Optimal Data Layout for Block-Level Random Accesses to Scratchpad

Shreyas G. Singapura[1], Rajgopal Kannan[2], and Viktor K. Prasanna[1]

[1]University of Southern California, Los Angeles, CA 90089
[2]Army Research Lab-West, Los Angeles, CA 90094
[1]{*singapur, prasanna*}*@usc.edu*
[2]*rajgopal.kannan.civ@mail.mil*

Abstract*—3D memory is becoming an increasingly popular technology to overcome the performance gap between memory and processors. It has led to the development of new architectures with scratchpad memory, which offer high bandwidth and user-controlled access features. The ideal performance of this scratchpad memory is peak bandwidth for any random block access. However, 3D memories come with their constraints on the "ideal" access patterns for which high bandwidth is guaranteed and the actual bandwidth is significantly lower for other access patterns. In this paper, we address the challenge of achieving high bandwidth for random block accesses to 3D memory. We present optimal data layout which achieves maximum bandwidth for each vault irrespective of the block accessed in a vault. Our data layout expressed as a mapping function determined by the architecture parameters exploits inter-layer pipelining to map the elements of each block among various layers of a vault in a specific pattern. By doing so, our data layout can absorb the latency of accesses to banks in the same layer and more importantly, hide the latency of accesses to different rows in the same bank irrespective of the block being accessed. We compare the performance of our proposed data layout with existing data layout using PARSEC 2.0 benchmarks. Our experimental results demonstrate as high as 56% improvement in access time in comparison with the existing data layout across various workloads.

I. Introduction

Memory performance has been lagging behind processor performance and the gap is increasing further due to faster and multiple cores [1]. The problem is two-fold with bandwidth and latency of memory not able to keep up with the speed of processors' need for data. This gap has given rise to the Memory wall getting stronger and has been a major concern for the last decade.

Recently, 3D memories [2], [3] have emerged which aim to address at least one limitation of the memory performance. These new memories claim to provide a bandwidth equivalent to $10\times$ the current memories [2], [3]. However, 3D memories cannot replace DDR3/4 due to constraints on the capacity. Further, the latency of 3D memories is in the same range as that of existing memories. These unique features have resulted in new type of architectures which use 3D memories as a user-controlled "scratchpad" memory at different levels of memory hierarchy. The Intel Knights Landing [4], [5] is one such example using 3D memory as a scratchpad.

This material was supported by the NSF under Grant Number CNS-1643351.

When incorporating a scratchpad, under ideal conditions, 3D memory should act as a large cache with a uniform bandwidth and latency for accessing any block in any access pattern as desired by the application. However, the high bandwidth of 3D memories come with caveats in terms of access patterns and a random access pattern does not necessarily achieve high bandwidth [6], [7].

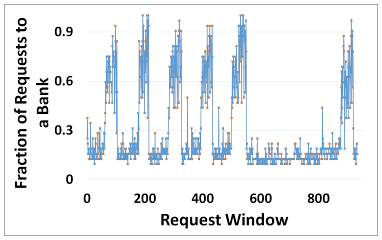
In this paper, we address the challenge of extracting high bandwidth from 3D memories for block level random access patterns. We observe from the structure of 3D memory that inter-layer pipelining can be exploited to hide the latency overhead of consecutive accesses to the same bank and layer in a vault. We develop an optimal data layout to achieve uniform bandwidth to access any block in a vault of 3D memory with the following key features: (1) Consecutive elements of each block are mapped to different layers rather than the same layer to exploit faster access across 3^{rd} dimension; (2) Accesses to the same layer are separated by sufficient number of intermediate accesses to other layers to absorb the access overhead. The main contributions of this paper are:

- We propose a new addressing scheme targeting block accesses to 3D memory.
- We present an optimized data layout for 3D memory that achieves high bandwidth for random block accesses in a vault.
- We mathematically prove that our proposed data layout is optimal for random block accesses.
- Our data layout achieves up to 56% improvement in memory access time compared with baseline data layout on variety of workloads in PARSEC 2.0 benchmarks.

