Send Orders of Reprints at reprints@benthamscience.org

Open Access

Cuda Parallel Implementation of Image Reconstruction Algorithm for Positron Emission Tomography

Belzunce MA[1,2,*], Verrastro CA[1,2], Venialgo E[1,2] and Cohen IM[1,3]

[1]Departamento de Electrónica, Facultad Regional Buenos Aires, Universidad Tecnológica Nacional. Medrano 951, (C1179AAQ) Ciudad Autónoma de Buenos Aires, Argentina

[2]Instrumentación y Control, Comisión Nacional de Energía Atómica. Presbítero Luis González y Aragón Nro.15, (B1802AYA) Ezeiza, Buenos Aires, Argentina

[3]Facultad Regional Avellaneda, Universidad Tecnológica Nacional. Av Mitre 750, (B1870AAU) Avellaneda, Buenos Aires, Argentina

Abstract: Although the use of iterative algorithms for image reconstruction in 3D Positron Emission Tomography (PET) has shown to produce images with better quality than analytical methods, they are computationally expensive. New Graphic Processor Units (GPUs) provide high performance at low cost and programming tools that make it possible to execute parallel algorithms in scientific applications. In this work, a GPU parallel implementation of the iterative reconstruction algorithm MLEM 3D has been developed using CUDA, a parallel model from NVIDIA. The Siddon algorithm was used as Projector and Backprojector. Acceleration factors up to 85 times were achieved, with respect to a single thread CPU implementation. The performance in GPU with Tesla and Fermi, which are respectively the first and the last generation of CUDA compatible architectures, has been compared. The image quality in each platform has been analyzed, showing a higher level of noise in GPU, due to race condition problems. The new features of Fermi architecture permitted to solve this problem using atomic operations.

Keywords: PET, Iterative image reconstruction, Graphics Processing Units, parallelization, CUDA.

1. INTRODUCTION

In Positron Emission Tomography (PET) an Image Reconstruction Algorithm is needed in order to generate an image from the measured projections, which quantifies the distribution of the radiotracer in the patient's body. Iterative Algorithms have shown better results [1, 2], regarding image quality and resolution, than the traditional methods based on Radon transform. On the other hand, iterative algorithms are computationally expensive; besides, the amount of involved data in 3D PET increased considerably with the improvements in scanner's spatial resolution. For this reason, multi-core processors, clusters and graphics processor units (GPUs) are currently used to attain reconstructed images in a practical time. Different parallelization strategies were proposed for those platforms [3-5].

First generation GPUs were a potential solution to provide high performance at low cost, but limited by their fixed function architecture. New Graphic Processor Units have rapidly evolved from a fixed-function processor, designed to satisfy graphic pipeline requirements, to a parallel programmable processor with a unified architecture. In addition, the introduction of CUDA in 2006 has provided an easier way to exploit the computational horsepower of GPUs and use them for general purpose applications [6, 7], thus allowing to replace traditional reconstruction architectures. First implementations of iterative reconstruction algorithms for GPU, developed before CUDA release, had shown promising results [8-11]. The use of CUDA [12], NVIDIA parallel processing architecture for general processing, in PET image reconstruction algorithms, has already proved to be an efficient tool to achieve speed ups of the computing times at low cost of hardware and development [13, 14].

A CUDA implementation of MLEM algorithm is presented in this work. Its performance was evaluated in two different GPUs: GTX-285, with the first CUDA architecture, Tesla [15]; and GTX-480, a board with Fermi architecture [16]. This comparison allows to verify the scalability of CUDA architecture and tests some of Fermi's innovations. All the GPU results, including time performance and image quality, were compared with a single core CPU implementation.

1.1. MLEM Algorithm

Iterative methods estimate the reconstructed image progressively, in order to reach the best solution according to a proposed model of the acquisition system. The best-known and most used algorithms are the Maximum-Likelihood Expectation-Maximization (MLEM) [1, 17] and its accelerated version Ordered Subsets Expectation Maximization (OSEM) [18].

*Address correspondence to this author at the Centro Atómico Ezeiza, Comisión Nacional de Energía Atómica; Tel: (+54) (11) 6779 8277. Fax: (+54) (11) 6779 8433. E-mail: martin.a.belzunce@gmail.com

In OSEM, the projections data set is divided into groups in order to accelerate the convergence of the MLEM algorithm. The strategy of this division depends on the kind of projection data being used and could affect the convergence of EM process. As the objective of this work is to evaluate the performance of GPU against CPU, a MLEM implementation, which could be considered as an OSEM with one subset, has been programmed.

MLEM algorithm uses the Maximum Likelihood as estimator and the Expectation Maximization as optimization algorithm. In MLEM the Poisson nature of the positron emission process is taken into account and a System Response Matrix (SRM) is needed to model the acquisition process. The iteration algorithm is formulated as:

$$x_j^{k+1} = \frac{x_j^k}{\sum_{i=1}^{m} a_{ij}} \cdot \sum_{i=1}^{m} \frac{a_{ij} . b_i}{\langle a^i, x \rangle} \qquad (1)$$

where x_j^k is the pixel *j* of *x* image at the *k*th iteration; b_i is the bin *i* of the measured projection (input projection), and a_{ij} is *(i,j)* coefficient of the SRM. This coefficient represents the probability that an emission in pixel *j* is detected in the line-of-response (LOR) *i*. This probability can consider geometrical as well as physical factors.

Three stages could be identified for each iteration: forward projection, backprojection and normalization. The first one is represented by the term

$$p_i = \langle a^i, x \rangle = \sum_{j=1}^{n} a_{ij} . x_j \qquad (2)$$

which projects the kth volume image into an *estimated projection*, using the SRM. The backprojection stage generates an *image with adjust coefficients* from the projection generated dividing bin to bin the *measured projections* by the *estimated projections*:

$$x\_adj_j = \sum_{i=1}^{m} a_{ij} . \frac{b_i}{p_i} \qquad (3)$$

Finally, in the normalization process, the actual image pixels (x_j^k) is multiplied pixel to pixel with the image of *adjust coefficients* ($x\_adj_j^k$) and then divided by a *sensitivity image*, setting up the new iteration image:

$$x_j^{k+1} = \frac{x_j^k}{\sum_{i=1}^{m} a_{ij}} . x\_adj_j \qquad (4)$$

The *sensitivity image* is generated before starting the iterative process, making a forward projection of all a_{ij} coefficients:

$$S_j = \sum_{i=1}^{m} a_{ij} \qquad (5)$$

1.1.1. Projector

In our implementation, a_{ij} coefficients are calculated using only geometrical information. The projector used was the Siddons algorithm [19] in its Jacob's fast version [20]. We selected this projector as it can be used in histogram and list-mode projection. This algorithm is very efficient programmed in a CPU, but it is not the best solution for GPU platforms because of its many conditional branches. However, we demonstrated that significant acceleration factors can be achieved utilizing this algorithm dividing the whole projection of the sinogram by one thread per bin.

