An Efficient SAR Processor Based on GPU via CUDA

Bin Liu, Kaizhi Wang, Xingzhao Liu and Wenxian Yu
Department of Electronic Engineering
Shanghai Jiao Tong University
Shanghai, P.R. China

***Abstract*—A novel and efficient Synthetic Aperture Radar (SAR) processor is introduced in this paper. This new processor is implemented on the Graphics Processing Unit (GPU). GPU is traditionally used for graphics rendering, but in recent years, it has rapidly evolved as a highly-parallel processor with tremendous computation capability and ultra-high memory bandwidth. The algorithm of the new SAR processor is developed via Compute Unified Device Architecture (CUDA) which is a popular GPU programming environment. The imaging results of the simulated data show that the resolution and the peak sidelobe ratio of this processor agree with the theoretical values. The time of processing on the actual data shows that this processor is more than 18 times as fast as quad-core CPU-based processor using OpenMP. The high efficiency of this GPU-based processor provides a promising way to solve the complicated problems of high resolution SAR systems.**

Keywords- SAR processor; GPU; CUDA;

I. INTRODUCTION

The Synthetic Aperture Radar (SAR) system provides an all-weather remote sensing means and produces high resolution images of the land under illumination of radar beam. Unlike optical sensors, the SAR system needs a post-processing procedure on the data acquired to form the final image [1].

There are lots of algorithms have been proposed since the first SAR system was built. The traditional algorithms include the algorithms of Range-Doppler type (RD) [1], the algorithms of Chirp Scaling type (CS) [2], the ω-k algorithm [3], the algorithms of Extended Exact Transform Function type (EETF) [4], the frequency scaling algorithm (FS) [5] and so on. All of these algorithms are designed to be executed on the CPU and a costly high performance computer is always needed due to the extensive computing during the processing.

Recently, the high rate developing of Graphics Processing Unit (GPU) attracts the attention of researchers from various domains. Originally, GPU is designed for and installed on the display card (i.e. the video card) of personal computer as a coprocessor to provide hardware acceleration of 2-D and 3-D graphics processing and display. To give a smooth and vivid display effect, the up-to-date programmable GPU has a very powerful float computing, parallel processing ability and higher bandwidth with respect to that of CPU. Besides, GPU is much cheaper than CPU. Fig. 1 gives a comparison between the mainstream CPU and Nvidia GPU about the float computation ability and the memory bandwidth [6]. From this figure, it can be seen that GPU is an ideal computation unit to execute the mission of high computation density.

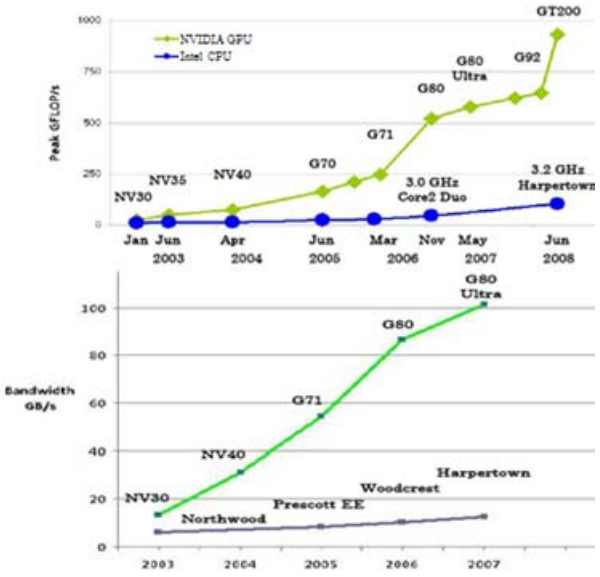

Figure 1. The comparison of CPU and GPU about the float computation ability and memory bandwidth

At the beginning, developers should know a lot about the Application Programming Interface (API) of GPU and the structure of hardware to program GPU to do some general purpose computations [7] [8] [9]. The release of Compute Unified Device Architecture (CUDA) changes this situation. CUDA is an extension of the C language, and provides a familiar programming model of GPU for developers. Developers can implement the general purpose computations on GPU without specific knowledge about the hardware and API of GPU. Many applications that process large data sets with computation intensity, such as SAR data processing, can use CUDA, a highly-parallel programming model, to speed up the computations.

