EFFICIENT IMAGE RECONSTRUCTION USING PARTIAL 2D FOURIER TRANSFORM[†]

L. Deng, C.-L. Yu, C. Chakrabarti

Dept. of Electrical Engineering
Arizona State University
email: {ldeng2, chi-li.yu, chaitali}@asu.edu

J. Kim, V. Narayanan

Dept. of Computer Science and Engineering
Pennsylvania State University
email: {jskim, vijay}@cse.psu.edu

ABSTRACT

In this paper we present an efficient way of doing image reconstruction using the 2D Discrete Fourier transform (DFT). We exploit the fact that in the frequency domain, information is concentrated in certain regions. Consequently, it is sufficient to compute partial 2D Fourier transform where only $m \times m$ elements of an $N \times N$ image are nonzero. Compared with the traditional row-column (RC) decomposition algorithm, the proposed algorithm enables us to reconstruct images with significantly smaller computation complexity at the expense of mild degradation in quality. We also describe the implementation of the new reconstruction algorithm on a Xilinx Virtex-II Pro-100 FPGA. For 512×512 natural and aerial images, this implementation results in 68% reduction in the number of memory accesses and 76% reduction in the total computation time compared to the RC method.

.

Keywords: Discrete Fourier Transform; image reconstruction; FPGA; row-column decomposition; two dimensional decomposition

1. INTRODUCTION

The Discrete Fourier Transform (DFT) and its inverse are important image processing tools. Fast implementations of these transforms[1] have made them popular in applications in image analysis, filtering, reconstruction and compression. In this paper, we focus on the application of fast two-dimensional (2D) (inverse) DFTs in efficient image reconstruction.

Typically, 2D DFTs have been implemented using the Row Column (RC) decomposition technique [2]. While its hardware implementation is simple, its latency is high because the column-wise 1D DFTs cannot be executed unless all the row-wise 1D DFTs are finished. Also if the data size is large, it cannot be efficiently mapped onto embedded systems where resources and memory are limited.

In some applications where part of the input data has less significance, these inputs are represented by 0's and FFT Pruning techniques [3] [4] can be applied. Examples include comb spectrum evaluation [5], the transition-region response in a filter design [3], etc. The data flow of the pruned structure is derived from the normal FFT structure by keeping only the branches corresponding to non-zero inputs. While the number of computations in pruned FFT is reduced, the resulting structure is quite irregular and highly dependent on the number and positions of the nonzero elements. Consequently, the hardware implementation is unwieldy and not scalable [5].

We introduce in this paper a new technique for efficient image reconstruction that is regular and scalable. It exploits the fact that in Fourier representations of 'natural' and 'aerial' images, information is concentrated only in the four corners. This feature is exploited in the development of a new two-dimensional (2D) decomposition algorithm for inverse DFT that reconstructs the image using a small subset of Fourier coefficients. We refer to this algorithm as partial 2D DFT. This algorithm is inherently parallel and requires a small number of memory accesses compared to the conventional RC decomposition.

The proposed algorithm has been implemented on an FPGA platform consisting of a Xilinx Virtex-II Pro -100 FPGA and a DDR SDRAM. It can process 512×512 sized Fourier image with 128×128 nonzero coefficients in $295,494$ clock cycles. While this architecture is highly scalable, its current implementation is limited by the memory/bus bandwidth.

The rest of the paper is organized as follows. In Section 2, we introduce the basics of the decomposition algorithm for 1D and 2D DFT, followed by the partial DFT decomposition. Section 3 describes the application of the 2D decomposition algorithm for efficient image reconstruction. Section 4 describes the FPGA implementation of the proposed image reconstruction technique. Experimental results illustrating the tradeoffs between image quality and number of nonzero coefficients are also presented. Section 5 concludes this paper.

2. DECOMPOSITION ALGORITHM FOR PARTIAL DISCRETE FOURIER TRANSFORM

In this section, we briefly discuss the basics of DFT and its decomposition when the number of zero elements is significant. We refer to this as *partial* one-dimensional DFT (1D DFT). Then we present a two-dimensional decomposition algorithm

[†] This work is supported by a grant from DARPA W911NF-05-1-0248

for efficient computation of partial 2D DFT.

