High Performance Computing for Tomography Visualizations

Neelam Khatri, Nisheet Saxena
Defence Research and Development Organization
Defence Laboratory, Jodhpur, Rajasthan, India
nisheets@dl.drdo.in

***Abstract*—High Performance Computing (HPC) technology addresses computational problems that demand significant processing power and resources. The goal of HPC is to reduce execution time and accommodate larger and more complicated problems particularly in the domains of scientific research and engineering.**

Computed Tomography (CT) is a powerful Non-Destructive Testing and Evaluation (NDT&E) technique for producing cross-sectional images of scanned components. CT image reconstruction from projection data is computationally intensive because large amount of projection data needs to be processed by time-consuming processing algorithms. The large computational requirement results in large times for CT reconstruction. Problem increases with large size multi-plane datasets for complete 3D visualizations. Running CT algorithms on specialized high-performance hardware Graphics Processing Unit (GPU) can reduce the reconstruction time significantly.

Compute Unified Device Architecture (CUDA) is a software development platform that allows us to run computational algorithms on the NVIDIA GPUs.

This paper presents the reconfigurable design of Filtered Back-Projection (FBP) algorithm for parallel CT image reconstruction using CUDA compatible GPUs. GPU-based FBP implementations have been found to be tremendously faster than CPU-based implementations while processing huge volumes of data.

Keywords—High Performance Computing; GPU; CUDA; Computed Tomography

I. Introduction

High Performance Computing abbreviated as HPC evolved due to increase in demands of processing speed. HPC concentrates on combining the power of computing devices, networking infrastructure and storage systems using parallel hardware, networking and software technology that utilizes human expertise to gain optimized and sustained performance for solving scientific, engineering, big data and research problems including grand challenges like computer modeling, simulations and analysis of data. HPC technology focuses on developing parallel processing systems, parallel programming algorithms and high bandwidth & low latency network by incorporating parallel computing techniques. Parallel computing refers to simultaneous use of multiple compute resources to solve computational problems. HPC systems offer parallelism at large scales, with hundreds and thousands of tasks running concurrently. Many complex algorithms have been developed over the years, to work on highly parallel, specialized hardware platforms. With the rapid progress of parallel hardware, suitably high performance is now available in commodity computers.

In the present study we have utilized the computational power of GPU (Graphics Processing Unit) for the purpose of CT (Computed Tomography) reconstruction which is known to be extremely computationally demanding. We focus on the FBP (Filtered Back-Projection) algorithm for CT reconstruction [1,2,3]. The parallel processing capabilities of the GPUs allows it to divide complex computing tasks into thousands of smaller tasks that can be run concurrently. The proposed algorithm has been implemented using high performance computing framework CUDA (Compute Unified Device Architecture). CUDA provides a fairly simple, minimalist abstraction of parallelism and inherits all the well-known semantics of C, it lets programmers develop massively parallel programs with relative ease [4].

II. GPU Computing

With the ever-increasing demand for more computing performance, the HPC industry is moving toward a hybrid computing model, where GPUs and CPUs work together to perform general purpose computing tasks. GPUs were originally designed for processing graphics and generating good quality games and videos, but the increase of performance in these units is a perfect fit for scientific computing. Today, GPUs have become an integral part of almost every computing device like desktops, smart phones, tablet computers and ultra-books. The wide availability of such devices and their multiprocessing capabilities are proving to be a cost-effective solution in the development of compute-intensive applications [5].

GPU computing is the use of a GPU together with a CPU to accelerate general-purpose scientific and engineering applications. NVIDIA revolutionized GPU computing world in 2006-2007 by introducing its massively parallel architecture called 'CUDA', allowing applications to offload their compute intensive tasks to GPU, instead of the CPU. CPU + GPU is a powerful combination because CPUs consist of a few cores optimized for serial processing, while GPUs consist of thousands of smaller, more efficient cores designed for parallel performance. Serial portions of the code run on the CPU while parallel portions run on the GPU. All NVIDIA GPUs—GeForce, Quadro, and Tesla— support GPU computing and the CUDA parallel programming model.

III. GPU ARCHITECTURE & EXECUTION MODEL

GPUs have many parallel cores that can run simultaneously. Each core can run several threads. Problems with data-parallel computation can be accelerated with millions of threads available on a GPU. A General Purpose GPU (GPGPU) is a modified form of stream (data) processor which transforms a modern graphic accelerator into a general purpose computing device. Modern GPUs are very efficient due to their highly parallelized architecture and devote more resources for computations as compared to CPUs (Fig.1).

