HIGH PERFORMANCE COMPUTING USING GPGPU FOR RADAR APPLICATIONS

Viswam Gampala[1] (visgam@yahoo.co.in), Akshay BM[1], A Vengadarajan[1], PS Avadhani[2]

1. Electronics & Radar Development Establishment, DRDO, C.V Raman Nagar, Bangalore-560093
2. Professor in Dept of Comp. Science and Systems Engg, AndhraUniversity,Vishakhapatnam-530001

Abstract**—**Modern Radar Sensor systems are being deployed to carry out multi tasking for detection and tracking of several objects simultaneously. Active Electronically steered Phased array technology is the key element being utilized for design and development of these modern radar systems. These are used for carrying out efficient and effective weapon system control functions for Air Defence missions against Aircrafts and Missiles. The target data provided by these radars play pivotal role in taking mission critical decisions in time and improve the efficacy of the overall system for better kill. However, as against conventional old generation radar systems, these modern radars are required to process, analyse and classify the kind of threat for better system vulnerability assessment and prioritization.

A radar system receives digitized video data from receiver(s) and carries out a set of highly compute intensive Data / Signal Processing activities.

The GPGPU provides a simple and easily implementable parallel software architecture paradigm using general purpose programming languages like C / C++. The entire data / signal processing task can be realized as a sequence of software activities taking the advantage of very high throughput possible with the GPUs.

This paper brings out a generic design for architecting a GPGPU based Signal / Data processing system for current day Radar systems Also it presents rational for proposing such a design, the data structures that are scalable for multi-channel handling with minimal / no modifications and sample implementation of few algorithms. A sample implementation is carried out using NVidia GPU with CUDA constructs on a commercial ASUS 2U Server with 64 GB RAM and performance measurements carried out.

Keywords - **Signal Processing, Radar, GPU, High Performance Computing, Real-time Processing, Waveforms, MTI, PC, FFT, IFFT, Device / Host Memory.**

I. INTRODUCTION

A typical radar system receives digitized video data from receiver(s) and carries out a set of highly compute intensive Data / Signal Processing activities. The data is received from several tens of channels (at times even hundreds of channels) in real-time and have to complete the processing activity in few milliseconds for all these channels. Processing involves frequency domain analysis, Beam formation, MTI, Coherent processing in Range and / or Doppler etc. Typically, these processes are data centric and are parallel in nature with inherent SIMD leniency. The order of complexity in hardware and software implementation is very high in such systems, posing major problem for implementation, maintenance and sustenance through-out the life cycle of the system. Advances in computing systems have reduced this complexity dramatically in the past few years by means of using dedicated FPGA based architectures, General Purpose Graphic Processor Units (GPGPU) etc. Several hundreds (even thousands) of GFLOPS throughput can be achieved with the current day commercially available hardware at very low costs. This paper brings out a generic design for architecting a GPGPU based Data / Signal processing system for current day Radar systems.

Definitions :

a. ***Sample / Range Cell :*** *Digital sample from ADC of Radar receiver. Contains a In-phase component and Quadrature component (I, Q). Represents the signal vector for a given range cell of the radar*
b. ***Pulse :*** *The radar operation is carried out generally in pulsed form. Each pulse of EM energy is transmitted for specific period of time (may few tens of micro seconds) and echo(s) received from target(s) to identify presence / absence of a target of interest.*
c. ***Burst :*** *A radar generates a set of pulse trains coherently, in order to extract parameters of targets. Each set of pulse train is termed as a burst.*

II. DATA RECEPTION

The data arriving from various receiver channels in digital form in real-time is a stream of baseband samples at a given bandwidth. This bandwidth typically depends on the radar functions, applications and the purpose for which the radar is being used. Further, the data is typically in the form of I, Q samples in order to preserve the phase of the signal which is essential for some of the signal processing algorithms to give proper results through interpretation. The data may be received in one of several possible interfaces using protocols like Serial FPDP (SFPDP), Ethernet or parallel interfaces. In a typical modern signal processing systems, it is typically a high speed Ethernet. The Signal processing system is also a high speed parallel processing computing system with interfaces to receive this data. Present day hardware is available with 10 Gbps Ethernet (one ore more interfaces) interfaces with PCIe as the high speed interconnect between the processing host and this interface.