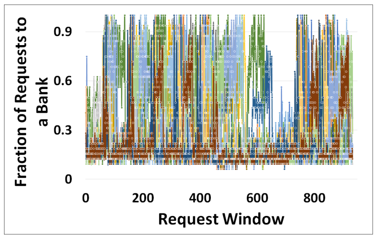
II. Background

A. 3D Memory

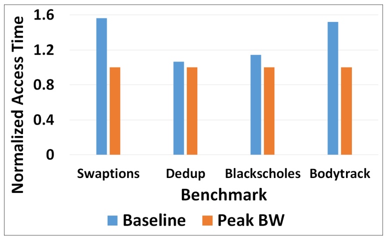
3D memory is a new type of memory consisting of stacks of memory layers and has become a viable and commercial option in recent times. Several versions of 3D memory have been proposed by the industry [2], [3]. Although, the organization and terminology can vary among different 3D memories, the common factor is the multi-layer stacking, and parallel access to groups of banks. Therefore, to keep our analysis platform independent and focus on a generic 3D memory, we use the



(a) Max. Percentage of Requests to a Bank in a Vault across Windows



(b) Request Distribution for all Vaults (different colors) across respective windows



(c) Performance Comparison of Existing Data Layout vs Peak Bandwidth

Fig. 1: Memory Access Statistics for Baseline Data Layout on *bodytrack* Benchmark

parameterized model developed in our previous work [8]. For completeness, we provide an overview of the model used for performance analysis in this paper.

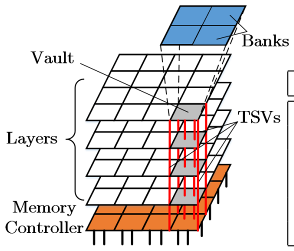


Fig. 2: 3D Memory

Notation	Definition
v	# vaults
l	# layers in a vault
b	# banks per layer
r	# rows in a bank
c	# columns in a row
col	size of a column fetch

TABLE I: 3D Memory Parameters

1) Architecture Parameters: The various parameters of the memory are listed in Table I. The vaults (v) represent the set of banks that can be accessed in parallel. The layers (l) in a vault share a set of TSVs used to access banks in that vault. Structure within a layer of a vault is similar to existing planar memories with multiple banks (b) on each layer and each bank divided into rows (r) and columns (c) to store the data. Each access to a bank results in a column of data (col) being available. A representative architecture of 3D memory with v=16, l=4, b=4 is illustrated in Figure 2.

2) Timing Parameters: 3D memory is organized as a set of vaults which can be accessed independently, wherein v elements can be accessed in parallel from v vaults. Further, due to the third dimension in 3D memories and fast vertical interconnnects, banks in different layers can be activated in quick succession by overlapping the activation overhead of different banks. Therefore, accessing data from different layers is faster in comparison with accessing data in the same layer [9], [10], [11]. Within a layer of a vault, the organization of 3D memory resembles that of DDR3. Banks can be accessed independently and, among the accesses to the same bank, data in the same row (different columns) can be accessed faster than accessing data from different rows. The timing parameters of 3D memory are:

- t_{vault}: time between accesses to *different vaults*
- t_{layer}: time between accesses to *different layers* in a vault
- t_{bank}: time between accesses to *different banks* in a layer in the same vault
- t_{row}: time between accesses to *different rows* in a bank
- t_{col}: time between accesses to *different columns* in a row

The above parameters (except t_{row}) are defined assuming the rows being accessed are already active.

B. Motivation

Although 3D memories achieve high peak bandwidth, the sustained bandwidth is dependent on the access pattern of the application. For example, sequential accesses will result in uniform data retrieval from the vaults, layers within a vault and banks within a layer. The high bandwidth can be achieved even for random access patterns provided the requests are uniformly distributed among the vaults, layers and banks. This is possible because the memory controller of each vault can store a window of requests and reorder the requests among banks to ensure high bandwidth is achieved. Since data from vaults can be retrieved independently, the distribution of requests to banks in a vault within a window is a critical parameter for memory bandwidth. The vault controller can only perform reordering of the requests in a single window. Above a certain threshold of number of requests mapped to the same bank, the optimization of reordering by the vault controller is limited by the bank timing parameters.

