

# FPGA Architecture for 2D Discrete Fourier Transform Based on 2D Decomposition for Large-sized Data

Chi-Li Yu · Jung-Sub Kim · Lanping Deng ·  
Srinidhi Kestur · Vijaykrishnan Narayanan ·  
Chaitali Chakrabarti

Received: 14 January 2010 / Revised: 25 May 2010 / Accepted: 26 May 2010  
© Springer Science+Business Media, LLC 2010

**Abstract** Applications based on Discrete Fourier Transforms (DFT) are extensively used in several areas of signal and digital image processing. Of particular interest is the two-dimensional (2D) DFT which is more computation- and bandwidth-intensive than the one-dimensional (1D) DFT. Traditionally, a 2D DFT is computed using Row-Column (RC) decomposition, where 1D DFTs are computed along the rows followed by 1D DFTs along the columns. Both application

specific and reconfigurable hardware have utilized this scheme for high-performance implementations of 2D DFT. However, architectures based on RC decomposition are not efficient for large input size data due to memory bandwidth constraints. In this paper, we propose an efficient architecture to implement 2D DFT for large-sized input data based on a novel 2D decomposition algorithm. This architecture achieves very high throughput by exploiting the inherent parallelism due to the algorithm decomposition and by utilizing the row-wise burst access pattern of the external memory. A high throughput memory interface has been designed to enable maximum utilization of the memory bandwidth. In addition, an automatic system generator is provided for mapping this architecture onto a reconfigurable platform of Xilinx Virtex-5 devices. For a  $2K \times 2K$  input size, the proposed architecture is 1.96 times faster than RC decomposition based implementation under the same memory constraints, and also outperforms other existing implementations.

---

This paper is an extension of our paper that appeared in SIPS '09. The added sections are: (1) Impact of large data size on conventional 2D DFT architecture (Section 2.2); (2) Detailed descriptions of the infrastructure components of the FPGA platform (Section 4.2); (3) Detailed description of the automatic 2D DFT system generator (Section 5); (4) Accuracy analysis of the 2D DFT (Section 6.4).

---

C.-L. Yu (✉) · L. Deng · C. Chakrabarti  
School of Electrical, Computer and Energy Engineering,  
Arizona State University, Tempe, AZ, USA  
e-mail: chi-li.yu@asu.edu

L. Deng  
e-mail: ldeng2@asu.edu

C. Chakrabarti  
e-mail: chaitali@asu.edu

J.-S. Kim · S. Kestur · V. Narayanan  
Department of Computer Science and Engineering,  
Pennsylvania State University, University Park, PA, USA

J.-S. Kim  
e-mail: jskim@cse.psu.edu

S. Kestur  
e-mail: kesturvy@cse.psu.edu

V. Narayanan  
e-mail: vijay@cse.psu.edu

**Keywords** Multi-dimensional signal processing ·  
DFT · Algorithm-architecture co-design

## 1 Introduction

Discrete Fourier Transform (DFT) is widely used in digital signal processing (DSP) and scientific computing applications. In particular, the two-dimensional (2D) DFT is used in a wide variety of imaging applications which need spectral and frequency-domain analysis such as, image watermarking, finger print recognition, synthetic aperture radar (SAR) processing and medical

imaging. The image sizes of many of the applications have increased over the years. In SAR processing [1], digital holographic imaging [2] and medical imaging, for instance, the required data size could be as large as  $2,048 \times 2,048$ . Therefore, there is a need for new algorithms and architectures to support 2D DFT of large data sizes.

Existing 2D DFT implementations include software solutions, such as FFTW [3], Spiral [4], Intel MKL [5] and IPP [6] which can run on general purpose processors, multicore architecture [7], and supercomputers [8, 9]. Though these platforms can achieve high performance, they are not suitable for embedded applications.

There are also many hardware solutions. Several of them are based on the Fast Fourier Transform (FFT) [10]. These include the dedicated FFT processor chips [2, 11–14], and field programmable gate array (FPGA) based implementations [15–20]. As for the FFT chips, their manufacturing cost is quite high, and once a chip is manufactured, its functionality and performance cannot be changed anymore. This is why FPGAs have become an attractive alternative for 2D DFT implementations. In fact, FPGAs are being widely used in various embedded signal and image processing systems such as smart cameras, radar image reconstruction in which FFT is a key component.

In this paper, we describe FPGA architectures for 2D DFT that are targeted for large data sizes. In architectures, such as those in [17], the performance degrades significantly when data size increases and the data does not fit in the on-chip memory. In these architectures, the bottleneck is the data transfer between the off-chip and on-chip memories. This problem has been addressed in [2], but it requires an additional transpose operation. In this work, we avoid transpose operation by implementing a new two-dimensional (2D) decomposition algorithm, and also design a customized memory interface which maximizes the external memory bandwidth. The proposed algorithm partitions the original data into a mesh of sub-blocks, performs butterfly type operations between sub-blocks and then computes local 2D DFT on each of the sub-blocks. The size of the sub-blocks is a function of the available FPGA resources and is determined automatically. The experimental results demonstrate that our architecture based on the 2D decomposition algorithm achieves better performance than optimized architectures based on Row-Column (RC) decomposition.

The rest of the paper is organized as follows. Section 3 briefly introduces 1D and 2D DFT and derives the proposed 2D decomposition algorithm. Section 4 describes in detail our novel FPGA archi-

tecture for 2D DFT. Section 5 describes the automatic system generator. Various configurations of our architecture are evaluated in Section 6, and concluding remarks are given in Section 7.

## 2 Background

In this section, we first briefly review the basic algorithms of 1D and 2D DFTs in order to better understand the proposed 2D decomposition (Section 2.1), and then present the problems of implementing 2D DFT by using RC decomposition in FPGA based architectures (Section 2.2).

### 2.1 1D DFT and 2D DFT

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

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

where  $W_N = e^{-j2\pi/N}$ . The computation complexity of an  $N$ -point DFT is  $O(N^2)$ , and an  $N$ -point FFT is  $O(N \log N)$ .

A 2D DFT of  $N \times N$  samples,  $x(i_1, i_2)$ , is defined by,

$$Y(k_1, k_2) = \sum_{i_1=0}^{N-1} \sum_{i_2=0}^{N-1} x(i_1, i_2) \cdot W_N^{k_1 i_1 + k_2 i_2}, \quad (2)$$

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

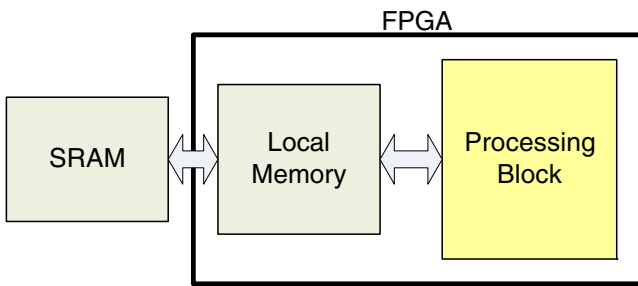
The Row-Column (RC) decomposition algorithm, decomposes a 2D DFT into multiple 1D DFTs as follows,

$$\tilde{X}(k_1, i_2) = \sum_{i_1=0}^{N-1} x(i_1, i_2) \cdot W_N^{k_1 i_1}, \text{ for } k_1 \in [0, N - 1], \quad (3)$$

and

$$Y(k_1, k_2) = \sum_{i_2=0}^{N-1} \tilde{X}(k_1, i_2) \cdot W_N^{k_2 i_2}, \text{ for } k_2 \in [0, N - 1]. \quad (4)$$

With RC decomposition, a 2D DFT on  $N \times N$  data can be computed by first performing  $N$  row-wise 1D DFTs, and then  $N$  column-wise 1D DFTs. If the 1D DFTs are implemented using FFT, the complexity is  $O(N^2 \log_2 N)$ .

