Hindawi
International Journal of Reconfigurable Computing
Volume 2018, Article ID 1403181, 17 pages
https://doi.org/10.1155/2018/1403181

Research Article

Algorithm and Architecture Optimization for 2D Discrete Fourier Transforms with Simultaneous Edge Artifact Removal

Faisal Mahmood,[1] **Märt Toots**,[2] **Lars-Göran Öfverstedt**,[2] **and Ulf Skoglund**[2]

[1]*Department of Biomedical Engineering, Johns Hopkins University, Baltimore, MD 21218, USA*
[2]*Structural Cellular Biology Unit, Okinawa Institute of Science and Technology (OIST), Okinawa, Japan*

Correspondence should be addressed to Faisal Mahmood; faisalm@jhu.edu

Received 18 December 2017; Revised 11 May 2018; Accepted 10 June 2018; Published 6 August 2018

Academic Editor: João Cardoso

Two-dimensional discrete Fourier transform (DFT) is an extensively used and computationally intensive algorithm, with a plethora of applications. 2D images are, in general, nonperiodic but are assumed to be periodic while calculating their DFTs. This leads to cross-shaped artifacts in the frequency domain due to spectral leakage. These artifacts can have critical consequences if the DFTs are being used for further processing, specifically for biomedical applications. In this paper we present a novel FPGA-based solution to calculate 2D DFTs with simultaneous edge artifact removal for high-performance applications. Standard approaches for removing these artifacts, using apodization functions or mirroring, either involve removing critical frequencies or necessitate a surge in computation by significantly increasing the image size. We use a periodic plus smooth decomposition-based approach that was optimized to reduce DRAM access and to decrease 1D FFT invocations. 2D FFTs on FPGAs also suffer from the so-called "intermediate storage" or "memory wall" problem, which is due to limited on-chip memory, increasingly large image sizes, and strided column-wise external memory access. We propose a "tile-hopping" memory mapping scheme that significantly improves the bandwidth of the external memory for column-wise reads and can reduce the energy consumption up to 53%. We tested our proposed optimizations on a PXIe-based Xilinx Kintex 7 FPGA system communicating with a host PC, which gives us the advantage of further expanding the design for biomedical applications such as electron microscopy and tomography. We demonstrate that our proposed optimizations can lead to 2.8× reduced FPGA and DRAM energy consumption when calculating high-throughput 4096 × 4096 2D FFTs with simultaneous edge artifact removal. We also used our high-performance 2D FFT implementation to accelerate filtered back-projection for reconstructing tomographic data.

1. Introduction

Discrete Fourier Transform (DFT) is a commonly used and vitally important function for a vast variety of applications including, but not limited to, digital communication systems, image processing, computer vision, biomedical imaging, and biometrics [1, 2]. Fourier image analysis simplifies computations by converting complex convolution operations in the spatial domain to simple multiplications in the frequency domain. Due to the fundamental nature of 2D DFTs, they are commonly used in a variety of image processing applications such as tomographic image reconstruction [3], nonlinear interpolation, texture analysis, tracking, image quality assessment, and document analysis [4]. Because of their computational complexity, DFTs often become a computational constraint for applications requiring high-throughput and real-time or near real-time operations, specifically for machine vision applications [5]. Image sizes for many of these applications have also increased over the years, further contributing to the problem.

The Cooley-Tukey fast Fourier transform (FFT) algorithm [6], first proposed in 1965, reduces the complexity of DFTs from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ for a 1D DFT. However, in the case of 2D DFTs, 1D FFTs have to be computed in two dimensions, increasing the complexity to $\mathcal{O}(n^2 \log n)$, thereby making 2D DFTs a significant bottleneck for real-time machine vision applications [7]. Recently, there has been substantial effort to achieve high-performance implementations of multidimensional FFTs to overcome this constraint [5, 7–14]. Due to their inherent parallelism and reconfigurability, Field Programmable Gate Arrays (FPGAs) are attractive targets for accelerating FFT computations. Being a highly

flexible platform, FPGAs can fully exploit the parallel nature of the FFT algorithm. 2D FFTs are generally calculated in stages where all elements of the first stage must be available before the second stage can be calculated. This creates the so-called "*intermediate storage*" problem associated with strided external memory access, specifically for large datasets.

While calculating 2D DFTs, it is assumed that the image is periodic, which is usually not the case. The nonperiodic nature of the image leads to artifacts in the Fourier transform, usually known as edge artifacts or series termination errors. These artifacts appear as several crosses of high-amplitude coefficients in the frequency domain, as seen in [15, 16]. Such edge artifacts can be passed to subsequent stages of processing, and in biomedical applications they may lead to critical misinterpretations of results. Efficiently removing such artifacts without compromising resolution is a major problem. Moreover, simultaneously removing these spurious artifacts while calculating the 2D FFT adds to the existing complexity of the 2D FFT kernel.

Contributions. In this paper we present solutions for a high-performance 2D DFT with simultaneous edge artifact removal (EAR) for applications which require high frame rate 2D FFTs such as real-time medical imaging systems and machine vision for control. Our proposed optimizations lead to a high-performance solution for removing edge artifacts while the transform is being calculated, thus preventing time consuming and possibly erroneous postprocessing steps. Moreover, the proposed optimizations reduce the overall energy consumption. This work builds on our previous work presented in [8]. Major contributions include the following:

(1) We propose optimized periodic plus smooth decomposition (OPSD) as an optimization for standard periodic plus smooth decomposition (PSD) for edge artifact removal (Section 4).

(2) Based on OPSD, we propose an architecture that can reduce the access to DRAM and can decrease the number of 1D FFT invocations by performing column-by-column operations on the fly (Section 4).

(3) Since OPSD is heavily dependent on efficient FPGA-based 2D FFT implementation which is limited by DRAM access problems, we design a memory mapping scheme based on *"tile-hopping"*, which can reduce row activation overhead while accessing columns of data from the DRAM (Sections 5 and 6).

(4) The proposed OPSD and memory "tile-hopping" optimizations also lead to better energy performance as compared to row-major access (Section 6.4).

(5) We use our implementation as an accelerator for filtered back-projection (FBP), an analytical tomographic reconstruction method, and show that for large datasets our 2D FFT with edge artifact removal (EAR) can significantly improve reconstruction run time (Section 7).

As compared to our previous work [8], the current implementation achieves better runtime, i.e., 1.5ms as compared to 32.4ms for a 512×512 image. This increased performance is achieved by using state-of-the-art hardware for high-bandwidth communication between the FPGA and CPU modules, and by utilizing an efficient memory mapping scheme. The current work also analyzes the energy consumption of the proposed paradigm. Moreover, we also show how the real-time 2D FFTs can be used for tomographic reconstructions.

Paper Outline. The paper follows FPGA image processing design methodology outlined in [16, 17], which involves carefully profiling the software solution to understand computational bottlenecks and overcoming them through careful reformulation of the algorithm within a parallel hardware framework. Section 2 gives a comprehensive background of high-performance 2D FFTs using FPGAs, the DRAM intermediate storage problem, and edge artifacts. Section 3 presents PSD in detail which is the implementation objective. Section 4 presents OPSD, an optimized solution to reduce the latency of the serial part of the algorithm which limits overall performance. Section 5 presents a memory mapping scheme that can reduce the column-wise strided external memory access. Section 6 presents experimental results and explains target selection in detail, experimental setup, and benchmarks results. Section 7 presents filtered back-projection as a proposed application. Section 8 presents conclusions.

2. Background

2.1. High-Performance 2D FFTs Using FPGAs. There are several resource-efficient, high-throughput implementation approaches of multidimensional DFTs on a variety of different platforms. Many of these methods are software-based and have been optimized for efficient performance on general-purpose processors (GPPs), for example, Intel MKL [11], FFTW [9], and Spiral [10]. Implementations on GPPs can be readily adapted for a variety of scenarios. However, GPPs consume more power as compared to dedicated hardware and are not ideal for real-time embedded applications. Several application-specific integrated circuit- (ASIC-) based approaches have also been proposed [18–20], but since it is not easy to modify ASICs, they are not cost-effective solutions for rapid prototyping of image processing systems. Graphical Processing Units (GPUs) on the other hand can achieve relatively high throughput but are energy inefficient and limit the portability of large-scale imaging systems.

Due to their inherent parallelism and reconfigurability, FPGAs are attractive for accelerating FFT computations, since they fully exploit the parallel nature of the FFT algorithm. FPGAs are particularly an attractive target for medical and biomedical imaging apparatus and instruments such as electron microscopes and tomographic scanners. Such devices do not have to be manufactured in bulk to justify application-specific solutions and require high bandwidth. Moreover, increasing mobility and portability constitute a future objective for many medical imaging systems. FPGAs are also more efficient for prototyping machine vision applications since they are relatively more fine-grained when compared to GPPs and GPUs and can serve as a bridge