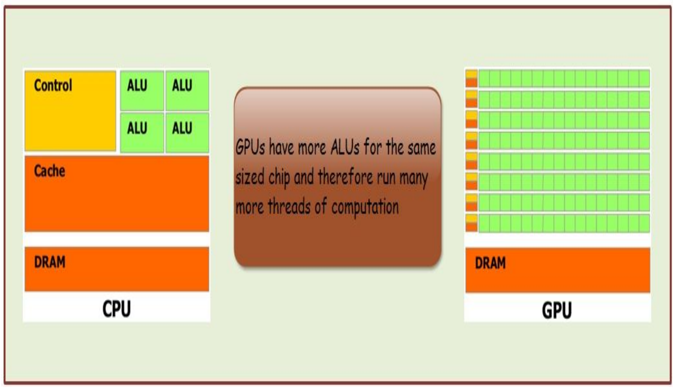
Control
ALU
ALU
ALU
ALU
Cache
DRAM
CPU
GPUs have more ALUs for the same sized chip and therefore run many more threads of computation
DRAM
GPU

Fig. 1. CPU & GPU Architectures

GPUs are designed to provide massive parallelism by hosting and managing hundreds of light weight threads. These threads, execute -- in parallel -- the same set of instructions on the provided data set, providing data parallelism. This approach is called SIMT (Single Instruction Multiple Threads). A GPU also uses various optimization techniques like 'batch execution of group of threads', to improve throughput and reduce bandwidth. Key components of GPU execution model are:

- Kernel -- A set of instructions to be executed in parallel, written as a C function.
- Thread -- Also called 'work-item', is nothing but a single instance of a kernel function that will get executed in parallel.
- Thread-block -- Also called 'work- group', is a group of threads that executes kernel instance, by sharing GPU resources. All the threads to be executed in parallel are divided into blocks of the same size.
- Stream multi-processor -- Executes one or more thread blocks/work groups concurrently (logical parallelism).
- Stream-processor -- Executes one or more threads/work-items in parallel (physical parallelism).

IV. CUDA PARALLEL COMPUTING

Compute Unified Device Architecture (CUDA) is NVIDIA's parallel computing architecture that enables dramatic increase in computing performance by harnessing the power of the GPUs. CUDA provides the ability to use high-level languages such as C with few extensions to develop applications that take advantage of the high level performance and scalability that the GPU's architecture offers [6]. CUDA programming paradigm is a combination of serial and parallel executions. The simple C code runs serially on CPU also called the host. Parallel execution is expressed by the kernel function that is executed on a set of threads in parallel on GPU also called the device. The number of thread blocks, and the number of threads within those blocks that execute this kernel in parallel are given explicitly when this function is called.

The CUDA platform is accessible to software developers through CUDA SDK (Fig.2). The CUDA Toolkit includes CUDA-accelerated libraries, compiler directives, and extensions to standard programming languages. CUDA compiler (nvcc) takes the input files and separates the GPU code and the CPU code. Then GPU code is executed in parallel with CPU code.

The CUDA-accelerated libraries such as cuFFT, cuBLAS, etc., allow users to leverage the floating-point power and parallelism of the GPU without having to develop a custom, CUDA implementation. In our implementation we used cuFFT library to compute the fourier transforms in FBP algorithm for CT reconstruction.

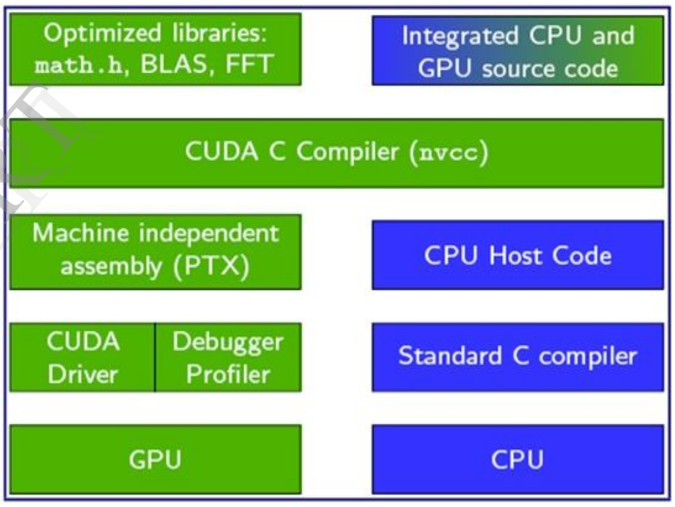
Optimized libraries: math.h, BLAS, FFT
Integrated CPU and GPU source code
CUDA C Compiler (nvcc)
Machine independent assembly (PTX)
CPU Host Code
CUDA Driver
Debugger Profiler
Standard C compiler
GPU
CPU