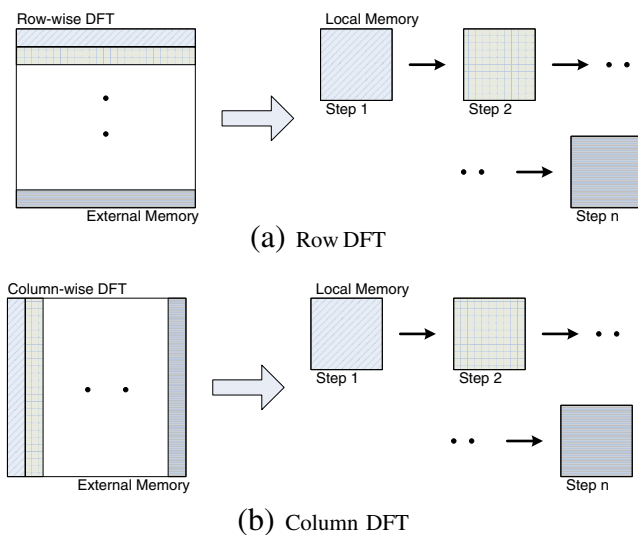


**Fig. 1** Basic 2D DFT architecture utilizing external memory.

### 2.2 Impact of Large Data Size on Conventional 2D DFT Architectures

In an FPGA implementation of RC decomposition based 2D DFT, the input 2D data is initially stored in the external memory, and row-wise DFTs followed by column-wise DFTs are performed. The basic architecture utilizing Synchronous Random Access Memory (SRAM) based external memory is shown in Fig. 1. For row-DFTs/column-DFTs, fixed number of rows/columns of input data are loaded into FPGA local memory, as shown in Fig. 2, and several 1D DFT cores are operated in parallel. This operation is repeated until all the rows/columns are traversed. This kind of implementation performs very well for small data sizes, since the row and column access times are same for SRAM.

For large data sizes, however, the SRAM solution becomes impractical due to the limited size and high cost. In this case, Synchronous Dynamic RAM (SDRAM), which has much larger capacity, should be used as



**Fig. 2** Data access pattern from external memory for row-DFT and column-DFT.

external memory. Because of its high throughput and high capacity, SDRAM is a standard component on most FPGA boards.

SDRAM has a three-dimensional organization, composed of banks, rows, and columns. It has non-uniform access time; accessing consecutive elements in a row has a low latency, while accessing data across different rows has a high latency. So if RC decomposition is used, row-wise DFTs are much faster than column-wise DFTs, since SDRAM memory organization favors data access along rows. Thus, column-wise data access is a major bottleneck for 2D DFT implementations.

Moreover, though the DDR2 and recent DDR3 standards for SDRAMs can achieve higher data rates for reads and writes to external memory, existing memory interface designs fail to utilize the bandwidth of the SDRAM device for column-DFTs. Even with a custom memory interface design, for large data sizes, conventional 2D DFT implementation with RC decomposition results in poor performance.

An alternative and a popular approach to overcome the imbalance of row-wise and column-wise memory accesses is to employ a transpose operation on the data as an intermediate step in 2D DFT [2]. However, it requires data to be transferred to and from memory three times, which is cumbersome and inefficient for large data sizes.

In the next section, we propose a new 2D decomposition algorithm which does not need the transpose operation to mitigate the data access problem of RC decomposition based algorithms, and can be efficiently mapped onto FPGA based architectures.

### 3 Decomposition Algorithms for DFT

In this section, the decomposition of 1D DFT is described in Section 3.1, followed by the 2D decomposition algorithm for 2D DFT in Section 3.2 and the functional components of the 2D DFT in Section 3.3.

#### 3.1 Decomposition of 1D DFT

A 1D DFT of length  $N$  can be decomposed and computed by a series of smaller transforms and permutations. We first represent DFT in the matrix-vector multiplication form as

$$[X_0 \ X_1 \ \dots \ X_{N-1}]^T = F_N \cdot [x_0 \ x_1 \ \dots \ x_{N-1}]^T, \quad (5)$$

where  $F_N$  is the twiddle factor matrix.

The decomposition of 1D DFT is essentially representation of  $F_N$  as a product of sparse matrices and is described as follows [21–23].

$$F_N = P_{N,p}(I_p \otimes F_m)\tilde{D}_N(F_p \otimes I_m), \tag{6}$$

where  $N = p \cdot m$ , where  $p$  and  $m$  are both integers.  $I_m$  is an  $m \times m$  identity matrix,  $\tilde{D}_N$  is a diagonal matrix of twiddle factors, and  $\otimes$  is the Kronecker or tensor product and can be expressed as

$$\tilde{D}_N(j, j) = W_N^{(j \bmod m) \cdot \lfloor j/m \rfloor} \text{ for } j = 0, 1 \dots N - 1, \tag{7}$$

$$A_n \otimes B_m = [a_{k,l} B_m]_{0 \leq k,l < n} \text{ for } A_n = [a_{k,l}]_{0 \leq k,l < n}. \tag{8}$$

Finally,  $P_{N,p}$  denotes permutation with stride  $p$ .

The traditional radix-2 FFT can be considered a specific case of recursive decomposition with factor  $p = 2$ .

### 3.2 2D Decomposition Algorithm

The general form of 2D DFT is described in matrix form as follows:

$$Y = F_M \cdot X \cdot F_N^T = F_M \cdot X \cdot F_N, \tag{9}$$

where input  $X$  and output  $Y$  are of size  $M \times N$ ;  $F_M$  and  $F_N$  are DFT matrices which are symmetric.

The expression  $(F_M \cdot X)$  is traditionally calculated by applying an  $M$ -point DFT for each column of  $X$ .

As described in Section 3.1, an  $M$ -point DFT can be replaced by the sparse matrix product form as depicted in Eq. 6. Hence, by partitioning a column of size  $M$  into  $p$  sub-blocks, the expression  $(F_M \cdot X)$  can be written as follows:

$$F_M \cdot X = P_{M,p} \cdot (I_p \otimes F_{M/p})\tilde{D}_M(F_p \otimes I_{M/p}) \cdot X, \tag{10}$$

where the permutation  $P_{M,p}$  term in Eq. 6 is taken into account in the final stage.

Similarly, the expression  $(X \cdot F_N)$  in Eq. 9 can be written as follows:

$$X \cdot F_N = X \cdot (F_q \otimes I_{N/q})\tilde{D}_N(I_q \otimes F_{N/q}) \cdot P_{N,q}. \tag{11}$$

Extending such a partitioning to both row-wise and column-wise elements, the 2D decomposition of Eq. 9 on a  $p \times q$  mesh is written as follows:

$$\tilde{Y} = (I_p \otimes F_{M/p})\tilde{D}_M(F_p \otimes I_{M/p}) \cdot X \cdot (F_q \otimes I_{N/q})\tilde{D}_N(I_q \otimes F_{N/q}). \tag{12}$$

$Y$  is obtained by applying a bit-reverse permutation ( $P$ ) on  $\tilde{Y}$  of Eq. 12.

