# Fully 3D GPU PET reconstruction

J.L. Herraiz [a,*], S. España [b], J. Cal-González [a], J.J. Vaquero [c], M. Desco [c,d], J.M. Udías [a]

[a] *Grupo de Física Nuclear, Departmento Física Atómica, Molecular y Nuclear, Universidad Complutense de Madrid, Spain*
[b] *Department of Radiation Oncology, Massachusetts General Hospital and Harvard Medical School, Boston, MA, USA*
[c] *Departmento de Bioingeniería e Ingeniería Espacial, Universidad Carlos III, Madrid, Spain*
[d] *Unidad de Medicina y Cirugía Experimental, Hospital General Universitario Gregorio Marañón, Madrid, Spain*

A R T I C L E  I N F O

Available online 16 December 2010

*Keywords:*
Tomographic reconstruction
Positron emission tomography
Graphics processing unit

A B S T R A C T

Fully 3D iterative tomographic image reconstruction is computationally very demanding. Graphics Processing Unit (GPU) has been proposed for many years as potential accelerators in complex scientific problems, but it has not been used until the recent advances in the programmability of GPUs that the best available reconstruction codes have started to be implemented to be run on GPUs.

This work presents a GPU based fully 3D PET iterative reconstruction software. This new code may reconstruct sinogram data from several commercially available PET scanners. The most important and time consuming parts of the code, the forward and backward projection operations, are based on an accurate model of the scanner obtained with the Monte Carlo code PeneloPET and they have been massively parallelized on the GPU. For the PET scanners considered, the GPU based code is more than 70 times faster than a similar code running on a single core of a fast CPU, obtaining in both cases the same images. The code has been designed to be easily adapted to reconstruct sinograms from any other PET scanner, including scanner prototypes.

## 1. Introduction

Graphics Processing Unit (GPU) has been proposed for many years as potential accelerators in complex scientific problems [1] like image reconstruction, with large amount of data and high arithmetic intensity. Indeed, tomographic reconstruction codes are suitable for massive parallelization, as the two main time consuming parts of the code (forward and backward projection) can be organized as single instruction multiple data (SIMD) tasks and distributed among the available processor units by assigning part of the data to each unit [2,3].

In a previous work, we developed the tomographic reconstruction code fast iterative reconstruction software for (PET) tomography (FIRST) [4] using the message passing interface (MPI) protocol [5] to launch parallel tasks and communicate results between a master and several slave processes, which run on the available CPUs (or CPU cores) in a cluster of computers. FIRST has proved to be a successful implementation of a tomographic code for high resolution small animal PET scanners [4,6].

However, programming a code like FIRST to take full advantage of GPU features was not an easy task. Good knowledge of targeted GPU architecture was required, and it was necessary to translate the operations in the algorithm into graphics related terms like vertex and fragment shaders [7], making it difficult to create complex codes for the GPU. Therefore, it has not been until the recent advances in the programmability of GPUs [8] that the best available reconstruction codes like FIRST have started to be implemented to be run on GPUs [9,10].

In this work, we have implemented a fully 3D PET iterative reconstruction software using CUDA [8] that makes use of the efficient computing capabilities of GPUs. The main goal was to obtain a significant acceleration of the code (similar to the ones obtained using a cluster of CPUs) without compromising with the quality of the reconstructed images. Therefore, we avoided approximations in the forward and backward projection kernels, and we used the same model, called the system response matrix (SRM), as it would be used in a CPU code. Furthermore, numerical approximations such as integer conversion of float image values were also avoided.

## 2. Materials and methods

The reconstruction code implemented in this work is based on the 3D OSEM algorithm [11]. In this statistical method, the reconstructed image is iteratively updated pursuing that the estimated projections obtained from that image were the most compatible ones with the acquired data that contains Poisson noise.

