Comparative Analysis of Performance Acceleration Technologies on SAR Imaging Algorithms

V Jithesh, Dhiraj Kumar Singh, M Justin Sagayaraj
Electronics and Radar Development Establishment, CV Raman Nagar, Bengaluru, India, Email: jithesh.v@lrde.drdo.in

***Abstract*—Data correlation is the most compute-intensive stage in Synthetic Aperture Radar (SAR) data processing. Generally specialized hardware is used for this purpose. But due to the never ending advancements in the areas of computing hardware and software technologies, even COTS hardware may be used to derive real-time performance from the correlator. This paper discusses the implementations of Range Doppler (the very first imaging algorithm and one of the most popular algorithms) and Back-projection (the most data & computationally intensive algorithm) algorithms used in SAR Remote Sensing using CPU & GPU parallel processing technologies like OpenMP, PPL, CUDA, OpenCL and C++ AMP and compare their performances. The time taken for the computation of each step in these algorithms are recorded and compared to assess the acceleration achieved. It has been observed that both of the algorithms are benefitted by these parallelization technologies and the acceleration is more for data parallel and compute intensive steps. The total acceleration achieved varies in the range of 23x to 2166x on an NVIDIA Tesla K20c GPU and assures that real-time performances can be achieved.**

I. Introduction

The invention of Synthetic Aperture Radar (SAR) is generally credited to Carl Wiley, of the Goodyear Aircraft Corporation, in 1951 and SARs are being used in Remote Sensing since 1978 (SEASAT-A satellite). In the SAR concept, the radar is placed on a moving platform, generally a satellite or airplane, and uses signal processing to produce an imagery of what the radar is seeing. Decades long researches in SAR have resulted in developing a family of remarkable signal processing techniques that are capable of producing imagery whose cross-range resolution is independent of range and much finer than is possible to achieve with any practically-deployable real-beam antenna [1], [2], [3], [4] & [5]. One of the major challenges associated with SAR is the processing of the voluminous data (size ranging from Giga Bytes to Tera Bytes) acquired during its mission. The data collected by a SAR has to be processed either in real-time or sent to a ground station for offline processing.

There are three stages in SAR data processing: Pre-processing, Correlation and Post-processing. The correlator uses popular SAR imaging algorithms such as Range Doppler Algorithm (RDA), Range Migration Algorithm (RMA), Chirp Scaling Algorithm (CSA), SPECAN, Polar Format Algorithm (PFA), Back-projection (BP) Algorithm, Kirchhoff Migration Algorithm, etc. All these algorithms are compute-intensive and become data-intensive as well since they have to work on massive amount of data. Specialized hardware such as Field Programmable Gate Arrays (FPGA), DSP boards, POWERPC, etc., are used as the platform for the correlator for achieving the necessary real-time performance. The major problem associated with these specialized hardware is that the binding between the software and hardware is too strong. The software may have to be re-written if we change the hardware.

An alternative is to use COTS hardware, but they fail to provide the real-time performance under normal conditions. Fortunately, the never ending advancements in the areas of computing hardware and software technologies offer a multitude of options here. These options include simple Multi-core programming, Distributed computing on Multiple CPUs, Graphics Processor Unit (GPU) based computing, Coprocessor based computing, etc.

This paper discusses the implementation of Open specifications for Multi-Processing (OpenMP), Microsoft Parallel Patterns Library (PPL), Compute Unified Device Architecture (CUDA), Open Computing Language (OpenCL) and C++ Accelerated Massive Parallelism (C++AMP) in the context of RDA and Back-projection algorithms. It also compares the acceleration offered by these technologies in the case of these algorithms. The reason for choosing RDA is that it is the very first and is one of the most popular SAR imaging algorithms and has computational complexity very much similar to other commonly used algorithms like RMA & CSA. Backp-projection is the most Compute-intensive and data parallel SAR imaging algorithm and hence is naturally the best choice for parallelization.

