# Get Out of the Valley: Power-Efficient Address Mapping for GPUs

Yuxi Liu[§‡], Xia Zhao[‡], Magnus Jahre[†], Zhenlin Wang[*], Xiaolin Wang[§], Yingwei Luo[§], Lieven Eeckhout[‡]
[§]Peking University [†]Norwegian University of Science and Technology
[*]Michigan Technological University [‡]Ghent University

***Abstract*—GPU memory systems adopt a multi-dimensional hardware structure to provide the bandwidth necessary to support 100s to 1000s of concurrent threads. On the software side, GPU-compute workloads also use multi-dimensional structures to organize the threads. We observe that these structures can combine unfavorably and create significant resource imbalance in the memory subsystem — causing low performance and poor power-efficiency. The key issue is that it is highly application-dependent which memory address bits exhibit high variability.**

**To solve this problem, we first provide an entropy analysis approach tailored for the highly concurrent memory request behavior in GPU-compute workloads. Our window-based entropy metric captures the information content of each address bit of the memory requests that are likely to co-exist in the memory system at runtime. Using this metric, we find that GPU-compute workloads exhibit *entropy valleys* distributed throughout the lower order address bits. This indicates that efficient GPU-address mapping schemes need to harvest entropy from broad address-bit ranges and concentrate the entropy into the bits used for channel and bank selection in the memory subsystem. This insight leads us to propose the *Page Address Entropy (PAE)* mapping scheme which concentrates the entropy of the row, channel and bank bits of the input address into the bank and channel bits of the output address. PAE maps straightforwardly to hardware and can be implemented with a tree of XOR-gates. PAE improves performance by 1.31× and power-efficiency by 1.25× compared to state-of-the-art permutation-based address mapping.**

## I. Introduction

GPUs need high-bandwidth memory systems to support their massively parallel execution model. Current DRAM solutions such as GDDR5 [1] and 3D-stacked memory [2], [3] deliver high theoretical performance. Unfortunately, it is difficult to reach this potential with contemporary GPU-compute workloads, leading to suboptimal bandwidth utilization, performance and power-efficiency [4]. To maximize bandwidth, DRAM interfaces are organized in a four-dimensional structure of channels, banks, rows and columns. The way the application memory access streams are mapped onto this structure has a significant impact on performance and power consumption. For the row bits, the addresses should change as little as possible to ensure high row buffer locality. For the channel and bank bits, the addresses should be highly variable to ensure uniform distribution of memory requests across channels and banks [5].

To find a good address mapping, it is critical to understand how entropy [6] is distributed across address bits. Informally, a low-entropy value indicates that it is unlikely that a bit will change. Conversely, a high-entropy value indicates that an address bit is likely to change. Thus, low-entropy address bits should be mapped to rows — to exploit row buffer locality — and high-entropy bits should be mapped to channels and banks — to exploit parallelism. Address mapping schemes have previously been proposed for single-core CPUs [5], multi-core CPUs [7] and GPUs [4], [8], [9], [10]. Our objective is to systematically analyze the entropy of the concurrent memory addresses in GPU-compute workloads and use this insight to derive efficient address mapping policies.

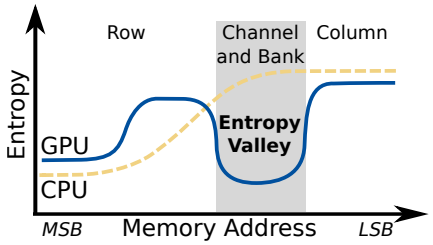



Figure 1: GPU and CPU address bit entropy distribution. *GPU workloads that exhibit an entropy valley in the channel or bank bits limit parallelism in the memory subsystem.*

**GPU address bit entropy.** Our first objective is to establish the address bit entropy distribution of GPU-compute workloads. This is challenging due to the highly threaded execution model of GPUs. Specifically, the application consists of *Thread Blocks (TBs)* that the hardware schedules on *Streaming Multiprocessors (SMs)*; an SM can execute several TBs concurrently. In-flight memory requests originate from different threads within a TB as well as from different TBs within an SM and across SMs. The number of concurrently executing TBs is bounded by the available hardware resources. Since TBs are highly parallel, it is likely that the memory requests of the TBs within this window co-exist in the memory system. We devise a window-based entropy metric that captures the information content of each address bit of these likely concurrent memory requests.

Our entropy analysis shows that GPU address bit entropy is significantly different from CPU address bit entropy, as illustrated in Figure 1. For CPUs, entropy results from spatial locality, e.g., accesses to arrays within loops where the loop index is used to access the array. Thus, the least significant address bits tend to change often (e.g., high entropy), and

entropy gradually decreases towards the most significant bits [11]. GPU address bit entropy is radically different, and lower order bits may have low entropy. In fact, many of our benchmarks contain an *entropy valley* in the lower order bits. Unfortunately, the entropy valley commonly overlaps with bits that have high entropy for CPUs. Thus, CPU-oriented address mapping schemes [4], [8], [9], [10] fail to deliver high memory request parallelism for GPU-compute workloads.

Which bits end up having high entropy is highly application-dependent because it depends on how the application accesses memory. The mapping problem is made even more challenging by observing that the high-entropy bits move across the address space as new TBs get scheduled and the application iterates through memory. Therefore, it is not sufficient to simply select bits that have high average entropy and use them where high entropy is needed. On the contrary, we need to harvest entropy across a broad selection of address bits to be robust to entropy-variation across applications and across phases within a single application. Prior work fails to achieve this requirement since they gather entropy across a narrow address range by XORing the bank and channel bits with the least significant row bits [4], [8], [9], [10].

**GPU address mapping.** Our second objective is to leverage the key insights of our entropy analysis to develop novel GPU address mapping schemes that consistently provide high-entropy bits for the channel and bank selection. Prior work [12], [13] has shown that memory remapping operations can be concisely represented with a *Binary Invertible Matrix (BIM)*. We observe that the BIM is able to represent all possible address mapping schemes that consist of AND and XOR operations. The reason is that the matrix covers all possible transformations, and the invertibility criterion ensures that all possible one-to-one relations are considered. Thus, the BIM abstraction forms a theoretical framework for reasoning about different classes of address mappings. Multiplying the original address with the BIM generates a new address with different entropy characteristics.

Armed with the BIM abstraction, we investigate strategies for gathering bit entropy from the whole address and concentrating it into the channel and bank bits. We investigate two strategies which we refer to as the *Full Address Entropy (FAE)* and *Page Address Entropy (PAE)* mapping schemes. With FAE, we randomly select bits from the complete input address to be part of the BIM-driven address mapping. The PAE scheme is limited to only selecting random bits from the input address fields that make up the DRAM page address (i.e., the row, channel and bank bits). For completeness, we also explore distributing the entropy across the complete memory address (i.e., the *ALL* mapping scheme).

Our experiments show that both the FAE and PAE address mapping schemes provide significantly higher performance and power-efficiency than previously proposed address mapping schemes. More specifically, PAE and FAE improve performance by 1.52× and 1.56× on average, respectively, over the baseline Hynix address map [14]. Moreover, we find that PAE is the most power-efficient mapping scheme: performance per Watt improves by 1.39× on average over the baseline, versus 1.36× for FAE. The ALL scheme performs similarly to FAE because increasing the entropy of the column and row bits do not further improve resource balance.

**Paper contributions.** In summary, this work makes the following major contributions:

- We extend memory address entropy analysis to cover GPU-compute workloads. In particular, we observe that a useful entropy metric must compute the information content of each address bit of the memory requests that are likely to co-exist in the memory system. Our novel entropy analysis methodology shows that there is significant opportunity for improving upon the current state-of-the-art GPU address mapping schemes.
- We observe that all DRAM address mapping schemes that use AND and XOR operations can be represented by a *Binary Invertible Matrix (BIM)*. With the BIM-abstraction, the new address can be computed with a binary matrix vector product and the invertibility criterion ensures that there is a one-to-one mapping from the input to the output address. The BIM is a unified representation that provides a theoretical basis for analyzing families of mapping schemes. Furthermore, BIMs can be straightforwardly implemented in hardware with a tree of XOR-gates.
- We develop the *Page Address Entropy (PAE)* mapping scheme which harvests entropy from the row, channel and bank fields of the input address and uses this to create well-distributed channel and bank fields in the output address. Among the address mapping schemes proposed in this paper, PAE is the most power-efficient one, improving performance per Watt by 1.25× over state-of-the-art address mapping.
- We also develop the *Full Address Entropy (FAE)* mapping scheme. FAE provides even more well-distributed channel and bank fields than PAE because it harvests entropy from the complete input address. Therefore, FAE is the best performing address mapping scheme among the ones proposed in this paper, improving performance by 1.34× compared to state-of-the-art address mapping. However, FAE has lower power efficiency than PAE because it reduces row buffer locality by distributing page hits to different banks which increases the number of DRAM page activations.

