GPU-Based Fast Iterative Reconstruction of Fully 3-D PET Sinograms

J. L. Herraiz, S. España, R. Cabido, A. S. Montemayor, M. Desco, J. J. Vaquero, and J. M. Udias

***Abstract*—This work presents a graphics processing unit (GPU)-based implementation of a fully 3-D PET iterative reconstruction code, FIRST (Fast Iterative Reconstruction Software for [PET] Tomography), which was developed by our group. We describe the main steps followed to convert the FIRST code (which can run on several CPUs using the message passing interface [MPI] protocol) into a code where the main time-consuming parts of the reconstruction process (forward and backward projection) are massively parallelized on a GPU. Our objective was to obtain significant acceleration of the reconstruction without compromising the image quality or the flexibility of the CPU implementation. Therefore, we implemented a GPU version using an abstraction layer for the GPU, namely, *CUDA C*. The code reconstructs images from sinogram data, and with the same System Response Matrix obtained from Monte Carlo simulations than the CPU version. The use of memory was optimized to ensure good performance in the GPU. The code was adapted for the VrPET small-animal PET scanner. The CUDA version is more than 70 times faster than the original code running in a single core of a high-end CPU, with no loss of accuracy.**

***Index Terms*—CUDA, graphics processing units, image reconstruction, positron emission tomography.**

I. Introduction

TOMOGRAPHIC reconstruction is computationally very demanding, especially when iterative methods based on realistic models for the emission and detection of radiation are used [1]. The increasing complexity of scanners implies a large number of data and reconstructed voxels [2], as well as more sophisticated acquisition protocols, such as dynamic studies [3], which require new approaches to reconstruct images in reasonable time. Tomographic image reconstruction codes are suitable for massive parallelization, as their two main time-consuming parts (forward and backward projection) can be organized as single instruction multiple data (SIMD) tasks and distributed among the available processor units [4], [5]. This allows for reducing the elapsed reconstruction time by means of large clusters of computers and multi-core processors

On the other hand, graphics processing units (GPUs) can be used to alleviate the intense computational demands posed by many scientific problems [6], including tomographic image reconstruction. The higher the ratio between data computation and data communication (or arithmetic intensity), the more is the potential gain with the use of GPUs. GPUs can handle large data sets working in, what NVIDIA and other researchers describe as single program multiple data (SPMD).

Designing general purpose codes so that they can take full advantage of GPU features is, however, not an easy task. Good knowledge of the targeted GPU architecture is required and new reconstruction algorithms, or at least new implementations of them, have to be developed. GPUs became easier to program with the recent introduction of “abstraction layers”, that is, higher-level programming tools which are highly independent on the particular targeted GPU. For instance, CUDA [7], the application programming interface (API) developed by NVIDIA, offers a unified hardware and software solution for parallel computing on CUDA-enabled GPUs. It consists of C language extensions which allows for easier integration with larger C programs. Kernels may be coded as C functions which may be invoked to execute high-performance numerical libraries. This eases the job of coding complex computational problems on the GPU. Through CUDA, the GPU can be viewed as a computing device able to execute a very high number of threads in parallel. The GPU capabilities for handling multiple instances of the same operation on multiple data are accessed transparently. The CUDA environment also provides with memory management functions, which are analogous to the standard C functions. Additionally, CUDA makes it possible for all threads in a given local group (known as a block of threads) to access a small pool of fast shared memory. CUDA algorithms can be run in different GPU models and will run essentially unmodified in the next generation of CUDA-capable GPUs.

Several iterative tomographic reconstruction codes using GPUs have already been implemented for CT and PET [8]–[14], and some of these [13], [14] are also based on CUDA. Nevertheless, in most cases, especially for PET, the codes running in GPUs make use of a simplified model of the scanner at play, or

Manuscript received September 01, 2010; revised December 17, 2010 and February 06, 2011; accepted April 25, 2011. Date of publication June 23, 2011; date of current version October 12, 2011. This work was supported in part by AMIT Project funded by CDTI (CENIT Programme), UCM (Grupos UCM, 910059), CPAN (Consolider-Ingenio 2010, CSPD-2007-00042), RECAVA-RETIC network, Comunidad de Madrid (ARTEMIS S2009/DPI-1802), Ministerio de Ciencia e Innovación, Spanish Government (ENTEPRASE grant, PSE-300000-2009-5 and TEC2007-64731/TCM), and European Regional funds.

J. L. Herraiz and J. M. Udias are with the Grupo de Física Nuclear, Departamento Física Atómica, Molecular y Nuclear, Universidad Complutense de Madrid, Madrid, Spain (e-mail: joaquin@nuclear.fis.ucm.es).

S. España was with the Grupo de Física Nuclear, Universidad Complutense de Madrid, Madrid, Spain, and is now with the MEDISIP, Department of Electronics and Information Systems, Ghent University-IBBT-IBiTech, Ghent, Belgium (e-mail: samuel@nuclear.fis.ucm.es).

R. Cabido and A. S. Montemayor are with the Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain (e-mail: antonio.sanz@urjc.es).

M. Desco is with the Departamento de Bioingeniería e Ingeniería Aeroespacial, Universidad Carlos III de Madrid, Madrid, Spain, and also with the Unidad de Medicina y Cirugía Experimental, Hospital General Universitario Gregorio Marañón, Madrid, Spain (e-mail:desco@mce.hggm.es).

