Parallel processing for SAR image generation in CUDA – GPGPU platform

Prajakta Tapkir, Saurabh Thakur, and C. Bhattacharya
Electronics Engineering Dept, DIAT, Pune - 411025, India
tapkir.prajakta@gmail.com

Abstract:

High resolution imagery from synthetic aperture radar (SAR) video data requires numerical computations of the order of gigaflops (GFLOP). The computational burden increases with the image size and the amount of input raw video signals. General purpose graphic processor units (GPGPU) can play a pivotal role in parallel processing the raw video data to generate SAR imagery in a much faster process. In this paper, we show utilization of GPGPU processors in compute unified device architecture (CUDA®) environment for implementation of a parallel algorithm for SAR image generation.

Keywords: SAR, GPGPU, CUDA, Parallel Processing.

I INTRODUCTION

Satellite-borne synthetic aperture radar (SAR) video raw data requires large number of computational procedures to generate an image. For example, an earth observation satellite RADARSAT-1 in fine beam mode of operation with a C-band SAR sensor as payload is 3.6Km in azimuth and 50Km in range [1]. The scale of numerical computations is of the order of gigaflops (GFLOPs) that may be appreciated from the fact that ground footprint of radar backscattering results in an image with 5.26m spatial fine resolution in azimuth and 7.2m ground range resolution.

In order to improve efficiency in SAR video data processing, one way is to distribute the computation load among several processing elements (PEs). There are several bottlenecks to such distributed processing such as dependencies on the configuration of PEs, their interconnectedness, message passing protocols, and data transmission bandwidth among PEs [2]. Although this enhances the speed of computation they do not include any parallel signal processing algorithms for SAR image formation.

However, in cases where the structure or flow of the algorithm does not map directly onto the architecture, we need to develop new methods to extract parallelism, and correspondingly improve performance. General purpose graphic processor unit (GPGPU) is architecture for high performance computing that uses graphics processing units (GPU) for multicore processing of data. GPGPU exhibit two properties such as data parallelism and intensive throughput of data. Data parallelism implies processor can execute operations on different data elements simultaneously. On the other hand, throughput intensive process means an algorithm is going to process lots of data elements whose execution will be in parallel. GPGPU platforms available from NVIDIA, ATI, and Intel have a large number of processors (of the order of a few hundred) structured to allow multiple threads of execution. Architecture of GPGPU is organized into an array of highly threaded streaming multiprocessors (SMs). It has number of streaming processors (SPs) that share control logic and instruction cache [3]. Comparative survey of latest GPGPUs, e.g., Fermi and Kepler series from NVIDIA shows that as the number of cores per SMs increases, core speed also gets increased. The number of cores in Tesla C2075 is 448 whereas Tesla K20C Kepler series has 2496 cores. Therefore the memory bandwidth also increases in advanced generation of GPGPUs.

In this work, we utilize compute unified device architecture (CUDA) as the programming platform. CUDA includes such a programming model along with hardware support that facilitates parallel implementation. It allows developers to harness the underlying massively parallel compute engine with C-programming language. An algorithm is broken into CUDA threads those are scheduled for concurrent execution. We describe a parallel algorithm of SAR image processing that inherently exploits parallelism in execution. Through this work we have developed *two levels of parallelism*, one is devising a parallel algorithm for SAR signal processing, and another is an actual implementation and execution in GPGPU.

In the following sections we describe working of the parallel algorithm that treats SAR data vectors in segmented blocks and does processing over the blocks simultaneously. We show overall computational speed improvement when this segmented or block processing algorithm is executed with CUDA programming in GPGPU. Finally, we show actual SAR image generation from RADARSAT-1 raw video data and demonstrate improved performance in computational speed.

