Using Graphics Processing Units to Parallelize the FDK Algorithm for Tomographic Image Reconstruction

Joel Sánchez Domínguez¹*, Luiz Fernando de Oliveira², Nilton Alves Junior³ and Joaquim Teixeira de Assis¹

1. Politecnical Institute of Rio de Janeiro State University, Nova Friburgo-RJ 28625-570, Brazil
2. Physics Institute of Rio de Janeiro State University, Rio de Janeiro 20550-900, Brazil
3. Brazilian center for physical research, Rio de Janeiro 22290-180, Brazil

Received: July 27, 2012 / Accepted: August 31, 2012 / Published: August 25, 2012.

Abstract: The paper presents the implementation of a parallel version of FDK (Felkamp, David e Kress) algorithm using graphics processing units. Discussion was briefly some elements the computed tomographic scan and FDK algorithm; and some ideas about GPUs (Graphics Processing Units) and its use in general purpose computing were presented. The paper shows a computational implementation of FDK algorithm and the process of parallelization of this implementation. Compare the parallel version of the algorithm with the sequential version, used speedup as a performance metric. To evaluate the performance of parallel version, two GPUs, GeForce 9400GT (16 cores) a low capacity GPU and Quadro 2000 (192 cores) a medium capacity GPU was reached speedup of 3.37.

Key words: Computed tomography, images reconstruction, FDK algorithm, GPUs, CUDA-C, parallel processing.

1. Introduction

Tomography is imaging by sections or sectioning, through the use of any kind of penetrating wave [1]. With this technology it is possible to obtain images from inside of solid objects without destroying them and get images in places where traditional methods cannot be used. It is successfully applied in various fields how radiology, archeology, biology, geophysics, oceanography, materials science, astrophysics and other sciences.

Variations of tomography known as CT (computed tomography) involve gathering projection data from multiple directions and feeding the data into a tomographic reconstruction software algorithm processed by a computer [1, 2]. This process called image reconstruction involves the manipulation of many images [3] which can be large or medium size as a function of quality generating a computationally expensive process. Generating a better reconstructed image as soon as possible is one of the continuing challenges in computed tomography.

A graphics processing unit or GPU is a specialized microprocessor that offloads and accelerates graphics rendering from the central processor [4]. The highly parallel structure of GPUs makes them more effective than general-purpose CPUs for a range of complex algorithms, especially those involving a data processing that could be parallelized [5]. Those are associated with image reconstruction process.

The use of a GPU do general purpose scientific and engineering computing, as known as GPU computing or GPGPU (general purpose computation on graphics hardware) [6, 7]. The GPGPU is being applied in different fields of science and engineering with excellent results improving the time to obtain experimental results by speed up the computations [8].

The CUDA C programming language was
developed by NVIDIA to facilitate the writing and development of applications using GPUs [4]. Enabling the expansion of this technology to the greatest number of software developers and the widespread use of the potential of GPGPU in the areas required to perform data processing with high cost computationally [9].

2. Computed Tomography

CT can be analyzed as a two-step process: in the first stage the body under study is explored using some type of radiation to produce detailed images of axial body. Multiple images or X-ray are obtained by rotation around the body of the set source-detector as shown in the Fig. 1 [10].

While the radiation passes through the body, part of it is absorbed depending on absorption coefficient of the material. Measuring the radiation that reaches the detector is possible to calculate the attenuation coefficient of the material and generate two-dimensional images of the sections of the body. These X-ray are stored on the computer [1, 10-13]. The Fig. 2 shows one X-ray of human tooth, one of the body used to test the serial and parallel version of the FDK algorithm.

![Fig. 1 Exploration of the samples in CT.](image)

In the second step, the set of images obtained are processed by a computed algorithm to generate a three-dimensional image; where it is possible to differentiate the internal structures of the body. These computed algorithms are called the reconstruction algorithms [1, 3].