We present a motivational example to demonstrate the significance of distribution of memory requests. In Figure 1, we illustrate the access pattern statistics and normalized access time for the memory trace of *bodytrack* from PARSEC 2.0 benchmark. Here, we assume a 3D memory consisting of v=32, l=4, b=4, r= 2048, col=16 B and each request retrieves a block of 1024 B (or 64 columns) from the memory. The simulation is performed using an in-house simulator described in Section VI-B [12]. As illustrated in Figure 1a, we observe that for a given vault, the percentage of requests mapped to the same bank in a window ranges from a low value of 15% to a high value of 95%. Therefore, there are windows with almost all the requests mapped to the same bank and as described earlier, this will stifle the vault bandwidth to that of a single bank. Further, as shown in Figure 1a, these windows can form a substantial portion of the trace. This problem is exacerbated by the fact that this distribution is seen among all the vaults of 3D memory as illustrated in Figure 1b which plots the distribution of requests for all the vaults (different colors) across their respective windows.

The effect of this non-uniform distribution of requests among the banks of a vault on the overall bandwidth of the 3D memory is depicted in Figure 1c. The normalized access time is calculated assuming the peak bandwidth of 3D memory which is achieved for sequential or uniformly distributed access patterns. The access pattern with non-uniform distribution is not limited to a single benchmark as shown in Figure 1c. In this paper, we address this challenge of attaining high bandwidth irrespective of the distribution of requests to the banks within a vault.

C. Scratchpad Memory Model

In [13], [14], the authors have proposed the scratchpad memory model. For the sake of completeness, we provide an overview of the model. The scratchpad memory has similar latency as the DRAM but higher bandwidth. The granularity of accesses to scratchpad memory is larger than the DRAM which results in a higher bandwidth in comparison with DRAM. The scratchpad is assumed to be user-controlled with the data in the scratchpad decided by the application developer. Therefore, the scratchpad can be viewed as an organization of blocks and the access requests to the scratchpad are addresses to these blocks in any random order. An example of a scratchpad of 4 GB (32 address bits) with 4 million blocks and 1024 B in each block is represented in Figure 3. In terms of 3D memory, an element is equal to a column of the bank.

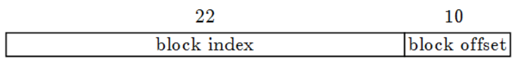


Fig. 3: Address Scheme: Application Developer View

III. Related Work

Addressing scheme optimizations have been studied in many works. In [15], [16], addressing schemes are proposed to target applications such as turbo decoders and FFT respectively. In [17], [18], row buffer conflicts and locality are improved. In the context of 3D memory, address remapping was studied in [19] to improve bandwidth for operations such as shuffle, swap, and transpose. In contrast, we target generic addressing scheme to achieve high bandwidth independent of the target application.

Different addressing schemes to extract higher bandwidth from 3D memory have been studied as well. In [6], permutation based address scrambling is used to generate uniformly random addresses to minimize bank conflicts in 3D memory. However, this method does not solve the problem caused due to accesses mapped to the same vault. And, this problem is more important in the context of scratchpad memory as the expectation is to access any block with uniform bandwidth which can lead to consecutive accesses mapped to the same vault. In [10], the effect of splitting a single block across layers is compared against block mapped to single layer. However, a generic addressing scheme or a data layout to optimize bandwidth of 3D memory for random block access patterns is not defined. In [20], inter-layer pipelining is not exploited which can result in significantly lower bandwidth for 3D memory.

Scratchpad (3D memory) based architectures have also been studied in the literature. In [13], [14], authors demonstrate the effectiveness of scratchpad architectures for K-means clustering and Sorting. It is assumed that the 3D memory can sustain peak bandwidth for the access pattern of the target application. However, this is not valid for all applications [6]. In our previous work [8], we developed optimized data layout for FFT on 3D memory. In this paper, we extend our work to develop optimized data layout targeting random block accesses using a generic addressing scheme which absorbs access overhead of various components of 3D memory. Our data layout delivers high bandwidth for each vault independent of the application and its access patterns.

IV. Problem Statement

For a vault in 3D memory consisting of a set of N blocks of e elements in each block, the goal is to develop optimal data layout which achieves the maximum bandwidth for any random order of requests to these N blocks.

V. Data Layouts

In this section, we describe the existing data layout and present its limitations for random block access patterns. Later, we present our proposed Optimized data layout and its addressing scheme. For notational simplicity, we assume N, e, v, l, b, r and c are powers of 2.

*A. **Baseline Data Layout***

In the existing 3D memory data layout and addressing scheme, all the elements of a block are mapped to the same row of a bank [6]. A location (address of an element) is defined by the quintuple $\{v(a_{ij}), l(a_{ij}), b(a_{ij}), r(a_{ij}), c(a_{ij})\}$ which maps element a_{ij} to a vault, layer, bank, row and column in the 3D memory. We assume block size $e \leq lbc$, i.e., multiple blocks can fit in a single row across all banks in a vault [2]. DL 1 presents the mapping scheme of Baseline data layout.

