Title
Sparse domain approaches in dynamic SPECT imaging with high-performance computing.

Journal
American Journal of Nuclear Medicine and Molecular Imaging, 7(6)

ISSN
2160-8407

Authors
Pan, Hui
Chang, Haoran
Mitra, Debasis
et al.

Publication Date
2017

Peer reviewed

Am J Nucl Med Mol Imaging 2017;7(6):283-294
www.ajnmmi.us /ISSN:2160-8407/ajnmmi0069155

Original Article

Sparse domain approaches in dynamic SPECT imaging with high-performance computing

Hui Pan[1*], Haoran Chang[1], Debasis Mitra[1*], Grant T Gullberg[2], Youngho Seo[2]

[1]School of Computing, Florida Institute of Technology, Melbourne, FL 32901, USA; [2]Department of Radiology and Biomedical Imaging, University of California, San Francisco, CA 94143, USA. [*]Equal contributors.

Received May 26, 2017; Accepted December 2, 2017; Epub December 20, 2017; Published December 30, 2017

Abstract: Iterative reconstruction algorithms often have relatively large computation time affecting their clinical deployment. This is especially true for 4D reconstruction in dynamic imaging (DI). In this work, we have shown how sparse domain approaches and parallelization for static 3D image reconstruction and 4D dynamic image reconstruction (directly from sinogram) in Single Photon Emission Computed Tomography (SPECT), without any intermediate 3D reconstructions, can improve computational efficiency. DI in SPECT is one of the hardest inverse problems in medical image reconstruction area and slow reconstruction is a challenge for this promising protocol. Our work hopefully, paves a new direction toward making DI in SPECT clinically viable. Our 4D reconstruction also is a novel application of non-negative matrix factorization (NNMF) in an inverse problem.

Keywords: Sparse image reconstruction, 4D reconstruction, dynamic SPECT, pre-clinical, high performance computing, GPU-based reconstruction

Introduction

Computation time is a well-known bottleneck for iterative image reconstruction algorithms development within any research laboratory where repeated experimentation with different versions and data are needed, and for eventual clinical translation of such algorithms. This is true across almost all imaging modalities. The problem is more acute in 4D reconstruction for dynamic imaging (DI), where the problem complexity increases by an order of magnitude. DI promises additional information relevant to diagnosis, as different types of tissues have different uptake and washout characteristics or temporal signature of the tracer.

A key to efficient computation in many imaging problems is the sparse nature of data. Sparsity comes in two forms: explicit zero values in an image or other relevant data (e.g., system matrix), or implicit, in terms of information content of data. For example, a Fourier transform may reduce the size of data by eliciting frequency components and then keeping only the important ones. Such a sparsification may be lossless or not (not completely recoverable).

In dynamic SPECT imaging (with fixed and rotating camera heads) we encounter an inverse problem where sparsification is not only needed for computational efficiency, but also needed to make it feasible to solve the problem [1]. In a typical SPECT DI, the gamma camera continuously collects data as it rotates and the imaging starts as soon as the tracer is being injected. In contrast, the conventional static image is acquired after the tracer concentration stabilizes. In SPECT DI only a limited view projection is available at each instance (one or two, depending on available SPECT camera heads) while the tracer concentration is still changing. This makes the optimization problem highly underdetermined with far more unknowns to solve (4D dynamic image) than the available data (sequence of 2D projections, each from a rotating camera view at different time instances). We have developed the *Spline-initialized Factor Analysis in Dynamic Structures* (SIFADS) algorithm [2], that addresses this problem by a sparsification approach and thus, by significantly reducing the dimensionality of the problem. Our approach is an application of the *non-negative matrix factorization* (NNMF) approach

popularized by [3] within the linear algebra and data science communities.

In order to handle large dimensionality of the problem we also deploy an explicit sparsification or data compression technique by eliminating zero values from our computation. This not only saves computational time but also memory requirements of the algorithm. We also used a novel way to use GPU for faster implementation of our iterative reconstruction process, borrowing from [4]. Each of the GPU processing elements synchronously compute the same instruction on different data elements and shares same memory. While there has been active adaptation of using high-performance computing hardware for reconstruction in positron emission tomography (PET), there are relatively fewer developments for SPECT reconstruction using similar techniques. Furthermore, SPECT DI reconstruction (which requires much more extensive computational power than that needed by static reconstruction) using a high-performance computing technique has not been investigated either.

Main contributions of this work are (1) propose a computational view of dynamic image reconstruction in SPECT, and (2) parallelization steps for a high performance implementation of the algorithm, and (3) to take advantage of a *Graphics Processing Unit* (GPU) architecture to optimize efficiency of the algorithm.

In the next section, we first describe a computational view of the MLEM algorithm and the SIFADS algorithm. In section three, we show how we validate both of these algorithms. The computational performance comparing CPU and GPU implementations are discussed in the section four. The last section concludes the article providing some future directions of this work.

Computational views of static and dynamic reconstruction

MLEM [5] uses a statistical model of Poisson distribution of photon counts on a detector pixel. The following formula summarizes the algorithm for computing estimated 3D SPECT image in each iteration $\widehat{v_k^{next}}$ from that before the iteration $\widehat{v_k}$ using the input sinogram g_n and the precomputed system matrix $a_{n,k}$, with n the sinogram pixel index, and k the image voxel index.

$$\widehat{v_k^{next}} := \frac{\widehat{v_k}}{\sum_n a_{n,k}} \sum_n \frac{g_n}{\sum_n a_{n,k} v_k} a_{n,k} \quad (1)$$

Each iteration in MLEM involves two primary components, forward projection (FP) and back-projection (BP). FP estimates sinogram $\widehat{g_n}$ (hat indicates estimated value rather than the input g_n as in eq. 1) from the 3D image as one of the denominators in eq. 1:

$$\widehat{g_n} := \sum_k a_{n,k} v_k \quad (2)$$

A colon (:) indicates direction of assignment or data flow. BP estimates a correction factor for each voxel of the image:

$$\sum_n \frac{g_n}{\widehat{g_n}} \left(a_{n,k} \right) =: B_k \quad (3)$$

Subsequently, normalized correction term updates the image voxel by voxel:

$$\frac{v_k B_k}{A_k} =: \widehat{v_k^{next}} \quad (4)$$

The constant normalization term for each voxel is pre-computed from the system matrix once before the iterations start:

$$\sum_n a_{n,k} =: A_k \quad (5)$$

Image v is arbitrarily initialized, typically with all ones for voxels, and iterations loop over eqs. 2-4.

