Performance study of filtered back-projection algorithms implemented on GPUs

Bruno Faria

***Abstract* – In recent years the use of graphical processing units (GPUs) in the diverse fields of science has increase dramatically. This increase is not only due to the GPU tremendous computational power, but also because they are relatively cheap when compared to clusters. In this work we explore the use of the GPU to reduce the computational time of the filtered back projection algorithm on computed tomography. To address the computational gain of the GPU implementation, two CPU implementations have been developed: single-core and multi-core. Both CPU implementations are benchmarked against the GPU implementation. In either case the GPU implementation proved to be a faster alternative. When compared to the CPU implementations the GPU was 273 times faster than the single-core CPU implementation and 34 times than the multi-core implementation.**

***Keywords* – Image Reconstruction, Tomography, Filtered-Backprojection, GPU**

I. INTRODUCTION

Computed tomography (CT) is a very important tool in clinical diagnoses. However, the reconstruction of high quality objects from tomographic data can be time consuming on a standard system. In this sense, it would be highly desirable to have a solution that provides not only real-time but also high resolution reconstruction. Such a solution would provide enormous advantages in both diagnostic and treatment interventions.

Several algorithms can be used on the reconstruction stage, but the most widely used one is the Filtered Back Projection (FBP) [**?**]. The FBP algorithm reconstructs the object from its X-ray projections. In this algorithm each pixel is the result of the sum of the corresponding projection contributions. In order to reconstruct the object a great number of projections is required. Adding this to the fact that for each pixel all projections must be processed, it is clear that this methods are computationally expensive (complexity $O(n^3)$).

One of the technologies that has been gaining more and more recognition are the graphical processing units (GPUs). These devices are suitable for tasks where there is data parallelism. Data parallelism is very important because GPUs are a single instruction multiple data (SIMD) devices. This data parallelism is inherent to the FBP algorithm because for every pixel in the reconstructed image the same functions are applied to the data.

The aim of this work is to implement the Filtered Back Projection algorithm on the GTX 580 and benchmark its performance. The benchmarks are made against single and multi core CPU implementations. In section **??** we will give a brief introduction of what a GPU is, and how it is organized. In section **??** we describe how an object is reconstructed from its projections using the Filtered Back Projection algorithm. The implementations are described in section **??**, and the benchmark results in section **??**. Finally in section **??** we draw some conclusions and directions for future work.

II. AN INTRODUCTION TO GPU DEVICES

The aim of this section is to give a simple introduction to how a GPU is organized. The graphic card used on this work is the GTX-580 with the Fermi GPU GF110. This GPU is organized into 16 highly threaded multiprocessors called Streaming Multiprocessors (SMs) [**?**], [**?**]. A representation of an SM is presented on figure **??**. In the GTX-580 each SM is composed by 32 streaming processors or SPs, for a total of 512 cuda cores.

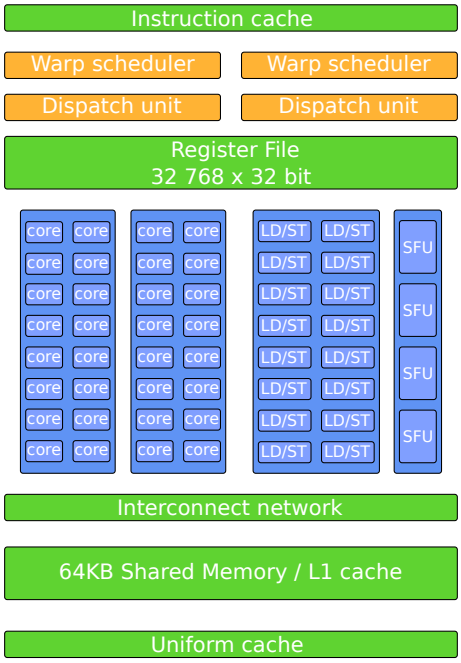

Fig. 1
GF110 STREAMMING MULTIPROCESSOR DIAGRAM.