The data thus arriving is received into the computing system for further processing. Each channel consisting of I, Q in a two's complement signed 16 bit integer is the typical data format on arrival. Each sample thus is of 4 bytes. A radar with 5 MHz bandwidth, and four channels (sum, delta Azimuth, delta Elevation and SLB), the total data to be handled by the interfaces would be in the order of 80 MB per second (4 x 5 MHz x 4 = 80 MB). The PCIe 2.0 8x, a common interface on many modern computer platforms can handle upto 4 GB/s of data without any problem. Therefore, a simple PC / Workstation with PCIe 8x may be more than adequate for handling the data for signal processing.

The data is received from the receiver in terms of pulses, one at a time, each having certain number of samples (named as range gates). The signal processing has to be carried out for a group of pulses, typically called a burst, all synchronized with respect to time and coherent in nature. As the data is arriving in discreet form i.e. data for each pulse arrives at different times, it will be required to format the data in memory after reception. This would help in parallelizing the data processing algorithms. Also, in conventional signal processing systems using hardware and / or CPU based, each pulse data is processed as and when it arrives in pipeline manner, in order to reduce latencies involved if waiting takes place till all the pulse data is arrived. As it will be seen shortly, it is strongly advised to carry out signal processing after receiving all the pulse data when employing GPGPUs for processing than immediate processing as a deviation from conventional methods.

III. GPGPU AND CUDA

The General Purpose Graphical Processing Unit (GPGPU) is a highly parallel computing engine with hundreds (even thousands) of cores available for handling hardcore computations in single precession, double precession floating point. Originally, the GPUs are built for handling the voluminous graphical computations involved in rendering complex 3D graphic images. Several thousands of transformations (typically floating point multiplications, additions) are to be carried out for displaying a meaningful image on screen in real-time. Advances in this have helped in using the computational power for general purpose applications such as fluid dynamics, simulations, signal processing, image processing which are inherently parallel in nature.

The speciality of these GPUs is to simultaneously fetch data for more than one core at a time using wide data bus interfaces available and supply the same for parallel computations by all the available cores as a batch. Though the data is stored in a conventional memory like GDDR5, as the interface is in the order of 256 bits (some-times even upto 384 bits), the system can reach bandwidths beyond 100 GB/s internal for processing. Also, GPUs have hardware thread scheduling mechanisms which will enable very fast context switches reducing the idle times in the processing pipelines.

Most popular GPGPUs are available from AMD and from NVIDIA. The programming model for these GPUs is different from CPU programming. The work carried out by these two vendors in releasing GPGPUs at affordable price lines and ease of programming is appreciable. Popular programming models for these GPUs are available in two flavours viz. Open CL and CUDA. The NVidia GPU can be programmed in both the models, whereas, AMD GPUs can be programmed only in OpenCL. The authors of this paper have worked extensively in CUDA and experimental results were analysed using the same. Though, programming OpenCL is also equally of similar complexity, CUDA was found to be more friendly by the authors (it's only a matter of choice).

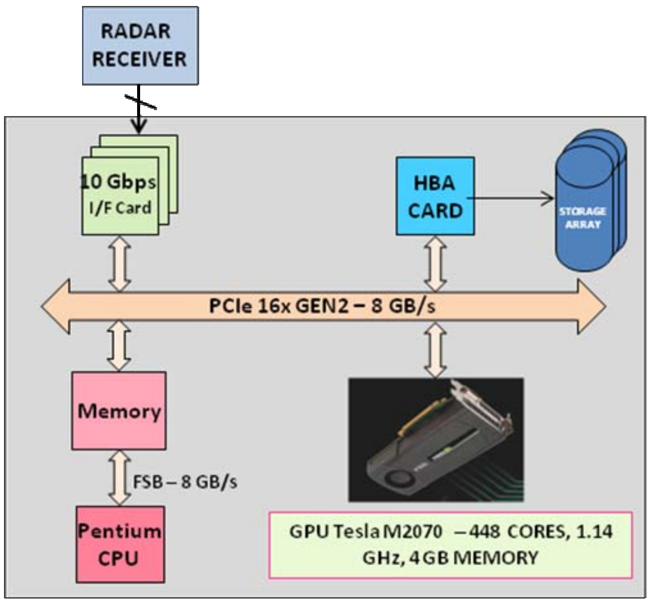


Fig 1 : **A generic hardware platform with one GPU card and interfacing to radar receiver(s)**