Fig. 2. CUDA Software Development

On running CUDA's compiled code, its primary execution takes place in CPU. When kernel call is made, the relevant segment is transferred to the GPU for processing but the application continues to execute the non-kernel functions on its CPU. At the same time the kernel function does its execution on the GPU. Thus, computation is processed in parallel between the CPU and the GPU. As the kernel function runs on the device, memory must be allocated on device in advance before kernel function invocation and if the kernel function has to execute on some data then the data must be copied from the host memory to the device memory. Similarly, after the execution of kernel function, data from device memory must be copied back to host memory in order to get results. Keeping all this in view the processing flow is as shown in Fig.3.

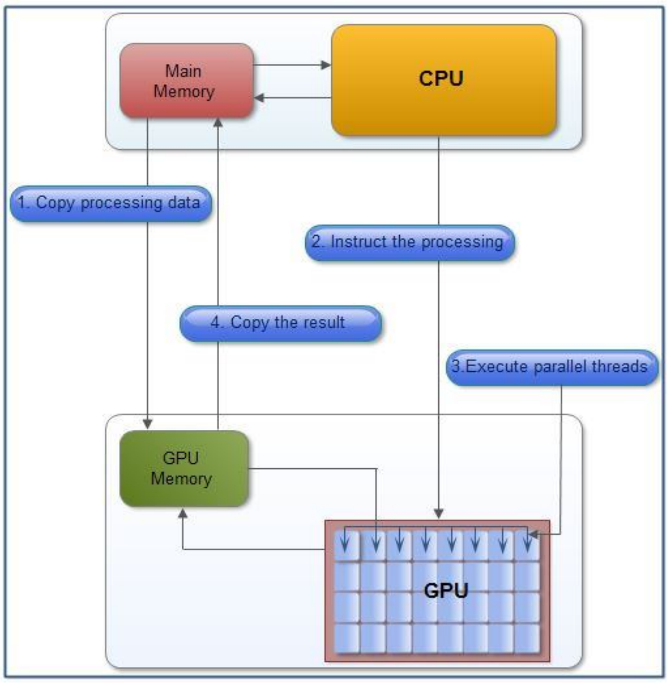

Fig. 3. CUDA Parallel Execution Flow

Like any other parallel programming model, CUDA computing also comes with a few challenges as below:

- Identifying concurrency/parallelism in the given functionality,
- Grouping of parallel tasks,
- Finding the optimal number of threads and blocks that will keep the GPU fully utilized,
- Data transformation and transportation,
- Synchronization among the tasks,
- Test & Measure performance in target environment.

V. Computed Tomography Principles

Computed Tomography is an advanced Non-Destructive Testing and Evaluation technique. CT refers to the cross-sectional imaging of an object from its projection data. The technique of tomography consists of passing a series of rays at multiple points through an object and measuring the attenuation in these rays by placing a series of detectors on the receiving side of the object. These measurements are called projections. This data is collected at various angles typically from 0 to 180 degrees. Specialized algorithms are used to reconstruct 2D cross-sectional images from projection data. The property that is actually computed is the linear attenuation coefficient of that object, at various points in the object's cross-section. If this process is repeated at various heights along the object, we obtain several 2D cross-sectional images, which can then be stacked one on the top of another to get complete internal 3D volume visualization of the object.

In the present study, we have implemented the Filtered Back Projection algorithm for 2D CT image reconstruction from parallel beam projection data. In this algorithm each projected sample is filtered and then individual filtered view is back-projected. The back-projection is formed by smearing each view back through the image in the direction it was originally acquired. The final back-projected image is then taken as the sum of all back-projected views (Fig.4).

