

BANDWIDTH-INTENSIVE FPGA ARCHITECTURE FOR MULTI-DIMENSIONAL DFT[†]

Chi-Li Yu¹, Chaitali Chakrabarti¹, Sungho Park² and Vijaykrishnan Narayanan²

¹: School of Electrical, Computer and Energy Engineering, Arizona State University, USA

²: Dept. of Computer Science and Engineering, Pennsylvania State University, USA
email: {chi-li.yu, chaitali}@asu.edu, {szp142, vijay}@cse.psu.edu

ABSTRACT

Multi-dimensional (MD) Discrete Fourier Transform (DFT) is a key kernel algorithm in many signal processing algorithms, including radar data processing and medical imaging. Although there are many efficient software solutions, they are not suitable for applications that require fast response time. In this paper we focus on FPGA-based implementation of MD DFT. The proposed architecture is based on a decomposition algorithm that takes into account FPGA resources and the characteristics of off-chip memory access, namely, the burst access pattern of the Synchronous Dynamic RAM (SDRAM). The architecture can support 2D, 3D, and even higher dimensional DFT with high performance. It has been implemented on a Xilinx Virtex-5 FPGA platform and its performance for 2D and 3D DFT measured and analyzed.

Index Terms— Multidimensional signal processing, DFT, FPGA, DRAM.

1. INTRODUCTION

Discrete Fourier Transform (DFT) is widely adopted in digital signal processing (DSP) and scientific computing. Multi-dimensional (MD) DFT is used in a variety of imaging applications which need frequency-domain analysis. The image sizes in many applications have become larger over the years. In synthetic aperture radar processing and medical imaging, for instance, the required data size could be as large as 2048×2048 . Therefore, there is a need for new algorithms and architectures to support MD DFT of large data sizes.

Existing MD DFT implementations include software solutions, which can run on general-purpose processors or cluster computers. Though these platforms can achieve high performance, they are power-consuming and not suitable for embedded applications. There are also many hardware solutions. These include the dedicated FFT chips [1] [2], and field programmable gate array (FPGA) based implementations [3] [4]. We concentrate on FPGA-based architectures since they can be reconfigured according to user-specified design parameters and offer flexibility while maintaining high performance.

Most of the existing 2D DFT solutions have been primarily based on Row-Column (RC) decomposition, where the 2D DFT is computed by successively applying 1D DFT along rows and then along columns. However, accessing data along a column requires large stride accesses, which dramatically degrades the performance on Synchronous Dynamic RAM (SDRAM). These memories only favor burst access, and data stored in consecutive locations in memory can be retrieved very efficiently. To avoid memory access with large strides, most previous works adopt transpose operations to re-align the column data into contiguous addresses. While this enables a long burst size to be maintained, the transpose operations takes additional time.

In this paper, we propose a new DFT architecture which avoids both large stride memory access and transpose operations. This is done by loading a strip of data from the SDRAM onto the local memory, where the width of the strip matches the burst size. If the local memory is not large enough to hold whole columns, smaller sized subcolumns are loaded. At the algorithm level, this translates to computing 1D DFT along columns to an equivalent 2D DFT. Furthermore, we design a conflict-free data organization in the local memory. When there are enough hardware resources to support multiple processing units (PE), this enables the PEs to access the local memory simultaneously. The architecture has been ported onto the BEE3 FPGA board [5], and its 2D DFT and 3D DFT functions have been verified for different sized images. Simulation results based on Xilinx ML605 board show that we can process $2K \times 2K$ images with complex 64-bit precision in less than 27ms.

The rest of the paper is organized as follows. In Section 2, the proposed MD DFT algorithm is derived. Section 3 describes in detail the proposed DFT architecture. The architecture has been implemented and evaluated in Section 4, and concluding remarks are given in Section 5.

2. PROPOSED MD DFT ALGORITHM

We first describe the memory-aware 2D DFT, and then show how this scheme can be extended for 3D DFT.

[†] This work was supported by DARPA-DESA W911NF-05-1-0248.