The code was implemented in CUDA [8], an application programming interface (API), which allows writing programs in C language with extensions to execute part of them (CUDA kernels) on the GPU. These kernels may execute large number of threads in parallel in

* Corresponding author. Tel./fax: +34 913 944 484.
*E-mail address:* joaquin@nuclear.fis.ucm.es.(J.L. Herraiz)

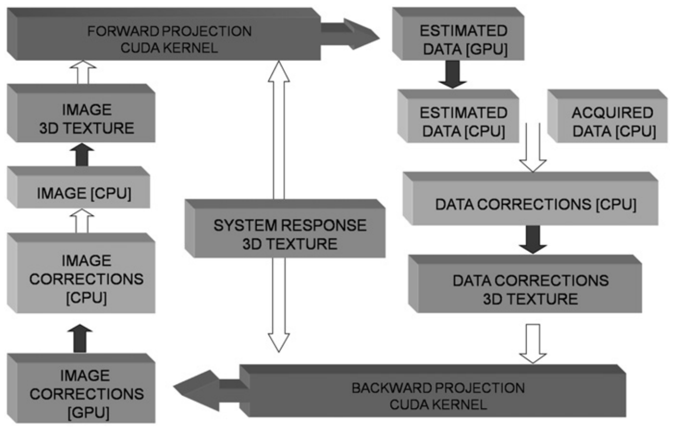

**Fig. 1.** Flowchart of the fully 3D iterative reconstruction code implemented using the GPU.

SIMD mode. In this work, only two CUDA kernels were required: the forward and the backward projection, as shown in Fig. 1.

Due to the large number of threads that can be executed in parallel on GPUs, the usual bottlenecks of these implementations are memory access. In our study, we used texture memory, which is a kind of global memory available in the GPU that is allocated and indexed for fast access [12]. In our code we defined three 3D textures, as shown in Fig. 1: one for the reconstructed image, another for the corrections applied in each iteration, and finally one to store the SRM. In this work, the time required for the data transfer between the CPU and the GPU memory in each iteration was negligible compared to the time spent in the forward and backward projection.

In this work, we show results from the high resolution small animal PET scanner VrPET [6], which is composed of two pairs of rotating plane detectors in coincidence with a transaxial field of view (FOV) of 86.6 mm and axial FOV of 45.6 mm. The sinograms are organized in 117 radial and 190 angular bins for each of the direct and oblique crystal combinations, making a total of 117 (radial) $\times$ 190 (angular) $\times$ 30 $\times$ 30 (axial) bins. The recon structed images were composed of 117 $\times$ 117 $\times$ 59 voxels.

### 2.1. System response matrix

Our fully 3D iterative reconstruction code is based on a realistic model of the emission and detection of the radiation in the PET scanner. This model was generated using the Monte Carlo code developed in our group PeneloPET [13], which is based on PENELOPE [14]. PeneloPET simulates the most relevant physical effects in a PET acquisition (positron range, non collinearity, interaction of the gamma rays with the scintillator crystals and electronics). Due to these effects, each line of response (LOR) connecting a pair of detector elements is related to a wide region of the field of view, commonly known as the tube of response (TOR). The TOR for a LOR ($i$) is composed of all voxels ($j$) with non zero coefficient $C_{ij}$, which represents the probability that a positron emitted in voxel $j$ is detected in LOR $i$.

In this work, each TOR is made of an array of 117 (longitudinal) $\times$ 7 (transaxial) $\times$ 7 (axial) coefficients. The coefficients from all the TORs in the scanner form the SRM. Due to the large size of the SRM, all the symmetries present in the system should be exploited in order to fit it into the texture memory of the GPU. This implies that only some TORs need to be computed and stored, because symmetrically equivalent LORs, as shown in Fig. 2, have the same probability distribution. When dealing with sinograms acquired using a con tinuous rotating scanner such as VrPET, the rotational symmetry yields an SRM that does not depend on the in plane angle, reducing the size of the SRM considerably.

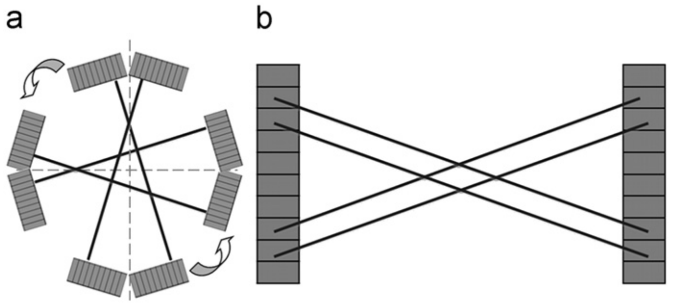

**Fig. 2.** Symmetries used to reduce the number of LORs that need to be stored in memory: (a) in-plane rotations and reflections, (b) translations and reflections in the axial direction. Coefficients for TORs in (a) and (b) have identical values.