The SIFADS algorithm [2] for SPECT DI estimates tissue specific time activity curves (TACs) directly from projections data. The forward problem of DI in this model is:

$$g_{n,t} := \sum_k a_{n,k} v_{k,t} \quad (6)$$

with t indicating time instance.

Factorizing v, the 4D image volume as in NNMF [3] we express,

$$v_k := \sum_j C_{k,j} f_{j,t} \quad (7)$$

where j is the index of factors, f are 1D time series used as basis vectors in factorization, and three dimensional (with the dimensions of image volume v) C are the corresponding coefficients. With this factorization the forward model in eq. 6 becomes,

$$g_{n,t} := \sum_{k,j} a_{n,k} C_{k,j} f_{j,t} \quad (8)$$

This factorization method is not completely new in the literature. The *spline-based method* [6] has also used a similar factorization

approach with a fixed set of b-splines for *f* as a set of non-orthogonal basis functions. The *factor analysis* [7] in dynamic structures or FADS optimized *f* as well as *C*, alternately within each iteration [8]. A major problem in both of them is that they are very sensitive to the arbitrary initial choice of values of *C* and *f*. SIFADS' main contribution is to use spline-based method for estimating the initial values of *C* and *f* for the subsequent FADS approach. This pre-processed initialization achieves strong stability, at the cost of a few extra iterations (around five) of the spline-based optimization. The factorization of the problem (eq. 8) also gains in computational speed, because the amount of data to be processed is reduced with the dimensionality reduction. A lesser data transfer between memory units also facilitates further gain in efficiency with GPU programs.

SIFADS uses *maximum a posteriori* (MAP) algorithm [9], which is a MLEM version where total variation (TV) regularization is applied to smooth the reconstructed image. The MAP objective function *L* for SIFADS is:

$$L(P \mid C,f) = \sum(-\sum(aCf) + g\ln(\sum(aCf)) - \ln(g!)) - U(C,f) \quad (9)$$

where *ln* is the natural logarithm, and *U* is the regularization function, and

$$U(C,f)=\lambda_1\Omega(C)+\lambda_2\Theta(C)+\lambda_3\Phi(f) \quad (10)$$

Ω(C) provides a penalty term for anisotropic (without crossing any object boundary) TV term for smoothness over each coefficient, and Φ(f) is the TV measured with L_1 norm over each factor. Θ(C) is a penalty term that prevents coefficients from overlapping with each other in space [2]. The equations for updating *f* and *C* in each iteration are obtained by equating the partial derivatives of *L* with respect to *f* and *C* respectively.

$$\widehat{C_{k,J}^{next}} := \frac{\widehat{C_{k,J}}}{\sum_{n,t}\left(a_{n,k}^T I_{n,t}\right)f_{J,t}^T - \frac{\partial U}{\partial C}} \sum_{n,t} a_{n,k}^T \frac{g_n}{\sum_{k,j,t} a_{n,k} C_{k,j} f_{j,t}} f_{j,t}^T \quad (11)$$

$$\widehat{f_{J,I}^{next}} := \frac{\widehat{f_{J,I}}}{\sum_{k,n} C_{k,j}\left(a_{n,k}^T I_{n,t}\right) - \frac{\partial U}{\partial f}} \sum_{k,n} \frac{g_n}{\sum_{k,j,t} a_{n,k} C_{k,j} f_{j,t}} a_{n,k} C_{k,j} \quad (12)$$

where $I_{x,y}$: a matrix of size (*x*,*y*) with all elements equal to 1, *n*: pixel index, *k*: voxel index, *j*: id of the factor, *t*: time index, same as id's of projections. Eqs. 11 and 12 contains same matrix operations as in static MLEM reconstruction (eqs. 1-5). In our parallelization strategy, each thread corresponded to one factor. Meanwhile, a limited number of nonzero (NZ) pixel values (e.g. ≤557, for our canine data) were loaded from the system matrix $a_{n,k}$ to the GPU thread. FP estimates the sinogram $\widehat{g}$:

$$\widehat{g_n} := \sum_{k,j,t} a_{n,k} C_{k,j} f_{j,t} \quad (13)$$

For eq. 11 FP estimates correction terms for coefficients for each voxel:

$$\sum_{n,t} a_{n,k}^T \frac{g_n}{\widehat{g_n}} \left(f_{j,i}^T\right) =: B_k \quad (15)$$

and correction terms for factor for each time unit, on eq. 12:

$$\sum_{n,t} \left(a_{n,k}\right) \frac{g_n}{\widehat{g_n}} \left(C_{k,j}\right) =: B'_k \quad (14)$$

Spatial smoothness function are applied to the voxels that belong to the same tissue type (determined by a threshold on coefficient value and from prior anatomical information provided additionally, e.g. in our case a 3D static reconstruction of consistent data of the later camera rotations 3rd through 20th):

$$\triangle C := \sum_{n,t} \left(a_{n,k}^T I_{n,t}\right) f_{j,t}^T - \frac{\partial U}{\partial C} \quad (16)$$

Temporal smoothness function minimizes the differences of the factor arrays:

$$\triangle f := \sum_{k,n} C_{k,j} \left(a_{n,k}^T I_{n,t}\right) - \frac{\partial U}{\partial f} \quad (17)$$

Finally, the new coefficients and new factors are:

$$\widehat{C_{k,J}^{next}} := \frac{\widehat{C_{k,J}}}{\triangle C} B_k \quad (18)$$

$$\widehat{f_{J,I}^{next}} := \frac{\widehat{f_{J,I}}}{\triangle f} B'_k \quad (19)$$

The ray-driven system matrix is generated (including point-response function) separately [10] and is stored in a compressed format (only NZ values) as a binary file. Our home-grown compression format is specifically designed for emission or transmission tomography, called *YZ* format, due to its originator Yuval Zelnik [11]. The *YZ* compressed representation uses a relational data structure (**Figure 1**) rather than the traditional compressed matrix representation. The system matrix file contains a sequence of four elements (n, nz_n, K_n, G_n): *n* is pixel id (of all pixels, i.e., sequence from 0