Siddons algorithm consists of doing a ray tracing of a line of response (LOR) then obtaining the length of the segment generated by the intersection between the LOR and the pixels of the image. Each segment length is called the a_{ij} Siddon's coefficient (Fig. **1**). In order to accomplish this operation, it is necessary to calculate the line equation of the LOR from the coordinates of the corresponding sinogram's bin. The starting and final points are obtained by calculating the intersection point between the LOR and the scanner Field of View (FOV). Afterwards, the ray tracing is done by going from the starting to the final point. (Fig. **1**)

1.2. CUDA

CUDA [12] is a general purpose parallel architecture, with a programming model and software environment that allows developers to use C-CUDA, a high level programming language, in applications with fine-grained parallelism and execute them in massive parallel threads.

A CUDA kernel is a function that is executed N times in N parallel CUDA threads. The programmer organizes these threads in thread blocks and grids of thread blocks. A kernel is launched in a grid of parallels thread blocks, each thread having a thread and block ID. Threads within a thread block execute an instance of the kernel with its own resources and can cooperate among themselves through barrier synchronization and shared memory. Thread blocks are executed independently and are scheduled in any order in the available processors. Grids of thread blocks share results in Global Memory space after kernel-wide global synchronization.

In CUDA graphic processors, threads within a block are executed and scheduled in groups of 32 called warps. Threads of a warp start in the same program address and execute simultaneously thanks to SIMT (Single Instruction Multiple Thread) Architecture. When threads in a warp are affected by flow control operations and follow different execution paths, their execution diverge and each execution path is serialized. This increases the total number of instructions executed per warp.

CUDA devices use several memory spaces. Each thread has its own private local memory and registers space. Threads within a block have a common shared memory, visible for all of them. All threads of every block have access to a large global memory space. Finally, constant and texture memories define two read-only and cached memory spaces. Shared memory is a low latency and fast memory, because its on-chip location. Global memory loads and stores of a half warp (Tesla) or a warp (Fermi) can be coalesced into one transaction when certain requirements are met. This is achieved when threads of the same warp access memory that fits into a segment size of 128 bytes for 32-bits words. For Compute Capability 1.2 or more, any access pattern within a segment is coalesced.

CUDA GPUs have a compute capability that describes the features of the hardware and establish the set of

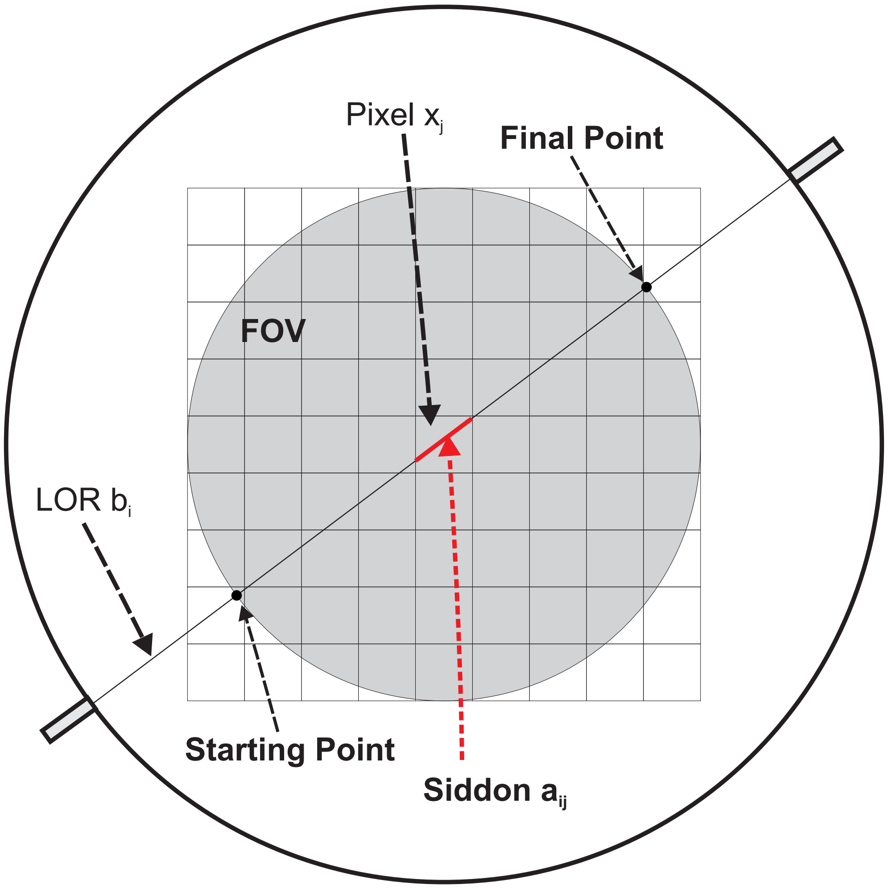

Fig. (1). Siddon Algorithm. Line equation of a LOR is calculated and intersection points with field of view are obtained. From the starting point, line path is covered, and the length of every intersected pixel is computed.

instructions supported by the device. It also specifies hardware parameters, such as the maximum number of threads per block or registers available per multiprocessor.

Tesla [15] was the first NVIDIA architecture to use CUDA model. These GPUs have a scalable number of multithreaded Streaming Multiprocessors (SM), which are provided of 8 streaming processor (SP), 16 kB of shared memory, two Special Function Units (SFUs) for single-precision transcendental function and a warp scheduler with a SIMT unit to execute one or more CUDA thread blocks. Each SM is hardware multithreaded and can execute up to 768 concurrent threads with zero scheduling overhead. The SM's SIMT multithreaded instruction unit creates, manages, schedules, and executes threads in warps of 32 parallel threads. Tesla devices have compute capabilities 1.x.

Each SM has its own register space with thousands of 32-bit registers. However, register pressure occurs when there are not enough registers available for a block of threads. For this reason, the amount of registers that a kernel uses must be taken into account, to run as many threads as possible within a SM.

Fermi [16] represents an evolution of Tesla architecture and counts with several innovations [21]. In Fermi, the SMs have 32 streaming processors, 4 SFUs, 64 kB of shared memory, dual warp scheduler and incorporate a configurable first-level cache. In addition, a 768 kB second-level cache was added to hide latency of DRAM in applications that have not enough data parallelism. Fermi GPUs have compute capabilities 2.x, with a new instruction set that includes atomic operations for floating points that were not available in devices with lower compute capabilities. Furthermore, floating-point units were updated from IEEE 754-1985 to IEEE 754-2008.