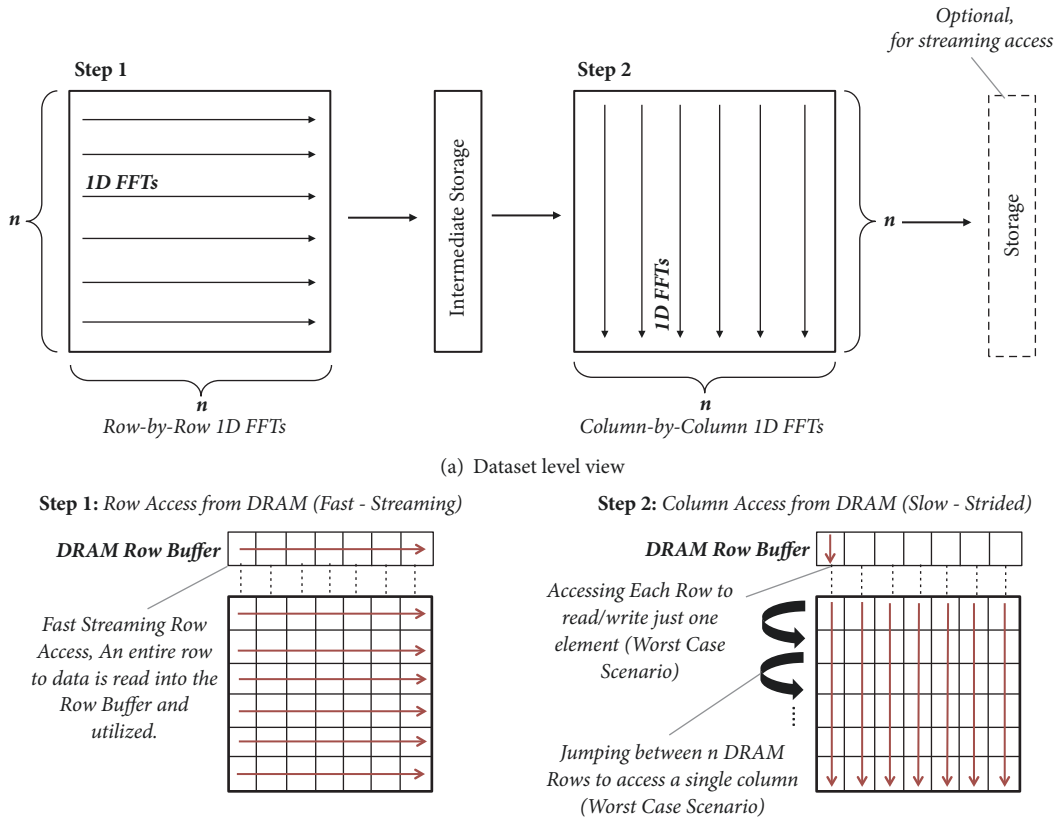


Figure 1: (a) An overview of row-column decomposition (RCD) for 2D FFT implementation. Intermediate storage is required because all elements of the row-by-row operations must be available for column-by-column processing. (b) An overview of strided column-wise access from DRAM as compared to trivial row-wise access. An entire row of elements must be read into the row buffer even to access a single element within a specific row.

between general-purpose and application-specific acceleration solutions.

2.2. DRAM Intermediate Storage Problem. There have been several high-throughput 2D FFT FPGA-based implementations over the past few years. Most of these rely on repeated invocations of 1D FFTs by row and column decomposition (RCD) with efficient use of memory [5, 7, 12, 21, 22]. RCD makes use of the fact that a 2D Fourier transform is separable and can be implemented in stages; i.e., a row-by-row 1D FFT can be proceeded by a column-by-column 1D FFT with intermediate storage (Figure 1). Most of the previous RCD-based 2D FFT FPGA implementation approaches have two major design challenges: (1) The 1D FFT implementation needs to have a reasonably high throughput and needs to be resource efficient. Moreover, spatial parallelism needs to be exploited by running several 1D FFTs simultaneously. (2) External DRAM needs to be efficiently addressed and to have a high bandwidth.

Since the column-by-column 1D FFT requires data from all rows, intermediate storage becomes a major problem for large datasets. Many implementations rely on local memory such as resource-implemented block RAM for intermediate storage which is not possible for large datasets [21]. Large datasets have to be offloaded to external DRAM because only a portion of the dataset that fits on the chip can be operated on at a given time. For complex image processing applications, this means repeated storage and access to the external memory during every stage of processing.

As shown in Figure 2(a) DRAM hierarchy from top to bottom is rank, chip, bank, row, and column. Each DRAM bank (Figure 2(b)) has a *row buffer* that holds the most recently referred row. There is only one *row buffer* per bank which means only one row from the data-grid can be accessed at once. Before accessing a row, it has to be activated by transferring the contents from internal capacitor storage into a set of parallel sense amplifiers. The *row buffer* is the so-called *"fast buffer"*, because when a row is activated and placed in the buffer, any element can be accessed at random.

If a new row has to be activated and accessed into the row buffer, a *row buffer miss* occurs and requires a higher latency, A_{miss} (Figure 2(c)). On the contrary if the desired row is already in the buffer, a *row buffer hit or page hit* occurs and the latency to access elements is substantially lower, A_{hit}. This implies $A_{miss} = A_{hit} + C_r$, where C_r is the overhead associated with accessing a new row to read a specific element (Figure 2(c)) [12]. There is also overhead involved in writing the row back to the data-grid (precharge), say, C_w.

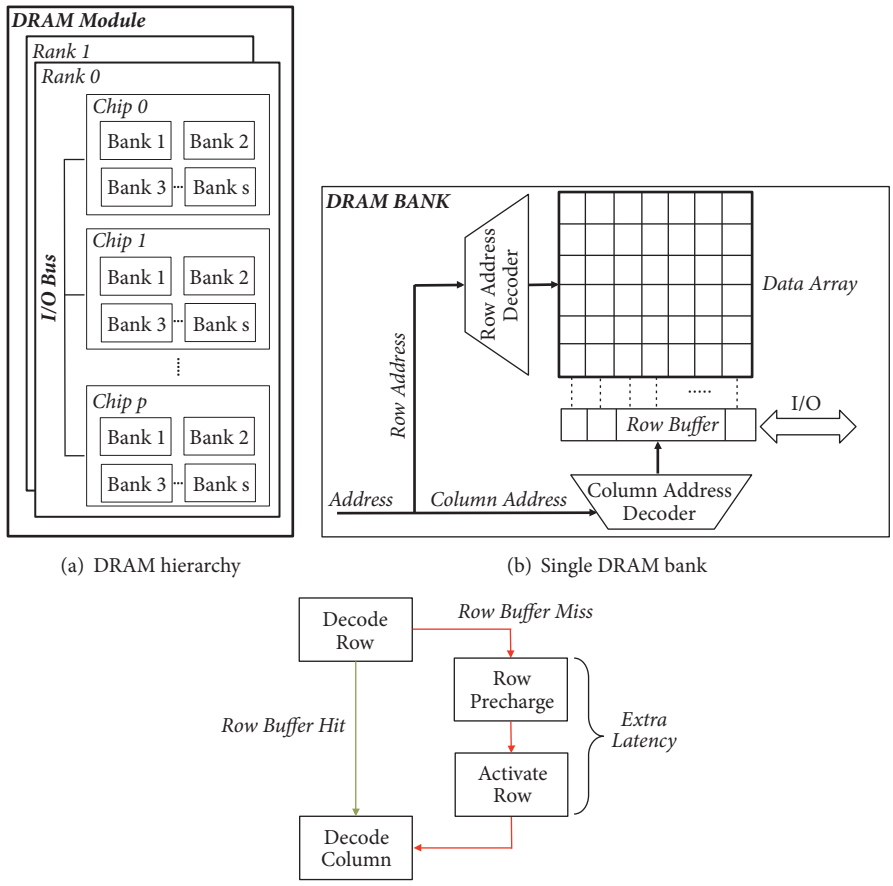


FIGURE 2: (a) An overview of the DRAM hierarchy. (b) Image showing the structure of a single DRAM bank. (c) Flow chart explaining additional latency introduced when a new row has to be referred to in the row buffer to access a specific element.

However, both C_r and C_w can be concealed by interleaving (switching between banks). Since row-wise access is trivial, the row-by-row 1D FFT part of RCD-based 2D FFT is easily accomplished. However, once the row-by-row 1D FFT is stored in the DRAM in standard row-major order, to access a single column, each row of the DRAM has to be accessed into the row buffer, rendering the read process extremely inefficient. This is typically the major bottleneck for high-throughput 2D FFTs (Figure 1(b)). We address this problem by designing a custom memory mapping scheme (Section 5).

2.3. Edge Artifacts. While calculating 2D DFTs, it is assumed that the image is periodic, which is usually not the case. The nonperiodic nature of the image leads to artifacts in the Fourier transform, usually known as edge artifacts or series termination errors. These artifacts appear as several crosses of high-amplitude coefficients in the frequency domain (Figure 3(b)). Such edge artifacts can be passed to subsequent stages of processing, and in biomedical applications they may lead to critical misinterpretations of results. No current 2D FFT FPGA implementation addresses this problem directly. These artifacts may be removed during preprocessing, using mirroring, windowing, zero padding, or postprocessing, e.g., filtering techniques. These techniques are usually computationally intensive, involve an increase in image size, and also tend to modify the transform.

The most common approach is by ramping the image at corner pixels to slowly attenuate the edges. Ramping is usually accomplished by an apodization function such as a Tukey (tapered cosine) or a Hamming window, which smoothly reduces the intensity to zero. Such an approach can be implemented on an FPGA as a preprocessing operation by storing the window function in a look-up table (LUT) and multiplying it with the image stream before calculating the FFT [16]. Although this approach is not extremely computationally intensive for small images, it inadvertently removes necessary information from the image. Loss of this information may have serious consequences if the image is being further processed with several other images to reconstruct a final image that is used for diagnostics or other decision-critical applications. Another common method is by mirroring the image from $N \times N$ to $2N \times 2N$. Doing so makes the image periodic and reduces edge artifacts. However, this not only increases the size of the image by 4×, but also makes the transform symmetric, which generates an inaccurate phase component.

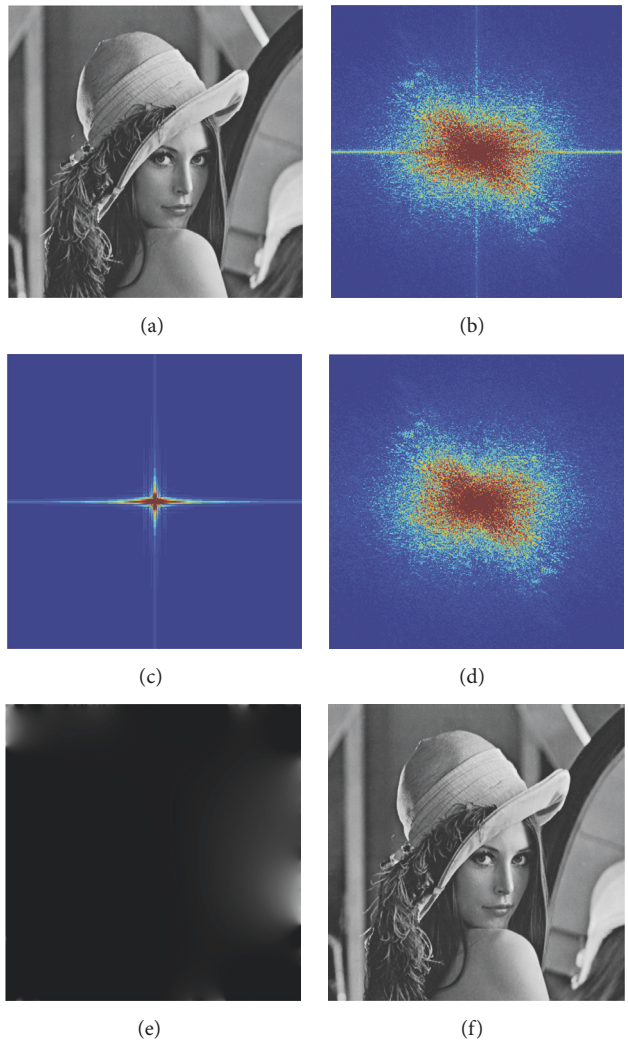


FIGURE 3: (1a) An image with nonperiodic boundary. (1b) 2D DFT of (1a). (1c) DFT of the smooth component, i.e., the removed artifacts from (1a). (1d) Periodic component, i.e., DFT of (1a) with edge artifacts removed. (1e) Reconstructed smooth component. (1f) Reconstructed periodic component.

