Data Convection: A GPU-Driven Case Study for Thermal-Aware Data Placement in 3D DRAMs

SOHEIL KHADIRSHARBIYANI, Pennsylvania State University, USA
JAGADISH KOTRA, AMD Research, USA
KARTHIK RAO, Advanced Micro Devices Inc., USA
MAHMUT TAYLAN KANDEMIR, Pennsylvania State University, USA

Stacked DRAMs have been studied, evaluated in multiple scenarios, and even productized in the last decade. The large available bandwidth they offer make them an attractive choice, particularly, in high-performance computing (HPC) environments. Consequently, many prior research efforts have studied and evaluated 3D stacked DRAM-based designs. Despite offering high bandwidth, stacked DRAMs are severely constrained by the overall memory capacity offered.

In this paper, we study and evaluate integrating stacked DRAM on top of a GPU in a 3D manner which in tandem with the 2.5D stacked DRAM increases the capacity and the bandwidth without increasing the package size. This integration of 3D stacked DRAMs aids in satisfying the capacity requirements of emerging workloads like deep learning. Though this vertical 3D integration of stacked DRAMs also increases the total available bandwidth, we observe that the bandwidth offered by these 3D stacked DRAMs is severely limited by the heat generated on the GPU. Based on our experiments on a cycle-level simulator, we make a key observation that the sections of the 3D stacked DRAM that are closer to the GPU have lower retention-times compared to the farther layers of stacked DRAM. This thermal-induced variable retention-times causes certain sections of 3D stacked DRAM to be refreshed more frequently compared to the others, thereby resulting in thermal-induced NUMA paradigms.

To alleviate such thermal-induced NUMA behavior, we propose and experimentally evaluate three different incarnations of Data Convection, i.e., Intra-layer, Inter-layer, and Intra + Inter-layer, that aim at placing the most-frequently accessed data in a thermal-induced retention-aware fashion, taking into account both bank-level and channel-level parallelism. Our evaluations on a cycle-level GPU simulator indicate that, in a multi-application scenario, our Intra-layer, Inter-layer and Intra + Inter-layer algorithms improve the overall performance by 1.8%, 11.7%, and 14.4%, respectively, over a baseline that already encompasses 3D+2.5D stacked DRAMs.

CCS Concepts: • **Hardware** → **3D integrated circuits**; **Temperature optimization**; **Chip-level power issues**; • **General and reference** → **General conference proceedings**; **Performance**; • **Computing methodologies** → *Graphics processors*; • **Software and its engineering** → **Main memory**.

Additional Key Words and Phrases: GPU; 3D Stacked DRAM; Thermal Issues; Data Placement

ACM Reference Format:
Soheil Khadirsharbiyani, Jagadish Kotra, Karthik Rao, and Mahmut Taylan Kandemir. 2022. Data Convection: A GPU-Driven Case Study for Thermal-Aware Data Placement in 3D DRAMs. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 1, Article 07 (March 2022), 25 pages. https://doi.org/10.1145/3508027

2476-1249/2022/3-ART07 $15.00
https://doi.org/10.1145/3508027

1 INTRODUCTION

In the past two decades, 3D integration of silicon dice has revolutionized the semiconductor industry. Among several contributing factors, the key breakthrough has perhaps been Through-Silicon-Via (TSV) technology [49]. Since this technology enables stacking different types of dice, multiple research groups have proposed innovative combinations of compute (CPUs, GPUs, and atomic ALU units) and memory (DRAM and SRAM) components [13, 41, 56]. Most practical and commercial use cases for such 3D stacking of memory include High-Bandwidth Memory (HBM) [26, 27], Hybrid Memory Cube (HMC) [28], and WideIO [30].

3D stacking of memory dice provides an order of magnitude improvement in bandwidth (256GB/s for HBM v2 vs. 25.6GB/s for DDR4 vs 28GB/s for GDDR5). The reduction in wire length translates to lower power consumption as well (10.66GB/sec/Watt for GDDR5 vs. ~35 GB/sec/Watt for HBM) [27]. While leading processor manufacturers market products with HBM-integrated GPUs in a 2.5D fashion through an interposer [6, 53], state-of-the-art proposals stack multiple memory dice directly on top of a CPU or GPU die [15, 41, 42, 47, 63]. Though HBMs offer higher bandwidth, the requirement for more device memory capacity in GPUs limits the benefit in GPUs as emerging workloads like Deep Learning require higher capacities to accommodate larger models in the GPU device memory [58]. As neither the die size nor the number of HBM dies are increasing significantly, emerging workloads are severely bottlenecked by the device memory capacity, resulting in excessive data movement between the host and device memories [58]. One needs to increase the number of HBM modules to increase the memory capacity which would increase the package size significantly [62].

To address these challenges, in addition to the 2.5D stacked DRAM (HBM), integrating 3D stacked DRAM vertically on top of a GPU is an option that is studied in recent works [62]. Together, 3D+2.5D stacked DRAMs would not only increase the overall memory capacity but they would also increase the available memory bandwidth. In particular, bandwidth-intensive applications show significant improvements in performance [11] [62]. However, in this configuration, we observed that the overall bandwidth that can be leveraged from the vertically stacked 3D DRAM is severely limited by the *heat* generated on the GPU. Additionally, we also observed that the heat generated from GPU affects the DRAM cell retention-times differently as different regions of the 3D stacked DRAM are subjected to varying temperatures causing the volatile DRAM cell capacitors to leak charge at different rates. This variability in temperatures is due to the following reasons:

- As the heat dissipated from compute die escapes vertically, layers of DRAM closer to the compute die are subjected to higher temperatures compared to ones that are farther from the compute die, resulting in Inter-layer thermal gradient.
- Since a GPU die contains multiple SRAM hardware structures like caches, the heat dissipated varies across different parts of the GPU, and hence, DRAM cells within a layer of 3D stacked DRAM are subjected to different temperatures.

Such variability in temperature causes variable retention times, which necessitate refreshing DRAM cells at different frequencies to maintain data integrity, thereby making DRAM refresh a performance bottleneck. This problem is more pronounced in multi-application scenarios where GPU is virtualized. Note that hardware virtualization has become a norm in Cloud Computing, where hardware resource sharing is inevitable. Cloud service providers promise a Quality-of-Service (QoS) to the end-user which in turn determines the monetary cost for the end-user. In such scenarios maintaining the promised QoS is a critical challenge due to hardware sharing. Even though cloud providers isolate the execution of applications by isolating the shared hardware resources, thermal coupling still plays a huge role in causing interference across applications. For example, a memory-sensitive application, co-residing with a compute-intensive application on a GPU, can still experience thermal problems because of the heat dissipated from the computational-intensive application. Prior works have observed this phenomenon in CPUs as well as GPUs [55, 57].

Although modern-day system software such as OS and hypervisor [36, 40] have non-uniform access latency (NUMA) aware data placement (allocation/migration) support, their NUMA cognizance is limited to *physical proximity*. The traditional NUMA paradigms were strongly coupled to the distance of data from the compute elements. In other words, memories that are closer to the compute elements incur lower access latencies compared to the ones farther away. In this paper, we demonstrate a new *thermal-induced non-uniformity* in access latencies across different sections of the two stacked DRAMs. Unlike traditional NUMA paradigms, we observe that physical proximity has a "reversible" effect on the memory access latencies: *sections of a vertically stacked DRAM that are closer to the compute elements (GPUs) have higher access latencies compared to the ones that are farther from the compute elements and the ones placed on 2.5D DRAM.* Based on this key observation, we expose a novel avenue of NUMA-latencies in different sections of 3D stacked DRAM. To that end, we propose "Data Convection", a thermal-induced NUMA-aware data placement that targets GPUs. Specifically, we make the following main **contributions** in this paper:

- We are the first to demonstrate a novel thermal-induced NUMA paradigm in a system where 3D stacked DRAMs are mounted vertically on GPUs.
- We demonstrate, using detailed cycle-level simulation, that such NUMA behavior is prevalent both within and across the layers of a 3D stacked DRAM.
- We identify that modern GPU runtime systems are agnostic to such thermal-induced NUMA behavior, and demonstrate that the bandwidth lost due to such retention-time agnostic data placement can hurt performance and energy severely. To address this issue, we propose and experimentally evaluate "Data Convection", a thermal-induced NUMA-aware data placement technique that migrates hot-data within a layer, across layers, and across both DRAMs, to balance both the bank-level and channel-level parallelism.
- Compared to a baseline with 3D+2.5D stacked DRAM with thermal-agnostic data placement, our Intra-layer, Inter-layer and Intra + Inter-layer Data Convection schemes improve performance by, on average, 1.8%, 11.7% and 14.4%, respectively.

2 BACKGROUND

2.1 DRAM Refresh

DRAM cells comprise of an access transistor and a leaky capacitor. They are volatile in nature as the capacitors leak charge over time. To maintain integrity of data, DRAM cells are *refreshed* periodically by Memory Controller (MC). The retention time of DRAM cells, often referred to as "refresh window" (represented by t_{REFW}), is in the order of several milliseconds. MC generates the refresh commands by using the previous refresh time for each segment, leaving the refresh circuit to handle refresh operations [51, 54]. The DRAM retention time is a function of the operating temperature as well as the process variation, and varies inversely with the operating temperature. The refresh rate usually varies across different manufactures. The typical DRAM retention time for DDR4 SDRAM chips is 64ms in environments operating in -40°C to 85°C temperature, while it drops to 32ms in environments operating in > 85°C to 95°C temperature, and it further drops to 16ms at >95°C to 105°C [48], [12],[9]. As the operating temperature *increases*, the data retention time *decreases*. Depending on the granularity of refresh employed, DRAM refresh can fall on the critical path of memory accesses. In particular, for a DRAM using per-bank refresh, none of the rows in the bank being refreshed can be accessed. Consequently, DRAM refreshes increase the memory access latency [37], and can be detrimental to overall performance and energy, if the hot data elements are mapped to the frequently-refreshed sections of a DRAM. Increased DRAM chip

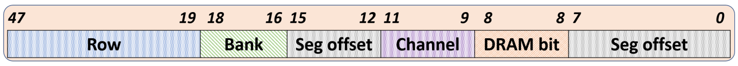


Fig. 1. Memory address mapping in Data Convection.