J. J. Vaquero is with the Departamento de Bioingeniería e Ingeniería Aeroespacial, Universidad Carlos III de Madrid, Madrid, Spain (e-mail: juanjose.vaquero@uc3m.es).

of the reconstruction algorithms, different from the optimum model or algorithm that would have been used for the CPU reconstruction.

We developed FIRST [1] (Fast Iterative Reconstruction Software for [PET] Tomography) using the message passing interface (MPI) protocol [15] to launch parallel tasks on the available CPUs (or CPU cores) in a cluster of computers. FIRST is based on a realistic scanner model obtained with the Monte Carlo simulation code PeneloPET [16]. The accurate scanner model allows for higher and more uniform resolution along the FOV. Indeed, FIRST has proven to be a successful implementation of a tomographic code for high-resolution small-animal PET scanners [1], [17], in terms of image quality and modest reconstruction time. One of the key advantages of FIRST with respect to other codes is the way it fits the large system response matrix (SRM) in a small amount of RAM, thus improving the performance of the code [1].

We adapted FIRST to use the efficient computing capabilities of GPUs, calling this new implementation GFIRST. The main goal was to obtain a significant acceleration of the algorithm without compromising the quality of the reconstructed images. This would prove that CUDA is powerful enough to translate CPU reconstruction codes into GPU codes, with no essential modifications and yet achieving optimal performance. We aimed to speed-ups large enough to compete with the reconstruction times obtained in a cluster of CPUs. Flexibility of the code was also an important requisite. We wanted a code with no GPU-specific optimizations and that would allow for possible future modifications with no significant additional effort. We also wanted the GPU code to be as similar as possible to the CPU code, thus making it easier to handle and debug.

One of our main concerns was that some approximations needed to adapt the code to the GPU could lead to loss of accuracy or artifacts in the final images. Therefore, we avoided approximations in the forward and backward projection kernels, and we used the same SRM as in the original CPU code. Furthermore integer conversion of float image values was avoided. Atomic instructions were avoided altogether, as floating-point atomic adds were not available in our GPUs. This did not represent a limitation in our code, as the backward projection kernel was implemented, as described later, in a voxel-driven fashion. This means that it was implemented destination-driven and, as parallelization over the destination avoids thread write locks, atomic adds were not necessary.

At variance with previously proposed reconstruction codes for the GPU [8], [19] which dealt with list-mode data, our code was designed to work with sinograms [18]. Although list-mode data, for which all the relevant information from each detected coincidence is stored, might provide optimal images, sinogram data organization also has some interesting features and advantages. First, sinograms are commonly used in most of the current commercial scanners [18], and they are often easily available to the user. Usually, their size is smaller than list-mode files, so they are easier to handle and store. Furthermore, in a sinogram, data are spatially ordered and can thus be accessed in a simple and ordered way. This allows for very fast backward projection implementations. Finally, under certain approximations imposed by the sinogram, the simulated system exhibits many symmetries, thus reducing the size of the SRM. This is described in detail in Section II.

The high-resolution small-animal PET scanner VrPET [17] was chosen as a test bed. This scanner is a good example of the large fraction of preclinical PET scanners consisting of an incomplete ring of rotating detectors.

II. Materials and Methods

A. Brief Description of the CPU Implementation

For the sake of completeness, we describe the main features of FIRST. For a more detailed description, the reader is referred to [1]. FIRST implements a fully 3-D iterative reconstruction of PET data based on a realistic model of radiation emission and detection. This model was generated using the Monte Carlo code PeneloPET [16], which is based on PENELOPE [20]. PeneloPET simulates PET systems composed of crystal array blocks coupled to photodetectors. It allows the user to define radioactive sources, detectors, shielding, and other parts of the scanner. The acquisition chain is simulated in high detail; for instance, electronic processing includes pile-up rejection mechanisms and time stamping of events. Due to blurring effects involved in emission (positron range, non-collinearity, voxel size) and detection of radiation in PET scanners (inter-crystal scatter, crystal size) [21], each line-of-response (LOR) is connected to a wide region of the field-of-view, commonly known as the tube-of-response (TOR), instead of a simple geometrical line connecting a pair of detector elements. 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. The coefficients from all the TORs in the scanner form the SRM. In order to fit the SRM into RAM, the symmetries present in the system should be exploited. Only some TORs need to be computed and stored, because symmetrically-equivalent LORs, as shown in Fig. 1, have the same probability distribution. Even with this approach, for most scanners the storage needed may exceed the available RAM of standard computers. This issue was resolved in a previous work [1] using quasi-symmetries taking advantage of the fact that LORs with a very similar angle with respect to the scintillator crystal would contain very similar coefficients [Fig. 1(c)]. Therefore, it suffices to simulate and store just a few TORs taken along selected LORs (super-LORs) without compromising the quality of the reconstructed images.

Scatter in the object is not taken into account when simulating the SRM. Incorporation of scattered photons in the SRM has been proposed [23], but it makes the SRM much larger and more difficult to handle. Furthermore, it would make the SRM object-dependent.

We target the high-resolution small-animal PET scanner VrPET [17], which is composed of two pairs of rotating plane detectors in coincidence. Sinograms are organized in 117 radial and 190 angular bins (180 degrees rotation) for each of the direct and oblique crystal combinations, making a total of $117(\text{radial})'190(\text{angular}) \times 30 \times 30(\text{axial})$ bins.

