Towards Application-Specific Address Mapping for Emerging Memory Devices

Shashank Adavally
University of North Texas

Krishna Kavi
University of North Texas

ABSTRACT

Recent advancements in 3D-stacked DRAM such as hybrid memory cube (HMC) and high-bandwidth memory (HBM) promise higher bandwidth and lower power consumption compared to traditional DDR-based DRAM. However, taking advantage of this additional bandwidth for improving the performance of real-world applications requires carefully laying out the data in memory which incurs significant programmer effort. To alleviate this programmer burden, we investigate application-specific address mapping to improve performance while minimizing manual effort. Our approach is guided by the following insights: (i) toggling activity of address bits can help determine strategies to improve parallelism within memory but this metric underestimates conflicts and (ii) modern memory controllers reorder address requests and therefore any toggling activity measured from an address trace is non-deterministic. Furthermore, our position is that analyzing *individual* address bits results in poor estimates for actual conflicts and exploited parallelism and that entropy needs to be calculated for *groups* of address bits. Therefore, we calculate window-based probabilistic entropy for groups of address bits to determine a near-optimal address mapping. We present simulation results for ten applications that show a performance improvement up to 25% over fixed address-mapping and up to 8% over previous application-specific address mapping for our proposed approach.

ACM Reference Format:
Shashank Adavally and Krishna Kavi. 2020. Towards Application-Specific Address Mapping for Emerging Memory Devices. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3422575.3422785

1 INTRODUCTION

High-bandwidth memory (HBM) offers much higher bandwidth than traditional DRAM. For instance, DDR5 provides up to 51.6 GB/s per module [13], while HBM2e provides up to 640 GB/s per stack [20]. However, extracting this additional bandwidth for real applications can be challenging. One possible reason is that the memory requests from a multicore CPU are often not distributed optimally across the HBM resources causing conflicts for channels, ranks, banks or rows.

One approach to maximize memory system performance for high-performance computing codes is to manually layout the data structure or reorder computational loops. Kripke [1], for instance, performs a sweep over many possible data layouts and loop ordering to find optimal layouts for memory accesses. This approach leads to bloated code bases that can be be hard to debug and maintain.

We posit that there is a better approach to achieving the same performance benefits—by making the mapping of address requests to HBM structures programmable, exposing it to the superuser by some mechanism, and developing a methodological approach to determining the optimal address mapping for a given application. Previous works have shown that application-specific address mappings are possible and that they can provide up to 16% performance improvement. However, they require tedious manual evaluation and the insights were based on a limited set of applications [4]. In this work, we leverage their insights and design a rigorous approach to determine *near optimal* address mapping for each application. Furthermore, our position is that analyzing *individual* address bits results in poor estimates for actual conflicts and exploited parallelism and that entropy needs to be calculated for *groups* of address bits.

Contributions. Towards supporting our position, we make the following contributions in this paper.

- We propose a new metric to determine optimal address mapping from an address trace by considering address bits in groups rather than individually. The proposed metric estimates parallelism and conflicts more accurately than previous entropy metrics.
- We present a methodology to determine application-specific address mapping. In our methodology, we hierarchically resolve mapping address bits to channels, columns, banks, bank groups, and rows (in that order) making use of our proposed metric in conjunction with metrics from past research.

We present experimental results of ten applications from high-performance computing and graph processing that demonstrate the performance improvements for our approach.

Findings. Application-specific address mapping based on *entropy* of individual bits results in 5% performance improvement on average over a baseline application-agnostic address mapping. Our approach which determines entropy of groups of address bits results in 8% performance improvement on average over the baseline.

Organization. Section 2 provides pertinent background information and Section 3 presents our approach. Our experimental setup is described in Section 4. We present our results in Section 5, discuss related work in Section 6, and conclude in Section 7.

MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA

ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422785

[1] https://github.com/LLNL/Kripke

2 BACKGROUND

In this section, we provide background on the internal organization of HBM and other pertinent information.

2.1 HBM Organization

Figure 1 shows an example 4-layer high-bandwidth memory (HBM) stack and its organization. The HBM stack is composed of a buffer die and 4 DRAM dies. Each DRAM die supports 2 independent channels for a total of 8 physical channels, each with an independent 128-bit interface to the host processor. The channels are clocked independently and are the primary source of parallelism within the memory subsystem. Each channel is composed of a set of 16 banks organized into 4 bank groups of 4 banks each. Each bank is further sub-divided into a number of DRAM rows and columns. In some cases, a physical channel may be subdivided into two pseudo-channels to save energy and improve performance for some access patterns.

An application's performance can be improved by carefully optimizing the data layout and choosing an optimal address mapping which maps a physical address to DRAM structures (channels, bank groups, banks, rows, and columns). Our hypothesis is that, for memory-sensitive applications, choosing the right address mapping can result in performance improvements comparable to careful data layout and loop ordering optimizations. We posit that an near-optimal address mapping can be identified automatically based on statistical techniques and without a knowledge of the algorithm. We expect that this technique is more productive than techniques that manually changes data layouts and/or reorder loops. In this paper, we are concerned with choosing an near-optimal mapping for a given application.

2.2 Row-buffer Locality and Bank-Level Parallelism

An application's performance can be improved by exploiting *row-buffer locality* and *bank-level parallelism* offered by a given HBM's organization.