DL 1: Baseline Data Layout

```
$a_i[j]$ : $j^{th}$ element in $i^{th}$ block, $j \in [0, e-1]$, $i \in [0, N-1]$, $e$
    elements in a block, $N$ blocks in a vault
$v(a_i[j])$ is same $\forall\ a_i[j]$
Address[$a_i[j]$] $\rightarrow \{l(a_i[j]), b(a_i[j]), r(a_i[j]), c(a_i[j])\}$
$l(a_i[j]) = i \bmod l$
$b(a_i[j]) = \lfloor \frac{i}{l} \rfloor \bmod b$
$r(a_i[j]) = \frac{i}{l \cdot b}$
$c(a_i[j]) = j$
```

Based on this data layout, the addressing scheme is illustrated in Figure 4. We have assumed a scratchpad of $4GB$ with v=32, l=4, b=4, r=2048, c=4, col=16B [2], [21] resulting in a block of size 1024 B (or 64 columns of data).

Limitation: This approach does not allow us to exploit inter-layer pipelining which maps requests to different layers and access data from the layers in a pipelined fashion. It has

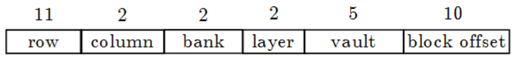


Fig. 4: Address Scheme: Baseline Data Layout

been observed that inter-layer pipelining [11], [10] can significantly improve the bandwidth of 3D memory since requests to different layers are limited by the t_{layer} which is much smaller compared with other timing parameters. Therefore, accesses to a vault need to be equally distributed among banks across layers in each request window for the vault controller to reorder the requests to obtain high bandwidth. However, since the distribution of accesses is application dependent, the bandwidth available can be much lower if there is non-uniform distribution of requests to banks.

Note: The ideal data layout would be to distribute the elements of a block across different vaults to access them in parallel. Although this data layout looks promising, distributing a single block across vaults will curtail the independence of vaults. The vaults will then proceed in lock step fashion with any access to an inactive row in a vault causing all the vaults to stall. Further, with the 3D memory expected to be used in high performance computing, there are multiple cores connected to the memory and at any point of time there are multiple requests in the memory. Mapping a block to a single vault restricts the stalls due to activation of an inactive row to a single core and more importantly, other vaults can proceed with their respective memory requests independently. Secondly, distributing elements of a block across vaults would mean that each block accessed would need to be assembled with inputs from different vaults and increases the complexity of interface between memory and processors.

B. *Optimized Data Layout*

Our optimized data layout is defined in terms of an addressing scheme which maps elements of a block to specific locations in the vault of the 3D memory. To derive our addressing scheme for optimized data layout, we make the following assumptions based on the current specifications of 3D memory [2], [3]: $t_{layer} < \{t_{bank}, t_{col}\} < t_{row}$ and $l \cdot t_{layer} \geq \{t_{col}, t_{bank}\}$.

Our proposed data layout ensures that consecutive accesses to 3D memory components (layers, banks) are sufficiently spaced to absorb activation overheads irrespective of the block being accessed in the vault. Our mapping scheme distributes elements of each block among banks across layers in a vault in a specific pattern to ensure maximum bandwidth for random accesses to blocks in that vault.

Considering a row across all banks and layers in a vault, there are lbc columns in this row. For a given block of size e columns, we want to distribute its elements uniformly among these lbc locations. We define the following parameter $x = \{\min_{1 \leq s \leq c}(s) \mid sl(b-1)t_{layer} \geq t_{row}\}$. Let $y = 2^{\lceil \log_2 x \rceil}$, i.e., x rounded to the nearest power of 2. In our layout, number of elements in a block $e = lby$, block size is $e \cdot col$ and y

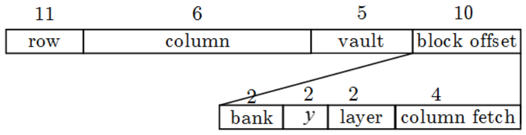


Fig. 5: Address Scheme: Proposed Data Layout

represents the number of consecutive accesses to the same row in a bank of a layer before the next bank in that same layer is accessed. Our mapping scheme is described in detail in DL 2 and the corresponding address scheme is illustrated in Figure 5.