Different varieties of GPU boards are available for using as Signal Processing hardware. Boards with 128 cores, 256 cores, 440 cores and 1510 cores are available from NVIDIA. These boards can be housed in general purpose PC / workstations with required power mounted on PCIe 2.0 8x / 16x slots. Depending on the response times required / aimed at, one can choose the appropriate hardware. Even the low cost 128 core GPU may be adequate for a four channel moderate radar signal processing application.

One more advantage that can be kept in view that the application is completely scalable in the sense that, if the application software is developed and tested on a 128 core GPU, same can be executed on another GPU with more cores with absolutely no code changes / modifications. This inherent scalability nature makes the GPU usage for signal processing more promising, as the same software can be run (if designed properly) for a higher functionality (more channels, more samples etc.) radar with minimal / no changes.

IV. GPU PROGRAMMING

The GPU hardware is housed as a slave board in a host computer. The data received from various receiver channels is initially available in CPU memory (called host memory). The GPU, typically, cannot access this memory directly (in some latest GPUs though it is possible with less bandwidth). The GPU has a local memory available (called device memory) from where all the data and instruction fetches are carried out for any meaningful computationally intensive problem solving applications. Therefore, in order to carry out any computation using GPUs, it is first required to make the data available in the device memory by copying all the data from the host memory.

In a typical radar burst, data is received for each pulse and stored in the CPU host memory. It may be possible to copy each of these pulse data into device memory as and when it arrives using *memcpy* functions (these functions are supplied by GPU drivers and uses DMA features) available for this purpose. However, it may not be optimal, as each pulse data movement needs as many function calls in small discrete quantities and for all the channels. An alternate approach is to collect the data for all pulses and all channels, format into a linear data structure that would be useful final processing by GPU and transfer all the data in single memcpy (with DMA). Even if one transfers data for each pulse individually to device memory, it would be of not much use as the parallelism of GPU cannot be effectively used on one pulse data at a time. As a matter of fact, the authors, have found through experiments that it would be less efficient and performance degrades significantly.

Following data structure (C Struct) represents typical I, Q data and it's layout in memory :

```
Struct tagCOMPLEX {
        Float    I,
                 Q;

};

Struct tagPULSE {
        tagCOMPLEX    sSample[1024];
}

Struct tagCHANNEL {
        tagPULSE      sPulses[256];
};
```

Fig 2 presents the memory layout representation in Host and Device memory for one channel.

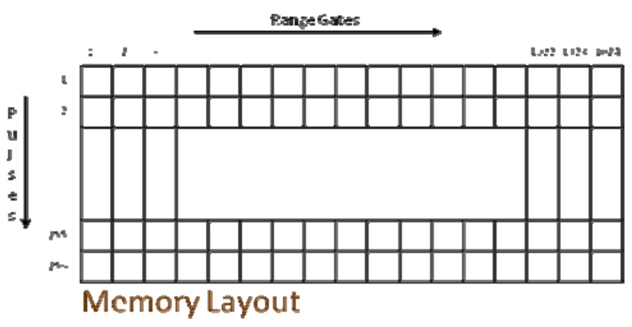


Fig.2 : **Memory layout of I,Q data in Host and device memory**

After transferring the data to the device memory, the algorithms need to be executed using the parallel computation paradigm of GPUs. This, however, needs a thorough understanding of the problem and converting the steps / identifying the inherent parallelism in the required processes. Apply these identified parallelisation unto solving the problem is the most crucial step, though it is simple if properly analysed.