Row-buffer Locality. Accessing data from the memory typically involves: (i) *opening* a row (or DRAM page) by issuing an `ACT` command, (ii) reading or writing one column worth of data via `RD/WR` command, and (iii) in some cases, *closing* a row by issuing a `PRE` command. In a standard HBM2 organization, activating a row fetches 2 KB-wide [14] data into a row buffer, from which data can be accessed in column width-sized chunks, with the width typically matching the size of a CPU cache line (e.g., 64B). Accesses to an already open row are faster as they avoid the latency incurred from `PRE` and `ACT` command. Applications exhibiting high spatial locality benefit from this wider row size relative to cache line and are said to exploit row-buffer locality. Back-to-back accesses to the same bank are serialized and to exploit parallelism, the accesses must be spread across banks.

Bank-Level Parallelism. While the banks can operate in parallel, there are a number of restrictions imposed on them due to some shared structures as well as current-draw limitations. Banks within a bank group share a local data bus and the bank groups within a channel share a global data bus.

Parallel data transfers to different banks in the same group are serialized as they share a narrow bus. However, accesses to different banks in different bank groups can be overlapped as they only share the global data bus. Banks within a bank group can partially operate in parallel. For instance, data transfer between bank A and the host processor can be overlapped with opening or closing of rows in bank B via `ACT` or `PRE` commands. That said, in a typical HBM, only four `ACT` command can be issued in a time window known as t_{FAW} (four-activation window) which further limits parallelism across banks.

2.3 Address Mapping

Given how HBM is organized, one may strive to improve bank-level parallelism by spreading out one regular stream of accesses across bank A of the four different bank groups and another stream of accesses across bank B of another bank group. It is difficult to analyze real application code to achieve this type of parallelism. Although, modern memories use permutation-based mapping, it is primarily focused on reducing row-conflicts. In permutation-based mapping, least significant bits of row are XORed with the bank's address bits to generate a new bank ID. So, it tries to change the bank ID whenever there is a change in row ID to prevent a row conflict. It is intended to reduce row conflicts for certain strided pattern. In real applications, the access patterns are more complex and an automated approach in determining the near-optimal address mapping strategy can help exploit both bank-level parallelism and row-buffer locality.

It is unreasonable to expect the programmer to understand the nuances of memory organization to properly layout their data in memory. To complicate matters further, when running applications in parallel, it can be hard to reason about access patterns seen by the memory controller since requests come from many different threads, resulting in an access pattern that was not originally intended by the programmer. Therefore, it becomes necessary to adopt an algorithm/code agnostic technique to determine the near-optimal address mapping for an application. The potential uplift in performance can be high — up to 60% performance improvement has been observed in some cases [19].

3 METHODOLOGY AND APPROACH

In this section, we describe our high-level methodology and proposed approach to determine a near-optimal application-specific address mapping.

3.1 Methodology Overview

Figure 2 summarizes our methodology to determine an application-specific address map. At a high-level, we run the application on a reference multi-core CPU. We collect memory address traces via a PIN tool [17]. From the address trace, we calculate three different statistical metrics — bit-flip count, bit-flip probability, and repetitive counter — which are measures of exploitable parallelism and toggle activities in address bits. Using a combination of these three metrics, we determine the near-optimal address mapping using the approach presented in Section 3.3. The metrics themselves and their limitations when used individually are described

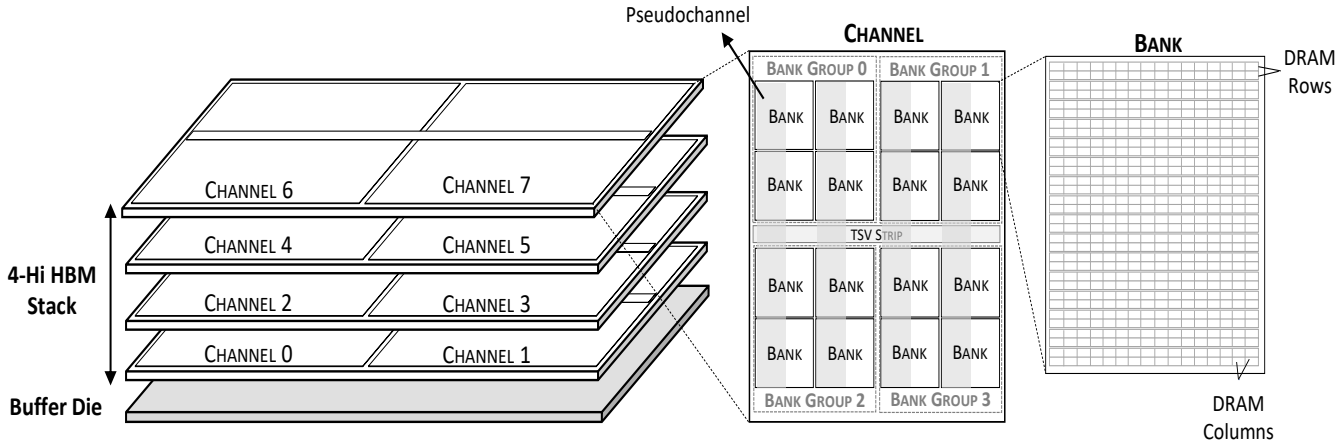

Figure 1: HBM Organization. Memory spans multiple layers of DRAM dice and is organized hierarchically into channels, banks, rows, and columns. Banks are grouped and channels are sub-divided into pseudo-channels.