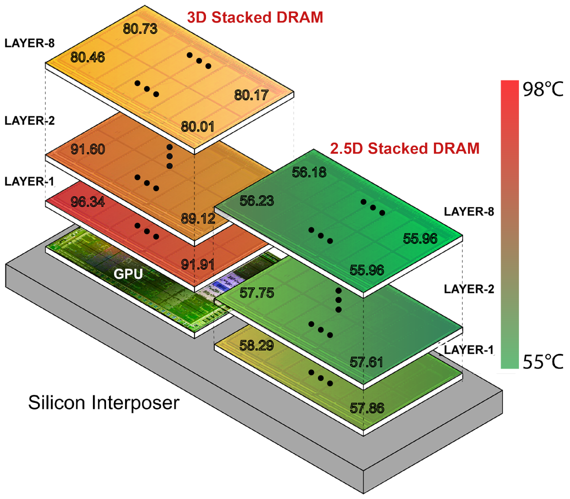


Fig. 2. Temperatures recorded across different sections of both 3D DRAMs on a GPU executing the BionomialOptions and ScalarProd benchmarks as part of a workload.

densities coupled with reduced retention times results in DRAM refresh becoming a performance bottleneck. In addition to the temperature, manufacturing variations also contribute to the DRAM refresh issue. An example is variations between cells in the same die. This variation can be either randomly spread out or can be spatially concentrated in a small area of the DRAM chip.

2.2 Data Mapping in 3D Stacked Memories

Memory address interleaving governs the destination channel, rank, bank, row, and even the location of the DRAM (2.5D or 3D) for each physical address, and it plays a crucial role in governing where the data corresponding to a physical address is mapped to. When employing two 3D stacked memories, we need to consider an additional bit for specifying which memory we want to assign the data. By using a finer-grained address mapping scheme, in which segment offset is divided into two sections, shown in Figure 1, we try to minimize the 'channel camping', where a larger number of requests compete for bandwidth from one channel while the other channel remains under-utilized. In our address interleaving scheme, shown in Figure 1, DRAM bit (bit 8) specifies whether segment should be assigned on the 3D stacked DRAM or on 2.5D stacked DRAM. The channel ID is determined using 3-bits (bits 9-11) and the bank ID is determined using 3-bits (bits 16-18). Further, depending on the granularity of segment size, the segment offset bits are determined. For example, for a 4KB DRAM segment, the lower 8-bits and bits 12-15 are used to determine the byte-offset within a DRAM row. In our implementation, since each channel consists of only one rank, we assume that each channel represents a single DRAM layer.

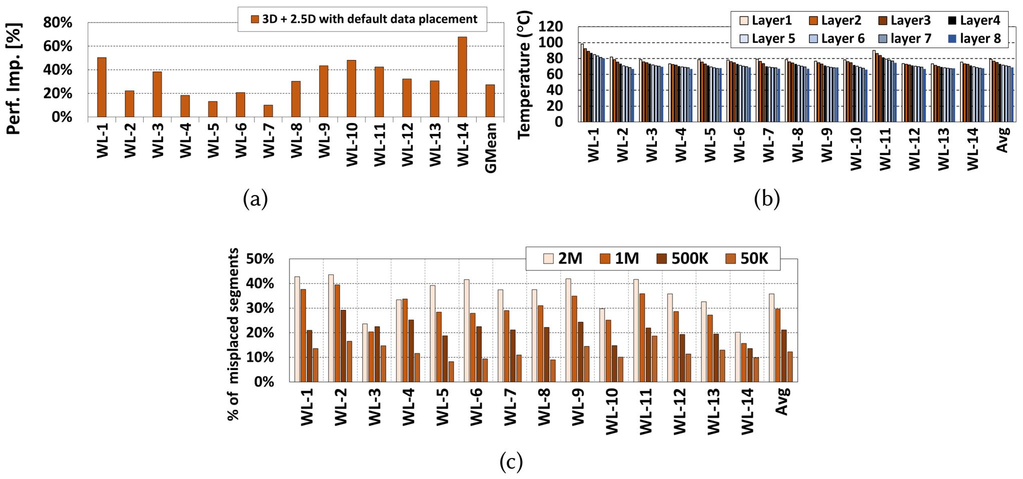


Fig. 3. (a) Performance improvement achieved by placing an additional 3D memory at top of GPU. (b) Maximum temperatures recorded for each layer in a 8-layer (8-high) of 3D stacked DRAM placed on top of a GPU. (c) Percentage of misplaced segments vs epoch duration. The details of our experimental setup and workloads are given in Section 5 and Table 3. *WL is an abbreviation for Workload.*

3 MOTIVATION

For our experiments in this section, we only used a small number of benchmarks (14 out of 196) so we can show the raw numbers for all the results reported in this section. For choosing the benchmarks, we fixed one of the two programs concurrently running on GPU to ScalarProd[52], which is a medium memory-intensive application, and choose the other program from all the workloads mentioned in Table 3. More details on our benchmarks and evaluation methodology are covered in Section 5.

3.1 Capacity Limitation in State Of the Art Proposals

In state-of-the-art proposals, stacked DRAMs can be packaged in a given GPU die in two configurations, i.e., 2.5D and 3D. In a 2.5D configuration, stacked DRAM and GPU are mounted on an interposer, and the data from stacked DRAM are fed to GPU via the links going through the interposer. There are commercially available products for this type of applications. However, one of the major limitations of 2.5D stacked DRAMs is the lower overall memory capacity they offer, which results in data movement between host and device memories for emerging workloads like deep learning [58]. Additionally, integrating more stacked DRAMs in a 2.5D fashion increases the package size, which is not preferable.

In comparison, in a 3D configuration, the stacked DRAM is mounted vertically on top of a GPU. The stacked DRAM designs have been explored extensively in the last decade. [2, 18, 21, 22, 25, 47, 65] are just a subset of the entire body of work. The promise of shorter wire lengths leading to lower energy per bit, higher package density and worse thermal performance, has been established by these prior works. However, the limited capacity and attainable bandwidth of the system are still not addressed using stacked DRAM. To solve these problems, in this work, we evaluate a system that contains two stacked DRAMs – one mounted in a 3D fashion vertically on top of a GPU and one in a 2.5D fashion, proposed in [62]. Note that doing so can increase *both* capacity and bandwidth while maintaining the same package size. As Figure 3a shows, by integrating additional 3D DRAM on the top of the GPU, the performance of the system increases by 27.3%, when memory footprint is extensive, which indicates that the 3D+2.5D proposal can have a huge advantage in comparison to the 2.5D system. This improved performance is due to the increased memory capacity that reduces the data movement between the host and device memories as well as the increased bandwidth offered by 3D stacked DRAM. However, we would like to emphasize that this package still suffers from *thermal issues* for the stacked DRAM mounted vertically on GPU. Our work proposes to reduce the thermal impact on bandwidth by migrating data dynamically. An additional difference between prior works and our paper is the use of a GPU, instead of multicore CPUs in the logic die of the stack. We believe that the thermal management challenge is not vastly different and that the GPU is merely a vehicle for exploration.

3.2 Thermal Gradient in 3D vs 2.5D stacked DRAMs

Figure 2 shows the temperatures recorded across the edges of both the DRAMs in our system, one 3D stacked DRAM and one placed in a 2.5D fashion, for a workload executing the BionomialOptions and ScalarProd benchmarks concurrently on a GPU. As can be observed, there is significant temperature gradient not just across different layers but within the same layer as well, on the 3D stacked DRAM. For the 2.5D DRAM, the inter-layer temperature variation is not significant, and does not affect the overall performance of the system significantly. Additionally, Figure 3b plots the maximum temperatures recorded on each of the 8 layers of the 3D stacked DRAM across all our evaluated workloads when one benchmark is fixed to ScalarProd. As can be seen, on average, the peak temperature gradient between layer-1 (bottom layer) and layer-8 (top layer) of the 3D stacked DRAM is 18.8°C, while the gradient is very small for 2.5D DRAM (around 2.4°C). This inter-layer temperature variation occurs because the heat generated on the GPU die escapes vertically, causing layer-1, which is closest to the GPU die, to be subjected to a higher temperature compared to layer-8, which is farther from the GPU die. From Figure 2, it can also be noted that, in 3D stacked DRAM, the temperature variation is not only observed across layers but it is also prevalent within the same layer. This high Intra-layer temperature variation in 3D stacked memory is due to the fact that the different parts of a GPU die dissipate different amounts of power spatially. That is, the section of a GPU die that contains compute units (like ALUs) dissipate more power, compared to the sections of a GPU die that contain SRAM-based structures like instruction/data caches and TLBs. This causes significant variations in power density across the different sections of the GPU die, causing significant thermal-gradient on the GPU die. Since heat escapes vertically, the sections of DRAM cells that are directly on top of the compute units are subjected to higher temperatures, compared to the sections that are over SRAM based caches. Besides, the difference in temperatures recorded on the same layer is maximum in the bottom-most layer (layer-1) and it decreases as the heat reaches the top-most layer (layer-8). This phenomenon is observed in all of our evaluated workloads. This temperature variation happens since the primary heat source is at the bottom, and the heat sink is at the top of our architecture. Consequently, in a steady state, the highest thermal gradient will be in the upward direction. While there will be lateral thermal gradients, the vertical gradient will be more significant. This observation is similar to prior works such as [50, 60].

Summarizing the observations from Figures 2 and 3b, it is evident that different sections of a 3D Stacked DRAM mounted on a GPU are subjected to higher temperature variation versus 2.5D stacked DRAM that will cause different sections of the both DRAMs to have variable retention times. Consequently, based on the variable retention times, different sections of both DRAMs have to be refreshed at *different frequencies* to maintain data integrity for better performance without conservatively employing the worst-case refresh frequency.

3.3 Software vs. Hardware Data Remapping[1]

As demonstrated earlier, using a retention time-agnostic data placement can cause the most frequently accessed data to be stalled by refreshes, resulting in sub-optimal performance. Thus, to alleviate the stall times caused by DRAM refreshes, one can envision a *retention-aware data remapping scheme*, implemented in software or hardware, that distributes data across the different sections of both 3D and 2.5D stacked DRAMs. We now discuss the trade-offs between software vs. hardware approaches for migrating data.

In a software-based remapping scheme, data remapping is performed by the GPU runtime. Remapping the data in software involves the following steps:

- Tracking and identifying the most-frequently accessed data to limit the amount of remapped data.
- Updating the GPU page tables to replace the old physical address with the remapped physical address.