V. RADAR SIGNAL PROCESSING USING GPU PARALLELISATION

In a Radar, signal processing steps are carried out at pulse level and also at burst level. Pulse compression process has to be carried out for each pulse. Similarly, MTI, CFAR in range etc., also to be carried out at pulse level. In case of Doppler Processing (an FFT on each range gate across pulses) has to be carried out at burst level.

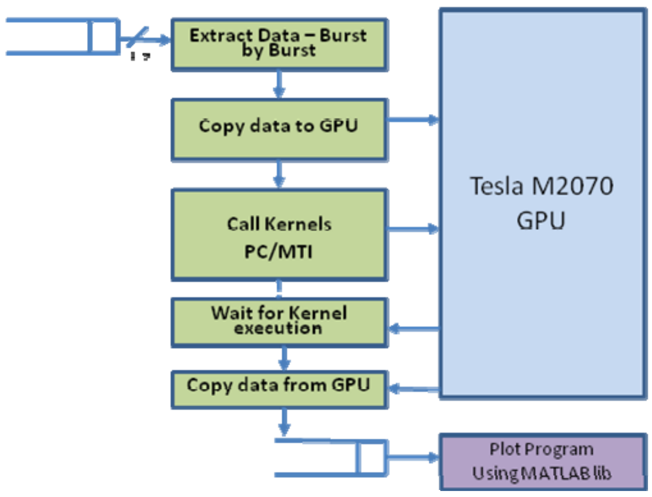

Extract Data – Burst by Burst
Copy data to GPU
Call Kernels PC/MTI
Wait for Kernel execution
Copy data from GPU
Tesla M2070 GPU
Plot Program Using MATLAB lib

Fig.3 : **Steps in Radar Signal Processing using GPU**

As can be visualized from the inherent parallelism of these processes, if the data is available in the device memory, GPU can be programmed / scheduled to process all the pulse data (of all the channels) in parallel i.e. when Pulse compression is being carried out for first pulse, other pulses also can be subjected to the same process in parallel. Also, MTI can be carried out for first three pulses (two pulse cancellation), when other pulses is scheduled in parallel. The natural parallelism defined through CUDA paradigm and GPU capabilities lends itself to highly parallel code, resulting in hitherto impossible programming constructs for solving computationally intensive radar signal processing problems in minimal possible time. Also, as brought out earlier, the applications are scalable both from higher capable GPUs usage and more number of radar channel processing with minimum / no modifications for the implementations. Many other algorithms can also implemented in similar way.

After completing the processing using GPU, certain non parallel steps / algorithms may have to be carried out in CPU (like ambiguity resolution, sorting the plots based on signal strength, data packet formatting, dispatch to data processor etc.). Thus, the processed results have to be copied back to CPU memory using memcpy (with DMA). Further, these results may be displayed using MATLAB like applications at various intermediate steps.

Fig 3 presents steps involved for radar signal processing applications using GPUs.

VI. OPTIMIZATIONS

The signal processing software being highly compute intensive, it would be more beneficial to utilize certain optimization features during software development. Following are more relevant for such High Performance software development :

a. Data arrived from receivers to be stored in page locked memory. Though, this will reduce the physical memory available in the system for other processes, this should not be a concern, as this may be the only application being executed when the system is configured for Signal Processing application. Also, in the current systems, RAM is literally unlimited and is very cheap to expand, if required.

b. Reduce number of memory copies (DMA transfers) from/to host/device memory. Less number of Large blocks of memory transfer would be more efficient than number of smaller block transfers.

c. Memory alignment (called coalescence) would significantly improve the performance due to decrease in memory access conflicts.

d. Utilize maximum registers, shared memory on the device to improve performance.

e. Complete as much of computation on device as possible before transferring the results to host memory. Every transfer on PCIe bus is relatively expensive compared to internal device memory transfers. Therefore, reduce unnecessary data transfers between Host / Device.

f. Single precision floating point operations are much faster than integer operations and double precision (in older generation GPUs). Maximize throughput by using floating point engines.