2.1. Decomposition of Partial 1D DFT

The DFT of an N-point discrete-time complex sequence $x(n)$, indexed by $n = 0, 1, \ldots N-1$, is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk}, k = 0, 1, \ldots, N-1 \quad (1)$$

where $W_N = e^{-j2\pi/N}$. The computation complexity of a N-point DFT is $O(N^2)$. The Fast Fourier Transform (FFT) algorithm reduces the computation complexity to $O(N \log_2 N)$ [1].

Eqn.(1) can be represented in the matrix-vector multiplication form as

$$X_{N\times 1} = F_N \cdot x_{N\times 1} \quad (2)$$

where F_N is an $N \times N$ matrix whose (n,k) entry is W_N^{nk}. F_N can be decomposed into a product of sparse factor matrices as shown in [6] [7], and the algebraic expression using Kronecker products is given in [8] as:

$$F_N = (F_p \otimes I_m)\tilde{D}_N(I_p \otimes F_m)P_{N,p} \quad (3)$$

where $N = p \cdot m$, I_m is a $m \times m$ identity matrix, $\tilde{D}_N$ is a diagonal matrix of twiddle factors, $\otimes$ is the Kronecker (or tensor) product, and $P_{N,p}$ denotes permutation with stride p.

$$\tilde{D}_N(j,j) = W_N^{(j \bmod m)\cdot\lfloor j/m \rfloor} \text{ for } j = 0, 1 \ldots N-1 \quad (4)$$

$$A_n \otimes B_m = [a_{k,l}B_m]_{0\le k,l<n} \text{ for } A = [a_{k,l}]_{0\le k,l<n} \quad (5)$$

Using the decomposition of F_N given in Eqn. (3), the 1D DFT in Eqn. (2) is rewritten as:

$$\tilde{X}_{p\times m} = F_p \cdot \tilde{x}_{p\times m} \odot \tilde{D}_{p\times m} \cdot F_m \quad (6)$$

where $\odot$ is the matrix-matrix dot (or direct) product. $\tilde{X}_{p\times m}$, $\tilde{x}_{p\times m}$ and $\tilde{D}_{p\times m}$ are permutations of $X_{N\times 1}$, $x_{N\times 1}$ and $\tilde{D}_{N\times 1}$ respectively. For $i = 0, 1, \ldots, p-1,\ j = 0, 1, \ldots, m-1$, the mappings are as follows:

$$\tilde{X}_{(i,j)} = X_{(i+j*p)},\ \ \tilde{x}_{(i,j)} = x_{(i*m+j)},\ \ \tilde{D}_{(i,j)} = W_N^{i*j} \quad (7)$$

.

For partial 1D DFT, assuming that only the first m out of N elements of vector $\underline{x}$ are non-zero, i.e. $\underline{x} = [\,x_0\ x_1\ \ldots x_{m-1}\ 0\ 0 \ldots\ 0]^T$, Eqn.(6) can be simplified as:

$$\tilde{X}_{p\times m} = \begin{pmatrix} x_0 & \cdots & x_{m-1} \\ \vdots & \ddots & \vdots \\ x_0 & \cdots & x_{m-1} \end{pmatrix}_{p\times m} \odot \tilde{D}_{p\times m} \cdot F_m \quad (8)$$

The first matrix on the righthand side of Eqn.(8) is nothing but $F_p \cdot \tilde{x}_{p\times m}$ from Eqn. (6). It is obtained by duplicating the m non-zero elements p times. Thus 1D partial DFT $\tilde{X}_{p\times m}$ can be obtained by first computing the dot-product of this matrix with scaling matrix $\tilde{D}_{p\times m}$, and then applying DFT of length m on each row. The total number of complex multiplications is then no more than ($N + \frac{N}{2}\log_2 m$).

