TRANSPOSE-FREE SAR IMAGING ON FPGA PLATFORM

Chi-Li Yu and Chaitali Chakrabarti

School of Electrical, Computer and Energy Engineering
Arizona State University, Tempe, USA
Email: {chi-li.yu, chaitali}@asu.edu

ABSTRACT

Range-Doppler Algorithm (RDA) and Chirp Scaling Algorithm (CSA) are two widely used Synthetic Aperture Radar (SAR) imaging schemes. Both require multiple transpose operations which increase the total processing time significantly. In this paper, we propose transpose-free flow for both RDA and CSA. This is achieved by modifying the existing flows in order to utilize the access patterns favored by the external memory. As a result, the peak performance of the memory is sustained and the processing time shortened. The proposed Field Programmable Gate Array (FPGA)-based implementation outperforms the existing SAR accelerators; it computes RDA and CSA on data size of $4,096 \times 4,096$ in 323ms and 162ms, respectively.

Index Terms— SAR, FPGA, DRAM, FFT.

1. INTRODUCTION

Synthetic Aperture Radar (SAR) has been widely used in military surveillance, environmental monitoring, and earth resource surveys. From an airborne or a space-borne platform, a SAR system generates high resolution images covering large areas in all weather conditions, day or night. The raw data collected by a SAR system is highly unfocused due to electromagnetic wave scattering and the relative motion between the radar and the earth surface.

Several algorithms have been developed for digital SAR imaging, which include range-Dopper algorithm (RDA) [1] and chirp-scaling algorithm (CSA) [2]. The key kernels of these algorithms are Discrete Fourier transform (DFT), interpolations, and convolutions. Since the algorithms can be highly parallelized, they can be computed efficiently on multiple parallel computing platforms, including Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA). Compared to GPU, FPGA consumes much lower power, which makes it more suitable for SAR image processing on airplanes or satellites.

For real-time SAR imaging, the bottleneck is data transfer between the chip and external memory. SAR images are typically very large, e.g. 4,096×4,096, and must be stored in a external memory, which is usually a Synchronous Dynamic RAM (SDRAM). SDRAM's transfer rate is slow compared to processor clock speeds, and so the performance of SAR imaging systems is determined by the memory bandwidth.

Furthermore, SAR imaging algorithms need to perform computations along row and column directions of a 2D image several times. Because SDRAM only favors row-wise burst access, most SAR processors [3, 4, 5] need to transpose the 2D data before column-wise operations. This is done by transferring column-wise data from the SDRAM to the chip, realigning the column data into adjacent addresses, and storing back to the memory. Then, another transpose operation is required before the next row-operation. Since all SAR imaging algorithms require multiple transpose operations, the timing performance of SAR imaging is worsened significantly.

To eliminate the transpose operation, the method in [6] stores SAR data into a multi-chip SDRAM array and takes advantage of the multi-banking memory organization to get rid of the overhead when accessing column-wise data. However, the design in [6] does not support general SDRAM modules, and significant customization has to be done. Another way to eliminate transpose operations is to increase the locality of data along column direction in a SDRAM module. The method in [7] re-maps the column-wise data into a physical page of SDRAM to increase the access efficiency. These methods can achieve 80% of SDRAM's peak rate for both row and column-wise accesses. However, the data re-mapping before the SAR imaging process needs extra time.

In this paper, we propose transpose-free SAR imaging flows for RDA and CSA. The flows are mapped to a unified architecture which is implemented on an FPGA-based platform. The implementation has superior timing performance, since it avoids transpose operations and utilizes the memory bandwidth efficiently. Simulation results based on the Xilinx ML605 platform show that the RDA and CSA computations with data size 4096×4096 can be completed in 323ms and 162ms respectively. This implementation outperforms existing SAR image accelerators, including FPGA- and GPU-based solutions [7, 8, 9].