### 3.3 Functional Components of the 2D Decomposition Algorithm

Equation 13 pictorially demonstrates the sequence of operations involved in the computation of the decomposed 2D-DFT.

$$Y = P \left( \underbrace{\underbrace{(I_p \otimes F_{M/p}) \tilde{D}_M(F_p \otimes I_{M/p}) \cdot X \cdot (F_q \otimes I_{N/q}) \tilde{D}_N(I_q \otimes F_{N/q})}_{\text{Step 1}}}_{\text{Step 2}} \right) \tag{13}$$

Step 3

Input  $X$  of size  $M \times N$ , is partitioned into a  $p \times q$  mesh where each sub-block in the mesh is of size  $M/p \times N/q$ . The four main steps are described below Fig. 3 presents the functional flow of the proposed 2D decomposition algorithm.

**Step 1** Row-wise data exchange with twiddle-factor multiplication:

There are  $q$  sub-blocks in each row, and each element inside one sub-block has to do data

exchange with the corresponding elements in the other  $(q - 1)$  sub-blocks. This data exchange (DX) operation can be implemented as a  $q$ -point 1D FFT followed by a twiddle-factor multiplication (with  $\tilde{D}_N$ ). Since there are  $M/p \cdot N/q$  elements in each sub-block, there are  $(MN/pq)(q \cdot \log_2 q)$  arithmetic operations with  $q$ -point 1D FFT for each row, and  $(MN/pq)(q \cdot \log_2 q)p$  for all rows. Including  $MN$  operations for twiddle-factor multiplication

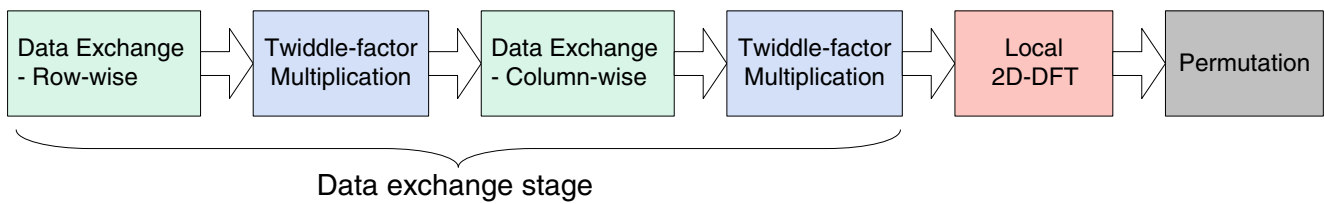


Fig. 3 The functional flow graph of 2D DFT.

( $\tilde{D}_N$ ), it takes a total of  $(MN/pq)(q \cdot \log_2 q)p + MN \approx O[MN(1 + \log_2 q)]$  arithmetic operations. Note that all  $q$ -point 1D FFTs can be computed in parallel.

Step 2 Column-wise data exchange with twiddle-factor multiplication:

$$Y_2 = \tilde{D}_M(F_p \otimes I_{M/p}) \cdot Y_1$$

Similar to Step 1, the DX is repeated for the  $p$  sub-blocks in each column. This step has a complexity of  $O[MN(1 + \log_2 p)]$  operations.

Step 3 Local 2D DFT on each sub-block:

$$\tilde{Y} = (I_p \otimes F_{M/p})Y_2(I_q \otimes F_{N/q})$$

After the row-wise and column-wise data exchange with twiddle-factor multiplications, 2D DFT computation is performed on each sub-block of size  $M/p \cdot N/q$ . This operation is fully parallel for all sub-blocks, and only limited by resource constraints in the underlying architecture. No data communication is required between any sub-blocks.

Step 4 Output permutation:

$$Y = P(\tilde{Y})$$

A bit-reverse permutation is required before generating the output. This is done by the host computer before display.

The pseudo code of the 2D decomposition algorithm for the case when the image is of size  $N \times N$  is presented in Fig. 4. The sub-blocks are of size  $K \times K$ , the mesh is of size  $(N/K) \times (N/K)$ , and the local memory is of size  $S \times S$ . Note that 1D DFT is computed along rows and columns in each stage. Also notice that there are other 2D decomposition methods, such as Vector Radix FFT [24], which recursively decomposes the 2D DFT into small-sized ones, like  $2 \times 2$  or  $4 \times 4$  2D DFT. While such methods can effectively reduce the number of multiplications, the recursive decomposition makes hardware implementation and memory addressing much more complicated.

#### 4 Proposed 2D-DFT Architecture

Figure 5 gives a bird's eye view of the FPGA architecture for implementing the 2D decomposition based 2D DFT. It is composed of several components that can be classified broadly into *Domain-specific components* and *Infrastructure components*. Components such as processing elements (PEs) are designed specially for the 2D DFT application and hence are domain-specific. Infrastructure components, including PowerPC, memory interface, host connection and UART, provide

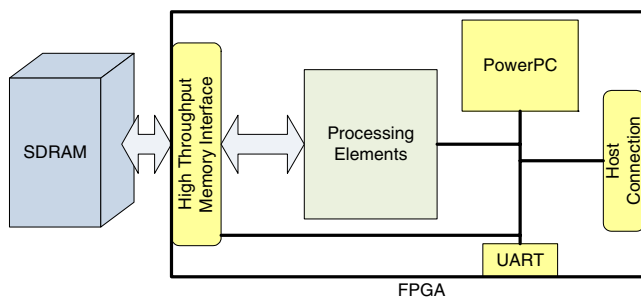
```

INPUT:
Input size:  $N \times N$ 
Local 2D DFT size:  $K \times K$ 
Mesh size:  $(N/K) \times (N/K)$ 
Local memory size:  $S \times S$ 

1. Data Exchange (DX)
for  $i = 1 : (N/S) \times (N/S)$ 
  Load DX data into local memory
  for  $j = 1 : 2(S/(N/K))$ 
    Compute  $(N/K)$ -point 1D FFT with twiddle-factor
    multiplication
  Store the result back to external memory

2. Local 2D DFT (LDFT)
for  $i = 1 : (N/S) \times (N/S)$ 
  Load LDFT data into local memory
  for  $j = 1 : 2(S/K)$ 
    Compute  $K$ -point 1D FFT
  Store the result back to external memory
  
```

Fig. 4 Pseudo code for 2D decomposition algorithm for 2D DFT.



**Fig. 5** Block diagram of the FPGA architecture for 2D decomposition based 2D DFT.

high-level control and support for the domain-specific components.

#### 4.1 Domain-specific Components

As shown in Fig. 6, the PE array consists of multiple PEs, where each PE is formed by a primitive 1D FFT IP core and a complex multiplier. The primitive FFT is used for both data exchange and local 2D DFT. In our design, we adopt Xilinx pipelined FFT IP [25]. The complex multiplier is used for twiddle-factor multiplication during data exchange. All PEs operate in parallel and access data from/to multi-banked local memory simultaneously without conflicts. The number of PEs and size of the primitive FFTs is determined by the memory bandwidth and available FPGA resources as explained in Section 5. Note that for ease of implementation, an input size of  $N \times N$  is always assumed, where  $N$  is a power of 2, otherwise zero-padding is applied.

The input data is stored in the external memory (SDRAM), and is logically partitioned into a mesh of sub-blocks. During each step mentioned in Section 3, portions of data are loaded into the FPGA local memory, processed, and then stored back to the external

memory. There are two identical local memories that serve as ping pong buffers. These local memories are implemented with dual-port Block RAM (BRAM) on the FPGA. The operations are described in details below.