2.2. 2D Decomposition of Partial 2D DFT

If the input data is of size $N \times N$, then 2D DFT is defined as

$$Y(k_1,k_2) = \sum_{i_1=0}^{N-1}\sum_{i_2=0}^{N-1} x(i_1,i_2) \cdot W_N^{k_1 i_1 + k_2 i_2} \quad (9)$$

where $k_1, k_2 = 0, 1, \ldots, N-1$.

The traditional Row-Column (RC) decomposition algorithm computes N 1D FFTs (one for each row), followed by N 1D FFTs (one for each column). The complexity of the RC algorithm is $O(N^2 \log_2 N)$.

The 2D DFT can be represented in matrix form as follows:

$$Y_{N\times N} = F_N \cdot X_{N\times N} \cdot F_N \quad (10)$$

where X and Y are of size $N \times N$, and F_N is the twiddle factor matrix. Applying the 1D decomposition in Eqn.(6) along both rows and columns, the 2D DFT of $N \times N$ points is written as:

$$\tilde{Y}_{N\times N} = (I_p \otimes F_m)\tilde{D}_N(F_p \otimes I_m)X(F_p \otimes I_m)\tilde{D}_N(I_p \otimes F_m) \quad (11)$$

The final output Y is obtained by applying both row- and column-wise bit-reverse permutations on $\tilde{Y}$.

$$Y_{N\times N} = P_{N,p}\tilde{Y}_{N\times N}P_{N,p} \quad (12)$$

In the computation of partial 2D DFT, we assume that only a subblock of $m \times m$ elements in input X are non-zero, i.e. only $A_{m\times m}$ is nonzero:

$$X = \begin{pmatrix} A_{m\times m} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{N\times N}$$

Then Eqn.(11) can be simplified as:

$$\tilde{Y} = (I_p \otimes F_m)\tilde{D}_N \begin{pmatrix} A & A & \cdots & A \\ A & A & \cdots & A \\ \vdots & \vdots & \ddots & \vdots \\ A & A & \cdots & A \end{pmatrix} \tilde{D}_N(I_p \otimes F_m) \quad (13)$$

Since $\tilde{D}_N$ is a diagonal matrix whose elements are computed using Eqn.(4), we can further simplify Eqn.(13) into:

$$\tilde{Y} = \underbrace{(I_p \otimes F_m)(\underbrace{\underbrace{\begin{pmatrix} A & A & \cdots & A \\ A & A & \cdots & A \\ \vdots & \vdots & \ddots & \vdots \\ A & A & \cdots & A \end{pmatrix}}_{step1} \odot D'_N)}_{step2}(I_p \otimes F_m)}_{step3} \quad (14)$$

where D'_N is a $N\times N$ full matrix whose elements are $D'_N(i,j) = \tilde{D}_N(i,i) \cdot \tilde{D}_N(j,j)$, for $i, j = 0, 1 \ldots N-1$.

Thus partial 2D DFT computation can be broken down into 4 steps:

Step 1. Non-zero subblock duplication:

Input X is partitioned into small sub-blocks of size $m \times m$, where only the top-left subblock contains non-zero elements. The non-zero subblock is then simply duplicated as shown in Eqn.(14).

For each subblock, do Steps 2, 3 and 4.

Step 2. Twiddle factor scaling:

Each element is multiplied with a specific twiddle factor, so there are m^2 complex multiplications.

Step 3. Local 2D FFT:

Local 2D FFT is computed on each subblock and requires $O(m^2 \log_2 m)$ complex multiplications.

Step 4. Output permutation:

The desired output is obtained by permuting the results generated in each of the sub-blocks, as shown in Eqn.(12). The permutation is a combination of row-wise and column-wise bit-reversals.

Fig.1 demonstrates the flow graph involved in the computation of partial 2D DFT. The total number of complex multiplications is $\frac{N^2}{m^2}(m^2 + m^2 \log m) = N^2 + N^2 \log m$.