2.1 Feldkamp, David e Kress Reconstruction Algorithm

The reconstruction algorithm calculates a numerical value for the absorption coefficient of each point of the volume, and then translates this numerical value in a shade of gray [14-16]. Offering an image where the structures with different absorption coefficients can be identified and studied.

The most used reconstruction algorithm is FDK (feldkamp david e kress) algorithm [3]. Is an analytical reconstruction algorithm for cone beam with circular trajectory of the radiation beam. We describe the algorithm without a rigorous mathematical character, which can be found in the references [14, 17-19].

When performing a tomographic analysis the intensity of emerging photons is proportional to the integral of all the attenuation coefficients $\mu(x, y)$ along a straight line.

$$g(r, \theta) = \ln(\frac{I_0}{I}) = \int_{r, \theta} \mu(x, y) dxdy \quad (1)$$

Where the subscript $r$ represents the measurements made on different parallel positions separated by a constant distance $\Delta r$, $e$, $\theta$ the angle of rotation of the shaft $(x, y)$ obtained in angular steps $\Delta \theta$.

Rewriting Eq. (1) for fan-beam geometry using the density function $f(r, \theta)$ from its projections $p(l, \theta)$:

$$f(r, \theta) = \frac{1}{4\pi} \text{Re} \int_0^\infty \int_0^{2\pi} p(l, \theta) e^{-i(l-r \cos \theta)} \, \text{d}l \, \text{d}\theta \quad (2)$$

Developing Eq. (2) for cone beam geometry, we obtain the expression of FDK algorithm:

$$f(\vec{r}) = \frac{1}{4\pi^2} \int_0^{2\pi} \int_0^\infty \frac{d^2}{(d+\vec{r} \cdot \vec{z})^2} \hat{P}(Y(r), Z(r)) \, \text{d}l \, \text{d}\theta \quad (3)$$

Where:
Using Graphics Processing Units to Parallelize the FDK Algorithm for Tomographic Image Reconstruction

\[
\tilde{P}_\phi(Y,Z) = \int_{-\frac{\Delta Y}{2}}^{\frac{\Delta Y}{2}} g_Y(Y) \mathcal{P}_\phi(Y',Z) \frac{d}{(d^2 + (Y - Y')^2 + (Z - Z')^2)^{3/2}} dY'
\]

\[
g_Y(Y) = \text{Re} \int_0^{\mathcal{W}_Y} W \ e^{2 \pi i Y Y} \ dW
\]

To make the discretization of Eq. (3), it is considered that \( P_{\phi}(Y, Z) \) varies much more slowly than \( g_Y(Y) \), so that it can be considered constant in each discretization interval.

\[
\hat{P}_\phi(Y_j, Z_k) = \sum_i P_{\phi_i}(Y_{i,j}, Z_{i,k}) \cos \phi_{i,k}
\]

\[
\int_{Y_{j-\frac{\Delta Y}{2}}}^{Y_{j+\frac{\Delta Y}{2}}} g_{Y}(Y - Y_j) dY \int_{Z_{k-\frac{\Delta Z}{2}}}^{Z_{k+\frac{\Delta Z}{2}}} g_{Z}(Z_k - Z) dZ
\]

The Eq. (6) showed the discrete form of FDK algorithm; to be implemented in test versions parallel and serial in this article.

3. Serial Version

Measuring the performance of the parallel version is necessary to program a serial version of the algorithm. The input data of the software are: number of X-ray, height and width of X-ray, X-ray images and initial and final slice of the reconstruction. The output of the software is a raw image that contains the slices reconstructed. In CT the RAW format is used to store the complete sequence of images obtained at the exploration of the body [15].