When dealing with sinograms acquired in a continuous rotating scanner such as VrPET, the probability of event detection does not depend on the co-polar angle θ, as any possible dependences are averaged out due to the rotation motion. As a result, rotational and axial symmetries yield an SRM that depends only

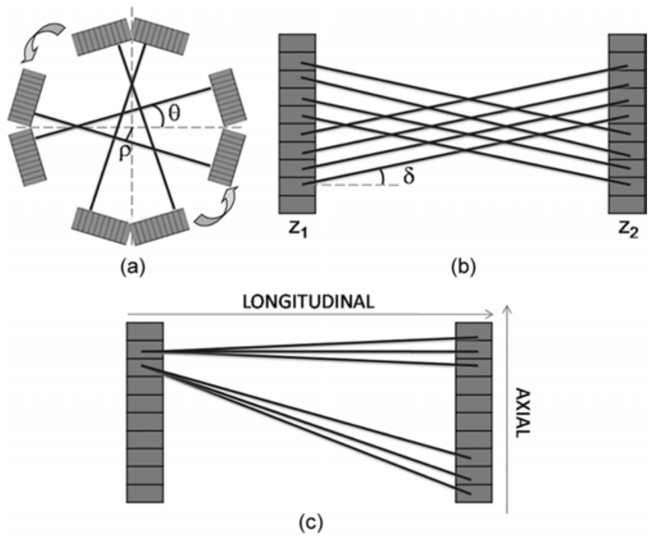

Fig. 1. Symmetries used to reduce the number of LORs that need to be stored in memory: (a) in-plane rotations, (b) translations and reflections in the axial direction, and (c) quasi-symmetries in the axial direction. Coefficients for TORs in (a), (b), and (c) have identical or very similar values.

TABLE I
SIZE OF THE SYSTEM RESPONSE MATRIX

Storage Method (Type of Symmetries Used)	Number of Super-LORs	Memory Size
None	117 × 190 × 30 ×30	214 GB
Axial	117 × 190 × 30	7.1 GB
Axial and Rotational	117 × 30	38 MB
Axial, Rotational and In-Plane	59 × 30	19.3MB

on the radial distance ρ and the ring-difference $|z_1 - z_2|$ of the LOR:

$$LOR(\rho, \theta, z_1, z_2) \rightarrow SuperLOR\left(|\rho|, |z_1 - z_2|\right) \qquad (1)$$

Using all the symmetries of this scanner, the number of super-LORs required is not too large, and the use of quasi-symmetries is not necessary, as shown in Table I. Each TOR of these super-LORs is composed of several thousand coefficients. The total size of the SRM is a few MB, allowing us to store the SRM as a 3-D texture of the GPU.

In this work, each TOR consists of an array of $117(\text{longitudinal}) \times 7(\text{transaxial}) \times 7(\text{axial})$ coefficients. In our tests, these values offered a good trade-off between accuracy and reconstruction time. The width of the TORs was large enough so that the values beyond the tails which are cut off were always below 5% of the maximum value. These maximum values usually appear along the axis of the TOR.

Following other authors [5], [24], we assume that all crystals in the detector block have the same response, independently of the row to which they belong. Thus, we did not take into account edge effects. The impact of this assumption on the reconstructed images is minimized by the crystal normalization procedure [1].

While in non-rotating scanners the sinogram may have large gaps [25] of missing data, for a continuous rotating scanner such as VrPET, there are no missing data due to gaps between detector blocks.

The reconstruction algorithm used in this work was OS-EM [26]. This algorithm updates the estimation of the image $X_j^{(t)}$ obtained at iteration t by multiplying it by $F_j^{(t)}$, a weighted average of a subset S of data corrections [(2)]. These corrections are obtained as the ratio of the measured data Y_i and the estimated data from the image $X_j^{(t)}$, with the SRM coefficients acting as the weighting factors C_{ij}. In this notation, i denotes the LOR index and j the voxel index:

$$\begin{aligned} P_i^{(t)} &= \sum_j C_{ij} \cdot X_j^{(t)} && \text{(Forward Projection)} \\ F_i^{(t)} &= \frac{\sum_{i\in S} C_{ij} \cdot Y_i^{(t)} / \mathrm{P}_i^{(t)}}{\sum_{i\in S} C_{ij}} && \text{(Backward Projection)} \\ X_j^{(t+1)} &= X_j^{(t)} \cdot F_i^{(t)} && \text{(Image Update).} \end{aligned} \qquad (2)$$

The forward projection operation evaluates the estimated data $P_i^{(t)}$ and the backward projection operation computes the corresponding image of correction factors $F_j^{(t)}$. These two steps are, by far, the most time-consuming parts of the code, accounting for more than 90% of the reconstruction time.

We sorted oblique sinograms defined by each crystal detector pair (z_1, z_2) using the axial slice $z_0 = (z_1 + z_2)$ and the ring difference $\Delta\mathrm{z} = (z_1 - z_2)/2$ [27]. The azimuthal angle δ was defined as $\tan(\delta) = \Delta z / D$ [28], where D represents the distance between detectors. All oblique sinograms are considered, i.e., no cuts in the maximum ring difference are imposed.

The reordered $[\rho, \theta, (z_0, \delta)]$ sinogram data were then divided into subsets. They were chosen so that all the voxels in each subset were connected to a similar number of LORs in order to uniformly span the FOV. This is important to avoid artifacts derived from the number of chosen subsets [29]. Each subset was composed of a fraction of the total number of 5700 angular θ, δ combinations, each one composed of all the radial and axial ρ, z_0 bins.

