978-1-5090-3525-0/16/$31.00 ©2016 IEEE

On-chip Memory Efficient Data Layout for 2D FFT on 3D Memory Integrated FPGA

Shreyas G. Singapura, Rajgopal Kannan and Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089, USA
{singapur, rajgopak, prasanna}@usc.edu

***Abstract*—3D memories are becoming viable solutions for the memory wall problem and meeting the bandwidth requirements of memory intensive applications. The high bandwidth provided by 3D memories does not translate to a proportional increase in performance for all applications. For an application such as 2D FFT with strided access patterns, the data layout of the memory has a significant impact on the total execution time of the implementation. In this paper, we present a data layout for 2D FFT on 3D memory integrated FPGA that is both on-chip memory efficient as well as throughput-optimal. Our data layout ensures that consecutive accesses to 3D memory are sufficiently interleaved among layers and vaults to absorb latency due to activation overheads for both sequential (Row FFT) and strided (Column FFT) accesses. The current state-of-the-art implementation on 3D memory requires $O(\sqrt{c}N)$ on-chip memory to reduce the strided accesses and achieve maximum bandwidth for an $N \times N$ FFT problem size and c columns in a 3D memory bank row. Our proposed data layout optimizes the throughput of both the Row FFT and Column FFT phases of 2D FFT with $O(N)$ on-chip memory for the same problem size and memory parameters without decreasing the memory bandwidth thereby achieving a $\sqrt{c}\times$ reduction in on-chip memory. On architectures with limited on-chip memory, our data layout achieves $2\times$ to $4\times$ improvement in execution time compared with the state-of-art 2D FFT implementation on 3D memory.**

I. Introduction

3D memory is a potential solution to the memory wall problem of low bandwidth with a promising bandwidth in the range of 300-400 GB/s [1], [2]. Although 3D memories provide higher bandwidth than existing 2D memory technologies such as DDR3 [3], the actual bandwidth available depends on the access pattern of the application and thus may be much lower than the peak bandwidth. In particular, the traditional row activation overhead can become a bottleneck due to random accesses to multiple rows. To maximize the available bandwidth from the 3D memory, optimizations specific to the access pattern of the application need to be developed.

Fast Fourier Transform (FFT) is an important kernel used in signal processing applications [4], [5]. Specifically, 2D FFT is widely used in image processing applications and requires high throughput for large image sizes [6], [7]. 2D FFT is implemented as a combination of Row and Column FFT phases [8]. The Row FFT phase consists of sequential accesses to the memory which result in high bandwidth and low latency. On the other hand, the Column FFT phase has strided accesses which translate to accesses to different rows of a bank. These non-sequential accesses to the memory result in low bandwidth and high latency due to the activation overhead of accessing multiple rows.

There have been works [9], [10] targeting optimizations to the data layouts for 2D FFT implementation on 3D memory. The previous works focus on achieving high bandwidth by storing multiple rows/columns of input data in on-chip memory and reducing number of strided accesses to the memory. Although high bandwidth is achieved by the proposed data layout, large number of complete rows/columns need to be stored in on-chip memory to achieve this high bandwidth. As the problem size increases, the amount of on-chip memory has to be increased proportionally to store the entire rows/columns and maintain high bandwidth. With limited on-chip memory, the data layouts of existing works is not able to extract maximum bandwidth from the 3D memory which results in higher execution time.

We observe that the structure of 3D memory can be exploited to hide the latency overhead of accessing multiple rows resulting from strided accesses. By employing the inter-layer pipelining and parallel vault access features of 3D memory, we develop an optimized data layout that provides maximum bandwidth for both sequential and strided accesses and requires only the necessary elements to be stored in on-chip memory. Our throughput-optimal data layout for 2D FFT on 3D memory has the following key features: (1) Consecutive elements are mapped to different layers to exploit faster access across 3^{rd} dimension than mapping elements to the same layer; (2) Vaults are accessed in parallel since the latency of multiple vault access is the same as that to a single vault and (3) Accesses to the same layer are separated by sufficient number of intermediate accesses to other layers to hide the access latency. The main contributions of this paper are:

- We present an optimized data layout for 2D FFT on 3D memory that achieves maximum bandwidth for both sequential and strided access patterns of 2D FFT.
- Our data layout achieves an on-chip memory requirement of $O(N)$ for a 2D FFT problem size of $N \times N$ without sacrificing the bandwidth and latency of 3D memory.
- Our data layout achieves $\sqrt{c}\times$ reduction in on-chip memory compared with state-of-the-art 2D FFT implementation given c columns in a row of a memory bank.
- On architectures with limited on-chip memory, our data

This material is based in part upon work supported by the Air Force Research Laboratory under Grant Number FA8750-15-1-0185.