From a CUDA programmer point of view the GPU is a device that executes a given number of threads at a time. Threads can be identified (indexed) using 1, 2 and 3 dimensions and are organized into blocks. A block can also be indexed by 1, 2 or 3 dimensions and executes on a SM. Till the present date, each block is executed in batches of 32 threads called warps. The maximum number of warps that can reside on a SM is 48. To allow scalability blocks are queued into the GPU.

In a memory respect, the GPU has access to four types of memory which are: i) global memory, ii) texture memory, iii) shared memory and iv) constant memory. Global memory is the biggest region of memory, but it is also the slowest because it is outside of the GPU. Texture memory is a special way of accessing the global memory and only presents advantages when there is data locality. Texture memory is accessed through the texture hardware. Shared memory resides inside of the SM and therefore it is very fast but it is also a very small region of memory. Finally constant memory is the fastest memory on the GPU but it has two downsides. First it is read-only (at least from the GPU side), and second it is a very small region.

In section **??** we will see how these aspects are used in the algorithm parallelization. However, before that we need to understand how the Filtered Back Projection algorithm works, which will be discussed in the next section.

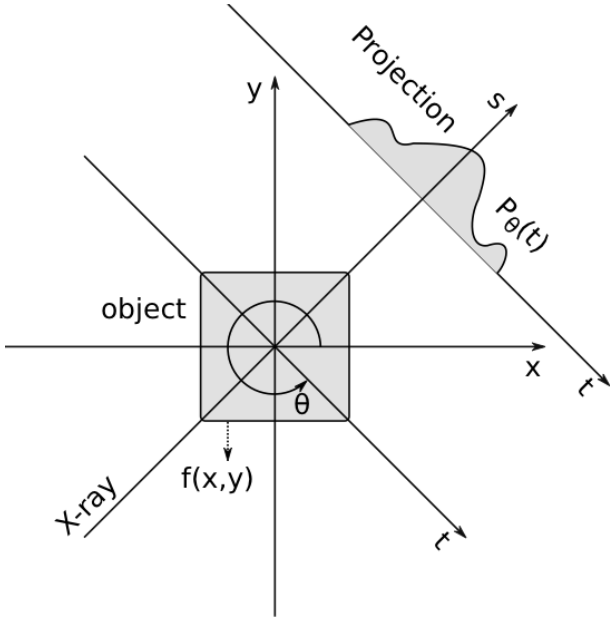

Fig. 2
AN OBJECT $f(x, y)$ AND ITS PROJECTION ARE SHOWN FOR AN ANGLE OF θ

III. TOMOGRAPHIC RECONSTRUCTION PRINCIPLE

Computed tomography (CT) is a powerful non-destructive evaluation technique for producing 2-D and 3-D cross-sectional images of an object from flat X-ray projections. CT images are very useful because they provide a way to characterize the object internal structure such as dimensions, shape, internal defects and density. The CT imaging system is composed by an X-ray source that is directed towards an object from multiple orientations and a system of detectors that measures the correspondent decrease in X-ray intensity.

Here we will only discuss the reconstruction process for a CT with parallel beam geometry. In this geometry, projections of the object are captured by a one dimensional sensor. Each projection is captured for a specific angle ($0 < \theta < \pi$) and placed on a image. The formed grey image is denoted as the sinogram. The grey levels in the image correspond to the X-ray attenuation which is primarily a function of the X-ray energy source and the material composition.

The object is then reconstructed from the multiple projections that compose the sinogram. There are several methods to reconstruct the object from its projections being the most popular one the filtered back projection (FBP) algorithm. In order to understand the FBP algorithm lets consider the process on figure **??**, that describes the acquisition of a given projection.

In this figure x, y denote the object coordinates and $f(x, y)$ the attenuation coefficient of the object under consideration. In this case and assuming there is no dispersion the projection at a given angle θ can be described as:

$$P_\theta(t) = \int\limits_{(\theta,t)line} f(x,y)ds. \tag{1}$$

This expression can be then expressed using the delta function as [?]:

$$P_\theta(t) = \int\limits_{-\infty}^{+\infty}\int\limits_{-\infty}^{+\infty} f(x,y)\delta(x\cos(\theta) + y\sin(\theta) - t)dxdy \tag{2}$$

The above expression is known as the radon transform of $f(x, y)$. A set of this functions for given angles define the object cross-section projections. If the angle is kept constant at a given projection then we get parallel beam projection data, if not we get fan beam data.