2. MATERIALS AND METHODS

Fully 3D MLEM reconstruction algorithm using Siddon as projector was implemented in CPU and GPU. In CPU a single thread implementation has been developed and compiled with optimization flags. Its performance was evaluated in different processors: AMD Phenom 955 BE and Intel i7 860. Different versions of the algorithm, with different levels of optimization for Tesla and Fermi Architecture, were programmed with CUDA for GPU. They were executed in GPUs GTX285 and GTX480, two High-end Nvidia Boards of Tesla T10 Series and Fermi GF100 Series, respectively. We selected those CPUs and GPUs because they were the high end processors in commodity hardware at the time this work was being carried out.

The reconstruction library works with projections stored as a Michelogram [22]. All input data sets were pre-corrected for attenuation, normalization and arc corrections. For this reason, the MLEM implementation presented uses an exclusive geometrical projector.

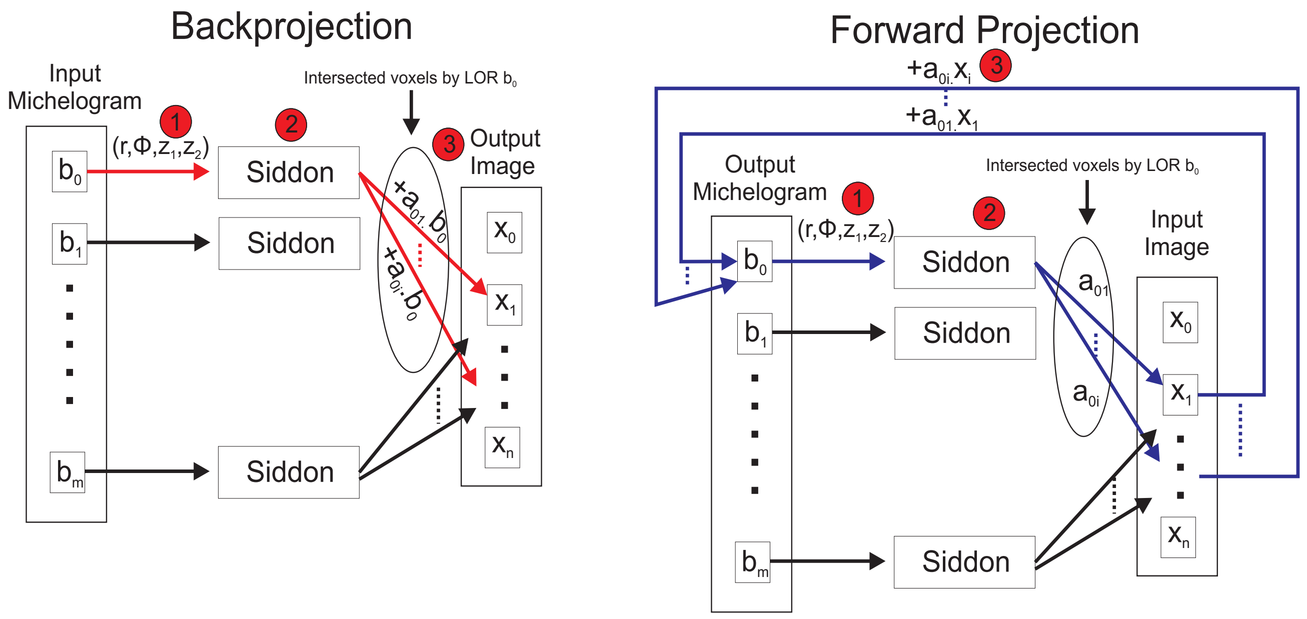

Fig. (2). Projection and backprojection implementation schemes.

2.1. Input Datasets

Three Michelograms of a GE Discovery STE scanner were used. Two of them are real data michelograms of a brain study and a measurement of cylinder phantom. The remaining was simulated with STIR library [24] using an approximated Nema Image Quality Phantom [23]. They are arranged in 553 direct and oblique two dimensional sinograms, each of them with 280 bins for Φ angle and 329 bins for *r* distance. The Field of View (FOV) of the scanner has a radius of 350 mm and an axial length of 156.96 mm.

2.2. GPUsç

The GTX-285 is a Tesla T10 board with compute capabilities 1.3, 240 CUDA cores with 1476 MHz of processor clock and 1 GB of DDR3 RAM. On the other hand, GTX-480 is a Fermi board with compute capabilities 2.0, 480 cores with 1401 MHz of Processor Clock and 1.5 GB of DDR5 RAM Table **1**.

2.3. Algorithm Implementation

Our implementation of MLEM algorithm computes each coefficient of the System Matrix on-the-fly, because the amount of coefficients of the SMR is much larger than the memory available in both CPU and GPU. Forward projection and backprojection are bin-driven computed. This means that for each bin of the Michelogram, Siddon's coefficients are calculated for every intersected voxel of the image. In forward projection the contribution of each pixel is added to the michelogram's bin corresponding to that LOR, whereas in backprojection each bin contributes to the pixels being intersected by the LOR. (Fig. **2**)

In each iteration Siddon's algorihtm is computed for each LOR in both forward projection and backprojection operation, which sums a total of around 100 millons Siddon's computations per iteration. This makes this kind of algorithm suitable for massively parallel architectures like GPUs. Each Siddon computation takes different quantity of instructions, depending on the amount of pixels intersected by the processed LOR.

2.4. GPU Implementations

Parallelization of MLEM algorithm has been performed for each stage of EM iteration: forward projection, backprojection and normalization. One kernel was developed for each of them. The two first operations were parallelized splitting them by bin. Each thread computes Siddon's coefficients for each bin of the Michelogram. The Normalization kernel is parallelized pixel-wise, so each thread works with a single pixel of x_j^k, x_est_j, S_j and x_j^{k+1} images.

The code was optimized taking into account the CUDA best practice guide [26]. In addition, NVIDIA CUDA Visual Profiler was used to evaluate the performance and find bottlenecks in GPU execution.

Forward projection and backprojection kernels are similar, the difference lies in the storage of the results. The forward projection adds, in a kernel's internal loop, all the Siddon's coefficients multiplied by its corresponding pixel value to the same output bin of the projected Michelogram. Whereas the backprojection adds each Siddon's coefficients, multiplied by the michelogram's bin value, to the intersected pixels. Since different threads can write on the same pixel memory address, race condition (race hazard) problems can occur in this last case (Fig. **3**). The probability of race condition events in the sum and store operation depends on the order that each michelogram's bin is processed and the pixels being intersected by each LOR.

Atomic sum is used in Fermi architecture, in order to avoid race conditions problems. This kind of operation is computationally expensive, as it is shown in the results.