Table 1. Platforms used for comparison

Hardware/Method	CPU	GPU
Processing cores	AMD Opteron 6128, 8 core	NVIDIA Tesla M2070, 448 cores
RAM	32 GB	6 GB
Implementation Function	C++	C++ with CUDA 2.0 kernel

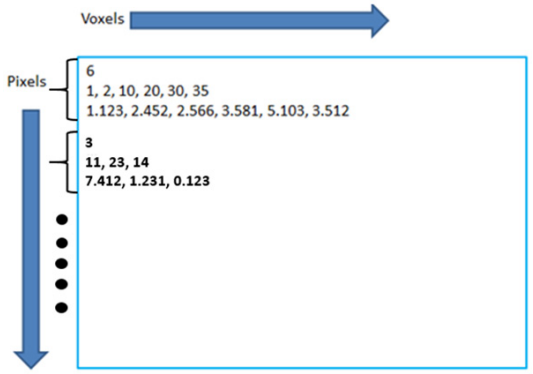

Figure 1. YZ sparse matrix format for system matrix.

through *N*-1, for *N* pixels), nz_n is the *number of* NZ voxel values for this pixel, K_n is a vector of nz_n voxel indices (each index is an 8-bit integer) and G_n is a vector of nz_n corresponding voxel values (each value is a 32-bit double precision float) in exactly the same sequence as that of K_n. A pixel that has no NZ voxels will still have its pixel id written followed by 0 written for nz_n on the file indicating absence of the following two vectors.

Sparse format creates additional complexity in matrix operations as the number of *NZ* elements (nz_n) is not a constant and the vectors lengths K_n and G_n vary from pixel to pixel. The system matrix elements, elements of the input sinogram, estimated sinogram, the voxel correction factors, and estimated voxel values, all need to be distributed over GPU computing elements and aligned in the available limited memory per thread, in order to maximize localization of computing. This is because local memory on each GPU element is not large and the whole system matrix cannot be loaded there. Data transfer between slower memory to faster memory located closer to the computing unit, is a major bottleneck in efficient image reconstruction. A memory distribution or "chunking", needs to be done even for any CPU implementation in case the system matrix is too large to fit in the available RAM memory. In a GPU implementation, the "chunking" operation needs to be sensitive to the localization of data (pixel or voxel elements that are to be estimated, along with the system matrix elements). We have also shown here how such improved memory assignments may enhance GPU performance further with two different experimentations, one without memory alignment and one with it. **Table 1** shows our CPU and GPU configurations for our experimentations.

GPU threads are organized in blocks and threads. Appropriate utilization of the architecture is crucial in efficient implementation with GPU. CUDA 2.0 allows maximum 65535× 65535 blocks, with 1024 concurrently running threads available per block. A thread is expressed with two dimensions *p* and *q*. Maximum allowed threads per block in our system is $p \times q = 1024$. Preferable number for *p* is 36 for best performance, and hence a maximum value of integer multiple of 36 that is ≤1024, or, 1008 number of threads may run concurrently on a block. A thread is a sequence of operations written as a kernel routine in the code. Multiple concurrent threads run the same sequence operations synchronously on different input.

The number of threads used in each CUDA *kernel* (code fragment whose copies run as multiple threads with differing data) is a multiple of the actual chunk size for the current pixel for FP (for the current voxel for BP, see eqs. 2 and 3 respectively) and on each thread the voxel ids and corresponding system matrix values are passed to the GPU.

As mentioned before the system matrix is too large to fit in the GPU memory, and so, needs to be loaded chunk by chunk during FP and BP operations. On the other hand, tackling a varying number of *NZ* voxel elements for each pixel in the system matrix makes it difficult to decide the size of such chunks. To overcome this difficulty, first, we load the system matrix as three vectors on CPU memory (a vector *A* of all NZ chunk sizes for all pixels (*nzn* for all *n*: a vector of unsigned int *nz1*, *nz2*, ...), a vector B of all voxel indices with NZ values (of *unsigned int K1*, *K2*, ...), and a vector *C* of all the correspond-

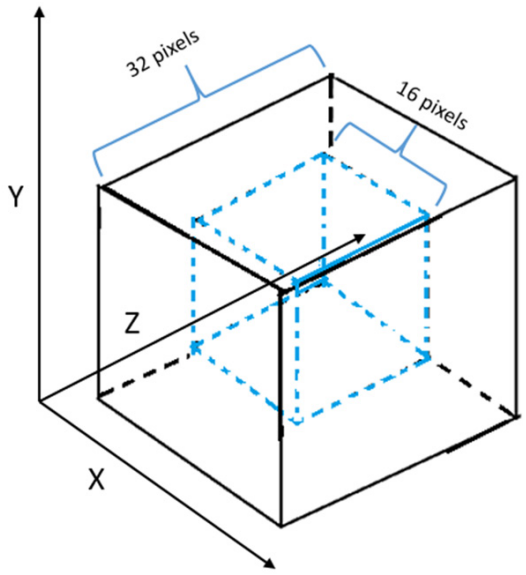

Figure 2. Cartoon for numerically simulated volumes (of a sample dimensions) used for static images reconstruction.

ing NZ values (of real numbers *V1*, *V2*, ...)). This resampling is done in CPU before iterations start with GPU.

We compute the maximum possible number of NZ values ($max(nz_n)$) over all pixels for FP with a scan of the first vector *A*. Subsequently, a multiple of this number $u \times max(nz_n)$, where *u* is an integer >0, is used as the chunk size for loading from *A*, *B* and *C* on memories of GPU elements (threads). Similarly for BP, we check for maximum number of NZ pixel values in the system matrix over each of the voxels (say, $max(nz_k)$) and the chunk size is determined according to this $max(nz_k)$.

Each block in CUDA is expressed with three dimensions *x*, *y*, and *z*. In FP (eqs. 14-15), for example, they are mapped to the number of factors, the maximum number of NZ voxels per pixel $max(nz_n)$, and the number of pixels in one projection respectively (e.g. for our canine data they are 4, 557 and 1280, respectively). Again, due to the GPU global memory size limitation, we cannot put data for all time units (e.g., 72 for canine data) into the thread level. In that case, we needed loops to load and compute with partial data into the thread level synchronously, e.g., 2 loops with 36 time elements or 35 projections for canine data (note again, each projection corresponds to a time-instance). For canine data the total dimension for the block and the thread is (4×557×1280)×(36×1). After all threads on each blocks are finished, the results are off loaded from GPU to CPU, and the next data block is loaded from CPU to GPU. This reduces the bandwidth between CPU and GPU communication, thus improving performance.