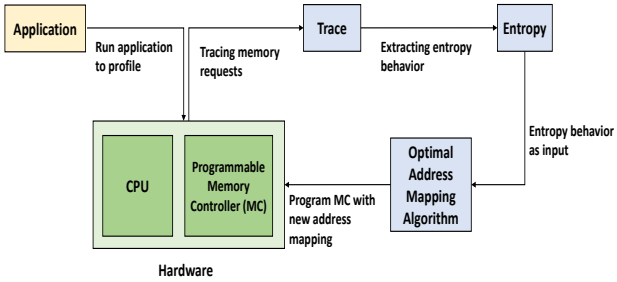

Figure 2: High level steps to determine near-optimal address mapping

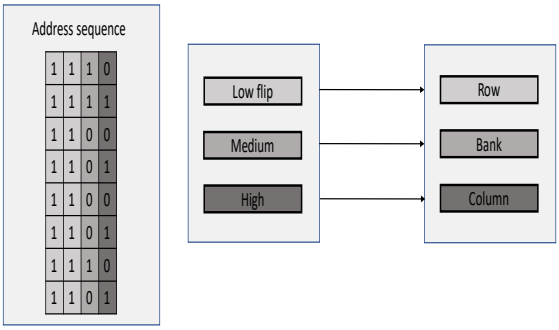

Figure 3: Based on the bit flip intensity, respective bits are mapped to a specific abstraction layer

in Section 3.2. Finally, we program the memory controller with the calculated near-optimal address mapping.

3.2 Statistical Metrics

In our approach, we introduce a statistical metric which we term "repetitive counter" to aid in the selection of an near-optimal address mapping. We use this metric in conjunction with two other statistical metrics previously deviced by other researchers from — termed "bit-flip entropy"[9] and "bit-flip probability"[16]. We note that each of these metrics only address some aspects of address bit entropies, but collectively these metrics better measure the randomness of memory addresses accessed by applications.

Entropy measures the frequency of an address bit flip (changing from zero to one or one to zero). It helps to determine the optimum address bits to use to identify rows, banks, columns etc. For example, Figure 3 shows a sequence of addresses in the left-most block and each of the bit positions have been color-coded based on the bit flip rate: bits with high flip rates are denoted with darker color, bits with medium flip rates are marked with medium and bits with low flip rate are denoted in lighter color. The intention is to map least flipping bits to rows to reduce row conflicts and bits with high flip rate to columns, banks etc to increase the spatial locality and parallelism.

Bit-Flip Entropy. Figure 4 shows 4 consecutive address requests to memory. Consider bit 2 as highlighted. The behaviour of each bit position is obtained by calculating the bit-flip rate. The final flip count is shown at the bottom. Per the figure, bits with low flip rate (i.e. bit flip count with 0 and 1) are used to identify rows (to reduce the row conflicts by mapping to bits that flip less frequently), bits with medium flip rate are used to identify banks and bits with high flip rates are used to identify columns (to increase spatial locality and bank-level parallelism).

Bit-Flip Probability. Figure 5 shows the same address sequence, but shows the Bit Value Ratio (BVR). For example for bit 4, the BVR is 0.75 since this bit has a value of one 75% of the time. Then bit flip probability is calculated using BVR ratios when the bit is zero and one, as min(BVR, 1-BVR). Based on the frequency of the bit flip rate, address bits are mapped to different memory structures. It should be noted that entropy is calculated based on a sliding-window fashion to negate the entropy variance due to the requests reordering in the memory controller. This method produces much precise behaviour of bit-flip. Since this technique presents more accurate entropy information for multi-threaded workloads, we used this method in our study.

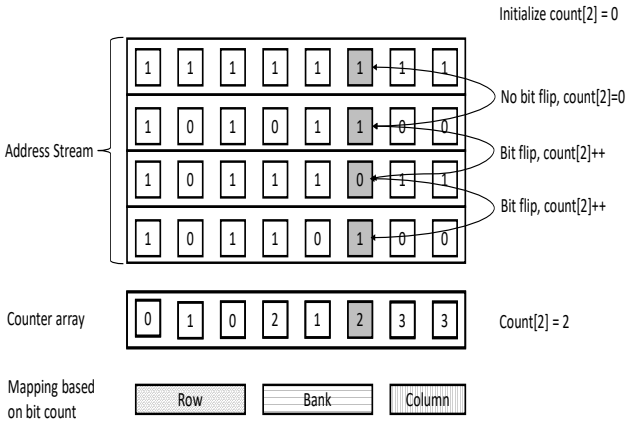

Figure 4: Bit flip count method showing the final bit flip count in the bottom for each address bit of the given address stream [9]

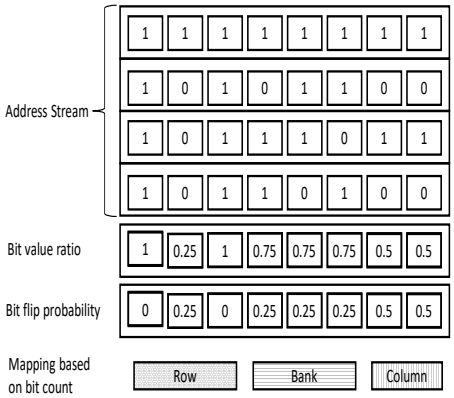

Figure 5: Bit Probability method describing bit flip probability of the given address stream [16]