## II. Understanding GPU-Compute Workload Address Behavior

**GPU thread organization.** The basic unit of a GPU-compute workload is a thread, and each thread is responsible for carrying out a desired computation on a subset of the

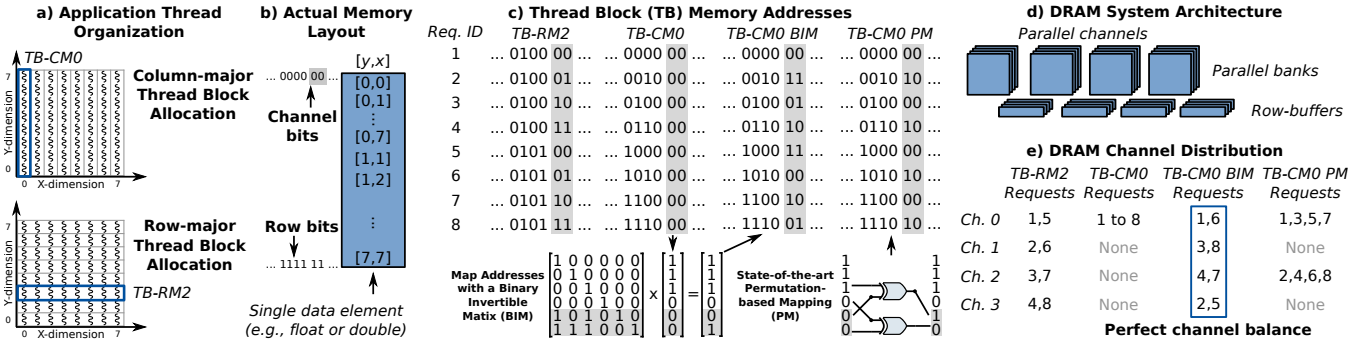



Figure 2: Removing the entropy valley with BIM-driven address mapping.

application's input data. Threads are organized into one or two-dimensional *Thread Blocks (TBs)*, and all threads in a TB run on the same *Streaming Multiprocessor (SM)*.

The threads in a TB can execute sequentially or in parallel and communicate with each other. The TBs are again organized into a grid, and this grid can have up to three dimensions. The purpose of this section is to provide an example of how the TB and grid structure can align unfavorably with the structure of the DRAM system and cause severe resource imbalance. We will also demonstrate that the problem can be easily fixed by applying our BIM-driven address mapping strategies.

Li et al. [15] find that threads can be organized using row-major, column-major, tiled or arbitrary patterns. Figure 2a shows a two-dimensional map of threads and the row-major and column-major strategies for creating one-dimensional TBs. For the purpose of this example, we assume that each thread accesses a single data element in a C-style two-dimensional array. Other thread mappings are also possible, but we choose these two as they clearly illustrate how application-level thread organization can create resource imbalance in the DRAM system. Application developers will choose a thread to TB and grid mapping that is a good fit for the computation and communication patterns of the algorithm. Thus, it is desirable that the GPU provides high performance regardless of which application-level mapping strategy is employed.

Each thread uses its position to compute its ID and uses this ID to determine which data elements to work on. In other words, the threads use *dimension-related indexing* of the input and output arrays. To keep the example simple, we focus on the thread-to-TB mapping. In the row-major case, the thread can compute its ID by computing the offset of its row and adding its x-coordinate (i.e., `i = threadIdx.y * blockDim.x + threadIdx.x`). For the column-major case, the thread first computes the offset of its column and then adds its y-coordinate (i.e., `i = threadIdx.x * blockDim.y + threadIdx.y`). In this example, we will focus on the first TB of the column-major strategy (i.e., *TB-CM0*) and the third TB of the row-major strategy (i.e., *TB-RM2*). TB-RM2 and TB-CM0 are highlighted in Figure 2a, and TB-RM2 is responsible for indices 16 to 23 while TB-CM0 is responsible for indices 0, 8, 16, 24, 32, 40, 48 and 56. For real applications, thread ID computation occurs deep within the multi-dimensional thread structure which makes it challenging to fully alleviate memory access alignment issues with application-level thread reorganization.

**Memory and DRAM organization.** The threads read from a contiguous C-style array in DRAM. Figure 2b shows how the grid of threads maps to the flat memory address space of the GPU. We assume that data is allocated in row-major order. Thus, the data element at index [0,0] is the first data element in memory, [0,1] the second (i.e., row 0 and column 1) and [7,7] the last data element. For convenience, we assume that the six address bits that correspond to the index of the array are 0 for element [0,0]. These address bits are incremented by one for each new data element. Further, we assume that the DRAM system implementation uses the two least significant of these bits to select the DRAM channel. Figure 2c shows the resulting memory addresses for TB-RM2 and TB-CM0. The key observation is that TB-RM2's addresses change in the three least significant bits and TB-CM0's addresses change in the three most significant bits.

Figure 2d shows the organization of the DRAM system. DRAM systems also have a complex, multi-dimensional structure to maximize bandwidth. Concretely, they consist of independent channels and banks, and a subset of the bits in the memory address is used to distribute requests among these units. Since the channels and banks can operate mostly in parallel, it is critical to evenly distribute requests across these units to achieve high throughput. To retrieve a data item from DRAM, the memory controller first provides the bank and row address. This causes a row (or page) of the DRAM array to be transferred to the row-buffer. The row-buffer is then accessed using the column address. Repeated accesses to the row-buffer are very efficient and have low latency. Since DRAMs are destructive read, the page must be written back to the DRAM array before a different page can be loaded into the row buffer.

Figure 2e shows how the memory requests of TB-RM2

and TB-CM0 are distributed across the DRAM channels. TB-RM2 achieves perfect channel balance since the address bits used to select the channel vary considerably. TB-CM0 exhibits remarkably different behavior, and all of its memory requests are mapped to DRAM channel 0 because both address channel bits are 0 for all threads. The root cause of this problem is that the application level thread organization maps unfavorably to the organization at the DRAM level. Such interactions are undesirable because they are difficult to identify and understand at the application level. Furthermore, fixing dimension-order performance problems may be difficult or even impossible for complex GPU compute workloads.

**Our BIM-based address mapping approach.** In this work, we show that the channel and bank imbalance problem can be remedied in hardware by employing a simple address mapping scheme. *The key idea is to harvest entropy from broad ranges of the address and concentrate this into the channel and bank bits.* The highly-parallel and multi-dimensional data-organization of GPU-compute workloads means that high-entropy bits are available in the address.

Figure 2c shows the memory addresses after applying our BIM-driven mapping strategy to the memory addresses of TB-CM0. At design time, we generate a binary matrix and ensure that it is invertible. The invertibility criterion guarantees that there is a one-to-one mapping between the new and old address. At run time, all memory addresses are multiplied by this matrix directly after memory coalescing. In Figure 2c, we show that multiplying the address 111000 by the BIM gives the new address 111001 which causes the request to move from channel 0 to channel 1. The matrix vector multiplication is very efficient because binary multiplication is an AND-operation and addition an XOR-operation. Figure 2e shows that applying our BIM-based address mapping also gives TB-CM0 a perfectly balanced DRAM-channel distribution.

**Comparison to state-of-the-art Permutation-based mapping (PM).** Figure 2c shows the memory addresses after applying the state-of-the-art PM [4], [5] scheme to the memory addresses of TB-CM0. PM increases channel address bit entropy by taking the exclusive-or of a channel and a non-channel address bit. It uses the output of this operation as a new channel bit, and repeats the procedure for all other channel bits. Figure 2e shows that PM is able to improve bank balance but still causes requests to cluster in channel 0 and 2. The reason is that the two bits PM used to increase entropy fails to capture the entropy available in TB-CM0's addresses. In the next section, we will show that this problem affects PM even more for real applications since the location of high entropy bits depends strongly on the application.

## III. Quantifying the Address Entropy of GPU-Compute Workloads

In this section, we extend memory address entropy analysis to investigate the address entropy of GPU-compute workloads. We start with Shannon's entropy function [6]:

$$H(p_1, p_2, \ldots, p_v) = -\sum_{i=1}^{v} p_i \log_v p_i. \quad (1)$$

For a sequence of bits, there are two probabilities: $p$ is the probability that a bit is 1 and $(1-p)$ is the probability that the bit is 0. Overall, entropy measures the amount of information contained in a sequence of values. A value that changes frequently contains a lot of information and has high entropy. Conversely, a value that changes rarely contains little information and has low entropy. Entropy is a number between 0 and 1 where 0 means that the bit has a constant value and 1 means that the bit has maximum variability (i.e., the probability of 0s and 1s are equal).