2.1. 2D DFT Algorithm

The general form of 2D DFT can be described in matrix form as follows. Here input U and output V_2 are of size $N_2 \times N_1$; F_{N_1} and F_{N_2} are twiddle factor matrices for row and column DFT computations, respectively:

$$V_2 = F_{N_2} \cdot U \cdot F_{N_1}. \quad (1)$$

In Eq. (1), $V_1 = U \cdot F_{N_1}$ can be done by applying 1D FFT along the rows of U , and $(F_{N_2} \cdot V_1)$ by applying 1D FFT along the columns. This is the traditional Row-Column (RC) decomposition. If the data along the rows of U are stored in a one-dimensional memory, say SDRAM, we can efficiently access data in consecutive location while executing row DFT. However, adjacent data along the columns are no longer in consecutive addresses, and the long strides that are necessary to gather data along columns could make the access latency about 10 times longer during column DFT computation [2].

To maintain row-wise burst for column DFT computations, we utilize the local memory on FPGA and store multiple columns of data. Let S be the size of the local memory, then we can store S/N_2 columns. If S/N_2 is less than the SDRAM's burst size B , then the SDRAM bandwidth is not utilized completely. For instance, if $B = 32$, $S = 16,384$ and $N_2 = 1024$, then $S/N_2 = 16 < 32$, the burst size. To utilize burst size, B , we decompose the column computations of size N_2 into 1D computation of size m followed by 1D computation of size p , where $N_2 = m \cdot p$. Let $L = S/B$, then $p \leq L$. The procedure can be represented mathematically as follows:

$$V_2 = P_{N_2,m} \cdot (I_m \otimes F_p) \cdot \tilde{D}_{N_2} \cdot (F_m \otimes I_p) \cdot U \cdot F_{N_1}, \quad (2)$$

where I_m and I_p are the identity matrices of sizes $m \times m$ and $p \times p$, \tilde{D}_{N_2} is a diagonal matrix of twiddle factors, $P_{N_2,m}$ is the column permutation, and \otimes is the Kronecker product. The corresponding algorithm is as follows:

[2D DFT Procedure]

- **Step 1. Row Operations:** This step includes two operations:
 - Step 1-a. Row FFT ($U \cdot F_{N_1}$): Compute the FFT along the rows of array U .
 - Step 1-b. Column stride FFT: Column FFT of size m followed by twiddle multiplications.
- **Step 2. Column local FFT:** Column FFT of size p followed by column-wise permutation.

Note that Step 1-b & Step 2 form the column FFT. The data access pattern for the 2D DFT is illustrated in Fig. 1. In Fig. 1(a), m rows spaced p rows apart are selected from the data array and $N_1 \times m$ data is sent to the FPGA local memory. Since the local memory is of size S , $S \geq N_1 \times m$. FFT of N_1

points is executed along each of these rows. Then, column-wise m -point FFT followed by twiddle factor multiplication is applied as shown in Fig. 1(b). The result is stored back in the same location of the data array in the SDRAM. After p iterations of Step 1-a & 1-b, all Row FFT and Column Stride FFT are completed. Then Step 2 which consists of p -point local FFT is computed. As shown in Fig. 1(c), L rows with B elements per row are stored in the local memory. Thus $S \geq B \times L$, where $L \geq p$. This enables p -point local FFT to be computed on these sub-columns. Note that if $N_2 \leq L$, column stride FFT can be skipped, because the whole column can fit in the local memory and the decomposition is not required. Next, the data along the rows have to be stored back to the correct row locations (based on the column-wise permutation, $P_{N_2,m}$) in the SDRAM.

2.2. 3D DFT Algorithm

3D DFT is a simple extension of 2D DFT. As illustrated in Fig. 1(d), the 2D DFT algorithm (described in Section 2.1) is computed on each of the N_3 2D slices parallel to the d_1 - d_2 plane. Next, 1D DFT of size N_3 is done along the d_3 -axis. Since adjacent data along the d_3 dimension are not in consecutive addresses in the SDRAM, to utilize the burst along d_1 dimension, we access data on a 2D slice parallel to d_1 - d_3 plane and apply the decomposition along d_3 dimension, as shown in Fig. 1(e). The mathematical description of this procedure is given by:

$$V_{1,3} = U_{1,3} \cdot F_{N_3} = U_{1,3} \cdot (F_{m'} \otimes I_{p'}) \tilde{D}_{N_3} (I_{m'} \otimes F_{p'}) P_{N_3,m'}, \quad (3)$$

where $U_{1,3}$ represents an input 2D array parallel to d_1 - d_3 plane and $V_{1,3}$ is the output array; $N_3 = m' \times p'$. The whole 3D DFT procedure can be summarized as follows:

[3D DFT Procedure]

- **Step 1. Run [2D DFT Procedure] on N_3 slices:** Compute 2D FFT on every 2D slice parallel to d_1 - d_2 plane.
- **Step 2. Stride FFT along d_3 dimension:** Compute m' -point stride FFT followed by twiddle multiplications on the 2D slice parallel to d_1 - d_3 plane.
- **Step 3. Local FFT along d_3 dimension:** Compute p' -point local FFT followed by the permutation on the 2D slice parallel to d_1 - d_3 plane.

Steps 2 & 3 have to be iterated N_2 times. After Step 3, the final results have to be stored to the correct locations in a different part of the SDRAM.

3. PROPOSED DFT ARCHITECTURE

The architecture proposed for MD DFT is shown in Fig. 2. The main components are an FPGA to do the processing and an SDRAM to store the data. The SDRAM controller fetches

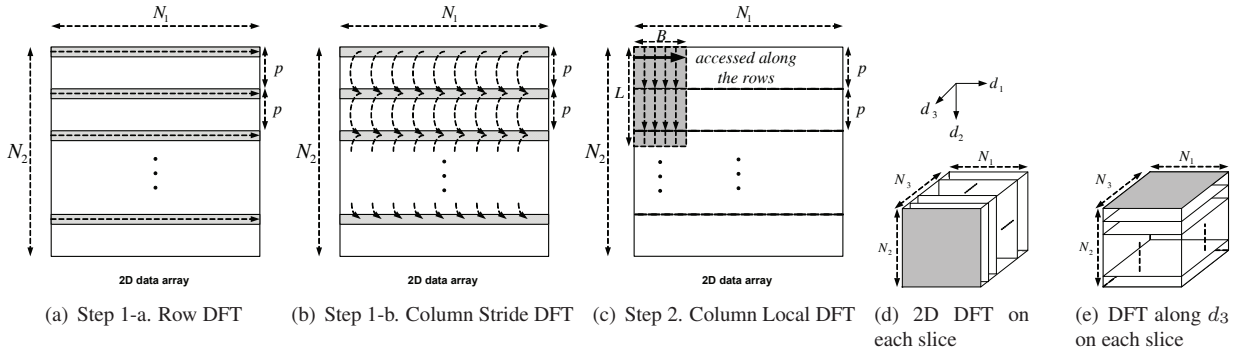


Fig. 1. The proposed data access pattern for 2D/3D DFT .

the input data from SDRAM and sends it to the local memory on the FPGA. The processing elements (PE) read this data, process it, and store the results back to the local memory. The SDRAM controller then reads these results and stores them back to the SDRAM. The main components of the architecture are described below:

Processing elements (PEs): A PE consists of a 1D FFT module followed by a complex multiplier. The 1D FFT module is of size N_1 for computing row FFTs as well as column stride/local FFT, where $N_1 \geq m, p$. In our design, we adopt Xilinx’s streaming FFT [6]. The complex multiplier is used to compute the twiddle multiplication after column stride FFT. The number of PEs depends on the required data throughput as well as the hardware resources available on the FPGA.

SDRAM: SDRAM is the main memory used to store the multi-dimensional data. In our implementation, a 2GB DDR2-400 DIMM is adopted. For 2D or 3D images, consecutive data along d_1 dimension is stored in consecutive locations in the SDRAM. We use Xilinx’s 64-bit Multi-Port Memory Controller (MPMC) [7] to control the SDRAM and a 128-bit Processor Local Bus (PLB) [8] to transfer the data between SDRAM and FPGA. The data is always accessed along d_1 dimension, thereby fully exploiting SDRAM’s bandwidth efficiency.