Using sinograms in fully 3-D mode and wide TORs, the number of subsets per iteration that can be used without compromising the quality of the reconstructed images is large. This is due to the high number of TORs to which each voxel is connected. In our tests, 50 subsets per iteration was a reasonable value (each one with 114 angular combinations). A single iteration sufficed to obtain valid reconstructions. The number of voxels in the images was set to $117 \times 117 \times 59$.

As the main purpose of our work was to study the performance of the GPU implementation [30], data were not corrected for randoms, attenuation, or scatter in either the GPU or the CPU reconstructions. Sensitivity normalization of the scanner was obtained acquiring a blank annulus filled with $^{68}\mathrm{Ge}$, as described in [17]. Misalignments of the detector blocks were taken into account when creating the sinogram.

The original code FIRST, written in FORTRAN, was translated into ANSI C. The code was compiled in both cases with Intel compilers (v10.1) [31]. The execution speed of both FORTRAN and C versions of FIRST was similar. The CPU C code was run in a single core of a CPU-Intel Core i7 (2.93 GHz) as a reference.

B. GPU Description as Exposed by CUDA

In this section, we describe the most relevant characteristics of the GPU for this work, as seen by NVIDIA CUDA [32]. A detailed description of CUDA is found in [7]. One of the most

TABLE II
CLOCK CYCLES REQUIRED TO ACCESS MEMORY IN GPU

GPU Memory Type	Number of Cycles
Register	1
Shared Memory	1
Constant Memory	1-100s (depending on cache locality)
Texture Memory	1-100s (depending on cache locality)
Global Memory	400 – 600

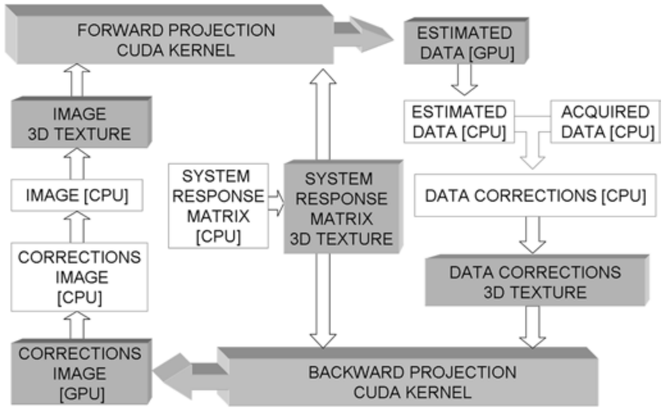

Fig. 2. Flowchart of the implementation of the code in the GPU.

TABLE III
DIMENSIONS OF SRM 3-D TEXTURES

Axis and Direction	Number of TORs	Number of coefficients in each TOR	Total number of coefficients
X (Longitudinal)	1	117	117
Y (Transaxial)	59	7	413
Z (Axial)	30	7	210

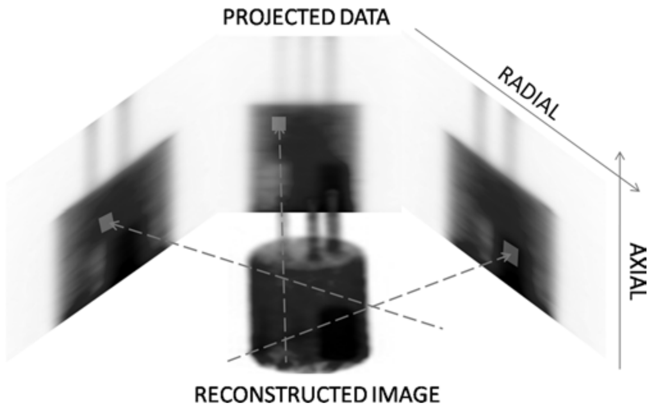

Fig. 3. Forward projection of LORs from different angles.

important parameters of a GPU is the total number of stream multiprocessors (SM). Each of these modules has its own resources and, therefore, the total number of threads which can be executed in parallel is proportional to the number of SM. To execute a particular kernel, CUDA distributes the computing resources of the GPU into a grid of blocks with up to 512 threads each. Each SM of the GPUs used can take up to 8 blocks simultaneously, with a maximum of 768 threads per SM. The optimal number of blocks and threads to be defined varies depending on the available resources of the GPU and the complexity and characteristics of the kernel.

Due to the large number of threads that can be used for parallel computation on GPUs, the usual bottlenecks of these implementations are memory access and memory transfers. In our study, we extensively used texture memory, which resides in device memory and is cached.

Table II presents an estimation of the number of clock cycles required to access each type of memory in the GPU [33], [34]. Access to registers and textures is much faster than access to global memory. Three-dimensional textures became available in recent versions of CUDA (>2.0), thus making the implementation of our code straightforward.

C. Description of the GPU Implementation of the Algorithm

Since forward and backward projections take up most of the reconstruction time, only these two steps were implemented as CUDA kernels called from the main reconstruction C code, running in the CPU. Fig. 2 shows the data flow between CPU and GPU. It is convenient to minimize communications between both devices due to the limited bandwidth (typically 4 GB/s for PCI Express 1.X interconnections) [35]. We verified that in our code, the time spent in these transfers is much less than the time spent in the CUDA kernels.

We defined three 3-D textures, one for the image being reconstructed, another for the SRM, and a third corresponding to the corrections obtained after comparing measured and estimated data. The dimensions of the 3-D texture that stores the SRM are shown in Table III. In this work, the size of the SRM is relatively small, and no size-limitation problems were found. In the case of larger SRMs, we used quasi-symmetries to reduce their size.