2.4.1. Performance Optimization

A general implementation that included most of the optimizations recommendations [26] and executable on any CUDA capable GPU has been carried out. Projection kernels compute the operation for each bin. Using ThreadID and

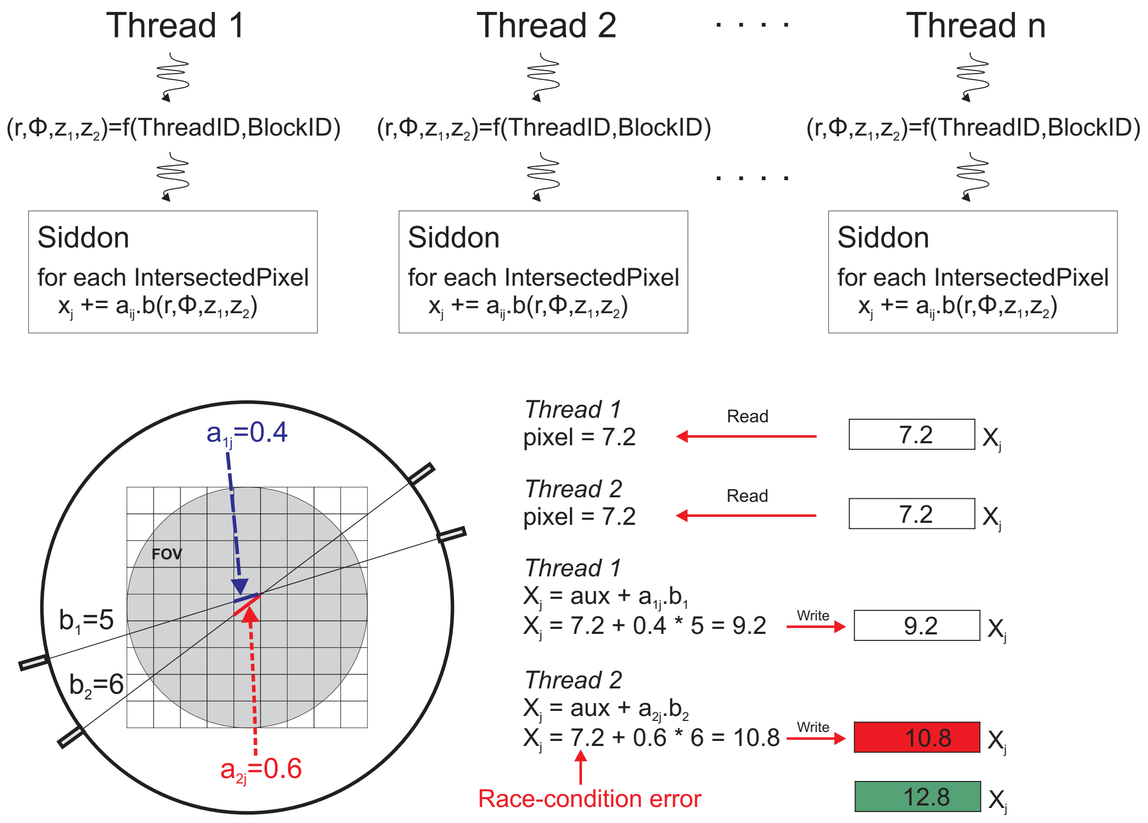

Fig. (3). Race condition problem in backprojection kernel. N threads running in parallel can have a race condition problem when a pixel value is updated simultaneously (top). A numerical example of error inserted by this problem is shown on bottom. Red highlighted value is the computed result; the right value is in green.

BlockID CUDA's structures, each thread gets the bin index of the michelogram represented by (i_r, i_Φ, $ring_1$, $ring_2$). These indexes are converted to real (r,Φ,z_1,z_2) values using a table in Constant Memory, and finally (x,y,z) coordinates of the two points on the detector cylinder that define the LOR are calculated. Both points determine the line equation of the LOR and Siddon algorithm can be executed, which is a straightforward implementation of CPU Siddon function.

Shared memory permits low latency access but its size is very limited. For this reason, image data could not be placed into this memory. Instead, michelogram data was stored in shared memory inside the kernels, since each thread accesses just one bin. All the parameters needed in the kernels were stored in Constant Memory, as it is cacheable (e.g., r and Φ values to convert Sinogram's bins indexes in respective geometrical values).

Each kernel is launched with a predefined size of threads per block. This parameter is important to have the GPU fully occupied, and is usually limited by the amount of registers used in a kernel, that for Projection kernels was 25 registers per thread. The number of threads per block was optimized for each of the GPUs, taking into consideration that threads per block should be multiples of 32, in order to have fully populated warps and facilitate coalescing [26]. Cuda Occupancy Calculator tool allows finding out the theoretical size of threads per block that used most of the GPU resources, which was 192 for GTX285 and 320 for GTX480. A full occupied GPU doesn't necessary translate to a better performance [26], so this parameters were verified running kernels with different size of threads per block.

Linear addressing was employed in each thread for bin indexing (Fig. **4**). The use of linear indexes in the same order than bins stored in global memory makes that each warp reads bins in a coalesced access. The order in which the bins are accessed was also chosen to avoid race condition problems. The strategy selected makes that threads within a block process mostly parallel LORs, which would reduce the amount of pixels simultaneously intersected by different threads of a block. This is depicted in Fig. (**4**).

2.4.2. Atomic Sum in Fermi

We implemented a modified version of the algorithm for GPU that uses atomic sum in backprojection, avoiding the potential race condition problem in this operation. This version is only executable in Fermi GPUs with CUDA Compute Capabilities 2.0, as the GTX 480, since atomic sum for floating point is required.

CUDA implementations were compiled for compute capabilities 1.3 using the "arch" nvcc compiler flag option [25]. Modified version with atomic operations were compiled for 2.0 compute capability, since atomic sum for floating point is only available for that feature. To get information about the resource usage, the "ptxas" flag option was added.

2.5. Image Quality Evaluation

The image quality should be quantitatively evaluated. Visual inspection can be useful when comparing two or more images, but it is clearly insufficient for a good comparison when the differences between them are not

