lack of early tag probing which leads to requests remaining in the controllers' buffers longer. The increased read requests causes contention in read buffer, increasing queue occupancy time and extends the process time of read demands. Figure 10 shows the average queueing delay of these read requests: the time taken since a read request enters the queue, until the read command for that demand is issued. The figure shows that the queueing delay is significantly shorter in TDRAM compared to other designs, thanks to TDRAM's early tag probing mechanism. By employing opportunistic tag probing, TDRAM allows a read request to leave the read queue as soon as the hit-miss indicator arrives on the HM bus in the case of a miss-clean, without even activating the data bank. This leads to fewer bank conflicts in the system and significantly impacts bank availability, resulting in reduced access time for future demands. For all read-misses in TDRAM, the controller request a read from main memory as soon as the miss indicator arrives on HM bus.

The benefit of early tag probing depends on the workload. Read-misses gain the most benefit from this mechanism as it accelerates main memory access with no overhead. With a smaller DRAM cache size or workloads with larger memory footprints (resulting in higher miss rates), TDRAM benefits more from early tag probing. In essence, TDRAM permits misses to occur while minimizing the miss penalty.

B. Overall Performance

Figure 11 compares the speedup of TDRAM to other designs. In all workloads TDRAM outperforms Cascade Lake, Alloy, BEAR, and NDC, providing a geo-mean speedup

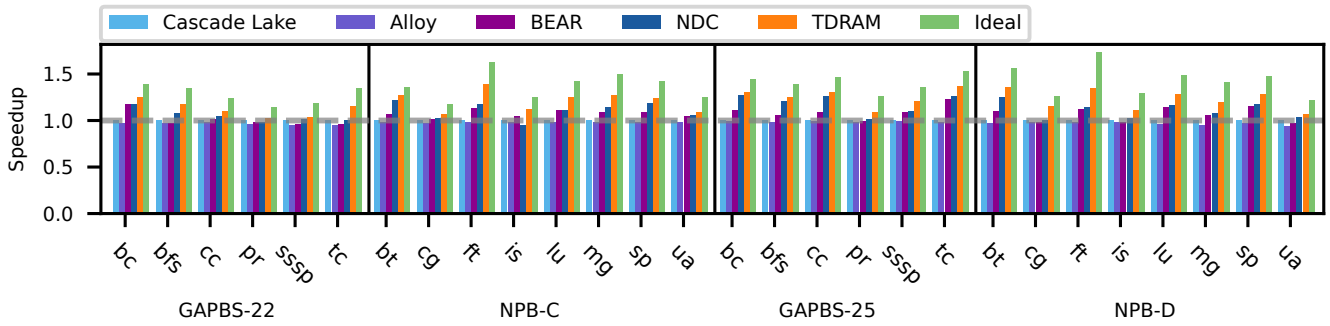

Fig. 11: System's speedup normalized to Cascade Lake. TDRAM provides 1.20×, 1.23×, 1.13×, and 1.08× speedup w.r.t. Cascade Lake, Alloy, BEAR, and NDC.

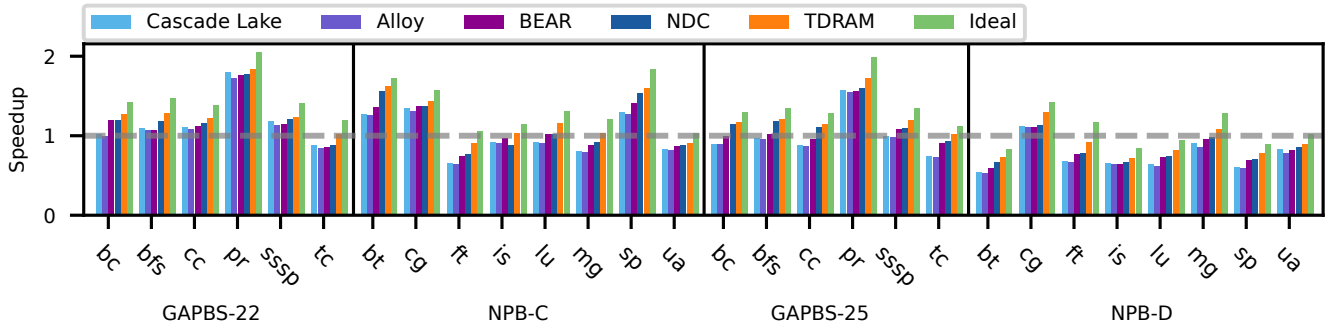

Fig. 12: Speedup normalized to system without DRAM cache. On a geo-mean, Cascade Lake, Alloy, and BEAR cause 8%, 10%, and 2% slowdown. NDC and TDRAM cause speedup of 3% and 11%, respectively.

