IJCSC Volume 5 • Number 1 March-Sep 2014 pp. 143-146 ISSN-0973-7391

Parallelization of Synthetic Aperture Radar (SAR) Imaging Algorithms on GPU

Bhaumik Pandya, Dr. Nagendra Gajjar
Department of Electronics and Communication Engineering, Nirma University, Ahmedabad, Gujarat, India
12mece31@nirmauni.ac.in,nagendra.gajjar@nirmauni.ac.in

ABSTRACT
The increased demand for higher resolution and detailed SAR imaging builds up a pressure on the processing power of the existing systems for real time or near real time processing. Exploitation of GPU processing power could suffice the increasing demands in processing. The processing of initial SAR systems was based on the principles of Fourier Optics. Lenses provided a real time two-dimensional Fourier transform of the data This document comprises results and analysis of parallelizing Range Doppler and Chirp scaling algorithms for SAR imaging and comparison of computational time over traditional CPU and GPU platform. The results shows that RDA in its essence gives better speed-up than CSA basically due to its less complex manipulations.

Keywords— CUDA, FFT, RDA, CSA, execution time.

1. INTRODUCTION

Synthetic Aperture radar is widely used; especially due its special benefits like all weather, day and night imaging capabilities over optical imaging. It finds applications in environmental monitoring, disaster management, military and defense, remote sensing etc. [5-6] Range Doppler and chirp scaling algorithms are applied to the raw data to produce image in visible format. However, the process is highly cumbersome involving large number of computations and difficult for real time practical realizations.

A further increase in the clock frequency in von Neumann architecture is no longer feasible and the only way to increase the processing power is to switch to alternatives like parallel computing machines. Many existing SAR processors are designed with special DSP processors such as TigerSharc TS201 [4], are in fact very expensive, power consuming and difficult to implement. The availability of technologies like CUDA which help exploiting power of the GPUs, algorithms can be parallelized over such vector machines.

GPU is intended to solve problems involving large data. The processing capabilities of GPU has increased drastically over last decade. For several years programmers used to program GPU using languages like Cg, GLSL and HLSL to program GPU but such languages needed high knowledge of hardware and of Application Programming Interface (API) of the GPU. With the launch of CUDA and its accelerated libraries, the NVIDIA CUDA complier (NVCC) and debugger are available on both Windows and Linux platform. With the windows platform it can be linked with Microsoft visual studio and the facilities of debugging and compiling are available while on Linux it uses NVCC along with GCC complier to generate applications. The availability of tools like Visual Profiler for the GPU accelerated application allows us to timestamp various kernels executed on GPU and analyze the program effectively.

We have optimized range Doppler and chirp scaling algorithms for SAR which provides increased speed up as compared to the speed up given by [7], which uses multiple GPU platform utilizing higher resources. On our part we use a single GPU with a high level of optimization.

The Radar Remote sensing algorithms involve function like FFTs, normalizations and convolution or match filtering in 2 different directions. The basic process i.e. multiplication and accumulation, is usually 32 bit floating point calculations.

2. ALGORITHMS

2.1 Range Doppler Algorithm

A. Data Specifications

The data is generated by sending the reference signal from the satellite and collecting the reflected signals back and transmitting the collected data back to the earth station.

The data under test here consists of 8k samples of reflected signals of 16k samples each. Each sample consists of real and imaginary part.

B. Range Compression

[1]Range compression is done by taking convolution of the reflected signal with the known reference signal in time domain. But in frequency domain it comprises taking 16k point fast Fourier transform (FFT) of each reflected signal and the reference signal. The reference signal is then conjugated. Both vectors- data vector and conjugated reference- are multiplied sample to sample and then an inverse FFT of the resultant vector is done. It is then normalized by dividing it with the total number of FFT points. This process is done for all the 8k reflected signals.

C. Corner Turn or Matrix transpose

Now the 8k x 16k matrix is transposed by turning each column is into row and each row into column. This transposed matrix is then sent for Azimuth Compression.

D. Azimuth Compression

Azimuth compression involves three steps which are performed for 16k rows.

1) Calculating number of azimuth replica points

[1]It involves generation of azimuth replica signal by calculating numbers of azimuth samples for all rows (i.e. 16k rows after taking the transpose). The number of azimuth samples for each row is calculated depending upon parameters like beam width of satellite antenna, velocity of satellite, the distance between the satellite and the location where the signal is incident, frequency of operation and chip rate.

2) Calculating replica signal

Once the number of samples is calculated the replica signal is generated which is an exponential function of pi, chip rate and square of the pulse repetition frequency.

3) Match Filtering

Now the convolution in the time domain is carried out i.e. conjugated multiplication in frequency domain with 8k FFT points. This process is carried out for all the 16k rows. Then inverse FFT and normalizations are carried out.

E. Back Transpose and absolute value

The transpose of the resultant matrix is taken and absolute value of each sample is calculated and a bit file is written. The bit file can be imported to an image viewer.

2.2 Chirp Scaling Algorithm

Below Figure [12] shows block diagram of the chirp scaling algorithm.