3.3 Proposed Approach

Before we describe our methodology, we will illustrate an existing work [9] with another example. Figure 6 shows a sequence of 8 addresses with bit flip counts shown at the bottom. For simplicity, we are assuming that the memory has only 1 channel, 0 bank groups, 8 Banks (similar to [9] configuration). We are required to map 8 address bits to rows, columns and banks. As proposed in [9], (and shown in Table 1), bits with (medium) bit-flip count of 2 (viz., bits 0, 2, 4) are used to identify banks, bits with (high) bit-flip count of 4 are used to identify columns and the remaining bits are used to identify rows. The main focus is to reduce the number of row conflicts. With today's HBM technology (comprising multiple channels, bank groups and banks), implementing this technique for address mapping resulted in performance improvement. However, based on a simple observation as discussed in Section 5, we can determine that this technique may not achieve optimal address mapping (as it may cause conflicts in banks). Also, with the availability of parallel structures such as channels, bank groups and banks, it would be

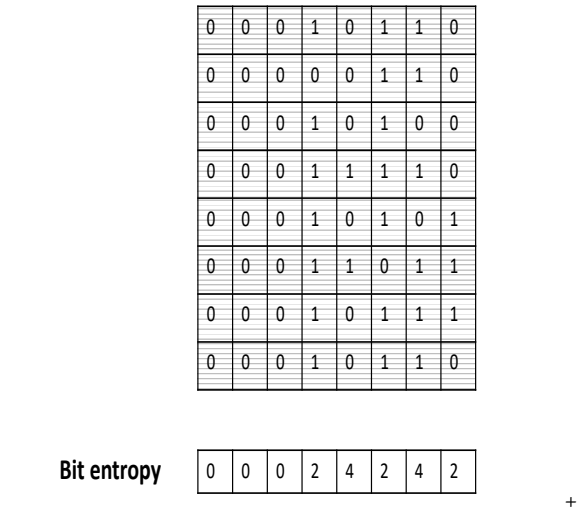

Figure 6: Sample address stream and its bit flip behaviour

beneficial to utilize them in parallel; that is, improve Channel level parallelism and Bank parallel utilization (BPU). [2]

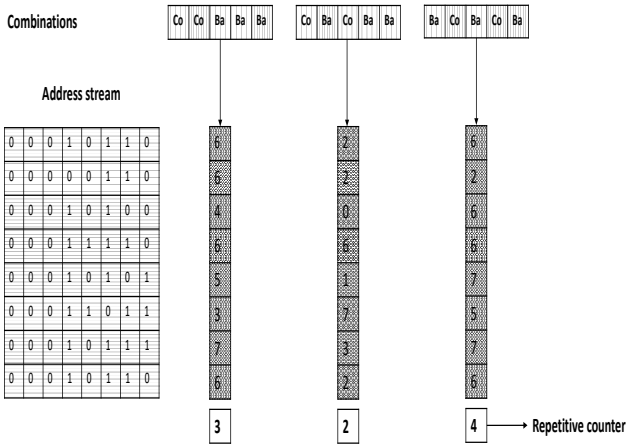

Figure 7: Methodology

We now introduce our technique that leads to a near-optimal address mapping, so that requests are more evenly distributed among channels, bank groups and banks. To simplify the explanation of our technique, we *assume* that the memory is configured with a single channel and has only banks (and no bank groups). As a first step, we consider high entropy bits, and explore all possible permutations of using these bits to identify banks and columns (not rows). For example, from Figure 6, we identified 5 bits with high entropy (viz., bits 0-4): for our example DRAM, we need two bits for column ID and three bits for bank ID. Thus we can explore all 60 permutations of mapping these 5 high entropy bits for column and

[2]BPU is quantified as the average number of banks in main memory that are being used concurrently [10]

bank addresses. We only show three possible permutations (out of possible 60) in Figure 7. For each possible mapping, in a sliding window fashion among memory requests, with a window size of 8 (since there are 8 banks), we track unique banks accessed in a window. In the figure 7, the array under each permutation shows the bank ID used for the specific address. At the end of address trace, for each possible mapping we compute a "repetitive count". A repetitive count indicates, for a given mapping how many times a bank is being used by multiple addresses. For example the right-most address mapping results in 4 addresses mapped to bank 6 (i.e., repetitive count of 3) and 2 addresses mapped to bank 7 (i.e., repetitive count of 1), giving a net repetitive count of 4. The permutation with least repetitive count is considered the near-optimal mapping configuration since it minimizes conflicts to banks (similarly for channels or bank groups) and improve parallelism. So, out of the 3 permutations shown in figure 7, second permutation has the least repetitive count for the example address trace, indicating that banks are being used in higher degree of bank level parallelism. So bits 0, 1 and 3 are used to identify banks and remaining bits 2 and 4 are used to identify columns.

Conventional address mappings try to preserve spatial locality so that consecutive addresses are mapped to the same row (least significant bits are used for column selection). However such an assignment does not benefit from bank and channel parallelism available. In our method, we select n bits with high entropy, where n is the number of bits needed for selecting channels, bank groups, banks and columns.

Algorithm 1: Near-Optimal Address Mapping

```
1:  procedure SelectMap(trace, map)
2:      Calculate bit flip count from trace
3:      Sort and extract indices of n highest values
4:      Permute all (nCc) combinations for channel assignment
5:      For each permutation calculate repetitive counter value
6:      Assign bits with the lowest repetitive counter to channels
7:      Assign bits with the next highest bit flip count to columns
8:      Permute remaining combinations for bank group
        assignment
9:      Identify optimum bank group assignment
10:     Permute remaining high entropy bits for bank assignment
11:     Identify optimum bank assignment via repetitive counter
```

Algorithm 1 outlines our method. Let us consider a realistic configuration of HBM that consists of 8 channels (i.e., 3 bits for channel address), 4 bank groups (i.e. 2 bits for bank group identification), 4 banks per group (2 bits for bank address) and 32 columns per row (5 bits for column address; each column consists of 64 bytes). We assume that each row in a bank contains 2KB. For such HBM system we select n=12 bits with highest entropy. We start with all possible permutations of 3 bits needed for channel address. Once we identify the 3 bits that result in least repetitive count (as explained in previous paragraphs) and are set aside for channel address. These bits are excluded from further consideration. We then set aside 5-bits with highest entropy as column address. We then repeat selecting optimal permutations (resulting in least repetitive count) for selecting bits to identify bank groups and then for selecting banks. The remaining address bits are used for row address. Our methodology identifies bits for the memory structures (channels, bank groups, banks, columns and rows) to maximize memory parallelism.