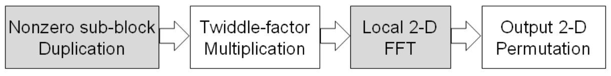
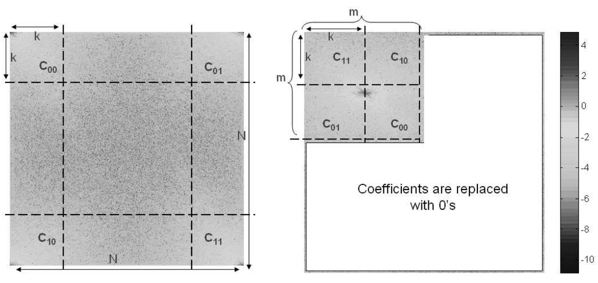


Fig. 1. The flow graph of partial 2D DFT decomposition

3. APPLICATION TO IMAGE RECONSTRUCTION

In Fourier representation of natural and aerial images, the bulk of the information is concentrated in the four corners. The partial 2D DFT algorithm described in Section 2.2 can be used for speeding up the image reconstruction. Thus the pixel image can be generated using only a subset of the coefficients with only a small loss in quality.



(a) Original (b) After frequency shift

Fig. 2. Fourier images of 'Lena': original and frequency shifted

Fig.2(a) describes the Fourier image of 'Lena' [9]. Note that most of the information is stored in the four corners of the Fourier image. Let the corners be represented by $k \times k$ sub-blocks. In Fig.2(a), $C_{00}, C_{01}, C_{10}, C_{11}$ are the sub-blocks in the four corners. We apply the frequency shift technique [5], and shift sub-blocks by an amount k first in the horizontal direction, and then in the vertical direction. The coefficients in the four corners are then mapped to the upper left corner as shown in Fig.2(b). The upper left corner now contains $2k \times 2k$ nonzero coefficients and all other coefficients are replaced by 0's. Even though the new Fourier image contains only a small number of nonzero coefficients, it can be used to generate a fairly good quality image as we will demonstrate in the subsequent sections.

Image reconstruction uses inverse DFT. Since the forward and inverse DFT are related, the partial 2D DFT algorithm described in Section 2.2 is equally applicable to inverse DFT. Let $Y_{N \times N}$ be the Fourier image of size $N \times N$, then:

$$DFT^{-1}(Y_{N \times N}) = \frac{1}{N^2} DFT(Y_{N \times N}^{\dagger})^{\dagger} \tag{15}$$

Here † means the conjugate transpose.

Our procedure is described below.

Step 1. Frequency-domain data transposition and scaling:

The coefficients in the Fourier image are represented in higher precision compared to the pixels in the original image. For efficient hardware implementation, the Fourier coefficients are represented in fixed point format. We first scale the frequency-domain matrix Y by its size N^2 (see Eqn.(15)), and then limit each element (both real and imaginary parts) to be in the range $(-128, 128)$. Our experimental data shows that representing scaled data (real and imaginary parts) in fixed point format (16,8) with 8 integer bits (including 1 sign bit) and 8 fractional bits still maintains good accuracy.

The conjugate transpose † in Eqn.(15) can simply be done by switching the row and column addresses and reversing the signs of the imaginary values when transferring data between FPGA and external memory.

Step 2. Thresholding to determine the subblock size:

As described earlier, Fourier representation of 'natural' and 'aerial' images typically contains four regions of high information content in the four corners. In this step we find the value of k (corresponding to the subblock of size $k \times k$ in each corner) which contains adequate information to reconstruct the image. We accumulate the magnitude of the Fourier coefficients in 'zigzag' order (similar to JPEG) for the subblock in the upper left corner of the Fourier image. The accumulated results of groups of 10 coefficients are shown in Fig.3.

For the example shown in Fig.3, we see that the slope decreases significantly after position 400, and so we can pick position 600 as the threshold value. Since each accumulation contains 10 elements, we can decide the value k by:

$$\frac{k(k-1)}{2} \leq 6000 \leq \frac{k(k+1)}{2} \tag{16}$$

For simplicity we assume that the image size is a power of 2. We also choose k to be a power of 2. In the 'Lena' case, the original image size is 512×512, and we choose $k = 128$.