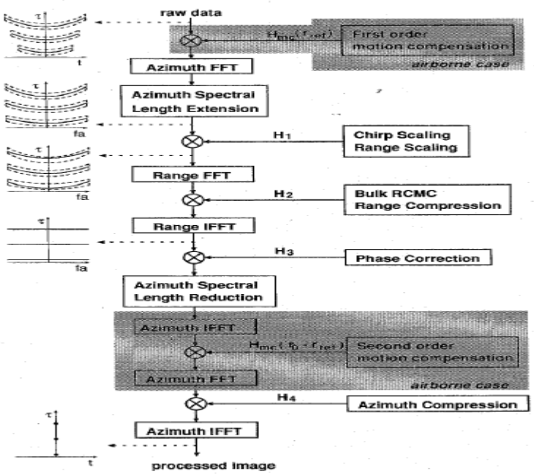

Fig1: Block Diagram of Chirp Scaling Algorithm

3. EXPERIMENTAL SETUP

The workstation consists of core i7 CPU and 32 GB of RAM memory with 500 GB of disk memory. The CPU-GPU link is of PCIe x16 Gen2 and power supply is 650W switch mode power supply (SMPS).

The GPU device used in the experiment is NVIDIA GTX770. [2]The specifications are as listed below:

- CUDA Cores: 1536
- Frequency of cores: 1.05 GHz
- Double precision[9] floating point performance (peak): 134 Gflops.
- Single precision floating point performance (peak): 3.21 Tflops.
- Total dedicated memory: 4GB GDDR5
- Memory speed: 1.11 Ghz
- Memory interface: 256-bit
- Memory bandwidth: 224.3 Gb/s
- System interface: PCIe x16 Gen3
- ECC memory[10]: Offers protection of data in memory to enhance data integrity and reliability for applications. Register files, L1/L2 caches, shared memory and DRAM all are ECC
(Error Checking & Correction) protected.
- Parallel Data Cache: This includes a configurable L1 cache per SMX block and a unified L2 cache for all of the processor cores.
- Asynchronous transfer: Turbochargers system performance by transferring data over the PCIe bus while the computing cores are crunching other data

Software platform includes

- Microsoft Visual Studio 2010
- Nvidia Cuda Toolkit 5.5 [11]
- Nvidia Parallel Nsight 3.1

4. PARALLEL IMPLEMENTATION

Each step in itself involves large portion of instructions that can be parallelized. Below are the steps for implementing RDA on GPU:-

- CUDA Memory Copy (Host to Device) copies the complex data and the range compression replica signal to the device over PCI express.
- CUDA FFT kernel for range compression uses cufft library for implementing complex to complex FFT.
- Range Compression match filter kernel does match filtering of the data samples.

- Cuda IFFT post range compression computes inverse FFT using cufft library
- Matrix transpose and normalization kernel normalize the data vector after inverse FFT and take matrix transpose.
- Cuda FFT for azimuth compression computes FFT of transposed matrix using cufft library.
- Azimuth replica generation kernel generates the azimuth replica signal in time domain using complex exponential function.
- Cuda FFT for Azimuth replica performs FFT of the replica signal using cufft library.
- Azimuth match filtering kernel does match filtering in the azimuth direction of the data vector.
- Cuda IFFT post azimuth compression kernel computes inverse FFT after azimuth compression
- Matrix transpose and normalization kernel normalize the data vector after inverse FFT post azimuth compression and take matrix transpose.
- Cuda memory copy (Device to host) copies the computed image vector to the host memory.

Steps for applying CSA on GPU:-

- All the constants need to be used into the algorithm have to be defined in the beginning.
- We need to store the data into some variable by firstly reading it and making a matrix of that.
- Azimuth FFT does FFT of all data vectors into the azimuth direction.
- Then we need to multiply the data with H1 Function in this way range scaling will be done.
- Range FFT does FFT of all data vectors into the range direction
- Then we need to multiply the data with H2 function and in this way Bulk RCMC is performed.
- Range IFFT will transform the data back into the range time azimuth frequency which is range Doppler domain.
- Then we need to multiply the data with H3 function which indeed does the Angle Correction
- Then we need to multiply data with the H4 function which indeed does the Azimuth Compression.
- Azimuth IFFT which transforms the data back into
- Visualization of results

All these kernels are executed sequentially on the device when called from the host side. In addition to this the kernel computations are done in place ensuring efficient use of device memory.

5. OPTIMIZATION

For the purpose of achieving higher throughput and peak performance various optimization techniques are used. It ensures 100% utilization of the GPU cores and minimum GPU ideal time during the program execution.

A. Block Size and Grid size

Due to linear nature of each reflected sample, a single dimension block is preferred containing 1024 threads per block. As the number of threads is a multiple of 32, the efficiency is higher. The wrap schedulers schedule 32 threads per wrap in the device. [3]Hence the number of threads being a multiple of 32 ensures that no core would remain free during any of the wrap.