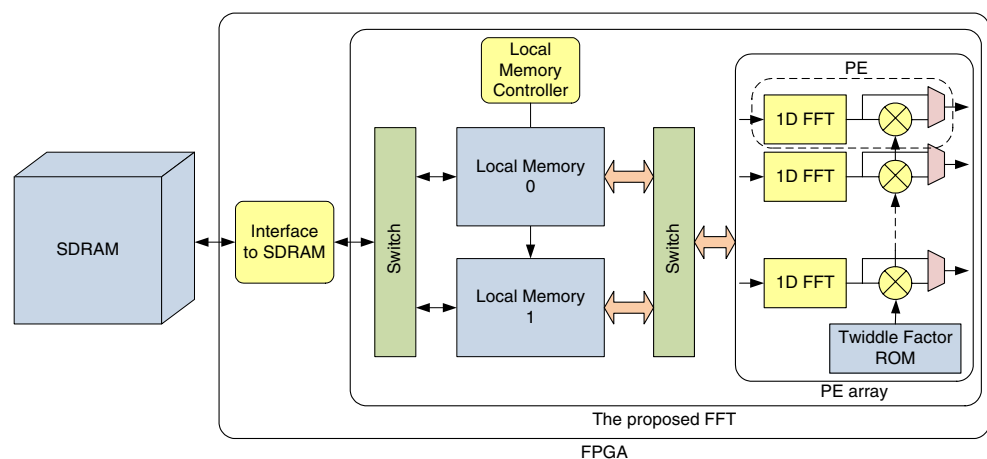
In the data exchange stage which consists of the first four blocks in Fig. 3, first, equal number of samples from the same position in each sub-block is loaded into local memory. This data is then used for computing both row-wise and column-wise data exchanges. Note that the data from the external memory is accessed only along the row direction as depicted in Fig. 7. This pattern is especially advantageous for accessing a dynamic memory, such as DDR2 and DDR3 SDRAM, which only favors row-wise burst access. The operations are repeated until the entire data is traversed, as shown in Fig. 7.

In the local 2D DFT stage (corresponding to the fifth block in Fig. 3), fixed number of contiguous sub-blocks of the 2D data are loaded into FPGA local memory and the PE array computes 1D transforms along rows and then along columns. This operation is repeated for all the blocks.

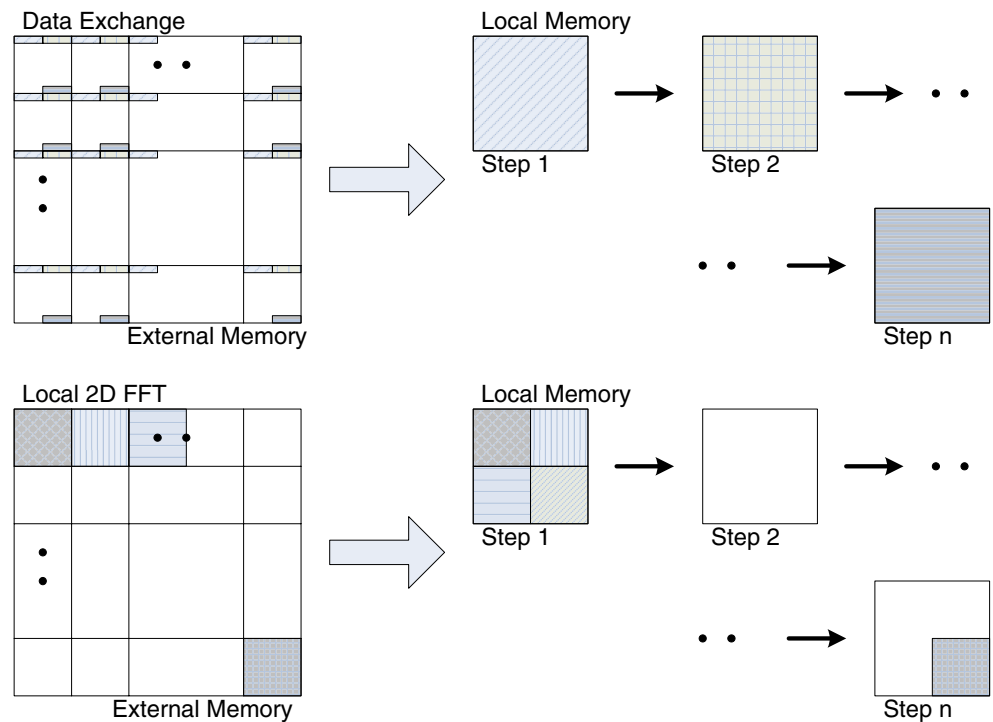
#### 4.2 Infrastructure Components

PowerPC is utilized for loading the input data into external memory, UART debugging and ethernet TCP/IP connections, and also run-time configurations for the domain-specific components, such as processing elements and data path controller. It is implemented as a Hard IP core in Xilinx FPGA, particularly the FX series in Virtex-4 and Virtex-5 device. If PowerPC can not be supported in case of LX, SX series of Virtex FPGA series, other soft-core processors like Microblaze can be utilized. UART block is used to provide a basic terminal to show status and debugging

**Fig. 6** The proposed 2D DFT architecture.



**Fig. 7** Data access pattern from external memory for data exchange and local 2D DFT stages.



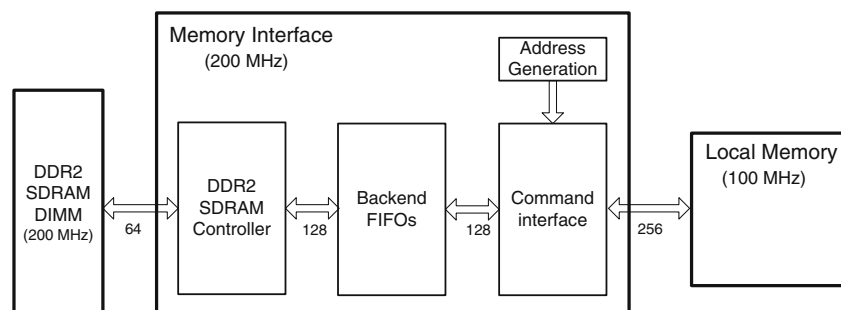
\* Pictorial mapping to memory space

information. For host connection, currently Ethernet interface is preferred because Xilinx supports Hard tri-mode MAC (TEMAC) for Virtex-4 and Virtex-5 [26]. A light-weight implementation of the TCP/IP protocol library, lwIP [27], is loaded to support ethernet communication.

*High Throughput Memory Interface* While the PE array can finish computations very fast, the bottleneck is the interface to external memory. Xilinx’s Multi-Port Memory Controller (MPMC), for instance, is a very versatile controller supporting SDRAM/DDR/DDR2 memory. However, its peak throughput is only 50% of a DDR2-SDRAM DIMM’s peak transfer rate. To alleviate this bottleneck, we design a customized high throughput memory interface.

The customized memory interface, as shown in Fig. 8, has a 128-bit wide internal data-bus. Since the DDR2 SDRAM device has an operating frequency of 200 MHz and an user application in FPGA runs at 100 MHz, the memory interface operates at 200 MHz and has a 256-bit wide data bus between the interface and the application. This enables data transfer rates of up to 256 bits at 100 MHz or 3,200 MB/s. Together with double buffering technique on local memory, the customized memory interface enables us to completely overlap communication with computation and avoid any loss of performance due to communication bottleneck. This memory interface can be ported onto any FPGA board with SDRAM DIMMs.