A flow chart is showed in Fig. 3:

3.1 Detailed Description of the FDK Algorithm Implementation

The steps to software implementation the FDK algorithm are:

1. Generate the reference image, where each pixel will be different between the arithmetic mean and filtered arithmetic mean of the pixels the input images;
2. For each one of the slices of reconstruction:
   - Loop through the set of input images, grouping the images into subgroups of 4 images and calculate for each subgroup the vectors: origin of the beam, orientation of the beam and perpendicular to the beam;
   - For each pixel of each plane by calculating the dot position in the array output and contribution that receives the input images according to the filtering of the relationship between the attenuation coefficient of the reference and attenuation coefficient of the input projections;
3.2 Normalize the pixels of the slice in grayscale

Normalize each pixel of each slice according to the lowest and highest value calculated between the pixels of the reconstruction.

The serial version of the algorithm allowed reconstructing the slices of the body. The Fig. 4 shows one slice reconstructed with the serial version.
4. Graphic Processing Units

The GPU is a specialized microprocessor to compute intensive, highly parallel, and therefore more transistors are dedicated to processing data. Their structure allows them a high computational power that exceeds by orders of magnitude the computational power of CPUs (central processing units) [4]. This feature will call the attention of researchers, and they began to leverage the capabilities of GPUs to parallelize complex algorithms and reduce its processing times [20-24].

The use of the first GPUs for general purpose computing has some limitations. They were eliminated with the development of the CUDA (computed united device architecture) and the programming language CUDA-C. These advances allowed the massive use of this technology by a large number of software developers and researchers.

The CUDA programming paradigm is a combination of executions in series and in parallel, where the sequential part of the application is executed on the CPU or host and the compute-intensive part is accelerated by the GPU or device to decompose the processing into sub-parallel processes [4].

The CUDA-C extends standard C and introduces a new type of functions called kernels, which allow you to implement parallelism. The kernels functions have two new parameters to specify the number and structure of threads, the segment of the program to be run in parallel will execute. These parameters are number of blocks and number of threads per block [4].

5. Parallel Processing

A parallel processing system is a set of processors that work cooperatively to solve a computational problem [25, 26]. Some examples are: supercomputers, networked computers forming a cluster, computers with multiple processors [27] and graphical processing units. Although these technologies are currently evolving, at present days, between these systems, GPUs have the best cost/benefit. GPUs give a tempting option for researchers to experience the possibilities of high-performance processing.

The design and analysis to build a parallel application is independent of the technology used for the parallel processing system. The methodology used to develop the parallel version of the FDK algorithm was [15]:

1. Sequential implementation: to analyze, implement and validate a sequential solution to the problem that we intend to solve;
2. Analysis of the division of work: to evaluate the possibility of dividing the data set of the problem between the different processes;
3. Assess the viability of pure data parallelism: to see if the problem can be solved only by running the serial algorithm in the different data sets;
4. Analysis of the need for communication: if data parallelism is not sufficient, identify the needs of communication between processes;
5. Evaluate the need for parallelism control: to analyze the need to introduce parallelism control in the implementation of parallel solution;
6. Validation of the parallel implementation: to check the parallel solution with the help of the serial algorithm;
7. Performance analysis: to evaluate different performance metrics to analyze the characteristics of the parallel algorithm implemented.

Some metrics are used to evaluate the performance of parallel algorithm. The speedup definition as the ratio between the runtime of the serial algorithm and the runtime of the parallel algorithm is the most used [27]. It is used for us to assess the improvement of our parallel implementation.

6. Parallel Implementation of FDK Algorithm

Once implemented and validated the serial version, it is analyzed to identify possible sub-process to be parallelized. The most intensity computational part of
the software is the FDK algorithm. It will be parallelized and executed on the GPU. The modified flow chart shows the division of processing between the CPU and GPU, the section of the program that will continue being processed serially (CPU) and section of the program that will run in parallel (GPU). The Fig. 5 shows the flow chart of software, identifying the process will be run in the CPU or in the GPU.

The FDK algorithm was divided in three steps, according to the description of the section 3.1. Each one of these steps was parallelized using independent kernels.

Parallelizing the generation of the reference image was used one kernel, with the number of threads equal to the number of image pixels to be generated; by using each thread to compute the value of each pixel of the image.