Methods

We have experimented SIFADS implementations for dynamic SPECT reconstruction. We have used three types of data sets: static imaging data generated with NCAT simulations, dynamic imaging data with the same and canine data from real SPECT DI. They are described below.

Static SPECT

In order to measure the GPU performance on different data sizes, we have created four numerically simulated volumes of different resolutions (or binary matrices). They are of dimensions: 16×16×16, 32×32×32, 64×64×64 and 128×128×128 with voxel values of 1 within the volume and 0 on the background [12]. **Figure 2** shows a cartoon of such a ground truth of volume in the center. In order to produce a sinogram for each data size, we generated respective SPECT system matrices for the following acquisition parameters: (1) 120 projections over 360 degrees, (2) single detector head of size corresponding to the respective volume (e.g., for volume with 64×64×64 voxels, detector head is of 64×64 pixels size), and (3) a low-energy high-resolution (LEHR) parallel-hole collimator. Finally, a forward projected sinogram was created for each of these to which Poisson noise was added. No attenuation or scatter corrections were applied to these system matrices.

The first choices for thread organizations in reconstructing for four data sizes are, respectively: 16×16=256 blocks and 16 threads/block; 32×32=1024 blocks and 32 threads/block; 64×64=4096 blocks and 64 threads/block; and 128×128=16384 blocks and 128 threads/block. The second set of GPU implementations have the threads reorganized respectively, according to available CUDA cores: 16 blocks with 16×16=256 threads/block; 32 blocks with 32×32=1024 threads/block; 64×4=256 blocks with 6×16=1024 threads/block; and 128×16=2048 blocks with 128×8=1024 threads/block. This takes the advantage of maximum allowed number of threads per block (1024) for our system.

We express the dimensionality of a static reconstruction problem by $U \times V \times P \times X \times Y \times Z$, where $U \times V$ is the detector dimension in pixels, P is the number of projections, and $X \times Y \times Z$ is the reconstructed volume's dimension.

Dynamic SPECT with NCAT phantom

The simulated dynamic datasets were generated from an NCAT phantom [13] of 64×64×41 voxels size for cardiac imaging with a real system matrix for GE Millennium VG3 SPECT machine. A dynamic sinogram is generated by forward projecting the phantom with different input TACs that resemble tissue TACs for the known activities of the tracer ^{99m}Tc-sestamibi for cardiac SPECT studies over three types of regions: blood-pool in heart, myocardium, and liver. We first segmented the phantom volume for these three ROIs. Then, the curves in **Figure 3A** are used to model the change of tracer activities in the blood pool, myocardium, and liver segments of the NCAT volume respectively. The curves are b-splines and are based on our experience on how the tracer behaves in the respective organs. These segments and the three curves are subsequently used as ground truths for comparing the corresponding reconstructed output TACs and their coefficients respectively. Acquisition parameters used for forward projection are: (1) LEHR parallel-hole collimation, (2) one detector head, (3) 64×64 bins per projection angle, (4) 72 projections over 360 degrees rotation, and (5) camera rotating at a speed of one second per projection, i.e., 72 seconds for a full rotation. No scattering or attenuation is used in generating system matrix over this experiment with simulated dynamic data. Generated dynamic sinogram with added Poisson noise (**Figure 3A**) is the input to the SIFADS implementations.

For this dynamic reconstruction, the natural thread organization used for the GPU implementation is: (64×41) 2624 blocks and (72×3) 216 threads/block, according to the dimensionality of the problem. After thread reorganization, the improved GPU implementation has: (16×41) 656 blocks, with (4×72×3) 864 threads/block for 3 factors (or tissue types).

Canine dynamic SPECT

Our third data set is from a pre-clinical canine cardiac rest-study performed with a GE Millennium VG3 Hawkeye SPECT/CT camera with LEHR parallel-hole collimator where the detector dual heads were in H-mode (two heads are opposite or at 180-degree angle to each other) and was rotating continuously. Injection of 3.7 mCi (1.37×10^8 Bq) of ^{201}Tl tracer was administered at the onset of acquisition that continued for 20 minutes. For each rotation, two sets of 72 one-second projections over 360 degrees were acquired. Each view contained 64×64 projection bins (4.42 mm). In the reconstruction with GPU the initial thread organization is: (64×20) 1280 blocks and (72×4) 288 threads/block, and the reorganized structure is: (32×20) 640 blocks, (2×72×4) 576 threads/block for 4 factors. The system matrix embeds attenuation (measured from CT scan prior to SPECT scan) and collimator scatter correction. Gullberg lab [14] provided data and the system matrix.

For expressing data sizes of dynamic image reconstruction problem, an added dimension is F, the number of factors. The number of time units is same as P with one detector head (one projection per time-unit) in SPECT, but must be an integer $P/2$ for two simultaneously rotating heads. Thus, the dimensionality for SPECT with single head camera is $U \times V \times P \times X \times Y \times Z \times F$.

We measured the spatial and temporal accuracies and the SNR for each of the reconstructed images from GPU and CPU implementations over the simulated data sets, where the ground truths are known. Spatial accuracy was measured from the estimated coefficients of each tissue type. After applying a threshold, the coefficients represented segmentations of the imaged volume into a number of tissue types based only on their temporal behaviors and not just their spatial locations. These segmentations were then evaluated using the dice similarity coefficient (DSC) metric [15]:

$$\mathrm{DSC}\left(\widehat{C_j}\right) = \frac{2 \times \left| \widehat{C_j^{est}} \, n \widehat{C_j^{true}} \right|}{\left| \widehat{C_j^{est}} \right| + \left| \widehat{C_j^{true}} \right|} \tag{20}$$

where $\widehat{C_j^{est}}$ denotes the binary mask created from the thresholded estimated coefficients for the factor j or tissue type j, $\widehat{C_j^{true}}$ denotes the corresponding ground truth from the original NCAT phantom, and $|C|$ indicates the number of voxels in C. DSC =1 indicates a perfect segmentation, and DSC =0 indicates no overlap between the estimated and the true segments. Note that, all our validation efforts are involved with the 3D coefficients and the correspond-