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

layout achieves 2× to 4× improvement in execution time compared with state-of-the-art 2D FFT implementation for large problem sizes.

The rest of the paper is organized as follows. Section II covers the related work for 2D FFT on 3D memories. Section III describes the target architecture and its components. Section IV describes the Baseline data layout and introduces the proposed Optimized data layout. Section V presents the evaluation methodology and performance analysis. Section VI concludes the paper.

II. Background and Related Work

2D FFT can be implemented in two phases of Row FFT and Column FFT using the row-column algorithm by performing 1D FFTs on rows and columns of inputs [8]. In the Row FFT phase of the algorithm, for a problem size of $N \times N$ input matrix, 1D FFT is applied on each row of the input matrix in sequential order. The outputs of the Row FFT phase act as inputs to the Column FFT phase. In the Column FFT phase, 1D FFT is applied on each column of the $N \times N$ matrix and the outputs of Column FFT phase represent the final output of 2D FFT on the original $N \times N$ input matrix.

2D FFT on 3D memory has been the focus of many research works. In [9], memory optimized data layouts is developed for FFT on hardware accelerators such as ASIC and FPGA. Block data layout is implemented for DDR3 memory and later extended to 3D memory. A block is mapped to a row of a bank and multiple blocks are distributed among banks to increase the bandwidth of the memory. For a block of size $t \times t$ and a problem size $N \times N$, the on-chip memory requirement is of the order $O(tN)$. In [11], a Logic-in-Memory (LiM) IC is developed to perform 2D FFT on 3D memory. Application specific logic cores are used to implement 2D FFT and energy efficiency and bandwidth are targeted as the performance metrics. Although inter-layer pipelining is utilized, block data layout from [9] is used. In [10], processing kernel on FPGA is developed to implement dynamic data layouts to reduce the number of row activations. Multiple rows/columns (p) of input data are prefetched from the memory and permutation network is used while writing back the outputs to memory to reduce the number of row activations. The on-chip memory requirement is of the order $O(pN)$, for $1 < p < t$. None of these works focus on the on-chip memory and require substantial amount of on-chip memory to achieve high bandwidth for large problem sizes.

In this paper, we propose an Optimized data layout to implement 2D FFT on 3D memory which achieves a minimum on-chip memory requirement without sacrificing the bandwidth and latency of 3D memory. We exploit inter-layer pipelining and parallel vault access to hide the latency of accessing elements in the same layer and overhead of accessing multiple rows. By achieving maximum bandwidth for both Row and Column FFT phases, our data layout stores only the necessary elements in on-chip memory and minimizes the on-chip memory requirement.

III. Target Architecture

Our target architecture is a 3D memory integrated FPGA consisting of 3D memory and an FFT Processing Unit (PU) on FPGA. The components of the architecture are illustrated in Figure 1.

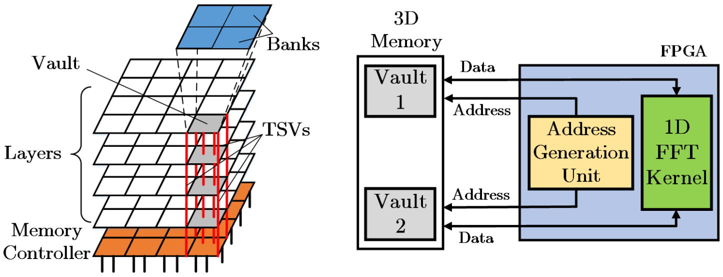


Fig. 1: (a) Architecture of a 3D Memory (b) FFT PU on FPGA

A. 3D Memory

3D memory is organized as a set of v vaults consisting of l layers and b banks per layer in a vault. Data in a vault is accessed using vertical interconnects (TSVs). A representative archtiecture of 3D memory consisting of 16 vaults with 4 layers and 4 banks per layer is illustrated in Figure 1(a). Vaults do not share TSVs with one another and hence can be accessed in parallel. Within a vault, data in different layers can be accessed at a faster rate than data in the same layer, a property known as *inter-layer pipelining* [12], [11], [13]. This is because the latency of activation overhead of rows in different layers can be overlapped due to fast TSVs. Within a layer, the structure of 3D memory is similar to the structure of DDR3 with data stored in rows and columns in each bank. Accessing data stored in different rows of the same bank incurs large latency due to row activation overhead whereas, bank interleaving can be used to reduce the latency by accessing data in different banks. Each data element stored in a 3D memory can be accessed by specifying the address in terms of vault, layer, bank, row and column. For each read/write request to the 3D memory, a specific row in a bank belonging to a layer in a vault is accessed and the bandwidth and latency of 3D memory depends on the access pattern of these requests.