The rest of the paper is organized as follows. A brief description of RDA and CSA is given in Section 2. In Section 3, the transpose-free imaging flows for RDA and CSA are

This work is supported in part by the Defense Advanced Research Projects Agency (DARPA) under Grant W911NF-05-1-0248.

proposed. A unified FPGA architecture for RDA and CSA is proposed in Section 4, and evaluated in Section 5. The paper is concluded in Section 6.

2. BACKGROUND

2.1. Range-Doppler Algorithm

In SAR imaging, range-Doppler algorithm (RDA) [1] is widely used. The classical RDA flow consists of four steps: 1. Range Compression; 2. Azimuth Discrete Fourier Transform (DFT); 3. Range Cell Migration Correction (RCMC); 4. Azimuth Compression and IDFT.

Range Compression operates on row-wise data. However, after Range Compression, the data has to be transposed, so that it can be accessed efficiently by the next step, Azimuth DFT. As mentioned earlier, an SDRAM only favors burst access to adjacent addresses, and so a transpose operation is required to realign the data along azimuth direction into contiguous addresses. Similarly, before and after RCMC, which is a computation along range direction, two other transpose operations are required. Thus in a traditional implementation, three transpose operations are required during which all computation units are idle. This results in significant timing overhead. RDA also requires time-consuming interpolation for RCMC, which is both time and hardware-intensive.

2.2. Chirp Scaling Algorithm

Chirp Scaling Algorithm (CSA) [2] is another popular SAR imaging algorithm. It avoids time-consuming interpolations and is computationally less intensive compared to RDA. The processing flow can be divided into 4 steps: 1. Azimuth DFT and Chirp Scaling; 2. Range DFT and Bulk RCMC; 3. Range IDFT and Azimuth Compression; 4. Azimuth IDFT. Similar to RDA, matrix transpose operations are required between Steps 1 and 2 and also between Steps 3 and 4.

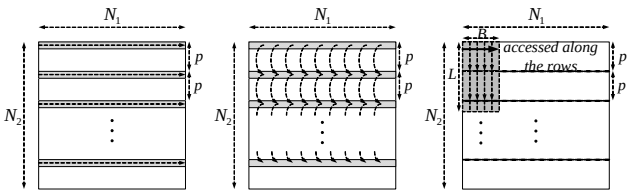
In summary, both RDA and CSA require multiple transpose operations. These operations hog the memory bandwidth and keep it busy. A consequence of this is that the computation units have to wait for the memory data and are idle longer. In the following section, we explain how the transpose operations can be avoided, resulting in significant improvement in the timing performance.

3. TRANSPOSE-FREE SAR IMAGING METHODS

To avoid the transpose operations and exploit the SDRAM access characteristics matrix transposes, we utilize three access patterns as shown in Fig. 1:

• **Pattern 1-a. Row Access:** Rows spaced p rows apart are accessed. It supports row-wise DFT and interpolation.
• **Pattern 1-b. Column Stride Access:** Data is processed across the rows read with Pattern 1-a. This pattern is required for q-point Column Stride DFT [10].
• **Pattern 2. Column Local Access**: Sub-columns are accessed for p-point Column Local DFT [10].

Note that: (1) The accessed data of Patterns 1-a and 1-b are from the same set of addresses in the memory. Therefore, if the adjacent two steps requires these two access patterns, they can be merged into one Step. (2) Patterns 1-b and 2 together can be used to compute full-length ($N_2(= p \times q)$-point) Column DFT. In short, the idea is to decompose the lengthy column DFT into shorter ones, and thus, more adjacent sub-columns of data can be read from or written to the SDRAM. Consequently, long row-wise bursts can be formed, and peak performance of the SDRAM can be sustained throughout the computation. Most importantly, transpose operations are no longer required.