DL 2: Optimized Data Layout

```
1 $a_i[j]$ : $j^{th}$ element in $i^{th}$ block, $j \in [0, e-1]$, $i \in [0, N-1]$, $e$
    elements in a block, $N$ blocks in a vault
2 $v(a_i[j])$ is same $\forall\ a_i[j]$
3 Address$[a_i[j]] \rightarrow \{l(a_i[j]), b(a_i[j]), r(a_i[j]), c(a_i[j])\}$
4 $\{y\}$ // block related parameter
5 $l(a_i[j]) = j \bmod l$
6 $b(a_i[j]) = \left\lfloor \frac{j}{y \cdot l} \right\rfloor \bmod b$
7 $r(a_i[j]) = \left\lfloor \frac{i}{(c/y)} \right\rfloor$
8 $c(a_i[j]) = \left(i \bmod \frac{c}{y}\right) \cdot y + \left\lfloor \frac{j}{l} \right\rfloor \bmod y$
```

1) Numerical Example: Based on the timing values of 3D memory [2], we choose timing parameters as follows: t_{layer}=1ns, t_{bank}=4ns, t_{col}=4 ns, t_{row}=40ns. The architecture parameters are: l=4, b=4, each column fetches an element, i.e., col (each element) =16 bytes. These values translate to a peak vault bandwidth of $\frac{col}{t_{layer}}$=16 GB/s. The block parameters of our data layout are y=4, e=64 and block size $(e \cdot col)$=1024 B.

64 elements of a block are logically divided into 4 sets with 16 elements in a set and each set is mapped to 4 banks of the same index across layers. 1st set of a block is mapped to banks indexed 1 across layers; 2nd set is mapped to banks indexed 2 from all layers and so on. This is repeated for all the blocks. Among the 16 elements in a set, 4 elements are mapped to a bank in each layer with a stride equal to 4. For example, in the 1st set, elements 1, 5, 9, 13 are mapped to bank indexed 1 in layer 1; elements 2, 6, 10, 14 are mapped to bank indexed 1 in layer 2. Considering elements in the same block, accesses to a bank are separated by 4 elements which are mapped to different layers. Therefore, the time difference is 4ns $(4 \cdot t_{layer})$ which is sufficient to hide t_{col} of a bank. With respect to different blocks, accesses to the same bank in a layer are separated by accesses to 3 sets of 16 elements, i.e., 3×16=48 elements which are mapped to different layers in a round robin fashion. The worst performance in terms of bandwidth occurs when consecutive accesses to a bank are to different rows, i.e., 2 blocks mapped to different rows. Based on the above example, considering any 2 blocks, the accesses to the same bank occurs after 48 accesses to other layers. This is equivalent to a time of 48ns $(48 \cdot t_{layer})$ and based on

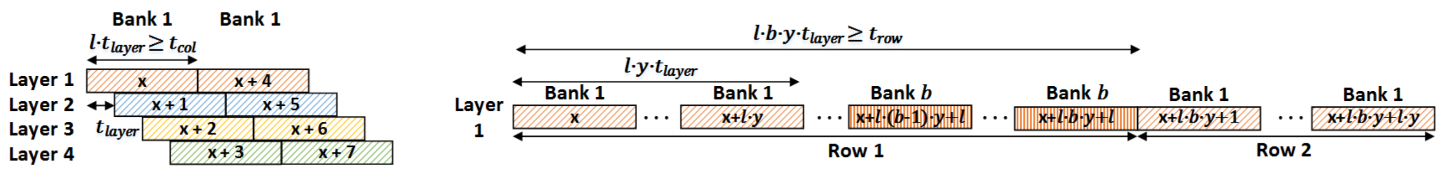


Fig. 6: Data Layout for hiding (a) t_{col} and (b) t_{row} overhead

timing parameters, this is sufficient to hide the row activation overhead of 40ns. Therefore, accesses to any of the blocks in a vault in any random access pattern will result in a maximum bandwidth of 16 GB/s.

2) Mathematical Analysis: In this section, we prove mathematically that our optimized data layout allows any blocks in a vault to be accessed at peak bandwidth. We show that t_{col} and t_{row} are absorbed when elements mapped to the same row or different rows of a bank are accessed.

Lemma 1. *Elements in a block mapped to different layers have indices with stride of 1.*

Proof. This follows from Line 5 of DL 2.