Dual local memory: There are two identical local memories of size S that serve as ping pong buffers. These local memories are implemented with dual-port Block RAMs on the FPGA. Unlike the SDRAM, non-consecutive addresses in the Block RAM can be accessed in contiguous clock cycles without performance penalty. Each local memory consists of multiple banks, so that multiple data can be received within one cycle from the SDRAM, and data in the local memory can be accessed by multiple 1D FFT IP cores at the same time. To support simultaneous accesses from multiple (up to r) PEs, each local memory is divided into r banks. If S is the size of the local memory, for the row operation, the SDRAM controller fetches the first S/m consecutive data (i.e. the first $S/(m \times N_1)$ rows) from the SDRAM and stores them into the banks starting from Bank 0. Then, the SDRAM controller

fetches the next S/m consecutive data starting from the p th row and stores them starting from Bank 1, and so on.

4. EVALUATION

The functionality of the proposed MD DFT architecture is first verified on the BEE3 board [5], which is equipped with a Virtex-5 LX155T FPGA. The FPGA can only accommodate one PE, which consists of a 2K-point FFT IP and a complex multiplier, and occupies 53 % of DSP48Es. The local memory, S , is 16,384 samples. Since there is one PE, one bank would have been sufficient. However, we choose to have 2 banks, since in each cycle the 128-bit PLB can transfer 2 complex samples with single precision and store them into the local memory. For maximum performance, PLB’s burst size is set to the largest value: 16 cycles. Thus, the effective burst size, B , is $16 \times 2 = 32$. Besides, m is set to be 8, because 8 rows of length 2048 can fit in the local memory. The clock frequency is set to 100MHz. A timer on the FPGA is used to count the clock cycles elapsed during the computations.

The computation time (measured) of 2D DFT for square and rectangular images of different sizes are listed in Table 1. These results show that column local FFT takes almost the same time as the row operations. This means that the row-wise burst access mode for the column local FFT computations have been able to achieve the same bandwidth efficiency as the row operations. The experiments also show that the proposed scheme works efficiently for both square and rectangular images.

Table 1. Measured computation time of 2D DFT on BEE3.

Shape	Image size ($N_1 \times N_2$)	Row operations (ms)	Column local FFT (ms)	Total (ms)
Square	128×128	0.89	0.90	1.79
	256×256	3.01	3.04	6.05
	512×512	12.14	12.72	24.86
	$1024 \times 1024^*$	50.21	52.42	102.63
	$2048 \times 2048^*$	202.45	209.65	412.10
Rectangle	$512 \times 2048^*$	50.34	52.37	102.71
	2048×512	50.11	52.47	102.58

(* : Stride column FFT is required.)

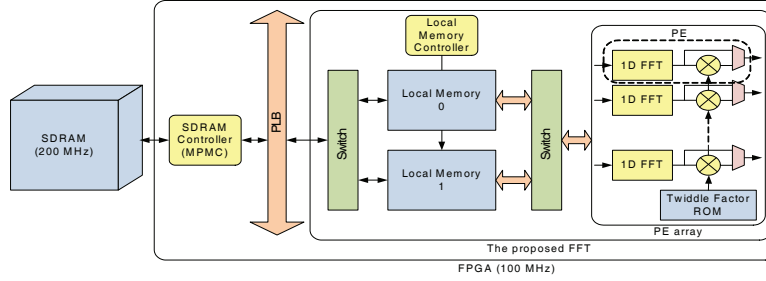


Fig. 2. The proposed MD DFT architecture.

In the current implementation on the BEE3 board, the SDRAM’s peak performance is not fully exploited due to the slow MPMC and PLB. In the $2K \times 2K$ DFT, the average data transfer rate is only 325.69MB/s, while the peak transfer rate of the SDRAM is 3200MB/s. To achieve a much higher performance, the proposed architecture is currently being ported onto the Xilinx ML605 FPGA board [9], which is equipped with a Virtex-6 LX240T FPGA and a DDR3-800 SDRAM. This FPGA accommodates 8 PEs and dual 8-bank local memory, where each bank is of size 2,048 samples. In addition, if a dedicated SDRAM controller is designed that can utilize 80% of the SDRAM’s bandwidth, the proposed design can complete $2K \times 2K$ DFT in 26.2ms. With this configuration, the proposed architecture will be able to outperform other 2D DFT solutions listed in Table 2 (assuming all solutions have the same data width). While our design requires only one SDRAM, other works require multiple memories (up to 4 banks, e.g. Eonic’s solution [1]). Lenart’s solution [2] requires additional transpose operation, while Dillon [3] utilizes SRAM which is faster. However, SRAM is usually more expensive, and its smaller capacity may not be sufficient for the large data volumes of typical 3D DFT applications.