II ALGORITHM FOR PARALLEL PROCESSING OF SAR IMAGE GENERATION

SAR signal processing works on the principle of matching received signal phase with transmitted signal phase. This can be done by fast Fourier transform (FFT) based techniques. Traditional brute force correlation causes serious overhead on the latency of the processor for large amount of received data. We have developed an algorithm for SAR image generation from RADARSAT-1 raw signal data which inherently executes parallelism for matched filtering of block data vectors. In this algorithm matched filter implementation can be done by dividing the impulse response function of matched filter into number of blocks of equal lengths. To produce final output, partial convolution results from Fourier transform domain is shifted and summed up.

Let us consider $y(n)$, $x^*(-n)$ to be the received data vector and the matched filter vector respectively. Matched filter is the conjugate of time-reversed transmitted complex linear FM chirp envelope $x(n)$ of finite duration T sampled at the rate of B_T, the transmitted bandwidth. These finite vectors $x(n)$, $y(n)$ are made into blocks respectively in R and S numbers of non-overlapping data blocks shown in Fig. 1, each block of being of M samples.

$$x(n) = \sum_{i=0}^{R-1} x_i(n)\ ,\ y(n) = \sum_{j=0}^{S-1} y_j(n) \tag{1}$$

where $x_i(n) = x(n)$, $iM \le n \le (i+1)M-1$, $y_j(n) = y(n)$, $jM \le n \le (j+1)M-1, R < S$. Matched filtering is a complex correlation between the block vectors. Summation of outer product of blocks of $x_i(n)$ and $y_j(n+l)$ for a lag of l samples produce identical results as in the correlation of $x(n)$, $y(n)$.

$$r(l) = \sum_{j=0}^{S-1}\sum_{i=0}^{R-1}\sum_{n=0}^{2M-|l|-2} y_j(n+l)\, x_i \tag{2}$$

Time domain correlation $r_{ij}(l)$ between any two blocks $x_i(n)$,$y_j(n+l)$ in (2) is implemented by FFT in transform domain.

$$r_{ij}(l) = IDFT\,[X^*_i(m)\, Y_j(m)],\ 0 \le m \le (2M-1) \tag{3}$$

The first M samples in (3) account for $r_{ij}(l)$ with one zero padding for $l < 0$, the rest M samples are for $r_{ij}(l)$ with $l > 0$ as shown in Fig 1. The first outer sum in (2) is the partial correlation of R blocks of $x(n)$ with each block of $y_j(n+l)$ depending on the shift index l. The outermost sum over all blocks of $r_{ij}(l)$ is realized by operator matrix $\boldsymbol{A}$. It is of size $(R+S) \times 2RS$; the factor 2 accounts for two sided even sequence of the correlation output. The non-zero elements of $\boldsymbol{A}$ are unity. The number of column elements being unity depends on the block segments to be added in the transform operation. For example, we take the case where the data blocks are divided in two ways as shown in Fig. 1(a) and (b), Fig. 2(a) and (b) respectively.

i) Two blocks of matched filter vector $x(n)$ and two blocks of received data vector $y(n)$.
ii) Three blocks of matched filter vector $x(n)$ and four blocks of received data vector $y(n)$.

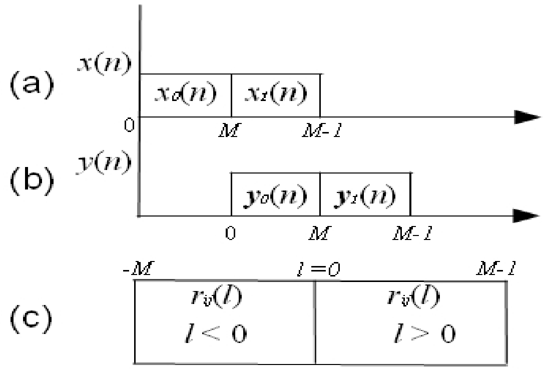

Fig 1. Block correlation algorithm; (a) two filter data blocks, (b) two received data blocks, (c) correlator output vector block.