Simultaneously removing the edge artifacts while calculating a 2D FFT imposes an additional design challenge, regardless of the method used. However, these artifacts must be removed in applications where they may be propagated to subsequent processing levels. An ideal method for removing these artifacts should involve making the image periodic while removing minimal information from the image. Periodic plus smooth decomposition (PSD), first presented by Moisan [4] and used in [23–25], is an ideal method for removing edge artifacts (specifically for biomedical applications) because it does not directly intervene with pixels beside those of the boundary and does not increase image size. Moreover, its inherently parallel nature makes it ideal for a high-throughput, FPGA-based implementation. We have further optimized the original PSD decomposition algorithm to make the overall implementation much more efficient, by decreasing the number of required 1D FFT invocations and by reducing external DRAM utilization (Section 4).

2.4. LabVIEW FPGA High-Level Design Environment. A major concern while designing complex image processing hardware accelerators is how to fully harness on the divide-and-conquer approach. Algorithms that have to be mapped to multiple FPGAs are often marred by communication problems, and custom FPGA boards reduce flexibility for large-scale and evolving designs. For rapid prototyping of our algorithms, we used LabVIEW FPGA 2016 (National Instruments), a robust data-flow-based graphical design environment. LabVIEW FPGA provides integration with National Instruments (NI) Xilinx-based reconfigurable hardware, allowing efficient communication with a host PC and high-throughput communication between multiple FPGAs through a PXIe (PCI eXtensions for Industry Express) bus. LabVIEW FPGA also enables us to integrate external Hardware Description Language (HDL) code and gives us the flexibility to expand our design for future processing stages. We used NI PXIe-7976R FPGA boards that have a Xilinx Kintex 7 FPGA and 2GB high-bandwidth (10GB/s) external memory. This platform has already been extensively used for rapid prototyping of communication standards and protocols before moving to ASIC designs. The optimizations and designs we present here are scalable to most reconfigurable computing-based systems. Moreover, LabVIEW FPGA provides efficient high-level control over memory via a smart memory controller.

3. Periodic Plus Smooth Decomposition (PSD) for Edge Artifact Removal (EAR)

Periodic plus smooth decomposition (PSD) involves decomposing the image into a periodic and smooth component to remove edge artifacts with minimal loss of information from the image [4]. This section presents an overview of the PSD algorithm and profiles the algorithm for possible parallelization and optimization to achieve efficient FPGA implementation.

Let us have discrete n by m gray-scale image $\boldsymbol{I}$ on a finite domain $\Omega = \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, m-1\}$. The discrete Fourier transform (DFT) of $\boldsymbol{I}$ is defined as

$$\widehat{\boldsymbol{I}}(s,t) = \sum_{(i,j)\in\Omega} \boldsymbol{I}(i,j) \exp\left(-\imath 2\pi\left(\frac{si}{n} + \frac{tj}{m}\right)\right). \quad (1)$$

This is equivalent to a matrix multiplication $\boldsymbol{WIV}$, where

$$\boldsymbol{W} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{n-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & w^{n-2} & w^{2(n-2)} & \cdots & w^{(n-2)(n-1)} \\ 1 & w^{n-1} & w^{2(n-1)} & \cdots & w^{(n-1)(n-1)} \end{pmatrix} \quad (2)$$

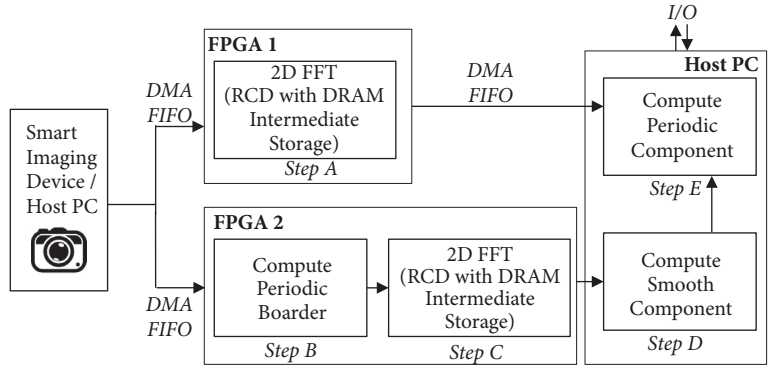


Figure 4: A top-level architecture for OPSD using two FPGAs and a host PC connected over a high-bandwidth bus. The steps are associated with Algorithm 1.

and

$$w^k = \exp\left(-i\frac{2\pi}{n}\right)^k = \exp\left(-i\frac{2\pi k}{n}\right). \tag{3}$$

$\boldsymbol{V}$ has the same structure as $\boldsymbol{W}$ but is m-dimensional. w^k has period n which means that $w^k = w^{k+ln}$, $\forall k, l \in \mathbb{N}$; therefore,

$$\boldsymbol{W} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 1 & w & w^2 & \dots & w^{n-2} & w^{n-1} \\ 1 & w^2 & w^4 & \dots & w^{n-4} & w^{n-2} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & w^{n-2} & w^{n-4} & \dots & w^4 & w^2 \\ 1 & w^{n-1} & w^{n-2} & \dots & w^2 & w^1 \end{pmatrix}. \tag{4}$$

Since in general $\boldsymbol{I}$ is not (n, m)-periodic, there will be high-amplitude edge artifacts present in the DFT stemming from sharp discontinuities between the opposing edges of the image as shown in Figure 3(b). Reference [4] proposed a decomposition of I into a periodic component $\boldsymbol{P}$, which is periodic and captures the essence of the image with all high-frequency details, and a smoothly varying background $\boldsymbol{S}$, which recreates the discontinuities at the borders, so $\boldsymbol{I} = \boldsymbol{P} + \boldsymbol{S}$. Periodic plus smooth decomposition can be computed by first constructing a border image $\boldsymbol{B} = \boldsymbol{R} + \boldsymbol{C}$, where $\boldsymbol{R}$ represents the boundary discontinuities when transitioning row-wise and $\boldsymbol{C}$ when going column-wise.

$$\begin{aligned} &\boldsymbol{R}(i,j) \\ &\quad = \begin{cases} \boldsymbol{I}(n-1-i,j) - \boldsymbol{I}(i,j), & i = 0 \text{ or } i = n-1 \\ \boldsymbol{0}, & \text{otherwise} \end{cases} \\ &\boldsymbol{C}(i,j) \\ &\quad = \begin{cases} \boldsymbol{I}(i,m-1-j) - \boldsymbol{I}(i,j), & j = 0 \text{ or } j = m-1 \\ \boldsymbol{0}, & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

It is obvious that the structure of the border image $\boldsymbol{B}$ is simple with nonzero values only in the edges as shown below:

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{R} + \boldsymbol{C} \\ &= \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1,m-1} & b_{1m} \\ b_{21} & 0 & \dots & 0 & -b_{21} \\ \dots & \dots & \dots & \dots & \dots \\ b_{n-1,1} & 0 & \dots & 0 & -b_{n-1,1} \\ b_{n1} & -b_{12} & \dots & -b_{1,m-1} & -b_{nm} \end{pmatrix}. \end{aligned} \tag{6}$$

The DFT of the smooth component $\boldsymbol{S}$ can be then found by the following formula:

$$\widehat{\boldsymbol{S}}(s,t) = \frac{\widehat{\boldsymbol{B}}(s,t)}{2\cos(2\pi s/n) + 2\cos(2\pi t/m) - 4}, \quad \forall (s,t) \in \Omega \setminus \{(0,0)\}. \tag{7}$$

The DFT of the image $\boldsymbol{I}$ with edge artifacts removed is then $\widehat{\boldsymbol{P}} = \widehat{\boldsymbol{I}} - \widehat{\boldsymbol{S}}$. Figures 3(c) and 3(d) show the DFT of the smooth and periodic components, respectively. Figures 3(e) and 3(f) show the reconstructed periodic and smooth components. On reconstruction, it is evident that there is negligible visual difference between the actual image and the periodic reconstructed image for this example.

3.1. Profiling PSD for FPGA Implementation. Algorithm 1 summarizes the overall PSD implementation. There are several ways of arranging the algorithm. We have arranged it so that DFTs of the periodic and smooth components are readily available for further processing stages. For best results, both the periodic and smooth components should undergo similar processing stages and should be added back together before displaying the result. However, depending on the application it might be acceptable to discard the periodic component completely. For an $n \times m$ image, steps A and C have a complexity of $\mathcal{O}(nm\log(nm))$ and steps B and D have complexity $\mathcal{O}(m + n)$ and $\mathcal{O}(mn)$, respectively. Computationally the performance of PSD is limited by steps A and C. Figure 4 shows a proposed top-level architecture

where step A and steps B, C, and D are completed on separate FPGAs while step E can be done on the host PC. Two high end FPGAs are used instead of one because resources on one FPGA are insufficient to compute a large size 2D FFT as well its edge artifact removal components. The overall performance may be limited by FPGA 2 where most of the serial part of the algorithm lies. There are two major factors which limit the throughput of such a design:

(1) While FPGA 1 and FPGA 2 can run in parallel, the result of step A from FPGA 1 has to be stored on the host PC while steps B, C, and D are completed on FPGA 2 before step E can be completed on the host PC.