In this paper, an efficient GPU-based SAR processor is introduced, which is implemented by CUDA. This SAR

978-1-4244-4131-0/09/$25.00 ©2009 IEEE

processor takes the advantage of the parallel compute engine in Nvidia GPU to solve the complex computational problems in a more efficient and less expensive way than on the CPU. It provides a promising way to solve the problems of high-performance SAR systems in the future, which produce real time and higher resolution images.

This paper is organized as follows: In Section II, the programming model of CUDA is summarized. The algorithm of the SAR processor is discussed in Section III. The experiments and results based on the simulated SAR data and actual SAR data are shown in Section IV. Conclusions appear in Section V.

II. PROGRAMMING MODEL OF CUDA

The programming model of GPU under CUDA environment is a processor with lots of computational cores. GPU can run and manage a very large number of threads in parallel. These threads executed on GPU are organized into a three-level hierarchy. Threads are grouped in blocks and many blocks are run in a grid. The codes defined in a kernel function are to be executed by each of these threads, and process the data stored in the device memory [10]. CUDA provides shared memory and barrier synchronization. Threads in the same block can cooperate among themselves through barrier synchronization and shared access to a memory space private to the block [11]. Take Nvidia Quadro FX3700 as an example: the maximum number of threads per block is 512, and the maximum size of a grid of thread blocks is 65535.

CUDA threads can access data from multiple kinds of memories during their execution. Table I describes the properties of the CUDA memories [10]. The proper choice of the memories to be used in each kernel function depends on several factors such as the memory size, the speed, whether the memory is read-only and so on.

CUDA supports a subset of the texture hardware operation and makes the texture memory access a very fast speed. An image warping example is presented in Fig. 2, which is done by a kernel function executed on GPU. The kernel function takes the original image data from memory as a texture, and then maps the texture onto a special plane. After that, the image is warped. This example shows that the image warping or unwarping can be done via texture mapping, and this gives a new way to perform range cell migration correction (RCMC).

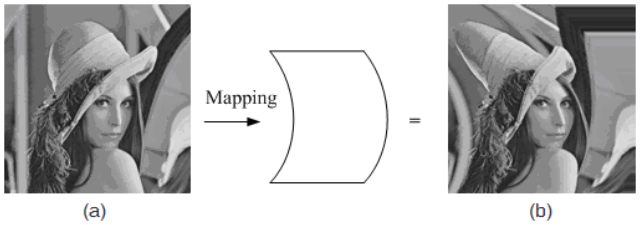

Figure 2. Image warping via texture mapping (a). The image before warping (b). The image after warping

The traditional way to perform RCMC is a range interpolation operation in the range Doppler domain. In this paper, we consider the range compressed SAR data before RCMC as a texture, and use the image warping kernel to map the texture onto a well designed plane, after that, the trajectories of the range compressed data are straightened out and parallel to the azimuth axis.

CUFFT is an FFT library in the CUDA programming environment. CUFFT can achieve a high performance on GPU [12]. We choose this library to perform FFT/IFFT in this paper.

III. ALGORITHM OF THE SAR PROCESSOR

A block diagram of the imaging algorithm implemented on GPU is given in Fig. 3. The data received from the SAR system are referred as signal data, which are processed by the steps as follows [13]:

1. Copy the signal data from CPU to GPU.

2. Compress the range signal by a matched filter set which is highly multithreaded. A range FFT is performed, followed by a range matched filtering, and finally a range IFFT, to complete the range compression.

3. Transform the range compressed data into range Doppler domain via an azimuth FFT.

4. RCMC is performed in the range Doppler domain via texture mapping as discussed in Section II. After this process, the target trajectories at different ranges are straightened out and parallel to the azimuth axis.