In case (i), for R to be 2 and S to be 2, the operator matrix $\boldsymbol{A}$ is given by,

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}_{(4 \times 8)}$$

Depending on the position in the lag sequence, the block segments of $r_{ij}(l)$ are added up. As shown in Fig 1(c), the lag sequence is divided in two parts. The non-zero elements of a matrix $\boldsymbol{B}$ of size $2RS \times (R+S)M$ are the subblocks of $r_{ij}(l)$ as result from output of(3). Continuing with example shown in Fig 1 (a) and (b),

$$\boldsymbol{B}(1,1{:}M) = r_{12}(-l)$$
$$\boldsymbol{B}(2,1{:}M) = r_{12}(l)$$
$$\ldots$$
$$\boldsymbol{B}(8,M+1{:}2*M) = r_{21}(l)$$

The final complex correlation coefficients can be found as block vectors of length M in the rows of the transformed matrix, $\boldsymbol{C}$ as

$$\boldsymbol{C} = \boldsymbol{A}\,\boldsymbol{B} \tag{4}$$

Matched filter output of the block correlation algorithm is derived from the non-zero elements of rows in $\boldsymbol{C}$ for samples indices $l > 0$ as filtering is a casual process here.

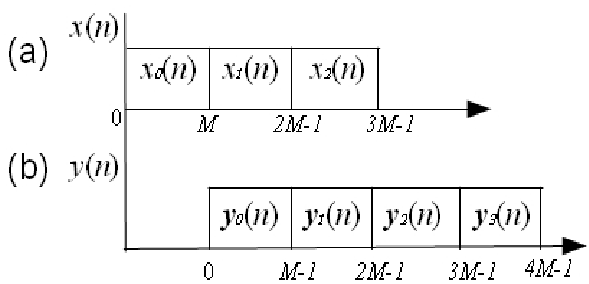

Fig 2. Block correlation algorithm; (a) three filter data blocks, (b) four received data blocks.

In case (ii), for R to be 3 and S to be 4, the operator matrix $\boldsymbol{A}$ is given by,

$$A = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}_{(7 \times 24)}$$

$$\boldsymbol{B}(1,1{:}M) = r_{14}(-l)$$
$$\boldsymbol{B}(2,M+1{:}2*M) = r_{14}(l)$$
$$\ldots$$
$$\boldsymbol{B}(24,M+1{:}2*M) = r_{31}(l)$$

Similarly as explained in case (i), block correlation matrix $\boldsymbol{B}$ derived as shown above. Final complex correlation coefficients were found in the rows of the transformed matrix $\boldsymbol{C}$ for case (ii) as well.

A. Comparison of Computational Load

For traditional brute force correlation, the computational requirement is one complex vector multiplication and one inverse transform. Consider the vector $X^*(m)$ is pre stored and if $NFFT = 2^n > L + F - 1$, the nearest radix-2 FFT length, then the no of complex multiplications in this is

$$Cmul1 = 2(NFFT/2 \log_2 NFFT - NFFT/2) + NFFT = NFFT \log_2 NFFT \quad (5)$$

In block correlation algorithm, consider $2M = NFFT/2^m$, hence the complex multiplication involved in computing is

$$Cmul = 2\left[NFFT / 2^{(m+1)} \log_2 (NFFT / 2^{(m+1)})\right] + NFFT/2^m.$$

If we consider, $L = NFFT/2$, then $S = 2^m$; total number of complex multiplication in block correlation algorithm is

$$Cmult2 = S.\ Cmul = NFFT \log_2 NFFT - NFFT_m. \quad (6)$$

Here, $NFFT_m = NFFT \log_2 NFFT$ that grows substantially with the received data vector becoming longer. Cmult2 shows saving in numerical complex computation by block correlator algorithm.

Consider the two cases to verify the computational load of complex multiplications as the number of received data blocks increases as shown in Fig 3.It is shown that with increase in the number of received data blocks that is increasing the length of received data vector $y(n)$ the number of complex multiplication decreases because of parallel algorithm in (6).