VII. RESULTS

A sample system has been successfully implemented in the laboratory setup and complete process established. A simulation system for simulating the radar echo at baseband I, Q level was also implemented, as if the data is received from radar receivers in real-time. Algorithms for Pulse Compression (PC) and MTI have been implemented using CUDA programming constructs and parallel kernels for the sample platform using NVidia GPU with 448 cores. Graphs in Fig 4 present some of the results obtained. Also, the same PC algorithm was implemented and executed using MATLAB on a conventional Intel Xeon CPU (8 cores) without using GPUs (given in Table.1). It may be seen that the time taken for PC is significantly low on GPU compared to CPU for different cases. It may be noticed that, the time taken for varying number of pulses and for different number of channels is not linear. This is because of the huge computation power that is available, which we are not able to load fully especially in case of less than 16 channels.

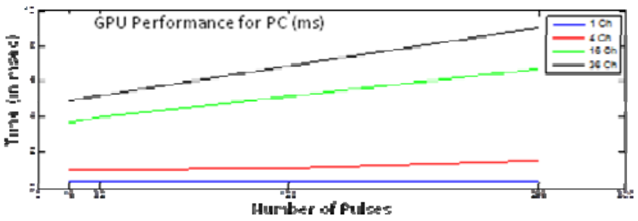


***Fig.4* : Performance of PC on GPU for varying number of channels and pulses (each with 1024 point FFT)**

	GPU (ms) **(Tesla GPU – 448 core)**	**MATLAB (ms)** **(Intel Xeon – 8 cores)**
PC	**1.5**	**60**
MTI	**0.2**	**8**

***Table.1* : Comparison of timings on GPU Vs CPU (1024 point FFT)**

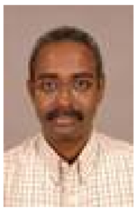
CONCLUSIONS

The Radar Signal Processing is highly demanding real-time software both in terms of computations and response times. More and more sophisticated algorithms are being proposed to get the precise target characteristics particularly in a cluttered target environment for effective use of data during engagement sequences. The GPGPU provides a very stable and implementable platform with high scalability. As it can be seen from the results, the GPGPU outperforms the conventional CPU implementations and also has better software / Algorithm control as compared to customized FPGA implementations. Multiple GPGPUs can be used as a cluster and throughput of several TFLOPS can easily be achieved, making it highly promising.

BIO DATA OF AUTHORS

G.Viswam, Sc'G', has joined LRDE, DRDO in 1989. He received his M.Sc. (Computer Science) from Andhra University in 1989 and M.Tech. (Computer Engg) in 1997 from IIT Madras. He has expertise in software design and development and system integration. His fields of interest are Real time systems, System engineering, integration of large scale systems and High performance computing.

Mr Akshay, Sc 'C' has joined LRDE in 2009. He has a B.Tech (Information Technology) from NIT, Suratkal, India. He has worked for design and development of high performance real-time software for radar applications. He has good experience in C, C++ and CUDA programming.



Dr. A Vengadarajan is working as a Scientist in DRDO since 1986. Received B.E degree in Electronics & Communication Engineering and M.Tech & Ph.D in Microwave Engineering. His areas of interest are Radar Signal Processing, Array Processing, STAP and System Engineering.

Dr. PS Avadhani, is Professor of Computer Science and Systems Engineering in Andhra University since 1986. He has taught Computer science subjects to several Graduate and Post graduate classes over the years and has guided PhD scholars in the department. His areas of interest are Parallel Processing, Cryptography and Data structures

Bibliography :

1. NVidia Team, "CUDA Programmer's reference manual vesrion 4.2", NVIDIA, May '2012

2. Khronos Team, "OpenCL 1.2 Programmer's reference manual", Khronos Group, Nov '2011

3. Mark A Richards, "Fundamentals of Radar Signal Processing", Mc GrawHill, May '2005

4. Fave A Briggs & Kai Hwang, "Computer Architecture and Parallel Processing", McGrawHill, Jul '2012

5. David B Kirk & Wen-mei W. Hwu, "Programming Massively Parallel Processors", Morgan Kaufmann, Dec '2012