The grid is also taken in single dimension as an array of blocks and is decided by the number of total data size and number of threads per block.

B. Shared memory per block

The access to the global memory of the device is relatively slow compared to the shared memory per block. [3]The access to the shared memory is 10x faster compared to the global memory. But the amount of shared memory is limited by the size of the cache memory; hence too much use of the shared memory restricts the optimization.

But optimized use of shared memory speeds up the kernel execution thus reduces the execution time. The optimized amount of the shared memory varies from device to device and their computation capabilities.

C. Registers per thread

The number of registers per thread also controls the performance of the processing units. [3]Large number of registers per thread drastically reduces the performance but as the registers access is 100x faster than the global memory access and so the optimized use of registers increases the performance.

D. Use of constant memory

The constant memory is located in the cache and is 10 x faster than the global memory. The reference signal is usually placed in the constant memory and hence increases the performance.

E. Use of special function units (SFU) available in architecture

The Nvidia Fermi architecture contains special hardware units to compute mathematical functions like sine and cosine. The hardware functions calculates up to 8 terms of the required trigonometric series as compared to the software functions which compute up to 20 terms, but when the demand for accuracy is of single precision floating point the SFU can provide high performance compared to the software functions.

F. Use of CUFFT and NPP library of NVIDIA

The use of highly accelerated libraries like CUFFT and NPP available with CUDA toolkit provides a high level of optimization. The CUFFT library has functions for implementing 1D, 2D, 3D FFTs. The NPP library has functions for signal processing like convolution, scaling, shifting etc.

6. RESULTS AND ANALYSIS

In this section we intend to discuss the results of this parallel implementation. Section A. shows the CPU and GPU comparison. which are computed for image of resolution 4096 x 4096.

G. Comparison of execution time of CPU and GPU

The table shows the execution time in seconds of various image resolutions for RDA and CSA . As the amount of data increases, the speed up also increases. This is due to two basic reasons.

- The overhead of calling the GPU kernel is divided among a large data.
- The percentage of GPU idle time which is out of the total execution time gets reduced.

Table 1: execution time of CPU and GPU platform for RDA

Image Size	4096 x 4096	8192 x 4096	8192 x 8192	16384 x 8192
CPU Time (Seconds)	238.97	350.940	853.896	2108.639
GPU Time (Seconds)	0.593	0.858	1.544	2.839
Speed up	403x	409x	553x	748x

Table 2: execution time of CPU and GPU platform for CSA

Image Size	4096 x 4096	8192 x 4096	8192 x 8192	16384 x 8192
CPU Time (Seconds)	256.65	363.92	923.23	2403.51
GPU Time (Seconds)	0.731	1.156	2.142	3.325
Speed up	351x	314x	431x	722x

7. CONCLUSION

Range Doppler and Chirp scaling both are reasonable approaches for RADARSAT data to its precision processing. While Chirp scaling algorithm is slightly more complex and takes more time in its implementation but promises better resolution in some extreme cases. Chirp Scaling algorithm is more phase preserving and it avoids computationally extensive and complicated interpolation used by the Range Doppler Algorithm.

REFERENCES

1. Curlander, J.C. and McDonough, R.N., 199 1, Synthetic Aperture Radar - Systems and Signal Processing, J. Wiley & Sons, USA.
2. Nvidia Tesla C2070 Whitepaper.
3. Programming Massively parallel processors – David Kirk, Wen-mei Hwu
4. BabuRao Kodavati, Jagan MohanaRao malla, Tholada AppaRao, T.Sridher, "Development of moving target detection algorithm using ADSP TS201 DSP Processor", International Journal of Engineering Science and technology Vol.2(8),3355-3363,2010
5. M. Soumekh, "Moving target detection in foliage using along track monopulse synthetic aperture radar imaging", IEEE transactions on Image Processing, Vol. 6, Issue: 8, p 1148 – 1163, Aug 1997.
6. Ritesh Kumar Sharma , B.Saravana Kumar, Nilesh M. Desai, V.R. Gujraty, "SAR for disaster management ", IEEE Aerospace and electronic system magazine, v23, n 6, p 4-9, June 2008
7. Xia Ning, Chunmao Yeh, Bin Zhou, Wei Gao, Jian Yang "Multiple-GPU Accelerated Range-Doppler Algorithm for Synthetic Aperture Radar Imaging"
8. http://en.wikipedia.org/wiki/PCI_Express
9. http://en.wikipedia.org/wiki/Double-precision_floating-point_format
10. http://en.wikipedia.org/wiki/ECC_memory
11. http://developer.nvidia.com/cuda/cuda-downloads
12. Alberto Moreira,Josef Mittermayer and Rolf Scheiber "Extended Chirp Scaling Algorithm for Air- and Spaceborne SAR Data Processing in Stripmap and ScanSAR Imaging Modes" , IEEE Transactions On Geoscience And Remote Sensing ,Vol. 34, No. 5,pp.1123-1133,Sepetember 1996.
13. I.G . Cumming and F.H. Wong," Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation" Artech House Publishers, first edition, 2005.