One of the limitations with previously reported works is that bit entropy alone does not lead to optimal address mapping since such a mapping can still limit potential memory parallelism. Entropy explains the bit flip behaviour but not the bit flip rate with respect to other bits. More details will be discussed in section 5.

Table 1: Simulation Setup

Configuration	Description
Core configuration	128-entry instruction window, 4 wide issue
Core count	8
Core frequency	3.2 GHz
Caches	L1 32kB 8-way set associative L2 128kB 8-way set associative L3 8MB 16-way set associative
Memory type	HBM 1Gbps
Memory channels	8
Memory size	4 GB
Memory controller queue size	Readq-32; Writeq-32
Scheduling Policy	FR-FCFS-Cap [18]
Baseline Memory Mapping	SK Hynix GDDR5 [1] (RoBaCoBaChCo)

4 EXPERIMENTAL SETUP

In this section, we describe the experimental setup used to evaluate our address mapping technique. We describe the simulation setup used to evaluated the effectiveness of our technique, the workloads used in this study and the technique to collect address traces.

4.1 Simulation Setup

We used Ramulator [14] for our simulations. The system configurations used are listed in Table 1. We assumed a realistic multi-core configuration with 128-entry instruction window, 4-wide issue running at 3.2 GHz. Cores are supported by 3 levels of caches of different sizes. We selected our memory type to be HBM with multiple levels of memory hierarchies like channels, banks, bank groups, rows and columns with a memory size of 4GB and optimal scheduling policies. We assume a memory mapping similar to SK Hynix GDDR5 (RoBaCoBaChCo) as our baseline.

We modified Ramulator to support and maintain OpenMP multi-core ordering of memory requests (using time stamps with memory traces as mentioned in section 4.2). Ramulator allocates all physical pages randomly but in a real system, some portion of the pages are allocated consecutively. So, we modified Ramulator such that the address translation results in 40 percent consecutive pages and 60 percent random pages to mimic operating system allocation.

4.2 Workloads

A wide variety of workloads have been considered in our study, including from Rodinia [8], graph processing [12], LLNL [15], HPCG [3], GAP [6] suites and different micro benchmarks. More details of the workloads can be found in Table 2. For each benchmark, we captured 1 billion instructions using modified Intel PIN tool [17] and replayed them through Ramulator. Our modified PIN tool captures actual CPU cycle-stamps along with other necessary data. So that the address traces are replayed in the specific order of the cycle-stamp to maintain the correct address ordering for multi-threaded programs.

Table 2: Benchmarks description

Benchmarks	Label	Description
Sequential access	SEQ	Vector Addition
Strided access	STR	Stride Vector addition
Random-10	R10	Vector addition with 10% of elements accessed randomly
Random-40	R40	Vector addition with 40% of elements accessed randomly
Random-70	R70	Vector addition with 70% of elements accessed randomly
Hotspot	HS	Temperature simulation tool from Rodinia
Kripke	KR	Deterministic particle transport code
Breadth First Search	BFS	Graph traversal workload
High-Performance Conjugate Gradient	HPCG	SpMV and dot product
Triangle Counting	TC	Order invariant with possible relabelling

5 RESULTS AND ANALYSIS

From Figure 8, it can be observed that most of the workloads are benefiting from our proposed technique except BFS. Our technique generates near-optimal mapping that distributes the memory requests across the channels, banks groups and banks. One of the key differences we notice is the reduction in row conflicts compared to the baseline and count-based entropy mapping. It should be noted that the strided access workloads have higher improvements over the baseline. This is because in strided workloads, the spacial locality is minimal and leads to higher row conflicts with sub-optimal address mappings. With three types of synthetic micro-benchmarks, R10, R40, R70 (with different randomized memory accesses), it can be seen that the difference in performance improvement between bit-flip count-based entropy technique and our method is decreased. This can be explained as the randomness in the application increases, performance difference between near-optimal and non-optimal mapping techniques decreases because the random requests cannot be guaranteed more even distribution across channels, bank groups and banks. For benchmark BFS, our methodology has lower row conflicts but smaller Bank Parallel utilization (BPU). Figure 9 shows the reduction of row conflicts compared to baseline. Using the proposed technique, row conflicts have improved except for R70. We have seen 2% increase in row conflicts for R70 but with equal distribution of requests among banks and channels, overall performance has improved by 7%. In General, we observed over 16% reduction in row conflicts on average compared to the baseline.

Now let us return to the issue we raised in section 3.3. Figure 10 shows a stream of addresses and bit flip counts shown at the bottom. It can be seen that the LSB portion of the bits have smaller entropy when compared to MSB portion. As explained previously, LSB bits are used to map to rows to reduce row conflicts and MSB bits to columns to increase locality or to banks to increase bank level parallelism. But the behaviour of each requests' LSB bits (3 bits) is seen to be varying for each address request. As per suggested mapping suggested in [9], had these bits been used to identify a row, each of the request would create a row conflict. With our technique, selection of the mapping bits is done based on the load distribution, eliminating this problem.