5. Azimuth compression is done by azimuth matched filter set which is also highly multithreaded.

TABLE I. PROPERTIES OF CUDA MEMORIES

Memory	Hierarchy	Size	Speed	Read-Only
Register	per-thread	very limited	fast	no
Local	per-thread	limited	slow, not cached	no
Shared	per-block	very limited	fast	no
Global	per-grid	large	slow, not cached	no
Constant	per-grid	limited	slow, but cached	yes
Texture	per-grid	large	slow, but cached	yes

6. Azimuth IFFT transforms the data back to the time domain, resulting in a fully focused complex image.

7. Copy the image data from GPU back to CPU.

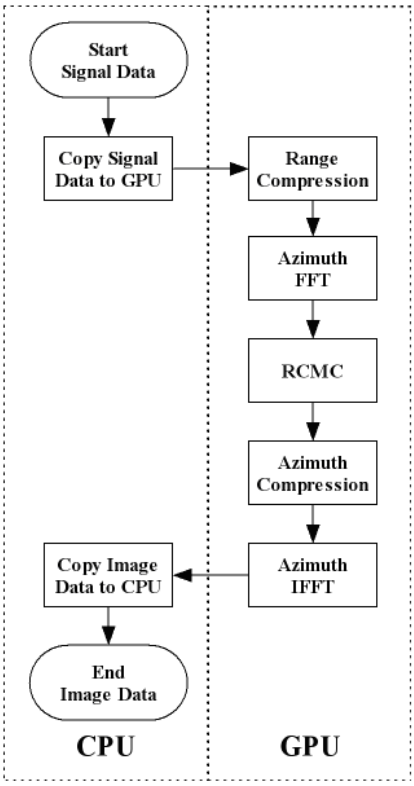

Figure 3. The block diagram of the imaging algorithm of the GPU-based SAR processor

IV. Simulation and Results

To examine the imaging results of the GPU-based SAR processor introduced in this paper, three targets are placed in a scene at the same azimuth time as shown in Fig. 4. Table II shows the simulation parameters.

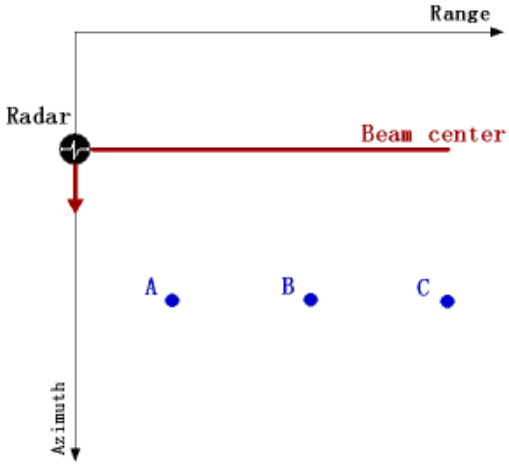

Figure 4. Positions of the three targets used in the simulation

TABLE II. L-band Airborne SAR Parameters Used in the Simulation

Parameter Name	Value
Radar center frequency	1.27 GHz
Transmitted pulse duration	2.5 μs
Range FM rate	32 MHz/μs
Signal bandwidth	80 MHz
Range sampling rate	177.9 MHz
Doppler bandwidth	75 Hz
Pulse Repetition Frequency	111.2 Hz

Fig. 5 shows the range compressed results of the three simulated point targets. These three trajectories show the range migration of the targets in the time domain. An azimuth FFT transforms the data into the range Doppler domain, where the RCMC is implemented by the image warping kernel as discussed in Section II. After that, these trajectories are straightened out and parallel to the azimuth frequency axis, as illustrated in Fig. 6.

After RCMC, a matched filter set is applied to focus the data in the azimuth direction. And the final image is formed after the azimuth compression. Fig. 7 shows the imaging results of the three simulated targets.

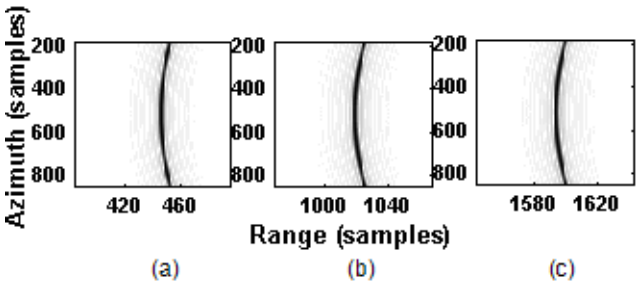

Figure 5. (a), (b) and (c) are range compressed results of targets A, B and C in the time domain