The remaining portions of this paper are divided as follows: Sections II & III will give an overview of the RDA and BP algorithms respectively. Section IV will discuss about the performance acceleration technologies. The implementation approach is elaborated in section V and the results are discussed in detail in section VI. Concluding remarks are in section VII.

II. The Range Doppler Algorithm

The Range Doppler Algorithm was developed in 1976 - 78 for processing SEASAT SAR data. The first digitally processed space-borne SAR image was made with this algorithm in 1978. The basic processing steps are shown in Fig. 1.

As suggested in [18], to optimize the focus when creating the RDA image using the low altitude CASIE data we used a hyperbolic azimuth chirp rather than the traditional RDA parabolic azimuth chirp. A hyperbolic azimuth chirp $Az(m)$

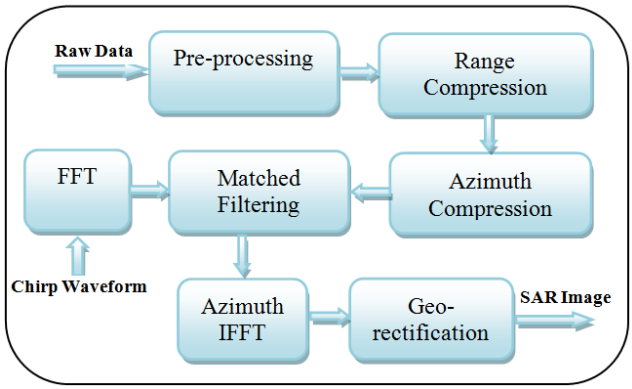

Fig. 1. RDA Block Diagram.

used for RDA azimuth compression at range r_a can be expressed as

$$Az(m) = e^{-j(2\pi f_c \tau(m) + K_r \tau^2(m))} \quad (1)$$

where

$$\tau(m) = 2\sqrt{r_a^2 + (\frac{mv}{PRF})^2} \quad (2)$$

K_r is the LFM frequency ramp rate, $m = [\frac{-N_p}{2}, \frac{N_p}{2}]$ is the pulse index, N_p is the number of pulses, f_c is the carrier frequency, c is the speed of light, PRF is the pulse repetition frequency, and v is the along-track velocity.

III. The Backprojection Algorithm

The back-projection algorithm is an exact algorithm for creating images from SAR data. The algorithm implements the azimuth matched Filter for each pixel, and accounts for both range cell migration and the motion of the aircraft. Back-projection normally operates on interpolated range compressed data. For each pixel, the range compressed data from a given LFM chirp is interpolated to the slant-range distance from the pixel to the platform. Then the matched filter phase is computed based on the distance, which corrects for motion during the pulse. The interpolated range-compressed signal and matched phase are multiplied and summed over all the chirps in the azimuth window. Sub-aperture processing can be accomplished by creating appropriate azimuth windows, and incoherently summing the sub-aperture images. The back-projection algorithm can optimally account for all flight conditions including non-ideal motion and gain fluctuation due to attitude variations [17].

For a pixel located (x, y, z), the SAR back-projection algorithm can be approximately expressed as [18]

$$I(x,y) = \sum_n S(n, d_n) P(d_n) e^{-j4\pi(\frac{d_n}{\lambda} - \frac{d_n^2 K_\tau}{c^2})} \quad (3)$$

$$d_n = \sqrt{(x_n - x)^2 + (y_n - y)^2 + (z_n - z)^2} \quad (4)$$

where $I(x, y)$ is the complex pixel value (the complex SAR image), λ is the wavelength of the transmit frequency at the center of the SAR band, d_n is the distance between (x, y, z) and the antenna phase center of the SAR antenna (x_n, y_n, z_n) for pulse number n.

In above equation, $P(d_n)$ is the range-compressed SAR data interpolated to slant range d_n. $S(n, d_n)$ is a range-dependent window to reduce azimuth side lobes.