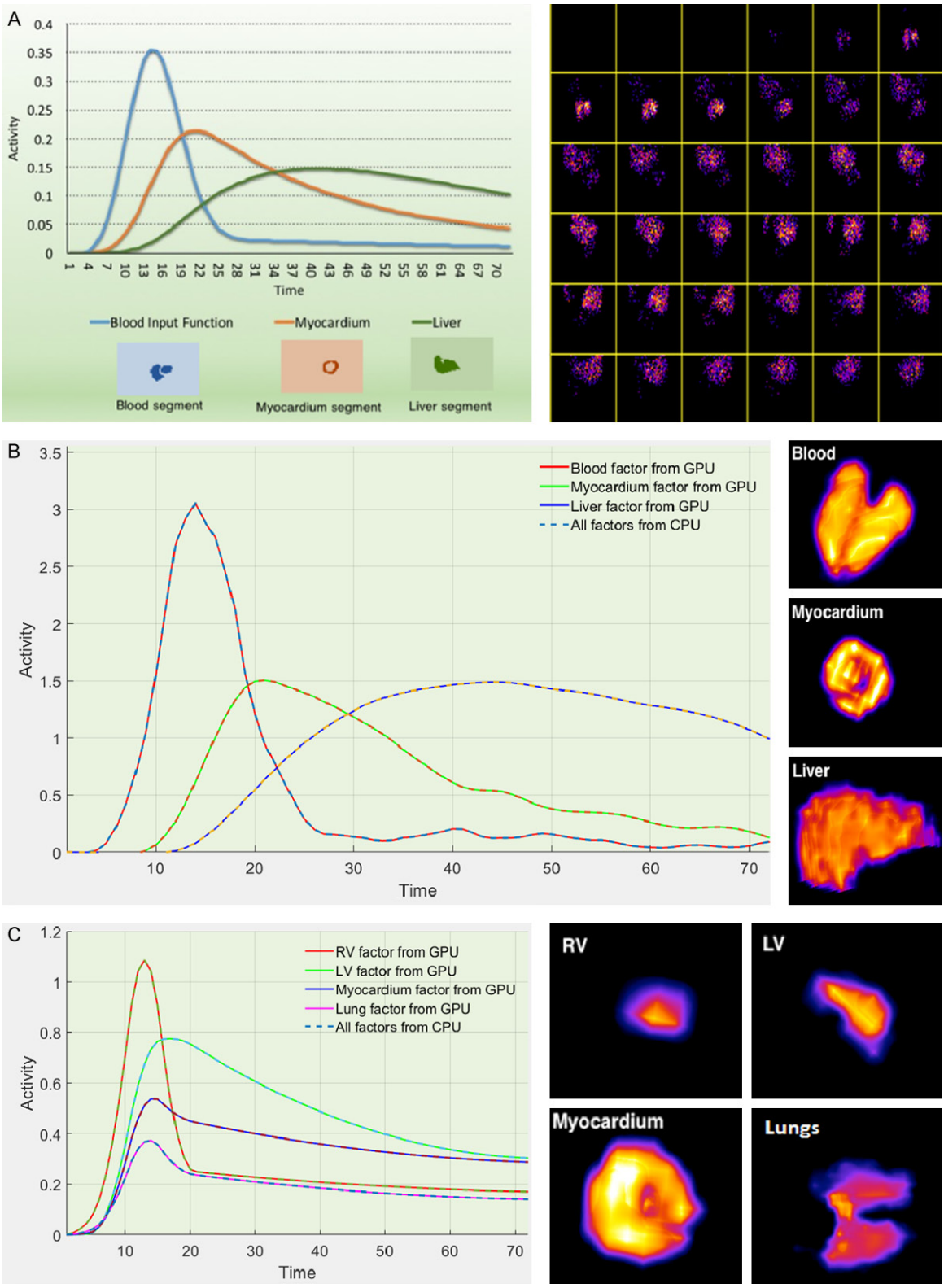

Figure 3. A. *Left*, the initial b-splines set and corresponding segments of NCAT used for producing the forward projected dynamic sinogram. *Right*, dynamic projections of NCAT data. B. *Left*, final reconstructed TACs, blood (blue), myocardium (red), and liver (green). Results from CPU implementation are the dashed curves. *Right*, blood-pool, myocardium and liver coefficients from GPU based dynamic SIFADS reconstructions of NCAT data over 30 iterations. C. Canine dynamic SPECT studies, results from a GPU version. *Left*, final TACs for four tissue types. *Right*, slices of reconstructed tissues (i.e. coefficients) with 30 iterations of SIFADS. Each image is scaled to fit the same sized box.

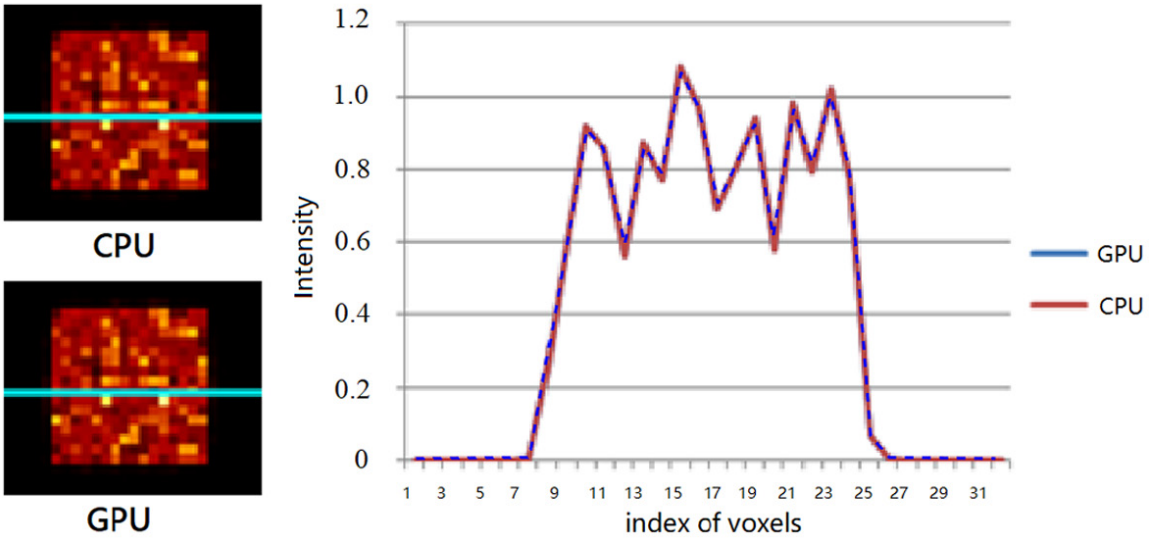

Figure 4. *Left,* the 16th transverse slices from GPU and CPU implementations-based static MLEM reconstruction algorithms for the volume size 32×32×32 with 20 iterations. *Right*, two plots are intensity values on Y-axis of the voxels against the line profiles on left. Curves from GPU and CPU are identical as expected and overlapped each other on the right figure.