The SRM is uploaded into GPU global memory as a 3-D array and then attached to a 3-D texture at the start of the program. As the texture is cacheable, access to these coefficients is much faster than if they were stored directly in global memory (see Table II). This texture is kept in the GPU memory and remains unmodified during the reconstruction.

Forward Projection: In the forward projection kernel, each thread projects one TOR by adding the contribution from all voxels connected with it (Fig. 3). The resulting sum is stored by each thread in a fast-access temporary register. Only the final total value is stored in global memory as an array of projections.

The forward projection kernel is invoked only once per subset. Several co-polar θ and azimuthal δ angles, with their corresponding radial and axial bins, are projected in the same way as in the CPU code. The number of TORs projected is much greater than the number of threads which can be executed simultaneously. This way, the code is ready to take advantage of more powerful GPUs.

Fig. 4 shows the pseudo-code describing schematically the forward projection step for several TORs, each one evaluated using different threads. In general, the points sampled by a TOR do not correspond to the center of the voxels of the image. In the present study, we used tri-linear interpolation within GPU-textures, which is very efficient, to obtain the value of the image at the points sampled by the TORs.

Backward Projection: Ratios between measured data and projections are computed by the CPU and uploaded into the GPU, where they are stored as a 3-D texture of corrections with radial, axial, and angular indexes. In this case, bi-linear interpolation in radial and axial directions within this texture is exploited.

```
For each TOR to be projected (One per thread)
  Find its corresponding Super-LOR
  For each POINT (i,j,k) in the TOR {
    From the Super-LOR and (i,j,k) → SRM index (XP,YP,ZP)
    Find the coordinates in the image (x,y,z)
    VALUE_SRM = tex3D(texSRM,XP,YP,ZP)
    VALUE_IMAGE = tex3D(texImg,x,y,z)
   SUM_PROJ+=VALUE_SRM*VALUE_IMAGE
  PROJECTION[LOR]=SUM_PROJ
```

Fig. 4. Pseudo-code of the forward projection.

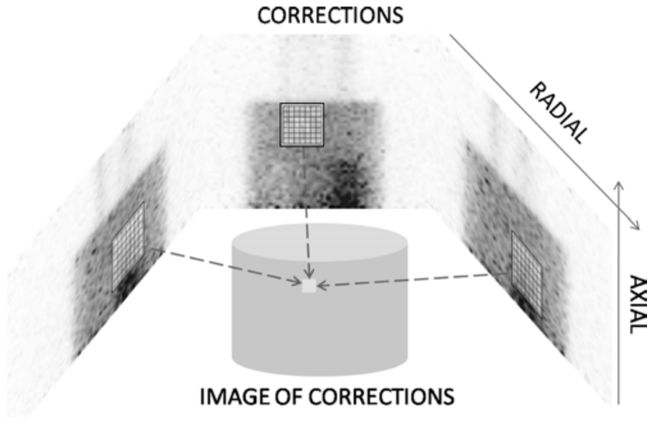

Fig. 5. Backward projection in a voxel from several TORs. The 7×7 grid represents the corrections connected to a voxel at each angle pair $[\theta, \delta]$.

```
For each VOXEL (One per thread)
  For each projected [θ,δ] angle
   Find radial and axial position of the voxel in projection space
   For each TOR connected with the voxel in that angle
    Find its corresponding Super-LOR
    Find voxel position in the TOR
    From Super-LOR and voxel pos. → SRM index(XP,YP,ZP)
    value_SRM = tex3D(texSRM,XP,YP,ZP)
    value_corr = tex3D(texCorr,radial_pos,axial_pos,angle)
    value_image_SRM+= value_SRM
    value_image_corr+= value_corr*value_SRM
  IMG_CORR[Voxel]+=value_image_corr
  IMG_SENS[Voxel]+=value_image_SRM
```

Fig. 6. Pseudo-code of the backward projection.

In the backward projection kernel, each thread is responsible for the back-projection of a given voxel by adding contributions from all previously projected LORs connected to it (Fig. 5). As described before, with our SRM, a voxel is connected to 7×7 TORs for each angle pair $[\theta, \delta]$.

Numerator and denominator required for the backward projection (2) are obtained by storing in temporary registers all the contributions from the TORs. At the end of the process, these accumulated values are stored in global memory as correction and sensitivity images, respectively. The pseudo-code for this kernel is shown in Fig. 6. The use of temporal registers instead of global memory to perform the sums speeds up this kernel noticeably. This reflects the need for appropriate use of the different memory types available to the GPU.

Code Optimization: NVIDIA CUDA architecture exhibits a small pool of fast *shared memory* (Table II) accessible by all threads in the same block. Nevertheless, shared memory has

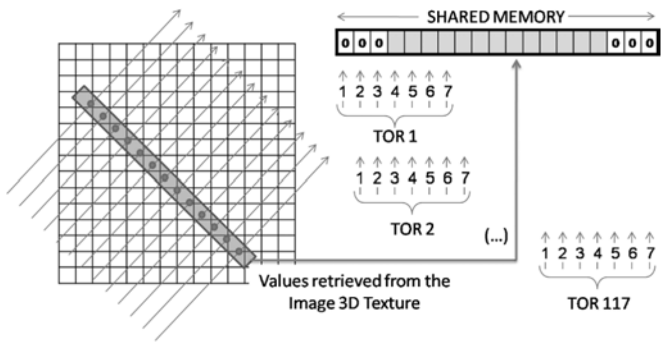

Fig. 7. Scheme of the optimization of the forward projection based on shared memory.

a very limited size and its use requires some recoding of the algorithm to continuously reallocate small portions of data from global or texture memory into the shared memory pool.