The SAR image is created by varying the x and y over a grid. The height z of the surface at (x, y) is typically computed from a digital elevation map (DEM). The challenge of back-projection is that it is computationally intensive; however it can be parallelized to achieve near real-time performance in many cases.

IV. Acceleration Technologies

Latest CPUs have up to 24 cores and 4 such CPUs (amounting to 96 cores) can be accommodated in one Server / Workstation. 32 core processors are in the making and this trend will continue. No distributed computing technology like MPI [10] is required to run a parallel algorithm on such a hardware platform. A parallel algorithm implemented in OpenMP [10] & [13] or PPL [12] can address all these 96 cores.

Graphics Processor Unit based computing is a fast growing technology for accelerating the efficiency of computation and data intensive algorithms. High-end GPUs with thousands of processing cores and above 5 TFLOPS double precision (10+ TFLOPS single precision) computing power are available now. A single such GPU can outperform hundreds of CPU cores and has reduced hardware size, weight and cost. There are at least 4 technologies now to program a GPU. They are NVIDIA CUDA [6], [9] & [10], OpenCL from Khronos Group [6], [7], [8] & [10], Microsoft C++ AMP [11] and NVIDIA Open Accelerators (OpenACC) [6].

GPU programming technologies extends the C programming language and allows GPU code to be written in regular C or C++. The code can either be targeted for the host processor (the CPU) or be targeted for the device processor (the GPU). The CPU spawns multi-threaded tasks onto the GPU device. The GPU has its own internal scheduler that will then allocate these tasks to the GPU hardware. The data has to be first transferred from the host memory to the device memory before running the tasks on the device and once the computation on the device are over, the results are to be transferred from the device memory back to the host memory.

Intel Xeon Coprocessor programming is another technology gaining momentum [14]. Xeon Coprocessors with up to 72 cores are available as of now.

In this work, we have considered the implementation of RDA using OpenMP & CUDA and Backprojection using PPL, CUDA, OpenCL & C++ AMP, in addition to their sequential implementations (Single Core) and comparisons are being made about the computational performances.

V. Implementation Strategy

The raw SAR data used in this study is the public domain sample SAR data set collected by the BYU/Artemis μASAR

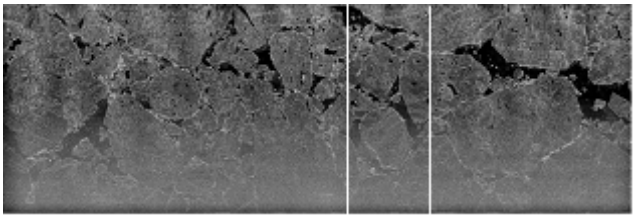

Fig. 2. Sample μASAR Image [18]

system during the Characterization of Arctic Sea Ice Experiment 2009 (CASIE-09) [15], [16] & [18]. This μASAR raw data collected from the Arctic Ocean from Svalbard Island is divided into 1 min segments typically 3.5 km long in the along-track dimension by 1.2 km wide in the cross-track dimension. The sample data set used here consists of a 13 sec portion of one of these segments (segment 9, collected on 25 July 2009). The primary observation swath is defined to be from just off nadir to an incidence angle of about 72^o. The dataset has 3885 columns and each of these columns represent the dechirped SAR data for each pulse. The geometry data file contains the SAR pulse number, latitude in degrees, longitude in degrees, and altitude in m.

The Satellite μASAR image shown in Fig. 2 shows an area approximately 3.5 km long by about 1.2 km wide. The data set corresponding to the region with boundary marked in white is used in this study.