5.1 Case Study: Kripke Application

Apart from regular compute workloads, this technique can be implemented for HPC application codes. We considered Kripke [15] as an initial scientific example. One of the main goals with Kripke is to investigate how different data layouts effect performance. Kripke supports 6 layouts using Directions(D). Groups (G) and Zones (Z), and we explored a relation between data layouts with different address mappings. Figure 12 shows the overall performance of different layouts running different workload configurations. It can be seen that the best performing layout is different for each workload size configuration. We can conclude that an optimal layout has to be determined for each configuration to achieve the best performance. Since Kripke is a 3-dimensional application, it is simpler to sweep all layout possibilities across the loops to calculate optimal layout. But this may not be a viable technique to determine optimal layout for more complex workloads like 6-dimensional applications. Instead of estimating the optimal mapping for each data layout, we calculate a near-optimal mapping of one layout for a workload configuration to improve the performance without modifying the layout but changing the mapping of addresses to different HBM structures.

5.2 Entropy Persistence

We wanted to explore if the computed entropy remains consistent across different runs for a given application. This is because as the system ages, the virtual-to-physical address mapping changes and we verify the amount of variance this could introduce in the calculated entropy. Such a study was not undertaken by others. Since the proposed technique uses physical addresses, it is also important to investigate the consistency of physical address bits across multiple runs. For this purpose, we collect physical address traces using "kpageflags" and "pagemap" files in linux, which provide necessary information on physical pages [2], and capture physical addresses through modified PIN tool (traces the physical addresses that are computed using the information provided by "kpageflags" and "pagemap" files) and replay them on Ramulator.

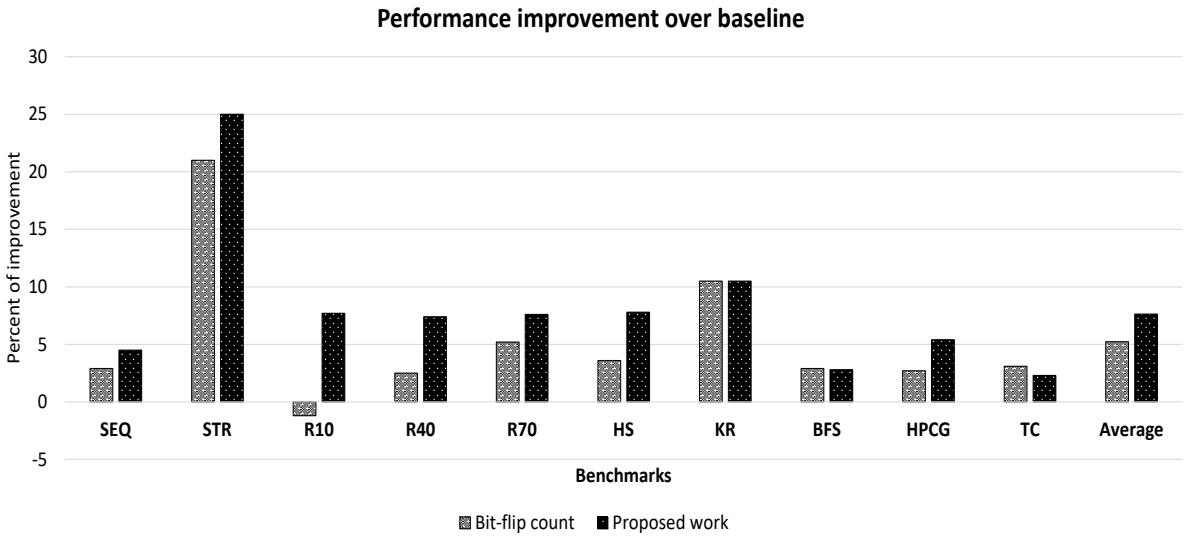

Figure 8: Performance results comparing past application-specific technique [9] and our proposed approach against a static baseline. Our proposed approach improves performance up to 25% over the static baseline and up to 8% over previous approaches [9].

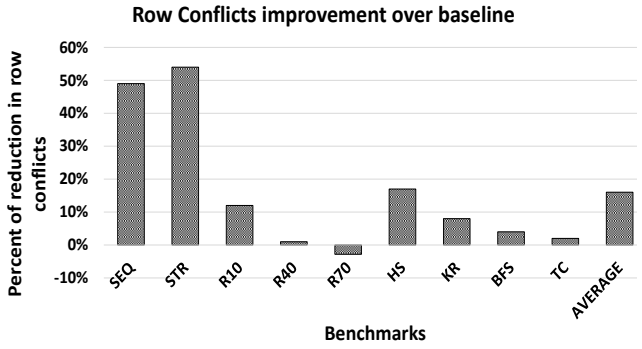

Figure 9: Row conflicts improvements the static baseline. Our proposed approach reduces row conflicts by 16% over the static baseline.

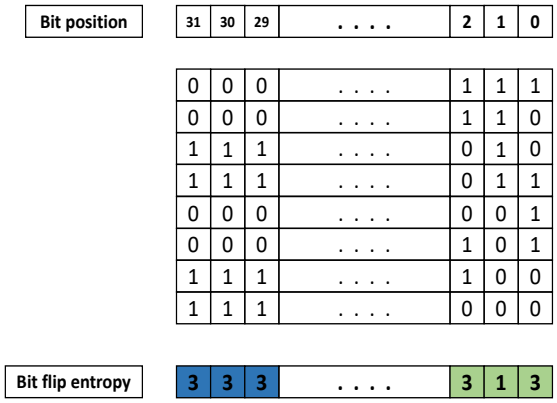

Figure 10: Observed limitation in generating optimal mapping by depending solely on entropy

We compared the change in performance of near-optimal address mapping achieved by our technique over multiple runs, each with potentially different physical addresses assigned to application. This process of collection of traces, simulation and compute near-optimal address mapping is repeated for multiple runs of each application to validate the stability of entropy across possibly different physical address assignments.