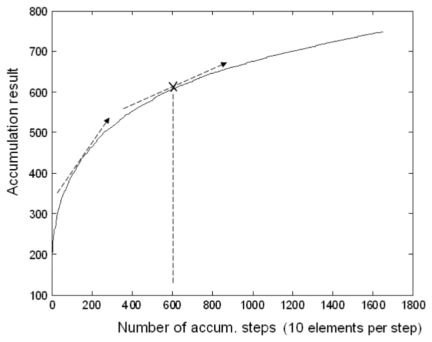


Fig. 3. Determining the size of the $k \times k$ subblock

Note that a larger value of threshold corresponds to a larger value of k.

Step 3. Frequency shifting of Fourier image:

The partial 2D DFT algorithm described in Section 2.2, especially Eqn. (13), requires that the nonzero elements be in the upper left corner of the input matrix. The 'frequency shift' technique allow us to shift subblocks in the Fourier image so that they are grouped in the upper left corner of the image, as shown in Fig.2.

Step 4. Efficient Partial 2D DFT computation:

After 'shifting', we only keep the data in the upper left corner as shown in Fig.2(b), and replace all other coeffcients with 0's. Note that here the new non-zero subblock after frequency shifting is of size $2k \times 2k$, and equals $m \times m$. Next we apply the 2D decomposition algorithm described in Section 2.2.

The inverse transform of a 'shifted' Fourier image has some phase difference compared with the 'original' one. For natural and aerial images, we can still construct an image of fairly good quality using only the magnitude component.

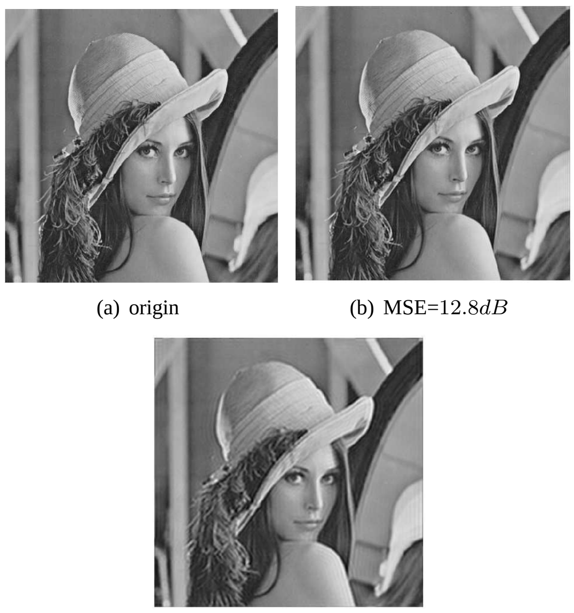
3.1. Tradeoffs between image quality and subblock size

As the size of the nonzero subblock increases, the quality of the reconstructed image improves, as expected. Sample images of 'Lena' are displayed in Fig.4. The images are of size 512×512 and are constructed in fixed point Matlab with 16 bit precision. We present the image quality in Mean-Square-Error (MSE). Let $x_{i,j}$ be the original pixel and $x'_{i,j}$ be the reconstructed pixel, then MSE (in dB) is calculated as:

$$10 \log_{10}\{\frac{1}{N^2}\sum_{i,j}(x_{i,j} - x'_{i,j})^2\}$$

3.2. Computation and Communication Efficiency

We compare our technique, based on partial 2D DFT decomposition, with the conventional RC decomposition and pruning FFT, with respect to number of complex multiplications and number of memory accesses. Here, the initial Fourier image is of size $N \times N$, the 'shifted' Fourier image only has a nonzero subblock of size $m \times m$. 'RC Full' means applying RC decomposition on the whole $N \times N$ Fourier image; 'RC pruning' applies pruning FFTs on the 'shifted' Fourier image.