of 1.20×, 1.23×, 1.13×, and 1.08×, respectively. We also evaluated the TDRAM without early tag probing which had a performance similar to NDC, thus we are not separately showing it in the figure. As discussed in §V-A, TDRAM effectively reduces tag check latency and queueing delay. This improvement positively impacts the hit and miss latency of the DRAM cache and the miss penalty of LLC. Consequently, the overall performance of the system is enhanced compared to existing designs, as Figure 11 shows. The ideal cache provides tag check results with zero latency, eliminating the need to endure queueing delay and DRAM access latency for tag checks, acting as a perfect Tags-in-SRAM cache. It sets a performance upper-bound for caching, and Figure 11 shows that TDRAM closely approaches this ideal, better than all the prior designs.

Figure 12 compares the speedup of all the designs to a system that has only a main memory (no DRAM cache). As the figure shows, for applications with lower miss ratios, DRAM caching can improve systems throughput. This improvement decreases as the miss ratio increases due to the miss penalty of DRAM cache that involves main memory access. Analyzing the data, Intel's Cascade Lake, Alloy, and BEAR caches cause a geo-mean slowdown of 8%, 10%, and 2%, respectively. NDC provides a geo-mean speedup of 1.03×. TDRAM increases the geo-mean speedup to 1.11×, primarily due to its reduced hit latency and miss penalty compared to all the other designs.

C. TDRAM's Energy Improvement

Prior work deines bandwidth bloat factor as: total number of bytes moved divided by total useful bytes moved [28]. Table IV shows the geo-mean bandwidth bloat across low and high miss ratio workloads. The table also shows the maximum reduction TDRAM achieves. TDRAM reduces the bandwidth bloat by a geo-mean of 39.9%, 25.1%, and 19.85% compared to Alloy, Cascade Lake, and BEAR, respectively. NDC's bandwidth bloat is similar to the TDRAM's since they both move the same amount of data for a demand.

TABLE IV: Bandwidth Bloat Factor

Design	**Low Miss Ratio**	**High Miss Ratio**	
Cascade Lake	1.35	2.75	
Alloy	1.68	3.43	
BEAR	1.41	2.40	
NDC	1.13	2.06	
TDRAM	1.13	2.06	
TDRAM Reductions			
Design	Low Miss Ratio	High Miss Ratio	Max
W.r.t. Cascade Lake	16.3%	25.1%	39.51%
W.r.t. Alloy	32.7%	39.9%	51.61%
W.r.t. BEAR	14.16%	19.85%	23.79%
W.r.t. NDC	0%	0%	0%

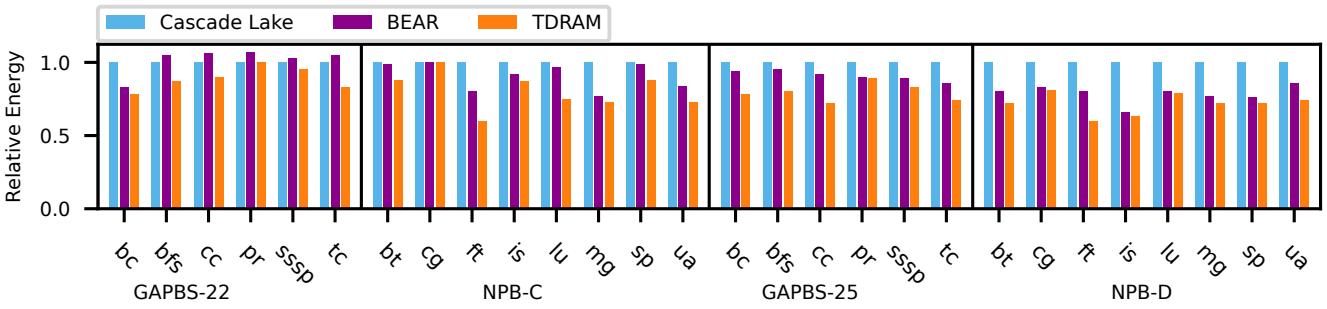

Fig. 13: Relative energy consumption of TDRAM and BEAR, normalized to Cascade Lake. Lower is better. Alloy is not shown as it is much higher than Cascade Lake. NDC is omitted as its relative energy consumption is similar to TDRAM.