**Fig. 8** High throughput memory interface.



### 5 Automatic 2D DFT System Generator

Given the specifications of input data and hardware platform, we propose a 2D DFT system generator to automatically generate a 2D DFT implementation based on the proposed 2D decomposition algorithm. The automation flow is shown in Fig. 9. A design optimizer first determines the best decomposition based on input specifications to obtain the size of the primitive FFT. Then, according to the available FPGA resources and memory bandwidth, the optimizer calculates the number of PEs. The information is passed to the system generator, which generates hardware module in Verilog or VHDL. Then, the HDL files are fed into the FPGA tool to produce final configuration bit-files. Note that no user intervention is required for the entire process.

#### 5.1 Choosing Size of the FFT IP Core

The input specifications include data size, target device, memory type, and accuracy requirement. The data size is a power of two, typically from  $512 \times 512$  to  $4,096 \times 4,096$ . The target FPGA platform is Virtex-5 from Xilinx. For memory type, only DDR2 SDRAM

is currently supported to provide large memory bandwidth. However, the memory interface can be easily extended to other memory types by replacing SDRAM controller block. The user can choose either 16-bit implementation for resource constrained designs, or 24-bit implementation for high accuracy designs.

Let  $N = K \cdot L$ . Then the sub-block size is  $L \times L$  for a mesh of size  $K \times K$ . The design optimizer is used to find the best decomposition, i.e.  $L$  and  $K$  for a given  $N$ . Since our architecture uses high throughput memory interface (Section 4.2) and double-buffering technique to overlap communication with computation, the communication cost is not be considered in the modeling. Moreover, if the local memory has a fixed size  $S \times S$ , then there is an implicit constraint on the possible values for  $K$  and  $L$  namely,  $K \leq S$  and  $L \leq S$ .

Assume that it takes  $T_{dx}$  to complete row-wise and column-wise data exchange operations, and  $T_{ldft}$  to complete local 2D DFTs, respectively. Then the total DFT computation time  $T_{total}$  for input data of size  $N \times N$  can be calculated as

$$T_{total} = (N^2/S^2) \cdot (T_{dx} + T_{ldft}). \tag{14}$$

Since the local memory can hold up to  $(S/K)^2$  blocks for data exchange and  $(S/L)^2$  blocks for local 2D DFTs each time, then if  $T_{dft(n \times n)}$  is defined as the time required for a 2D DFT computation of size  $n \times n$ ,

$$T_{dx} = (S/K)^2 \cdot T_{dft(K \times K)}, \tag{15}$$

$$T_{ldft} = (S/L)^2 \cdot T_{dft(L \times L)}. \tag{16}$$

For pipelined implementations,  $T_{dft(n \times n)}$  is proportional to the 2D DFT size  $n \times n$ , and

$$T_{dx} = (S/K)^2 \cdot c \cdot K \cdot K = c \cdot S^2, \tag{17}$$

$$T_{ldft} = (S/L)^2 \cdot c \cdot L \cdot L = c \cdot S^2. \tag{18}$$

where  $c$  is a constant. It can be seen that both data exchange time  $T_{dx}$  and local 2D DFT time  $T_{ldft}$  depend only on the local memory size.

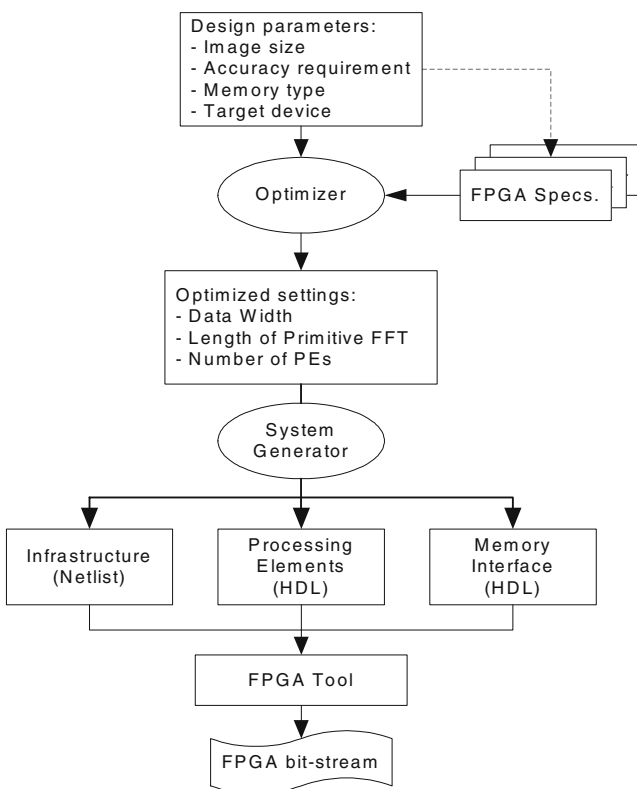
To fully utilize standard IP cores, only  $Q$ -point DFT will be implemented to accomplish both  $K$ -point and  $L$ -point DFTs, where  $Q = \max\{K, L\}$ . So if the FPGA can support  $P$   $Q$ -point processing elements, then

$$T_{dx} = T_{ldft} = c \cdot S^2/P. \tag{19}$$

Combining Eq. 19 with Eq. 14, we have

$$T_{total} = 2 \cdot c \cdot N^2/P. \tag{20}$$

This implies that the time to complete DFT depends on data size  $N \times N$ , and the number of processing



**Fig. 9** Automation flow of generating architecture for 2D decomposition based 2D DFT.

**Table 1** Resources required for Xilinx pipelined FFT IP.

		FFT point	32	64	128	256
Virtex-4	24 bit	DSP48	16	16	24	24
		BRAM	0	2	6	8
	16 bit	DSP48	6	6	9	9
		BRAM	0	1	3	4
Virtex-5	24 bit	DSP48E	24	24	36	36
		BRAM	0	2	5	7
	16 bit	DSP48E	8	8	12	12
		BRAM	0	1	3	4

elements,  $P$ , which is, in turn, determined by the resources available on the FPGA.

As primitive FFT size increases, the number of DSPs and BRAMs both increase, as shown in Table 1. For an FPGA with limited DSP and BRAM resources, we need to keep the value of  $Q$  as small as possible, so that more PEs can be accommodated. Since  $Q = \max(K, L)$ , and  $K \cdot L = N$ , the smallest value of  $Q$  is  $\sqrt{N}$ . However, if  $\sqrt{N}$  is not an integer, we need to find a factorization of  $N$  so that  $K$  and  $L$  are close to each other. Once the optimal factorization is found, the length of the primitive FFT cores,  $Q$ , can be determined. For instance, if  $N = 1,024$ ,  $Q = 32$ , and if  $N$  increases to 4,096,  $Q = 64$ .

### 5.2 Choosing the Number of PEs

After choosing the size of the FFT IP core, the number of PEs is determined. Based on the hardware resources of the target FPGA and resources required by the FFT IP core given in Table 1, the 2D DFT generator calculates the maximum number of PEs,  $P_{RES}$ , that can fit onto the FPGA. The external memory type/interface plays an equally important role in determining the number of PEs, since the pipelined Xilinx FFT IP [25] can input 1 datum/cycle. For instance, suppose that the data width of a complex datum is 32 ( $16 \times 2$ ) bits, and if FFT IP is running at 100 MHz, its data rate is 400 MB/s. Now, let  $P_{BW}$  be the maximum number of PEs that can be supported by the available memory bandwidth. If the external memory is one DIMM of DDR2-400 SDRAM, which can offer 3,200 MB/s data rate, up to 8 PEs can be supported, and  $P_{BW}$  would be set to “8”. In this case, the performance of the 2D DFT cannot be improved with more PEs. Eventually, the number of PEs is decided by the minimum of  $P_{RES}$  and  $P_{BW}$ , i.e.