### A. Window-based Entropy

GPU-compute workloads are highly concurrent. Therefore, the entropy metric cannot rely on memory request ordering assumptions because requests from different threads can interleave in many ways. To avoid this problem, we identify the memory requests that are likely to co-exist in the memory system and compute the entropy of each address bit for these requests. Concretely, address bit entropy is determined by the addresses of the requests issued by the threads within each TB (e.g., intra-TB entropy) and by requests issued by concurrently executed TBs (e.g., inter-TB entropy). In this work, we propose a new entropy metric called *window-based entropy* which accounts for both sources of entropy.

To compute the window-based entropy, we start with estimating intra-TB entropy. For each TB, we gather the addresses of all memory requests issued by the TB. Then, we compute the *Bit Value Ratio (BVR)* of all address bits. The BVR is the fraction of 1-values for a single address bit across all memory requests within a TB. The key feature of the BVR is that it quantifies the likelihood of address bit variability without making request ordering assumptions.

To account for inter-TB entropy, we first sort the BVR-values in ascending order using the TB identifiers. The reason is that TBs are issued sequentially by the TB-scheduler. However, the TB-scheduler may also issue multiple TBs concurrently if sufficient hardware resources are available. We approximate this behavior with the *window size* parameter $w$ which indicates the number of TBs that can execute concurrently in all SMs.

The main benefit of modeling inter-TB entropy is that it can compensate for low intra-TB entropy. We illustrate this with a simple example. Consider a single address bit and two TBs $A$ and $B$ with BVRs of 0 and 1, respectively. In this case, intra-TB entropy is low for both $A$ and $B$ since a BVR of 0 (1) means that the address bit value is 0 (1) for all requests issued by the TB. However, co-executing $A$ and $B$ increases entropy since the concurrent memory addresses will contain a mix of zeros and ones for this address bit.

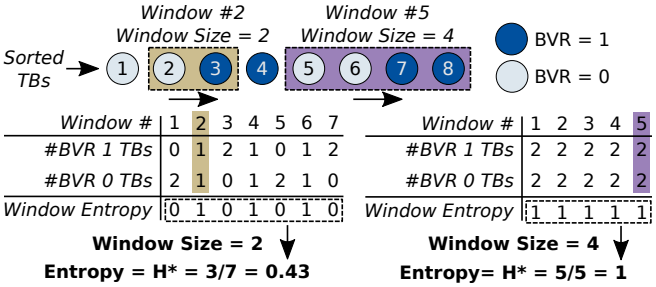



Figure 3: Window-based entropy example.

Equation 2 shows the details of how we compute the window-based entropy $H^*$ for a single address bit:

$$H^* = \frac{\sum_{i=1}^{n-w+1} H_i^W(p_1^{\text{BVR}}, p_2^{\text{BVR}}, \ldots, p_v^{\text{BVR}})}{n-w+1}. \quad (2)$$

We first find the $v$ unique BVRs that occur for this address bit across all TBs within a window $i$. Within the window, we then count the number of times each BVR-value occurs and compute the probability of each BVR-value $p^{\text{BVR}}$ by dividing the number of occurrences by the window size $w$. Then, we use Equation 1 to compute the entropy $H_i^W$ with these probabilities.[1]

The window-based entropy $H^*$ is the arithmetic mean of all window entropies. The sum ends at $n-w+1$ because this is the number of windows when there are $n$ TBs and the window size is $w$. We calculate $H^*$ for each kernel of the application since the TBs of different kernels do not execute concurrently in our setup. Then, we take the weighted average of the per-kernel entropy distributions to create an entropy distribution for the complete application. The weight of each kernel is the number of memory requests it contains.

The window size $w$ is affected by both the architecture and the application because it depends on the hardware resources provided by the SM and the hardware needs of the TBs. In this work, we leverage the observation that state-of-the-art warp scheduling policies, such as Greedy-Then-Oldest (GTO) [16], strive to execute instructions from the same warp until it stalls, after which it picks the oldest warp, determined by the time the warp was assigned to the SM. Because all warps from the same TB have the same age and a TB gets assigned to an SM as a unit, this suggests that we can heuristically set the window size equal to the number of SMs, which is what we do in our experiments. Other warp-scheduling policies may benefit from different window sizes. The key insight is that the combination of warp-scheduler and hardware resources determines which TBs are likely to issue concurrent requests.

**Example.** Figure 3 illustrates how we compute the window-based entropy. In this example, we assume an application with 8 TBs where half of the TBs have a BVR-value of 1 and the other half have a BVR-value of 0. The TBs are sorted according to their identifiers, and we use sliding window sizes of 2 and 4. We slide the window across the TBs from left to right and identify the windows by the identifier of the first TB covered. In other words, window #1 starts with TB #1, window #2 starts with TB #2, and so on.

We first consider the case with a window size of 2, and focus on window #2. For each window, we count the occurrences of each BVR-value and use this to compute the probabilities by dividing the number of occurrences by the window size. Then, we use the probabilities to compute the window-entropy. For instance, window #2 contains one TB with each BVR-value which means that both probabilities are $1/2$. This gives a window-entropy of 1 (following Equation 1). Window #1 contains two TBs with BVR-value 0. This gives a probability of 1 for BVR-value 0, 0 for BVR-value 1 and a window-entropy of 0. The overall entropy is the average of the window entropies (i.e., $H^* = 3/7$).

Using a window size of 4 changes the window-based entropy significantly. Here, all windows contain two TBs with BVR-value 0 and two TBs with BVR-value 1. Thus, the probability is $1/2$ in both cases. This results in all window-entropies being 1 and an overall entropy $H^*$ of 1. This illustrates that modeling the effect of the window size is beneficial since it can have a substantial impact on the amount of entropy available.

[1] Example: Consider a window with three TBs where two TBs have a BVR of 0 and a single TB has a BVR of 1. In this case, there are two unique BVR values which gives $p_1^{\text{BVR}} = 2/3$, $p_2^{\text{BVR}} = 1/3$, and $H^W = 0.92$.

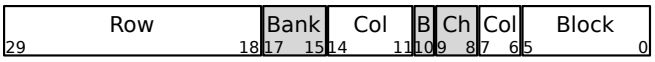


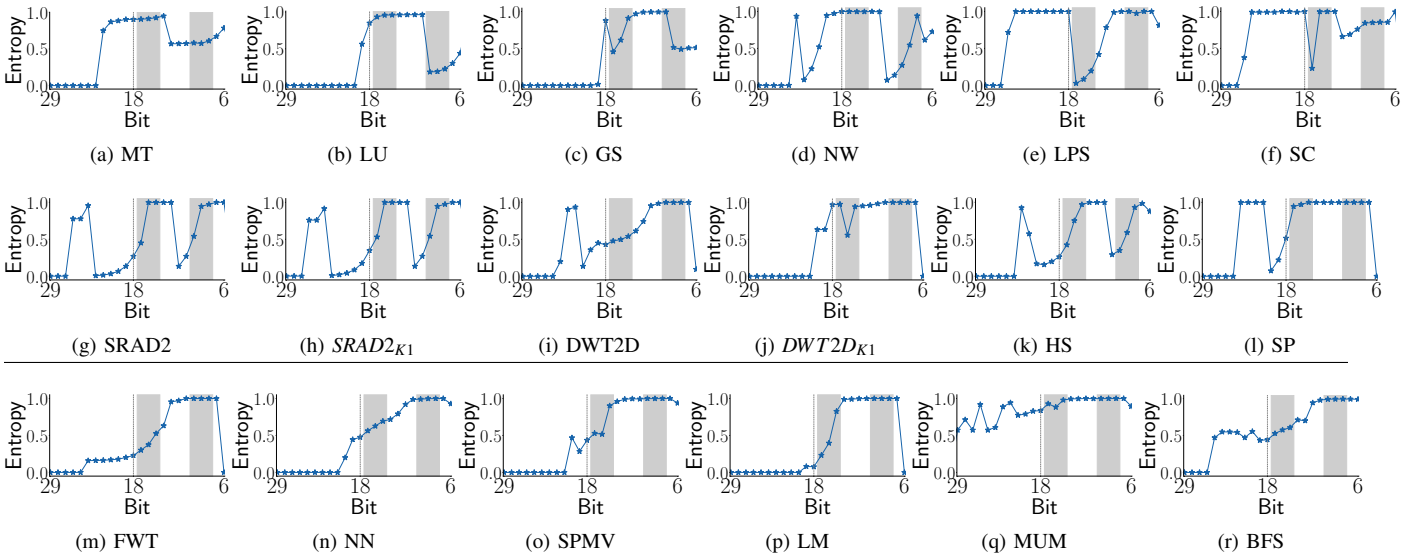

Figure 4: 30-bit address map for our baseline 1 GB Hynix GDDR5 memory [14].