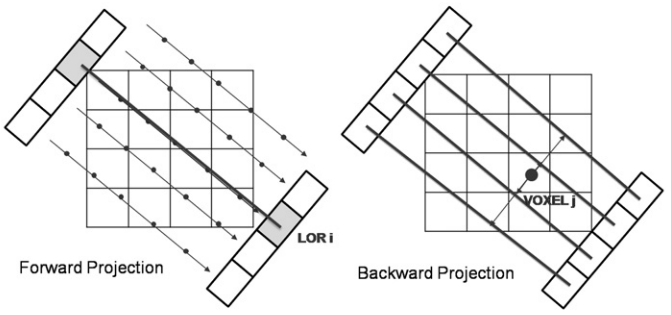

**Fig. 3.** Schematics description of the forward and backward projections. Note that in the code the sampled points represent a three-dimensional grid.

### 2.2. Forward projection kernel

In the forward projection kernel, each thread of the GPU projects one LOR, adding the contribution from all the points connected to it, as shown in Fig. 3. This way, a large amount of LORs can be projected simultaneously. Each sampled point corresponds to a coefficient of the SRM. On the other hand, the values of the image at each of these points are obtained by tri linear interpolation, which is easily accessible in 3D textures [12].

### 2.3. Backward projection kernel

In the backward projection kernel, each thread of the GPU back projects one voxel and computes the corrections from all the LORs connected with that voxel that was projected previously (see Fig. 3). In this case it is necessary to compute the distance of the voxel to the center of each LOR to obtain the SRM coefficient by tri linear interpolation within the 3D texture. It is important to note that this can be easily done because in this work we are dealing with sinograms that are spatially sorted data. With other data formats, like list mode acquisitions, it would be difficult to find which LORs are connected with a specific voxel.

### 2.4. Equipment

In this work we compare the performance of the reconstruction code using a 4 core front end computer (Intel® Core™ i7 (2.93 GHz)) with two different GPUs. The reconstruction time of the CPU version run in a single core was taken as a reference. For comparison purposes, the CPU code was also parallelized using the MPI protocol [5] and executed using 8 threads on the same 4 core multithread able CPU. On the other hand, we used a low cost GPU (8600 GT) with 4 stream multiprocessors (SM) and a more powerful one (Tesla C1060) with 27 SM, to run the GPU version of the code. The maximum number of threads that can be executed in parallel on the GPU is proportional to

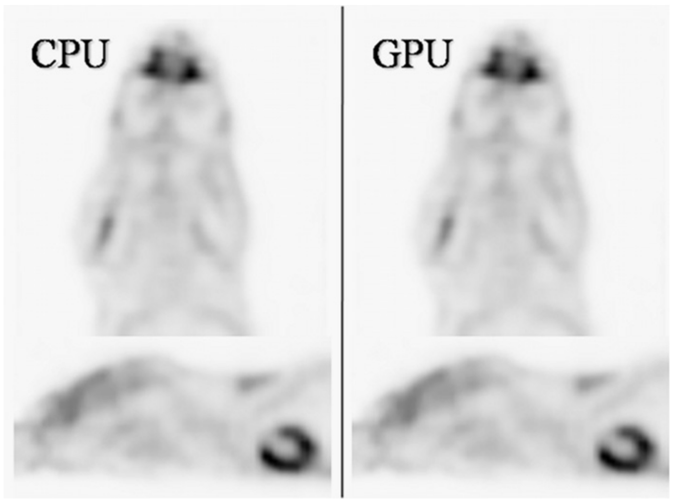

**Fig. 4.** Image reconstructed from a real acquisition both in CPU and GPU. Coronal view (top) and sagittal view (bottom) of the reconstructed image of a 200 g rat FDG acquisition.