(a) Pattern 1-a. Row Access (b) Pattern 1-b. Column Stride Access (c) Pattern 2. Column Local Access

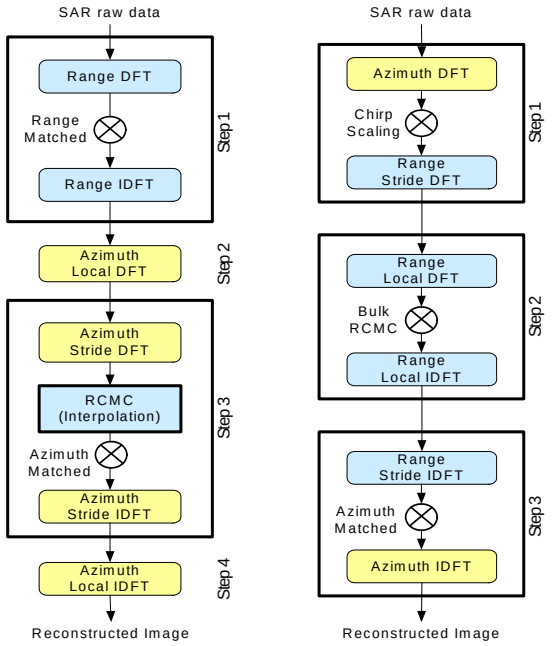
Fig. 1. Proposed data access patterns.

3.1. Transpose-free RDA Flow

Based on the access patterns, we propose a new RDA flow, which is illustrated in Fig. 2(a). Here, we assume that SAR raw data is stored in the external memory in range-major order. This helps in accessing memory data efficiently during computation of Range DFT and IDFT in Step 1. To avoid transpose operations, Azimuth DFT is decomposed into Local DFT in Step 2 and Stride DFT in Step 3. To perform Azimuth Stride DFT in Step 3, the entire row along range direction needs to be loaded onto the local memory. After the DFT, RCMC can be computed along the rows in the same step. To transform the data back to the space domain, the Azimuth Stride and Local IDFT are performed. Note that Azimuth Stride IDFT is computed in Step 3 to avoid extra memory transactions. Compared to the original flow, the new flow still requires 4 steps but eliminates the 3 transpose operations.

3.2. Transpose-free CSA Flow

For CSA, we assume that the SAR raw data is stored in the SDRAM in azimuth-major order. With this data organization, Azimuth DFT or IDFT achieves the highest performance because row data can be retrieved efficiently using long row-wise burst access along rows in SDRAM. To avoid transpose, we decompose the Range (I)DFT into (1) Range Stride (I)DFT and (2) Range Local (I)DFT, and obtain a new process flow for CSA, as illustrated in Fig. 2(b). There are only three major steps now:
Step 1. Since Azimuth DFT can access data with Pattern 1-a, and Range Stride DFT needs to access data with Pattern 1-b, they can be merged into one step. In addition, Chirp Scaling



(a) RDA (b) CSA

Fig. 2. Proposed RDA/CSA flow.

function can be implemented with a multiplication right after Azimuth DFT.

Step 2. Range DFT and IDFT are computed on the same set of data in every iteration, so they are merged into one step. The RCMC function is just a multiplication after Range local DFT.

Step 3. This step is just an inverse operation of Step 1, except for the Azimuth compression function. Range Stride IDFT accesses data with Pattern 1-b, while Azimuth IDFT accesses data with Pattern 1-a, and thus they can be merged into one step.

To sum up, the proposed CSA procedure reduces the number of steps from four to three, while avoiding the two extra transpose operations.

4. UNIFIED ARCHITECTURE FOR PROPOSED RDA/CSA FLOWS

Based on the proposed flows, we design a unified architecture to compute both RDA and CSA on the same platform. As illustrated in Fig. 3, the architecture consists of an FPGA for accelerating SAR computation and an external SDRAM for storing the SAR data. The main components are as follows:

• **Processing-Element (PE) Array:** A PE contains an FFT module for Range and Azimuth (I)DFT. Here, we adopt Xilinx's streaming FFT IP to implement this module. The interpolator supports the computation of RCMC in RDA and can be removed if only CSA is to be supported. It is a pipelined FIR filter with programmable coefficients. The multiplier implements the multiplications of twiddle factors and reference phase functions. The coefficients are pre-computed and stored in a local memory aside the PEs.

