GPU Based Performance Acceleration of Radar Imaging Algorithms

V. Jithesh[1], **Dr. K. Poulose Jacob**[2]

[1]Electronics and Radar Development Establishment (LRDE), Bangalore, India
[2]Department of Computer Science, Cochin University of Science & Technology (CUSAT), Cochin, India
jithesh_lrde@rediffmail.com

Abstract

We consider the performance acceleration of the conventional Time Domain Backprojection and Kirchhoff Migration algorithms for imaging concealed targets. The Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) are used here for accelerating these algorithms on Graphics Processing Units (GPUs). Data generated by means of analytical methods, simulation and experiment are used for validation and performance comparison.

Keywords: *Radar Imaging, Backprojection, Kirchhoff Migration, SAR, GPR, TWIR, CUDA, OpenCL, GPU, GPRMax*

I. INTRODUCTION

Radar imaging has potential applications in the field of Synthetic Aperture Radar (SAR), Medical Imaging, Ground Penetrating Radar (GPR), Through–the–Wall Imaging Radar (TWIR), Foliage Penetration Radar, Forensic Investigation, Baggage Scanning, Stand-off detection of threat objects concealed under clothing etc. The common radar imaging algorithms include Backprojection (time & frequency), Kirchhoff Migration, Range Doppler Algorithm (RDA), Range Migration Algorithm (RMA), Chirp Scaling Algorithm (CSA), Stripmap SAR, Impulse SAR, etc.

Most of these imaging algorithms are data-massive and compute-intensive and hence the overall throughput of the application is heavily dependent on the computational efficiency of the underlying hardware. Thus for practical implementations a parallel architecture is always preferable to achieve a real-time performance. Conventional dual or quad core CPU does not meet the massive data processing requirements for radar imaging, especially for very large data sets and high resolution.

During the past couple of years Graphics Processing Units (GPUs) have evolved into highly parallel, multithreaded, many-core co-processors with tremendous computational power. They have very high memory bandwidth and are capable of executing general-purpose computations with extreme efficiency. GPU computing has created a quantum shift in computing architecture by introducing a hybrid model in which GPUs work in conjunction with CPUs. Technologies like NVIDIA Compute Unified Device Architecture (CUDA), Khronos Open Computing Language (OpenCL) and Microsoft C++ AMP are widely used in GPU computing.

This paper describes the GPU implementation aspects of two of the most commonly used radar imaging algorithms namely Time Domain Backprojection and Kirchhoff Migration in detecting concealed targets. Both CUDA and OpenCL implementations are discussed here and the performances of these implementations are compared against CPU implementations. Synthetic, simulated and experimental data sets were used for this study.

The next two sections briefly outline the Time domain Backprojection and Kirchhoff Migration algorithms. The CUDA and OpenCL implementation aspects are explained in Sections IV & V respectively. Both the implementations use NVIDIA CUDA SDK 4.2 on an NVIDIA GeForce GT 610 GPU. The results and performance comparison details are described under section VI.

II. TIME DOMAIN BACKPROJECTION

Backprojection is a matched filter implementation of time-domain correlation. The idea is to correlate data collected at each aperture position as a function of round-trip delay time. Backprojection coherently sums the sampled radar returns for each pixel of the image map. Coherent summation is a process whereby the signal obtained at each aperture position is time-shifted to match or align, it to a particular pixel element in the image map.

The time domain backprojection is performed through the following equation:

$$I(y, z) = \sum_{k=1}^{K} b(y_k, t)$$

where (y, z) is the point/pixel to be focused, K is the number of measurement locations, $b(y_k, t)$ is the B-scan data [2] collected along a single line at discrete points (y_k, z'), k = 1, 2, ..., K, at time instant t, and t is the two-way travel time from Transmitting antenna at the k^{th} location to the point (y, z) and back to the k^{th} Receiving antenna. If t does not match with any time sample, numerical interpolation may be employed to compute an approximate value of b at t. More details on the theory can e found in [9]-[12].