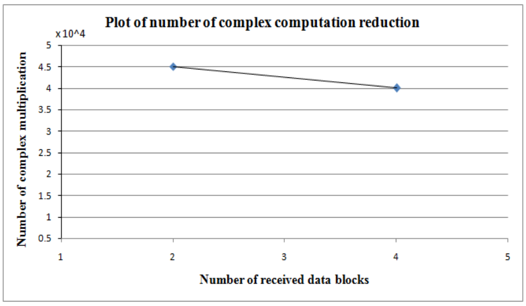

Fig 3. Comparison of computation load of complex multiplications.

III IMPLEMENTATION ON CUDA-GPGPU PLATFORM

CUDA programs consist of a number of C functions called kernels that are to be executed on devices as threads and for that, data stored in the device memory is being processed. CUDA provides shared memory and barrier synchronization [4]. For example, Nvidia Quadro FX 1800 having 512 maximum number of threads per block and 64 processing core.

A GPGPU device has its own memory system including the global memory (large but slow), constant and texture read-only memories providing reduction of memory latency. Each SM has also a 16 kB of fast shared memory that can be used for sharing data among threads within a block.

A. Parallel Algorithm Implemented in CUDA-GPGPU

The SAR raw data samples forming a signal vector are processed using CPU - GPGPU combination for case (i) in section 2 in the following manner:

- Transmitted and received signal vector blocks are partitioned into TX1 & TX2 and RX1 & RX2 of equal length. These four individual blocks are copied from CPU to GPU on which FFT computation are performed using CUFFT CUDA [5] library.
- After FFT operation on each received signal block, it is multiplied with both FFT output of transmit vector i.e. TX1 & TX2 using multiple threads followed by IFFT on the multiplication output.
- The four individual result obtained from IFFT in GPU is copied back to CPU.
- These individual result obtained are summed up to get final result.

A block diagram of the parallel algorithm implemented on GPU is given in Fig. 4.

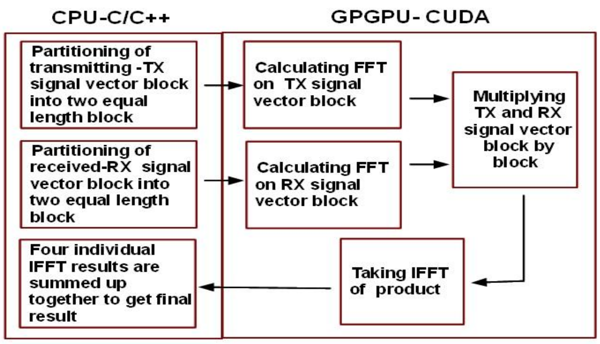

Fig 4. Block diagram of the parallel algorithm implemented on GPGPU for case (i) in section 2.

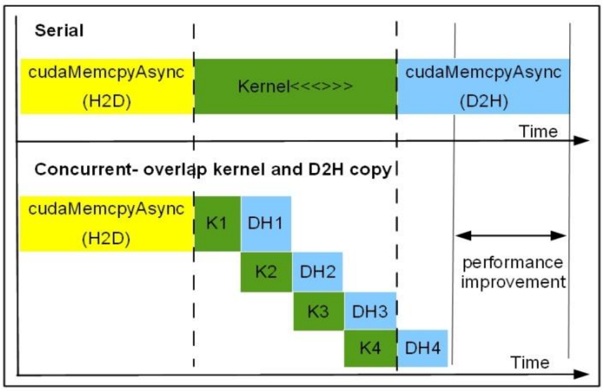

Fig 5. Serial and Concurrent implementation of algorithm in CUDA- GPGPU

B. Radar Filter Implementation

The radar filter for the vectors $y(n)$, $x^*(-n)$ without the parallel algorithm is implemented on CPU having Intel® Xeon® Processor E5-2620 with 2.20GHz processing speed and on GPGPU using CUDA on a GeForce FX1800. The computational time of CPU-based and GPU-based processors is summarized in Table 1 and graphically illustrated by Fig 6. It can be seen clearly that substantial computational speed improvement is achieved using GPGPU as number of FFT sample points goes on increasing with large length of received vector.