To parallelize the compute of each one slice of the reconstruction was used one kernel, with the number of threads equal to the number of slices to be reconstructed; by using each thread to generate each slice of the volume will be reconstructed. It was analyzed decompose this process using other kernels in the internal stages of step 2. But the data dependency prevented the possibility of parallelize these internal steps.

Finally, to parallelize the normalization of each of the pixels of all slices of the reconstruction was used one kernel, with the same structure to the threads that the kernel used in the previous step.

6.1 Validation of Parallel Version

The parallel version has to deliver equal or equivalent results with serial version. The Fig. 6 shows the same one slice reconstructed with the serial version showed on Fig. 4, now reconstructed using parallel version.

To compare the two images, the images produced by serial and parallel versions respectively, the subtract images command was used, from the image processing software ImageJ. Fig. 7 shows the image of the difference between the images of Figs. 6 and 4.

Since the values of the pixels of the two images

![Fig. 6 Slice reconstructed with parallel version.](image)

![Fig. 7 Difference between images produced with serial and parallel versions.](image)
are very similar, the difference of them produces a predominately black image. By varying the image contrast we can better observe the distribution of these differences as is show in Fig. 8.

In the Fig. 8, it is verified that the differences between the pixels have the same distribution as the materials with different coefficients of attenuation in the slice. We can explain these differences considering the variations of numeric processing, and the normalization processes of the FDK algorithm.

Some metrics can be used to compare the two images, as the distances and energy [14].

7. Performance Tests

To evaluate the performance of the parallel version about the serial version were used X-ray or projections of two bodies test. 500 projections with size $400 \times 308$ pixels of a human tooth and 180 projections with size $1024 \times 1024$ pixels of a piece of concrete were made. Sample images of a projection and a reconstructed slice for the first test body were shown in the Figs. 2 and 4 respectively; by stacking all slices of the reconstruction process it is possible to visualize the complete reconstructed volume, Fig. 9.

On the Figs. 10 and 11 samples images of one projection and one rebuilt slice for the second test body is shown; and the Fig. 12 shows the complete volume.

---

Fig. 8  Contrasted image of the difference between images.

Fig. 9  Reconstructed volume for first test body.

Fig. 10 Example of X-ray of second test body.

Fig. 11 Slice reconstructed of the second test body.

Fig. 12 Reconstructed volume for second test body.
The parallel version of the algorithm was tested on two GPU cards, NVIDIA Geforce 9400 (GPU1) and NVIDIA Quadro2000 (GPU2). The GPU1 is a low capacity GPU with only 16 cores, core clock 550 MHz and processor clock 1400 MHz. While the GPU2 is a medium capacity GPU with 192 cores, core clock 625 MHz and memory clock 1300 MHz.

7.1 Comparison of Versions

Tests were made for different amounts of input data and output slices, to evaluate the versions for different volumes of data. The Table 1 shows the runtime in seconds for both versions of FDK algorithm to the first test body; line 1 displays the number of input projections (Input) and the number of slices reconstructed (Output); the next lines shows the runtime in seconds for serial version (CPU) and parallel version on both GPUs, GPU1 and GPU2.

With the same structure of the Table 1, Table 2 shows the runtime in seconds for the second test body.

Analyzing Tables 1 and 2, it appears that the GPU1 does not decrease the runtime of parallel version compared with serial version, while the GPU2 can reduce this runtime. This behavior is repeated for all of the studied data volumes. Highlighting that since the volume of data being processed increases, the GPU2 achieves best execution time while the GPU1 worsening its running time, if to be compare with the serial version runtime. The number of pixel of the X-ray of the test bodies, of $1,232 \times 10^4$ for the first body and $10,486 \times 10^4$ for the second body, which directly influences the volume of data processed, has a direct relation to the processing time algorithm.

Tables 3 and 4 showed the computation of speedup on GPU2 for the two test bodies.