In our previous work [10], [14], [15], we developed a parameterized model of the 3D memory to identify the parameters which have a significant impact on the bandwidth and latency of 3D memory. Our model characterizes the 3D memory in terms of timing parameters which take into account the architecture and different access patterns. For the sake of completeness, we describe once again the parameters of the 3D memory model:

- 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

It should be noted that all the above parameters except t_{row} are defined assuming the rows being accessed are already active.

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

B. FFT Processing Unit (PU) on FPGA

The FFT processing unit consists of a 1D FFT kernel and an address generation unit. 1D FFT kernel processes inputs of size N to produce FFT outputs. The address generation unit maps the inputs and outputs of the 1D FFT kernel to the required addresses in the memory. Since the kernel can process streaming data, we use different vaults to read inputs and write the outputs. In Figure 1, in the Row FFT phase, Vault 1 acts as the input vault and Vault 2 acts as the output vault. In the Column FFT phase, their roles are reversed.

IV. Data Layouts

In this section, we describe the Baseline data layout and its limitations. Later, we present our proposed Optimized data layout along with the mapping technique. The parameters of the architecture used in our analysis and their definitions are described in Table I. We assume each access to a vault results in a column of data being available from the memory. For notation convenience, we assume there are $2v$ vaults in the memory; v vaults are used to read inputs and v vaults are used to write the outputs. This assumption does not affect our proposed data layout or the performance analysis.

Notation	Definition
$N \times N$	problem size
$2v$	# vaults in 3D memory
l	# layers in a vault
b	# banks per layer in a vault
r	# rows in a bank
c	# columns in a row of a bank

TABLE I: Parameters of 3D Memory

A. Baseline Data Layout

We use the block data layout proposed in [9] as the Baseline data layout. In this data layout, a block or a tile of size $t \times t$ is mapped to a row of a bank in the memory and multiple such blocks are mapped to different banks. The value of t ranges between $[1, \sqrt{c}]$. In order to perform a 1D FFT of a row of N elements, an entire row of blocks (equivalent to t rows of $N \times N$) are transferred from the 3D memory to on-chip memory. An FFT kernel is used to process the data stored in on-chip memory and the outputs are written back to memory. This process is repeated for all the rows in the input matrix to complete the Row FFT phase. In the subsequent Column FFT phase, entire column of blocks are transferred to the on-chip memory and processed to produce the final Column FFT outputs. Although the Baseline data layout can enable high throughput, we observe the following limitations of this data layout.

Limitation 1: Bandwidth of the 3D memory is proportional to the block size with $t^2 = c$ achieving maximum bandwidth.

For $t^2 = c$, blocks are accessed from different layers and the latency overhead of accesses to the same bank is overlapped with accesses to banks in other layers and the bandwidth is limited by t_{layer}. For $t^2 < c$, majority of the consecutive blocks are mapped to banks in the same layer and t_{bank} and t_{col} will limit the bandwidth of the 3D memory. The effect of small block sizes on performance is evident in [9], with $t = (4, 8)$ achieving $(33\%, 50\%)$ of the performance in comparison with that of $t = 32$.

Limitation 2: For an $N \times N$ problem size, $O(\sqrt{c}N)$ on-chip memory is required to achieve maximum bandwidth.

At any point of time, an entire row/column of blocks of data (tN elements) need to be stored in on-chip memory to process N elements of a row/column. Based on Limitation 1, maximum bandwidth is achieved for $t^2 = c$. Therefore, the on-chip memory required is $\sqrt{c}N$ elements of data. In [9], the authors use block size $t = 32$ to achieve maximum bandwidth. For problem sizes $N = [8192, 32768]$ complex single-precision (2×32 bits per word) inputs, this translates to a large on-chip memory requirement in the range of $16 - 67$ Mbits.

Therefore, the Baseline data layout requires large on-chip memory to achieve maximum bandwidth from 3D memory and on limited on-chip memory architectures, bandwidth of 3D memory reduces which translates to higher execution time.

B. Optimized Data Layout

Our data layout is defined by two mapping functions, corresponding to each phase of 2D FFT. Each function is a mapping of an $N \times N$ matrix to locations in 3D memory. A location (address) is defined by the quintuple $\{v(a_{ij}), l(a_{ij}), b(a_{ij}), c(a_{ij}), r(a_{ij})\}$ which maps matrix element a_{ij} to a vault, layer, bank, column and row in the 3D memory. The first mapping function (DL 1) describes the layout of FFT input matrix A in 3D memory before the start of the Row FFT phase. The second mapping function (DL 2) is used to write the elements of matrix $\widehat{A}$, the output of the Row phase, to 3D memory. The same layout is then used to read columns of $\widehat{A}$ during the Column FFT phase. The outputs of this phase are the final outputs and can follow either data layout above, depending on how the resultant matrix is to be used further.