### B. GPU-Compute Workload Entropy Distributions

To investigate the entropy distribution of GPU-compute workloads, we apply our entropy metric to 16 GPU-compute benchmarks and 2 kernels[2]. We model a 1 GB Hynix GDDR5 memory [14] with a 30-bit physical address space and the physical address map from Figure 4. The resulting entropy distributions are shown in Figure 5. The most significant address bit is at index 29 and on the left side of the plot. We do not show the DRAM block address bits (bits 5 through 0) since these are offsets within a DRAM page and have no impact on the behavior of the DRAM system. Thus, bit 6 is the least significant bit and shown to the right in the distributions. The bits used for bank and channel selection are shown with gray background.

The main take-away from Figure 5 is that all benchmarks have high entropy bits, but they are distributed throughout the address map. Many benchmarks have deep entropy valleys (e.g., LU, NW and LPS), and the location of these valleys

[2] We describe our experimental setup in detail in Section V.




(a) MT (b) LU (c) GS (d) NW (e) LPS (f) SC

(g) SRAD2 (h) $SRAD2_{K1}$ (i) DWT2D (j) $DWT2D_{K1}$ (k) HS (l) SP

(m) FWT (n) NN (o) SPMV (p) LM (q) MUM (r) BFS

Figure 5: Overall entropy distribution for 16 GPU-compute benchmarks and 2 kernels. *The benchmarks above the horizontal line have entropy valleys while the benchmarks below the line have the entropy concentrated in the lower order address bits. Bits that are used for bank and channel selection have gray background.*

is highly application-dependent. This shows that an effective address mapping scheme must be able to harvest entropy from broad address bit ranges to be robust to inter-application entropy variation.

Entropy can also vary between the kernels of a single application. Figure 5i shows the overall entropy distribution for DWT2D, and Figure 5j shows the entropy distribution for one of its kernels. The entropy distribution of the application shows a broad entropy valley while the profile of the kernel has a narrow valley. This illustrates that entropy can vary considerably between the kernels of a single benchmark. For other benchmarks such as SRAD2, the entropy distributions of its kernels are similar to the overall profile (see Figure 5g and 5h).

## IV. GPU Address Mapping Schemes

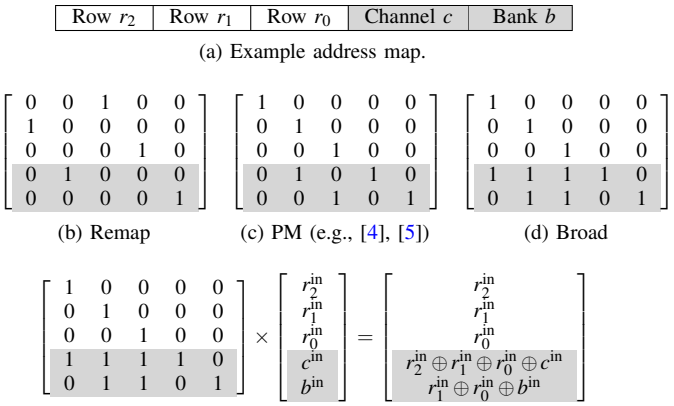
We have now established that many GPU-compute workloads have entropy valleys in the bits state-of-the-art DRAM mapping schemes use to select channels and banks. Furthermore, we have demonstrated that different applications have entropy valleys in different locations and that a single application can have entropy valleys at different locations for different kernels and phases. In this section, we explain how our BIM-based mapping schemes alleviate bank and channel congestion by gathering entropy from broad ranges of address bits and concentrating this entropy into the bits used for channel and bank selection. Our BIM-based address mapper is located after the memory coalescer unit and is therefore orthogonal to memory coalescing.

### A. Representing Address Mapping Schemes with BIMs

The *Binary Invertible Matrix (BIM)* [12], [13] is a generic abstraction for representing memory address mapping schemes. The key observation is that address transformation can be represented as a matrix-vector product $\mathrm{BIM} \times \vec{a}^{in} = \vec{a}^{out}$ where an input address $\vec{a}^{in}$ is mapped to a new address $\vec{a}^{out}$ by multiplying $\vec{a}^{in}$ with the BIM. In binary arithmetic, multiplication maps to the bit-wise AND-operation and addition maps to the XOR-operation. Since the BIM is required to be invertible, we are guaranteed that each input address is mapped to exactly one output address. The BIM abstraction provides a powerful theoretic framework for reasoning about address mapping strategies. In fact, the BIM is able to represent all possible one-to-one address mapping strategies for the AND- and XOR-operators.

Figures 6b, 6c and 6d show the BIM-representation of three classes of address mapping strategies for the simple address map of Figure 6a. Figure 6b shows the *Remap (RMP) strategy* which rearranges the address bits that tend to have high average entropy to be used as channel and bank bits. Thus, a remapping BIM has a single '1' for all rows and columns. The address map of the Hynix' GDDR5 [14] (Figure 4) and the minimalist open-page mapping scheme [7] are examples of applying the Remap-strategy.

The remap strategy is unable to account for different applications having high entropy bits in different locations and that high entropy bits change over time. The *Permutation-based Mapping (PM)* strategy [4], [5] improves upon the Remap strategy by using single row bits to increase the entropy of the channel and bank bits. Figure 6c shows an




(a) Example address map.

(b) Remap (c) PM (e.g., [4], [5]) (d) Broad

(e) Address map computation for the *Broad* BIM example.

Figure 6: Example showing BIM-representations of the *Remap*, *PM* and *Broad address mapping strategies* for a simplified address map.

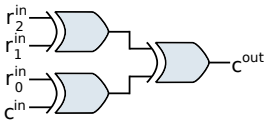



Figure 7: Hardware for address translation of the $c^{out}$ bit.

example of a BIM representing a PM-strategy. The key characteristic of BIMs for PM-strategies is that there are two ones in the rows of the BIM that is responsible for generating the new channel and bank bits. Thus, PM strategies increase the entropy of the bank and channel bits by combining the entropy of narrow ranges of address bits.

Although PM-strategies can improve upon Remap-strategies, they are in general unable to account for entropy valleys and entropy changes due to application phase behavior. The key observation is that a remapping scheme needs to gather entropy from a *broad range of address bits* to be robust to these changes. We refer to this as the *Broad* strategy, and Figure 6d contains a BIM that implements this strategy. The key difference from the PM-strategy is that the rows of the BIM responsible for generating the bank and channel bits now contain multiple ones. Figure 6e shows how this pattern affects the new address computation. Here, the output address channel bit $c^{out}$ is the XOR of the row bits $r_2^{in}$, $r_1^{in}$ and $r_0^{in}$ and the original channel bit $c^{in}$. In other words, the new channel bit leverages the combined entropy of the original channel and row bits. By distributing the input bits throughout the input address, the Broad-strategy is robust to both inter- and intra-application address bit entropy variation.

Another advantage of the BIM-abstraction is that it can be efficiently realized in hardware. Since the BIM is constant for a given GPU architecture, it can be realized as a fixed-function hardware unit. Figure 7 contains a simple circuit that implements the generation of the new channel bit $c^{out}$ for the example in Figure 6e. Since the AND of a bit $b$ and 1 is always $b$, we can simply select the input lines where there are ones in the BIM. The wide XOR-operations can be be realized with a tree of two-input XOR-gates. The result is that the address mapping function can be carried out in a single cycle on contemporary GPUs.

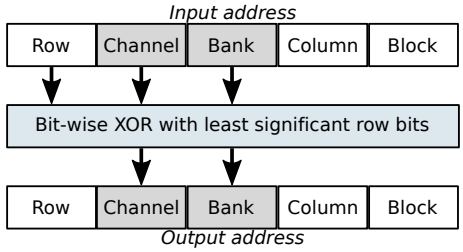



Figure 8: Permutation-based (PM) mapping scheme [4], [5].

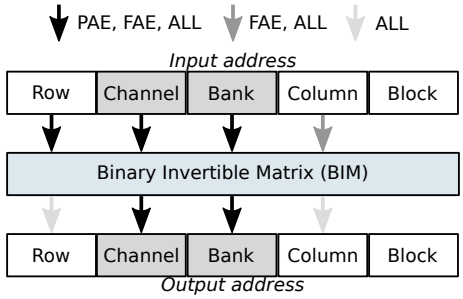
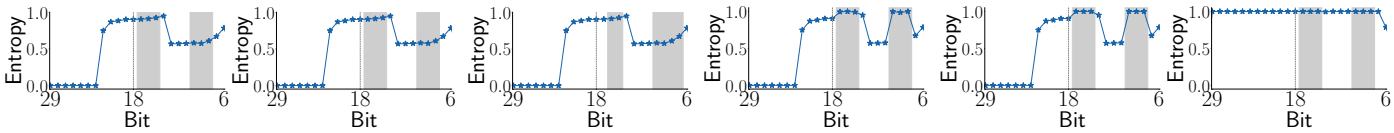



Figure 9: Our Broad-strategy mapping schemes.