(2) The DRAM intermediate storage problem explained in Section 2.2 and Figures 1 and 2 has to be addressed since strided access to the DRAM for column-wise operations can significantly limit throughput.

As for (1), it has been addressed in the next section where we make use of the inherent symmetry of the boundary image to reduce the time required to compute the 2D FFT of the boundary image. As for (2), it has been addressed by designing a semi-custom memory mapping controller which *"tiles"* the DRAM floor and *"hops"* between several tiles so as to minimize strided memory access.

4. Proposed: Optimized Periodic Plus Smooth Decomposition (OPSD)

In this section, we optimize the original PSD algorithm. This optimization is to effectively reduce the number of 1D FFT invocations and the number of times the DRAM is accessed.

Equation (6) shows that, except the corners, the boundary image $\boldsymbol{B}$ is symmetrical in the sense that boundary rows and columns are algebraic negation of each other. In total $\boldsymbol{B}$ has $n+m-1$ unique elements, with the following relations between corners with respect to columns and rows:

$$\begin{aligned} b_{11} &= r_{11} + c_{11} \\ b_{1m} &= r_{1m} - c_{11} \\ b_{n1} &= -r_{11} + c_{n1} \\ b_{nm} &= -r_{1m} - c_{n1} \end{aligned} \tag{8}$$

$$\Longrightarrow b_{nm} = -b_{11} - b_{1m} - b_{n1}. \tag{9}$$

In computing the FFT of $\boldsymbol{B}$, one normally proceeds by first running 1D FFTs column-by-column and then 1D FFTs row-by-row or vice versa. An FFT of a column vector $\boldsymbol{v}$ with length n is $\boldsymbol{W}\boldsymbol{v}$, where $\boldsymbol{W}$ is given in (4). The column-wise FFT of the matrix $\boldsymbol{B}$ is then

$$\widehat{\boldsymbol{B}} = \boldsymbol{W}\boldsymbol{B}. \tag{10}$$

Let us have a closer look at the first column, denoted by $\boldsymbol{B}_{\cdot\mathbf{1}}$. The 1D FFT of this vector is

$$\begin{aligned} \widehat{\boldsymbol{B}}_{\cdot\mathbf{1}} &= \boldsymbol{W}\boldsymbol{B}_{\cdot\mathbf{1}} \\ &= \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & w & \dots & w^{n-1} \\ 1 & w^2 & \dots & w^{2(n-1)} \\ \dots & \dots & \dots & \dots \\ 1 & w^{n-2} & \dots & w^{(n-2)(n-1)} \\ 1 & w^{n-1} & \dots & w^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} b_{11} \\ b_{21} \\ b_{31} \\ \dots \\ b_{n-1,1} \\ b_{n1} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{i=1}^{n} b_{i1} \\ \sum_{i=1}^{n} b_{i1} w^{i-1} \\ \sum_{i=1}^{n} b_{i1} w^{2(i-1)} \\ \dots \\ \sum_{i=1}^{n} b_{i1} w^{(n-2)(i-1)} \\ \sum_{i=1}^{n} b_{i1} w^{(n-1)(i-1)} \end{pmatrix}. \end{aligned} \tag{11}$$

It can be shown that the 1D FFT of the column $j \in \{2, 3, \dots, m-1\}$ is

$$\begin{aligned} \widehat{\boldsymbol{B}}_{\cdot j} &= \boldsymbol{W}\boldsymbol{B}_{\cdot j} \\ &= \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & w & \dots & w^{n-1} \\ 1 & w^2 & \dots & w^{2(n-1)} \\ \dots & \dots & \dots & \dots \\ 1 & w^{n-2} & \dots & w^{(n-2)(n-1)} \\ 1 & w^{n-1} & \dots & w^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} b_{1j} \\ 0 \\ 0 \\ \dots \\ 0 \\ -b_{1j} \end{pmatrix} \\ &= b_{1j} \begin{pmatrix} 0 \\ 1 - w^{n-1} \\ 1 - w^{n-2} \\ \dots \\ 1 - w^2 \\ 1 - w \end{pmatrix} = b_{1j}\boldsymbol{\nu}, \end{aligned} \tag{12}$$

Input: $\boldsymbol{I}(i, j)$ of size $n \times m$
Output: $\widehat{\boldsymbol{P}}(s,t)$ $\widehat{\boldsymbol{S}}(s,t)$
Step A: Compute the 2D DFT of image $\boldsymbol{I}(i, j)$:
$\boldsymbol{I}(i, j) \xrightarrow{\mathscr{F}} \widehat{\boldsymbol{I}}(s,t)$
Step B: Compute periodic border $\boldsymbol{B}$:
while $1 < j < m$ **do**
  while $1 < i < n$ **do**
    if $(i = 0 \vee i = n-1)$ **then**
      $\boldsymbol{R}(i, j) \longleftarrow \boldsymbol{I}(n-1-i, j) - \boldsymbol{I}(i, j)$
    else
      $\boldsymbol{R}(i, j) \longleftarrow 0$
    end if
    if $(j = 0 \vee j = m-1)$ **then**
      $\boldsymbol{C}(i, j) \longleftarrow \boldsymbol{I}(i, m-1-j) - \boldsymbol{I}(i, j)$
    else
      $\boldsymbol{C}(i, j) \longleftarrow 0$
    end if
  end while
end while
$\boldsymbol{B} \longleftarrow \boldsymbol{R} + \boldsymbol{C}$
Step C: Compute the 2D DFT of $\widehat{\boldsymbol{B}}(s,t)$:
$\boldsymbol{B}(i, j) \xrightarrow{\mathscr{F}} \widehat{\boldsymbol{B}}(s,t)$
Step D: Compute the Smooth Component $\widehat{\boldsymbol{S}}(s,t)$:
$\widehat{\boldsymbol{D}}(s,t) \longleftarrow 2\cos(2\pi s/n) + 2\cos(2\pi t/m) - 4$
$\widehat{\boldsymbol{S}}(s,t) \longleftarrow \widehat{\boldsymbol{B}}(s,t) \div \widehat{\boldsymbol{D}}(s,t)$
Step E: Compute the Periodic Component $\widehat{\boldsymbol{P}}(s,t)$:
$\widehat{\boldsymbol{P}}(s,t) \longleftarrow \widehat{\boldsymbol{I}}(s,t) - \widehat{\boldsymbol{S}}(s,t)$
return $\widehat{\boldsymbol{P}}(s,t)$ $\widehat{\boldsymbol{S}}(s,t)$

Algorithm 1: Periodic plus smooth decomposition (PSD).

and the 1D FFT of the last column $\boldsymbol{B}_{\cdot\boldsymbol{m}}$ is

$$\widehat{\boldsymbol{B}}_{\cdot\boldsymbol{m}} = \boldsymbol{W}\boldsymbol{B}_{\cdot\boldsymbol{m}} \tag{13}$$

$$= \begin{pmatrix} -\sum_{i=1}^{n} b_{i1} \\ -\sum_{i=1}^{n} b_{i1} w^{i-1} + (b_{11} + b_{1m})\left(1 - w^{n-1}\right) \\ -\sum_{i=1}^{n} b_{i1} w^{2(i-1)} + (b_{11} + b_{1m})\left(1 - w^{2(n-1)}\right) \\ \cdots \\ -\sum_{i=1}^{n} b_{i1} w^{(n-2)(i-1)} + (b_{11} + b_{1m})\left(1 - w^{(n-2)(n-1)}\right) \\ -\sum_{i=1}^{n} b_{i1} w^{(n-1)(i-1)} + (b_{11} + b_{1m})\left(1 - w^{(n-1)(n-1)}\right) \end{pmatrix} \tag{14}$$

$$\widehat{\boldsymbol{B}}_{\cdot\boldsymbol{m}} = -\widehat{\boldsymbol{B}}_{\cdot\boldsymbol{1}} + (b_{11} + b_{1m})\,\boldsymbol{\nu}. \tag{15}$$

Therefore, the column-wise FFT of the matrix $\boldsymbol{B}$ is

$$\widehat{\boldsymbol{B}} = \left(\widehat{\boldsymbol{B}_{\cdot\boldsymbol{1}}} \;\; b_{12}\boldsymbol{\nu} \;\; \cdots \;\; b_{1,m-1}\boldsymbol{\nu} \;\; -\widehat{\boldsymbol{B}}_{\cdot\boldsymbol{1}} + (b_{11} + b_{1m})\,\boldsymbol{\nu}\right). \tag{16}$$

To compute the column-by-column 1D FFT of the matrix, $\boldsymbol{B}$, we only have to compute the FFT of the first vector and then use the appropriately scaled vector, $\boldsymbol{\nu}$, to derive the remainder of the columns. The row-by-row FFT has to be calculated in a row burst normal way. Algorithm 2 presents a summary of the shortcut for calculating $\widehat{\boldsymbol{B}}(s,t)$. The steps presented in Algorithm 2 can replace step C in Algorithm 1. By reducing column-by-column 1D FFT computations for the boundary image, this method can significantly reduce the number of 1D FFT invocations, reduce the overall DRAM access, and eliminate problematic column-wise strided DRAM access for an efficient FPGA-based implementation. For column-wise operations, a single 1D FFT of size m is required rather than nm 1D FFTs of size m. Moreover, since one has to simply store one column of data, it can be stored on the on-chip local memory (BRAM or SRAM). This can be implemented by temporarily storing the initial vector $\widehat{\boldsymbol{B}}_{\cdot\boldsymbol{1}}$ and scaling factors b_{1j} in the block RAM/register memory, drastically reducing DRAM access, and lowering the number of required 1D FFT invocations. Performance evaluation for this has been presented in the results section.