In order to derive our mapping scheme for optimal on-chip storage and bandwidth maximizing 2D FFT data layout, we make the following basic assumption about the timing parameters of 3D memory: $t_{layer} \leq \{t_{bank}, t_{col}\} \leq t_{row}$. We also assume the number of layers is sufficient to make $l \cdot t_{layer} \geq \{t_{col}, t_{bank}\}$ (we also describe the performance results in Section V-B when this assumption is relaxed). These assumptions are based on our estimates of the timing and architecture parameters of 3D memory, as described in [1]. The 3D memory in [1] has a peak bandwidth of 8 GB/s per vault and an element of 64 bits can be accessed for each memory request, which translates to an access time of 1 ns for each element. Therefore, we assume $t_{layer} = 1\ ns$ which represents the least possible latency of memory accesses. Further, since the structure of a layer in a 3D memory is similar to DDR3 [3], we estimate the values of other timing parameters as $t_{bank} = 2\ ns$, $t_{col} = 4\ ns$ and $t_{row} = 40\ ns$ based on timing parameters described in [3].

The key characteristics of our mapping scheme based on the above assumptions are as follows: Since vaults can be

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

accessed in parallel, it is trivial to distribute elements across vaults to maximize bandwidth. Our data layout further maps accesses within a vault to different layers to ensure the minimum possible latency of t_{layer} for each access. Now, considering accesses within a vault, our layout maximizes bandwidth by hiding the latency of consecutive accesses to the same row or different rows in a bank through a number of intermediate accesses to other layers, utilizing $p \cdot t_{layer} \geq t_{col}$ and $q \cdot t_{layer} \geq t_{row}$. For example, choosing $p \geq 4$ and $q \geq 40$ based on the parameters above, will hide the latency of t_{col} and t_{row} and incur a minimum latency of t_{layer}. Hiding the latency of accesses to different rows and columns is possible due to the large number of banks [1] and faster access across the 3^{rd} dimension of 3D memory [11], [13]. We prove that our data layout minimizes the latency in Section V-B.

For notational simplicity and WLOG, in our description of the mapping schemes, we assume that parameters N, v, l, b, r and c are powers of 2 and that $k = \sqrt{vlbc}$ is an integer (power of 2). These assumptions can be relaxed at the cost of increased notational complexity in the description of our layout scheme. We also assume $N \leq \sqrt{vlbrc}$ (to ensure the problem fits in memory).

Data Layout 1 (DL 1): Our first mapping scheme for the Row FFT phase is a straightforward round-robin mapping of the rows of A over vaults, layers, banks, columns and rows. Each row of N input elements from A is distributed in a round-robin fashion across v vaults (line 3). Similarly, in a round-robin fashion, the N/v elements within a vault are distributed among l layers in that vault and the $N/(vl)$ elements within a layer distributed among its b banks (line 4). Finally, the $N/(vlb)$ elements assigned to a bank are distributed in row major order among its c columns and r rows (line 5). This mapping function is repeated for all the N rows of the input matrix.

DL 1: Mapping Function for Matrix A (Row FFT Inputs)

```
1 $a_{ij} : (i,j)^{th}$ element of $A$, $0 \leq i,j \leq N-1$
2 Address$[a_{ij}] \rightarrow \{v(a_{ij}), l(a_{ij}), b(a_{ij}), c(a_{ij}), r(a_{ij})\}$
3 $v(a_{ij}) = (i \cdot N + j) \bmod v$
4 $l(a_{ij}) = \left\lfloor \frac{(i \cdot N+j)}{v} \right\rfloor \bmod l$ ;    $b(a_{ij}) = \left\lfloor \frac{(i \cdot N+j)}{vl} \right\rfloor \bmod b$
5 $c(a_{ij}) = \left\lfloor \frac{(i \cdot N+j)}{vlb} \right\rfloor \bmod c$;    $r(a_{ij}) = \left\lfloor \frac{(i \cdot N+j)}{vlbc} \right\rfloor \bmod r$
```