III. KIRCHHOFF MIGRATION

The basic idea in the Kirchhoff Migration is to back-propagate the measured wave front to a plane at $t = 0$, using an integral solution method to the scalar wave equation. So this migration method involves back-propagation or inverse extrapolation to remove the effects of wave field propagation ([2]-[8]).

The imaging equation for Kirchhoff Migration is as follows:

$$I(y, z) = \frac{1}{2\pi v_m} \sum_{k=1}^{K} \frac{\partial}{\partial t} b(y_k, t = \frac{|r - r'|}{v_m}) \frac{\cos\theta}{|r - r'|}$$

where, $r = (y, z)$ is the point to be migrated, $r' = (y_k, z')$ is the source (Transmitting Antenna) location, $|r - r'|$ is the distance between r and r', K is the number of measurement locations, v_m is the velocity of the wave inside the medium of propagation, $b(y_k, t)$ is the B-scan data collected along a single line at discrete points y_k, $k = 1, 2, ..., K$, at time instant t, and $\cos\theta = |z - z'| / |r - r'|$

IV. CUDA IMPLEMENTATION

The CUDA programming ([19]-[25]) supports the CUDA threads executed on a physically separate device (GPU) that operates as a coprocessor to the host (CPU) running the program. The program consists of a series of sequential and parallel execution phases. Sequential phases have little or no parallelism, which are executed on the CPU as host code. Parallel phases have rich data parallelism, which are implemented as a set of kernel function (piece of code to be executed on the GPU) on GPU.

The fundamental means of parallel execution in CUDA is data-parallel threads. Launching a CUDA kernel function creates a grid of parallel threads, all of which execute the kernel function. Each CUDA grid typically is comprised of thousands to millions of lightweight GPU threads per kernel invocation. At the top level, each grid consists of one or more thread blocks. All blocks in a grid have the same number of threads.

CUDA C implementation includes the following steps: 1. Allocation of host memory and loading the B-scan data to host memory. 2. Allocation of device memory through the function *cudaMalloc*. 3. Copying data from host memory to device memory through *cudaMemcpy*. 4. Invocation of kernel function(s) to perform the computation. 5. Copying data from device memory to host memory through *cudaMemcpy*. 6. Deallocation of host and device memory spaces.

V. OPENCL IMPLEMENTATION

OpenCL ([24]-[28]) makes use of a data parallelism model that has direct correspondence with the CUDA data parallelism model. When a kernel function is launched, its code is run by work items, which correspond to CUDA threads. An index space defines the work items and how data are mapped to the work items; that is, OpenCL work items are identified by global dimension index ranges (NDRanges). Work items form work groups, which correspond to CUDA thread blocks. The NDRange (CUDA grid) contains all work items. The following are the steps involved in the implementation.

1. Discover and initialize the platforms (*clGetPlatformIDs*)
2. Discover and initialize the devices (*clGetDeviceIDs*)
3. Create a context (*clCreateContext*)
4. Create a command queue (*clCreateCommandQueue*)
5. Create device buffers (*clCreateBuffer*)
6. Write host data to device buffers (*clEnqueueWriteBuffer*)
7. Create and compile the program (*clCreateProgramWithSource*)
8. Create the kernels (*clCreateKernel*)
9. Set the kernel arguments (*clSetKernelArg*)
10. Configure the work-item structure
11. Enqueue the kernel for execution (*clEnqueueNDRangeKernel*)
12. Read the output buffer back to the host (*clEnqueueReadBuffer*)
13. Release OpenCL resources (*clReleaseXXX*)

VI. RESULTS

An NVIDIA GeForce GT 610 GPU (810 MHz, 2 GB Graphics memory, 48 cores, 8 blocks, CUDA 2.1 compute support) on an Intel Core 2 Duo (3.6 GHz, 4 GB RAM, Windows 7 OS) computer has been used in this study. CUDA-C and NVIDIA OpenCL versions were used in the CUDA and OpenCL implementations of the algorithms. The input data and final results of the analysis are stored in comma separated variables (csv) format and are plotted using MATLAB.