Table 2. Comparison on computation times of $2K \times 2K$ DFT.

	Technology (clock freq.)	Complex data (bits)	External memory	Time (ms)	Normalized time [†] (ms)
Eonic[1]	ASIC, 0.18 μ m (128 MHz)	2×32	Quad SDRAM	84.0	84.0
Lenart[2]	ASIC, 0.13 μ m (250 MHz)	2×16	Dual SDRAM	50.0	100.0
Dillon[3]	Virtex-2p \times 2 (120 MHz)	2×8	Dual SDRAM	8.3	33.3
Proposed*	Virtex-6 (100 MHz)	2×32	Single SDRAM	26.2	26.2

(*: Timing based on simulations on ML605.)

([†]: Normalized to the same data width, 64 bits.)

The performance of the 3D DFT is summarized in Table 3. The results presented here are measured values, but they match very well with estimated result derived from 2D DFT measurements. In 128^3 DFT, for example, 2D DFT on the $128 d_1-d_2$ planes takes $128 \times 1.79 = 229.12$ ms, and column FFTs on the $128 d_1-d_3$ planes takes $128 \times 0.9 = 115.2$ ms. Thus, the total estimated time of 128^3 3D DFT is 344.32ms, which is pretty close to the experimental result, 348.24ms. This means SDRAM’s bandwidth efficiency in 2D DFT is maintained in

3D DFT. As in the 2D case, the time-consuming transpose operations have been avoided. With the 2GB SDRAM, the architecture can support up to 2^{27} samples, e.g. a 512^3 3D image. However, because of driver issues, we can currently only test data volumes up to 2^{22} samples. Finally, the performance can be significantly improved when the architecture is ported onto Xilinx ML605.

Table 3. Measured computation time of the proposed 3D DFT.

Image size ($N_1 \times N_2 \times N_3$)	Total (ms)
$128 \times 128 \times 128$	348.24
$64 \times 256 \times 256$	757.21
$256 \times 64 \times 256$	646.81
$256 \times 256 \times 64$	697.09

5. CONCLUSION

An efficient architecture of MD DFT has been proposed that does not require long stride memory accesses. It has been implemented on BEE3 FPGA board and validated for 2D and 3D data. To achieve higher performance, we are currently porting the architecture on the newly released Xilinx ML605 FPGA board. It will be able to support $2K \times 2K$ 2D DFT at more than 30 frames/sec.

6. REFERENCES

- [1] “PowerFFT ASIC,” <http://www.eonic.com/index.asp?item=32>.
- [2] T. Lenart and et al., “A hardware acceleration platform for digital holographic imaging,” *Journal of Signal Processing System*, vol. 52, no. 3, pp. 297–311, September, 2008.
- [3] T. Dillon, “Two Virtex-II FPGAs deliver fastest, cheapest, best high-performance image processing system,” *Xilinx Xcell Journal*, vol. 41, pp. 70–73, 2001.
- [4] J. Kim and et al., “FPGA architecture for 2D Discrete Fourier Transform based on 2D decomposition for large-sized data,” in *Proc. of IEEE Workshop on Signal Processing Systems (SIPS)*, October 2009.
- [5] “The BEE3 Hardware Platform,” <http://www.beecube.com/platform.html>.
- [6] “Xilinx FFT Logicore,” <http://www.xilinx.com/products/ipcenter/FFT.htm>.
- [7] “Xilinx Multi-port Memory Controller,” http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf.
- [8] “Processor Local Bus,” http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.
- [9] “Virtex-6 FPGA ML605 Evaluation Kit,” <http://www.xilinx.com/products/devkits/EK-V6-ML605-G.htm>.