GPUs, like CPUs, lack sophisticated hardware support to track the access frequency of data and will have to rely on the access-bit information in the Page Table Entries (PTEs) that indicate whether the page is accessed or not. This is severely limiting, as the hot vs. cold data cannot be isolated accurately with a single bit. Even if we have sophisticated support to isolate the hot vs. cold data, since the process of remapping involves updating the old physical address with the remapped physical address in the PTEs, a major drawback of the software-based remapping would be the overheads involved in the remapping itself. Note that updating PTEs with the remapped physical address requires invalidating *all* prior address translations in TLBs, warranting an expensive TLB shootdown process.

Finally, GPU runtime needs to be scheduled on the CPU periodically to remap the mis-placed data across and within DRAMs so that the most-frequently accessed data are less interfered by the refresh operations. This increases scheduling overheads on the CPU. It is imperative that the GPU runtime should only be executed infrequently on a CPU, which causes the mis-placed data in both DRAMs to be remapped only at coarser microsecond if not millisecond granularities. Thus, software-based GPU runtime-initiated data remapping will react slowly, compared to the nimble hardware-based remapping schemes, which can perform data remapping at very fine – tens-to-hundreds nanosecond – granularities. Figure 3c shows the percentage of requests where the data are fetched from the mis-placed locations of the memories with varying data remapping epochs. The mis-placed data correspond to the cases where the most-frequently accessed data are fetched from sections of memory that are refreshed more frequently, and the least-frequently accessed data are fetched from sections that are refreshed less frequently. As can be observed from this figure, for a coarse-grain remapping epoch of 2 Million cycles, about 35% of the requests are fetched from the mis-placed locations, which reduces to less than 10% for a fine-granular 50K cycles epoch. This result indicates that a hardware-based remapping policy which can react faster to the GPU access patterns does a better job at placing the data in a retention-aware fashion.

Hardware-based remapping schemes do not require updating the PTEs with the remapped physical address. The remapped segments'[2] original-to-updated physical addresses are tracked by an additional hardware structure. This approach operates by using the original (un-updated) physical address during the cache lookups and, upon a last-level cache miss, the new updated physical address is looked up in the remapping table to access the correct location on DRAMs. Consequently, this approach does not require shooting down the TLB translations during a segment migration.

[1]We use the terms 'remapping' and 'migration' interchangeably in this paper.
[2]"Segment" refers to granularity of remapped data.

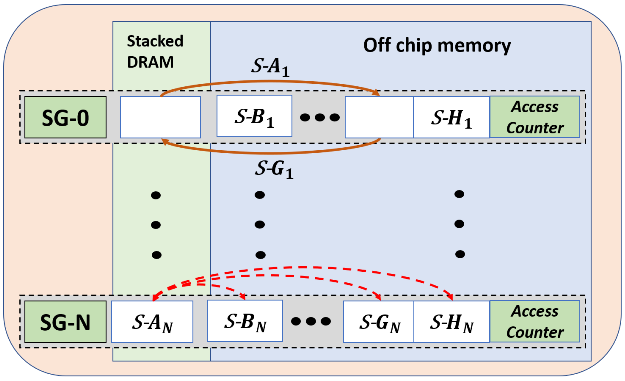


Fig. 4. Segment groups in [38, 59].

Hence, we adopt a hardware-managed remapping scheme in our Data Convection design. Summarizing the observations from Sections 3.1, 3.2, and 3.3, we conclude that, using a thermal-aware data placement in the 3D+2.5D system can improve the performance and bandwidth utilization of the system, compared to the 2.5D configuration, 3D stacked DRAM , and 3D+2.5D baseline. 3D DRAMs mounted on a GPU not only experience inter-layer temperature variances but they also experience intra-layer temperature variances, which cause the different sections of 3D Stacked DRAM to be refreshed with different frequencies. Further, a retention-time agnostic data placement algorithm hurts performance as the most-frequently accessed data segments are stalled by the DRAM refresh, resulting in sub-optimal performance. Additionally, software-based remapping schemes are *not* nimble enough to react to the hardware access patterns and, as a result, they are not effective in fast data remapping compared to the hardware-based schemes. To alleviate the sub-optimal performance due to retention time-agnostic data placement, we present Data Convection, a dynamic data remapping scheme that remaps data across and within 3D DRAMs in a retention-aware fashion by considering the number of contentions at the same time.

4 DATA CONVECTION

In this section, we present our optimizations that are oriented towards addressing the bandwidth lost due to sub-optimal data placement in stacked DRAMs. As explained in Section 3, the data mapped to the regions of DRAMs that have lower retention times due to thermal exposure can hurt performance, as the high frequency refresh operations *interfere* with the data accesses. However, we cannot assign all of the hot segments to the coldest layer of the 2.5D DRAM, because doing so would affect the performance due to increased bandwidth contention. To address these issues, we propose *Data Convection*, our thermal-induced NUMA-aware data remapping strategy, which takes into account access parallelism while data remapping. More specifically, we propose three variants of Data Convection, i.e., i) Intra-layer only, ii) Inter-layer only, and iii) Intra + Inter-layer approaches.

Our Intra-layer Data Convection optimization remaps data only *within a same layer in each DRAM*. It targets remapping sub-optimally placed data segments within different sections of the same layer in a retention-aware fashion. Our second algorithm, Inter-layer Data Convection, remaps data segments *across layers of both DRAMs* in a thermal-induced NUMA-aware fashion. Our final proposal, Intra + Inter- layer Data Convection, takes a combined approach by allowing the data to be remapped within a layer, across layers, and across both DRAMs. While our Intra-layer Data Convection strategy remaps data only in a single DRAM, our Inter-layer and Intra + Inter- layer proposals migrate and distribute data across both the 2.5D and 3D DRAMs. Since the degree of freedom in remapping is higher in our Intra + Inter- layer Data Convection algorithm, we expect this algorithm to perform better than our Intra-layer only and Inter-layer only algorithms.

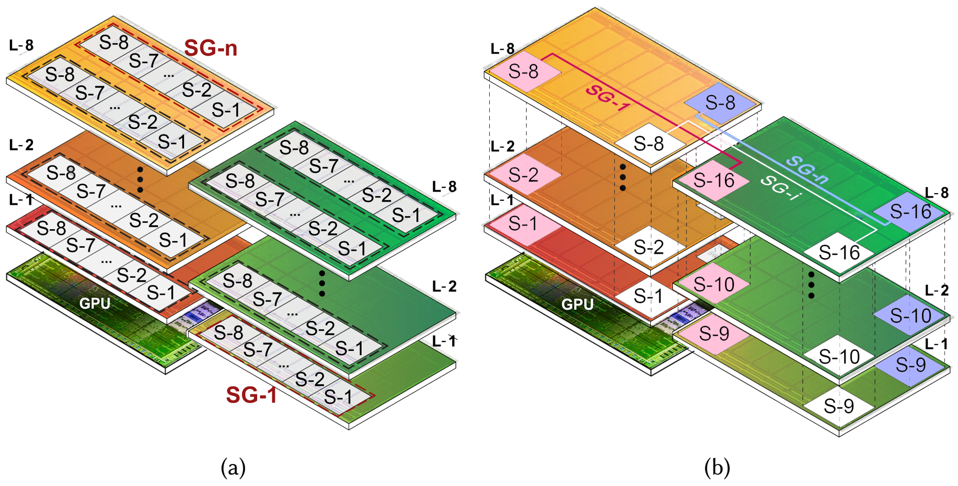
Before presenting the details of our Data Convection algorithms, it is important to briefly discuss the prior hardware-managed data remapping techniques proposed in the context of heterogeneous memories [16, 38, 59] that affect our design. In systems containing regular DDR off-chip DRAM, these prior hardware-managed proposals enabled efficient integration of stacked DRAMs into the overall memory hierarchy, with the objective of increasing the overall memory capacity by using stacked DRAM as part of the main memory (POM). These POM proposals, originally developed for systems employing heterogeneous stacked and off-chip DRAMs, target increasing the data fetched from the high-bandwidth stacked DRAM by remapping data in a NUMA-aware fashion. However, since the NUMA-paradigms arise from heterogeneity in bandwidth, we would like to point out that these proposals can naturally extend to our thermal-induced NUMA-paradigms as well. The original POM proposals employed additional meta hardware structures, referred to as "Segment Remapping Tables" [38, 59] or "Line Location Tables" [16], which contain the remapping information of the original physical address to the remapped physical address. The granularity of remapping is referred to as "segment". While the segment granularity was 64B in [16], it was 2KB in [38, 59]. Note that the segment granularity plays a crucial role in the overall space occupied by the Remapping Table. Additionally, to reduce the overheads involved in swapping the segments between the different memory regions, these architectures group certain segments into a Segment Group (or Congruence Group). Consequently, segments from the same "Segment Group" are allowed to be swapped with one another, thereby allowing Segment Restricted Remapping.

Figure 4 depicts Segment Groups proposed in prior POM architectures [16, 38, 59]. As can be observed, the prior POM proposals targeting the heterogeneous stacked and regular off-chip DRAM systems have a Segment Group comprising segments from the stacked and off-chip DRAMs. This allows segments to be remapped between the stacked and off-chip DRAMs. Motivated by this design, in our thermal-aware data remapping architectures, the Segment Groups are spanned across the different sections of both 3D DRAMs such that each Segment Group consists of sections that have significant thermal gradient. More details on the Segment Group formation in our proposal are covered later.

4.1 Hardware Modifications Overview

The first hardware modification required for our hardware-managed Data Convection are Temperature sensors, which are essential to make our proposal adaptable in 3D DRAM due to dependability of our proposals on recording the temperature from different sections of DRAM. Although the physical locations of on-die thermal sensors are typically not disclosed by manufacturers, several prior articles [1, 17, 43, 46] indicate that, in today's CPUs and GPUs, we have many thermal sensors distributed throughout the die area (24 in IBM's POWER5 [17]), and the number of sensors keeps increasing over time [46]. Note that, integrating a similar number of sensors on the HBM die area is not vastly different, and in our proposal, only 8 sensors are required per layer (equal to the number of banks). They can be implemented as a current sensor with metal layer resistance and will not impact DRAM capacity. Instead we will observe better performance due to our data migration algorithms. In the situation where there are fewer sensors on the DRAM layers, we can still estimate the temperature at different DRAM layers using the temperature measured across the GPU die using a thermal model similar to HotSpot [24, 29]. Besides, as results in Section 6 shows, our Inter-layer algorithm can achieve considerable performance improvement using only one temperature number per layer, which can be achieved using the current formation of temperature sensors.