Table 1 Summary of Computational Timing in C/C++ and CUDA.

Computational timing for execution of Radar filter in C/C++ and CUDA				
NFFT	C/C++(ms)	CUDA(ms)	Difference	%
	Xeon E5-2620 CPU	FX 1800 - 64 Cores GPU		
1024	0.30	0.40	-0.10	-33.33
2048	0.80	0.50	0.30	37.50
4096	2.00	0.90	1.10	55.00
8192	4.30	1.20	3.10	72.09
16384	8.70	2.50	6.20	71.26

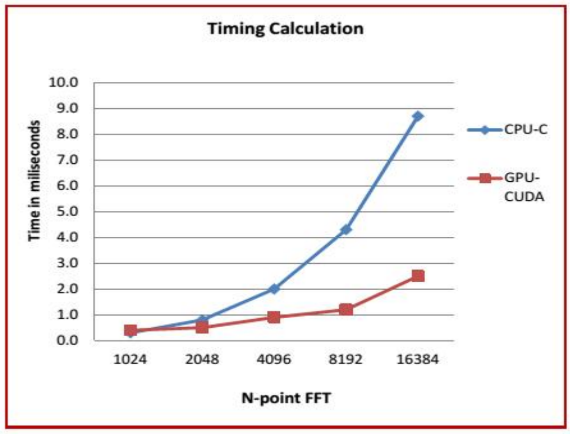

Fig 6. Matched Filter Computational timing

IV SAR IMAGE GENERATION IN CUDA-GPGPU PLATFORM

Signal processing for SAR image formation is done mainly in two steps, the first one is processing of the input raw video data in range dimension followed by processing the range image processed data in the azimuth dimension.

In this paper for SAR data processing, video data block of size (1024 × 2048) and transmit data vector of size (1349×1) are first read into CPU. Each block is of 512 samples is formed by partitioned transmitting data vector block. Video data is rearranged in S = 4 partitioned data blocks as mentioned in section II above. In all, twelve blocks of partitioned correlation operations are to be performed in this example for every range line of video data.

After partitioned data blocks, FFT related operations are performed on GPU as show in Fig. 4. Kernels are created to perform the micro-operations shown in the Fig 5. For optimum usage of on-chip memory in SP devices and to reduce latency in communication from CPU to GPU following optimization are done:

- Threads are identified corresponding to the algorithms those are computationally intensive (e.g., data conversion, FFT, complex transpose, etc.)
- To reduce time in host-device memory transfer prudent memory usage and data transfer are adopted
- The efficient utilization of resources in SMs of GPU is made by dynamic assignment of thread blocks.

For the matrix multiplication of complex valued signal, CUDA CUBLAS routine is used in GPGPU. Every range line processed row data is returned to the host memory space. After completion of processing in range dimension, processing in azimuth dimension starts. Azimuth signal processing executes almost in a similar way as range signal processing is done.

TABLE II
RANGE PROCESSING PERFORMANCE FOR (1024 x1024) BLOCK.

CPU/GPU	Core Details	Timing(sec)	Speedup
AMD Athlon X2 Dual Core	1	45.337	1x
Fx1800	64	19.50	2.32x
Tesla GeForce GTX 275	192	1.8751	24.17x

TABLE III
AZIMUTH PROCESSING PERFORMANCE FOR (1024 x 2048) BLOCK

CPU/GPU	Core Details	Timing(sec)	Speedup
AMD Athlon X2 Dual Core	1	50.065	1x
Fx1800	64	36.60	1.368x
Tesla GeForce GTX	192	3.725	13.44x

TABLE IV
COMPLETE PROCESSING PERFORMANCE FOR (1024 x 2048) BLOCK

CPU/GPU	Core Details	Timing(sec)	Speedup
AMD Athlon X2 Dual Core	1	95.00	1x
Fx1800	64	36.60	2.6x
Tesla GeForce GTX	192	5.563	17.07x