We have compared the performance of a version of the GPU code in which shared memory was not used and threads of the same block were not synchronized [*without shared memory optimization* (WO-SM)] with an optimized version where the forward projection was accelerated by working with synchronized threads that used shared memory [*with shared memory optimization* (W-SM)].

Fig. 7 sketches how the sampled values of the image retrieved from the texture are stored temporarily in shared memory and later reused by all overlapping TORs, thus reducing the number of texture access by a factor 7. Some additional zero values were added at both edges of the shared memory array to ensure that all threads get the same number of data, avoiding divergent branches. In Fig. 7, the numbers in each TOR represent the order in which each thread accesses shared memory values. The size (16 kB) of the available shared memory in our GPUs and the limitation on the maximum number of threads in a block imposed that values for just one line across the image were stored at the same time. In our case, this represents 117 values (the number of threads in our blocks). This could be adapted to more general configurations, by splitting the data retrieved from the texture in small subsets.

In order to obtain the best performance from a GPU code, it is important to consider the SIMD nature of the streaming multiprocessors. Thread divergences as a result of different threads in a block taking different code branches may cause a significant loss of performance. In this work we did not expect many thread divergences, as all the LORs in the forward projection and all the voxels in the backward projection are connected with the same number of data, and thus, all threads have to perform the same number of operations. We verified this with the CUDA Visual Profiler [36]. We found only a 2% of divergent branches in the forward projection and a 5% in the backward projection.

D. Performance Evaluation

We compared CPU and GPU codes using simulated data from a PeneloPET simulation of an image quality (IQ) phantom filled with ^{18}F as well as real data from a 30-min acquisition of a 200-g rat injected with FDG. In both cases, we studied the total processing time required for reconstruction of a single-bed, single-frame acquisition in single precision. The CPU where the code was run was among the fastest on the market at the time of writing. The CPU code was compiled

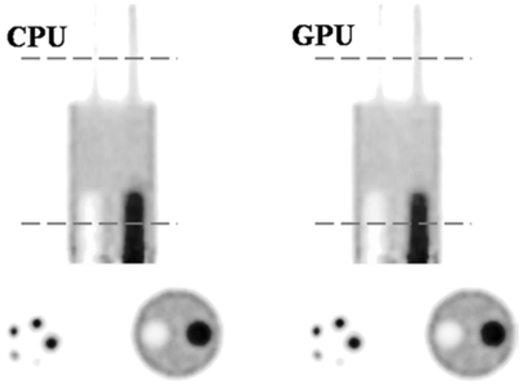

Fig. 8. Coronal view (top) and two transverse views (bottom) of the reconstructed image obtained from a simulated IQ phantom using CPU and GPU codes. Transverse views correspond to the slices marked with a line in the top figure.

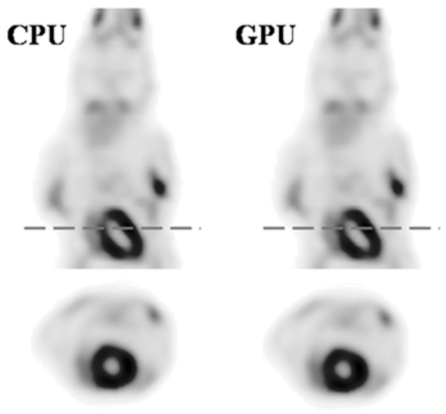

Fig. 9. Image reconstructed from a real acquisition both in CPU and GPU. Coronal view (top) and transverse view (bottom) of the reconstructed image of a 200-g rat FDG acquisition. Transverse views correspond to the slice marked with a line in the upper figure.

with the Intel C compiler (v10.1) [31] with-*fast* optimization option that includes loop-unrolling. As mentioned above, the time required for reconstructing one image using one core of a multi-core CPU is taken as reference. The reconstruction time required in a cluster of computers is easily estimated, as good parallelization has been reported elsewhere [1]. The CUDA code we propose here was run in different GPUs to study the performance for several number of stream processors.

Differences between the image reconstructed by the CPU (X_j^{CPU}) and the GPU (X_j^{GPU}) were evaluated from the root mean square (rms) deviation (3) using the total number of voxels J:

$$\sigma_{rms} = \sqrt{\frac{1}{J}\sum_{j=1}^{J}\left[\left(X_j^{GPU} - X_j^{CPU}\right)^2 \Big/ X_j^{CPU}\right]}. \quad (3)$$

III. Results

The reconstructed images obtained using both the CPU and the GPU codes are compared in Figs. 8 and 9. A coronal view of the reconstructed image of the simulated IQ phantom is shown in Fig. 8. The differences between the images are visually negligible. The deviation is $\sigma_{\text{rms}} = 0.07\%$.

Fig. 9 shows coronal and transverse views of the image of the 200-g rat injected with FDG. The difference between both images is also very small, with a $\sigma_{\text{rms}} = 0.09\%$.