Data Layout 2 (DL 2): Our second mapping scheme ensures that consecutive accesses to 3D memory components (vaults, layers etc.) are sufficiently spaced to absorb respective component activation overheads both during the row major write phase at the end of the Row FFT as well as during the column major read phase at the start of Column FFT. Consider the same row index across all banks, layers and vaults of 3D memory. Given c columns per row, there are $vlbc$ locations corresponding to this row index across the entire 3D memory. We want to repeatedly distribute elements from the rows and columns of the $N \times N$ output matrix $\widehat{A}$ of the Row FFT phase uniformly among these $vlbc$ locations for each row index. Note that $\widehat{A}$ is only available one row at a time and the writing to memory occurs as per our mapping function after each row of $\hat{A}$ becomes available. We start by dividing $\widehat{A}$ into contiguous $k \times k$ blocks, with $k = \sqrt{vlbc}$. It should be noted that although we divide the matrix into blocks, our blocks as well as our mapping function (as described below) are quite different from the Baseline data layout [9] which maps $\sqrt{c} \times \sqrt{c}$ blocks to a single bank row. Define the following parameter $x = \{\min_{1 \leq s \leq c}(s) \mid sl(b-2)t_{layer} \geq t_{row}\}$. Let $y = 2^{\lceil \log_2 x \rceil}$, i.e., x rounded to the nearest power of 2. In our layout, y represents the number of consecutive accesses to the same row in a bank of a layer in a vault before the next bank in that same layer and vault is accessed. DL 2 describes in detail each of the mapping functions.

DL 2: Mapping Function for matrix $\widehat{A}$ (Row FFT Outputs)

```
1 $a_{ij} : (i,j)^{th}$ element of $\widehat{A}$, $0 \leq i,j \leq N-1$
2 Address$[a_{ij}] \rightarrow \{v(a_{ij}), l(a_{ij}), b(a_{ij}), c(a_{ij}), r(a_{ij})\}$
3 $\{k, y\}$ // block related parameters
4 $v(a_{ij}) = (i + j) \bmod v$
5 $l(a_{ij}) = (\lfloor \frac{i}{v} \rfloor + \lfloor \frac{j}{v} \rfloor) \bmod l$
6 $b(a_{ij}) = (\left\lfloor \frac{i}{vly} \right\rfloor + \left\lfloor \frac{j}{vly} \right\rfloor) \bmod b$
7 $r(a_{ij}) = \frac{N}{k} \lfloor \frac{i}{k} \rfloor + \lfloor \frac{j}{k} \rfloor$
```

The key properties of the data layout that enable us to maximize bandwidth while limiting on-chip storage are described below in the form of lemmas. We will utilize these lemmas in our performance analysis in Section V-B.

We first show that when $N \times N$ matrix $\widehat{A}$ is divided into contiguous $k \times k$ blocks, every element within a block is mapped to the same row index across all banks, layers and vaults of 3D memory. Furthermore, distinct blocks are mapped to distinct rows across memory.

Lemma 1. *$r(a_{ij}) = r(a_{mn})$ if and only if $\lfloor i/k \rfloor = \lfloor m/k \rfloor$ and $\lfloor j/k \rfloor = \lfloor n/k \rfloor$. Each row of $k \times k$ blocks from $\widehat{A}$ is laid out over N/k successive row indices in memory. The row indices of successive blocks in a column of blocks increases by N/k. The total number of rows used in DL 2 is N^2/k^2.*

Proof. This follows from Line 7 of DL 2.

Lemma 2. *Under DL 2, the two closest elements in a row or column of $\widehat{A}$ that are mapped to the same vault are mapped to successive layers (modulo l).*

Proof. Using Lines 4 and 5 of DL 2, if $v(a_{ip}) = v(a_{iq})$ or $v(a_{pj}) = v(a_{qj})$, then, (1) $q \equiv p \pmod{v}$ and (2) if $|q-p| = m \cdot v$, then $l(a_{iq}) = (l(a_{ip}) + m) \bmod l$. Thus, every layer in a vault is accessed in round robin fashion both for writing rows and reading columns of of $\widehat{A}$. Elements in the same row or column of $\widehat{A}$ that are mapped to the same vault *and* layer are exactly "vl" apart.

From Lemma 2, note that $k/(vl)$ represents the number of elements from a $k \times k$ block mapped to a single layer within a vault. In this paper, we assume that $k/(vl) \geq y$ in order to achieve minimum latency. This assumption is validated given the parameters from [1] as described earlier.