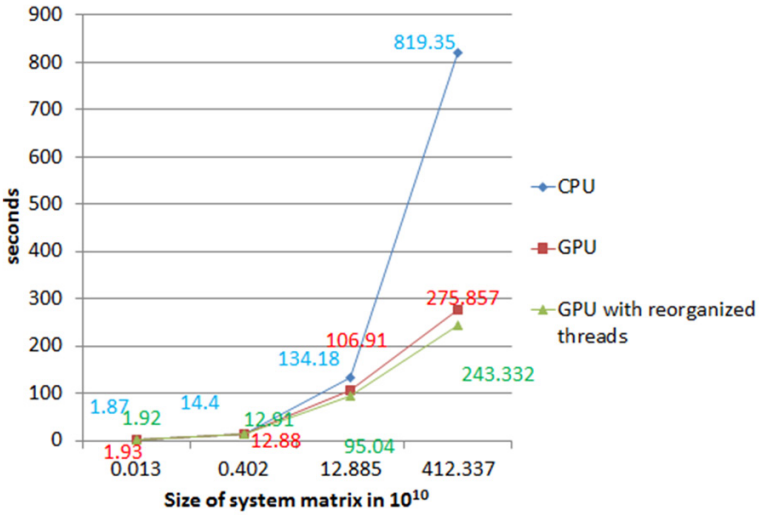

Figure 5. The X-axis is data size represented by that of the system matrix equal to the size of volumes times the size of the corresponding sinograms, varying from 163×162×120 to 1283×1282×120. Y-axis is time taken for 20 iterations (arbitrarily chosen for acceptable quality of reconstruction) of MLEM.

ing factors. These (*C* and *f*) may be multiplied to produce a 4D image if necessary.

Temporal accuracy is measured by computing the *Root Mean Square* (RMS) difference between the reconstructed TACs and the known ground truth:

$$RMS(TAC_j) = \sqrt{\frac{\sum_i \left[(TAC_j^{est})_i - (TAC_j^{true})_i\right]^2}{\sum_i (TAC_j^{est})_i^2}} \quad (21)$$

where $(TAC_j^{est})_i$ denotes the value of the estimated TAC of tissue type j at time t_i, $(TAC_j^{true})_i$ is the corresponding ground truth TAC value used as an input in the simulation. RMS =0 indicates perfect recovery of the TACs.

Signal-to-noise ratio (SNR) is measured for the reconstructed images:

$$SNR = \frac{\sum_{k=1}^{N} x_k / N}{\sqrt{\sum_{k=1}^{N} \left(x_k - \sum_{k=1}^{N} x_k / N\right)^2 / N}} \quad (22)$$

where x_k denotes the value of the k^{th} voxel in the signal region (segments) and N denotes the number of those voxels. In the cases of dynamic images SNR is measured over the coefficients *C*.

Results

Comparison of CPU and GPU implementations of MLEM with varying sinogram sizes

Figure 4 depicts a reconstructed image for 32×32×32 voxels volume from the CPU and the

Table 2. Scalability of GPU SPECT static 3D reconstruction with NCAT phantom (run time in seconds)

Size for each system matrix	CPU (s)	GPU (s)	GPU with threads reorganization	Ratio (CPU vs GPU)	Ratio (CPU vs GPU with reorganization)
$16^3 \times 16^2 \times 120$	1.87	1.93	1.92	0.97	0.97
$32^3 \times 32^2 \times 120$	14.4	12.85	12.91	1.12	1.12
$64^3 \times 64^2 \times 120$	134.18	106.91	95.04	1.26	1.41
$128^3 \times 128^2 \times 120$	819.35	275.86	243.33	2.97	3.37

Table 3. Measure of qualities: Dimensionless measures of quality of SPECT static 3D reconstructions for NCAT phantom

Size for each sinogram	SNR_CPU	DSC
$16^3 \times 16^2 \times 120$	0.443	0.908
$32^3 \times 32^2 \times 120$	0.531	0.911
$64^3 \times 64^2 \times 120$	0.536	0.914
$128^3 \times 128^2 \times 120$	0.552	0.931

They are exactly same (up to three decimal) for all three set of experimentations: on CPU, on GPU and with thread-optimization on GPU.

GPU implementations, verifying that the two implementations result in same images.

Comparison of MLEM reconstruction-times over the different sinogram sizes

Figure 5 and **Table 2** show scalability of the GPU implementation over the CPU implementation, even though for a small data size GPU may take more time than that with CPU (row 1, **Table 2**) because of the communication overhead. With 32 times increase in size of the sinogram, the CPU needs about 7.7 times more computation to do the reconstruction, while the GPU-based computation time has increased by 5.8 times, and threads reorganized GPU implementation time grows by 5.3 times. **Table 3** compares the qualities of the images. They are mostly similar, but may have differed slightly, because of better precision GPU processors [16].

Timing comparison between dynamic reconstructions with CPU and GPU versions of SIFADS

For 4D SPECT reconstruction we have used an NCAT phantom, and a canine cardiac study. The **Table 4** summarizes the timing comparisons for the two data sets: with CPU, with GPU, and with threads reorganized GPU. For measuring data sizes in 4D we incorporate the numbers of tissue types reconstructed, i.e. multiply the volume with the number of factors. Subsequent subsections will discuss more detail of the results obtained by the SIFADS reconstructions from each data set.