TABLE IV
Reconstruction Time for One Image (One Bed, One-Frame Acquisition, One Full Iteration) for Different Architectures

Architecture	Time (s)	Speed-up factor
CPU–Intel® Core™ i7 (2.93GHz) (6GB) DDR3 - 800 MHZ RAM	3456	1×
GPU - 8600 GT - 256 MB – 4 SM	509	7×
GPU - 8800 GTS - 640MB – 12 SM	175	20×
GPU - 8800 GT - 512 MB – 14 SM	128	27×
GPU - TESLA C1060 - 4GB – 27 SM	49	72×

Table IV shows the total time, including overheads, required to reconstruct the simulated IQ phantom in different hardware architectures. The GPU code includes the optimization described in Section II. It is noteworthy that the same CUDA code running on different GPUs shows significant performance differences, even by one order of magnitude. Reconstruction time in the GPU decreases almost linearly with the number of available SMs.

We verified that the time spent in data transfers between the CPU and the GPU is much less than the time spent in the forward and backward projection kernels. According to the CUDA Visual Profiler [36], these transfers only amount to 0.5% of the reconstruction time.

The reconstruction time of the WO-SM and W-SM codes in our fastest GPU, a TESLA C1060, was 72 and 49 s, respectively. Thus, the reconstruction time is reduced by a factor of 1.5 with shared memory optimization, and the same images are obtained. This speed-up factor is almost the same for other GPUs.

IV. Discussion and Conclusions

A fast fully 3-D reconstruction code based on sinograms and a realistic SRM is of great interest. PET studies requiring many frames will benefit enormously from fast reconstructions. We successfully implemented the iterative fully 3-D reconstruction software FIRST in CUDA. CUDA allowed for a straightforward GPU implementation (GFIRST) that was to a large extent independent of the version of the NVIDIA GPU card running the code. Furthermore, when memory allocation and access were looked over to avoid slow memory access, the slightly recoded GPU version ran 50% faster than the non-optimized one. Images reconstructed with GFIRST and FIRST were essentially identical, with rms deviations of 0.09% or less.

We achieved a very significant improvement in reconstruction time (with a speed-up factor of up to 72) for the fastest GPU compared to a single core of a high-end CPU. This is remarkable, as FIRST had already been shown to be a highly optimized reconstruction code.

Furthermore, GFIRST running on different GPUs obtains speed-ups that are approximately proportional to the number of SMs of each GPU card, thus hinting to the scalability of the code for different GPUs with no need to re-code. This shows the benefit of implementing applications based on abstraction schemes such as CUDA (or also eventually Khronos' OpenCL for heterogeneous computing [37]), which will make it possible to benefit fully from GPUs with more streaming processors.

GFIRST has been tested with sinograms from the small-animal PET scanner VrPET, which is a relatively simple device with a few detectors. Nevertheless, the methods described in this work are flexible enough to be easily adapted to other scanners.

The extension of this work to other PET systems is currently under development.

The code includes a method to speed up the forward projection, as described in this work, but further optimizations are still possible. For instance, the number of divergent branches could be reduced or avoided with some modifications, and the number of voxels of the reconstructed images could be changed to better fit with the characteristics of the GPU. This will be explored in a future work.

Our results highlight the need of image reconstruction codes to adapt to the current paradigm shift in computer architecture, which is moving from a single, fast execution unit with a large memory to several execution units with a small local memory and more power-efficient execution. Nowadays, even consumer-level systems commonly possess multi-core processors. These less power-consuming, high-performance multi-core systems are successfully replacing more power-consuming desktop computers, and the industry predicts that future computing systems will increasingly rely on scalable technology [38]. Reconstruction software implemented with CUDA or other abstraction layers, such as the one described in this work, will immediately benefit from the next generation of GPUs.

References

[1] J. L. Herraiz *et al.*, "FIRST: Fast iterative reconstruction software for (PET) tomography," *Phys. Med. Biol.*, vol. 51, no. 18, pp. 4547–4565, Sep. 2006.

[2] D. Brasse, P. E. Kinahan, R. Clackdoyle, M. Defrise, C. Comtat, and D. Townsend, "Fast fully 3D image reconstruction in PET using planograms," *IEEE Trans. Med. Imag.*, vol. 23, no. 4, pp. 413–425, Apr. 2004.

[3] P. Dupont and J. Warwick, "Kinetic modelling in small animal imaging with PET," *Methods*, vol. 48, no. 2, pp. 98–103, Jun. 2009.

[4] M. D. Jones and R. Yao, "Parallel programming for OSEM reconstruction with MPI, OpenMP, and Hybrid MPI-OpenMP," in *Proc. IEEE Nuclear Science Symp. Med. Imag. Conf.*, 2004, pp. 3036–3042.

[5] I. K. Hong *et al.*, "Ultra fast symmetry and SIMD-based projection-backprojection (SSP) algorithm for 3-D PET image reconstruction," *IEEE Trans. Med. Imag.*, vol. 26, no. 6, pp. 789–803, Jun. 2007.

[6] General-Purpose computing on Graphics Processing Units Repository. [Online]. Available: http://www.gpgpu.org.

[7] NVIDIA CUDA Programming Guide v.2.3.1, NVIDIA Corp., 2009. [Online]. Available: http://www.nvidia.com/object/cuda_home.html.

[8] G. Pratx, G. Chinn, P. D. Olcott, and C. S. Levin, "Fast, accurate and shift-varying line projections for iterative reconstruction using the GPU," *IEEE Trans. Med. Imag.*, vol. 28, no. 3, pp. 435–445, Mar. 2009.

[9] V. B. Bhat, "High-speed reconstruction of low dose CT using iterative techniques for image-guided interventions," Ph.D. dissertation, Dept. Elect. Eng., Univ. Maryland, College Park, MD, 2008.

[10] B. Keck *et al.*, "High resolution iterative CT reconstruction using graphics hardware," in *Proc. IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2009, pp. 4035–4040.