In addition to temperature sensors, Data Convection approach requires hardware structures to track the access count per-each segment as well as the remapping information that points the



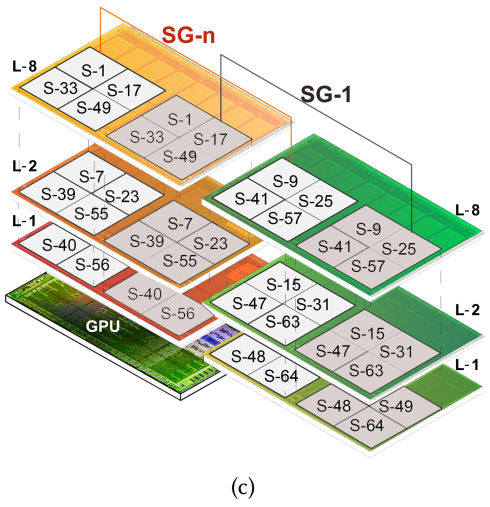


Fig. 5. Segment Groups in (a) Intra- (b) Inter- and (c) Intra + Inter-layer schemes. DRAMs are aligned for better visibility of the segments (L-x represents Layer-x; S-x represents Segment-x, while SG-x represents Segment Group-x).

original physical address to the remapped physical address. To that end, our Data Convection proposal requires tracking the most frequently accessed segments through a filter cache proposed in [32]. These most frequently accessed segments are remapped at the end of every remapping epoch. Since our mechanism proposes remapping segments within and across the stacked DRAMs, the segment remapping requires swapping the candidate mis-placed segment with the segment at the destination location. Hence, similar to [16, 38, 59], our approach still requires a remapping table to track the segments. Similar to the prior proposals [38, 59], to reduce the total remapping table size, our hardware-based Data Convection approach only allows remapping segments within the same segment group. That is, a most frequently accessed mis-placed segment from a segment group can only be swapped with a segment within the same segment group that resides in the region that is refreshed less frequently. Additionally, to reduce the overheads of remapping the segments, we allow the fast-swap based remapping proposed in [59], which reduces the overall number of swaps, thereby reducing the time spent in swapping the segments. The fast-swap approach reduces the number of swaps by allowing segments to be entirely remapped within the same segment group. This is possible as the remapping table entry for each segment group contains additional storage to store the possible remapping information of every segment (even if the segment is originally not remapped). By confining the possible number of remapping locations, the Segment Restricted Remapping reduces the payload for storing the remapped location. For migrating the data, we need a buffer space in the memory controller to store the segments and forward them to the new location during the migration process. In the next subsection, we discuss our thermal-gradient-aware segment group formation.

4.2 Thermal-Gradient-Aware Segment Groups

As explained before, prior works [38, 59] target remapping the data between the stacked and the off-chip DRAMs. Consequently, since the segments are remapped only within a segment group, in their approach, a segment group comprises segments from the stacked and off-chip DRAMs, spreading across both the stacked DRAM and off-chip DRAM. Unlike the prior proposals, since our Data Convection proposal targets addressing the thermal-induced NUMA issues within and across 3D and 2.5D stacked DRAMs, the segment groups in our proposal are spread across different sections of both 3D and 2.5D stacked DRAMs that exhibit significant temperature gradient. Such thermal-gradient-aware segment groups enable remapping the mis-placed segments within a segment group and minimize the swap and remapping table overheads. That is, the segments that are most frequently accessed and allocated in regions that experience higher temperatures can be remapped to the regions that are relatively colder, thereby reducing the stall times induced by refreshes. Additionally, we also consider bandwidth contention when remapping the segments so as to avoid artificial hotspots created by remapping. Hence our Data Convection algorithms remap the segments to address thermal-induced NUMA bottlenecks while balancing the access parallelism at the same time.

Figures 5a, 5b, and 5c show the thermal-gradient-aware segment groups in our Data Convection proposals. In the Intra-layer Data Convection approach, a segment-group comprises segments within the same layer of each DRAM, as depicted in Figure 5a. As a result, segments can be remapped with another segment within a segment group from the same layer. Since each layer is a channel containing only one rank and multiple banks, assuming a stacked DRAM layout where multiple banks are subjected to spatial thermal-gradient, an Intra-layer segment group can be formed such that it spreads across the same row within multiple banks of a layer. Consequently, as depicted in Figure 5a, a segment-0 from bank-0 and row-'X' which is more frequently accessed and is subjected to higher temperature, can be remapped with another segment (say segment-7) in bank-7 and row-'X' that is subjected to the lowest temperature within that segment group. As can be noted, this scheme does not allow segments to be remapped across layers, even though there is significant temperature-gradient across layers.

In Inter-layer Data Convection, as depicted in Figure 5b, a segment group is spread across multiple layers of both 3D and 2.5D stacked DRAMs. Figure 5b depicts a sample segment-group, formed out of the bank and row from across layers in our Inter-layer Data Convection approach. Consequently, since layer-1 of 3D stacked DRAM (which is closest to GPU) is at a higher temperature compared to layer-8 of the 2.5D stacked DRAM (which is coldest layer on the chip), a segment from layer-1 of bank-0 and row-'X' of the 3D stacked DRAM can be swapped with a segment residing in bank-0 and row-'X' of layer-8 of 2.5D stacked DRAM. As can be noted, this scheme does not allow remapping segments within the same layer, even though there is a notable thermal-gradient across the sections of the 3D stacked DRAM within the same layer.

These Intra- and Inter-layer only approaches restrict remapping of the segments either within a layer or across layers respectively, limiting the degree of freedom in remapping. These schemes could potentially result in sub-optimal performance in a scenario where a large data structure like an array that was mapped to contiguous banks and ranks during the initial allocation is accessed more frequently compared to the other data structures which cause contentions for most frequently accessed data. To alleviate such contention, we propose a *combined* Intra + Inter-layer remapping scheme, which allows remapping the segments in the two dimensions, that is, within a layer and across layers of both the stacked DRAMs.

Figure 5c shows an example segment-group formation in our Intra + Inter-layer Data Convection algorithm. As can be observed, the segment-group in this case spans across multiple banks in the same layer as well as across the layers of both the stacked DRAMs. As a result, a segment in a bank can be migrated to the same row in multiple banks within a layer, across layers, *and* across DRAMs. Hence, this Intra + Inter-layer Data Convection algorithm allows more degrees of freedom, and consequently, results in better performance, as will be presented later in Section 6.

4.3 Segment Remapping Algorithms

As mentioned in Section 4.1, the efficacy of our scheme depends on the amount of data migrated as the remapping overheads can become detrimental to performance. To reduce the remapping overheads, we propose only migrating the most-frequently accessed segments in an epoch. We track the most frequently accessed segments through a filter cache structure similar to [32]. More specifically, the most frequently accessed segments are tracked using the original physical address and, at the end of each remapping epoch, the segments in the filter cache are sorted by the hardware before the mis-placed segments are remapped based on the temperature information collected from the temperature sensors. Consequently, the filter cache size plays a crucial role in capturing the most-frequently accessed segments in our approach. The advantages of ranking segments based on the access count from the filter cache at the end of each remapping epoch is two-fold:

- First, it helps us in taking better remapping decisions when dealing with multiple hot segments per segment group.
- Second, since the segment groups in our techniques are spread across layers, banks, as well as DRAMs, just remapping segments in a retention-aware fashion would unfortunately create scenarios where all hot segments are remapped to a single bank or a single channel of a stacked DRAM, causing signification loss in bank-level and/or channel-level parallelism. Ranking the segments not only helps us in performing retention-aware remapping of the mis-placed segments but it also helps us in remapping, taking into account the bank-level and channel-level parallelism.

Algorithm 1 gives the pseudo-code of the routines used in remapping the mis-placed most frequently accessed segments within their corresponding segment groups. The overall algorithm contains three main routines that enable remapping.

The MIGRATOR routine in line-11 of Algorithm 1, which is invoked at the end of each epoch, performs the remapping for all most frequently accessed captured by the Filter Cache by calling the REMAP_SEGMENT routine.

The REMAP_SEGMENT is the most important routine; it remaps the mis-placed segments in a thermal-induced retention-aware fashion, taking into account the bank-level and channel-level parallelism at the same time. Each segments' segment group number is obtained from the SEG_GRP_NUM routine. Note that, the position of a segment in a sorted list, which indicates how frequently a segment is accessed, plays a crucial role in our scheme. The target location where a remapped segment will be migrated to (if it is mis-placed) is calculated based on its global ranking, as shown in line-19. As explained in lines-20 to 22, the target location, by considering the global ranking automatically, balances the bank-level and channel-level parallelism. The mis-placed condition check is performed by comparing the refresh rates of the current segment's location and the new target location, and the segment is remapped only if the target refresh rate is different from the current locations' refresh rate. If the target locations' refresh rate is the same as the current locations' refresh rate, the remapping is *skipped* as there will not be any improvement in performance by remapping to a location which is refreshed at the same rate.

Finally, the SEG_GRP_NUM routine returns the segment group number corresponding to the original physical address of a segment. As the number of segments within a segment group varies based on the Data Convection scheme employed, the segment group number generated is a function of the Data Convection scheme as well as the segment granularity itself. For example, assuming segment granularity to be the same as the DRAM row size, in the Intra-layer scheme, since a segment group spreads across multiple banks in the same layer consisting of segments with the same DRAM row-id, the number of segment groups per layer in the Intra-layer scheme is the same as the number of DRAM rows in a bank. Similarly, in the Inter-layer Data Convection, as the segment group spreads across multiple layers, again assuming that the segment granularity is the same as the DRAM row, the number of segments in a segment group is equal to the number of layers multiplied by the number of DRAMs. Finally, for the Intra + Inter-layer Data Convection, as the segment group spreads across multiple layers on multiple DRAMs and across multiple banks in each layer, the number of segments (and hence the segment group number) is a function of the total number of banks per layer, total number of layers, and number of DRAMs involved in a segment group formation. The formula for calculating the segment group for the Intra-layer, Inter-layer, and Intra + Inter-layer schemes are shown in line-36, line-38, and line-40, respectively, in Algorithm 1.