RDA implementation was carried out in C++ 11, OpenMP and CUDA-C++, where as the Back-projection was carried out in C++ 11, PPL, OpenCL, C++ AMP and CUDA-C++. During the implementation it was found that CUDA implementation of the preprocessing step in RDA takes more time than a CPU implementation (approximately 2.7 times). Hence to get better acceleration, it was decided not to use CUDA for the preprocessing (except for FFT). The FFT steps used in the CUDA implementations of both the algorithms use cuFFT library. It should be note that no optimization strategies were included in the implementations. All implementations were in 64 bit double precision.

A Windows 7 based 6-core (2.4 GHz, single CPU) workstation with 48 GB DDR3 primary memory and an NVIDIA Tesla K20c GPU (2496 cores, 5 GB DDR5 Memory) was used in this study. The development environment includes Microsoft Visual Studio 2013 (for Serial C++, OpenMP, PPL & C++ AMP) and CUDA Toolkit version 7 (for CUDA & OpenCL).

In RDA, the raw data is first preprocessed. The preprocessing included even spacing of the data, windowing (Hanning) and zero padding. A column-wise FFT is then applied for Range Compression, which is followed by a Row-wise FFT to perform Azimuth Compression. The output of this step gives the SAR data in Range-Doppler domain. The Chirp waveform is converted to frequency domain and convoluted with the Range-Doppler SAR data to perform a Matched Filtering. An inverse FFT (in azimuth direction) is then applied on this filtered data, which is followed by Geo-rectification

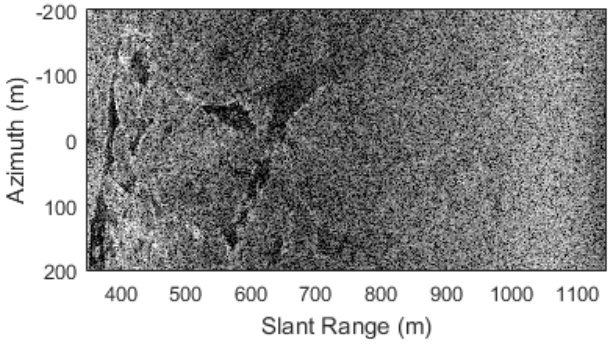

Fig. 3. Single-look RDA SAR Image

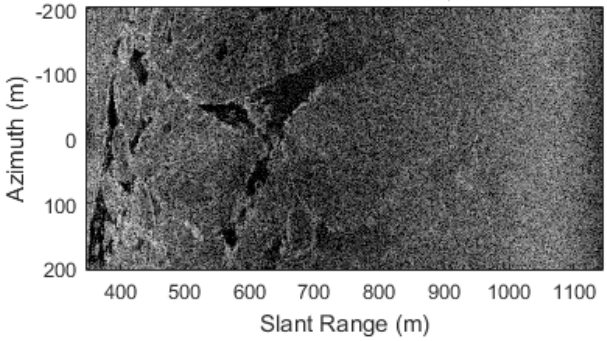

Fig. 4. Multi-look RDA SAR Image

to generate the corrected single-look SAR image. In order to reduce the azimuth resolution and speckle noise, an incoherent averaging (multi-looking) was applied in the azimuth direction on the pixels of the SAR image.

The Back-projection doesn't need data to be even spaced. The Preprocessing step here includes Range FFT & windowing only. Back-projection algorithm is then applied to generate a single-look SAR image. Multi-look processing as described in the case of RDA is applied then.

VI. Results & Discussion

The outputs of one of these computer programs are shown in figures 3, 4, 5, 6, 7 & 8 (generated using MATLAB 2017a). All these images are to be rotated 90^o clockwise to get the original view in Fig. 2. In case of RDA, Single & Multi look SAR images generated before applying geo-rectification are shown in figures 3 & 4 respectively. The geo-rectified versions of these images are in figures 5 & 6. Single & multi-look Back-projected images are in figures 7 & 8 respectively. The RDA images have a resolution of 2048 $\times$ 512 and the back-projected images are of resolution 2810 $\times$ 1208. No image processing techniques were applied on these images to enhance their quality.