Table 1 shows a comparison of mirroring, PSD, and our proposed OPSD with respect to DRAM access points. Mirroring has been used for comparison purposes because it is an alternative technique that reduces edge artifacts while maintaining maximum amplitude information. However, due to replication of the image, most of the phase information is lost. Figure 5 graphically shows that our OPSD method significantly reduces reading from external memory and reduces

Input: $\boldsymbol{B}(i, j)$ of size $n \times m$
Output: $\widehat{\boldsymbol{B}}(s, t)$
$2D\ \mathcal{F}(\boldsymbol{B}) \Longleftrightarrow \boldsymbol{B} \xrightarrow{\mathcal{F}_{cw}} \widehat{\boldsymbol{B}}_{\boldsymbol{cw}} \xrightarrow{\mathcal{F}_{rw}} \widehat{\boldsymbol{B}}$
Column-by-Column DFT via Symmetrical Short-cut:
1: $\boldsymbol{B}_{\cdot\mathbf{1}} \xrightarrow{\mathcal{F}} \widehat{\boldsymbol{B}}_{\cdot\mathbf{1}}$
2: **while** $1 < j < m$ **do**
3: $\quad \widehat{\boldsymbol{B}}_{\cdot j} \longleftarrow b_{1j}\boldsymbol{\nu}$
4: **end while**
5: $\widehat{\boldsymbol{B}}_{\cdot\boldsymbol{m}} \longleftarrow -\widehat{\boldsymbol{B}}_{\cdot\mathbf{1}} + (b_{11} + b_{1m})\boldsymbol{\nu}$
6: $\widehat{\boldsymbol{B}}_{\boldsymbol{cw}} \longleftarrow$ *Concatenate* $\widehat{\boldsymbol{B}}_{\cdot\mathbf{1}}\ \widehat{\boldsymbol{B}}_{\cdot\mathbf{2}}\ \ldots\ \widehat{\boldsymbol{B}}_{\cdot\boldsymbol{m}-\mathbf{1}}\ \widehat{\boldsymbol{B}}_{\cdot\boldsymbol{m}}$
Row-by-Row DFT:
7: $\widehat{\boldsymbol{B}}_{\boldsymbol{cw}} \xrightarrow{\mathcal{F}_{rw}} \widehat{\boldsymbol{B}}(s, t)$
8: **return** $\widehat{\boldsymbol{B}}(s, t)$
cw: column-wise/column-by-column
rw: row-wise/row-by-row

Algorithm 2: Proposed symmetrically optimized computation of $\widehat{\boldsymbol{B}}(s, t)$.

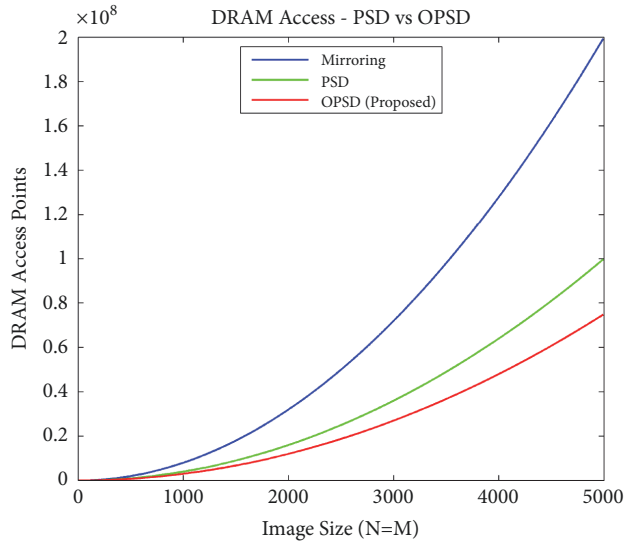


Figure 5: Graph showing DRAM access (equal to number of DFT points to be computed) with increasing image size for mirroring, periodic plus smooth decomposition (PSD), and our proposed optimized period plus smooth decomposition (OPSD).

the overall number of DFT computations required. It should be noted that such optimization is only possible for either column-wise or row-wise operations because after either of these operations the output is not symmetrical anymore. Completing the column-wise operation first prevents strided reading; however, this results in strided writing to the DRAM before row-wise traversal can start. This can be minimized by making efficient use of the local block RAM. Output columns are stored in the block RAM before being written to the DRAM in patches such that each row buffer access writes elements in several columns.

5. Proposed: Tile-Hopping Memory Mapping for 2D FFTs

In this section we propose a *tile-hopping* external memory access pattern for efficiently addressing external memory during intermediate storage between row-wise and column-wise 1D FFT operations to calculate a 2D FFT. As explained in Section 2.2 and Figure 2, column-wise reads from DRAM can be costly due to the overhead associated with activating and precharging. In the worst case scenario it can limit DRAM bandwidth up to 80% [26]. This is a problem with all such image processing operations where one stage of the processing has to be completed on all elements before the next stage can start. In the past there have been several implementations using local memory; however, with growing demand for larger image sizes external memory has to be used. There have been several DRAM remapping attempts before, such as [5, 13]. They propose a tile-based approach where an $n \times n$ image (input array) is divided into $n/k \times n/k$ tiles where k is the size of the DRAM row buffer which allows for very high-bandwidth DRAM access. Although, this

Table 1: Comparing mirroring, PSD, and OPSD.

Algorithm	DRAM Access Points	DFT Points
Mirroring	$8NM$	$8NM$
P + S Decomposition (PSD)	$4NM$	$4NM$
Optimized PSD (Proposed)	$3NM + N + M - 1$	$3NM + M$

method may be ideal to maximize the DRAM performance for 2D FFTs, it incurs a high resource cost associated with local memory transposition and storing large chunks of data (entire row/column of tiles) in the local memory. Moreover, tiling in the image domain also requires remapping row-by-row operations. Another approach to reducing strided DRAM access has been presented in [27]. They present a 2D decomposition algorithm which decomposes the problem into smaller subblock 2D FFTs which can be performed locally. This introduces extra row and column data exchanges, and total number of operations are increased from $\mathcal{O}(n^2 \log n)$ to $\mathcal{O}(n^2(1+\log n))$. Other implementations do not address the external memory issue in detail.

We propose *tile-hopping* address mapping which reduces the number of row activations required to access a single column. Unlike [5], our approach does not require significant local operations or storage. The proposed memory mapping controller was designed on top of LabVIEW FPGA's existing memory controller which efficiently controls interleaving and issues activation and precharge commands in parallel with data transfer. The reduced number of row activations also reduces the amount of energy required by the DRAM. This will be further discussed in the energy evaluation presented in Section 6.4 and Table 3 where we demonstrate that the proposed *tile-hopping* address mapping can reduce energy consumption up to 53%.

Instead of writing the results of the row-by-row 1D FFT in row-major order we remap the results in a blocked or tiled pattern as shown in Figure 6. This means that when accessing an image column, several elements of that column can be retrieved from a single DRAM row access. For an $n \times n$ image, each row of size n can be divided into h tiles (i.e., $n = h.N(t)$, where $N(t)$ is the number of elements in each tile). These tiles can be remapped onto the DRAM floor as shown in Figure 6. If the size of the row is small enough, it may be possible to convert it into a single tile (i.e., $h = 1$). However, this is unlikely for realistic image sizes. For a tile of size $p \times q$, a single row of the image is written into the DRAM by transitioning through $p.h$ rows. If k is the size of the row buffer, there are k/q distinct tiles represented in each DRAM row and it contains the same number of elements from a single image column. Given regular row-major storage when accessing column-wise elements, one would have to transition through n DRAM rows to read a single image column. However, with this approach, when accessing an image column, k/q elements of that column could be read from a single DRAM row which has been referred to in the row buffer. Although the cost of writing an image row is higher when compared to a standard row-major DRAM writing pattern, (i.e., referring to $p.h$ rather than n rows), the number of DRAM row referrals during column-wise read is reduced to $n.q/k$ which is $n.(1 - q/k)$ less row referrals for a single column read.

We refer to this method as *tile-hopping* because it entails mapping data onto several DRAM tiles and then hopping between the tiles such that several elements of the image column exist in a DRAM row which has been referred to in the row bank. Although this mapping scheme has been developed for column-wise access required during 2D FFT calculation, the scheme is general and can be adapted to other applications. Performance evaluation of this method has been presented in the experiments section.

6. Experimental Results and Analysis

6.1. Hardware Configuration and Target Selection. Since 2D DFTs are usually used for simplifying convolution operations in complex image processing and machine vision systems, we needed to prototype our design on a system that is expandable for next levels of processing. As mentioned earlier, for rapid prototyping of our proposed OPSD algorithm and tile-hopping memory mapping scheme, we used a PXIe-based reconfigurable system. PXIe is an industrial extension of a PCI system with an enhanced bus structure that gives each connected device dedicated access to the bus with a maximum throughput of 24GB/s. This allows a high-speed dedicated link between a host PC and several FPGAs. The LabVIEW FPGA graphical design environment is efficient for rapid prototyping of complicated signal and image processing systems. It allows us to effectively integrate external HDL code and LabVIEW graphical design on a single platform. Moreover, it allows a combination of high-level synthesis (HLS) and custom logic. Since current HLS tools have limitations when it comes to complex image and signal processing tasks, LabVIEW FPGA tries to bridge these gaps by streamlining the design process.

We used FlexRIO (Flexible Reconfigurable I/O) FPGA boards plugged into a PXIe chassis. PXIe FlexRIO FPGA boards are adaptable and can be used to achieve high throughput, because they allow direct data transfer between multiple FPGAs at rates as high as 8GB/s. This can significantly simplify multi-FPGA systems, which usually communicate via a host PC. This feature allows expansion of our system to further processing stages, making it flexible for a variety of applications. Figure 7 shows a basic overview of a PXIe-based, multi-FPGA system with a host PC controller connected through a high-speed bus on a PXIe chassis.

Specifically, we used two NI PXIe-7976R FlexRIO boards which have Kintex 7 FPGA and 2GB external DRAM with theoretical data bandwidth up to 10GB/s. This FPGA board was plugged into a PXIe-1085 chassis along with a PXIe-8880 Intel Xeon PC controller. PXIe-1085 can hold up to 16 FPGAs and has 8 GB/s per-slot dedicated bandwidth and an overall system bandwidth of 24 GB/s.