Lemma 3. *Under DL 2, for each layer d ($0 \leq d \leq l-1$), of every vault e ($0 \leq e \leq v-1$), $q = k/(vly)$ successive banks are accessed (modulo b), when writing each row and reading each column of a $k \times k$ block. For each such bank, the same row p, $0 \leq p \leq (N^2/k^2)-1$ is accessed y consecutive times. For successive $k \times k$ blocks in the same row (column, resp.) of blocks, within the same layer d and vault v, the next set of q banks (modulo b) are accessed in a similar manner, but for the next row $p+1$ ($p+(N/k)$ respectively).*

Proof. First, applying Lemma 2 we note that the same layer d of every vault e is visited (mapped) every vl elements of a row or column of a block. Consider traversing a row i of the block. Looking at increasing values of j from Line 6 of DL 2, we get that the same block is accessed y consecutive times before the next block (modulo b) is visited, again y consecutive times. When moving to the next $k \times k$ block in the row of blocks, j has increased by $k = qlvy$ from the start of the current block and so, we start with the q^{th} succeeding bank (modulo b). A similar logic applies when considering traversing down a column j of the block. Bank indices increase by 1 with every vly increase in i and by q after the block is traversed. The last line of the claim comes from applying Lemma 1. All y accesses to these $q = k/(vly)$ blocks are to the same row.

Due to space considerations, we have not described the exact column mapping in DL 2. Clearly, if there are m mappings to a bank in a given layer and vault from a block, since all m accesses are to the same row, m columns of the row are mapped. Once this row is filled, the column index is reset and the row index will have changed.

V. Evaluation Methodology and Analysis

A. Performance Metric

In this paper, we compare the performance of Baseline and Optimized data layouts using the following metrics:

- **Total Execution Time**: measured as the amount of time to perform 2D FFT including memory access time and computation time.
- **On-chip memory consumption**: measured in terms of number of elements of input data stored in on-chip memory to perform 2D FFT.

B. Performance Analysis

The FFT kernel can process streaming data and therefore, the time to write the output of FFT of a row/column of inputs can be overlapped with the time to read next row/column of inputs. The total execution time of 2D FFT can be divided into components represented in Equation 1. The *Initial Computational Latency (ICL)* is a constant and relatively small compared to other components of the equation and will be ignored in the analysis of total execution time.

$$t_{total} = \textit{Row FFT}_{write} + \textit{Column FFT}_{write} + 2 \times \textit{ICL} \qquad (1)$$

In the Baseline data layout [9], for a block size of $t^2 = c$, interleaving between banks on different layers results in $c \cdot l$ elements being available in a time span of $c \cdot t_{col}$. The entire row/column of blocks, i.e., $(N/\sqrt{c})$ blocks need to be stored in on-chip memory. On the other hand, for $t^2 < c$, consecutive accesses are mapped to the same bank and only c elements are available in the same time span. Therefore, the Baseline data layout requires $O(c \cdot N/\sqrt{c})$, i.e., $O(\sqrt{c}N)$ on-chip memory to achieve maximum bandwidth. This access pattern is repeated $N/\sqrt{c}$ times to complete the Row FFT phase. The Column FFT phase follows similar access pattern. [9] does not discuss how to achieve maximum bandwidth when accessing partial blocks from 3D memory. Based on this analysis, the range of execution time of Row/Column FFT phase is,

$$\begin{aligned} \textit{Row/Column FFT}_{read/write} &= (\frac{N}{\sqrt{c}})(\frac{N}{\sqrt{c}})[\frac{c}{vl}t_{col}, \frac{c}{v}t_{col}] \\ &= [\frac{N^2}{vl}t_{col}, \frac{N^2}{v}t_{col}] \end{aligned} \qquad (2)$$

Now, we analyze the performance of Optimized data layout.

Theorem 1. *During the read phase of Row FFT, using DL 1, elements of a row of A can be accessed with a latency of t_{layer} and v such elements are available simultaneously.*

Proof. Follows from the round-robin mapping used in DL 1.

In the following, we focus on DL 2 for $\widehat{A}$ used in write phase of Row FFT and read phase of Column FFT.

Theorem 2. *The latency of two consecutive accesses to the same vault is t_{layer}. The latency of two consecutive accesses to banks in the same vault and layer is $l \cdot t_{layer}$.*

Proof. From Lemma 2, consecutive accesses to a vault are to successive layers modulo l. Since the latency of accessing different layers in the same vault is t_{layer}, the result follows.

Theorem 3. *DL 2 ensures that the latency of accesses to the same row of a bank and to different banks in the same layer is hidden by accesses to other layers.*

Proof. From Lemma 3 and Theorem 2, the latency of accessing the same row of a bank or different banks in the same layer is $l \cdot t_{layer}$. This number is $\geq \{t_{col}, t_{bank}\}$ and therefore, these latencies are hidden.