The computation time (in milliseconds) and acceleration achieved for these algorithms under the acceleration technologies over their Single Core C++ implementation are summarized in Tables I & II. In CUDA Back-projection, the memory transfer time for host to device and back are not

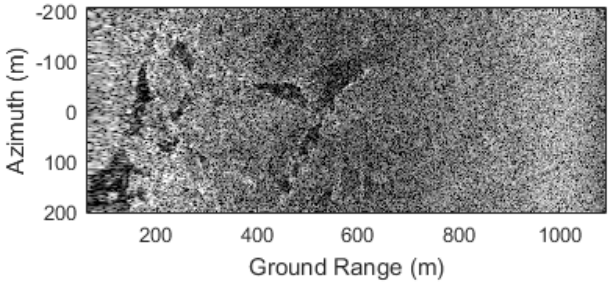

Fig. 5. Single-look RDA SAR Image after Geo-rectification

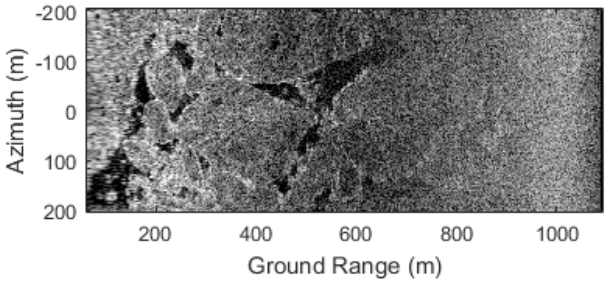

Fig. 6. Multi-look RDA SAR Image after Geo-rectification

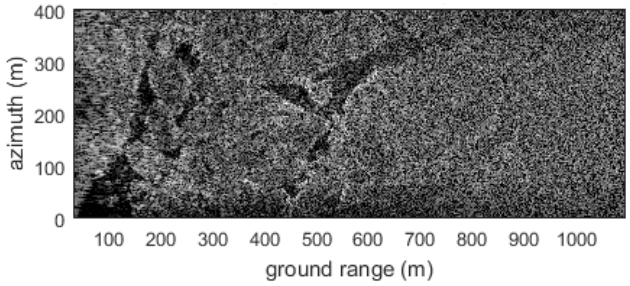

Fig. 7. Single-look Back-projected SAR Image

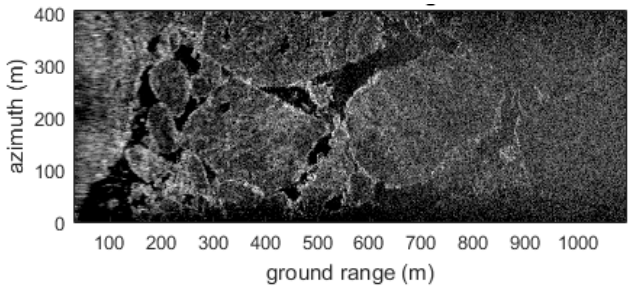

Fig. 8. Multi-look Back-projected SAR Image

shown explicitly. However, they are included in steps 1 & 3 respectively (in Table I). The raw data (input) reading time and processed data (output) saving time are not considered in the comparison. It may be noted that the accelerations mentioned in these tables are not the Speed-ups as we are considering only the parallelizable steps in BP & RDA for the comparison.

From Tables I & II, it is clear that in a multi-core environment with 6 cores, technologies like PPL and OpenMP offer

TABLE I
COMPUTATION TIME AND ACCELERATION FOR BACK-PROJECTION ALGORITHM

Computation Step	Computation Time (ms)				
	1 Core	PPL	CUDA	OpenCL	AMP
Pre-processing	2356	468	936*	2106	453
Singlelook BP	2027214	348925	47	1388	468
Multilook	203	812	16	47	16
Total Time (ms)	2029773	350205	999	3541	937
Acc. over 1 Core (Times)		5.8	2032	573	2166