$$P = \min\{P_{RES}, P_{BW}\}. \tag{21}$$

Once the design optimizer decides the number and type of the primitive FFT IPs and memory partitioning,

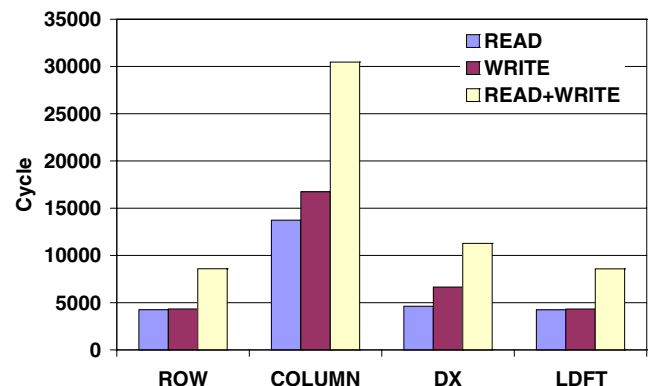
the system generator generates all the required hardware modules, namely, infrastructure block, FFT core block and memory interface block. The infrastructure blocks such as UART and host connection are fixed, whereas other blocks are user specified and provided in a template format, with several parameters that are set based on decisions of the design optimizer. After the generation of these modules, scripts for Xilinx flow are produced to run Xilinx tool automatically. Finally, the configuration bit file consisting of both hardware bit file and software binaries are generated from FPGA tool.

## 6 Evaluation

In this section, the 2D DFT architecture generated using the DFT system generator is evaluated. First, the evaluation of high throughput memory interface is presented in Section 6.1, and the resource utilization is analyzed in Section 6.2. Then, the evaluation of performance for various input sizes is presented and compared with existing solutions in Section 6.3. Finally, the numerical accuracy of the 2D DFT is analyzed in Section 6.4. For the evaluations, Xilinx 10.1 tool set and Modelsim 6.4 are used, and Virtex-5FX device is considered as the candidate FPGA device.

### 6.1 Memory Throughput for RC- and 2D Decomposition- Based Architectures

The performance of the memory interface is measured in terms of the number of cycles taken to read/write data from/to DDR2 SDRAM to/from local memory. The performance for read, write and read + write operations for various memory access patterns for an input size of  $2,048 \times 2,048$  and local memory size of



**Fig. 10** Memory access time for different access patterns when on-chip memory size is  $128 \times 128$ .

**Table 2** Resource utilization and memory requirement for different configurations.

Primitive FFT	Bit width	# of PEs	DSP	BRAM (36 kbits)	Slice LUTs	Mem. BW required (MB/s)
32-point FFT	16 bits	4 PEs	36	137	25,536	1,600
		8 PEs	72	137	35,238	3,200
	24 bits	16 PEs	144	145	54,442	6,400
		4 PEs	112	145	33,294	3,200
64-point FFT	16 bits	8 PEs	224	145	46,326	6,400
		4 PEs	36	139	26,882	1,600
	24 bits	8 PEs	72	141	37,958	3,200
		4 PEs	112	149	34,991	3,200
		8 PEs	224	153	50,132	6,400

$128 \times 128$  is shown in Fig. 10. It can be observed that column access is much slower compared to other access patterns, and hence it would be the bottle-neck for direct RC method. On the contrary, data exchange (DX) and local DFT (LDFT) are both row-wise accesses. Therefore, the proposed 2D DFT avoids the performance penalty caused by column access and hence can achieve a higher performance.

## 6.2 Resource Utilization

The resource utilization of the PEs (Xilinx FFT IPs, complex multipliers, and the control units) and the memory requirement in terms of bandwidth, are summarized in Table 2. This evaluation is based on the device XC5FX200T, which has 384 DSP slices, 456 BRAMs (16 Mbs), 30,720 logic slices, and 122,880 slice LUTs. For large image sizes from  $1,024 \times 1,024$  through  $4,096 \times 4,096$ , the size of the FFT IP can be either 32-point or 64-point. The data representation can be either 16-bit or 24-bit. The infra-structure block uses fixed number of resources of 8,149 logic slices and 47 BRAMs including TEMAC for host interface. We assume the operating frequency of the IPs is 100 MHz and the external memory device to be one DDR2-400 SDRAM DIMM operating at 200 MHz.

From Table 2, we see that the number of BRAMs and Slice LUTs utilized by 64-point FFT are slightly higher than 32-point FFT, though the number of DSP

resources utilized is the same for both the cases. The number of DSP resources and the memory bandwidth required increase almost linearly with the number of PEs. Also, there is a large increase in DSP resources and memory bandwidth, when the data-width is changed from 16-bit to 24-bit.

From Table 2 we also see that the target FPGA can fit either 16 32-point FFT cores with 16-bit data width, or 8 32-point FFT cores with 24-bit data width, or 8 64-point FFT core with 16-bit, or 8 64-point FFT cores with 32-bit data width. The DSP resource requirement for the 16 FFT cores (32-point and 64-point) with 24-bit data representation exceeds that supported by the Virtex5 FPGA and hence cannot fit in the device. Also, the configuration with 16 64-point FFT cores with 16-bit data width cannot be supported due to local memory constraints. Thus, for both cases, the 2D DFT system generator (see Section 5) would set  $P_{RES}$  as “8”. Note that  $P_{RES}$  could be higher for other FPGAs with larger capacities.

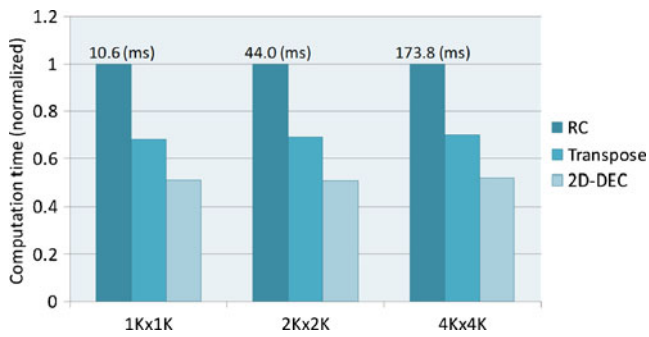
Table 2 also list the required bandwidth for each configuration, and based on the table, the 2D DFT system generator is able to determine  $P_{BW}$ . For example, if we have 1 DIMM of DDR2-400 SDRAM with a bandwidth of 3200 MB/s, then for 64-point FFT IP with 24-bit representation,  $P_{BW} = 4$ . Although  $P_{RES}$  in this case is “8”, the final number of PEs is “4”, based on Eq. 21. However if we have 2 DIMMs of DDR2-400 SDRAM, then the available memory bandwidth is 6,400 MB/s, and the final number of PEs is 8. Note that  $P_{BW}$  could be higher if more DIMMS or a faster memory device, such as DDR2-800 SDRAM, is used.

**Table 3** Performance for different input sizes.

Input size	Primitive	# of PE	Computation time (ms)	Frame rates
$1,024 \times 1,024$	32PT FFT	8	5.4	182
		4	10.7	92
$2,048 \times 2,048$	64PT FFT	8	22.4	44
		4	43.4	23
$4,096 \times 4,096$	64PT FFT	8	90.4	11
		4	174.3	5

**Table 4** Performance comparison for different architectures for input size  $2048 \times 2048$ .

	RC	Transpose	2D-DEC
Computation time (ms)	44.0	30.4	22.4
Frame rates	23	33	44



**Fig. 11** Comparison of the computation times (normalized to RC-based method) for different image sizes.

### 6.3 Comparison of Timing Performance

Table 3 shows the performance of the 2D DFT architecture for different input sizes and different architecture configurations. The performance is specified in terms of frame rate, which refers to the number of frames of input data that the 2D DFT architecture can process in a second. The size of the FFT primitive is chosen by the design optimizer (see Section 5). It can be seen that the performance is inversely proportional to the input data size, but directly proportional to the number of PEs.