To analyze the energy consumption, we developed an HBM3 power model using HBM2 power data in [55] and scaled it for HBM3 speeds and timings (Table III). Processor interface power is calculated from our validated HBM3 PHY design. Compared to a standard HBM3 DRAM, TDRAM's power is increased to account for on-die tag storage and associated operations, and both DRAM cache and processor interface power are increased for the additional signals and HM buses and associated logic. Figure 13 shows the relative energy consumption (power × runtime) of TDRAM and BEAR caches, normalized to Cascade Lake as the baseline. We are not showing Alloy's energy consumption as it is much higher than Cascade Lake. Moreover, the energy consumption of NDC and TDRAM is comparable, as both systems move the same amount of data, as shown in Table IV. We anticipate a slight increase in partial energy consumption for NDC due to its larger tag MATs and additional data column operations for cases where data transfer is unnecessary (e.g., read-miss-cleans), which TDRAM eliminates. These additional operations contribute negligibly to the total energy consumption, which is primarily driven by the amount of data transferred–identical in both designs. Thus, the overall relative energy consumption of NDC and TDRAM remains the same.

On geo-mean, TDRAM's energy savings compared to Cascade Lake and BEAR is 21% and 12%, respectively. This saving is primarily due to reducing bandwidth bloat in the TDRAM's protocol. Applications with more write-hits or read/write miss-cleans (e.g., ft and is) show more energy savings with TDRAM's compared to Cascade Lake. Applications with more miss-cleans (e.g., ft, ua) show more energy savings with TDRAM compared to BEAR. BEAR like Alloy has an access granularity of 80B for every 64B demand, which consumes more energy even on hits. As the bandwidth bloat increases, more energy is consumed since more data is transferred. TDRAM eliminates unnecessary data transfers in its protocol, saving energy while, servicing the same number of memory demands as the other designs.

TDRAM's Power Change *:* To access both tag and data banks in parallel, TDRAM activates all involved banks simultaneously. On the other hand, TDRAM compares tags in DRAM to selectively send data to the controller only when needed, significantly reducing bandwidth bloat (Table IV). Prior works have shown HBM2 spends 62.6% of power in moving data between the DRAM core and the controller [10]. This makes designs like Alloy and BEAR more power-inefficient due to their 80B data transfer per 64B memory demand. In contrast, TDRAM saves power system-wide by streaming 64B data to the controller only when necessary, based on in-DRAM tag comparison. Extra activates in TDRAM increase power slightly, but it is small compared to data transfer. TDRAM's overall power savings compared to Cascade Lake and BEAR are 7% and 5%, respectively. In the DRAM itself, there will be higher power consumption than normal DRAMs since more bits are accessed in parallel. However, spare vias in the HBM package can be used for more power delivery, and the increase in power should not cause significant heat problems. As mentioned above, this increase in the internal DRAM power consumption is much less than what is saved by selective data streams on data bus.

D. Performance Impact of Predictors and Prefetchers

We designed TDRAM orthogonal to the prefetchers and predictors, i.e., any existing or future prefetchers/predictors can be used with TDRAM to further improve the performance. We evaluated the impact of using a MAP-I [58] predictor on DRAM cache performance. The results showed predictors have a minor impact on overall performance, with an overall speedup of 1.03-1.04× compared to caches without predictors. However, predictors cannot guarantee a predicted miss isn't a hit, causing inefficiencies since the line's state (dirty or clean) is only known after reading the tags. For writes, predictors must always read data to avoid overwriting potential dirty lines. TDRAM in its ActWr protocol ensures this by reading the dirty data (if any) into flush buffer, before writing incoming data. For reads, predictors can accelerate main memory accesses on misses but must read cache data before cache fill to avoid overwriting dirty lines. TDRAM with early tag probing accelerates main memory accesses on read misses deterministically rather than through speculation. Predictors can be detrimental to DRAM cache performance by increasing bandwidth bloat through unnecessary data transfer.

In contrast, early tag probing in TDRAM only accesses tag storage during unused slots in the HM and command buses, minimizing timing impact and avoiding bandwidth bloat.

