CONTACT NAME
Vishal Mehta: vishal.m8791@gmail.com

Parallel Implementation of Range Doppler Algorithm for Synthetic Aperture Radar on GPU

Prof. Nagendre Gjajar[1], Vishal Mehta[1],Nilesh M. Desai[2]
[1]Institute of Technology, Nirma University, Ahmedabad, India
[2]Space Application Center, Indian Space Research Organization, Ahmedabad, India
nagendra.gajjar@nirmauni.ac.in, vishal.m8791@gmail.com, nmdesai@sac.isro.gov.in

Abstract

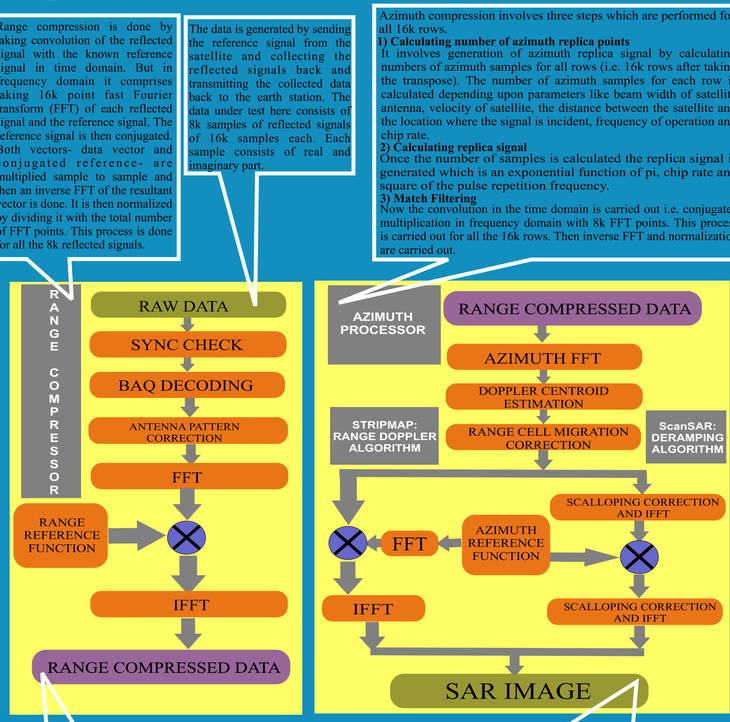
The increased demand for higher resolution and detailed SAR imaging builds up a pressure on the processing power of the existing systems for real time or near real time processing. Exploitation of GPU processing power could meet the increasing demands in processing. This poster comprises results and analysis of parallelizing Range - Doppler algorithm for SAR imaging and comparison of computational time over traditional CPU and NVIDIA TESLA platform.

Algorithm and Parallel Implementation

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.

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.

Azimuth compression involves three steps which are performed for all 16k rows.
1) Calculating number of azimuth replica points
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 normalization are carried out.



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.

Mumbai, Maharashtra, India
ORBIT 137 RISAT 1, SAC, ISRO

Optimization of the algorithm

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 warp schedulers schedule 32 threads per warp in the device. Hence the number of threads -being a multiple of 32- ensures that no core would remain free during any of the warp. 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. 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. 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 trigonometri 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.

Results and Analysis

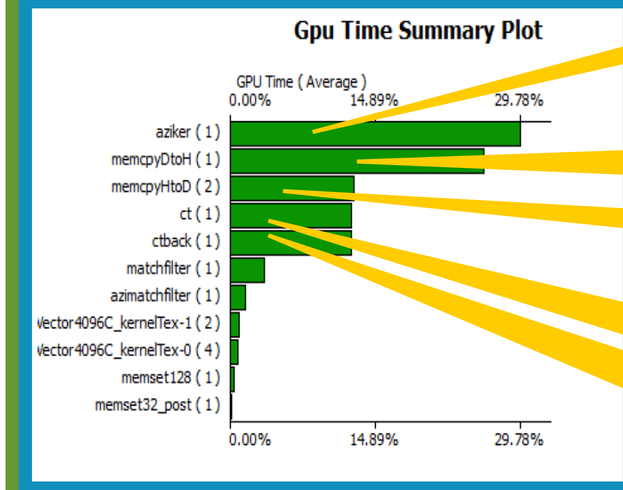
A. Comparison of execution time of CPU and GPU

IMAGE SIZE	4096 x 4096	8192 x 4096	8192 x 8192	16384 x 8192
*CPU TIME (seconds)	10.774	18.967	43.34	88.234
**GPU TIME (seconds)	0.593	0.858	1.544	2.839
Speed up	18x	22x	28x	31x

* OpenMP optimized C code for Quad Core, Intel Xeon E5 (Sandy bridge) based machine with 24GB RAM.
** CUDA optimized C code for GPU, NVIDIA TESLA C2075 and Intel Core i5 based machine with 6 GB RAM.

The table showing execution time of various image resolutions is shown above. 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.

B. Comparison of execution time of various kernels on GPU



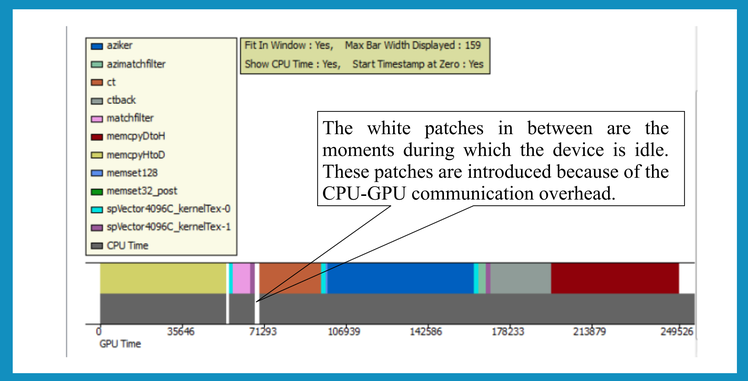
Maximum time is taken by the azimuth replica generation which involves sine and cosine functions.

The memory copy host to device and device to host also contributes to 30 - 35 % of the execution time. The azimuth algorithm demands complete data set to be loaded in GPU memory for a single image processing. Hence the asynchronous functions cannot be used to overlap data transfer and kernel computation.

The matrix transpose operation takes around 30% of the execution time. Along with the Azimuth kernel the transpose operation also limits the use of asynchronous functions -used to overlap data transfer and kernel operation- or processing still radar images.

Nvidia Visual Profiler SnapShot

C. Time line showing sequential execution process of kernels



Nvidia Visual Profiler SnapShot

Future Work:

1. Implementation of continuous radar image processing which may involve use of asynchronous functions for overlapping data transfer and kernel computation which is not possible in present case.
2. Implementing the GPU algorithm for continuous imagery over GPU cluster or hybrid cluster.
3. Implementing the algorithm on Intel Xeon Phi and comparison of the power utilization with GPU.

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://developer.nvidia.com/cuda/cuda-downloads

Project Consortium: NIRMA UNIVERSITY; अंतरिक्ष उपयोग केन्द्र Space Applications Centre; इसरो isro
Project Funding: NIRMA UNIVERSITY INSTITUTE OF TECHNOLOGY; NVIDIA