Table 4 compares the performance in terms of computation time and frame rate for FPGA implementations corresponding to (1) RC decomposition, (2) Transpose (row-DFT – transpose – row-DFT) and (3) the proposed 2D decomposition algorithm. This comparison is done for identical architecture constraints such as one DDR2-400 SDRAM DIMM, 128 × 128 local memory size, 16-bit implementation and for an input data size of 2,048 × 2,048. The evaluation of the candidate architectures is performed using the proposed high throughput memory interface for the case when there are 8 PEs. The table shows that under identical conditions, the 2D decomposition architecture provides a 96.4% improvement compared to the RC decomposition architecture and a 35.7% improvement compared to the transpose-based architecture. The improvement can be maintained across different image sizes, as shown in Fig. 11.

Further, Table 5 presents a comparison of other existing 2D DFT implementations with the proposed 2D decomposition based DFT (2D-DEC) architecture that utilizes maximum memory bandwidth on Virtex-5 FPGA. Uzun et al. [28] have used SRAM as the external memory with Virtex-E FPGA. Our 2D-DEC architecture provides significant performance improvement over [28] for the same input data size of 1,024 × 1,024. Dillon [20] has used two cascaded Virtex-II FPGAs for row-DFTs and column-DFTs and large SRAMs for intermediate data storage. In contrast, our 2D-DEC architecture uses a single FPGA with cheaper DDR2 SDRAM as external memory. Furthermore, our datawidth is 32 bits compared to 16 bits in [20]. Lenart et al. [2] have implemented a transpose based architecture with DDR SDRAM memory. Our 2D-DEC architecture operates at 100 MHz compared to 250 MHz and still provides performance improvement by avoiding transpose operation.

### 6.4 Accuracy Evaluation

As mentioned earlier, the proposed 2D DFT is implemented using Xilinx pipelined FFT IPs, which are fixed-point cores. Due to finite wordlength effects, accuracy is a major design issue that needs to be analyzed.

An  $N$ -point Xilinx pipelined FFT core consists of  $\log_2(N)$  stages of radix-2 butterflies. For an  $N \times N$  2D DFT, therefore, every input element needs to be operated by  $2 \log_2(N)$  stages of radix-2 butterflies, irrespective of the decomposition algorithm used. To prevent overflow, Xilinx provides corresponding function (right-shifting 0–3 bits) on each pair of two stages in the FFT IP. The corresponding control bits can be programmed by the user. In this work, we first scale the input array so that  $|x(i_1, i_2)| < 1/2.4142$  [29] to prevent overflow in the first stage. Then we right-shift 1 bit for every pair of stages to prevent overflow in the FFT computation. This half-a-bit-per-stage scheme works for a broad class of signals [30].

To quantify the output accuracy, we feed uniformly distributed random numbers into the 2D DFT. We

**Table 5** Performance comparison with existing works.

Input size	Method	Technology	External memory	Complex data (bits)	Frame rate (fps)
1024 × 1,024	Uzun [28]	Virtex-E, 180 nm, 27 MHz	Quad SRAM	2 × 16	13
	2D-DEC	Virtex-5FX, 65 nm, 100 MHz	Single SDRAM	2 × 16	182
2,048 × 2,048	Dillon [20]	Virtex-II × 2, 130 nm, 125 MHz	Dual SRAM	2 × 8	120
	Lenart [2]	ASIC, 130 nm, 250 MHz	Dual SDRAM	2 × 16	20
	2D-DEC	Virtex-5FX, 65 nm, 100 MHz	Single SDRAM	2 × 16	44

**Table 6** SNR(dB) of proposed 2D DFT, where the input set is drawn from uniformly distributed random number and  $|x(i_1, i_2)| < 1/2.4142$ .

DFT size	64 × 64	128 × 128	256 × 256	512 × 512	1,024 × 1,024	2,048 × 2,048
16 bit	70.56	69.94	69.67	69.36	68.72	67.72
20 bit	94.50	94.50	93.92	92.97	92.16	91.59
24 bit	118.77	118.17	117.52	117.15	116.81	116.03

adopt a commonly used criterion, Signal-to-Noise-Ratio (SNR), which is defined as:

$$\text{SNR}(dB) = 10 \log_{10} \frac{P_{\text{output}}}{P_{\text{quantnoise}}},$$

where  $P_{\text{output}}$  is average output power, and  $P_{\text{quantnoise}}$  is acquired by comparing the hardware output with Matlab floating point results. The SNR results for randomly generated input data are listed in Table 6.

From Table 6, three observations can be made. First, SNR of the 2D DFT is mainly dominated by Xilinx FFT IP. For example, a 16-bit 1,024-point Xilinx FFT has an SNR around 73 dB [25]. The 16-bit 2D DFTs have lower SNRs than 73 dB, because the 2D DFTs have more stages of radix-2 butterflies. Second, we can gain about 6 dB of SNR when 1 additional bit is added to the data width. This observation is consistent with the results in [25]. Based on this, the user can decide the data width. Thirdly, the accuracy is insensitive to the 2D DFT problem size (as shown in this case). To explain this, one should notice that the denominator of the SNR equation is dominated by data width, which is fixed, and the numerator is dominated by the signal power (or amplitude). In Fig. 12, we record the internal maximum amplitude of every stage in the 2D DFT. The

values are maintained around a certain value (0.75 in this case), irrespective of the number of butterfly stages. That means the signal power is almost fixed throughout the whole computation. As the denominator and numerator are both fixed, the SNR value can be maintained. Besides, Fig. 12 indicates that the internal data utilizes full data-width but never lets overflow occur. It also proves that the scaling scheme works well.

If accuracy is not critical, it is recommended to use a smaller data-width to minimize the hardware resources occupied by the primitive FFTs. Typically, a 16-bit Xilinx FFT core occupies only 30–40% of resources compared with a 24-bit design. With saved resources, we can put more primitive FFTs into the FPGA to further accelerate the 2D DFT computation.

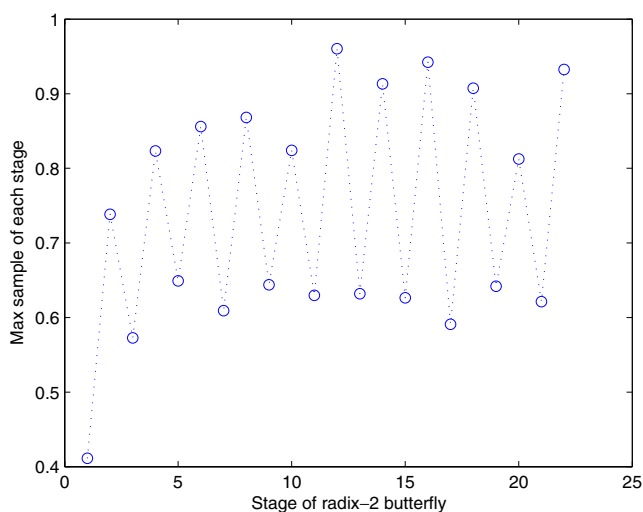
## 7 Conclusion

In this paper, we have proposed an efficient architecture to implement the 2D DFT for large input sizes based on a novel 2D decomposition algorithm. This architecture provides high performance by leveraging the inherent parallelism of the 2D decomposition and by scheduling data communication to overlap with computation. The memory bandwidth problem is alleviated by employing a custom-designed high throughput memory interface. In addition, a system generator is provided, which can automate the generation of an optimized version of the 2D DFT architecture for various input sizes. The evaluation of this architecture shows significant performance enhancements over existing 2D DFT implementations.