Our preliminary analysis shows incremental performance gain from prefetchers as well. The reason is prefetchers introduce interference with demand accesses and consume excessive bandwidth. DRAM caches, due to their larger size and higher latency compared to on-chip caches, are particularly sensitive to resource utilization including bandwidth and buffers. Depending on the prefetch granularity (page, block, etc), it adds tail latency to critical demands response time increasing the hit latency. Moreover, prefetchers can cause unnecessary data movement if gaining low accuracy per application, increasing bandwidth bloat and energy consumption. All these aspects contribute to the incremental performance improvement from prefetchers. Future work could explore specialized predictors and prefetchers for DRAM caches.

E. Flush Buffer Size Sensitivity Analysis

We assessed the sensitivity to the flush buffer size with 8, 16, 32, and 64 entries. The results showed the flush buffer consistently avoids becoming full, preventing TDRAM stalls, except for lu in NPB-D with buffer size of 8. In this case, TDRAM stalled only 13 times, resulting in negligible performance overhead. Most applications rely on read-miss-clean accesses to unload the flush buffer. Notably, lu and bc highly utilized the refresh cycles for unloading. This data confirms the effectiveness of TDRAM's opportunistic behavior in minimizing data transfer overhead. The flush buffer's average occupancy was 5 entries, with a maximum of 12 entries. Setting the buffer size to 16 prevents TDRAM stalls. Thus, the overhead of flush buffer is minimal.

F. Set-Associative TDRAM

TDRAM applies equally well to direct-mapped and set-associative caches. In Figure 4, if pairs of bank groups (e.g., 0 and 1, 2 and 3, etc.) form two ways of a set, tag comparisons can be performed in parallel if each way has its own comparator. A signal from the matching way is sent to the internal control logic to select the proper column in the data mats. Implementations without in-DRAM tag comparators send all tags in the set to the controller, and the controller subsequently sends a request for the proper column to the DRAM, incurring extra latency and energy consumption [48].

Set-associativity helps applications with high miss conflicts. However, our analysis showed the tested HPC workloads have negligible miss conflicts on DRAM cache. Thus, they did not gain significant performance improvement from set-associativity compared to direct-mapped cache. Our results show direct-mapped and 2,4,8,16 ways set-associative caches have similar speedup (over a system with main memory only).

VI. Related Work

Table I and §II-A provides a comparison of TDRAM with prior work. Loh and Hill [48] proposed one of the earliest block-based DRAM cache where tag and data access were stored in the same row with a MissMap to avoid accessing the DRAM cache on predicted misses. Alloy [58] reduces latency by streaming data and tags together in a single burst. Furthermore, they introduced a memory access predictor, which incurred less overhead compared to the MissMap technique. Retagger [25] uses tags in the controller to mitigate the DRAM row buffer miss cost. RedCache [22] adapts at runtime to start and stop caching for individual blocks. While these works have explored different approaches for storing tags and data in DRAM caches, they all require the tags to be moved to the controller for tag comparison and checks to be performed. R-Cache proposed to use RRAM memory for on-die tag storage [23]. Due to longer latency of RRAM compared to DRAM, it can extend the tag check latency, exacerbating the hit and miss latencies of DRAM cache. In contrast, our work modifies the DRAM microarchitecture to enable tag checks to be performed inside the DRAM, thereby reducing the data movement overhead and improving overall cache efficiency.

The Footprint Cache [42] and Unison Cache [41] blend block and page-based designs to lower off-chip traffic. This coarse-grain tracking leads to bandwidth waste and poor utilization of cache capacity in contrast to block-based caches like TDRAM. Several works [44], [45], [72] combine software and hardware techniques for DRAM caching. Also, Hong et al. proposed a DRAM cache specifically for GPUs working with storage-class memories [38]. We envision potential software/OS integration benefits for TDRAM, as well as GPU-specific changes which are directions we plan to explore.

Stockdale et al. leverages HBM's embedded logic die for cache management [66] by enhancing the base HBM DRAM layer with a cache result signal and reserving one pseudo-channel for tags. The eTag DRAM cache uses eDRAM storage on the processor die, with tag comparison preceding DRAM cache access [69], but eTag cannot scale with increased off-chip capacity as eDRAM size limits the data cache capacity. Hameed et al. proposed a DRAM cache with a separate tag and data storage that relies on a predictor and a Data-Absence-Table [35]. These speculation-based designs are orthogonal to TDRAM. TDRAM minimizes amplification, using the HM bus for cache outcomes and tag transfers, and employs tag probing to mitigate read miss impact. TDRAM puts tags and data on each channel which scales with cache capacity.