Our REMAP_SEGMENT not only performs retention-aware remapping of segments, but it also improves the access parallelism in the respective schemes. For example, in the Intra-layer scheme, the bank-level parallelism is improved during remapping as the segments are remapped across banks in a round-robin fashion, as indicated in line-20 of Algorithm 1. For example, the Intra-layer algorithm places the hottest segment into the coldest bank in its segment group, the second-most hottest segment is remapped to the second-most coldest bank of its segment group, etc. Thus, the most frequently accessed segments are spread across multiple banks, which helps to ensure that the overall-bank level parallelism will be higher. In the Inter-layer approach, our algorithm improves the channel-level parallelism by remapping the segments across the layers of both DRAMs in a round-robin fashion based on the segments' position in the sorted list. Thus, the hottest segment is remapped to the coldest layer of two DRAMs, the second hottest segment is remapped to the second coldest layer of two DRAMs, and so on. As a result, our Inter-layer approach, by remapping across layers in a retention-aware fashion, also increases the channel-level parallelism. Our Intra + Inter-layer approach on the other hand, targets remapping segments in a retention-aware fashion and improves *both* bank-level and channel-level parallelism. As shown in line-22, we distribute the segments across banks and layers of both DRAMs in a round-robin fashion to increase both the bank-level and channel-level parallelism. For example, in our Intra + Inter-layer algorithm, the hottest segment will be remapped to the coldest bank of coldest layer in two DRAMs, the second hottest segment will be remapped to the coldest bank in the second coldest layer of two DRAMs, and so on.

5 EVALUATION SETUP AND METHODOLOGY

In this section, we describe the evaluation methodology, setup and workloads used in our evaluation. Unlike a baseline with conservative refresh frequencies based on the hottest layer, we consider

Algorithm 1 Segment Remapping Algorithm

```
$segLoc[seg\_j]$   ▷ contains the location of each segment within its group
$num\_channels = 8$   ▷ Number of channels(layers) in each DRAM.
$num\_banks = 8$   ▷ Number of banks in each channel.
$num\_drams = 2$   ▷ Number of DRAMs in our config (one 3D stacked and one 2.5D stacked.)
$num\_rows$   ▷ Set by user in BIOS.
$NUM\_BANKS\_PER\_GRP$   ▷ Number of banks in the segment group.
$NUM\_CHANNELS\_PER\_GRP$   ▷ Number of channels in the segment group.
$NUM\_DRAMS\_PER\_GRP$ ▷ Number of drams in the segment group (1 in Intra and 2 in both Intra + Inter and Inter)
$MAX\_SEGMENTS\_PER\_SEG\_GRP = NUM\_CHANNELS\_PER\_GRP * NUM\_BANKS\_PER\_GRP * NUM\_DRAMS\_PER\_GRP$
    ▷ Total number of segments in each group( 128 in Intra + Inter, 8 in Intra, and 16 in Inter algorithm.)
function MIGRATOR
    for All segments in FCache do
        REMAP_SEGMENT($segment$)
    end for
end function
function REMAP_SEGMENT($seg\_num$)
    $curr\_loc = segLoc[seg\_num]$
    /* tmp_position is the segment ranking based on its access frequency. */
    $tmp\_position$ = FCACHERANKING($seg\_num$)
    /* For Intra-layer: target_loc=i ⟶ remaps segment to i-th coldest bank in the seg-group. */
    /* For Inter-layer: target_loc=i ⟶ remaps segment to i-th coldest layer in the seg-group */.
    /* For Intra + Inter-layer: target_loc=i ⟶ remaps segment to (i/num_channels)-th coldest bank in (i%num_channels)-th coldest layer in the seg-group. */
    $target\_loc = tmp\_position\%MAX\_SEGMENTS\_PER\_SEG\_GRP$
    $target\_refresh\_rate = target\_loc.refreshRate()$
    $old\_refresh\_rate = curr\_loc.refreshRate()$
    if ($target\_refresh\_rate != old\_refresh\_rate$) then
        $Migrate(seg\_num \longrightarrow$ SEG_GRP_NUM($seg\_num$), $target\_loc)$
        $segLoc[seg\_num] = target\_loc$
    end if
end function
function SEG_GRP_NUM($seg\_number$)
    if Algorithm==Intra-layer then
        $dram\_id = seg\_number\%num\_drams$
        $channel\_id = (seg\_number \gg log2(num\_drams))\%num\_channels$
        $row\_id = (seg\_number \gg (address\_size - log2(num\_rows * num\_drams)))$
        return $(row\_id * num\_channel + channel\_id) * num\_drams + dram\_id$
    else if Algorithm==Inter-layer then
        return $seg\_number \gg (log2(num\_channels) + log2(num\_drams))$
    else if Algorithm==Intra + Inter-layer then
        return $seg\_number \gg (log2(num\_channels * num\_banks * num\_drams))$
    end if
end function
```

Temp. (°C)	Retention Time (ms)	Temp. (°C)	Retention Time (ms)
70 - 75	128	90 - 95	32
75 - 80	96	95 - 100	24
80 - 85	64	100 - 105	16
85 - 90	48	>105	NA

Table 1. DRAM Retention Time vs. Temperature

a performance-optimized baseline (explained in Section 5.2) that employs *"variable refresh rates" based on the subjected temperature*. Table 1 summarizes the retention-times used in our evaluations at different temperatures. Note that, thermal sensors are already in-place in the commercial systems to ensure that the processor operates within Thermal Design Power (TDP) [8].

5.1 Experimental Setup

We used GPGPUSim v3.2.2 [10] simulator to model various components in a GPU. More specifically, we used Mafia [33] framework to model a *virtualized GPU* that can execute multiple applications on a GPU concurrently. We modeled 3D and 2.5D stacked DRAMs using Ramulator [23, 35], which is integrated with Mafia [33]. More details about our simulated configuration can be found in Table 2. GPUWattch [10] is used to obtain the power dissipation values for components of GPU and DRAMs. These power numbers are then fed to Hotspot v6.0 [29] along with the GPU floor plan for thermal modeling. Hotspot is modeled in the transient-state mode. We *integrated* Mafia with Ramulator and Hotspot to create a “closed-loop framework" that enables thermal-feedback to the GPU hardware modeled in Mafia. For starting temperature, we ran all the benchmarks on GPU to reach a steady-state, and used the “average" as our ambient temperature.

5.2 Methodology

We performed our experiments in a multi-program environment. We want our proposal to adapt to datacenter systems since GPU virtualization and multi-programming are becoming more common in this type of environment. Besides, temperature variations can also occur in a single application scenario (like deep learning) since applications can execute multiple kernels with different characteristics concurrently.

For evaluating our proposal, we execute two applications at the same time. We choose these two applications from a group of *memory-intensive* and *compute-intensive* benchmarks to evaluate the effects of our Data Convection using different memory access characteristics. In order to simulate the applications with large footprints, we increased their input datasets to a maximum of 4GB. Our benchmarks along with their abbreviations, LLC MPKI (Miss Per Kilo Instructions) values, and their categories (compute-intensive vs memory-intensive) are listed in Table 3. The first nine workloads are more compute-intensive, and the remaining workloads are more memory-intensive. Therefore, for each of our experiments, we run 196 (14×14) different combinations of applications in order to comprehensively evaluate Data Convection. In each experiment, we simulated a two benchmark-bundle for 20 million instructions. In every 1 million GPU clock cycles, using the power data generated by GPUWattch[39], we updated the temperature through Hotspot [29]. Furthermore, after each epoch of 50,000 cycles, we invoked our Data Convection algorithm to *remap* segments.

For our baseline, we consider a system containing both 3D and 2.5D stacked DRAMs with thermal-aware variable refreshes with eight refresh rate levels (see Table 1) [9, 12, 48] and without any data placement optimizations. In addition, as part of our motivational experiments, we also consider another system with only one 2.5D DRAM and with thermal-aware variable refreshes with eight refresh rate levels. In both of these systems, our GPUs employ Dynamic Voltage Frequency Scaling (DVFS) to ensure that they operate within the TDP. We used 8-levels of voltage and frequency based on prior proposals [64][45][5]. Table 4 gives the voltage and frequency values used in our

	Module	Config
GPU Config	**SM Details**	16 SMs running @ 1000MHz
	Cache block size	128B
	L1 Data Cache	16KB 4-way
	L1 Instruction Cache	2KB 4-way
	L1 Texture Cache	12KB 24-way
	L1 Constant Cache	8KB 2-way
	Shared Memory	48KB
	L2 Cache	16-way 128 KB/memory channel (768KB in total)
	Filter Cache	MFC[32] with 512KB size and 8 way associativity. The counter size for each segment is 8 bits.
Stacked DRAM Config	**Type**	2 dice of HBM v2
	Capacity per Stacked DRAM	4GB
	Channels/Ranks	8 channels/1 rank per channel
	Pseudo Channels	2 Pseudo Channels per channel
	Banks	8 (4 per Pseudo Channels)
	Rate	2000 Gbps
	Frequency	1000 MHz
	Maximum Bandwidth (Bps)	256GBps
	Channel Width (bits)	128
	Segment Size	4KB
	Refresh Config	Single bank refresh $tRFC_{sb}$ = 160ns
Power Characteristic	**Average energy consumption per bit**	3.7 pJ/bit for DRAM and 6.8 pJ/bit for logic cells [43]
	Silicone specific heat	1.75e6 $J/(m^3.K)$
	Silicone thermal conductivity	100 $W/(m.K)$
	Heat sink and heat spreader specific heat	3.55e6 $J/(m^3.K)$
	Heat sink and heat spreader thermal conductivity	400 $W/(m.K)$
	Heat sink thickness	0.0069m
	Interface specific heat in	4.0e6 $J/(m^3.K)$
	Interface thermal conductivity	4.0 $W/(m.K)$

Table 2. Evaluated Configuration.

baseline. We employ a single (per-) bank refresh that incurs lower refresh overheads compared to all-bank refresh [14, 37].

6 EVALUATION RESULTS

6.1 Performance Results

Figure 6a shows the categorization of benchmarks based on the improvements brought by our three proposed algorithms, with respect to the *3D+2.5D baseline.* Figure 6c shows the different statistics regarding each result, such as the maximum performance loss and improvement and the average performance improvement we achieve when using our proposals.