### B. Generating BIM Representations

The Remap-, PM- and Broad-strategy can each be realized with different BIM representations since many invertible matrices exist within the bits-per-row restrictions outlined in the previous section. Since the number of matrices is large, we need more refined design strategies to make sense of the design space. For the Remap-strategy, we first gather the entropy of all our GPU-compute benchmarks and aggregate this into a global entropy profile. Then, we allocate the 6 bits with the highest average entropy to bank and channel selection (i.e., bits 8-11, 15, and 16). In the remainder of the paper, we refer to this as the *Remap (RMP)* mapping scheme. Figure 8 shows our implementation of the PM-strategy. Like prior work [4], [5], we take the XOR of the least significant row bits and the bank and channel bits (as exemplified in Figure 6c). We refer to this BIM-instantiation as *Permutation-based Mapping (PM)*.

To cover the design space of Broad-strategy mapping schemes, we define three sub-strategies which gather entropy from different locations in the input address and distribute it to different locations in the output address (see Figure 9). The first strategy, called *Page Address Entropy (PAE)*, gathers entropy from randomly selected address bits within the page address (i.e., channel, bank and row) and uses the entropy of these bits to generate new bank and channel bits. The




(a) BASE (b) PM (c) RMP (d) PAE (e) FAE (f) ALL

Figure 10: MT's entropy distribution for the six address mapping schemes. *PAE and FAE effectively remove the entropy valley at bits 8–10; ALL removes all valleys.*

| SM Configuration | |
|---|---|
| No. SMs | 12 SMs |
| SM resources | 1.4 GHz, 32 SIMT width, 48 KB shared memory<br>Max. 1536 threads (48 warps/SM, 32 threads/warp) |
| Scheduler | 2 warp schedulers per SM, GTO policy |
| L1 data cache | 16 KB per SM (4-way, 32 sets),<br>128 B block, 32 MSHR entries |
| LLC | 512 KB in total (8 slices, 8-way, 64 sets),<br>120 clock cycles latency |
| NoC | $12 \times 8$ crossbar, 700 MHz<br>32-byte channel width, 179.3 GB/s |
| **DRAM Memory Configuration** | |
| DRAM Timing | Hynix GDDR5 [14] |
| DRAM configuration | 924 MHz, 4 Memory Channels/Controllers (MC),<br>16 banks/MC, 4 K rows/bank, 64 columns/row,<br>FR-FCFS [17], open page mode, 118.3 GB/s,<br>12-12-12 (CL-tRCD-tRP) timing |
| **3D-stacked Memory Configuration** | |
| Memory stack configuration | 4 memory stacks, 16 vaults/stack,<br>16 banks/vault, 64 TSVs/vault<br>1.25 Gb/s TSV signaling rate, 640 GB/s |

Table I: Simulated GPU architecture.

| Benchmark | Abbr. | APKI | MPKI | #Knls | #Insns |
|---|---|---|---|---|---|
| Transpose [18] | MT | 7.44 | 5.69 | 4 | 0.19 B |
| LU Decomposition [18] | LU | 12.32 | 1.97 | 1022 | 2.22 B |
| Gaussian [19] | GS | 9.09 | 0.01 | 510 | 0.43 B |
| Needle [19] | NW | 5.25 | 5.12 | 255 | 0.21 B |
| Laplace [20] | LPS | 2.27 | 1.66 | 2 | 2.33 B |
| StreamCluster [19] | SC | 4.24 | 3.58 | 50 | 1.71 B |
| Srad_v2 [19] | SRAD2 | 3.29 | 1.85 | 4 | 2.43 B |
| DWT2D [19] | DWT2D | 1.56 | 1.21 | 10 | 0.33 B |
| Hotspot [19] | HS | 0.71 | 0.08 | 1 | 1.3 B |
| Scalar Product [18] | SP | 2.17 | 2.16 | 1 | 0.12 B |
| Fast Walsh Transform [18] | FWT | 2.69 | 1.38 | 22 | 4.38 B |
| NN [20] | NN | 2.33 | 0.2 | 4 | 0.31 B |
| SPMV [21] | SPMV | 5.95 | 2.75 | 50 | 0.19 B |
| LavaMD[19] | LM | 18.23 | 0.01 | 1 | 2.11 B |
| MUMmerGPU [19] | MUM | 25.63 | 22.53 | 2 | 0.23 B |
| BFS [19] | BFS | 26.92 | 18.14 | 24 | 0.46 B |

Table II: GPU-compute benchmarks considered in this study.

second strategy, called *Full Address Entropy (FAE)*, adds the column bits in addition to the input fields used in PAE, but still only changes the bank and channel bits. Finally, the *All* strategy uses the same input as FAE, but generates new row, column, channel and bank bits. Neither strategies use or modify the block address bits since these are offsets within a DRAM page and therefore have no impact on the behavior of the DRAM system. In the evaluation, we randomly generate three BIMs for each strategy, and unless otherwise specified we report the results of the best performing BIM for each strategy. We find performance to be (relatively) insensitive to the particular BIM instantiation, see Section VI-D.

**Example.** We now illustrate how the different address mapping schemes affect the entropy distribution for one specific benchmark, namely MT, see Figure 10 — we observe similar results for the other benchmarks. Hynix (BASE) exhibits a clear entropy valley for channel bits 8–9 and bank bit 10. PM and RMP are unable to remove this valley. PAE and FAE on the other hand effectively removes the valley. ALL removes all valleys.

## V. Experimental Setup

**Performance simulation.** We use the GPGPU-sim v3.2.2 simulator [22] for all our experiments. The simulated baseline GPU architecture features 12 SMs, which can support up to 48 warps of 32 threads each. We assume the GTO warp scheduling policy within an SM [16]. Each SM features 48 KB shared memory along with a 16 KB L1 data cache. The last-level cache (LLC) is partitioned into 8 slices across the four memory controllers; each LLC slice is 64 KB in size for a total LLC capacity of 512 KB. The SMs are connected to the LLC slices and memory controllers through an $12 \times 8$ crossbar network-on-chip (NoC). We assume a single-cycle latency penalty to compute the address transformation for all but the baseline address mapping scheme.

The simulated DRAM system is Hynix' GDDR5 [14] configured with 4 channels, 16 banks per channel, 4 K rows per bank and 64 columns per row for a total capacity of 1 GB. We further assume an open-page policy with an FR-FCFS memory controller. In one of our experiments, we also simulate a 3D-stacked memory system, configured the same as in prior work [8], consisting of 4 memory stacks with 16 vaults per stack and 16 banks per vault.

**Power modeling.** For evaluating GPU power, we use GPUWattch [23]. For computing DRAM power, we use the detailed power model developed by Micron [24] configured using Hynix' GDDR5 specification [14], as done in previous work [25]. The DRAM power model includes background power, refresh power, activation & precharge power, read power and write power.

**Workloads.** We use a set of GPU-compute workloads from three benchmark suites: CUDA SDK [18], Rodinia [19], and Parboil [21]; and one other source [20]. Table II reports the benchmarks' LLC APKI (LLC accesses per kilo-instructions),

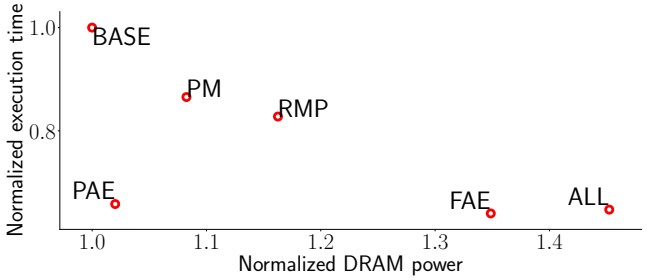



Figure 11: Performance vs. DRAM power consumption. *PAE achieves an average 1.52× speedup over Base while consuming 3% more DRAM power; FAE and ALL achieve slightly higher speedup, 1.56× and 1.54×, yet consume 35% and 45% more DRAM power, respectively.*

LLC MPKI (LLC misses per kilo-instructions), the number of kernel launches and the number of dynamically executed instructions in billions. The ten benchmarks at the top exhibit address bit entropy valley behavior; the bottom six do not, see also Figure 5.