The performance of SAR image generation in different GPGPU platforms such as Fx1800, Tesla C870 are compared with performance of workstations with AMD Athlon(tm) 64 X2 Dual Core Processor 4600 at 2.411GHz. The accelerated speedup in execution of range, azimuth, and complete image generation are shown in Table II-IV for a block of (1024 × 2048) RADARSAT-1 SAR video complex data. The final RADARSAT SAR magnitude image chip generated by the CUDA flow diagram is displayed in Fig 7.

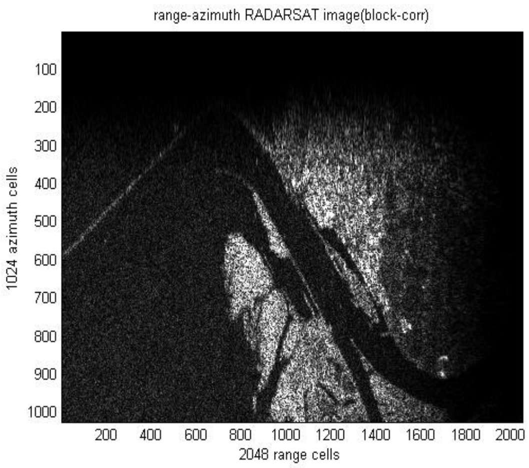

Fig.7 Processed SAR image from RADARSAT-1 video data in CUDA programming.

CONCLUSION

Parallel processing of raw, video SAR data in high performance computing platforms such as GPGPU is shown in the paper. The programming environment is CUDA that allows direct access to GPGPU. Detailed analysis of speed performance in SAR image generation is shown in the paper.

ACKNOWLEDGEMENT

The authors acknowledge the support from Vice Chancellor, DIAT, Pune received in the form of in-house project grant to execute the work in the paper.

REFERENCES

1. I. G. Cumming and F. H. Wong , Digital Processing of Synthetic-Aperture Radar Data: Algorithms and Implementations, Artech House, 2005
2. J. Suh , U. Monte, and V. K. Prasanna, Parallel implementation of synthetic aperture radar on high performance computing platforms, Proc. IEEE-ICAPP 97, 3rd Int. Conf. on Algorithms and Architectures for Parallel Processing, pp. 557 - 570, Dec 1997.
3. J. Nickolls, I. Buck, M. Garland and K. Skadron, Scalable parallel programming with CUDA , Queue, ACM, vol.6, no. 2, pp. 40-53, 2008.
4. D. B. Kirk and W. W. Hwu, Programming Massively Parallel Processors A Hands-on Approach, NVIDIA, 2010.
5. CUFFT library, NVIDIA, Oct 2012.

BIO DATA OF AUTHOR(S)

Prajakta R.Tapkir was born in Pune, India, in 1990. She received her Bachelor's degree in electronics and telecommunication engineering from University of Pune, India, in 2012. She is currently working as a junior research fellow in Defence Institute of Advanced Technology (DIAT), Pune. Her research interests include image, SAR signal processing, embedded system.

Saurabh Thakur was born in Nagpur, India, in 1986.He is working as senior research fellow (SRF) in Defence Institute of Advanced Technology (DIAT), Pune. He has completed his bachelor in '09 and master in '11engineering from University of Pune (UOP).His area of interest is High Performance Computing (HPC), Image and SAR Signal processing, Wireless Sensor Network.

Dr. C. Bhattacharya (M'01) of the Ministry of Defence, Government of India. Prior to this he was a scientist with Defence Electronics Applications Lab (DEAL), Dehradun, India since December 1996, and he headed the Sensor Signal Processing Section for millimetre wave sensors there. His research areas include signal processing for various radar and sensor configurations with particular interest in SAR data processing, stochastic processes, and Bioinformatics.