A recent work has introduced NDC which similar to TDRAM extends DRAM to store tags and performs tag checks inside the DRAM [60]. while both TDRAM and NDC enable HBM-like devices to be used as a cache, their nature is different in the following ways. While TDRAM extends the existing DRAM protocol and interface, NDC modifies the DRAM sensing circuit, protocol, and interface. TDRAM builds on existing fast DRAM for tag management (e.g., RLDRAM) whereas NDC introduces a new CAM-like DRAM structure. TDRAM keeps the data and metadata storage technology intact, where NDC introduces changes in the structure of CAM for tag matching and storage support.

Main differences in their protocols are as follows. In TDRAM hit/miss status is determined during data bank ac-

tivation, thus enabling *conditional RD/WR command to the data bank according to the hit/miss status*. For instance, in case of a miss-clean for a read request, TDRAM does not issue a RD command. While in NDC, the command is always issued and hit/miss status is always determined during column operation. In other words, TDRAM skips column operation if not necessary which *saves latency and energy consumption*. Besides, TDRAM uses idle states of the data bus (e.g., refresh cycles and in read-miss-clean requests that do not send back data) to actively empty the flush buffer. Whereas in NDC, they require a specific new command (RES) to empty their victim buffer. This mandates a bubble on the data bus for NDC since the write request data and the data from the victim buffer have different directions on the data bus. In addition, TDRAM proposes the early probing that deterministically identifies the hit/miss status of a request with zero overhead, since it is performed in otherwise unused slots of buses. NDC, by tying the hit/miss indication to the RD/WR commands, is unable to leverage this technique. Our results show that early tag probing can improve tag check latency up-to 70% on large workloads with high miss rate.

Finally, by having a HM bus, TDRAM does not need an extra 2 beats (1 cycle) to send the tag from DRAM to the controller. Thus, TDRAM saves 1 cycle every request and the power to transmit the tag on hits and miss cleans (the common cases). We provide a detailed cost breakdown of the extra pins in the table in Figure 4A. Note that NDC provides the overhead for a single channel whereas we show the overhead for a whole 64 GiB stack. NDCs single-bit bus is similar to [66]. Also, the evaluation of TDRAM focuses on large applications (up to 80 GiB footprint) and realistic-sized DRAM caches (8 GiB), whereas NDC uses much smaller workloads (inputs not given) and a smaller cache (1 GiB).

Another group of prior works, TL-DRAM [46], LISA [26], CROW [36], FIGARO [68], propose in-DRAM caching where the DRAM is heterogeneous by having a fast (or near) segment (or row) and a slow (or far) segment (or row) and the fast subcomponent caches for the slow subcomponent internally. These work fundamentally differ from TDRAM since TDRAM caches for an external independent backing store and the fast part only serves for tag and metadata storage.

Conclusion

In this paper we introduce TDRAM, a tag-enhanced energy-efficient DRAM for caching, to optimize caches hit and miss latencies. We showed TDRAM's 1.2× speedup and 21% energy saving over commercial designs. These improvements highlight TDRAMs potential to enhance memory performance and energy efficiency, making it a promising solution for modern computing systems that demand higher performance and reduced energy consumption. Further exploration of TDRAMs integration into existing memory hierarchies could unlock benefits for diverse architectures.

Acknowledgment

We thank the anonymous reviewers and the members of DArchR Lab at University of California, Davis for providing us with their feedback and comments which improved the paper tremendously. This work was sponsored in part by National Science Foundation grant numbers 1925724 and 2144883.