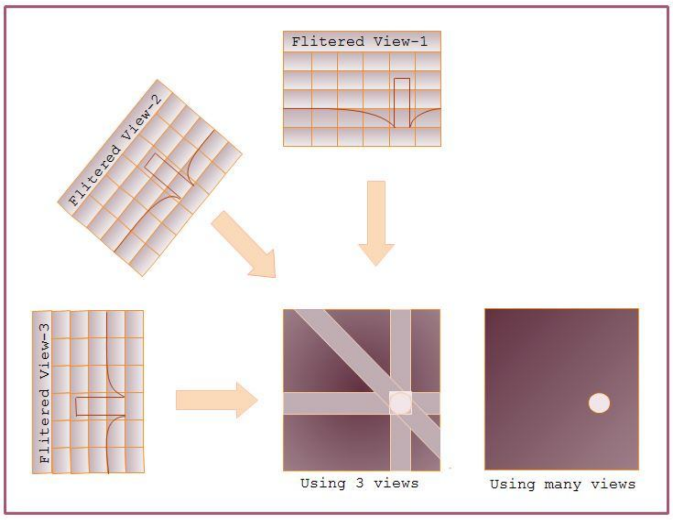

Fig. 4. CT Image Reconstruction using Filtered Back Projection

VI. Computation & Results

In order to estimate the utility of CUDA, parallel beam FBP algorithm for CT imaging has been used. Reconstructions were carried out using non-CUDA variants and CUDA variants. Execution flow for sequential and parallel implementations is shown in Fig.5.

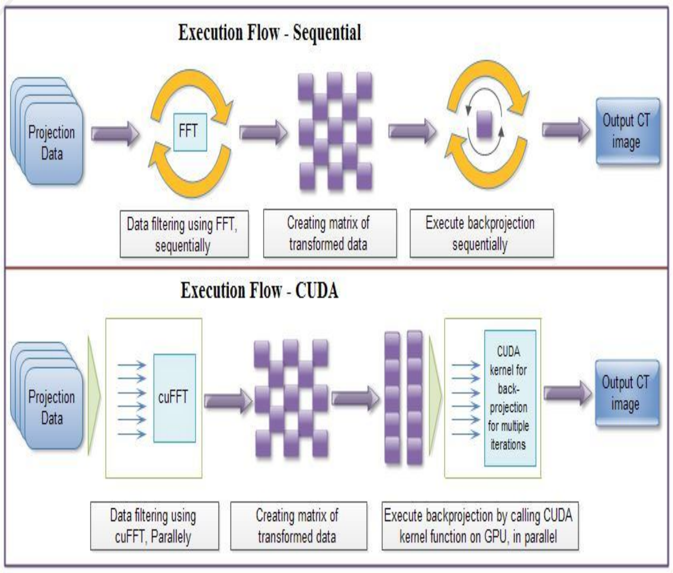
Execution Flow - Sequential
Projection Data
FFT
Output CT image
Data filtering using FFT, sequentially
Creating matrix of transformed data
Execute backprojection sequentially
Execution Flow - CUDA
Projection Data
cuFFT
CUDA kernel for back-projection for multiple iterations
Output CT image
Data filtering using cuFFT, Parallely
Creating matrix of transformed data
Execute backprojection by calling CUDA kernel function on GPU, in parallel

Fig. 5. Filtered Back-Projection Execution Flow in Serial & Parallel Variants

Projection dataset was simulated for the Shepp-Logan Phantom and used for reconstruction using the two implementations. Computation times were noted for different datasets.

Two different computing environments have been used for the study.

A. *System-1 Configuration*

- Processors – Dual AMD Opteron CPU @ 2.6 GHz
- RAM – 32GB DDR2 (333MHz)
- OS – Ubuntu 10.04, 64-bit
- CUDA driver version – CUDA 3.0
- Graphics Processer Unit – Nvidia Quadro fx5600 GPU (128 cores)
- Single precision floating point peak performance of GPU – 518.4 GFLOPS

Computation times observed in above configuration for FBP implementations are shown in TABLE I & II.

B. *System-2 Configuration*

- Processors – Dual Intel Xeon CPU X5660 @ 2.8 GHz
- RAM – 32GB DDR3 (667MHz)
- OS – Ubuntu 12.04 LTS, 64-bit
- CUDA driver version – CUDA 5.5
- Graphics Processor Unit – Nvidia Quadro 6000 GPU (448 cores)
- Single precision floating point peak performance of GPU – 1030.4 GFLOPS
- Double precision floating point peak performance of GPU – 515.2 GFLOPS

Computation times observed in above configuration for FBP implementations are shown in TABLE III & IV.