Figure 11 shows the bit-flip probability of hotspot benchmark for each address bit. Y-axis shows bit-flip ratio (i.e. higher the value, higher the probability that the bit flips) for each address bit (X-axis). This figure includes bit-flip probability data from three different runs of the benchmark and, measuring the entropy for each run. The entropy for address bits is similar for most of the workloads but in some some workloads, the entropy for Most Significant bits (MSB) tends to differ slightly (it can also be seen in Figure 11). One of the reason could be the change in locality of the physical pages across multiple runs could have led to higher variation of MSB bits. Overall, we observe that the average entropy variation is as low as 1% and at most as high as 6%. Although the variation in entropy may reach 6%, it is limited to specific range of bits (i.e. MSB bits 25-31). We also observe that these variations do not lead to significant changes in the near-optimal address mapping configurations. And performance variations between these different mappings over multiple runs is small (between 1%-4%). This supports that entropy behaviour may show small variations between different runs, but the impact on the overall performance using our address mapping

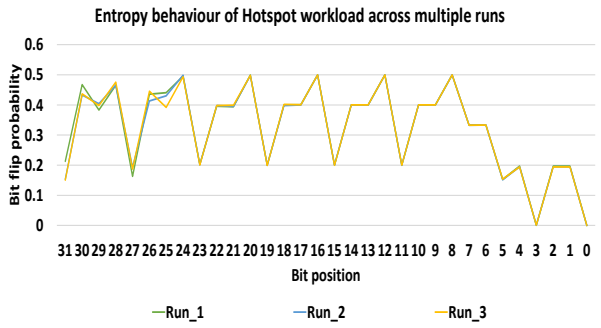

Figure 11: Entropy consistency over multiple runs

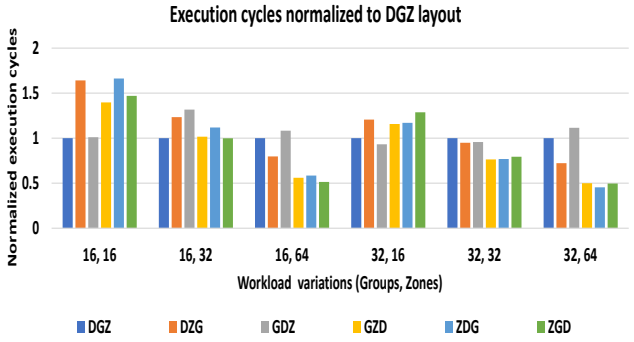

Figure 12: Multiple Layout performance under different workload configurations

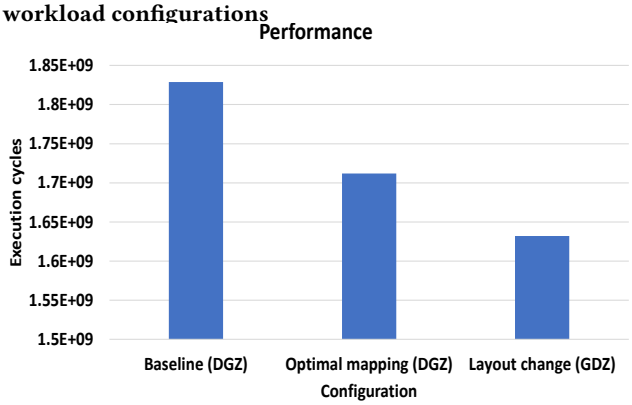

Figure 13: Comparison

does not vary significantly. Thus, we only need to run the application once, collect traces and compute near-optimal address mapping for the application. This mapping should be suitable for all future executions of the application.

6 RELATED WORK

Liu et al. [21] were among the earliest to show that the memory subsystem performance can be improved by manipulating the address map. Specifically, they proposed to hash bank bits of a physical address with other address bits to create modified bank bits which increased bank-level parallelism (and row-buffer hit rate). Our work in comparison explores optimizing row, column, bank group, and channel bits in addition to bank bits for better memory performance. Bojnordi et al. [19] demonstrated the importance of having a application specific mapping and their work shows performance gains of up to 12%. While they projected the performance benefits, reliable techniques to achieve the optimal mapping was not described. Other works [21] propose permutation-based page interleaving technique that decreases row-buffer conflicts by remapping the conflicting request to a different bank. Our proposed work focuses on broader level of memory hierarchy on distribution of workload equally to among multiple channels, banks groups and banks. DREAM [9] propose a method to extract memory access pattern and estimate optimized address pattern. This study is limited to one channel and our investigation revealed that the similar estimation technique for multi-channels may not work for all applications. In our work, we proposed a technique that performs better for some applications in multi-channel configuration and also investigated the reliability of entropy. Berkin [5] introduced a mathematical framework to optimize data reorganization process for common operations such as swap, transpose. Impulse [7] is an improved memory controller that improves the effective use of memory bandwidth by prefetching only the useful data in case of scientific workloads like sparse matrix-vector product. Our work focuses on optimizing address mapping for different types of applications. Self-Optimizing Memory Controller [11] is a Reinforcement learning (RL) based memory controller, that dynamically adjusts DRAM command scheduling policy based on the requirement, that improves DRAM bandwidth usage efficiently. While it is important to improve memory controller scheduling techniques, it is also crucial to optimize the address mapping for different category of applications.