(a) origin (b) MSE=$12.8dB$

(c) MSE=$18.5dB$

Fig. 4. Image 'Lena': (a) original (b) constructed from $1/4$ of Fourier image (c) constructed from $1/16$ of Fourier image

Table 1. Computation and Memory Access Comparison

Algorithm	# Multiplications	# Memory Accesses
2D Decomp.	$N^2 \log_2 2m$	$N^2 + m^2$
RC(Full)	$N^2 \log_2 N$	$4N^2$
RC(Pruning)	$(N+m)N \log_2 m$	$N^2 + m^2 + 2Nm$

From Table 1 we can see that 'RC Full' has the highest number of computations and memory accesses. Unless $m \ll N$, 'RC pruning' has no advantage w.r.t computation complexity compared with the proposed partial 2D DFT algorithm but always consumes more memory accesses.

4. HARDWARE ARCHITECTURE AND FPGA IMPLEMENTATION

The proposed 2D partial algorithm is mapped onto a platform consisting of a DDR-SDRAM and a Xilinx-II Pro FPGA, as shown in Fig.5. The original $N \times N$ Fourier image is stored in the SDRAM, and the $m \times m$ 2D DFT engine is implemented with the Xilinx-II Pro FPGA.

The execution procedure is described as follows:

1. Read the $m \times m$ nonzero subblock from the external memory, and store it into the source memory of the 2D DFT

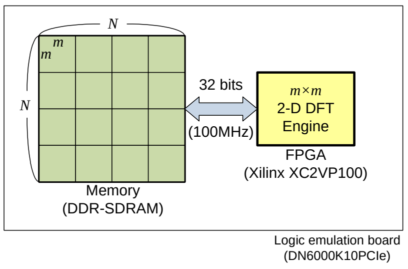


Fig. 5. Verification system for the proposed architecture

engine.

2. Use the 2D DFT engine to reconstruct an $m \times m$ subblock of the required image, and write it back to the main memory.

3. Repeat Step 2 N^2/m^2 times (corresponding to the N^2/m^2 subblocks) till the whole image is reconstructed.

The detailed architecture of the $m \times m$ 2D DFT engine is depicted in Fig.6. It contains two m-point 1D FFT Xilinx IPs[10], which are used to compute row- and column-wise 1D FFTs respectively. Each data is multiplied by a twiddle factor using a complex multiplier. A ROM table is used to generate the twiddle factors. Note that the size of the table is only $\frac{N}{8}$, because we can generate twiddle factors of range [0, 2π] by using only the coefficients within [0, $\pi/4$]. After the row-wise 1D FFT, the data is stored in the transpose memory for column computation. At the output, the magnitude of each pixel is computed by a CORDIC unit. The permutations are performed when the quantized image pixels are written back to the external main memory.

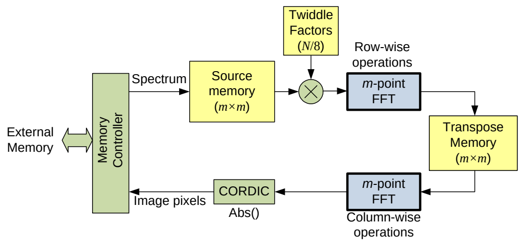


Fig. 6. Internal architecture of 2D DFT engine

FPGA Verification Platform:

To verify our design, we adopt DINI Group's DN6000K10 -PCIe logic emulation board, which is equipped with Xilinx XC2VP -100 FPGAs and Micron's DDR-SDRAM modules. We only adopt one FPGA for the 2D DFT engine and one DDR-SDRAM module for storage of the image (see Fig.5).

We use Xilinx Core Generator to generate the m-point pipelined FFTs, dual-port memories, and the CORDIC-based magnitude computation unit. All units are pipelined and can achieve a throughput of 1 output/clock. These IP cores and memories are generated as Verilog-HDL files, which are synthesizable and can be ported to the FPGA.