TABLE II
COMPUTATION TIME AND ACCELERATION FOR RANGE DOPPLER ALGORITHM

Computation Step	Computation Time (ms)		
	1 Core	OpenMP	CUDA
Pre-processing	234	171	230*
Host to Device Data Transfer	–	–	94
Range Compression	4275	905	156
Chirp FFT	811	281	78
Azimuth Compression	858	328	78
Matched Filtering	202	93	62
IFFT	905	93	62
Geo-rectification	16910	2090	47
Multi-look Processng	1545	296	15
Device to Host data Transfer	–	–	297
Total Time (ms)	25740	4492	1119
Acc. over Single Core (Times)		5.7	23

up to 5.8 times acceleration over Single Core computation. Among the GPU acceleration technologies, CUDA and C++ AMP performs better than OpenCL and their performances are very much similar. One of the reasons for the poor performance of the OpenCL based implementation is that the OpenCL libraries bundled with CUDA Toolkit was used for this implementation and the CUDA toolkit is performance optimized for CUDA, not for OpenCL.

Another interesting observation(*) is that in the case of the Pre-processing (Windowing & FFT) step in Back-projection, CUDA's performance is inferior to PPL & C++ AMP. In the pre-processing step(*) in RDA also CUDA underperforms. This is due to the fact that in the CUDA based implementation, NVIDIA cuFFT library was used; whereas in the other implementations, a custom made FFT was used and this custom FFT outperformed cuFFT. Thus the performance of the CUDA based implementation can further be improved if we perform the Windowing & FFT step on CPU with OpenMP (or PPL).

As seen from these tables, the Back-projection algorithm had benefitted the most by the acceleration technologies. This is expected since the Back-projection algorithm is very much data parallel and more compute-intensive. Even in RDA the most compute-intensive step namely Geo-rectification benefitted most from the GPU acceleration. It may also be noted that the performances of the GPU implementations of these

two algorithms are very much similar (for example CUDA-BP takes 999 ms and CUDA-RDA takes 1119 ms), where as the Single core CPU implementations of these algorithms vary drastically (in the order of 800 times) in terms of performance.

VII. Conclusion

A study has been carried out to compare the performances of two Radar imaging algorithms under different acceleration technologies to their suitability to perform SAR data correlation in a real-time environment. The results are found to be very promising even with mid-range GPUs. This can be enhanced by using strategies like: optimizing the code by selecting optimum registry and block sizes and other performance optimization approaches like usage of shared memory concept; using higher-end GPUs like NVIDIA K40, K80, or P100; using single precision computing (since GPUs offer 2 or 3 times single precision computing power than their double precision computing power); and by using multiple GPUs on a single Workstation / Server (up to 4 GPUs are supported as of now). The performance can further be enhanced if one uses a distributed computing environment (using technologies like MPI) with multiple compute nodes. Performance of the I/O operations may be enhanced by using Parallel I/O libraries under a Parallel File System.

Thus it is evident that GPUs are an alternative to embedded technologies (like FPGA, DSP Board, POWERPC, etc.,) to derive real-time performance out of these imaging algorithms. But it may be noted that the power requirement for a COTS GPU is in the order of 100s of Watts and is much higher than that required by the embedded technologies. Hence COTS GPUs cannot be considered in situations wherein there is a stringent power constraint, which GPUs cannot meet. However, custom GPUs for Embedded platforms (needing very nominal power) are available, even though they won't offer the massive computing power that the COTS GPUs offer.

Acknowledgment

The authors would like to thank Mr. Surya Prakash K.S., Associate Director, LRDE, for his valuable help, encouragement and motivation during the implementation of the work described in this paper.

References

[1] Margaret Cheney, *A Mathematical Tutorial on Synthetic Aperture Radar,* SIAM REVIEW, Vol. 43, No. 2, pp. 301-312, 2001.