6.2. Experimental Setup. As per Algorithms 1 and 2, discussed in previous sections, implementation involves five stages: (A) calculating the 2D FFT of an image frame, (B)

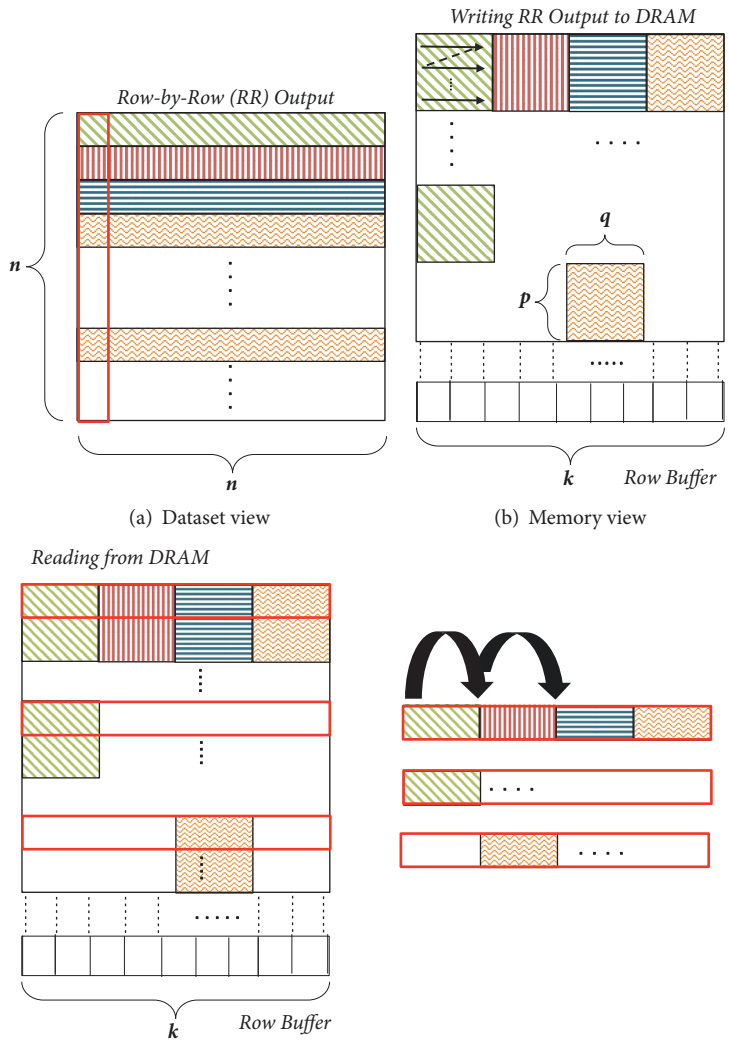


FIGURE 6: Image showing tile-hopping. (a) Image-level view showing tiles. (b) DRAM-level view showing tile placement while writing. (c) DRAM-level view showing column reading from the tiles.

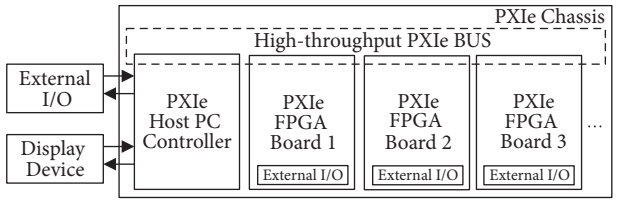


FIGURE 7: Block diagram of a PXIe-based multi-FPGA system with a host PC controller connected through a high-speed bus on a PXIe chassis [8].

calculating the boundary image, (C) calculating the 2D FFT of the boundary image, (D) calculating the smooth component, and (E) subtracting the smooth component from the 2D FFT of the original image to achieve the periodic component. The bottleneck consistently occurs in A and the serial part of the algorithm (A $\longrightarrow$ B $\longrightarrow$ C). The limitation due to A is reduced removing the so-called "memory wall" by using our proposed *tile-hopping*-based memory mapping. The limitations due to the serial part of the algorithm are reduced by using OPSD rather than PSD. For quantification, the delay for A is 0.62ms for a 512×512 image.

The design flow presented in Figure 4 was followed. Data-flow is clearly shown in a graphical programming environment making it easier to visualize how a design efficiently fits on an FPGA. Highly efficient implementations of 1D FFT were used from LabVIEW FPGA for parallel row-by-row operations and by integrating Xilinx LogiCORE for column-by-column operations. Each stage of the design was dynamically tested and benchmarked. The image was streamed from the host PC using a Direct Memory Access (DMA) FIFO. 1D FFTs are performed in parallel rows of 8 and stored in the DRAM via local memory in a tiled pattern as explained in the previous section. This follows reading several rows to extract a single column which is Fourier-transformed using Xilinx LogiCORE and is sent back to

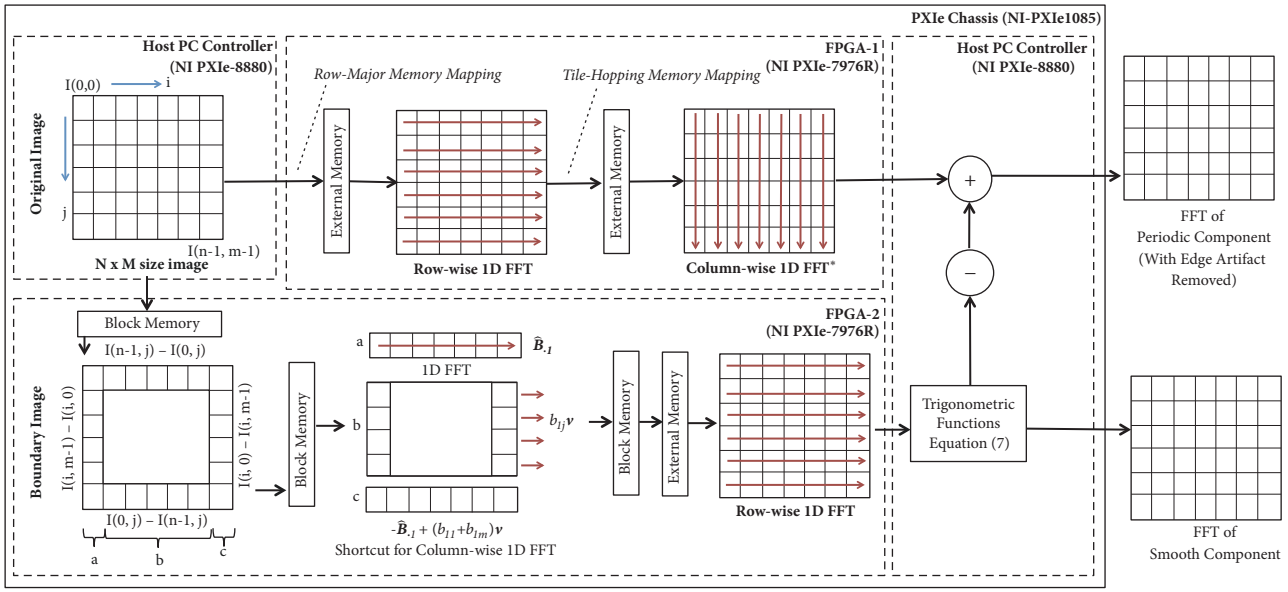


Figure 8: Functional block diagram of PXIe-based 2D FFT implementation with simultaneous edge artifact removal using optimized periodic plus smooth decomposition. The OPSD algorithm is split among two NI-7976R (Kintex-7) FPGA boards with 2GB external memory and a host PC connected over a high-bandwidth bus. The image is streamed from the PC controller to FPGA 1 and FPGA 2. FPGA 1 calculates the row-by-row 1D FFT followed by column-by-column 1D FFT with intermediate tile-hopping memory mapping and sends the result back to the host PC. FPGA 2 receives the image, calculates the boundary image, and proceeds to calculate the 1D FFT column-by-column FFT using the shortcut presented in (16) followed by row-by-row 1D FFTs and the result is sent back to the host PC.

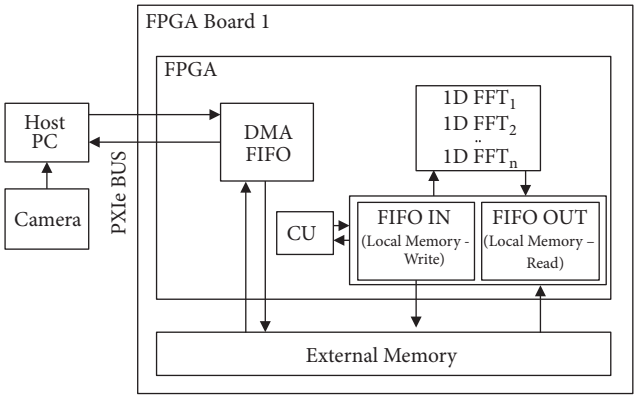
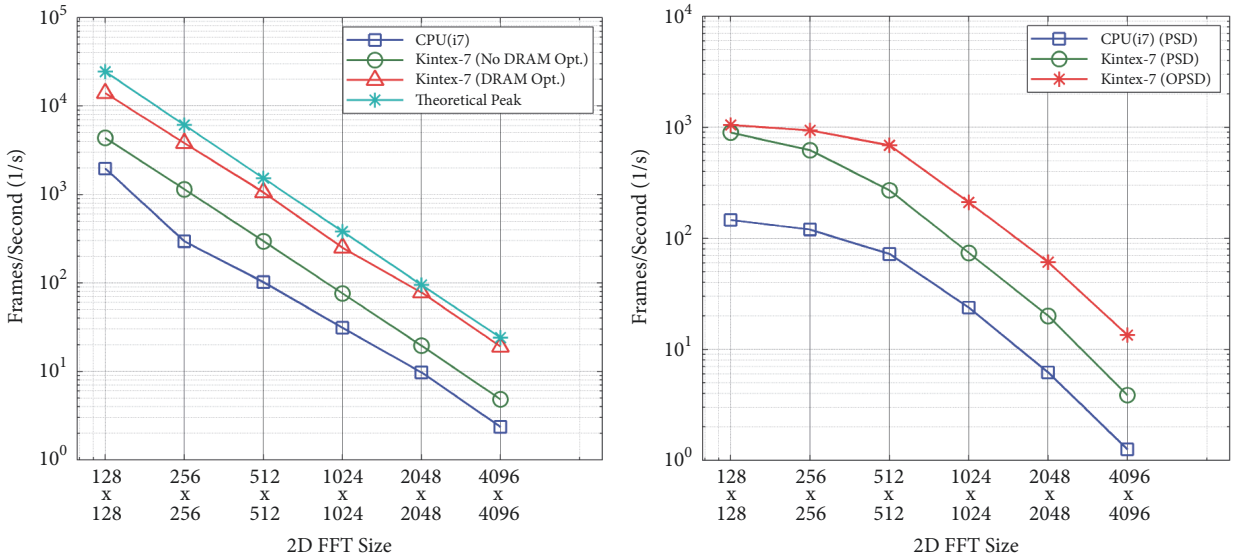


Figure 9: Block diagram of 2D FFT showing data transfer between external memory and local memory scheduled via a Control Unit (CU).

the host PC. If the image is being streamed directly from an imaging device which scans and provides random or nonlinear sequence of rows, it is necessary to store a frame of the image in a buffer. This can also be accomplished by streaming the image flow from the host PC or using a smart camera which can delay image delivery by a single frame. Local memory shown in Figure 9 is used to buffer data between external memory and 1D FFT cores. This memory is divided into read and write components and is implemented using FPGA slices. Block RAM (BRAM) is used for temporary storage of vectors required for calculating the 2D FFT of the boundary image (in the case of FPGA 2). The Control Unit (CU) organizes scheduling for transferring data between local and external memory. CU is based on LabVIEW's existing memory controller and our memory mapping scheme presented in Section 5.