Dynamic SPECT NCAT data: A slice of the coefficients for myocardium reconstructed with each of the GPU and CPU implementations is shown in **Figure 6** left. Line profiles on CPU and GPU generated images (**Figure 6** right) show that they are very similar. Improvement of GPU performance is shown in **Table 4**, while **Table 5** shows the quality metrics of NCAT image reconstructions from each implementation. **Figure 3B** shows the result of SIFADS (from GPU): a small region around each coefficient, and the corresponding TACs for each tissue type.

Canine dynamic SPECT data: **Figure 7** compares the two reconstructed coefficients C for the myocardium from CPU vs GPU. **Figure 3C** shows the reconstructed tissues for the GPU version (CPU and GPU versions have same results). We have used four factors here, for left and right ventricles of the blood-pool inside heart, myocardium, and lungs. We were able to slice the projections to eliminate liver from interfering with heart.

Table 6 presents the SNR in each of the reconstructed coefficient-images representing a segmentation for each tissue type. Note again that each coefficient-image is reconstructed independently (in the 3D image domain) within the iterations of the algorithm. Regions having stronger signal content shows higher SNR values as expected.

Discussion

GPU-based sparse image reconstruction shows expectedly a better scalability for both static and dynamic cases. However, reorganization of threads by appropriately mapping the dimensionality of the problem to the system architecture of the GPU platform, improves perfor-

Table 4. Comparison of timing with three versions of SIFADS algorithm over two data sets

30 iterations SIFADS	Data size	CPU Time (sec)	GPU Time (sec)	GPU with threads reorganization	Ratio of CPU vs GPU	Ratio of CPU vs GPU with reorganization
NCAT	4.65×10^{10}	583	198	193	2.94	3.02
Canine	6.05×10^{10}	1862	542	537	3.43	3.46

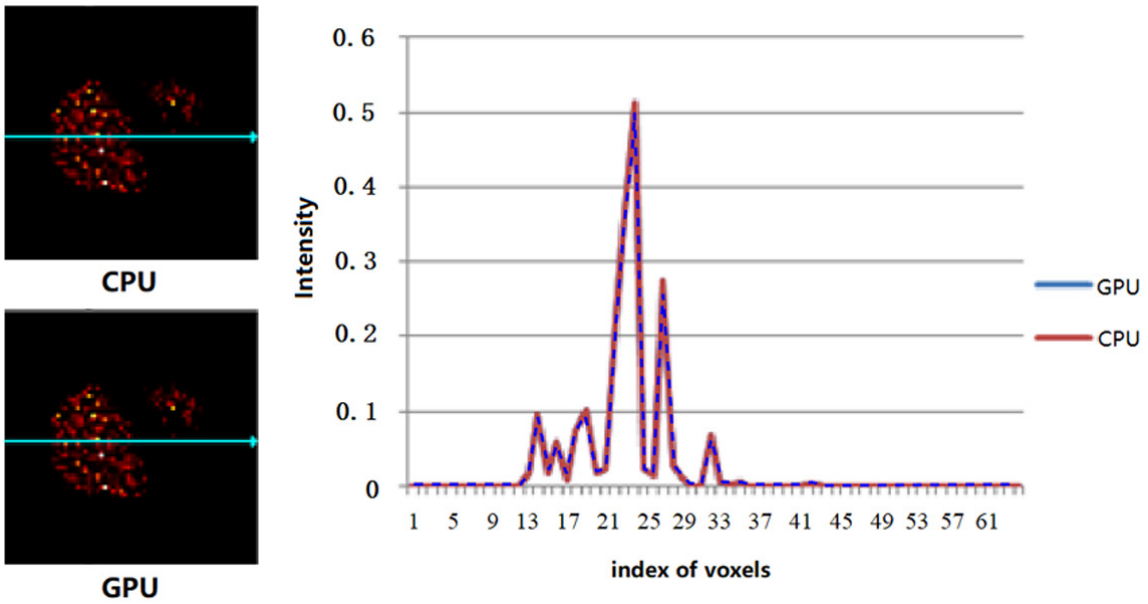

Figure 6. *Left*, the 23rd liver coefficient's transverse slices from CPU (top) and GPU (bottom) based dynamic SIFADS reconstruction algorithms for NCAT data. *Right*, two plots are values of the voxels (blue GPU curve is completely overlapped by the red CPU curve). Curves from GPU and CPU are identical as expected and overlapped each other on the right figure.

Table 5. Quality metrics of 4D reconstruction (for dynamic projections generated from NCAT data) by SIFADS against the ground truth NCAT segments

SIFADS reconstruction of NCAT Data	Spatial accuracy with CPU (DSC)	Spatial accuracy with GPU (DSC)	RMS
Blood	0.981	0.971	0.003
Myocardium	0.867	0.631	0.005
Liver	0.928	0.891	0.007

DSC values are exactly same for both the experimentations on GPU: with and without thread optimization, and slightly better than those found on CPU. RMS values of TACs are exactly same for all three experimentations.

mance further. A major problem in thread reorganization for a GPU implementation of dynamic reconstruction is with some constraints in the GPU system structure itself. For example, the number of threads cannot be more than 1024 in our system, and using a lesser number of threads may not achieve full parallelization and under-utilize the system. This means that the mapping of the dimensions of the problem to the threads and blocks should be such that as many threads (≤1024) as possible are utilized. Also, it is recommended that the number of threads should be a multiple of 32, which provides a better utilization of the physical computing nodes (in CUDA 2). Note that we could use all threads (1024) in static reconstruction with each of the four sizes of data of the numerical phantom, thus achieving maximum feasible parallelization (not presented here but in [17]). However, in the cases of dynamic image reconstructions maximum numbers of threads per block that we could use, given the dimensionalities of our respective reconstruction problems, are: for NCAT 864 (32×27) and for Canine data 576 (32×18), thus, full parallelization could not be achieved. Any GPU implementation needs to be sensitive to this issue.