Theorem 4. *DL 2 ensures that the latency of accesses to different rows of a bank is hidden by accesses to other layers.*

Proof. From Lemma 3, the earliest possible access to a different row of the same bank when traversing a row or column of $\widehat{A}$ is after $(b-2)y$ accesses to banks in the same layer. From Theorem 2, each such access has latency $l \cdot t_{layer}$. Therefore, the latency of accesses to different rows of the same bank is $\geq yl(b-2)t_{layer}$. By definition of y, this value is $\geq t_{row}$, thereby effectively hiding the latency of access to different rows in the same bank .

Based on Theorem 3, our mapping function has ensured elements in each set of k elements are mapped to different layers and can be accessed with a latency of t_{layer}. This implies that consecutive elements in a row of $\widehat{A}$ can be written to the memory with a latency of t_{layer}. Based on Theorem 4 and Lemma 3, the access latency for elements across different sets in a block of $k \times k$ is also t_{layer}. Therefore, consecutive elements in the same row/column of a block of $k \times k$ can be accessed with a latency of t_{layer}. Based on definition of k, a single row across banks in all layers and vaults consists of k elements from k different rows of $\widehat{A}$. Another way to look at a block of $k \times k$ is that it consists of k elements from k different columns of $\widehat{A}$. Therefore, our data layout ensures consecutive elements of a column of $\widehat{A}$ can be read from the memory during the Column FFT phase with a latency of t_{layer}. Hence, our data layout ensures minimum latency of t_{layer} during both Row FFT and Column FFT phase. By doing so, only a single row or column of $\widehat{A}$, i.e., $O(N)$ elements is stored in on-chip memory. The execution time of Row/Column FFT is,

$$\textit{Row/Column FFT}_{\textit{read/write}} = N \cdot N \cdot (t_{layer}/v) \qquad (3)$$

If we relax our assumption that $l \cdot t_{layer} \geq t_{col}$, l elements are available in a time span of $\max[l \cdot t_{layer}, t_{col}]$ and the total execution time is bounded by $(\frac{2N^2}{vl} \cdot t_{col})$. Based on this analysis, the range of execution time and the on-chip memory of Baseline and Optimized data layouts is listed in Table II.

Data Layout	Range of Execution Time (t_{total})	On-chip Memory for Max. Bandwidth
Baseline	$[\frac{2N^2}{vl} \cdot t_{col}, \frac{2N^2}{v} \cdot t_{col}]$	$O(\sqrt{c}N)$
Optimized	$[\frac{2N^2}{v} \cdot t_{layer}, \frac{2N^2}{vl} \cdot t_{col}]$	$O(N)$

TABLE II: Performance Comparison of Data Layouts

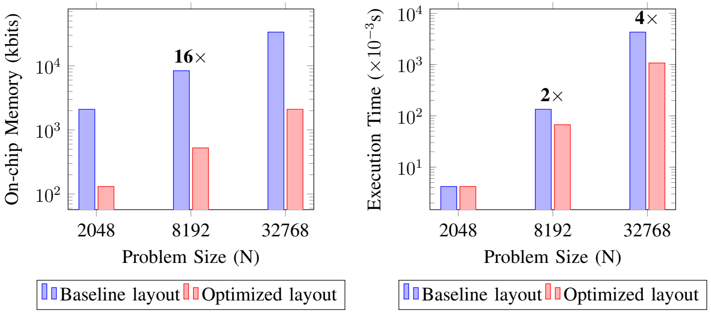
C. Performance Evaluation

3D memories have become popular recently, and since the exact internal architecture is proprietary, existing cycle accurate simulators do not capture all the features of the 3D memory. For example, [16], [17] do not provide the feature of inter-layer pipelining and are limited to specific types of 3D memory. HMCSim [18] does not reveal the internal architecture details due to Intellectual property rights. Therefore, in this paper, we use the 3D memory model described in Section III-A and analyze the performance of different data layouts. We do not claim cycle accurate performance comparison as we are looking for higher order performance estimate of 2D FFT on 3D memory.

Parameter	v	l	b	r	c	Vault Bandwidth
Values	4	4	4	4096	256	8 GB/s

TABLE III: 3D Memory Parameter Values

For the performance analysis, timing parameters of 3D memory are estimated as: $t_{layer} = 1\ ns$, $t_{bank} = 2\ ns$, $t_{col} = 4\ ns$ and $t_{row} = 40\ ns$ [1], [3] (Section IV-B). We assume vaults can be accessed in parallel making v elements available from v vaults in a time equal to the latency of accessing one element from 1 vault, i.e., $t_{vault} = 0\ ns$. We assume the inputs are complex single-precision floating point numbers (2 × 32 bits per word) and each access to a vault ensures 1 column/element of data, i.e., 64 bits are available to the FPGA. The parameters of 3D memory are tabulated in Table III. We assume a streaming FFT Processing Unit on FPGA with 128 Gbits/s (16 GB/s) throughput [19]. For a vault with a bandwidth of 8 GB/s [1], 2 vaults saturate the throughput of the FFT processing unit. Therefore, 2 vaults are used to read inputs and 2 vaults are used to store the outputs.