The FBP algorithm then reconstructs the object from the projection data by applying a set of steps which can be summarized as follows [**?**] [?]:

1. Compute the one dimensional Fourier transform of each projection as:

$$S_\theta(\omega) = \int_{-\infty}^{+\infty} P_\theta(t)e^{-i2\pi\omega t}dt \tag{3}$$

2. Apply the filter $\rho(\omega)$ in the frequency domain. Several filters can be used in this step such as: Ramp (standard), Cosine, Hamming, Hanning, and SheppLogan . However, for simplicity we will use the ramp filter [?]

defined as $\rho(\omega) = |\omega|$.

$$M_\theta(\omega) = S_\theta(\omega)\rho(\omega) \tag{4}$$

3. Compute the one dimensional inverse Fourier transform of $M_\theta(\omega)$:

$$Q_\theta(t) = \int_{-\infty}^{+\infty} M_\theta(\omega) e^{i2\pi\omega t} d\omega \tag{5}$$

4. Find the back projections. This step is the smearing of the filtered projections back onto the object, and is mathematically represented as:

$$f(x,y) = \int_0^\pi Q_\theta(x\cos(\theta) + y\sin(\theta)) d\theta \tag{6}$$

The last step accounts for more than 80% of the total computational time [**?**]. For images with n by n pixels and n projections the complexity of this step is of the order of n^3. As for every pixel in the reconstructed image it is required to evaluate the integral, this step is a good candidate to be implemented in parallel. In the next section we describe how such parallelization can be achieved.

IV. IMPLEMENTATIONS

For every pixel in the reconstruction step, the same expression needs to be evaluated (**??**). As there is no relation in the reconstruction stage, other than that for each pixel the same data is used. One possible parallelization of this step can be to make each thread process a given reconstructed image pixel (pixel driven approach). This approach led us to consider a two dimensional grid composed of two dimensional blocks. Each block has 256 threads, 16 in each dimension (x and y). A representation of this division is presented in figure **??**.

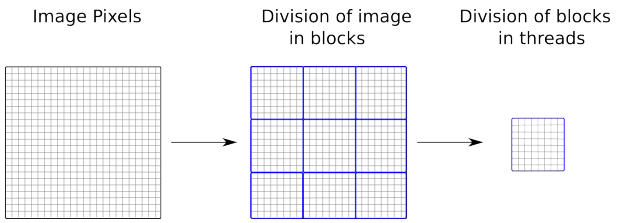

Fig. 3
THE GPU PARALLELIZATION STRATEGY ADOPTED.

The filtered projection data was stored in texture memory. The use of texture memory presents two advantages. First it is a fast access memory when there is a great number of accesses to the same neighbouring address, and second the interpolation is done in hardware. The down side of this approach is that only single precision is available. Due to this restriction the CPU implementations were also done in single precision.

The parallelization strategy adopted in the multi-core CPU implementation (using OpenMP or OMP) was to make each thread process a line of pixels rather than a single pixel. This strategy was adopted to reduce the thread creation overhead. In the case of the single-core implementation, it is identical to the multi-core one but uses only one thread to process all the image pixels.

In the next section we benchmark both CPU and GPU implementations, and present the results of the GPU computational gains.

V. PERFORMANCE RESULTS

In this section we present two types of results. The results arise from the different benchmarks. We benchmark both CPU implementations, single-core and multi-core, and compare them against the GPU implementation. Each benchmark is the result of averaging the time required by the algorithm to reconstruct the object over 10 sequential runs.

To have a better estimation of the computational gains these benchmarks were run for different reconstructed image sizes. The reconstructed image size ranged from $(2^1, 2^1)$ to $(2^{13}, 2^{13})$. The data used in the reconstruction process was generated by applying the *matlab* radon function to the *Shepp-Logan* phantom image.