7 CONCLUSION

In this paper, we investigated an address mapping technique that achieves near-optimal performance gains. We also evaluated the stability of entropy (and thus consistency of address mapping) across different runs of an application. Our technique is evaluated with different workloads and noticed on average 8 percent performance gains, and in some cases as high as 25 percent gains, when compared to the baseline. We have also shown that our approach to computing entropy is consistent across different runs with possibly different physical addresses. We noticed very small performance variations, between 1%-4%, across runs with different physical addresses, suggesting that entropy varies very little for each application and thus can be used more reliably for mapping addresses near optimally.

ACKNOWLEDGMENTS

The authors sincerely thank Vignesh Adhinarayanan (AMD), Derrick Aguren (AMD), Jagadish Kotra (AMD), Karthik Rao (AMD) for their feedback on this work. The research is supported in part by NSF award #1828105, and the NSF Net-Centric Industry/University Cooperative Research Center.

REFERENCES

[1] [n.d.]. Hynix GDDR5 SGRAM Part H5GQ1H24AFR. http://www.hytic.net/upload/files/2014/05/HYNIX-H5GQ1H24AFR.pdf.

[2] [n.d.]. Linux pagemap. https://www.kernel.org/doc/Documentation/vm/pagemap.txt.

[3] 2019. High Performance Conjugate Gradient. https://github.com/hpcg-benchmark/hpcg.

[4] Shashank Adavally and Krishna Kavi. 2018. 3D-DRAM Performance for Different OpenMP Scheduling Techniques in Multicore Systems. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 675–683.

[5] Berkin Akin, Franz Franchetti, and James C. Hoe. 2015. Data reorganization in memory using 3D-stacked DRAM. *42nd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)* (2015), 131–143.

[6] Scott Beamer, Krste Asanovic, and David A. Patterson. 2015. The GAP Benchmark Suite. *CoRR* abs/1508.03619 (2015). arXiv:1508.03619 http://arxiv.org/abs/1508.03619

[7] J. Carter, W. Hsieh, L. Stoller, M. Swanson, Lixin Zhang, E. Brunvand, A. Davis, Chen-Chi Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. 1999. Impulse: building a smarter memory controller. In *Proceedings Fifth International Symposium on High-Performance Computer Architecture*. 70–79. https://doi.org/10.1109/HPCA.1999.744334

[8] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[9] Mohsen Ghasempour, Jim D. Garside, Aamer Jaleel, and Mikel Luján. 2015. DReAM: Dynamic Re-arrangement of Address Mapping to Improve the Performance of DRAMs. *ArXiv* abs/1509.03721 (2015).

[10] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Understanding the Interactions of Workloads and DRAM Types: A Comprehensive Experimental Study. *ArXiv* abs/1902.07609 (2019).

[11] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. 2008. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *2008 International Symposium on Computer Architecture*. 39–50. https://doi.org/10.1109/ISCA.2008.21

[12] Jewillco. 2015. Graph500-v2-spec. https://github.com/graph500/graph500/tree/v2-spec.

[13] Dongkyun Kim, Minsu Park, Sungchun Jang, Jun-Yong Song, Hankyu Chi, Geunho Choi, Sunmyung Choi, Jaeil Kim, Changhyun Kim, Kyungwhan Kim, et al. 2019. 23.2 A 1.1 V 1ynm 6.4 Gb/s/pin 16Gb DDR5 SDRAM with a Phase-Rotator-Based DLL, High-Speed SerDes and RX/TX Equalization Scheme. In *2019 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 380–382.

[14] Y. Kim, W. Yang, and O. Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (Jan 2016), 45–49. https://doi.org/10.1109/LCA.2015.2414456

[15] Adam Kunen, Teresa S. Bailey, and Peter N. Brown. 2015. KRIPKE - A MASSIVELY PARALLEL TRANSPORT MINI-APP.

[16] Y. Liu, X. Zhao, M. Jahre, Z. Wang, X. Wang, Y. Luo, and L. Eeckhout. 2018. Get Out of the Valley: Power-Efficient Address Mapping for GPUs. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 166–179. https://doi.org/10.1109/ISCA.2018.00024

[17] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Languagen Design and Implementation* (Chicago, IL, USA) *(PLDI '05)*. ACM, New York, NY, USA, 190–200. https://doi.org/10.1145/1065010.1065034

[18] Onur Mutlu and Thomas Moscibroda. 2007. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors.. In *MICRO*. IEEE Computer Society, 146–160.

[19] Mahdi Nazm Bojnordi and Engin Ipek. 2012. PARDIS: A programmable memory controller for the DDRx interfacing standards. *ACM Transactions on Computer Systems (TOCS)* 31, 13–24. https://doi.org/10.1109/ISCA.2012.6237002

[20] Chi-Sung Oh, Ki Chul Chun, Young-Yong Byun, Yong-Ki Kim, So-Young Kim, Yesin Ryu, Jaewon Park, Sinho Kim, Sanguhn Cha, Donghak Shin, et al. 2020. 22.1 A 1.1 V 16GB 640GB/s HBM2E DRAM with a Data-Bus Window-Extension Technique and a Synergetic On-Die ECC Scheme. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 330–332.

[21] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. 2000. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*. 32–41. https://doi.org/10.1109/MICRO.2000.898056