Three types of data sets are used in this study: synthetic data, simulated data and experimental data. It should be noted that travel time corrections were not performed while computing the Time of Arrival (TOA [7]) so that the actual positions of the targets and their positions in the focused images won't match. Moreover, no data pre-processing techniques were applied on these data sets.

The synthetic B-scan data was generated by analytically computing the response of two point targets to a Gaussian input pulse (0.5 ns pulse width) in a 10 m × 10 m domain [10]. The targets were placed at (4, 4) and (8, 3) and their reflections were recorded at 96 locations along the cross range for a duration of 100 ns at each location (1024 samples). The problem geometry and the focused images using Kirchhoff & Backprojection algorithms are shown in Figure 1.

For generating simulated B-scan data, the public domain software GPRMax 2.0 (based on the Finite-

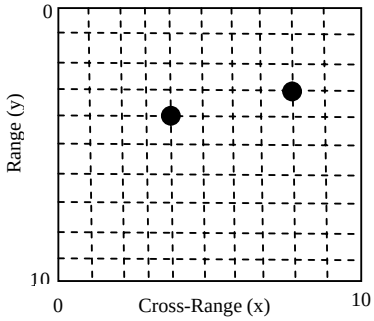
Difference Time-Domain method for Electromagnetics) has been used [29]. For GPR data simulation, a problem domain of size 2.3 m × 0.2 m was considered. The problem geometry is shown in Figure 2. The targets are circular and 8 of them are Perfect Electric Conductors (the first four and the last four) and the rest are vacuum (shown in the middle) and are buried in wet sand. Return signals were recorded at 115 transmit/receive locations and 2036 range bins were considered in this case. A Kirchhoff migrated output is shown in Figure 2.

Simulated B-scan data was generated using GPRMax for TWIR case as well, and a 0.3 m × 0.6 m domain was considered. A concrete wall separates the transmit/receive antenna and the circular PEC target. Vacuum is considered everywhere else in the problem domain. 1357 data samples were recorded at each of the 41 transmit/receive locations. The problem geometry and corresponding focused image are shown in Figure 3.

An experimental set-up was used for collecting data for Through Wall Radar Imaging. This set-up consists of an EATON Advanced Electronics' Impulse Generator (60 kHz - 1GHz) and a Tektronix DPO 71604 Digital Phosphor Oscilloscope with 16 GHz/50 GS/s.

For this measurement a bi-static pair of horn antennas was used with antenna separation of 14 cm (centre to centre). To generate a 2-D scanning array of antennas the antenna platform was manually moved along the cross range (X) and height (Z) directions. A cross range resolution of 30 cm and a height resolution of 15 cm were considered in this experiment. Initial height from ground is 110 cm, and measurements were taken at 125 cm and 140 cm as well. A scan array size of 8 × 3 was considered for the C-scan measurement (8 points along cross range and 3 points along height directions). Distance of the Transmitter/ Receiver pair from wall is 27.5 cm.

A horn antenna, a conical antenna and a metallic cylinder were used as targets. A wooden wall of thickness 8 cm was present between objects and the antenna. Another wooden sheet (with metallic frame), which is part of a cubicle, was present in between the targets and the first wall. In this configuration, a part of the metallic frame also falls in the imaging area (which makes the number of targets four). The data set has 1250 samples and a B-scan subset of this data set at a height of 125 cm was chosen for imaging. The focused image is shown in Figure 4.