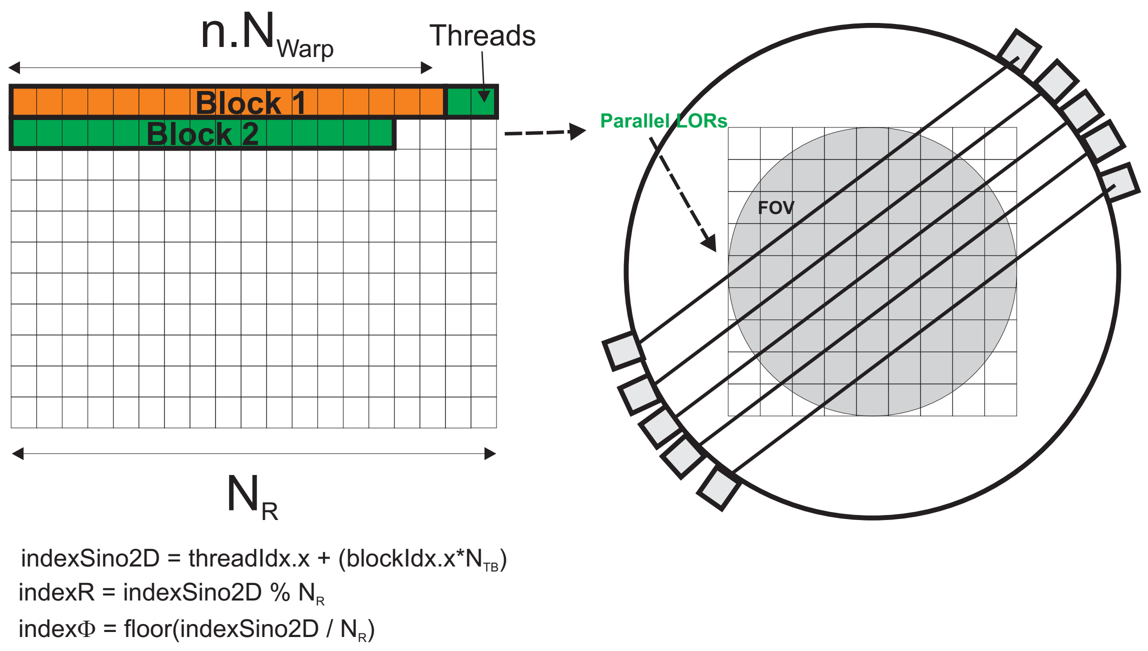
n.N_Warp
Threads
Block 1
Block 2
N_R
Parallel LORs
FOV
indexSino2D = threadIdx.x + (blockIdx.x*N_TB)
indexR = indexSino2D % N_R
indexΦ = floor(indexSino2D / N_R)

Fig. (4). Parallelization strategy regarding threads per block inside each 2D sinogram. A linear index for the whole sinogram is used and threads per block are a multiple of Warp's size (left). Most of the threads of a block belong to a same row of the sinogram, which correspond to parallel LORs.

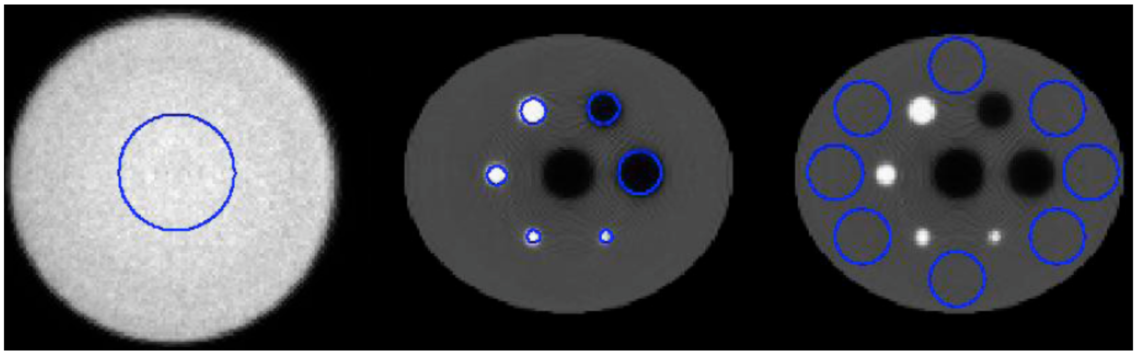

Fig. (5). Regions of Interest used to compute noise to signal ratio in uniform areas and recovery contrast in hot and cold spheres. In cylinder phantom a central circle was utilized (left). In Simulated IQ Nema phantom, a ROI for each hot and cold sphere was used to evaluate recovery contrast (middle) and eight circular ROIs were used to analyze the background (right).

easily recognized. Thus, the standard deviation in an expected uniform Region of Interest (ROI) was used for quantifying the noise of each GPU version. A circle of 25 mm radius was selected as ROI in the central slice of the cylinder image. In Simulated IQ Nema, eight circular ROIs of 15 mm radius were used in the uniform background of the phantom, and then the standard deviation inside them was calculated. (Fig. **5**). The noise to signal ratio (NSR) was obtained by calculating the rate between standard deviation and mean values in the background ROIs.

Moreover, we obtained recovery contrast for hot and cold spheres in reconstructed images of simulated IQ Nema. We drew a circular ROI with a diameter equal to 80% of the sphere radius for each of them. Hot spheres in the phantom had an activity concentration of four times the background, so that recovery contrast is defined as:

$$Q_{H,j} = \frac{\frac{C_{H,j}}{C_{B,j}} - 1}{4-1}.100\% \qquad (6)$$

Where $Q_{H,j}$ is the recovery contrast for hot spheres, $C_{H,j}$ is the average counts in the ROI for sphere j and C_{BJ} is the average counts in the background ROIs. Recovery contrast for cold spheres is calculated by:

$$Q_{C,j} = (1 - \frac{C_{H,j}}{C_{B,j}}).100\% \qquad (7)$$

3. RESULTS

MLEM algorithm was executed in each platform for each of the input michelograms. Images of 128x128x47 and 256x256x47 pixels were reconstructed, with 40 iterations and using full scanner FOV. In addition, images of 256x256x47 pixels with a FOV of 20 cm radius were reconstructed. The latter were used for evaluating the image quality parameters mentioned above.

Total reconstruction, iteration and operation times were recorded. An average time per iteration and operation was calculated to compare and compute the performance of each processor and implementation. A comparison of iteration

Table 1. Comparison of the GPUs used in this Work

	GTX-285	GTX-480
CUDA Cores	240	480
Processor Clock	1476 MHz	1401 MHz
Memory Interface	GDDR3	GDDR5
Memory Bandwidth (GB/sec)	159 GB/sec	177.4 GB/sec
Total Memory	1 GB	1.5 GB
Compute Capabilities	1.3	2.0

Table 2. Iteration Time for each GPU and CPU Implementation

	AMD Phenom 955BE	Intel i7 860	GTX285	GTX480	GTX480 with Atomics
128x128 FullFOV	254.6 sec	238.2 sec	6.2 sec	2.8 sec	13.6 sec
256x256 Full FOV	435.1 sec	395.5 sec	21.7 sec	8.5 sec	16.7 sec
256x256 20 cm radius FOV	249.7 sec	309.3 sec	12.4 sec	6.9 sec	23.3 sec

Table 3. Speed up Factors for each GPU Implementation and Hardware

	AMD Phenom 955BE	Intel i7 860	GTX285 V1	GTX480 V1	GTX480 V3
128x128x47 FullFOV	254.6	238.2	38.4	85.1	17.5
256x256x47 Full FOV	435.1	395.5	18.2	46.5	23.7
256x256x47 20 mm radius FOV	249.7	309.3	25.0	44.8	13.3

Table 4. Processing Times by Stage, in one Iteration. Results for Reconstructed Image of Simulated IQ Nema Phantom of 256x256x47 Pixels and Full FOV

	AMD Phenom 955BE	Intel i7 860	GTX285	GTX485	GTX485 with Atomics
Forward projection	187 sec	211 sec	5.1 sec	3.8 sec	6.1 sec
Backprojection	248 sec	183 sec	15.5 sec	4.0 sec	17.2 sec
Normalization	0.01 sec	0.04 sec	0.0005 sec	0.0005 sec	0.0005 sec

times for every reconstructed image can be seen in Table **2**. Speed up factors were compared with the CPU Intel i7 860, which had the best performance between the processors evaluated. They are shown in Table **3**. Speed up factors of 38 and 85 were respectively achieved in GTX285 and GTX480 for images of 128x128x47 pixels, but with higher noise, due to race condition problems. When atomic sum was used, speed up factors were reduced to 20, in average. For images of 256x256x47, GPU performance was not as good as it was for smaller images.