Step B was accomplished using standard LabVIEW FPGA HLS tools for programming (5) using the graphical programming environment. In step C the 2D FFT of the boundary image needs to be calculated by row and column decomposition. However, as shown mathematically in the previous section, the initial row-wise FFTs can be calculated by computing the 1D FFT of the first (boundary) vector and the FFTs of reaming vectors can be computed by appropriate scaling of this vector.

We need the boundary column vector for 1D FFT calculation of the first and last columns. We also need the boundary row vector for appropriate scaling of $\hat{\boldsymbol{v}}$ for the 1D FFT of every column between the first and last columns. Row and column vectors of the boundary image are stored in block RAM (BRAM). Figure 8 shows a functional block diagram of the overall 2D FFT with optimized PSD process. Steps D and E are performed on the host PC to minimize memory clashes and to access the periodic and smooth components of each frame as they become available. 86% of resources are used on FPGA 1 and 41% resources are used on FPGA 2. The resource utilization is reported according to LabVIEW FPGA synthesis and compilation experiments. It should be noted that part of the reason for high resource utilization is because of using LabVIEW FPGA high-level synthesis tools as well as Xilinx LogiCORE tools. Using standardized tools makes it harder to optimize for resource utilization. The current implementation was optimized for performance in terms of run time and energy consumption.



(a) 2D FFT performance: DRAM tile-hopping

(b) 2D FFT with EAR performance: PSD versus OPSD

FIGURE 10: Performance evaluation in terms of frames per second for (a) 2D FFTs with tile-hopping memory pattern. (b) 2D FFTs with edge artifact removal (EAR) using OPSD. The performance evaluation shows the significance of the two optimizations proposed. Both axes are on a log scale.

6.3. Performance Evaluation. The overall performance of the system was evaluated using the setup presented in Figure 9. The data was streamed from the host PC; in certain cases high frame rate videos as well as direct camera input were streamed from the host. All results presented are for 16-bit fixed-point precision. Figure 10(a) presents the effectiveness of our proposed tile-hopping memory mapping scheme. It clearly shows the effectiveness of our proposed memory mapping since it is closer to the theoretical peak performance. Figure 10(b) presents the overall results comparing PSD and OPSD and demonstrating the effectiveness of our proposed optimization. PSD was also implemented on the same platform but the optimization presented in Section 4 was not used. This rendered the serial portion of the algorithm to be the bottleneck which reduced overall performance. Table 2 shows a comparison of our implementation in contrast to recent 2D FFT FPGA implementation approaches and shows that we achieve a better performance even with simultaneous edge artifact removal. Although, our implementation is tested with 16-bit fixed-point precision which limits the accuracy of the transform, the precision may be sufficient for a variety of speed critical applications where alternative edge artifact removal methods (e.g., filtering) may decrease overall system performance. Although, the dynamic range of fixed-point data is smaller than floating-point data, which can lead to errors in the 2D FFT; for certain applications it is more important to have an artifact-free transform as compared to a highly accurate transform.

6.4. Energy Evaluation. The overall energy consumption of the custom computing system depends on (1) power performance of the system components and (2) throughput disparity. Throughput disparity results in idle time for at least one of the components and lowers the overall system throughput. A throughput-optimized system minimizes instances where certain components of the architecture are idle. The proposed optimizations in Sections 4 and 5 clearly reduce throughput disparity and minimize the idle time of the system. Thus, besides causing delays due to significant overhead, standard column-wise DRAM access also contributes to the overall energy consumption. This is not only due to high count of DRAM row charges but also because of energy consumed by the FPGA in idle state. Ideally, maximizing the DRAM bandwidth limits the amount of energy consumption. Proposed *"tile-hopping"* memory mapping scheme improves the DRAM bandwidth as seen in Figure 10 and hence reduces the overall energy consumption. The same is true for the proposed OPSD method where reduced DRAM access and 1D FFT invocations lead to reduced energy consumption. In this section we analyze the amount of improvement in energy consumption based on the proposed optimizations.

We estimate the DRAM power consumption for both the baseline (standard, strided) and the optimized (*"tile-hopping"*) memory access using MICRON DRAM power calculator. The energy is calculated in *nJ* for each read, i.e., energy per read. This is accomplished by calculating the run time for a specific 2D FFT and estimating the amount of energy consumed by the DRAM power calculator. Table 4 depicts the DRAM energy consumption for 2D FFTs before and after the proposed *"tile-hopping"* optimization for column-wise DRAM access. As mentioned earlier, row-wise DRAM access is fast and row buffer size data can be accessed by a single row activation. According to Table 4 the energy required for DRAM access is reduced by 42.7%, 48.8%, and 52.9% for 1024×1024, 2048×2048, and 4096×4096 size 2D FFTs, respectively.

Table 2: Comparison of OPSD[1] 2D FFT with regular RCD-based implementation.

Platform	SEAR[2] Yes/No	Precision bits	RT[3] (ms) 512 × 512	1024 × 1024
Kintex 7, 28nm (ours)	**Yes**	**16 (fixed)**	**1.5**	**4.8**
Kintex 7, 28nm (ours)	**No**	**16 (fixed)**	**0.9**	**4.1**
Kintex 7, 28mm [8]	Yes	16 (fixed)	32.4	116.7
Stratix IV [5]	No	64 (double)	-	6.1
Virtex-5-BEE3, 65nm[14]	No	32 (single)	24.9	102.6
Virtex-E, 180nm [21]	No	16 (fixed)	28.6	76.9
ASIC, 180nm	No	32 (single)	21.0	-

[1] Optimized periodic + smooth decomposition (OPSD)
[2] Simultaneous edge artifact removal
[3] Runtime (ms).

Table 3: DRAM energy consumption baseline vs tile-hopping.

	1024 × 1024 *nJoule*	2048 × 2048 *nJoule*	4096 × 4096 *nJoule*
EPR* CW° Read (Baseline)	4.46	5.77	7.12
EPR CW Read *Tile-Hopping* **(Proposed)**	2.54	2.95	3.36
Reduction (%)	**42.7%**	**48.8%**	**52.9%**

*Energy per read (EPR).
°Column-wise memory access (CW).

Table 4: 2D FFT + EAR energy consumption baseline vs optimized (OPSD + tile hopping).

	1024 × 1024 *nJoule*	2048 × 2048 *nJoule*	4096 × 4096 *nJoule*
EPP† 2D FFT + EAR Baseline	36.92	41.25	48.35
2D FFT+EAR (Opt.) *Tile-Hopping + OPSD* **(Proposed)**	15.88	17.06	18.11
Improvement	2.3×	2.4×	2.8×

†Energy per point (EPP).

The metric used to compare the overall energy optimization achieved for 2D FFTs with EAR is energy per point, i.e., the amount of average energy required to compute the 2D FFT of a single point in an image with simultaneous edge artifact removal. This was achieved by calculating the energy consumed by Xilinx LogiCORE IP for 1D FFTs, the DRAM, and the edge artifact removal part separately. The estimated energy calculated does not include energy consumed by the PXIe chassis and the host PC. Essentially, the FPGA-based architecture presented here could be used without the host controller. The energy consumption incorporates dynamic as well as static power. The overall energy consumption per point is reduced by 56.9%, 58.6%, and 62% for calculating 1024 × 1024, 2048 × 2048, and 4096 × 4096 size 2D FFTs with EAR, respectively.

7. Application: Filtered Back-Projection for Tomography

In order to further demonstrate the effectiveness of our implementation, we use the created 2D FFT module as an accelerator for reducing the run time for filtered back-projection (FBP). FBP is a fundamental analytical tomographic image reconstruction method. In depth details regarding the basic FBP algorithm have been left out for brevity, but can be found in [28, 29]. The method can be used to reconstruct primitive 3D tomograms from 2D data, which can then be used as a basis for more complex regularization-based methods such as [30–32]. The algorithmic flow is based on the Fourier slice theorem; i.e., 2D Fourier transforms of projections are an angular component of the 3D Fourier

Table 5: Comparing filtered back-projection (FBP) runtime (as an application for using the proposed 2D FFT with simultaneous EAR).