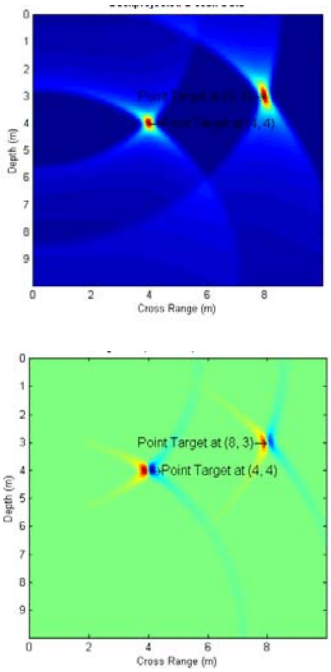


Figure 1. Synthetic Data - Problem Domain (top), Backprojected Image (middle) & Migrated Image (bottom)

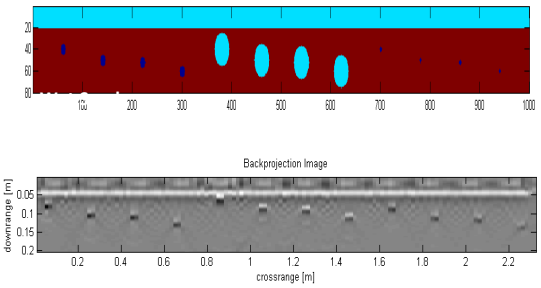


Figure 2. Simulated Data – GPR (Not Scaled) Problem Domain (top) & Migrated Image (bottom)

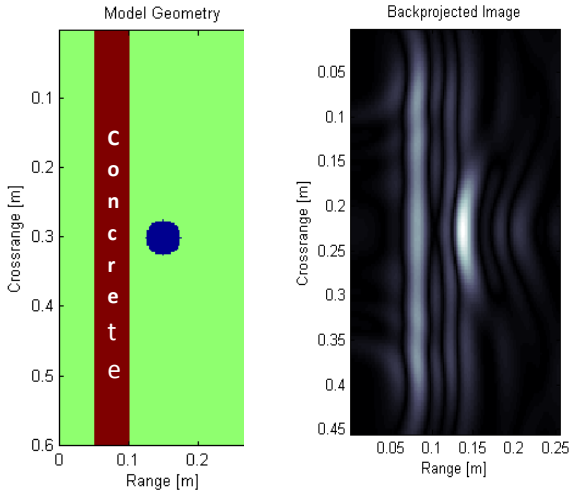


Figure 3. Simulated Data - TWIR Problem Domain (left) & Backprojected Image (right)

Figure 4. Experimental Data - Measurement Setup (top), Targets (middle) & Backprojected Image (bottom)

Figure 5 shows the computation time of the two algorithms for C++ (for CPU), CUDA-C & OpenCL implementations and the performance acceleration achieved by the CUDA and OpenCL implementations on the GPU over a pure CPU execution (single core) using C++, in case of the synthetic data. The executions were performed 25 times and the average computation time in millisecond is plotted here. The data read/write time is not considered. As expected CUDA offered more acceleration than OpenCL in our study. This is mainly due to the fact that the NVIDIA version of OpenCL was used here and it will definitely underperform CUDA, which is optimized to run on NVIDIA GPUs. Probably an AMD/ATI OpenCL version can perform better on this device and may outperform on an ATI GPU.

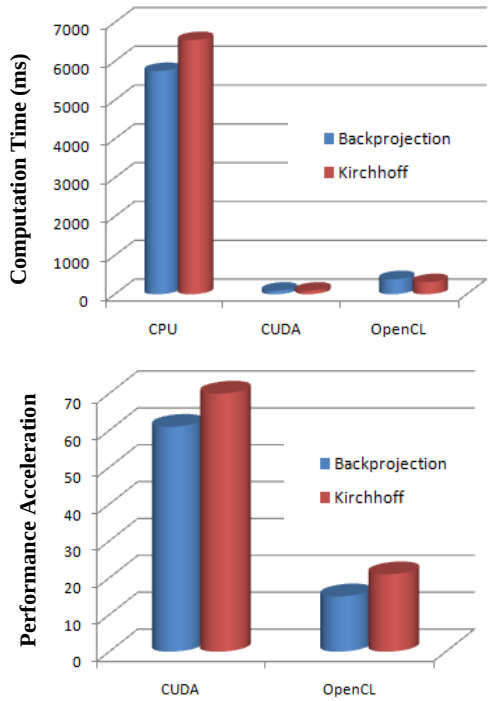


Figure 5. CPU Vs GPU for Synthetic Data (Image Size: 512 × 512 Pixels) - Computation Time (top) & Performance Acceleration (bottom)

VII. CONCLUSIONS

Considerable performance acceleration (in the order of 60 to 70 times) could be achieved by running the Backprojection and Kirchhoff Migration algorithms on a low-end GPU. CUDA and OpenCL technologies were used here. The fundamental difference between CUDA and OpenCL is that CUDA code can run only on NVIDIA GPUs, where as OpenCL code is heterogeneous.