• **SDRAM:** We use Xilinx's AXI interface to fetch the input data from SDRAM and sends it to the local memory. The processing elements (PEs) read this data, compute Range/Azimuth (I)DFT and twiddle multiplications, and store the results back to the local memory. Finally, the SDRAM controller reads these results from the local memory and stores them back to the SDRAM.

• **Local Memory on FPGA:** Two local memories consisting of Block RAMs (BRAMs) form a ping pong buffer to overlap the computation and communications between the FPGA and SDRAM.

5. EVALUATION

Our target platform is Xilinx ML605 FPGA board, which is equipped with a Xilinx Virtex-6 XC6VLX240T FPGA and a DDR3-800 SDRAM module. Two configurations have been mapped onto the FPGA: The dual-mode configuration supporting both RDA and CSA and the CSA-only configuration. The occupied resources of both configurations are listed in Table 1. The dual mode configuration includes an 8-tap interpolator in the PE, which requires additional 64 DSP48E1s. As a result, only 4 PEs can fit on the FPGA. The CSA-only configuration does not need the interpolator, so it can accommodate 8 PEs.

Table 1. Hardware resource utilization of the proposed SAR imaging processor on a Xilinx Virtex-6 XC6VLX240T FPGA (Data format: Single precision).

Configuration	# of PEs	Slices	DSP48E1s	BRAMs
Dual mode (with interpolators)	4	47% (17,701/37,680)	73% (560/768)	48% (399/832)
CSA only (no interpolator)	8	78% (29,385/37,680)	79% (608/768)	64% (531/832)

The computation kernel on the FPGA is clocked at 100MHz, and 80% of peak transfer rate of the DDR3-800 SDRAM module is achieved by using Xilinx's AXI interface. The computation times for the two configurations for three data sizes are listed in Table 2. RDA is computationally more intensive because of the interpolation step. To provide the hardware resources required for interpolation in RDA, the dual mode can only house 4 PEs. As a result, CSA takes about 60% longer time on this configuration, compared to the one which only supports CSA and houses 8 PEs. Note that the timing performance of the CSA-only configuration is only constrained by the SDRAM bandwidth, while that of the dual mode configuration is constrained by the on-chip resources.

In Table 3, we compare our implementation with other SAR imaging processors. Zhou [7] also proposed a transpose-free FPGA design for CSA computation. By reordering the SAR data, this implementation minimizes the frequency of opening/closing physical pages of memory, and hence, sustains the performance of the SDRAM. However, it still requires extra time for reordering the data, while our design

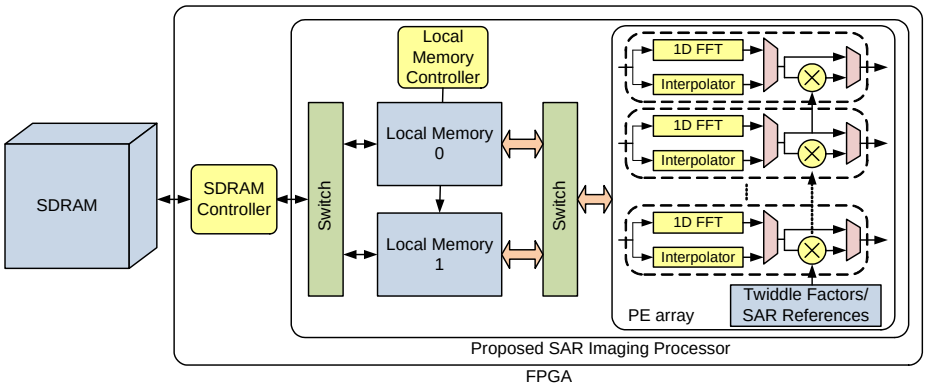


Fig. 3. Proposed architecture for RDA and CSA imaging.

Table 2. Simulated computation times (ms) of the proposed SAR imaging processor.