**Table 1**
Reconstruction times for one image (one-bed, one-frame acquisition, one full iteration of 50 subsets) in different architectures. The speed-up factors are computed against the reference CPU using a single thread.

| | Reconstruction time (s) | Speed-up factor |
|---|---|---|
| CPU–Intel® Core$^{TM}$ i7–2.93 GHz (1 Core) | 3456 | 1 × |
| CPU–Intel® Core$^{TM}$ i7–2.93 GHz (4 Cores – Multithread) | 623 | 5.5 × |
| GPU – 8600 GT–256 MB – 4 SM | 509 | 7 × |
| GPU – TESLA C1060–4 GB – 27SM | 49 | 72 × |

the number of SM, so a significant difference in the reconstruction time is expected between these GPUs.

## 3. Results

### 3.1. Image quality

Fig. 4 shows a transverse and coronal view of the images reconstructed of the 200 g rat injected with FDG using both CPU and GPU codes. The differences between both images are visually negligible, with a mean square difference smaller than 0.1%.

### 3.2. Reconstruction time

Table 1 shows the time required for the reconstruction of one image from one bed, one frame acquisition using 1 full iteration of 50 subsets for different architectures. The reference time correspond to the one obtained with a fast CPU using a single core. For comparison purposes, the CPU code was also parallelized using the MPI protocol [5] and executed using 8 threads on a multithread capable 4 core front end computer. The speed up factor of 5.5 is smaller than the one obtained with the worse GPU employed here (8600 GT). It is notice able that in our best available GPU, the reconstruction time was reduced by a factor 72 compared to the CPU code.

## 4. Conclusions

We have implemented a GPU based fully 3D PET iterative reconstruction software. This new code reconstructs sinogram data from simulated and commercially available PET scanners and it is up to more than 70 times faster than a similar code running as a single thread on a single core of a fast CPU, obtaining in both cases identical images. It is remarkable that a single Tesla C1060 GPU card would be comparable to 18 quad core CPU high performance workstations for these reconstructions. In our cheapest GPU card, the code was even faster than an expensive modern computer with quad core CPUs.

The code has been implemented using CUDA and it is easily adaptable to reconstruct sinograms from any other PET scanner; so it may also be used for fast and accurate reconstruction of acquisitions from scanner prototypes. Further improvement of the code using the MPI protocol to run it on several GPUs is currently under development.

## Acknowledgments

The authors would like to thank Dr. R. Cabido and Dr. A.S. Montemayor for their advice in some GPU related technical issues. This work has been supported by MEC (FPA2007 62216), UCM (Grupos UCM, 910059), CPAN (Consolider Ingenio 2010) CSPD 2007 00042, the RECAVA RETIC network, ARTEMIS S2009/ DPI 1802, European Regional Development, ENTEPRASE grant, PSE 300000 2009 5 and TEC2008 0675 C02 01, Ministerio de Ciencia e Innovación, Spanish Government.

## References

[1] General-Purpose Computing on Graphics Processing Units repository, 〈www.gpgpu.org〉, September 2010.
[2] M.D. Jones, R. Yao, IEEE NSS/MIC Conference Record, 2004, p. 3036.
[3] I.K. Hong, et al., IEEE Trans. Med. Imag. 26 (2007) 789.
[4] J.L. Herraiz, et al., Phys. Med. Biol. 51 (2006) 4547.
[5] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, 1999.
[6] E. Lage, et al., Phys. Med. Biol. 54 (2009) 5427.
[7] F. Xu, F.K. Mueller, Phys. Med. Bol. 51 (2007) 3405.
[8] NVIDIA CUDA Programming Guide v.2.5.0, 〈http://developer.nvidia.com/object/gpu_programming_guide.html〉, September 2010.
[9] J.L. Herraiz et al., IEEE NSS/MIC Conference Record, 2009, p. 4064.
[10] X. Jia, Y.F. Lou, R.J. Li, Med. Phys. 37 (2010) 1757.
[11] H.M. Hudson, R.S. Larkin, IEEE Trans. Med. Imag. 13 (1994) 601.
[12] J. Sanders, E. Kandrot, Cuda by Example, Addison-Wesley, 2010.
[13] E. España, et al., Phys. Med. Biol. 54 (2009) 1723.
[14] J Baró, et al., Nucl. Instr. and Meth. Phys. Res. B100 (1995) 31.