Our Intra-layer migration algorithm improves performance by a geo-mean of 1.8% (maximum of up to 11.3%) across all workloads, over the 3D+2.5D baseline with default data placement. It is to be noted that, our Intra-layer migration only moves data across different sections of the stacked DRAMs within the “same layer", based on the “thermal gradient". The thermal gradient in a layer in the 3D stacked DRAM is due to the different compute intensities across the workloads running on the SMs. To leverage the shared SRAM structures across SMs, kernels from the same application are generally mapped to spatially-contiguous sections; this results in power dissipation variance if we have a compute-intensive application co-running with a memory-intensive application. This variance in power dissipation transpires to spatial variance in temperatures within a layer. Leveraging this

Benchmark	WL	LLC-MPKI	Type
Binomial Options[52]	WL-1	**0.2**	C
MaxFlops[19]	WL-2	**1.2**	C
dtc8x8[52]	WL-3	**0.3**	C
dwtHaar1D[52]	WL-4	**0.8**	C
mergeSort[52]	WL-5	**0.2**	C
MonteCarl[52]	WL-6	**1.2**	C
quasirandomGenerator[52]	WL-7	**0.8**	C
SobolQRNG[52]	WL-8	**1**	C
sortingNetworks[52]	WL-9	**1.3**	C
BlackScholes[52]	WL-10	**1.7**	M
FFT[19]	WL-11	**1.8**	M
fastWalshTransform[52]	WL-12	**2.4**	M
radixSort[19]	WL-13	**1.5**	M
scalarProd[52]	WL-14	**3**	M

Table 3. Benchmark Characteristics. (WL: Workload)

Level	f(MHz)	VDDC(V)	Level	f(MHz)	VDDC(V)
0	300	0.750	4	1251	1.062
1	600	0.825	5	1294	1.100
2	927	0.850	6	1339	1.143
3	1179	0.993	7	1380	1.187

Table 4. DVFS Levels.

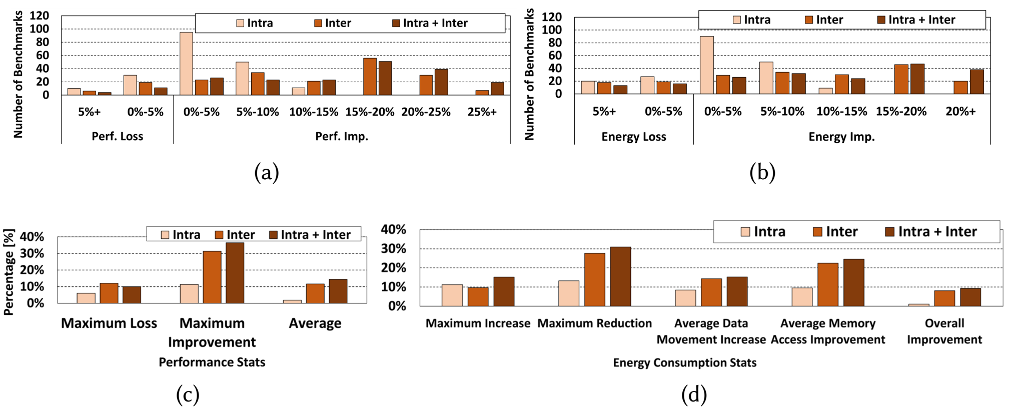


Fig. 6. (a) Performance Improvement with Data Convection, (b) Energy Consumption Improvement with Data Convection, (c) Performance Improvement Overall Stats, (d) Energy Consumption Overall Stats.

spatial variance in temperatures, our Intra-layer remapping scheme *remaps* the most frequently-accessed segments to segments with lower retention time within the same segment-group, as explained in Section 4. The maximum temperature gradient we observed in Figure 3b is **6.6**°C in the bottom layer of the 3D stacked DRAM for WL-1, and this gradient decreases as we move away from the GPU layer. We see that the Intra-layer algorithm finds small avenues to remap the most frequently accessed segments with other segments within the same layer over the 3D+2.5D baseline with default data placement. Note that our Intra-layer scheme also increases bank-level parallelism by remapping the hot segments in a round-robin fashion within a layer.

On the other hand, our Inter-layer remapping scheme targets moving segments *across* the layers of both the stacked DRAMs, and consequently, it finds more avenues to remap segments across layers within a segment group as the thermal gradient across layers is quite significant, as discussed in Section 3. Overall, it improves performance by 11.7% (maximum of up to 31.3%) across all workloads over the optimized baseline. Note that our Inter-layer scheme increases channel-level parallelism by remapping the hot segments in a round-robin fashion and only need one sensor per layer.

Finally, in our Intra + Inter-layer remapping scheme, since the segments can be remapped *within and across layers of both the stacked DRAMs*, there are more opportunities for remapping the segments in a thermal-aware fashion. As a result, this algorithm improves performance by 14.4% (maximum of up to 36.4%) over the optimized 3D+2.5D baseline, as it improves not only bank-level but also channel-level parallelism. The only downside to this approach is the need for additional sensors per layer.

Our proposal improves the parallelism of the system by distributing the access across banks/channels. The improvement due to bank/channel parallelism is dependent on the number of migrations in each epoch. If the temperature was not an issue, our proposal does not achieve good performance (13.3% performance reduction) since in each epoch we migrate all the segments in our Filter Cache according to their hotness (unlike our proposal, which only migrates if there is a difference in refresh period in destination), which is substantial. However, by swapping the segments when their number of accesses differ significantly (around 20% of the total accesses to the segment with the lowest number of accesses) and all the segments are part of one segment group, our Intra+Intra-layer migration can achieve a geo-mean improvement of 3.4%, compared to the baseline, which shows that our proposal can effectively improve the channel/bank-level parallelism by decreasing the number of conflicts if we can control the migration overhead. Still, the dominant improvement is coming from temperature-aware migration techniques.

To summarize, our Intra-layer Data Convection algorithm brings low performance improvement due to lower temperature gradient across layers. In comparison, Inter-layer data remapping scheme benefits from higher temperature gradient across layers. However, it is not optimal as it cannot leverage the intra-layer variations in temperature. Combining the two approaches, our Intra + Inter-layer scheme achieves the best performance improvement.

6.2 Energy Consumption Results

Figures 6b and 6d show the categorization and statistical results regarding energy consumption of our Data Convection proposals, compared to the 3D+2.5D baseline. These results contain two components – data migration and memory accesses. While our proposals increase the data migration energy usage due to relocating the hot segments, improvements brought by lower memory accesses energy consumption (due to smaller number of collisions) and lower execution time usually surpass the data migration energy usage, which ultimately leads to lower overall energy consumption. It is important to note that in all experiments, the power consumption from additional data structures (implemented using DRAM) is considered by using the average energy consumption values mentioned in Table 2. These results indicate that data migration increases the energy consumption of Intra + Inter-layer proposal by a geo-mean of 15.3%. However, by improving the system's performance and minimizing the collisions, we decrease the memory access energy consumption by an average of 24.6%, leading to lower overall consumption. As shown in Figure 6d, over a 3D+2.5D baseline, the overall energy consumption decreases by 9.3% on average, as the DRAM refresh operations minimally stall the most frequently accessed data. Only in small set of benchmarks, our proposal increases the energy consumption.

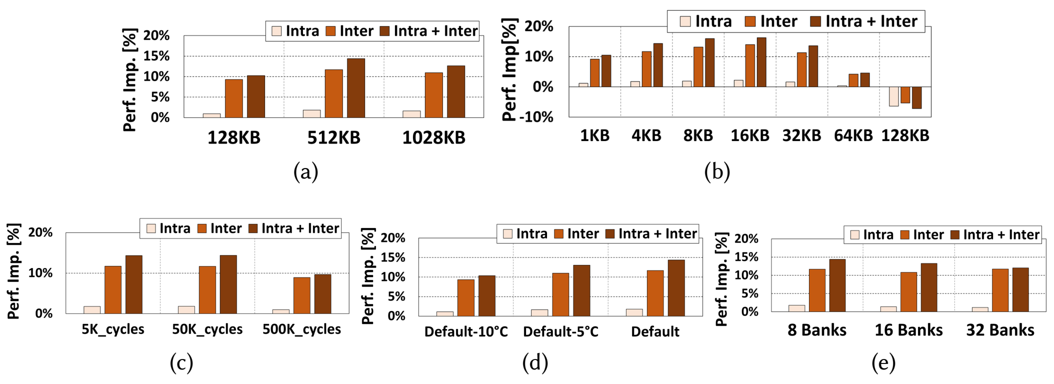


Fig. 7. Results for different (a) filter cache sizes, (b) segment granularities, (c) remapping epochs, (d) starting temperatures, and (e) banks per DRAM layer.

6.3 Bandwidth Consumption

We also performed an experiment to evaluate the bandwidth consumption increase for our Intra + Inter-layer proposal. The results indicate, our proposal consumes between 4.3% and 22.1% (with an average of 12.6%) more bandwidth compared to the 3D+2.5D baseline for migrating the data both within and between 3D DRAMs. While our proposal can migrate all the segments within Filter Cache, since we only migrate if the refresh rate is different between source and destination, the number of migrations is limited, thereby minimizing the incurred bandwidth consumption overhead.

6.4 Sensitivity Results

In this subsection, we present sensitivity results for various important parameters in our evaluation.

6.4.1 Filter Cache Size Sensitivity. Filter cache plays a crucial role in identifying the hot segments. Figure 7a shows how the performance changes with various Filter cache sizes. For Filter Cache sensitivity, we evaluated three different sizes: 128KB, 512KB, and 1MB. It is to be noted that the Filter Cache size governs the number of mis-placed segments that need to be migrated at the end of a remapping epoch. In comparison to the 3D+2.5D baseline with the default data placement, our Intra + Inter-layer Data Convection scheme improves performance by 10.2%, 14.4%, and 12.6% for the 128KB-, 512KB-, and 1MB-sized Filter Caches, respectively. When the Filter Cache capacity is set to 128KB, it cannot track many segments, which limits the mis-placed segments that can be remapped and that in turn limits the potential performance improvement. On the other hand, increasing the Filter Cache size to 1MB, allows tracking more segments that need to be remapped at the end of a remapping epoch. However, migrating a large number of segments wastes the precious memory bandwidth for remapping, which consequently degrades performance by 1.8%, compared to a 512KB Filter Cache configuration.