(a) On-chip Memory for Maximum Bandwidth

(b) Execution Time on Limited On-chip Memory

Fig. 2: Performance Comparison of Data Layouts

In Figure 2(a), we analyze the amount of on-chip memory required to achieve maximum bandwidth for Baseline and Optimized data layouts. The Baseline data layout uses a block size of $t = 16$ and on-chip memory of 33 Mbits to achieve maximum bandwidth. We observe that the Optimized data layout achieves maximum bandwidth with substantially lower on-chip memory (16×). In Figure 2(b), we assume the architecture has a limited on-chip memory of 4 Mbits. For the Baseline data layout, the available on-chip memory is sufficient to achieve maximum bandwidth for small problem sizes ($N = 2048$). For large problem sizes ($N = 8192, 32768$), due to small amount of on-chip memory, Baseline data layout is restricted to small block sizes and majority of the consecutive accesses are mapped to the same layer and the bandwidth is limited by t_{bank} or t_{col}. This translates to a higher execution time in comparison with the Optimized data layout. On the other hand, the Optimized data layout does not suffer any degradation in performance since the available on-chip memory is sufficient to store the required $O(N)$ elements of input data and ensures maximum bandwidth is achieved resulting in $[\frac{t_{bank}}{t_{layer}}$ to $\frac{t_{col}}{t_{layer}}]$, i.e., 2× to 4× reduction in execution time.

VI. Conclusion

We have presented an on-chip memory efficient data layout to implement 2D FFT on 3D memory. Our data layout exploits inter-layer pipelining and parallel vault access to hide the latency overhead of strided accesses. The data layout ensures maximum bandwidth is available from 3D memory with the on-chip memory requirement of $O(N)$ for a problem size of $N \times N$. In comparison with the Baseline data layout, our Optimized data layout reduces the on-chip memory by $\sqrt{c}\times$ for c columns in a row of a memory bank. With limited on-chip memory, our data layout achieves 2× to 4× reduction in execution time compared with the Baseline data layout.

References

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

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

[3] Micron DDR3 Datasheet. https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/2gb_ddr3_sdram.pdf.

[4] James W Cooley and John W Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of computation*, 19(90):297–301, 1965.

[5] Lawrence R Rabiner and Bernard Gold. Theory and Application of Digital Signal Processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.*, 1, 1975.

[6] Stefan Langemeyer, Peter Pirsch, and Holger Blume. Using SDRAMs for Two-Dimensional Accesses of Long $2^n \times 2^m$-point FFTs and Transposing. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 242–248. IEEE, 2011.

[7] Chi-Li Yu, Jung-Sub Kim, Lanping Deng, Srinidhi Kestur, Vijaykrishnan Narayanan, and Chaitali Chakrabarti. FPGA Architecture for 2D Discrete Fourier Transform based on 2D Decomposition for Large-sized Data. *Journal of Signal Processing Systems*, 64(1):109–122, 2011.

[8] Ren Chen and Viktor K Prasanna. Energy Optimizations for FPGA-based 2D FFT Architecture. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pages 1–6. IEEE, 2014.

[9] Berkin Akin, Franz Franchetti, and James C Hoe. Understanding the Design Space of Dram-Optimized Hardware FFT Accelerators. In *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, pages 248–255. IEEE, 2014.

[10] Ren Chen, Shreyas G Singapura, and Viktor K Prasanna. Optimal Dynamic Data Layouts for 2D FFT on 3D Memory Integrated FPGA. In *Parallel Computing Technologies*, pages 338–348. Springer, 2015.

[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] 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.

[13] 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.

[14] Shreyas G Singapura, Anand Panangadan, and Viktor K Prasanna. Towards Performance Modeling of 3D Memory Integrated FPGA Architectures. In *Applied Reconfigurable Computing*, pages 443–450. Springer, 2015.

[15] Shreyas G Singapura, Anand Panangadan, and Viktor K Prasanna. Performance Modeling of Matrix Multiplication on 3D Memory Integrated FPGA. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 154–162. IEEE, 2015.

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

[17] 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.

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

[19] Synopsys Parallel FFT Core. https://www.synopsys.com/company/publications/synopsysinsight/pages/art1-fpga-signal-processing-issq1-13.aspx.