Data Size	1024×1024			2048×2048			4096×4096		
# of PEs	4 (Dual Mode)		8 (CSA only)	4 (Dual Mode)		8 (CSA only)	4 (Dual Mode)		8 (CSA only)
Algorithm	RDA	CSA	CSA	RDA	CSA	CSA	RDA	CSA	CSA
Step 1	5.45	5.43	3.39	20.78	20.95	13.09	85.19	84.55	52.78
Step 2	3.37	5.49	3.90	13.20	20.86	14.29	53.81	86.78	55.90
Step 3	8.30	5.49	3.36	31.29	21.03	13.15	129.78	85.21	53.10
Step 4	3.31	–	–	13.52	–	–	54.21	–	–
Total	20.43	16.41	10.65	78.80	62.84	40.53	322.99	256.54	161.78

needs no data rearrangement. As for RDA, we compare the proposed design with two GPU solutions. With the efficient transpose-free procedure, our design is able to outperform both GPU-based accelerators [8, 9] in terms of timing. In addition, since the power consumption of the Virtex-6 FPGA is no more than 15 Watts, while the GPUs usually consume more than 70 Watts, our FPGA-based SAR processor is highly power-efficient compared to the GPU-based counterparts.

Table 3. Comparison on the performance of SAR imaging accelerators. (Data size: 4096×4096).

	Device (clock freq.)	External memory	Algorithm	Time (ms)
Proposed, 4PEs (Dual mode)	Xilinx Virtex-6 (100 MHz)	SDRAM (DDR3-800)	RDA CSA	322.99 256.54
Proposed, 8PEs (CSA only)	Xilinx Virtex-6 (100 MHz)	SDRAM (DDR3-800)	CSA	161.78
Liu[8]	nVidia Quadro FX3700 (500 MHz)	GDDR3	RDA	392.00
Ning[9]	nVidia Tesla C2050 (1.15 GHz)	GDDR5	RDA	832.00
Zhou[7]	Altera Stratix-II (128 MHz)	SDRAM (PC-100)	CSA	1070.10

6. CONCLUSION

In this paper, we propose transpose-free flows for two popular SAR imaging algorithms, RDA and CSA. This is done by reorganizing the flows so that the row-wise burst access pattern favored by SDRAM can be utilized. The high transfer rate of the external memory can be sustained, and the computation units are no longer idle. Consequently, our FPGA-based implementation can achieve higher performance than the existing FPGA and GPU-based SAR image accelerators.

7. REFERENCES

[1] I. Cumming and J. Bennett, "Digital processing of seasat sar data," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, Apr. 1979, pp. 710 – 718.

[2] R. Raney and et al., "Precision SAR processing using chirp scaling," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 32, no. 4, pp. 786–799, Jul. 1994.

[3] H. Izumi and et al., "An efficient technique for corner-turn in SAR image reconstruction by improving cache access," in *Proc. of International Parallel and Distributed Processing Symposium*, 2002, pp. 3–8.

[4] Y. Pi and et al., "A SAR parallel processing algorithm and its implementation," in *Proc. of Future Intelligent Earth Observing Satellites Conference*, vol. 1, 2004, pp. 211–214.

[5] M.-M. Bian and et al., "High-performance system design of SAR real-time signal processing," in *Proc. of International Conference on Computer Application and System Modeling*, vol. 12, Oct. 2010, pp. 126–129.

[6] B. Tu and et al., "Two-dimensional image processing without transpose," in *Proc. of International Conference on Signal Processing*, vol. 1, Aug. 2004, pp. 523–526.

[7] J. Zhou and et al., "Window memory accesses method in alternate row/column matrix access systems," in *Proc. of International Conference on Computer Engineering and Technology*, vol. 3, 2010, pp. 201–205.

[8] B. Liu and et al., "An efficient signal processor of synthetic aperture radar based on GPU," in *European Conference on Synthetic Aperture Radar*, Jun. 2010, pp. 1054–1057.

[9] X. Ning and et al., "Multiple-GPU accelerated range-doppler algorithm for synthetic aperture radar imaging," in *IEEE Radar Conference*, May 2011, pp. 698–701.

[10] C.-L. Yu and et al., "Multidimensional DFT IP generator for FPGA platforms," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 755 –764, Apr. 2011.