## VI. Results

In the evaluation we consider the following address mapping schemes:

- **BASE**: the Hynix address mapping scheme [14].
- **PM**: permutation-based address mapping following [5], [4] which randomizes the bank and channel bits with the lowest order row bits.
- **RMP**: the bits with the highest average entropy across all benchmarks are used as channel and bank bits.
- **PAE**: randomized channel and bank bits using page bits as input.
- **FAE**: randomized channel and bank bits using the full address as input.
- **ALL**: randomized full address bits using the full address as input.

The evaluation is done in a number of steps. We first evaluate performance and power. We then dive into explaining the performance and power numbers. Finally, we perform a number of sensitivity analyses in which we vary the number of SMs; consider 3D-stacked memory; evaluate BIM sensitivity; and consider non-entropy valley benchmarks. When we report arithmetic or harmonic means in figures, they are computed across the benchmarks listed in the respective figure.

### A. Performance vs. Power

Figure 11 plots execution time as a function of DRAM power for the different address mapping schemes. The closer to the origin, the better. All address mapping schemes are normalized to BASE, for both performance and power.

The key take-away is that PAE achieves an average 1.52× speedup over BASE, while increasing DRAM power

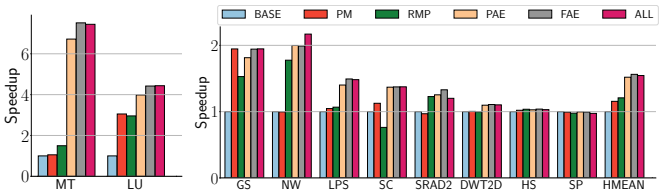
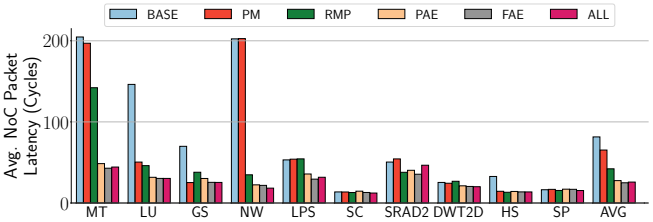
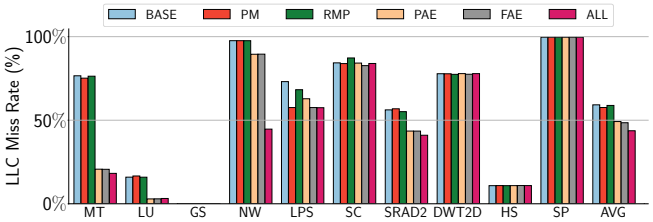



Figure 12: Per-benchmark speedup over BASE. *PAE, FAE and ALL lead to dramatic speedups for several benchmarks, averaging to 1.52×, 1.56× and 1.54×, respectively.*




(a) NoC packet latency




(b) LLC miss rate

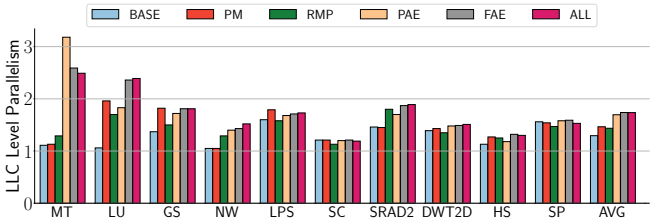
Figure 13: NoC packet latency and LLC miss rate. *PAE, FAE and/or ALL lead to a dramatic reduction for NoC packet latency and a substantial reduction in LLC miss rate.*

consumption by 3%. FAE achieves a slightly higher speedup than PAE (1.56×), however it consumes 35% more power. PM and RMP are not nearly as competitive: performance improves by 1.16× and 1.21× over BASE, while power consumption increases by 8% and 16%, respectively. ALL is comparable to FAE performance-wise, although it consumes 45% more DRAM power. PAE and FAE improve performance over state-of-the-art PM by 1.31× and 1.34×, respectively.

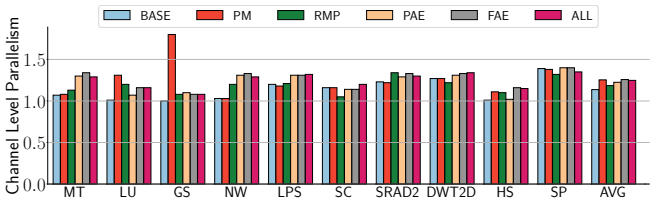
### B. Explaining Performance

Figure 12 reports per-benchmark performance results normalized to BASE. For the two benchmarks in the left graph, we observe dramatic speedups for PAE, FAE and ALL, up to 7.5×. These benchmarks are memory-intensive, see also Table II, and benefit dramatically from an improved address mapping. The other benchmarks are somewhat less memory-intensive, yet they still experience substantial performance speedups. On average, we report a speedup of 1.52×, 1.56× and 1.54× for the PAE, FAE and ALL address mapping schemes, respectively.

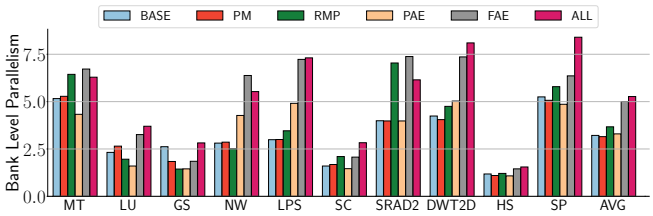
To understand where these performance improvements come from, we now take a closer look at a couple memory




(a) LLC-level parallelism




(b) Channel-level parallelism




(c) Bank-level parallelism

Figure 14: Memory-level parallelism. *PAE, FAE and ALL improve parallelism at the LLC, channel and bank level.*

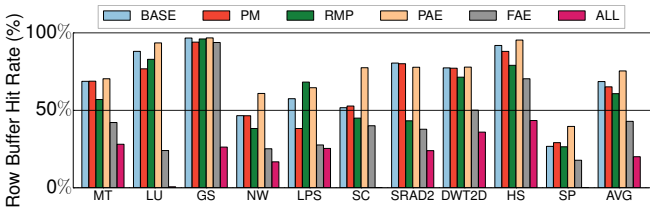



Figure 15: DRAM row buffer hit rate. *PAE achieves the highest row buffer hit rate, whereas FAE and ALL degrade row buffer locality.*

subsystem metrics. Figure 13 reports NoC packet latency and LLC cache miss rate; Figure 14 quantifies parallelism in the memory hierarchy, at the LLC, channel and bank level; and Figure 15 reports row buffer hit rate. The parallelism metrics are defined as the number of outstanding requests if at least one is outstanding. Note that bank-level parallelism is quantified per channel. This implies there is a multiplier effect when both channel-level and bank-level parallelism increase, i.e., the total number of outstanding requests is the number of parallel channel requests multiplied by the number of parallel bank requests per channel.

Address mapping affects the various memory hierarchy characteristics substantially, which collectively leads to the high performance improvements previously reported. Take

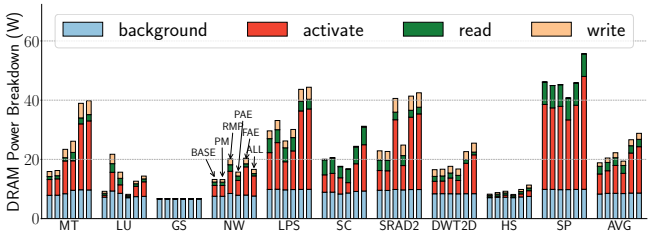



Figure 16: DRAM power consumption breakdown. *Address mapping primarily affects the activate power component; FAE and ALL increase activate power substantially.*

the MT and LU benchmarks as an example: under BASE, a single LLC slice receives most of the requests at a given point in time, i.e., LLC-level parallelism equals one, see Figure 14a. This implies that memory requests effectively serialize, which leads to a dramatically high LLC miss rate and NoC packet latency, see Figure 13. Address randomization through PAE, FAE and ALL distributes the accesses across the LLC slices which reduces the LLC miss rate (Figure 13b) and increases LLC-level parallelism (Figure 14a), which ultimately leads to a dramatic reduction in NoC packet latency (Figure 13a).

Figure 14c and Figure 15 reveal an interesting relationship between bank-level parallelism and DRAM row buffer hit rates. Load imbalance results in some banks having more DRAM requests than others which may cause frequent row switching and hit rate degradations. PAE improves row buffer hit rate because it creates sufficiently good load balancing while keeping good-locality requests within the same bank. FAE sometimes distributes good-locality requests to different banks. This causes additional row activations which reduces row buffer hit rates and increases power consumption compared to PAE.