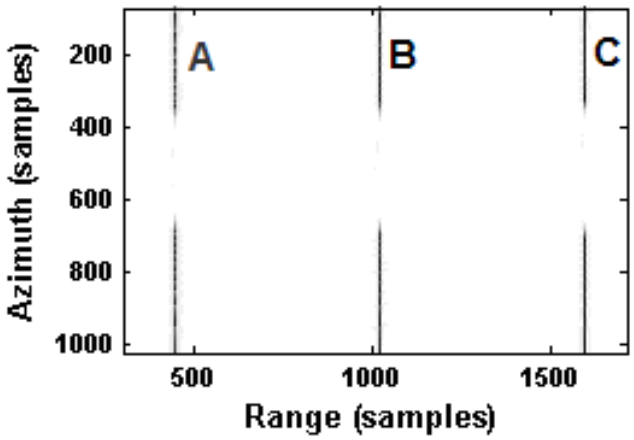

Figure 6. RCMC result in the range Doppler domain

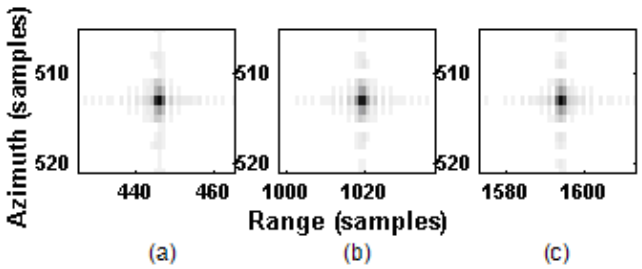

Figure 7. (a), (b) and (c) are the imaging results of targets A, B and C in the time domain

To examine the imaging results in more detail, the range and azimuth profiles of the three targets are shown in Fig. 8 and Fig. 9.

The theoretical range resolution is 1.97 range samples. The theoretical azimuth resolution is 1.31 azimuth samples. Table III shows the measured range and azimuth resolutions, which agree well with the theoretical values.

The measured peak sidelobe ratio (PSLR) is shown in Table IV. The range PSLR of three targets is less than -13dB, agreeing with the value expected from the rectangle window. The azimuth PSLR of -13dB is correct and is due to the antenna pattern.

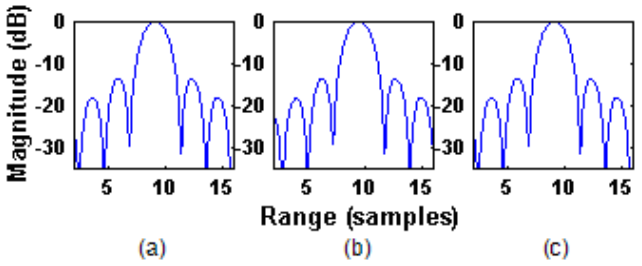

Figure 8. (a), (b) and (c) are the range profiles of the imaging results of targets A, B and C

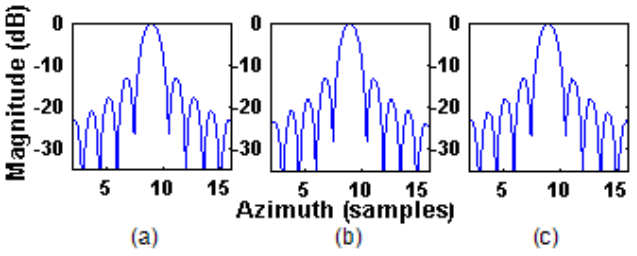

Figure 9. (a), (b) and (c) are the azimuth profiles of the imaging results of targets A, B and C

TABLE III. THEORETICAL AND MEASURED RESOLUTION (SAMPLES)

Direction	Theoretical	Target A	Target B	Target C
Range	1.97	1.991	1.985	1.989
Azimuth	1.31	1.339	1.322	1.341

TABLE IV. MEASURED PSLR (DB)

Direction	Target A	Target B	Target C
Range	-13.75	-13.01	-13.24
Azimuth	-13.76	-13.48	-13.73

To test the efficiency of the GPU-based SAR processor, data from an actual airborne SAR was processed by the CPU-based and GPU-based SAR processors respectively (the CPU is Intel Xeon E5140, and the GPU is Nvidia Quadro FX3700). The size of data is 8192×8192. The used time of CPU-based and GPU-based processors is summarized as follows and schematically illustrated by Fig. 10:

Condition 1: Only one core of CPU is used. The processing time is 164.997 seconds.