6.4.2 Segment Size Sensitivity. Segment size, which is the granularity of tracking and migration, plays a crucial role in the overall design of our schemes. A smaller segment granularity will require a lot of filter cache entries to track contiguously allocated most-frequently data structures. Consequently, filter cache space is exhausted as we store more entries with a smaller segment granularity. With a larger segment size on the other hand, the number of filter cache entries is reduced, resulting in tracking a higher number of most-frequently accessed segments. Figure 7b presents the normalized performance as the segment granularity is varied from 1KB to 128KB (recall that our original results used 4KB as the segment granularity). The performance improvements brought by the Intra + Inter layer algorithm increase from 10.5% to 16.3% when we increase the segment size from 1KB to 16KB, compared to the 3D+2.5D baseline. This is because, the larger segment granularities can effectively track more number of hot segments causing the mis-placed segments to be remapped, thereby improving the performance. After this threshold got exceeded however, the performance improvements start to diminish since the overhead of migration increases which surpasses the improvement achieved via data migration. In fact, when employing a segment size of 128KB, we see an average performance loss of 7.1%, due to the migration overheads incurred.

6.4.3 Remapping Epoch Sensitivity. The default remapping epoch used in our evaluations so far was 50K cycles. We also carried out experiments with two other remapping epochs sizes – 500K and 5K cycles. Figure 7c shows how the performance improvements vary with different remapping epoch sizes. We see that the performance improvement drops from 14.4% to 14.3% for 5K cycles, because the additional re-mappings increase the time spent in remapping. At the end, the performance improvement does not change significantly. However, for 500K cycles, since we react late to the access patterns and hence the remapping is not timely (causing the most-frequently accessed segments to be stalled by DRAM refreshes), the performance improvement drops from 14.4% to 9.7%.

6.4.4 Starting Temperature Sensitivity. As mentioned earlier, we executed all the applications until they reach a steady-state, and used the average as our starting temperature (referred to as Default). To show the effect of lower starting temperatures, we ran two experiments with Default-5°C and Default-10°C as starting temperatures. With starting temperatures of Default-5°C and Default-10°C, our Intra + Inter-layer Data Convection algorithms improve performance by 13.1% and 10.4% over the 3D+2.5D baseline vs. 14.4% at the original starting temperature. The lower initial temperature increases the refresh period of some of the banks, thereby reducing the upper limit for improvement. However, since the temperature figures we report are based on a scaled-down version of GPU, one can predict that in commercial GPUs such as Radeon VII [7] with more compute units, temperature variations can be even worse.

6.4.5 Bank Number Sensitivity. The number of banks per layer plays a significant role in all three of our algorithms, especially for Intra-layer and Intra + Inter-layer remapping schemes. The default number of banks per layer used in our evaluations so far was 8 banks per layer. We also run our experiments using 16 banks and 32 banks per layer, shown in Figure 7e. The result shows that, for our Intra + Inter-layer migrations, we achieve an average speedup of 14.38%, 13.24%, and 12.05% for 8 banks, 16 banks, and 32 banks, respectively. As the results indicate, the performance improvement decreases by increasing the number of banks per DRAM layer. The primary reason is that by increasing the number of banks, we see fewer conflicts in the baseline, thereby decreasing the maximum potential improvements. In addition, the temperature difference across each layer is not that significant, causing two nearby DRAM banks to have higher than 5 degrees difference, needing different refresh rates. Still, the results indicate that in all these cases, we are achieving speedup since the significant part of our improvement is due to temperature differences across layers. Therefore, our proposal is valid for all the architectures with different banks and channels.

6.5 Area Overhead

To implement Data Convection, we need two additional data structures and FilterCache [32], for sorting the segments based on the number of accesses. The first data structure is a "remapping table" that maps the original physical address to a new location. This table contains $2^{\text{SegmentsNumber}} = 2^(33 - 12)$ entries, each entry having 7 bits (to store new Bank, Channel, and DRAM bits), requiring a total of 2MB. Data Convection needs another data structure to store the temperature of the different banks of DRAMs. The total number of entries in this table is equal to the total number of sensors (which is equal to 128; 64 per DRAM). Since the sizes of these tables are not significant, they can be implemented using SRAMs. Furthermore, we use FilterCache, a previously-proposed mechanism for identifying hot pages and sorting the pages based on their rank. This data structure makes use of 512KB, which can be easily stored in both SRAM and DRAM.

7 RELATED WORK

We divide the relevant prior research into various categories, and briefly discuss the benefits of our work.

Physical structure: As described earlier, a full 3D architecture implies that compute and main memory are integrated into a single package. Furthermore, full 3D architectures can be subdivided into memory-on-top and processor-on-top (see Agrawal et al. [2] for more details). Thermal herding [56] proposes moving the hottest datapaths closer to the heat sink for better heat dissipation. Thermal TSVs [2] are also used as a means to extract heat vertically, with algorithms to opportunistically boost clock frequencies. Zhang et al. [66] report improvements in the performance of a 3D stacked GPGPU with DRAM, compared against a 2D equivalent system, but they do not discuss thermal implications. Our work is, in a sense, a more completed version of these papers and more importantly complements the prior proposals.

Memory architecture: Zhao et al. [67] proposed optimizing energy efficiency by reconfiguring memory interfaces via bus width, voltage and frequency modulation. Ahn et al. [3] proposed dynamically disabling the off-chip links of HMCs for reducing energy consumption. In addition, they also proposed a two-level prefetching mechanism to prevent extra main memory accesses. There are a couple of design space exploration works [4, 63] that analyzed performance, power and thermal behavior of 3D DRAMs. Khurshid et al. [34] attempt to reduce the maximum temperature and its variation by employing data compression at the memory controller of an HMC style 3D DRAM. In all these five papers, the architecture under consideration is 2.5D, which does *not* experience thermal stresses as much as our 3D stacked DRAM evaluated in this paper.

Access parallelism boosting proposals: Ding et al. [20] proposed a compiler-based technique to increase the performance of the system by enhancing bank-level and memory controller-level parallelism by identifying the memory banks that will be accessed for each cache miss. Jeong et al. [31] proposed a hardware technique to combine partitioning the internal memory banks between different cores with memory sub-ranking to increase performance and reduce power consumption. Tang et al. [61] explored a compiler/runtime based loop iteration scheduling strategy to improve bank-level parallelism for the applications that have irregular data access patterns. Lym et al. [44] proposed ERUCA, a hardware technique to improve the available parallelism in a DRAM chip by allowing efficient sub-banking and increasing the global-chip interconnect bandwidth. While all these proposals focused on boosting the access parallelism, they are *not* triggered by thermal-gradient. Consequently, these schemes do not consider the thermal-induced NUMA bottlenecks observed in 3D stacked DRAMs unlike our proposal.

8 CONCLUSION

This paper proposed and evaluated Data Convection, a novel approach that dynamically migrates data segments to address thermal-induced variable-retention-aware NUMA bottlenecks. We proposed three different incarnations of Data Convection and evaluated their effectiveness using 196 combinations of benchmarks in a system, with respect to a 3D+2.5D stacked DRAM baseline. Our results indicate that the three proposed data placement algorithms provide average performance improvements of 1.8%, 11.7%, and 14.4% over the 3D+2.5D stacked DRAM baseline with the default data placement. The paper also presented a detailed sensitivity study with a wide range of simulation parameters used in our evaluation.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous SIGMETRICS reviewers, for their constructive comments and feedback, and Kaveh Razavi, for shepherding the paper. The material presented in this paper is based upon work supported by the National Science Foundation under Grant Numbers 2119236, 2122155, 2028929, 1931531, and 1763681. The content of this paper is solely the responsibility of the authors and does not necessarily represent the official views of NSF.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

REFERENCES

[1] 2021. NVIDIA GPUs Have Hotspot Temperature Sensors Like AMD. https://www.techpowerup.com/forums/threads/nvidia-gpus-have-hotspot-temperature-sensors-like-amd.278606/

[2] Aditya Agrawal, Josep Torrellas, and Sachin Idgunji. 2017. Xylem: enhancing vertical thermal conduction in 3D processor-memory stacks. In *50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[3] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. 2014. Dynamic power management of off-chip links for hybrid memory cubes. In *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[4] Ahmed Al Maashri, Guangyu Sun, Xiangyu Dong, Vijay Narayanan, and Yuan Xie. 2009. 3D GPU architecture using cache stacking: Performance, cost, power and thermal analysis. In *IEEE International Conference on Computer Design*.

[5] AMD Inc. [n. d.]. The Polaris Architecture. https://www.amd.com/system/files/documents/polaris-whitepaper.pdf

[6] AMD Inc. 2017. Radeon RX Vega 64. https://www.amd.com/en/products/graphics/radeon-rx-vega-64

[7] AMD Inc. 2019. AMD RADEON VII. https://www.amd.com/en/products/graphics/amd-radeon-vii

[8] AMD Inc. 2020. HIGH BANDWIDTH MEMORY (HBM) JEDEC 235C. https://www.jedec.org/standards-documents/docs/jesd235a. [Online; accessed 15-April-2020].

[9] S. Baek, S. Cho, and R. Melhem. 2014. Refresh Now and Then. *IEEE Trans. Comput.* 63, 12 (2014), 3114–3126. https://doi.org/10.1109/TC.2013.164

[10] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*.

[11] Prasanna Balaprakash, Darius Buntinas, Anthony Chan, Apala Guha, Rinku Gupta, Sri Hari Krishna Narayanan, Andrew A Chien, Paul Hovland, and Boyana Norris. 2013. Exascale workload characterization and architecture implications. In *Proceedings of the High Performance Computing Symposium (HPC)*.

[12] Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. 2015. DRAM refresh mechanisms, penalties, and trade-offs. *IEEE Trans. Comput.* (2015).

[13] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H Loh, Don McCaule, Pat Morrow, Donald W Nelson, Daniel Pantuso, et al. 2006. Die stacking (3D) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[14] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu. 2014. Improving DRAM performance by parallelizing refreshes with accesses. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 356–367.

[15] Yi-Jung Chen, Chia-Lin Yang, Ping-Sheng Lin, and Yi-Chang Lu. 2015. Thermal/performance characterization of CMPs with 3D-stacked DRAMs under synergistic voltage-frequency control of cores and DRAMs. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems*.

[16] C. C. Chou, A. Jaleel, and M. K. Qureshi. 2014. CAMEO: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[17] Joachim Clabes, J. Friedrich, Mark Sweet, Jack DiLullo, Sam Chu, Donald Plass, James Dawson, Paul Muench, Larry Powell, Michael Floyd, Balaram Sinharoy, Mike Lee, Michael Goulet, James Wagoner, Nicole Schwartz, Stephen Runyon, Gary Gorman, P.J. Restle, Ronald Kalla, and J. Dodson. 2004. Design and implementation of the POWER5™ microprocessor. 670–672. https://doi.org/10.1109/DAC.2004.240518

[18] Ayse Coskun, David Atienza, Mohamed Sabry, and Jie Meng. 2011. Attaining single-chip, high-performance computing through 3D systems with active cooling. *IEEE Micro* 31, 4 (2011), 63–75.