Lemma 2. *Elements in a block mapped to the same bank and layer have indices with a stride of l.*

Proof. Based on Line 6 of DL 2, the elements mapped to the same bank index ($b(a_i[j])$) are separated by $l \cdot y$. Based on Lemma 1, consecutive elements of a block are mapped to different layers and elements mapped to same layer have a stride of l. Since $y > 1$, indices of elements in the same bank and layer differ by a value of l.

Lemma 3. *Elements in a block are mapped to the same row. Elements mapped to different rows belong to different blocks.*

Proof. Based on Line 8 of DL 2, elements in a block are mapped to the same row and elements in different rows belong to different blocks. Also, (c/y) blocks are mapped to the same row index.

We will utilize these lemmas to derive theorems and mathematically prove optimality of our proposed data layout.

Theorem 1. *Latency of accesses to elements in the same block is t_{layer}.*

Proof. Based on Lemma 1, elements with consecutive indices are mapped to different layers. Since the access time to elements in different layers is t_{layer}, elements in a block can be accessed consecutively in intervals of t_{layer}.

Theorem 2. *Latency of accesses to elements in a block mapped to the same bank in a layer is $l \cdot t_{layer}$ and is hidden by accesses to other layers.*

Proof. Using Lemma 2, accesses to the same bank are separated by l requests. Combining Lemma 1 and Theorem 1, the time span between accesses to the same bank is equal to $l \cdot t_{layer}$. Based on the timing parameters of 3D memory, $l \cdot t_{layer} > t_{col}$, i.e., the access time to different columns of a bank is hidden by accesses to other layers.

Theorem 3. *Latency of accesses to elements mapped to different rows in a bank are separated by a time span $\geq t_{row}$ and is hidden by accesses to other layers.*

Proof. Based on Lemma 3, elements mapped to the different rows belong to different blocks. Hence, indices of elements in different rows have a stride of $(c/y) \cdot lby$, i.e., lbc which is greater than the block size lby. Out of the lby elements in a block, only y elements are mapped to the same bank in a layer (Lemma 2). Hence, at least $l(b-1)y$ elements have to be retrieved from other banks before accessing same bank. Further, these elements are accessed at t_{layer} time interval (Theorem 1). Therefore, the time difference between accessing elements in different rows of a bank is $l(b-1)yt_{layer}$. This value is sufficient to hide the row activation overhead of t_{row}.

Figure 6 illustrates the mapping of elements of a block using our optimized data layout to hide overhead of t_{col} and t_{row}. It can be observed that the indices of consecutive elements accessed in Bank 1 of Layer 1 are separated by 4. Similarly, indices of elements in different rows of Bank 1 of Layer 1 are separated by a value of $l(b-1)y$, i.e., 48. These values prove that the access overhead of 4ns (t_{col}) and 40ns (t_{row}) are hidden using our optimized data layout.

Theorem 4. *Access to any random block in a vault will be retrieved at the peak bandwidth.*

Proof. Based on Lemma 3, random access to different blocks can result in 2 scenarios: (1) blocks belong to same row of a bank (2) blocks belong to different rows of a bank. Based on Theorem 2, accessing elements of blocks in the same row of a bank hides the access latency of t_{col}. Theorem 3 proves that accessing elements in different rows hides the access latency of t_{row}. Further, based on Theorem 1, consecutive elements can be accessed at t_{layer} time intervals. Therefore, irrespective of the access pattern, blocks can be accessed at $\frac{16B}{t_{layer}}$, i.e., 16 GB/s or peak vault bandwidth.

Figure 7 illustrates the mapping of elements of 2 blocks to different rows of a bank in the same layer using our optimized data layout. It can be observed that elements in different blocks mapped to different rows of the same bank are separated by $l(b-1)y$=48 accesses to elements in other banks. Closing Row 1 and opening Row 2 requires 40ns (t_{row}) which is smaller

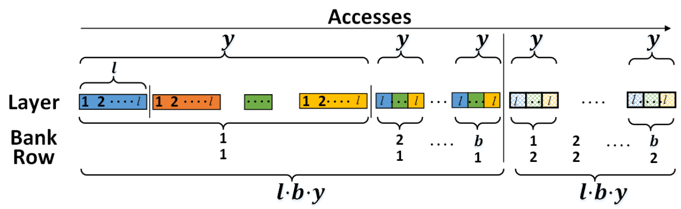


Fig. 7: Data Layout for 2 blocks in 2 rows

48·t_{layer}=48ns. Therefore, blocks in different rows of a bank can be accessed at peak bandwidth of the vault.