Image resolution beyond 1024 × 1024 pixels resolution could not be considered during this study. This is due to an inherent problem in Windows XP/Vista/7 Operating System (OS). These higher resolution GPU computation takes more than 2 seconds to complete and Windows watchdog application (part of the OS) kills applications sharing the same resources and taking more than 2 s to complete. Here the display driver was also sharing the same GPU since Windows OS disables onboard graphics once the Graphics card is installed. A BIOS configuration may be possible to enable both onboard graphics and GPU at the same time, but it was not supported in our computer. Modifying the Windows registry to set the watchdog's time out value higher is another option, but is not a recommended approach and is not a permanent solution. Using an non-Windows OS like Linux may help in resolving the issue.

The performance can further be improved by optimizing the code by selecting optimum registry and block sizes and other performance optimization strategies. Executing the code on advanced GPUs like NVIDIA Tesla C 2075 (512 cores) or Tesla K10, K20 & K20X which are having more than 2000 cores and higher clock speed & graphics memory will accelerate it further. Intel Xeon Phi Coprocessor is another promising hardware for accelerating the performance of these algorithms. These possibilities will be explored in future.

The acceleration observed is mainly due to the fact that, in these algorithms, the image intensity is calculated on a pixel-by-pixel basis. Hence how far GPU can accelerate other radar imaging algorithms has to be seen. CPU implementations (C++) of some of these algorithms like Frequency Domain Backprojection, RDA, RMA and Extended CSA were done in this regard ([16]-[18]). The GPU implementation is part of the future work in this dimension.