The test bed system was composed of an intel core i7-980X extreme (CPU) with 24GB of RAM (DDR3), and a GTX-580 DirectCuII (GPU) with 1.5GB of RAM (GDDR5) from Asus. The results of the computational gains can be observed in figure **??**.

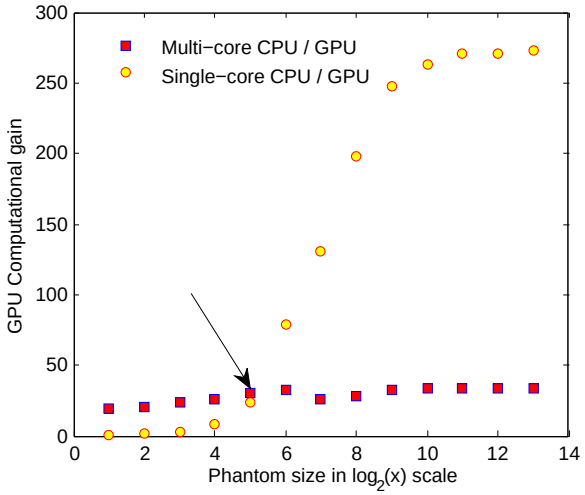

Fig. 4
COMPUTATIONAL GAINS OF THE GPU OVER THE SINGLE-CORE (RED-YELLOW DOTS) AND MULTI-CORE (BLUE-RED SQUARES) CPU IMPLEMENTATIONS

Even though images with sizes of $(2^1, 2^1)$ provide no medical information, they are still useful for benchmarking. In figure **??** it is plotted the computational gain of the GPU implementation over the CPU ones. It can be seen that for images with sizes less or equal to $(2^5, 2^5)$, the CPU single-core implementation is faster than the multi-core one. This happens due to the overhead in the thread creation. It can also be seen that for bigger images the multi-core CPU implementation can out pace the single-core one by a factor

of 8. Nonetheless, in all of the considered cases the GPU implementation was considerably faster than the CPU ones. For a reconstructed image with a size equal or bigger than $(2^{13}, 2^{13})$, the GPU can be 273 times faster than the CPU single-core and 34 times faster than the CPU multi-core implementation. One thing to notice though, is that the GPU only reaches its peak performance when the reconstructed image size is $(2^{10}, 2^{10})$. Which means that only images with sizes bigger than 1024 x 1024 produce enough work to keep the GPU busy.

VI. CONCLUSIONS AND FUTURE WORK

In this work we have shown how the filtered back projection algorithm can be parallelized. Two different CPU implementations have been provided, single-core and multi-core. Benchmarks comparing these implementations against the GPU one have been provided. It was shown that the CPU single-core implementation only presents advantage over the multi-core one when the image size was inferior to $(2^5, 2^5)$. From this point on the CPU multi-core implementation was considerably faster. One important result was that no matter the size of the reconstructed image, the GPU implementation was always faster than the CPU ones. For image sizes bigger than $(2^{10}, 2^{10})$ the GPU implementation can be 273 times faster than the CPU single-core implementation, and 34 times faster than the multi-core one.

Even though in this work the GPU implementation was only done for a slice of data, it proved to be a faster alternative to the CPU implementations. One thing that we intend to explore is if this performance increases or at least is kept, when the reconstruction is performed on multiple slices of data. Another thing to explore is if the iterative reconstruction algorithms can also benefit from the GPU. Iterative algorithms are extremely slow when compared to the filtered back projection algorithms, but provide better image quality when only a small number of projections is available.

REFERENCES

[1] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*, IEEE Press, New York, 1988.

[2] David B. Kirk and Wen-mei W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*, Morgan Kaufmann, 1 edition, Feb.

[3] Jason Sanders and Edward Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, 1 edition, July.

[4] J. Hsieh, *Computed Tomography: Principles, Design, Artifacts, and Recent Advances*, SPIE Press monograph.

[5] Nikhil Subramanian, "Abstract a c-to-fpga solution for accelerating tomographic reconstruction", 2009.