VI. Evaluation

A. Performance Metric

We compare the performance of Baseline and Optimized data layouts using **Memory Access Time** as the metric. It is defined as the total amount of time to retrieve a trace of requests from the memory.

B. Experimental Setup

3D memories have become popular recently and their simulators are a work in progress. Simulators in [22], [23] do not differentiate between accesses across layers and within layers. HMCSim [24] does not reveal the internal architecture details due to Intellectual property rights. These simulators do not support all the features of 3D memory and do not allow us to modify the addressing scheme as required by our data layout.

1) Simulator: We have developed an in-house simulator [12] based on the model described in Section II-A to evaluate the performance of Baseline and Optimized data layouts. It accepts a trace of memory requests as input and outputs the retrieval time from memory. Each memory request in the trace contains the address of the block to be retrieved. The memory controller in the simulator is based on FCFRFS policy and performs reordering among requests in a window to reduce the memory access time. The vaults can be accessed in parallel and it is assumed that switching the data from different vaults across the links of 3D memory is carried out later. Detailed information about the simulator is available in [12].

2) Parameters: We consider a 3D memory of 4GB with v=32, l=4, b=4, r=2048, c=4, e=64, col=16B, t_{layer}=1ns, t_{bank}=4ns, t_{col}=4ns, t_{row}=40ns and a Request window of size 32. Block size is 1024 B ($e \cdot col$). Since the vaults can be accessed in parallel, there is no access overhead for elements in different vaults, i.e., t_{vault}=0ns. The peak vault bandwidth is 16 GB/s.

3) Benchmark: We compare the performance of Baseline and Optimized data layouts using PARSEC 2.0 Benchmarks [25]. These benchmarks model actual workloads and gives a better estimate of performance on real world applications. The performance of different data layouts is compared against a normalized access time of peak performance which is calculated assuming peak bandwidth for each vault and uniform distribution of requests across banks in a vault.

C. Results

Access times of Baseline and Optimized data layouts for different traces are illustrated in Figure 8. We observe that the Optimized data layout improves the access time for *Swaptions* by 56%, *Bodytrack* by 52% and *Blackscholes* by 14%. On the other hand, the improvement is marginal for the remaining benchmarks (*Dedup*, *Ferret*, etc and other benchmarks not listed here). This is due to the fact that they had close to uniform distribution of requests among banks of a vault even in the Baseline data layout as depicted in Figure 9.

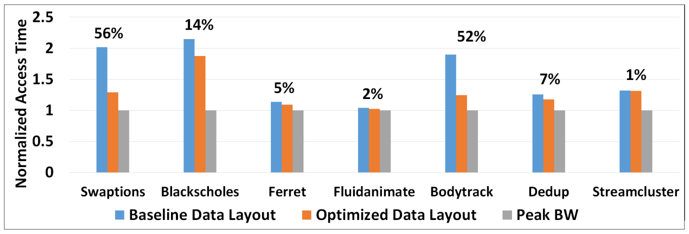


Fig. 8: Access Time Comparison

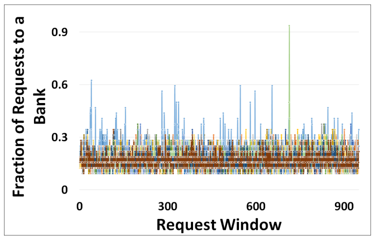


Fig. 9: Request Distribution for all Vaults: *Dedup*

It can be observed that in most of the benchmarks, the access time of the optimized data layout is close to peak performance except for *Blackscholes*. If the memory requests are uniformly distributed among vaults, optimized data layout will guarantee peak bandwidth for each vault and thereby ensuring overall high bandwidth for the 3D memory. But, in the case of Baseline data layout, the restriction is more severe in that the requests in each vault need to distributed uniformly among the banks in addition to the distribution among vaults. We observed that the requests in *Blackscholes* is not uniformly distributed across vaults and this is the reason for optimized data layout not achieving close to peak bandwidth.