Execution times were also recorded by stage; they are indicated in Table **4** for the case of Simulated IQ Nema phantom and reconstructed images of 256x256x47 pixels with Full FOV. The distribution of time per operation can be observed in Fig. (**6**); GPU backprojection operation is the more demanding operation, while in CPU there are no relevant differences between both operations. Normalization takes insignificant time compared with the other two stages.

A set of reconstructed images can be seen in Fig. (**7**) and Fig. (**8**), where central slices of reconstructed cylinder phantom and IQ Nema Phantom for each platform are shown. By visual inspection, different noise levels can be distinguished. In order to quantify the noise level, an evaluation of the standard deviation of pixel intensity in uniform ROIs has been made in Simulated IQ Phantom and Uniform Cylinder images, as described before Table **5**. The noise to signal ratio is higher in GTX285 than in GTX480, and both are higher than in CPU. When using atomic add in Fermi the NSR is equivalent to the CPU version. The additional noise present when atomic operations are not used is, in fact, a consequence of race conditions problems. In Fig. (**3**) we showed that, in backprojection operation, pixel values are underestimated when a race condition occurs. This was verified measuring the mean value of the cylinder, that it was 2% lower when atomic operations were not used.

Recovery contrast was evaluated for all the spheres of IQ Nema Phantom. Its values are presented in Table **6**. In this case, there was not significant difference between the different versions. This was an expected result since race conditions affects regions with cold and hot spheres, as well as the background region. In GTX285, recovery contrast values are slightly higher because the background mean value is lower as a consequence of the many events with race conditions that take place during backprojection operation.

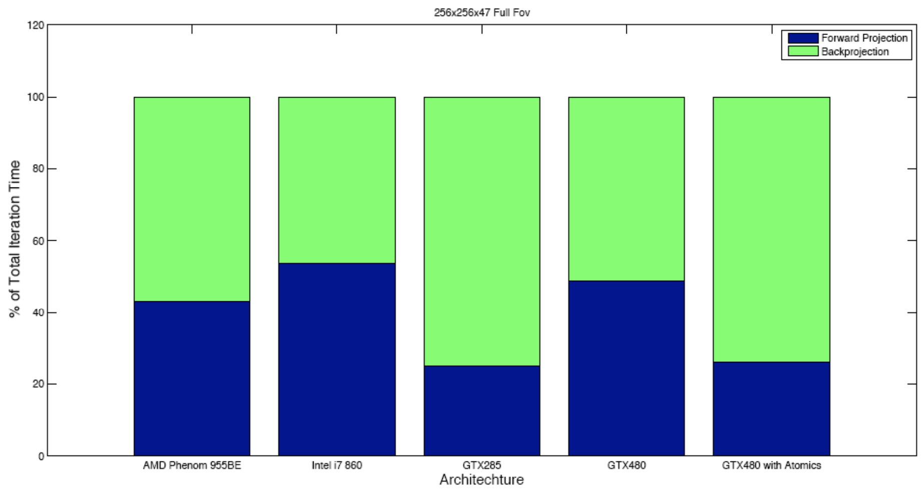

Fig. (6). Distribution of processing times per stage for 256x256x47 and full FOV reconstructed image.

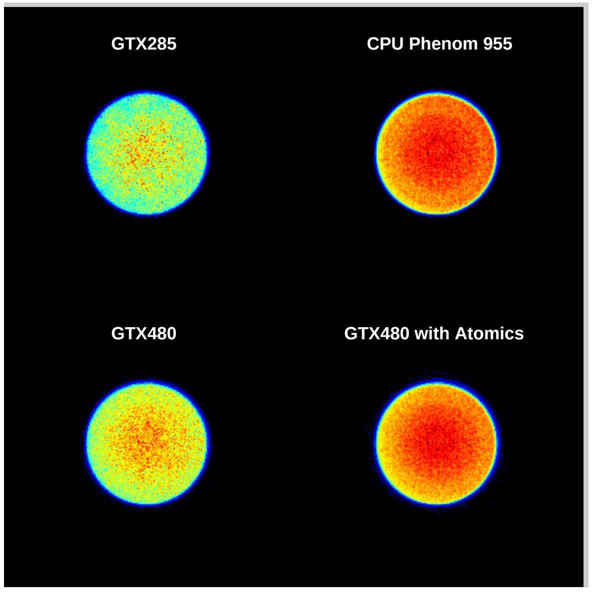

Fig. (7). Central slices of cylinder phantom reconstructed images for all platforms, each having 256x256 pixels.

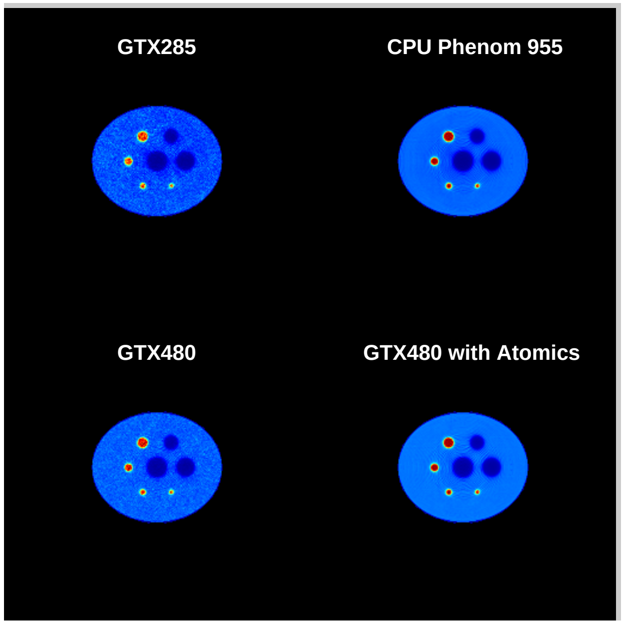

Fig. (8). Central slices of Simulated IQ Nema reconstructed images for all platforms, each having 256x256 pixels.