REFERENCES

[1] David J Daniels, "EM Detection of Concealed Targets", *Wiley*, 2009.
[2] David J Daniels, "Ground Penetrating Radar", Second Ed., *IEE*, 2004.
[3] O. Yilmaz, "Seismic Data Analysis Vol I & II", *Society of Exploration Geophysicists*, 2001.
[4] Jon F.Claerbout and IdaGreen, "Basic Earth Imaging" , *Stanford University*, 2009.
[5] Jon F. Claerbout, "Fundamentals of Geophysical Data Processing", *Blackwell Scientific Publications*, 1985.
[6] Gary F. Margrave, "Numerical methods for Exploration Seismology", 2003.
[7] Michal Aftanas, "Through Wall Imaging With UWB Radar System", *PhD Thesis*, Technical University of Kosice, 2009.
[8] Bart SCHEERS, "Ultra-Wideband Ground Penetrating Radar, with Application to the Detection of Anti Personnel Landmines", *PhD Thesis*, Universite Catholique De Louvain Laboratoire D'hyperequences, Brussels, 2001.
[9] J Boutros and Greg Barrie, "Ultra-wideband Synthetic Aperture radar Imaging", *Defence R & D Canada (DRDC)*, 2003.
[10] Greg Barrie, "Ultra-wideband Synthetic Aperture Data and Image Processing", *Defence R & D Canada (DRDC)*, 2003.
[11] GregBarrie, "Through-Wall Synthetic Aperture Radar (TWSAR) 3D Imaging", *Defence R & D Canada (DRDC)*, 2004.
[12] S. Gauthier, E. Hung, and W. Chamma, "Surveillance Through Concrete Walls", *Proceedings of SPIE 5403*, pp. 597-608, Dec. 2004.
[13] L. V. Kempen, "Ground Penetrating Radar for Anti-personnel Landmine Detection", *Ph.D. Dissertation*, Vrije Universiteit Brussel, 2006.
[14] E. M. Johansson and J. E. Mast, "Three-dimensional Ground-Ppenetrating Radar Imaging Using Synthetic Aperture Time-domain Focusing", *Proceedings of SPIE*, vol. 2275, pp. 205-214, Sep. 1994.
[15] C. Lei and S. Ouyang, "Through-wall Surveillance using Ultra-wideband Short Pulse Radar: Numerical Simulation", *Industrial Electronics and Applications*, pp. 1551- 1554, May 2007.
[16] Walter G. Carrara, Ron S. Goodman, and Ronald M. Majewski. "Spotlight Synthetic Aperture Radar Signal Processing Algorithms", *Artech House*, 1995.
[17] Ian G Cumming, Frank H Wong, "Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation", *Artech House*, Boston, 2005.
[18] William L. Melvin, James A. Scheer, "Principles of Modern Radar: Vol. II: Advanced Techniques", *SciTech Publishing*, 2013.
[19] Jason Sanders Edward Kandrot, "CUDA by Example ", *Addison-Wesley*, 2011.
[20] Rob Farber, "CUDA Application Design and Development", *Elsevier*, 2011.
[21] "NVIDIA CUDA C Programming Guide", Version 4.2, 2012.
[22] "CUDA API Reference Manaul", Version 4.2, March 2012.
[23] Shane Cook, "CUDA Programming: A Developer's Guide to Parallel Computing with GPUs", *Morgan Kaufmann (Elsevier)*, 2013.
[24] David B. Kirk and Wen-mei W. Hwu, "Programming Massively Parallel Computers", *Elsevier*, 2010.
[25] Thomas Rauber, Gudula Runger, "Parallel Programming for Multicore and Cluster Systems", *Springer-Verlag*, 2010.
[26] Matthew Scarpino, "OpenCL in Action", *Manning Publications Co.*, 2012.
[27] A. Munshi, et al., "OpenCL Programming Guide", *Pearson*, 2012.
[28] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa, "Heterogeneous Computing with OpenCL", *Morgan Kaufmann (Elsevier)*, 2012.
[29] "GPRMax 2D/3D User's Manual Version 2.0".

BIODATA OF AUTHORS

V. Jithesh obtained 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. He then joined Electronics and Radar Development Establishment, Bangalore in the year 2003. His areas of interests include Computational Electromagnetics, High Power Electromagnetic Copling, High Performance/GPU Computing, Radar Imaging and Radar Target Discrimination.

Dr. K. Poulose Jacob, Professor of Computer Science at Cochin University of Science and Technology (CUSAT) since 1994, was the Director of the School of Computer Science Studies till July 2013. Since July 2013 he is the Pro-Vice Chancellor of the University.

He is the founder Director of the School of Computer Science Studies and has been Director of the CUSAT Computer Centre and had initiated the Centre for Information Resource Management. He has been the Dean of the Faculty of Engineering and is presently Chairman, Board of Studies in Computer Science. He is a member of Academic Council and has been in the University Senate for 10 years. He has held additional administrative responsibilities at CUSAT, like Head of the Department of Computer Applications, Director - Centre for MHO Co-operation, Director - Strategic Planning.

He has presented research papers in several International Conferences in Europe, USA, UK, Australia and other countries. He has delivered invited talks at several national and international events. He has served as a Member of the Standing Committee of the UGC on Computer Education & Development. He is the Zonal Coordinator of the DOEACC Society under the Ministry of Information Technology, Government of India. He serves as a member of the AICTE expert panel for accreditation and approval. He has been a member of several academic bodies of different Universities and Institutes. He is on the editorial board of two international journals in Computer Science.

Dr. K.Poulose Jacob is a Professional member of the ACM and a Life Member of the Computer Society of India. He has more than 75 research publications to his credit. His research interests are in Information Systems Engineering, Intelligent Architectures and Networks.

He has been listed in the 2010 Edition of *Who's Who in the World*® representing the world's most accomplished individuals.