VII. Conclusion

In this paper, we have presented an Optimal data layout targeting block accesses in 3D memory. We have highlighted the limitations of existing addressing scheme and proposed a new addressing scheme which can achieve peak bandwidth for random block accesses in a vault. Our data layout exploits inter-layer pipelining to absorb access overhead to different components of 3D memory. Experimental results on variety of workloads from PARSEC 2.0 benchmarks demonstrate up to 56% improvement in memory access time compared with the Baseline data layout.

References

[1] Wm A Wulf and Sally A McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[2] Micron HMC. http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf.

[3] HBM. https://www.jedec.org/sites/default/files/docs/JESD235A.pdf.

[4] Intel Knights Landing. https://www.hpcwire.com/2014/06/24/micron-intel-reveal-memory-slice-knights-landing/.

[5] Trinity. https://nnsa.energy.gov/mediaroom/pressreleases/trinity.

[6] Erfan Azarkhish, Christoph Pfister, Davide Rossi, Igor Loi, and Luca Benini. Logic-base interconnect design for near memory computing in the smart memory cube. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1):210–223, 2017.

[7] Berkin Akin, Franz Franchetti, and James C Hoe. Data reorganization in memory using 3d-stacked dram. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 131–143. ACM, 2015.

[8] Shreyas G. Singapura, Rajgopal Kannan, and Viktor K. Prasanna. On-chip memory efficient data layout for 2d fft on 3d memory integrated fpga. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*, pages 1–7. IEEE, 2016.

[9] Feihui Li, Chrysostomos Nicopoulos, Thomas Richardson, Yuan Xie, Vijaykrishnan Narayanan, and Mahmut Kandemir. Design and Management of 3D Chip Multiprocessors using Network-in-Memory. *ACM SIGARCH Computer Architecture News*, 34(2):130–141, 2006.

[10] Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. *ACM Trans. Archit. Code Optim.*, 12(4):63:1–63:29, January 2016.

[11] Qiuling Zhu, Bilal Akin, H Ekin Sumbul, Fazle Sadi, James C Hoe, Larry Pileggi, and Franz Franchetti. A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing. In *3D Systems Integration Conference (3DIC), 2013 IEEE International*, pages 1–7. IEEE, 2013.

[12] 3D Memory Simulator Code Repository. https://github.com/shreyas-singapura/3D-Memory-Simulator.

[13] Michael A Bender, Jonathan Berry, Simon D Hammond, Branden Moore, Benjamin Moseley, and Cynthia A Phillips. k-means clustering on two-level memory systems. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 197–205. ACM, 2015.

[14] Michael A Bender, Jonathan W Berry, Simon D Hammond, K Scott Hemmert, Samuel McCauley, Branden Moore, Benjamin Moseley, Cynthia A Phillips, David Resnick, and Arun Rodrigues. Two-level main memory co-design: Multi-threaded algorithmic primitives, analysis, and simulation. *Journal of Parallel and Distributed Computing*, 2017.

[15] Jing-ling Yang. Parallel interleavers through optimized memory address remapping. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(6):978–987, 2010.

[16] Pei-Yun Tsai and Chung-Yi Lin. A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing fft processors with rescheduling. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(12):2290–2302, 2011.

[17] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 32–41. ACM, 2000.

[18] Kurt Ferreira, Kevin T Pedretti, Michael Levenhagen, and Ron Brightwell. Exploring memory management strategies in catamount. In *Proceedings of the 2008 Cray Users Group Annual Technical Conference. Helsinki, Finland*, 2008.

[19] Berkin Akin, Franz Franchetti, and James C Hoe. Data reorganization in memory using 3d-stacked dram. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 131–143. ACM, 2015.

[20] Gabriel H Loh. 3d-stacked memory architectures for multi-core processors. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pages 453–464. IEEE, 2008.

[21] Dong Hyuk Woo, Nak Hee Seong, Dean L Lewis, and Hsien-Hsin S Lee. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.

[22] Ke Chen, Sheng Li, Naveen Muralimanohar, Jung Ho Ahn, Jay B Brockman, and Norman P Jouppi. CACTI-3DD: Architecture-Level Modeling for 3D Die-Stacked DRAM Main Memory. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 33–38. EDA Consortium, 2012.

[23] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters*, PP(99):1–1, 2015.

[24] John D Leidel and Yong Chen. HMC-Sim: A Simulation Framework for Hybrid Memory Cube Devices. *Parallel Processing Letters*, 24(04):1442002, 2014.

[25] Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, volume 2011, 2009.