[11] F. Xu and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Phys. Med. Bol.*, vol. 51, pp. 3405–3419, 2007.

[12] X. Jia, Y. F. Lou, and R. J. Li, "GPU-based fast cone beam CT reconstruction from undersampled and noisy projection data via total variation," *Med. Phys.*, vol. 37, pp. 1757–1760, 2010.

[13] S. Schaetz and M. Kucera, "Integration of GPGPU methods into a PET reconstruction system," in *Proc. Parallel and Distributed Computing and Networks Conf.*, Mar. 2010.

[14] W. Craig, S. Thada, and W. Dieckmann, "A GPU-accelerated implementation of the MOLAR PET reconstruction package," in *Proc. IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2009, pp. 4114–4119.

[15] W. Gropp, E. Lusk, and A. Skjellum, *MPI Using MPI: Portable Parallel Programming With the Message-Passing Interface*. Cambridge, MA: MIT Press, 1999.

[16] E. España *et al.*, "PeneloPET, a Monte Carlo PET simulation toolkit based on PENELOPE: Features and validation," *Phys. Med. Biol.*, vol. 54, no. 6, pp. 1723–1742, Mar. 2009.

[17] E. Lage *et al.*, "Design and performance evaluation of a coplanar multimodality scanner for rodents imaging," *Phys. Med. Biol.*, vol. 54, no. 18, pp. 5427–5441, Sep. 2009.

[18] F. H. Fahey, "Data acquisition in PET imaging," *J. Nucl. Med. Technol.*, vol. 30, no. 2, pp. 39–49, Jun. 2002.

[19] A. Reader *et al.*, "One-pass list-mode EM algorithm for high-resolution 3-D PET image reconstruction into large arrays," *IEEE Trans. Nucl. Sci.*, vol. 49, no. 3, pp. 693–699, Jun. 2002.

[20] J. Baró *et al.*, "PENELOPE: An algorithm for Monte Carlo simulation of the penetration and energy loss of electrons and positrons in matter," *Nucl. Instrum.. Meth. Phys. Res. B*, vol. 100, pp. 31–46, May 1995.

[21] J. R. Stickel and S. R. Cherry, "High-resolution PET detector design: Modeling components of intrinsic spatial resolution," *Phys. Med. Biol.*, vol. 50, no. 2, pp. 179–195, 2005.

[22] J. L. Herraiz, S. España, E. Vicente, J. J. Vaquero, M. Desco, and J. M. Udías, "Noise and physical limits to maximum resolution of PET images," *Nucl. Instrum.. Meth. Phys. Res. A*, vol. 580, no. 2, pp. 934–937, Oct. 2007.

[23] P. J. Markiewicz, M. Tamal, P. J. Julyan, D. L. Hastings, and A. J. Reader, "High accuracy multiple scatter modelling for 3D whole body PET," *Phys. Med. Biol.*, vol. 52, pp. 829–847, 2007.

[24] C. A. Johnson and A. Sofer, in *Proc. 7th Symp. on the Frontiers of Massively Parallel Computation*, Los Alamitos, CA, 1999, pp. 126–137, IEEE Computer Society Press.

[25] J. L. Herraiz, S. España, E. Vicente, E. Herranz, J. J. Vaquero, M. Desco, and J. M. Udías, "Frequency selective signal extrapolation for compensation of missing data in sinograms," in *Proc. IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2008, pp. 4299–4302.

[26] H. M. Hudson and R. S. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," *IEEE Trans. Med. Imag.*, vol. 13, no. 4, pp. 601–609, Dec. 1994.

[27] M. Defrise and P. E. Kinahan, "Data acquisition and image reconstruction for 3D PET," in *Theory and Practice of 3D PET*. Dordrecht, The Netherlands: Kuwer, 1999.

[28] M. Takahashi and I. Ogawa, "Selection of projection set and the order of calculation in ordered subsets expectation maximization method," in *Proc. IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 1997.

[29] M. Defrise, P. E. Kinahan, D. W. Townsend, C. Michel, M. Sibomana, and D. F. Newport, "Exact and approximate rebinning algorithms for 3-D PET data," *IEEE Trans. Med. Imag.*, vol. 16, no. 2, pp. 145–158, Feb. 1997.

[30] J. L. Herraiz, S. España, S. Garcia, R. Cabido, A. S. Montemayor, M. Desco, J. J. Vaquero, and J. M. Udias, "GPU acceleration of a fully 3D iterative reconstruction software for PET using CUDA," in *Proc. IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2009, pp. 4064–4067.

[31] [Online]. Available: http://software.intel.com/en-us/intel-compilers.

[32] CUDA Version 2.3 With NVIDIA Drivers Version 191.07.

[33] D. Kirk and W. W. Hwu, The CUDA Memory Model, ECE 498 AL: Applied Parallel Programming Course, 2007–2009.

[34] S. Ryoo *et al.*, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," *Proc. 13th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, pp. 73–82, 2008.

[35] Peripheral Component Interconnect Special Interest Group (PCI-SIG), PCI Express 2.0 Specification, Jan. 2007.

[36] [Online]. Available: http://www.nvidia.com/object/cuda_programming_tools.html.

[37] Khronos OpenCL Working Group, The OpenCL Specification (Version: 1.0), 2009.

[38] K. Asanovic *et al.*, The landscape of parallel computing research: A view from Berkeley Tech. Report Elect. Eng. Comput. Sci. Dept., Univ. California, Berkeley, Dec. 2006.