Condition 2: Four cores of CPU are used via OpenMP. The processing time is 47.908 seconds.

Condition 3: The GPU-based SAR processor is used. The processing time is 2.556 seconds.

Condition 4: Without counting the memory copy time between CPU and GPU, the processing time of GPU-based SAR processor is 1.566 seconds.

The GPU-based SAR processor introduced and described in this paper shows excellent performance. The imaging results of the simulated SAR data show that the resolution and PSLR of GPU-based SAR processor agree well with the theoretical values. The processing time of actual SAR data shows that the GPU-based processor is more than 64 times as fast as the CPU-based processor without using OpenMP, and more than 18 times as fast as the CPU-based processor using OpenMP.

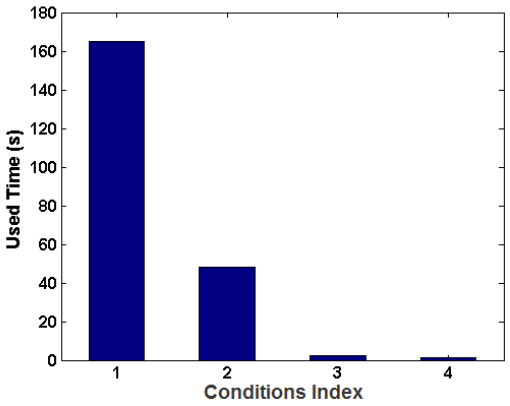

Figure 10. Actual SAR data processing time

V. CONCLUSIONS

In this paper, we introduce an efficient GPU-based SAR processor, which leverages the parallel compute engine in GPU to solve the complex computational problems. This GPU-based processor has the same imaging capability as the traditional

processors implemented on CPU, but uses much less time and costs less.

The high efficiency of the GPU-based processor provides a promising way to address the complicated problems of high-performance SAR processors in two aspects. First, given the GPU-based processor is suitable for compute intensity processing, it can be designed to increase arithmetic intensity to get more precise images. Second, considering the high speed of this processor, the real time SAR systems can be produced.

REFERENCES

[1] I. Cumming and J. Bennett, “Digital processing of Seasat SAR data”, Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '79, vol. 4, 1979, pp. 710-718.

[2] R.K. Raney, H. Runge, R. Bamler, I. Cumming and F. Wong, “Precision SAR Processing Using Chirp Scaling”, IEEE Transactions on Geoscience and Remote Sensing, vol. 32 no. 4, Jul 1994, pp. 786-799.

[3] C. Cafforio, C. Prati and F. Rocca, “Full Resolution Focusing of SEASAT SAR Images in the Frequency-Wave Number Domain”, Proc. 8th EARSel Workshop, 1988, pp. 336-355.

[4] K. Eldhuset, “A new fourth order SAR processing algorithm for spaceborne SAR”, IEEE Transactions on Aerospace and Electronic Systems, vol. 34 no. 3, Jul. 1998, pp. 824-835.

[5] J. Mittermayer, A. Moreira and O. Loffeld, “Spotlight SAR data processing using the frequency scaling algorithm”, IEEE Transactions on Geoscience and Remote Sensing, vol. 37 no. 5, 1998, pp. 2198-2214.

[6] Nvidia, “NVIDIA CUDA Programming Guide Version 2.1”, 2008.

[7] J. Spitzer, "Implementing a GPU-Efficient FFT." NVIDIA course presentation, SIGGRAPH 2003.

[8] Moreland, Kenneth, and Edward Angel. 2003. "The FFT on a GPU." In Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003, pp. 112–120.

[9] Marwan Ansari, "Video Image Processing Using Shaders." Presentation at Game Developers Conference, 2003.

[10] Svetlin A. Manavski and Giorgio Valle, “CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment”, BMC Bioinformatics, 2008, 9(Suppl 2):S10, 2008.

[11] J. Nickolls, I. Buck, M. Garland and K. Skadron, “Scalable parallel programming with CUDA” , Queue, ACM, vol.6 no. 2, 2008, pp. 40-53.

[12] Vasily Volkov and Brian Kazian, "Fitting FFT onto the G80 Architecture", 2008.

[13] I. Cumming and F. Wong, Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation, Artech House, Chapter 6, 2005.