Table 5. Noise to Singal Ratio in Uniform ROIs of Phantoms

	GTX285	GTX480	GTX480 with Atomics	CPU
Cylinder	16.8%	10.6%	4.9%	4.9%
Simulated IQ Nema 256	14.9%	9.3%	1.8%	1.2%

Table 6. Recovery Contrast Values for each Sphere of Reconstructed iq Nema Phantom with Size of 256x256x47 Pixels and 20 cm Radius Fov

	GTX285	GTX480	GTX480 with Atomics	Intel i7 860
10 mm	46.3%	46.0%	44.7%	44.6%
13 mm	69.1%	67.4%	68.6%	68.5%
17 mm	76.5%	74.7%	74.9%	74.9%
22 mm	83.0%	80.7%	80.4%	80.2%
28 mm	66.9%	64.7%	64.4%	64.1%
37 mm	73.3%	70.9%	71.1%	71.0%

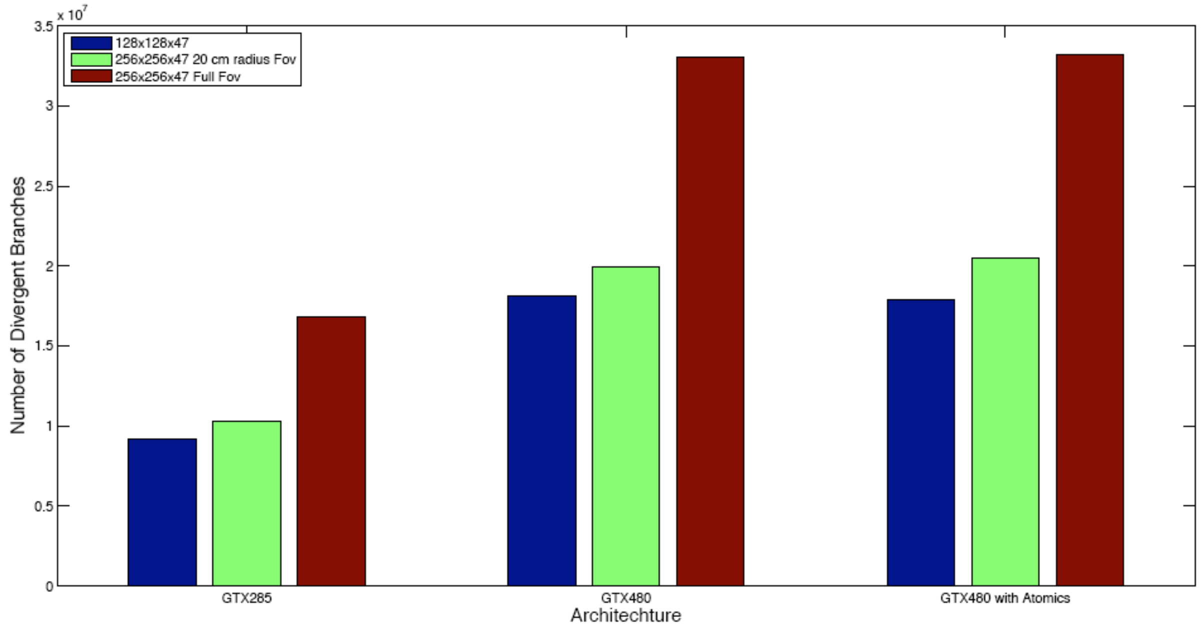

Fig. (9). Number of divergent branches in thread execution in Forward Projection operation.

4. DISCUSSION

Different speed up factors were achieved for each GPU and image size. Such as it was expected, larger images presented lower acceleration rates; in a bin-wise projector, the amount of issued threads is independent of the image size, thus a larger image means more instructions per thread instead of the execution of more threads. The latter would be advantageous in a massive parallel architecture as GPUs. Moreover, threads that execute Siddon's algorithm in larger paths increase the chance of branch divergence within a warp. This can be observed in Fig. (**9**).

A maximum speed up factor of 85 times was achieved for GTX480 without atomic operations, but with images with higher noise to signal ratio. GTX480 doubled the performance of GTX285, this is mostly justified by the number of thread processors of each GPU and the introduction of the L1 cache in Fermi.

The images reconstructed with GPU, without atomic operations, showed worse NSR than CPU, due to the error introduced by race condition problems. On the other hand, recovery contrast values didn´t show any significant difference between both platforms. When using atomic add, read-modify-write process is executed without interference from other threads, thus ensuring that no other thread can access that memory address until the write operation has finished. As a consequence, noise levels are equivalent to CPU images at a cost of a reduction of speed up factors between 3 and 4 times.

As was stated above, GPU backprojection takes most of the iteration time. On the other hand, in CPU it takes approximately the same time than forward projection. Both operations have the same computing load, but differ in memory access patterns. In GPU, the only difference between both kernels arises when writing the results. In forward projection the result in each thread is written in the

Table 7. Speed up Factors Per Stage for 256x256x47 and Full FOV Images

	GTX285	GTX480	GTX480 with Atomics
Forward projection	41.4	55.5	34.6
Backprojection	11.8	45.8	10.6

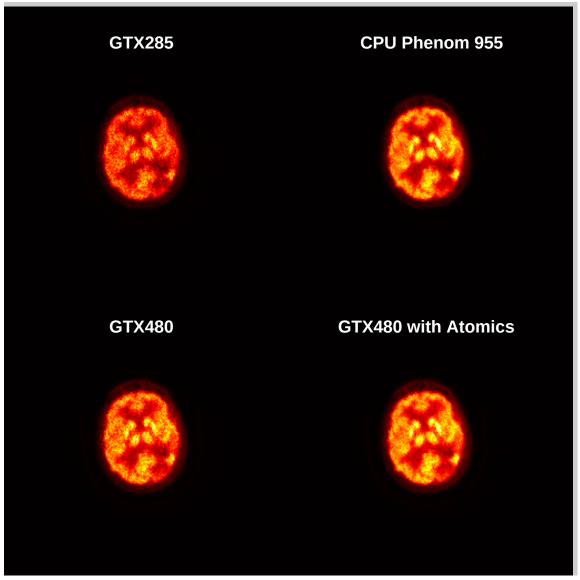

Fig. (10). Central slice of brain study reconstructed image, each having 256x256 pixels.

same memory address, accessing contiguous addresses inside a thread warp. In contrast, backprojection writes the different memory addresses of intersected pixels, so threads in a warp hardly access contiguous memory locations. Because of that, backprojection has much less coalesced memory accesses; therefore, serialization of memory access and instruction execution occurs inside a warp. In GPU image data is in global memory which access is expensive, especially when it is not coalesced, as in the case of backprojection. The presence of a L1 cache in GTX480 could reduce this problem, and is verified by the fact that in this graphic processor backprojection has an acceleration factor close to the one obtained in forward projection (Table **7**).