[19] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*.

[20] Wei Ding, Diana Guttman, and Mahmut Kandemir. 2014. Compiler support for optimizing memory bank-level parallelism. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, United Kingdom) *(MICRO-47)*. IEEE Computer Society, USA, 571–582. https://doi.org/10.1109/MICRO.2014.34

[21] Ronald G Dreslinski, David Fick, Bharan Giridhar, Gyouho Kim, Sangwon Seo, Matthew Fojtik, Sudhir Satpathy, Yoonmyung Lee, Daeyeon Kim, Nurrachman Liu, et al. 2013. Centip3de: A 64-core, 3d stacked near-threshold system. *IEEE Micro* 33, 2 (2013), 8–16.

[22] David Fick, Ronald G Dreslinski, Bharan Giridhar, Gyouho Kim, Sangwon Seo, Matthew Fojtik, Sudhir Satpathy, Yoonmyung Lee, Daeyeon Kim, Nurrachman Liu, et al. 2012. Centip3De: A 3930DMIPS/W configurable near-threshold 3D stacked system with 64 ARM Cortex-M3 cores. In *2012 IEEE International Solid-State Circuits Conference*. IEEE, 190–192.

[23] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. In *Abstracts of the 2019 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems* (Phoenix, AZ, USA) *(SIGMETRICS '19)*. Association for Computing Machinery, New York, NY, USA, 93. https://doi.org/10.1145/3309697.3331482

[24] Yongkui Han, Israel Koren, and C. Krishna. 2007. TILTS: A Fast Architectural-Level Transient Thermal Simulation Method. *J. Low Power Electronics* 3 (04 2007), 13–21. https://doi.org/10.1166/jolpe.2007.106

[25] Syed Minhaj Hassan and Sudhakar Yalamanchili. 2016. Understanding the impact of air and microfluidics cooling on performance of 3d stacked memory systems. In *Proceedings of the Second International Symposium on Memory Systems*. 387–394.

[26] HBM. [n. d.]. High Bandwidth Memory (HBM) DRAM. https://www.jedec.org/standards-documents/docs/jesd235a

[27] HBM. 2015. High-Bandwidth Memory (HBM) reinventing memory technology. https://www.amd.com/system/files/documents/high-bandwidth-memory-hbm.pdf

[28] HMC. [n. d.]. Hybrid Memory Cube Consortium. http://www.hybridmemorycube.org/

[29] Wei Huang, Mircea R. Stan, Kevin Skadron, Karthik Sankaranarayanan, Shougata Ghosh, and Sivakumar Velusam. 2004. Compact thermal modeling for temperature-aware design. In *Proceedings of the 41st Annual Design Automation Conference*.

[30] JEDEC. [n. d.]. JEDEC publishes Wide I/O 2 mobile DRAM standard. https://www.jedec.org/news/pressreleases/jedec-publishes-wide-io-2-mobile-dram-standard

[31] Min Kyu Jeong, Doe Hyun Yoon, Dam Sunwoo, Mike Sullivan, Ikhwan Lee, and Mattan Erez. 2012. Balancing DRAM locality and parallelism in shared memory CMP systems. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA '12)*. IEEE Computer Society, USA, 1–12. https://doi.org/10.1109/HPCA.2012.6168944

[32] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. 2010. CHOP: adaptive filter-based DRAM caching for CMP server platforms. In *In proceedings of The Sixteenth International Symposium on High-Performance Computer Architecture (HPCA)*. 1–12.

[33] Adwait Jog, Onur Kayiran, Tuba Kesten, Ashutosh Pattnaik, Evgeny Bolotin, Niladrish Chatterjee, Stephen W. Keckler, Mahmut T. Kandemir, and Chita R. Das. 2015. Anatomy of GPU memory system for multi-application execution. *1st International Symposium on Memory Systems (MEMSYS)* (2015).

[34] Mushfique Junayed Khurshid and Mikko Lipasti. 2013. Data compression for thermal mitigation in the hybrid memory cube. In *IEEE 31st International Conference on Computer Design (ICCD)*.

[35] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: a fast and extensible DRAM simulator. *IEEE Comput. Archit. Lett.* (2016).

[36] J. B. Kotra, S. Kim, K. Madduri, and M. T. Kandemir. 2017. Congestion-aware memory management on NUMA platforms: A VMware ESXi case study. In *IEEE International Symposium on Workload Characterization (IISWC)*.

[37] Jagadish B. Kotra, Narges Shahidi, Zeshan A. Chishti, and Mahmut T. Kandemir. 2017. Hardware-Software co-design to mitigate DRAM refresh overheads: a case for refresh-aware process scheduling. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[38] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir. 2018. CHAMELEON: a dynamically reconfigurable heterogeneous memory system. In *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[39] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: enabling energy optimizations in GPGPUs. In *Proceedings of the 40th Annual International*

Symposium on Computer Architecture.

[40] M. Liu and T. Li. 2014. Optimizing virtual machine consolidation performance on NUMA server architecture for cloud workloads. In *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*.

[41] Gabriel H Loh. 2008. 3D-stacked memory architectures for multi-core processors. In *ACM SIGARCH computer architecture news*.

[42] Gian Luca Loi, Banit Agrawal, Navin Srivastava, Sheng-Chih Lin, Timothy Sherwood, and Kaustav Banerjee. 2006. A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy. In *Proceedings of the 43rd annual Design Automation Conference*.

[43] Jieyi Long, Seda Ogrenci Memik, Gokhan Memik, and Rajarshi Mukherjee. 2008. Thermal monitoring mechanisms for chip multiprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)* 5, 2 (2008), 1–33.

[44] S. Lym, H. Ha, Y. Kwon, C. Chang, J. Kim, and M. Erez. 2018. ERUCA: efficient DRAM resource utilization and resource conflict avoidance for memory system parallelism. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 670–682.

[45] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi. 2017. Dynamic GPGPU power management using adaptive model predictive control. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 613–624.

[46] Seda Ogrenci Memik, Rajarshi Mukherjee, Min Ni, and Jieyi Long. 2008. Optimizing Thermal Sensor Allocation for Microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 3 (2008), 516–527. https://doi.org/10.1109/TCAD.2008.915538

[47] Jie Meng, Katsutoshi Kawakami, and Ayse K Coskun. 2012. Optimizing energy efficiency of 3-D multicore systems with stacked DRAM under power and thermal constraints. In *Proceedings of the 49th Annual Design Automation Conference (DAC)*.

[48] Micron. [n. d.]. DDR4 SDRAM. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/8gb_ddr4_sdram.pdf

[49] Makoto Motoyoshi. 2009. Through-silicon via (TSV). *Proc. IEEE* (2009).

[50] Lifeng Nai, Ramyad Hadidi, He Xiao, Hyojong Kim, Jaewoong Sim, and Hyesoon Kim. 2018. CoolPIM: Thermal-Aware Source Throttling for Efficient PIM Instruction Offloading. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 680–689. https://doi.org/10.1109/IPDPS.2018.00077

[51] Prashant Nair, Chia-Chen Chou, and Moinuddin K. Qureshi. 2013. A case for Refresh Pausing in DRAM memory systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 627–638. https://doi.org/10.1109/HPCA.2013.6522355

[52] NVIDIA Corporation. 2011. CUDA C/C++ SDK Code Samples.

[53] NVIDIA Corporation. 2017. NVIDIA TITAN V. https://www.nvidia.com/en-us/titan/titan-v/

[54] Jong S. Park. U.S. Patent 5 583 823, Dec. 1996. Dram refresh circuit.

[55] Indrani Paul, Srilatha Manne, Manish Arora, W Lloyd Bircher, and Sudhakar Yalamanchili. 2013. Cooperative boosting: needy versus greedy power management. In *ACM SIGARCH Computer Architecture News*.

[56] Kiran Puttaswamy and Gabriel H Loh. 2007. Thermal herding: microarchitecture techniques for controlling hotspots in high-performance 3D-integrated processors. In *IEEE 13th International Symposium on High Performance Computer Architecture*.

[57] Karthik Rao. 2018. *Coordinated management of the processor and memory for optimizing energy efficiency*. Ph. D. Dissertation. Georgia Institute of Technology. https://smartech.gatech.edu/handle/1853/60234

[58] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. 2016. VDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 18, 13 pages.

[59] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim. 2014. Transparent hardware management of stacked DRAM as part of memory. In *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[60] Chong Sun, Li Shang, and Robert P. Dick. 2007. Three-dimensional multiprocessor system-on-chip thermal optimization. In *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 117–122.

[61] Xulong Tang, Mahmut Kandemir, Praveen Yedlapalli, and Jagadish Kotra. 2016. Improving bank-Level parallelism for irregular applications. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 57, 12 pages.

[62] T. Vijayaraghavan, Y. Eckert, G. H. Loh, M. J. Schulte, M. Ignatowski, B. M. Beckmann, W. C. Brantley, J. L. Greathouse, W. Huang, A. Karunanithi, O. Kayiran, M. Meswani, I. Paul, M. Poremba, S. Raasch, S. K. Reinhardt, G. Sadowski, and V. Sridharan. 2017. Design and Analysis of an APU for Exascale Computing. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[63] Christian Weis, Igor Loi, Luca Benini, and Norbert Wehn. 2013. Exploration and optimization of 3-D integrated DRAM subsystems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2013).

[64] Gene Y. Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)* (2015), 564–576.

[65] Tiansheng Zhang, Jie Meng, and Ayse K Coskun. 2015. Dynamic cache pooling in 3D multicore processors. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* (2015).

[66] Yuang Zhang, Li Li, Axel Jantsch, Zhonghai Lu, Minglun Gao, Yuxiang Fu, and Hongbing Pan. 2015. Exploring stacked main memory architecture for 3D GPGPUs. In *IEEE 11th International Conference on ASIC (ASICON).*

[67] Jishen Zhao, Guangyu Sun, Gabriel H Loh, and Yuan Xie. 2013. Optimizing GPU energy efficiency with 3D die-stacking graphics memory and reconfigurable memory interface. *ACM Transactions on Architecture and Code Optimization (TACO)* (2013).