Tables 3 and 4 show that speedup is small if the volume of data processed is small, and increasing the volume of data improves performance of parallel version until a certain limit. Despite the size of the input x-rays have several orders of difference the speedup has only a small decrease between the first

### Table 1  Runtimes in seconds to the serial and parallel versions for first test body.

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>100/10</th>
<th>100/50</th>
<th>100/75</th>
<th>100/100</th>
<th>200/100</th>
<th>200/200</th>
<th>500/300</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>15.8</td>
<td>92.3</td>
<td>137.7</td>
<td>180.6</td>
<td>358.5</td>
<td>669.3</td>
<td>2561.8</td>
</tr>
<tr>
<td>GPU1</td>
<td>43.5</td>
<td>199.4</td>
<td>298.7</td>
<td>400.4</td>
<td>780.5</td>
<td>1450.2</td>
<td>5420.1</td>
</tr>
<tr>
<td>GPU2</td>
<td>12.1</td>
<td>30.25</td>
<td>44.1</td>
<td>57.4</td>
<td>113.2</td>
<td>209.1</td>
<td>760.4</td>
</tr>
</tbody>
</table>

### Table 2  Runtimes in seconds ($10^2$) to the serial and parallel versions for second test body.

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>100/10</th>
<th>100/50</th>
<th>100/75</th>
<th>100/100</th>
<th>200/100</th>
<th>200/200</th>
<th>500/300</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1.34</td>
<td>7.87</td>
<td>11.70</td>
<td>15.35</td>
<td>27.64</td>
<td>54.70</td>
<td>78.35</td>
</tr>
<tr>
<td>GPU1</td>
<td>3.74</td>
<td>17.34</td>
<td>25.99</td>
<td>35.03</td>
<td>63.07</td>
<td>123.9</td>
<td>189.21</td>
</tr>
<tr>
<td>GPU2</td>
<td>1.10</td>
<td>2.63</td>
<td>38.74</td>
<td>5.06</td>
<td>9.03</td>
<td>17.70</td>
<td>25.27</td>
</tr>
</tbody>
</table>

### Table 3  Speedup calculation on GPU 2 to first test body.

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>100/10</th>
<th>100/50</th>
<th>100/75</th>
<th>100/100</th>
<th>200/100</th>
<th>200/200</th>
<th>500/300</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU2</td>
<td>1.30</td>
<td>3.05</td>
<td>3.12</td>
<td>3.14</td>
<td>3.16</td>
<td>3.20</td>
<td>3.37</td>
</tr>
</tbody>
</table>

### Table 4  Speedup calculation on GPU 2 to second test body.

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>100/10</th>
<th>100/50</th>
<th>100/75</th>
<th>100/100</th>
<th>200/100</th>
<th>200/200</th>
<th>500/300</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU2</td>
<td>1.22</td>
<td>2.99</td>
<td>3.02</td>
<td>3.03</td>
<td>3.06</td>
<td>3.09</td>
<td>3.10</td>
</tr>
</tbody>
</table>
and second test body.

With a medium capacity GPU, Quadro 2000, we get decrease in 3.37 times the processing time of the FDK algorithm.

8. Conclusions

Was implemented a parallel version of the FDK algorithm for three-dimensional reconstruction of tomographic images using graphics processing units and the CUDA-C programming language. Developed parallel and serial versions of the FDK algorithm and proved the equivalence between them. Using GPUs as an option to perform parallel processing and speed up the calculations of complex algorithms.

It was shown that GPUs with low capacity was not possible to obtain gain speed in the parallel version of the FDK algorithm. However it is valid to use these cards in the learning and experimentation with GPU technology. With medium capacity GPU was reached 3.37 of speedup. We assume that with high capacities GPUs is possible to reach higher speedups.

The speed gains achieved relatively modest compared with the gains reported by other authors using GPUs, are the product of the high complexity of the reconstruction algorithm with high dependence of data between processes or tasks to parallelize. Resulting in heavy threads is a not recommended for general purpose computing on GPUs.

References

Using Graphics Processing Units to Parallelize the FDK Algorithm for Tomographic Image Reconstruction

170-176.