[2] Walter G. Carrara, Ron S. Goodman, and Ronald M. Majewski, *Spotlight Synthetic Aperture Radar Signal Processing Algorithms,* Artech House, 1995.

[3] I. G. Cumming, and F. H. Wong, *Digital Processing of Synthetic Aperture Radar Data,* Artech House, Norwood, 2005.

[4] John C. Curlander, and Robert N. McDonough, *Synthetic Aperture Radar Systems and Signal Processing,* John Wley & Sons Inc, 1993.

[5] William L. Melvin, and James A. Scheer, *Principles of Modern Radar: Vol. II: Advanced Techniques,* SciTech Publishing, Edison, NJ, 2013.

[6] David B. Kirk, and Wen-mei W. Hwu, *Programming Massively Parallel Computers,* Elsevier, 2010.

[7] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, and Dana Schaa, *Heterogeneous Computing with OpenCL,* Morgan Kaufmann (Elsevier), 2012.

[8] Mathew Scarpino, *OpenCL in Action,* Manning Publications Co, 2012.

[9] Shane Cook, *CUDA Programming: A Developers Guide to Parallel Computing with GPUs,* Morgan Kaufmann (Elsevier), 2013.

[10] Thomas Rauber, and Gudula Runger, *Parallel Programming for Multi-core and Cluster Systems,* Springer-Verlag, 2010.

[11] Kate Gregory, and Ade Miller, *C++ AMP: Accelerated Massive Parallelism with Microsoft Visual C++,* Pearson Education, 2012.

[12] Colin Campbell, and Ade Miller, *Parallel Programming with Microsoft Visual C++,* Microsoft Press, 2011.

[13] Barbara Chapman, Gabriele Jost, and Ruud van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming,* MIT Press, 2008.

[14] Endong Wang, Qing Zhang, Bo Shen Guangyong Zhang, Xiaowei Lu Qing Wu, and Yajuan Wang, *High-Performance Computing on the Intel Xeon Phi: How to Fully Exploit MIC Architectures,* Springer International Publishing Switzerland, 2014.

[15] E. Zaugg, D. Hudson, and D. Long, *The BYU SAR: A small, student-built SAR for UAV operation,* IEEE International Conference on Geoscience and Remote Sensing Symposium, pp. 411-414, 2006.

[16] D.G. Long, E. Zaugg, M. Edwards, and J. Maslanik, *The MicroASAR Experiment on CASIE-09,* Proceedings of the International Geoscience and Remote Sensing Symposium, pp. 3466-3469, Honolulu, Hawaii, 2010.

[17] Craig Stringham, and David G. Long, *Improved Processing of the CASIE SAR data,* IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2011.

[18] David G. Long, and Craig Stringham, *The Sample BYU CASIE-09 MicroSAR Dataset,* http://www.mers.byu.edu.

V Jithesh did his M.Sc and M.Phil Degrees in Mathematics from the University of Kerala and M.Tech Degree in Computer Science from Cochin University of Science and Technology (2001). He joined Electronics and Radar Development Establishment, Bangalore in the year 2003. His areas of interest include High Performance Computing, Machine Learning, Radar Target Classification, Signal & Data Processing Algorithms, and Computational Electromagnetics.

Dr. Dhiraj Kumar Singh did his B.E in electronics & communication enginnering from Magadh University in 2001, MTech in Microwave Engg. in 2003 and Ph.D from IIT Kharagpur in 2014. He joined Electronics and Radar Development establishment, Bangalore in 2003. His areas of interest include Applied Electromagnetics, UWB Antenna Design and Synthetics Aperture Radar systems.

Justin Sagayaraj M is a scientist in Electronics and Radar Development Establishment, Bangalore. He has received M.Tech in Computer science & Engineering from Indian Institute of Technology, Madras, MSc (computer Science) From Madurai Kamaraj University, and BSc (Special Mathematics) from Madurai Kamaraj University. He has worked for many radar tracking systems and communication systems.