TABLE I. NO. OF RAYS = 1022, NO. OF PROJECTIONS = 1200, IMAGE GRID = 1022 X 1022 PIXELS

	Filtering Time	**Back-Projection Time**	**Total Time**
FBP in C	5.27 secs.	190.63 secs.	196.04 secs.
FBP in CUDA	1.79 secs.	1.24 secs.	3.22 secs.
Speed up	2.94 X	153.73 X	60.88 X

TABLE II. NO. OF RAYS = 2047, NO. OF PROJECTIONS = 1200, IMAGE GRID = 2047 X 2047 PIXELS

	Filtering Time	**Back-Projection Time**	**Total Time**
FBP in C	9.07 secs.	730.81 secs.	740.57 secs.
FBP in CUDA	1.83 secs.	4.58 secs.	7.19 secs.
Speed up	4.95 X	159.56 X	103.00 X

TABLE III. NO. OF RAYS = 1022, NO. OF PROJECTIONS = 1200, IMAGE GRID = 1022 X 1022 PIXELS

	Filtering Time	**Back-Projection Time**	**Total Time**
FBP in C	3.01 secs.	116.44 secs.	119.75 secs.
FBP in CUDA	1.33 secs.	0.23 secs.	1.68 secs.
Speed up	2.26 X	506.26 X	71.27 X

TABLE IV. NO. OF RAYS = 2047, NO. OF PROJECTIONS = 1200, IMAGE GRID = 2047 X 2047 PIXELS

	Filtering Time	**Back-Projection Time**	**Total Time**
FBP in C	5.56 secs.	472.34 secs.	478.33 secs.
FBP in CUDA	1.37 secs.	0.86 secs.	2.65 secs.
Speed up	4.04 X	549.23 X	180.50 X

Fig.6 shows reconstructed images of Shepp-Logan Phantom from non-CUDA and CUDA implementations for 2047 X 2047 grid size and 1200 projections.

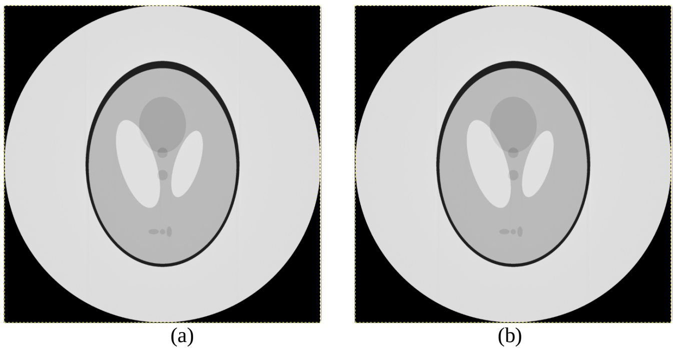

Fig. 6. CT reconstructed images of Shepp-Logan Phantom using (a) CPU & (b) GPU

VII. CONCLUSION

Present study clearly highlights that CUDA provides tremendous speed ups depending upon input datasets and image grid sizes to be reconstructed. Quality of reconstructed image is maintained in CUDA outputs. We conclude that the CUDA-enabled GPU is highly suited for delivering high-speed reconstructions in CT. We believe that for improving performance in the face of increasing complexity and large size of data, it is important to consider multiple performance-improvement techniques.

ACKNOWLEDGEMENT

The authors would like to thank Dr. S R Vadera, Director, Defence Laboratory Jodhpur, for his constant encouragement and support during this research work. We also thank Mr. G L Baheti for his guidance.

REFERENCES

[1] G T Herman, "Image Reconstruction from Projections: The fundamentals of computerised tomography", Academic Press, New York, 1980.

[2] A C Kak, M Slaney, "Principles of Computerized Tomographic Imaging", IEEE Press, New York, 1988.

[3] N Saxena, G L Baheti, D K Tripathi, K C Songara, L R Meghwal, V L Meena, "CUDA-based GPU computing for fast tomography visualizations", Insight - Journal of The British Institute of NDT, Vol 52, No 5, pp 262-264, May 2010.

[4] M Garland, S Le Grand, J Nickolls, J Anderson, J Hardwick, S Morton et al, "Parallel computing experiences with CUDA", IEEE Micro, Vol 28, No 2, pp 13-27, July-Aug 2008.

[5] Amit Badheka, "Using Graphics Card for Accelerated Computing", CSI Communications, Sept. 2013.

[6] "CUDA C Programming Guide", NVIDIA Corporation, 2013.