5. CONCLUSION

Acceleration rates between 15 and 85 have been achieved with a GPU implementation of MLEM image reconstruction algorithm, despite of the use of a projector that best adapts to CPU architecture. The problems existing in Tesla architecture to generate similar results to CPU, regarding image quality, were solved with the introduction of atomic operations for floating points. In Fermi, unlike Tesla, an acceptable noise level is achieved even when atomic operations are not employed, and speed up factors of 85 times are possible. With the use of atomic operations the execution times are slowed down up to 4 times, but noise levels equivalent to CPU are observed. (Fig. **10**)

Due to the scalability of CUDA model and the fact that graphics processors are permanently increasing their parallel computing power, higher acceleration rates are expected to be obtained with the launch of new GPUs in the market, without needing to rewrite the code. In addition, new features, such as cache memories and atomic operations introduced in Fermi, can improve the performance of algorithms both in speed and quality. Moreover, CUDA permits to run multi-GPU applications with low programming effort, and it would be expected to achieve a scaled performance when using a rack of GPU boards.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

This work was partly funded by UTN PhD fellowship to Martín Belzunce and by CNEA's AR-PET project. Authors wish to acknowledge the Digital Processing Laboratory (DPLab) of UTN-FRBA for providing the GTX-480 used in the experiments presented in this paper.

REFERENCES

[1] Shepp LA, Vardi Y. Maximum likelihood reconstruction for emission tomography. IEEE Trans Med Imaging 1982; 1:113-22.
[2] Reader JA, Visvikis A, Erlandsson K, *et al*. Intercomparison of four reconstruction techniques for positron volume imaging with rotating planar detectors. Phys Med Biol 1999; 43: 823-34.
[3] Chen C, Lee S. On Parallelizing the EM algorithm for PET image reconstruction. IEEE Trans Parallel Distrib Syst 1994; 5: 860-73.
[4] ones JP, Jones WF, Keheren F. SPMD cluster-based parallel 3-D OSEM. IEEE Trans Nucl Sci 2003; 50(5): 1498-502.
[5] Schellmann M, Voerding J, Gorlatch S, *et al*. Cost-effective medical image reconstruction: from clusters to graphics processing units. In: Proceedings of the 2008 conference on Computing frontiers; 2008: New York; pp. 283-92.
[6] Nickolls J, Dally WJ. The GPU computing era. IEEE Micro 2010; 30(2): 56-69.
[7] Owens JD, Houston M, Luebke D, *et al*. GPU Computing. Proc IEEE 2008; 96: 879-99.

[8] Bai B, Smith AM. Fast 3D iterative reconstruction of PET images using PC graphics hardware. In: IEEE Nuclear Science Symposium Conference Record; 2006 Oct 26: San Diego, USA; pp. 2787-90.

[9] Pratx G, Chinn G, Habte F, *et al.* Fully 3-D list-mode OSEM accelerated by graphics processing units. In: IEEE Nuclear Science Symposium Conference Record; 2006 Oct 26: San Diego, USA; pp. 2196-202.

[10] Pratx G, Chinn G, Olcott P, *et al*. Fast accurate and shift-varying line projections for Iterative reconstruction using the GPU. IEEE Trans Med Imaging 2009; 28: 435-45.

[11] Xu F, Mueller K. Real-time 3D computed tomographic reconstruction using commodity graphics hardware. Phys Med Biol 2007; 52: 3405-19.

[12] NVIDIA. CUDA Programming Guide 3.2. NVIDIA, 2010. Available from: http://developer.down-load.nvidia.com/compute/-cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf [cited 2012 Mar 1].

[13] Herraiz JL, España D, García D, *et al*. GPU acceleration of a fully 3D iterative reconstruction software for PET using CUDA. In: IEEE Nuclear Science Symposium Conference Record; 2009 Oct 24: Orlando, USA; pp. 4064-7.

[14] Zhou J and Qi J. Fast and efficient fully 3D PET image reconstruction using sparse system matrix factorization with GPU acceleration. Phys Med Biol 2011; 56: 6739-57.

[15] Lindholm E, Nickolls J, Oberman S, *et al.* NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro 2008; 28: 39-55.

[16] NVIDIA. Fermi: NVIDIA's Next Generation CUDA Compute Architecture. NVIDIA 2009. Available from: http://www.nvi-dia.com/content/PDF/fermi_white_papers/NVIDIAFermi_Comput e_Architecture_Whitepaper.pdf [cited 2012 Mar 1].

[17] Lange K, Carlson R. EM reconstruction algorithms for emission and transmission tomography. J Comput Assist Tomogr 1984; 8: 306-16.

[18] Hudson HM, Larkin RS. Accelerated image reconstruction using ordered subsets of projection data. IEEE Trans Med Imaging 1994; 13:601-9.

[19] Siddon RL. Fast calculation of the exact radiological path for a three-dimensional CT. J Med Phys 1985; 12: 252-5.

[20] Jacobs F, Sundermann E, De Sutter B, *et al.* A fast algorithm to calculate the exact radiological path through a pixel or voxel space. Comput Inf Tech 1998; 6: 89-94.

[21] Patterson D. The top 10 innovations in the new NVIDIA Fermi architecture, and the top 3 next challenges. Parallel Computing Research Laboratory: U.C. Berkeley; 2009. Available from: http://www.nvidia.com/content/PDF/fermi_-white_papers/D.Patter-son_Top10InnovationsInNVIDIAFermi.pdf [cited 2012 Mar 1].

[22] Bendriem B, Townsend DW. The theory and practice of 3D PET. 1st ed. USA: Springer 1998.

[23] NEMA. Performance Measurements of Postiron Emisión Tomoraphs. Nat. Elect. Manufact. Assoc., Washington, DC; NEMA Standards Pub., NU2-2001; 2001.

[24] Thielemans K, Mustafovic S, Tsoumpas Ch. STIR: Software for Tomographic Image Reconstruction Release 2. IEEE Nuclear Science Symposium Conference Record; 2006 Oct 26: San Diego, USA; pp. 2174-6.

[25] NVIDIA. The Cuda Compiler Driver NVCC. NVIDIA Corporation 2010.

[26] NVIDIA. Cuda C Best Practices Guide. Version 3.2, NVIDIA Corporation; 2010. Available from: http://developer.download.nvi-dia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Best_Prac tices_Guide.pdf [cited 2012 Mar 1].

Received: April 12, 2012 Revised: August 09, 2012 Accepted: August 25, 2012

© Belzunce *et al*.; Licensee *Bentham Open*.

This is an open access article licensed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/), which permits unrestricted, non-commercial use, distribution and reproduction in any medium, provided the work is properly cited.