This is also the underlying reason why FAE has a lower row buffer hit rate than PAE for MT (see Figure 15), even if FAE provides more address entropy (see Figure 14c and Figure 10). PAE provides slightly (1-2%) lower entropy than FAE for most bank and channel bits. Since the entropy function is logarithmic, a 2% drop in entropy equates to a 10% drop in address bit probabilities. Further, the effect of entropy loss is additive since each bit contributes independently to load imbalance. Thus, the small entropy reduction is responsible for PAE having larger load imbalance than FAE for MT in Figure 14c.

### C. Explaining Power

Figure 16 breaks down DRAM power consumption[3] into its four major components: background, activate, read and write power. GPU address mapping primarily affects the activate power component. We observe a small increase in DRAM power consumption for PAE over BASE (by 3%

[3]In our experiments, DRAM power consumption accounts for up to 40% of total system power.

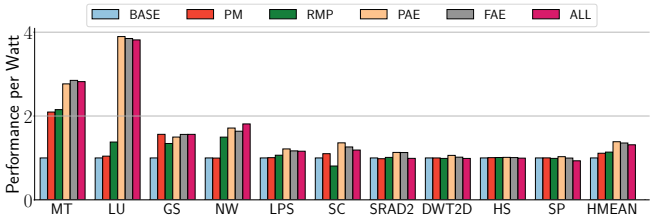



Figure 17: Normalized performance per Watt while considering total system (GPU+DRAM) power. *PAE, FAE and ALL improve performance per Watt by 1.39×, 1.36× and 1.31× on average, respectively.*

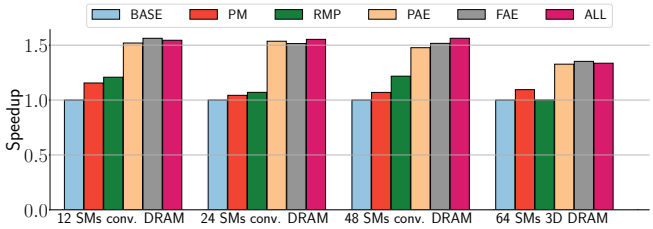



Figure 18: Performance sensitivity to the number of SMs and DRAM memory configuration. *PAE, FAE and ALL consistently improve performance across SM counts (from 12 to 64) and DRAM configuration (conventional memory organization versus 3D-stacked memory).*

on average). FAE and ALL on the other hand lead to a substantial increase in DRAM power consumption, by 35% and 45% on average, respectively. This increase is a result of the reduced row buffer hit rate, as quantified in the previous section, which leads to a substantial increase in the number of row activations.

We now revert to total system (GPU+DRAM) power consumption. System power consumption increases by 9%, 15% and 18% on average under PAE, FAE and ALL, respectively. However, because of the (much) higher performance improvement, this leads to a substantial improvement in performance per Watt, as reported in Figure 17. PAE, FAE and ALL improve performance per Watt by 1.39×, 1.36× and 1.31× on average, respectively, compared to BASE. PAE and FAE improve performance per Watt by 1.25× and 1.22× over state-of-the-art PM. This makes PAE the most power-efficient scheme.

### D. Sensitivity Analyses

**Varying the number of SMs.** We first vary the number of SMs from 12 to 24 and 48. Figure 18 reports average speedup of the various address mapping schemes over BASE at different SM counts (see the three sets of bars on the left). The proposed address mapping schemes consistently improve performance across SM count; performance seems to be somewhat lower at 48 SMs compared to 24 SMs as a result of increased saturation in the memory system.

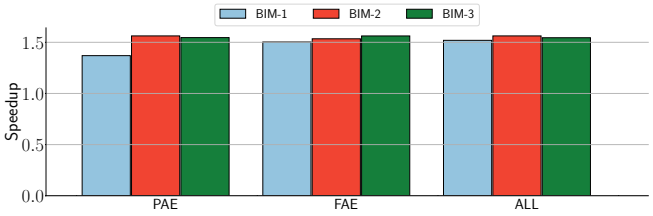



Figure 19: Average speedup for three randomly generated BIMs per address mapping scheme. *Different BIMs lead to similar performance improvements.*

**3D-stacked memory.** 3D-stacked memory leads to a dramatic increase in memory bandwidth compared to traditional DRAM. We now consider 3D-stacked DRAM with 4 memory stacks for a total of 640 GB/s; to make sure our system is balanced, we assume 64 SMs and a 960 GB/s NoC. Because 3D stacked memory is organized differently from conventional DRAM, we need to change the address mapping to randomize 2 channel bits, 4 vault bits and 4 bank bits. The rightmost set of bars in Figure 18 reports average speedup for the 3D stacked memory system with 64 SMs. PAE, FAE and ALL achieve consistently high performance improvement over BASE. RMP performs similarly to BASE in this configuration since applications do not have enough high entropy bits to achieve good load balancing.

**BIM sensitivity.** As mentioned in Section IV-B, BIMs are generated randomly. We now evaluate how sensitive performance is to the specific BIM. We consider three randomly generated BIMs for the PAE, FAE and ALL address mapping schemes, and report the resulting average speedup, see Figure 19. For FAE and ALL, the different BIMs result in roughly the same overall performance improvement. In other words, overall performance improvement is insensitive to the specific BIM. PAE seems to be (slightly) more sensitive. The reason is that PAE uses page address bits only, not the full address like FAE and ALL, hence we need to make sure that enough bit entropy information is incorporated into the new generated channel and bank bits. Note that even though PAE is slightly more sensitive to the specific BIM, even the lowest performing BIM still leads to a substantial performance improvement.

**Non-entropy valley benchmarks.** Not all workloads exhibit address bit entropy valleys, as mentioned in Section III. Nevertheless, it is important to demonstrate that the proposed address mapping schemes do not adversely affect the performance of such workloads. Figure 20 reports speedup for a set of non-entropy valley benchmarks. Note that these benchmarks are still memory-intensive, see Table II. The key observation is that address mapping has a relatively minor impact on these benchmarks, and PAE and FAE lead to small average performance improvements.

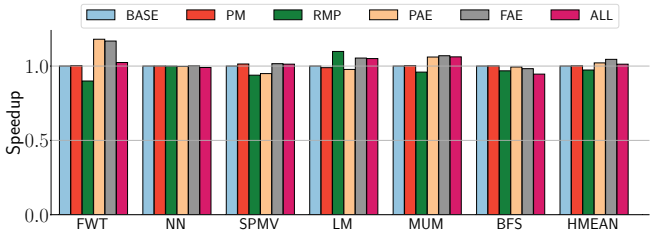



Figure 20: Normalized performance for non-entropy valley benchmarks. *The proposed address mapping schemes do not affect performance for benchmarks that do not exhibit address bit entropy valleys.*

## VII. Related Work

**Address mapping schemes.** DRAM address mapping schemes have been explored for single and multi-core CPUs. The permutation-based mapping scheme [5] increases the entropy of the bank address bits by taking the XOR of each bank bit and a single row bit. In addition, researchers have investigated the impact of changing the address bits that map to the bank, channel and column address fields. For instance, Kaseridis et al. [7] find that moving the bank and channel bits closer to the least significant bits allow streaming applications to improve their bank-level parallelism. For GPUs, Chatterjee et al. [4] propose to extend the XOR-operations of the PM-scheme to also cover channel bits. We compare against all of these schemes in our evaluation and show that they are unable to achieve good bank and channel balance for GPU-compute workloads.

**Quantifying the entropy of memory address bits.** Measuring the bit-level entropy of collections of memory addresses has been investigated by prior work. Both Akin et al. [12] and Ghasempour et al. [26] monitor the bit flip rate of the memory access stream and use this to estimate entropy. Bit flip rate is difficult to apply to a GPU as memory requests from different TBs are generated in parallel. Hence, they interleave in many different ways, causing unreliable bit flip measurements. Our approach avoids this problem by computing the window-based entropy.

**Memory address reorganization.** A large body of research has targeted ways of remapping memory addresses for different purposes. The most similar to this work are Akin et al. [12] and Qureshi et al. [13]. Akin et al. use the BIM-abstraction in a reshape accelerator implemented within 3D-stacked DRAM, while Qureshi et al. use it as a component in a wear-leveling technique for phase change memory. In this work, we provide another use case for matrix-based address transformations.

In addition, a number of researchers have proposed to reorganize the data layout to improve performance [27], [28]. Another line of research focuses on restructuring loop nests to improve parallelism and locality [29], [30], [31]. Jang et al. [32] use data transformation to optimize memory access patterns in loop bodies for GPU-compute workloads, and Sung et al. [33] propose a compiler tool that automatically reorganizes data layout. Finally, researchers have proposed techniques that dynamically reorganize the data layout to improve performance or energy-efficiency [34], [35], [36], [26]. Our mapping scheme reduces the need for such approaches since it limits the performance loss caused by poor application-level memory organization.