References

[1] "Amd epyc 9654p." [Online]. Available: https://www.techpowerup.com/cpu-specs/epyc-9654p.c2934
[2] "Direct rdram." [Online]. Available: https://datasheetspdf.com/pdf-file/623377/HynixSemiconductor/HY5R288HC745/1/
[3] "Iedm 2022: Did we just witness the death of sram?" [Online]. Available: https://fuse.wikichip.org/news/7343/iedm-2022-did-we-just-witness-the-death-of-sram/
[4] "Intel. rldram ii and rldram 3 features." [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/710283/17-0/rldram-ii-and-rldram-3-features.html
[5] "Intel xeon platinum 8468h." [Online]. Available: https://www.itcreations.com/product/140851
[6] "Intel's cascade lake: 2nd generation intel xeon scalable processors." [Online]. Available: https://www.intel.com/content/www/us/en/products/platforms/details/cascade-lake.html
[7] "Jedec. high bandwidth memory dram (hbm3), jedec standard jesd238, jan 2022." [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd238a
[8] "Micron. async/page/burst cellularram 1.0 memory mt45w2mw16bgb." [Online]. Available: https://www.digchip.com/datasheets/parts/datasheet/301/MT45W2MW16BGB-701_IT-pdf.php
[9] "Micron rldram 3 specifications." [Online]. Available: https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/1,-d-,125gb_x18_x36_rldram3.pdf
[10] "Power consumption in hbm." [Online]. Available: https://semiengineering.com/where-power-is-spent-in-hbm/
[11] "Rambus inc. hbm3 controller." [Online]. Available: https://www.rambus.com/interface-ip/hbm/hbm3-controller/
[12] "Rambus inc. hbm3: Everything you need to know." [Online]. Available: https://www.rambus.com/blogs/hbm3-everything-you-need-to-know/,Oct2023.
[13] "Samsung. hbm3 icebolt: Powering the next frontier." [Online]. Available: https://www.semiconductor.samsung.com/us/dram/hbm/hbm3-icebolt/
[14] "Synopsys. what is high bandwidth memory 3 (hbm3)?" [Online]. Available: https://www.synopsys.com/glossary/what-is-high-bandwitdth-memory-3.html
[15] "[tech day 2022] dram solutions to advance data intelligence." [Online]. Available: https://semiconductor.samsung.com/news-events/tech-blog/dram-solutions-to-advance-data-intelligence/
[16] A. R. Alameldeen and D. A. Wood, "Ipc considered harmful for multiprocessor workloads," *IEEE Micro*, vol. 26, no. 4, pp. 8–17, 2006.
[17] M. Arafa, B. Fahim, S. Kottapalli, A. Kumar, L. P. Looi, S. Mandava, A. Rudoff, I. M. Steiner, B. Valentine, G. Vedaraman, and S. Vora, "Cascade lake: Next generation intel xeon scalable processor," *IEEE Micro*, vol. 39, no. 2, pp. 29–36, 2019.
[18] M. Babaie, A. Akram, and J. Lowe-Power, "Enabling design space exploration of dram caches for emerging memory systems," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2023, pp. 340–342.
[19] M. Babaie, A. Akram, and J. Lowe-Power, "Enabling design space exploration of dram caches for emerging memory systems," *arXiv preprint arXiv:2303.13029*, 2023.
[20] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The nas parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.
[21] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.
[22] P. Behnam and M. N. Bojnordi, "Adaptively reduced dram caching for energy-efficient high bandwidth memory," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2675–2686, 2022.

[23] P. Behnam, A. P. Chowdhury, and M. N. Bojnordi, "R-cache: A highly set-associative in-package cache using memristive arrays," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 423–430.

[24] A. Biswas and S. Kottapalli, "Next-Gen Intel Xeon CPU - Sapphire Rapids," in *Hot Chips 33*, 2021.

[25] M. N. Bojnordi and F. Nasrullah, "Retagger: An efficient controller for dram cache architectures," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[26] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-cost inter-linked subarrays (lisa): Enabling fast inter-subarray data movement in dram," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 568–580.

[27] 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 *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 1–12.

[28] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3S, pp. 198–210, 2015.

[29] L. Eeckhout, "Computer architecture performance evaluation methods," *Synthesis Lectures on Computer Architecture*, vol. 5, no. 1, pp. 1–145, 2010.

[30] 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 *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 89–92.

[31] B. Gao, H.-W. Tee, A. Sanaee, S. B. Jun, and D. Jevdjic, "Os-level implications of using dram caches in memory disaggregation," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 153–155.

[32] 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 *40th Annual IEEE/ACM international symposium on microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 134–145.

[33] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-modal dram cache: Improving hit rate, hit latency and bandwidth," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 38–50.

[34] F. Hameed, L. Bauer, and J. Henkel, "Simultaneously optimizing dram cache hit latency and miss rate via novel set mapping policies," in *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. IEEE, 2013, pp. 1–10.

[35] F. Hameed, A. A. Khan, and J. Castrillon, "Improving the performance of block-based dram caches via tag-data decoupling," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1914–1927, 2020.