3D Density	CPU (i7) Sec	FPGA + Host PC (i7) Sec
128 × 128 × 128	21.3 sec	19.5 sec
256 × 256 × 256	47.5 sec	42.4 sec
512 × 512 × 512	94.8 sec	81.3 sec
1024 × 1024 × 1024	322.3 sec	275.3 sec
2048 × 2048 × 2048	1687.7 sec	1364.4 sec
4096 × 4096 × 4096	16463.1 sec	12599.4 sec

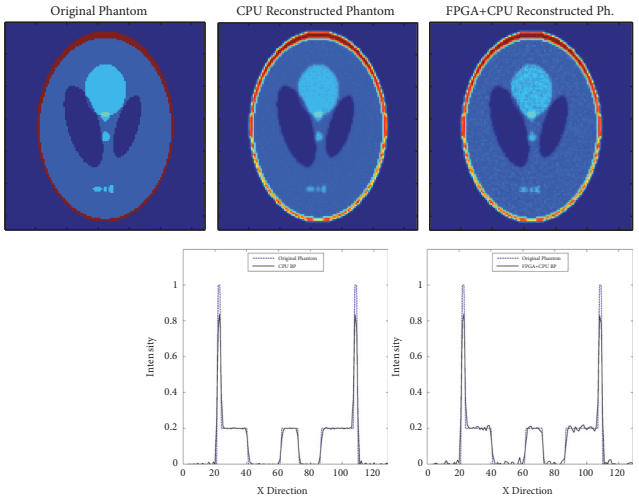


Figure 11: Figure showing a thin slice of filtered back-projection results by reconstructing a 128 × 128 × 128 Shepp-Logan phantom. The 3D density was reconstructed from 180 equally spaced simulated projections using standard linearly interpolated FBP and using a Ram-Lak filter. It can be seen that the results from the FPGA + CPU solution have some errors; this is due to the fact that the 2D FFT of each projection is less accurate. The results are good enough to be used as a basis for further optimization based refinement methods.

transform of the 3D reconstructed volume. Our 2D FFT accelerator was used to calculate the 2D FFTs of the projections as well as for initial stages of the 3D FFT which was then completed on the host PC. Similar to the 2D FFT, the 3D FFT is separable and can be divided into 2D FFTs and 1D FFTs. The results have been shown in Table 5. It can be seen that the improvement for smaller size densities is not significant because their FFTs are quite fast on general-purpose CPUs. However, for larger densities the FFT accelerator can give a significant improvement. If the remaining components are also implemented on an FPGA, significant speed increase can be achieved. Results of a thin slice from a 3D simulated Shepp-Logan [33] phantom have been shown in Figure 11. It can be seen that the results from the hardware accelerated FBP are of slightly lower quality. This is due to the fact that our 2D FFT implementation is less accurate (16 bit, fixed-point) as compared to the CPU-based implementation (FFTW, double-precision floating). The accelerated FBP was also tested with real Electron Tomography (ET) data.

8. Conclusion

2D FFTs often become a major bottleneck for high-performance imaging and vision systems. The inherent computational complexity of the 2D FFT kernel is further enhanced if effective removal (using PSD) of spurious artifacts introduced by the nonperiodic nature of real-life images is taken into account. We developed and implemented an FPGA-based design for calculating high-throughput 2D DFTs with simultaneous edge artifact removal. Our approach is based on a PSD algorithm that splits the frequency domain of a 2D image into a smooth component which contains the high-frequency, cross-shaped artifacts and can be subtracted from the 2D DFT of the original image to obtain a periodic component that is artifact-free. Since this approach calculates two 2D DFTs simultaneously, external memory addressing and repeated 1D FFT invocations become problematic. To solve this problem we optimized the original PSD algorithm to reduce the number of DFT samples to be computed and DRAM access. Moreover, to reduce strided access from the DRAM during column-wise reads we presented and analyzed *"tile-hopping"*, a memory mapping scheme which reduces the number of DRAM row activations when reading a single column of data. This memory mapping scheme is general and may be used for a variety of other applications. We demonstrate that *"tile-hopping"* memory mapping can reduce the DRAM energy consumption by 52.9%. Moreover, we show that the proposed optimizations lead to 2.8× less energy consumption for the overall 2D FFT with EAR architecture.

Our methods were tested using extensive synthesis and benchmarking using a Xilinx Kintex 7 FPGA communicating with a host PC on a high-speed PXIe bus. Our system is expandable to support several FPGAs and can be adapted to various large-scale computer vision and biomedical applications. Despite decomposing the image into periodic and smooth frequency components, our design requires less run time, compared to traditional FPGA-based 2D DFT implementation approaches and can be used for a variety of highly demanding applications. One such application, filtered back-projection, was accelerated using the proposed implementation to achieve better results specifically for larger size raw tomographic data.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by Japanese Government OIST Subsidy for Operations (Ulf Skoglund) Grant no. 5020S7010020. Faisal Mahmood and Märt Toots were additionally supported by the OIST Ph.D. Fellowship. The authors would like to thank National Instruments Research for their technical support during the design process. The authors would also like to thank Dr. Steven D. Aird for language assistance and Shizuka Kuda for logistical arrangements.

Supplementary Materials

File: 2D FFT-EAR-Demo.mp4: video demonstrating the FPGA output of 2D FFT with edge artifact removal. *(Supplementary Materials)*

References

[1] R. N. Bracewell, *The fourier transform and iis applications*, vol. 5, New York, NY, USA, 1965.

[2] *Theory and application of digital signal processing*, vol. 1, Prentice-Hall, Inc, Englewood Cliffs, NJ, USA, 1975.

[3] R. A. Brooks and G. Di Chiro, "Theory of image reconstruction in computed tomography," *Radiology*, vol. 117, no. 3 I, pp. 561–572, 1975.

[4] L. Moisan, "Periodic plus smooth image decomposition," *Journal of Mathematical Imaging and Vision*, vol. 39, no. 2, pp. 161–179, 2011.

[5] B. Akin, P. A. Milder, F. Franchetti, and J. C. Hoe, "Memory bandwidth efficient two-dimensional fast Fourier transform algorithm and implementation for large problem sizes," in *Proceedings of the 20th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012*, pp. 188–191, Canada, May 2012.

[6] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[7] H. Kee, N. Petersen, J. Kornerup, and S. S. Bhattacharyya, "Systematic generation of FPGA-based FFT implementations," in *Proceedings of the 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pp. 1413–1416, USA, April 2008.

[8] F. Mahmood, M. Toots, L.-G. Ofverstedt, and U. Skoglund, "2D Discrete Fourier Transform with simultaneous edge artifact removal for real-Time applications," in *Proceedings of the International Conference on Field Programmable Technology, FPT 2015*, pp. 236–239, New Zealand, December 2015.

[9] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, pp. 1381–1384, IEEE, May 1998.

[10] M. Püschel, J. M. F. Moura, J. R. Johnson et al., "SPIRAL: code generation for DSP transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232–273, 2005.

[11] E. Wang, Q. Zhang, B. Shen et al., *High-Performance Computing on the Intel Xeon Phi*, Springer International Publishing, 2014.

[12] B. Akin, F. Franchetti, and J. C. Hoe, "FFTS with near-optimal memory access through block data layouts," in *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2014*, pp. 3898–3902, Italy, May 2014.

[13] B. Akin, F. Franchetti, and J. C. Hoe, "Understanding the design space of DRAM-optimized hardware FFT accelerators," in *Proceedings of the 25th IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2014*, pp. 248–255, Switzerland, June 2014.

[14] C.-L. Yu, K. Irick, C. Chakrabarti, and V. Narayanan, "Multidimensional DFT IP generator for FPGA platforms," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 755–764, 2011.

[15] D. He and Q. Sun, "A practical print-scan resilient watermarking scheme," in *Proceedings of the IEEE International Conference on Image Processing 2005, ICIP 2005*, pp. 257–260, Italy, September 2005.

[16] D. G. Bailey, *Design for embedded image processing on FPGAs*, John Wiley Sons, 2011.

[17] B. G. Batchelor, "Implementing Machine Vision Systems Using FPGAs," in *Machine Vision Handbook*, 1136, p. 1103, Springer, London, UK, 2012.

[18] T. Lenart, M. Gustafsson, and V. Öwall, "A hardware acceleration platform for digital holographic imaging," *Journal of Signal Processing Systems*, vol. 52, no. 3, pp. 297–311, 2008.

[19] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 453–464, 2008.

[20] Z. Qiuling, B. Akin, H. E. Sumbul et al., "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *Proceedings of the IEEE International 3D Systems Integration Conference*, pp. 1–7, October 2013.

[21] I. S. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," pp. 283–296.

[22] T. Dillon, "Two Virtex-II FPGAs deliver fastest, cheapest, best high-performance image processing system," " *Xilinx Xcell Journal*, vol. 41, pp. 70–73, 2001.

[23] A. Hast, "Robust and invariant phase based local feature matching," in *Proceedings of the 22nd International Conference on Pattern Recognition, ICPR 2014*, pp. 809–814, Sweden, August 2014.

[24] B. Galerne, Y. Gousseau, and J.-M. Morel, "Random phase textures: theory and synthesis," *IEEE Transactions on Image Processing*, vol. 20, no. 1, pp. 257–267, 2011.

[25] R. Hovden, Y. Jiang, H. L. Xin, and L. F. Kourkoutis, "Periodic Artifact Reduction in Fourier Transforms of Full Field Atomic Resolution Images," *Microscopy and Microanalysis*, vol. 21, no. 2, pp. 436–441, 2014.

[26] D. G. Bailey, "The advantages and limitations of high level synthesis for FPGA based image processing," in *Proceedings of the the 9th International Conference*, pp. 134–139, Seville, Spain, September 2015.

[27] W. Wang, B. Duan, C. Zhang, P. Zhang, and N. Sun, "Accelerating 2D FFT with non-power-of-two problem size on FPGA," in *Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2010*, pp. 208–213, Mexico, December 2010.

[28] F. Natterer, *The Mathematics of Computerized Tomography*, John Wiley & Sons, 1986.

[29] R. A. Crowther, D. J. DeRosier, and A. Klug, "The Reconstruction of a Three-Dimensional Structure from Projections and its Application to Electron Microscopy," *Proceedings of the Royal Society A Mathematical, Physical and Engineering Sciences*, vol. 317, no. 1530, pp. 319–340, 1970.

[30] U. Skoglund, L.-G. Öfverstedt, R. M. Burnett, and G. Bricogne, "Maximum-entropy three-dimensional reconstruction with deconvolution of the contrast transfer function: A test application with adenovirus," *Journal of Structural Biology*, vol. 117, no. 3, pp. 173–188, 1996.

[31] F. Mahmood, N. Shahid, P. Vandergheynst, and U. Skoglund, "Graph-based sinogram denoising for tomographic reconstructions," in *Proceedings of the 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2016*, pp. 3961–3964, USA, August 2016.

[32] F. Mahmood, N. Shahid, U. Skoglund, and P. Vandergheynst, "Adaptive Graph-Based Total Variation for Tomographic Reconstructions," *IEEE Signal Processing Letters*, vol. 25, no. 5, pp. 700–704, 2018.

[33] L. A. Shepp and B. F. Logan Jr., "The Fourier reconstruction of a head section," *IEEE Transactions on Nuclear Science*, vol. 21, no. 3, pp. 21–43, 1974.