**Memory request scheduling.** A large body of research has investigated memory request scheduling techniques. Rixner et al. [17] propose the First-Ready First-Come-First-Served (FR-FCFS) scheduler which dynamically reorders memory requests to improve row buffer hit rates. For CPUs, a number of researchers have improved upon FR-FCFS to better exploit row buffer locality, better utilize bank-level parallelism or improve fairness among co-executing processes [37], [38], [39], [40], [41], [42]. For GPUs, Lakshminarayana et al. [43] propose the Shortest-Job-First (SJF) scheduling policy. SJF dynamically trades off the latency of completing all memory requests of a warp against the bandwidth utilization benefits of FR-FCFS. Chatterjee et al. [10] propose a static reordering scheme to reduce the toggling rate in DRAM. Scheduling is orthogonal to address mapping because it attempts to increase row buffer hit rates while address mapping attempts to evenly distribute memory requests across channels and banks.

**Bandwidth-oriented DRAM organization.** A number of researchers have proposed improvements to DRAM organization. Kim et al. [44] propose a mechanism that exposes parallelism at the DRAM-subarray level, and O'Connor et al. [9] propose a DRAM organization which enables using the bandwidth of the DRAM banks simultaneously. Chatterjee et al. [10] observe that GPUs tend to have low row buffer hit rates and therefore propose to reduce the activation-granularity to save energy. In summary, these proposals suggest to increase the number of independent memory system units to achieve higher bandwidth. Address mapping techniques like ours will presumably combine favorably with such systems since ensuring an even request distribution becomes more important as the number of parallel units increases.

## VIII. Conclusion

In this work, we study and improve GPU memory address mapping. To comprehensively analyze GPU address behavior, we devise a novel entropy metric, called window-based entropy, tailored for highly parallel GPU-compute workloads. Window-based entropy quantifies the entropy of each address bit across the memory requests within and across TBs that are likely to co-exist in the memory subsystem at run time. We compute window-based entropy for a broad set of GPU-compute workloads and find that application-dependent entropy valleys exist distributed throughout the lower order

bits. Based on this key insight, we devise novel GPU address mapping schemes that harvest entropy from broad address-bit ranges and concentrate it into the address bits used for bank and channel selection to maximize parallelism in the memory subsystem. We propose the PAE, FAE and ALL address mapping schemes which cover different combinations of input and output address-bit-field selection. PAE considers bits from the DRAM page address to create well-distributed channel and bank bits, and is the most power-efficient mapping scheme, improving performance per Watt by 1.25× compared to PM, the state-of-the-art mapping scheme [4], [5]. FAE considers bits from the full address and is the highest performing mapping scheme, improving performance by 1.34× compared to PM.

## ACKNOWLEDGEMENTS

We thank the reviewers for their thoughtful comments and suggestions as well as extend our gratitude to Moinuddin Qureshi for his valuable feedback. This work is supported by the European Research Council (ERC) Advanced Grant agreement No. 741097, FWO projects G.0434.16N and G.0144.17N, NSFC under Grant No. 61232008, 61472008, 61672053 and U1611461, NSF CSR-1618384 and the 863 Program of China under Grant No. 2015AA015305, Shenzhen Key Research Project No. JCYJ20170412150946024. Yingwei Luo is also affiliated with Shenzhen Key Lab for Cloud Computing Technology & Applications, SECE, Peking University.

## REFERENCES

[1] JEDEC, "JEDEC Standard JESD212: GDDR5 SGRAM." https://www.jedec.org/system/files/docs/JESD212.pdf, 2009. JEDEC Solid State Technology Association, Virginia, USA.

[2] "NVIDIA Tesla P100." https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf, 2016. Nvidia.

[3] "NVIDIA TESLA V100." https://images.nvidia.com/content/volta-architecture/pdf/Volta-Architecture-Whitepaper-v1.0.pdf, 2017. Nvidia.

[4] N. Chatterjee, M. O'Connor, G. H. Loh, N. Jayasena, and R. Balasubramonian, "Managing DRAM Latency Divergence in Irregular GPGPU Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 128–139, 2014.

[5] Z. Zhang, Z. Zhu, and X. Zhang, "A Permutation-based Page Interleaving Scheme to Reduce Row-buffer Conflicts and Exploit Data Locality," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 32–41, 2000.

[6] C. E. Shannon, "Prediction and Entropy of Printed English," *Bell Labs Technical Journal*, vol. 30, no. 1, pp. 50–64, 1951.

[7] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 24–35, 2011.

[8] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 204–216, 2016.

[9] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017.

[10] N. Chatterjee, M. O'Connor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally, "Architecting an Energy-Efficient DRAM System for GPUs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 73–84, 2017.

[11] L. Yen, S. C. Draper, and M. D. Hill, "Notary: Hardware Techniques to Enhance Signatures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 234–245, 2008.

[12] B. Akin, F. Franchetti, and J. C. Hoe, "Data Reorganization in Memory Using 3D-stacked DRAM," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 131–143, 2015.

[13] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 14–23, 2009.

[14] "Hynix GDDR5 SGRAM Part H5GQ1H24AFR Revision 1.0." http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf, 2009. Hynix.

[15] A. Li, S. L. Song, W. Liu, X. Liu, A. Kumar, and H. Corporaal, "Locality-Aware CTA Clustering for Modern GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 297–311, 2017.

[16] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-Conscious Wavefront Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 72–83, 2012.

[17] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 128–138, 2000.

[18] "NVIDIA CUDA SDK Code Samples." https://developer.nvidia.com/cuda-downloads. Nvidia.

[19] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *International Symposium on Workload Characterization (IISWC)*, pp. 44–54, 2009.

[20] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU Microarchitecture through Microbenchmarking," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 235–246, 2010.

[21] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," *Center for Reliable and High-Performance Computing*, 2012.

[22] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 163–174, 2009.

[23] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch : Enabling Energy Optimizations in GPGPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 487–498, 2013.

[24] "Calculating Memory System Power for DDR3." https://www.micron.com/~/media/documents/products/technical-note/dram/tn41_01ddr3_power.pdf, 2007. Micron.

[25] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 86–98, 2013.

[26] M. Ghasempour, A. Jaleel, J. D. Garside, and M. Luján, "DReAM: Dynamic Re-arrangement of Address Mapping to Improve the Performance of DRAMs," in *Proceedings of the International Symposium on Memory Systems (MEMSYS)*, pp. 362–373, 2016.

[27] I.-J. Sung, G. D. Liu, and W.-M. W. Hwu, "DL: A Data Layout Transformation System for Heterogeneous Computing," in *Proceedings of the Innovative Parallel Computing (InPar), 2012*, pp. 1–11, IEEE, 2012.

[28] B. Akin, F. Franchetti, and J. C. Hoe, "FFTs with Near-Optimal Memory Access Through Block Data Layouts," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3898–3902, 2014.

[29] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 2, no. 4, pp. 452–471, 1991.

[30] M. E. Wolf and M. S. Lam, "A Data Locality Optimizing Algorithm," in *Proceedings of the International Conference on Programming Language Design and Implementation (PLDI)*, pp. 30–44, 1991.

[31] K. S. McKinley, S. Carr, and C.-W. Tseng, "Improving Data Locality with Loop Transformations," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 18, no. 4, pp. 424–453, 1996.

[32] B. Jang, D. Schaa, P. Mistry, and D. Kaeli, "Exploiting Memory Access Patterns to Improve Memory Performance in Data-Parallel Architectures," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 1, pp. 105–118, 2011.

[33] I.-J. Sung, J. A. Stratton, and W.-M. W. Hwu, "Data Layout Transformation Exploiting Memory-Level Parallelism in Structured Grid Many-Core Applications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 513–522, 2010.

[34] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, "Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 219–230, 2010.

[35] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page Placement in Hybrid Memory Systems," in *Proceedings of the International Conference on Supercomputing (ICS)*, pp. 85–95, 2011.

[36] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, 2010.

[37] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair Queuing Memory Systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 208–222, 2006.

[38] N. Rafique, W.-T. Lim, and M. Thottethodi, "Effective Management of DRAM Bandwidth in Multicore Processors," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 245–258, 2007.

[39] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 146–160, 2007.

[40] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 63–74, 2008.

[41] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, 2010.

[42] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 65–76, 2010.

[43] N. B. Lakshminarayana, J. Lee, H. Kim, and J. Shin, "DRAM Scheduling Policy for GPGPU Architectures Based on a Potential Function," *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 33–36, 2012.

[44] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 368–379, 2012.