In this design, m is set to be 128, and the data width per pixel is set to 16 bits for both real and imaginary components. The data width of the DDR-SDRAM is 32 bits so one Fourier coefficient can be read at a time. The final synthesis result is summarized in Table 2. The maximum clock rate for the 2D DFT engine is 158MHz. However, to communicate with the DDR-SDRAM whose clock rate is 100MHz, the 2D DFT engine is also set to 100MHz. As a result, the ideal throughput is 10^6 pixels/sec. In addition, the pipelined 2D DFT engine has a latency of 33,350 clocks (including reading the 128×128 subblock) to generate the first pixel. Thus for instance, the total computation time for a 512×512 image is 33,350 + 512×512= 295,494 clock cycles, i.e. 2,955 μs.

Table 2. FPGA resource and timing performance of the 128×128 2D DFT engine. (Data width: 16 bits)

Resources			Frequency	Latency
Slice	BRAM	MULT18	(MHz)	(clock cycles)
5933 (13%)	16 (3%)	24 (5%)	100 (max:158)	33,350

Minimized Data Access to the Memory:

From the execution procedure, we see that only m^2 ($\ll N^2$) data are read from the memory, and only N^2 data are written to the memory. Compared to the $2N^2$ reads and $2N^2$ writes of RC decomposition, it is obvious that the number of accesses to the external memory is greatly reduced. This not only shortens the communication time but also helps in reducing the power consumption significantly.

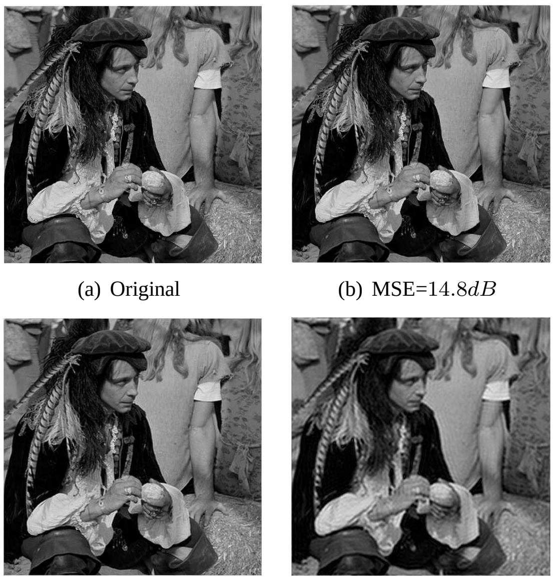
Scalability of the Architecture:

From Table 2, we see that the FPGA resources have only been partially utilized. This implies that more 2D DFT engines can be put onto the FPGA. Due to the high parallelism of the proposed algorithm, the performance of our system can potentially be accelerated k times if k 2D DFT engines are utilized. However, system performance is constrained by memory bandwidth. As mentioned above, we can only access one Fourier coefficient at a time in this verification platform. Under this constraint, it is meaningless to add more 2D DFT engines to the FPGA. If we can customize the verification board and connect more memory modules to the FPGA, the bottleneck can be eliminated. Then, our architecture can be further parallelized and the speed can be significantly increased.

Validation on the FPGA platform:

Next we demonstrate the Mean-Square-Error and timing performance of our procedure when implemented on the hardware platform. The images used in these experiments are from 'miscellaneous' and 'aerials' categories in USC-SIPI image database [11]. The data representation for the hardware implementation is fixed point format (16,8) with 8 integer bits (including 1 sign bit) and 8 fractional bits.

Fig.7 and Table 3 illustrate the quality degradation for different nonzero subblock sizes. Fig.7(b) shows that even when $1/4$ of the given Fourier image is used, the reconstructed im-



(a) Original (b) MSE=$14.8dB$

(c) MSE=$20.0dB$ (d) MSE=$23.5dB$

Fig. 7. Image ‘Man’:(a) original (b) constructed from $1/4$ of Fourier image (c) $1/16$ of Fourier image (d) $1/64$ of Fourier image.

age quality is still quite good compared with the original one. With further reduction (like when only $1/64$ of Fourier image is used), the image as shown in Fig.7(d) is still acceptable.