**Acknowledgements** This work is supported in part by a grant from DARPA W911NF-05-1-0248. In addition, the authors gratefully acknowledge the help of Dr. Nikos Pitsianis and Dr. Xiaobai Sun of Duke University.

## References

- Chan, Y. K., & Lim, S. Y. (2008). Synthetic aperture radar (SAR) signal generation. *Progress In Electromagnetics Research B*, 1, 269–290.



**Fig. 12** Maximum amplitude of every stage in a 2048 × 2048 2D DFT, where the input is uniformly distributed random numbers.

2. Lenart, T., Gustafsson, M., & Owall, V. (2008). A hardware acceleration platform for digital holographic imaging. *Journal of Signal Processing System*, 52(3), 297–311.
3. Frigo, M., & Johnson, S. (1998). FFTW: An adaptive software of the FFT. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3, 1381–1384.
4. Püschel, M., et al. (2005). SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE*, 93(2), 232–275.
5. Intel Math Kernel Library (MKL). <http://software.intel.com/en-us/intel-mkl/>.
6. Intel Integrated Performance Primitives (IPP). <http://software.intel.com/en-us/intel-ipp/>.
7. Franchetti, F., et al. (2009). Discrete Fourier transform on multicore. *IEEE Signal Processing Magazine, Special Issue on "Signal Processing on Platforms with Multiple Cores"*, 26(6), 90–102.
8. Eleftheriou, M., et al. (2005). Scalable framework for 3D FFTs on the blue gene/l supercomputer: Implementation and early performance measurements. *IBM Journal of Research and Development*, 49, 457–464.
9. Fang, B., et al. (2007). Performance of the 3D FFT on the 6D network torus QCDQC parallel supercomputer. *Computer Physics Communications*, 176(8), 531–538.
10. Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine computation of complex Fourier series. *Mathematics of Computation*, 19, 297–301.
11. Yeh, W.-C., & Jen, C.-W. (2003). High-speed and low-power split-radix FFT. *IEEE Transactions on Signal Processing*, 51, 864–874.
12. Lin, Y.-W., et al. (2005). A 1-GS/s FFT/IFFT processor for UWB applications. *IEEE Journal of Solid-State Circuits*, 40, 1726–1735.
13. PowerFFT ASIC. <http://www.eonic.com/index.asp?item=32>.
14. Baas, B. (1999). A low-power, high-performance, 1024-point FFT processor. *IEEE Journal OF Solid-state Circuits*, 34(3), 380–387.
15. Uzun, I., Amira, A., & Bouridane, A. (2005). FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEE Proceedings. Vision, Image, and Signal Processing*, 152(3), 283–296.
16. Sasaki, T., et al. (2005). Reconfigurable 3D-FFT processor for the car-parrinello method. *The Journal of Computer Chemistry, Japan*, 4(4), 147–154.
17. D'Alberto, P., et al. (2007). Generating FPGA accelerated DFT libraries. In *IEEE symposium on field-programmable custom computing machines (FCCM)* (pp. 173–184).
18. Kumhom, P., Johnson, J., & Nagvajara, P. (2000). Design, optimization, and implementation of a universal FFT processor. In *IEEE ASIC/SOC conference* (pp. 182–186). IEEE.
19. Milder, P. A., et al. (2008). Formal datapath representation and manipulation for implementing DSP transforms. In *Design automation conference (DAC)* (pp. 385–390).
20. Dillon, T. (2001). Two virtex-II FPGAs deliver fastest, cheapest, best high-performance image processing system. *Xilinx Xcell Journal*, 41, 70–73.
21. Milder, P. A., et al. (2007). *Discrete Fourier transform compiler: From mathematical representation to efficient hardware*. Carnegie Mellon University, Tech. Rep. CSSI-07-01.
22. Van Loan, C. (1992). *Computational framework of the fast Fourier transform*. Philadelphia, PA: SIAM.
23. Pitsianis, N. P. (1997). *The Kronecker product in approximation and fast transform generation*. Dissertation for the degree of Doctor of Philosophy, Cornell University.
24. Wu, H. R., & Paoloni, F. J. (1989). The structure of vector radix fast Fourier transforms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(9).
25. FFT Xilinx LogiCore. <http://www.xilinx.com/products/ipcenter/FFT.htm>.
26. Hard tri-mode MAC. [http://www.xilinx.com/products/design\\_resources/conn\\_central/protocols/gigabit\\_ethernet.htm](http://www.xilinx.com/products/design_resources/conn_central/protocols/gigabit_ethernet.htm).
27. lwIP. <http://www.sics.se/~adam/lwip/>.
28. Uzun, I., Amira, A., & Bouridane, A. (2005). FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEE Proceedings. Vision, Image, and Signal Processing*, 152(3), 283–296.
29. Elam, D., & Lovescu, C. (2003). *A block floating point implementation for an N-point FFT on the TMS320C55X DSP*. Application Report SPRA948, Texas Instruments, Dallas, Texas, USA.
30. Welch, P. (1969). A fixed-point fast Fourier transform error analysis. *IEEE Transactions on Audio and Electroacoustics*, 17(2), 151–157.



**Chi-Li Yu** (S'10) received the BS and MS degrees in electrical engineering from National Central University, Taiwan, in 1998 and 2000, respectively. From 2001 to 2005, he was an associate engineer in Industrial Technology Research Institute (ITRI), Taiwan. Currently, he is a PhD candidate in School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe. His research interests include high-performance/low-power algorithm-architecture co-design for DSP, image processing, and communication applications. He is a student member of the IEEE.



**Jung-Sub Kim** (S'07) received the BS and MS degrees in electrical engineering from Yonsei University, Seoul, Korea, in 1995 and 1997, respectively, and the PhD degree in electrical engineering from the Pennsylvania State University in 2008. From 1997

to 2003, he was with the Embedded System Laboratory, R&D Center, LG Industrial System, Anyang, Korea. Currently, he is a senior engineer with Architecture Research Laboratory, DMC R&D Center, Samsung Electronics, Suwon, Korea. His research interests include high-performance reconfigurable systems design and reliable circuit design. He is a student member of the IEEE.



**Lanping Deng** received the BS and MS degrees in electrical engineering from Tsinghua University, Beijing, China, in 2003 and 2005, respectively. In 2009, he received the PhD degree in electrical engineering from Arizona State University, Tempe. His research interests include hardware-software codesign, FPGA based accelerator design, and EDA tool design.



**Srinidhi Kestur** (S'10) received the BE degree in Electronics and Communication Engineering from R V College of Engineering, India in 2005. From 2005 to 2006, he was a software engineer in the DSP software group at Analog Devices,

Bangalore. Currently, he is a PhD candidate in Electrical Engineering at Pennsylvania State University, University Park. His research interests are in high-performance reconfigurable systems design for applications in signal processing, vision and high-performance computing. He is a student member of the IEEE.



**Vijaykrishnan Narayanan** is a professor of computer science and engineering at The Pennsylvania State University, University Park. His research interests are in computer architecture, embedded systems, and nanoarchitectures.



**Chaitali Chakrabarti** (SM'02) is a professor of Electrical Engineering at Arizona State University, Tempe. Her research interests are in the areas of low-power embedded systems design and algorithm-architecture co-design of signal processing, image processing, and communication systems.