[36] H. Hassan, M. Patel, J. S. Kim, A. G. Yağlıkçı, N. Vijaykumar, N. M. Ghiasi, S. Ghose, and O. Mutlu, "Crow: A low-cost substrate for improving dram performance," *Energy Efficiency, and Reliability. In ISCA*, 2019.

[37] M. Hildebrand, J. T. Angeles, J. Lowe-Power, and V. Akella, "A case against hardware managed dram caches for nvram based systems," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 194–204.

[38] J. Hong, S. Cho, G. Park, W. Yang, Y.-H. Gong, and G. Kim, "Bandwidth-effective dram cache for gpu s with storage-class memory," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 139–155.

[39] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014, pp. 51–60.

[40] 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 *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2011, pp. 1–4.

[41] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 25–37.

[42] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 404–415, 2013.

[43] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 368–379.

[44] Y. Kim, H. Kim, and W. J. Song, "Nomad: Enabling non-blocking os-managed dram cache via tag-data decoupling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 193–205.

[45] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir, "Chameleon: A dynamically reconfigurable heterogeneous memory system," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 533–545.

[46] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency dram: A low latency and low cost dram architecture," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 615–626.

[47] J. Lee and et al., "A 48-gb 16-high 1280-gb/s hbm3e dram with all-around power tsv and a 6-phase rdqs scheme for tsv area optimization," International Solid State Circuits Conference (ISSCC) 2024.

[48] G. Loh and M. D. Hill, "Supporting very large dram caches with compound-access scheduling and missmap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, 2012.

[49] 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 *3rd Workshop on SoCs, Heterogeneous Architectures and Workloads*, vol. 20, 2012, p. 12.

[50] J. Lowe-Power, *On Heterogeneous Compute and Memory Systems*. The University of Wisconsin-Madison, 2017.

[51] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," 2020.

[52] 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 *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 2009, pp. 262–274.

[53] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 126–136.

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

[55] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 41–54.

[56] 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 *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, 2012, pp. 83–92.

[57] M.-J. Park, J. Lee, K. Cho, J. Park, J. Moon, S.-H. Lee, T.-K. Kim, S. Oh, S. Choi, Y. Choi *et al.*, "A 192-gb 12-high 896-gb/s hbm3 dram with a tsv auto-calibration scheme and machine-learning-based layout optimization," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 256–269, 2022.

[58] 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 *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 235–246.

[59] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter, "Hemem: Scalable tiered memory management for big data applications and real nvm," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 392–407.

[60] Y. Ryu, Y. Kim, G. Jung, J. H. Ahn, and J. Kim, "Native dram cache: Re-architecting dram as a large-scale cache for data centers," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1144–1156.

[61] A. Sabu, H. Patil, W. Heirman, and T. E. Carlson, "Looppoint: Checkpoint-driven sampled simulation for multi-threaded applications," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 604–618.

[62] D. D. Sharma, "System on a package innovations with universal chiplet interconnect express (ucie) interconnect," *IEEE Micro*, vol. 43, no. 2, pp. 76–85, 2023.

[63] J. Sim, G. H. Loh, V. Sridharan, and M. O'Connor, "Resilient die-stacked dram caches," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 416–427, 2013.

[64] A. Sodani, R. Gramunt, J. Corbal, H.-s. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, mar 2016. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7453080

[65] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing memory access latency with asymmetric dram bank organizations," in *Proceedings of the 40th annual international symposium on computer architecture*, 2013, pp. 380–391.

[66] T. Stocksdale, M.-T. Chang, H. Zheng, and F. Mueller, "Architecting hbm as a high bandwidth, high capacity, self-managed last-level cache," in *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, 2017, pp. 31–36.

[67] 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*, pp. 36–47, 2009.

[68] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi *et al.*, "Figaro: Improving system performance via fine-grained in-dram data relocation and caching," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 313–328.

[69] K.-H. Yang, H.-J. Tsai, C.-Y. Li, P. Jendra, M.-F. Chang, and T.-F. Chen, "etag: Tag-comparison in memory to achieve direct data access based on edram to improve energy efficiency of dram cache," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 4, pp. 858–868, 2016.

[70] 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 *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 187–192.

[71] V. Young, C. Chou, A. Jaleel, and M. Qureshi, "Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 328–339.

[72] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient dram caching via software/hardware cooperation," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 1–14.

[73] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, "Exploring dram cache architectures for cmp server platforms," in *2007 25th International Conference on Computer Design*. IEEE, 2007, pp. 55–62.