Table 3. MSE (dB) for reconstructed sample images

Size	Images	RC(Full)	m^2/N^2		
			$1/4$	$1/16$	$1/64$
512×512	Lena	0.2	12.8	18.5	22.5
	Girl (Elaine)	19.0	20.4	21.2	22.8
	Fishing boat	7.6	17.3	22.0	24.9
	Stream and bridge	0.5	20.8	24.7	27.0
$1K \times 1K$	Man	4.7	14.8	20.0	23.5
	Pentagon	20.9	22.5	24.0	25.2
	Airport	4.9	17.7	22.3	25.1
	Airplane(U-2)	4.9	13.7	15.1	15.9

Table 3 gives the MSE for 6 images. We see that the amount of MSE degradation varies from image to image. Also some images are more sensitive to subblock size. In many of the images, the MSE of partial 2D FFT is significantly higher than that of RC decomposition. However there is not much difference in their perceptual quality.

Table 4 compares timing performance of the proposed partial 2D DFT algorithm with RC decomposition. All results are for ‘Lena’ of size 512×512. The timing improvement obtained by the proposed algorithm is significant. If $m^2/N^2 = 1/4$, or $m = 256$, the reduction on the latency is 84% and on the total computation time is 76%. The image reconstructed by partial 2D algorithm is still of quite high quality, as shown in Fig.4(b).

Table 4. Timing performance of RC decomposition and the proposed algorithm for ‘Lena’

Results	RC(Full)	m^2/N^2		
		$1/4$	$1/16$	$1/64$
Latency (cycles)	840,807	132,050	33,350	8,570
Total time (cycles)	1,678,336	394,194	295,494	270,714

5. CONCLUSION

In this paper, we described a technique to do efficient image reconstruction of Fourier images. It utilizes the fact that for natural and aerial images, most of the information is concentrated in the four corners of the Fourier image. We present an algorithm for computing partial 2D Fourier transform and show how this can be used for efficient image reconstruction. The 2D algorithm has reduced computation complexity compared with conventional RC decomposition algorithm, and more importantly, reduces the number of memory accesses. We mapped this algorithm onto a Xilinx Virtex-II Pro-100 FPGA platform. The hardware implementation is very efficient and has a very low latency. Moreover, the algorithm is inherently parallel and the throughput can be significantly increased if mapped to platforms with higher communication/memory bandwidth.

ACKNOWLEDGEMENTS

The authors thank Dr. Nikos Pitsianis and Dr. Xiaobai Sun in Duke University for assistance in the decomposition algorithm for 2D DFT.

6. REFERENCES

[1] J. W. Cooley and J. W. Tukey, “An algorithm for the machine computation of complex Fourier series,” *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[2] M. Fleury and A.F. Clark, “Parallelising a set of 2-D frequency transforms in a flexible manner,” *IEE Proceedings. Vision, Image, and Signal Processing*, vol. 145, pp. 65–72, 1998.

[3] H. V. Sorensen and et al., “Efficient computation of the DFT with only a subset of input or output points,” *IEEE Transactions on Signal Processing*, vol. 41, pp. 1184–1200, 1993.

[4] D. P. Skinner, “Pruning the decimation-in-time FFT algorithm,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, pp. 193–194, 1976.

[5] S. He and M. Torkelson, “Computing partial DFT for comb spectrum evaluation,” *IEEE Signal Processing Letters*, vol. 3, pp. 173–175, 1996.

[6] P.A. Milder and et al., “Discrete Fourier transform compiler: from mathematical representation to efficient hardware,” Tech. Rep., Carnegie Mellon University, 2007.

[7] C. Van Loan, *Computational framework of the fast Fourier transform*, Society for Industrial and Applied Mathematics (SIAM), 1992.

[8] N. P. Pitsianis, *The Kronecker product in approximation and fast transform generation*, Dissertation for the degree of doctor of philosophy, Cornell University, Jan. 1997.

[9] M. Wakin, “Standard test images - Lena-Lenna,” `http://www.ece.rice.edu/~wakin/images/`.

[10] “FFT Xilinx Logicore,” Website, `http://www.xilinx.com/products/ipcenter/FFT.htm`.

[11] “The USC-SIPI Image Database,” Website, `http://sipi.usc.edu/database/`.