Large size of the system matrix poses a great challenge in working with small local memory

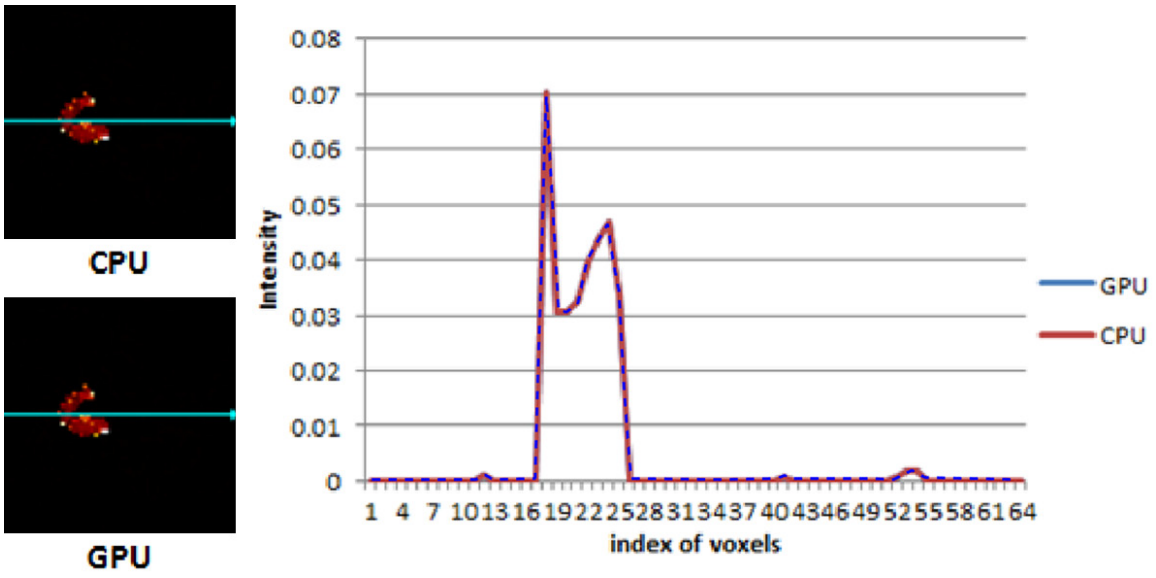

Figure 7. *Left*, the 11th slices myocardium along transverse views from CPU (top) and GPU (bottom) based dynamic SIFADS reconstructions. *Right*, two plots are values on the line profiles on left. Curves from GPU and CPU are identical as expected and overlapped each other.

Table 6. Quality of reconstructions by SIFADS over each tissue type from canine studies

Canine Data	SNR of reconstruction
LV	0.035
RV	0.025
Lungs	0.206
Myocardium	0.328

SNR values on second column are exactly same for all three set of experimentations: on CPU, on GPU and with thread-optimization on GPU.

on GPU, even in cases of our SPECT reconstruction. This gets further complicated with sparse form of the matrix that is a necessity for any efficient implementation. On a different project [12], we have observed more than ten times improvement with GPU in the case CT data because of the use of on-the-fly scheme for generating system-matrix (as is the convention in CT reconstruction, because the size of a system matrix may be too large to fit even on a CPU RAM), where GPU is particularly suitable for a repeated system matrix computation within the iterations. A ray-tracing algorithm was first proposed in [18], which is highly parallelizable. This algorithm lies at the heart of our system matrix generation.

Conclusion

We have studied here the scalabilities of both static and dynamic image reconstructions in SPECT with GPU based implementations. Static reconstruction was performed with conventional MLEM iterative algorithm, which forms the basis of our dynamic reconstruction algorithm SIFADS. Dynamic reconstruction in SPECT is a highly underdetermined problem for limited angle projections. SIFADS takes advantage of the sparse nature of data and uses matrix factorization in order to make the problem solvable. Parallelization significantly reduces computational time (3 to 4 times). This is independent of the speed up of system matrix generation with GPU, where GPU is very suitable for ray-tracing within system-matrix generator, but that is not part of this work as we use pre-computed system matrix.

In future, we will use a different shared memory architecture than that in GPU, and use a distributed memory architecture for our 3D and 4D reconstruction problems. We have also reported here scalability with respect to data size. To the best of our knowledge no such systematic study has been done before, especially for dynamic SPECT image reconstruction. Our future efforts will also be directed in reducing the communication between the processors by parallelizing over multiple GPU units as in [19, 20], where ordered subset expectation maximization-like [21] approaches are used for partial reconstructions on each GPU unit that are combined time to time after certain fixed number of iterations.

Acknowledgements

This work was supported by National Institute of Health grant numbers R01EB07219, R01HL50663, R01HL135490 and R01CA-154561. Yuval Zelnick developed the YZ sparse system matrix data format. We acknowledge Rostyslav Boutchko for providing the canine data.

Disclosure of conflict of interest

None.

Address correspondence to: Debasis Mitra, School of Computing, Florida Institute of Technology, 150 West University Blvd., Melbourne, FL 32901, USA. E-mail: dmitra@cs.fit.edu

References

[1] Gullberg GT, Reutter BW, Sitek A, Maltz JS, Budinger TF. Dynamic single photon emission computed tomography basic principles and cardiac applications. Phys Med Biol 2010; 55: R111-91.

[2] Abdalah M, Boutchko R, Mitra D and Gullberg GT. Reconstruction of 4-D dynamic SPECT images from inconsistent projections using a spline initialized FADS algorithm (SIFADS). IEEE Trans Med Imaging 2015; 34: 216-228.

[3] Lee D and Seong S. Learning the parts of objects by non-negative matrix factorization. Nature 1999; 401: 1999.

[4] Alhassen F, Kim S, et al. Ultrafast multipinhole single photon emission computed tomography iterative reconstruction using CUDA. IEEE Nucl Sc Symp Conf Record 2011.

[5] Lange K and Carson R. EM reconstruction algorithms for emission and transmission tomography. J Comput Assist Tomogr 1984; 8: 306-316.

[6] Reutter BW, Gullberg GT and Huesman RH. Direct least-squares estimation of spatiotemporal distributions from dynamic SPECT projections using a spatial segmentation and temporal B-splines. IEEE Trans Med Imaging 2000; 19: 434-450.

[7] Dipaola R, Bazin JP, et al. Handling of dynamic sequences in nuclear medicine. IEEE Trans Nucl Sci 1982; 29: 1310-1321.

[8] Sitek A, Di Bella EV and Gullberg GT. Factor analysis with a priori knowledge application in dynamic cardiac SPECT. Phys Med Biol 2000; 45: 2619-2638.

[9] Levitan E and Herman GT. A maximum a posteriori probability expectation maximization algorithm for image reconstruction in emission tomography. IEEE Trans Med Imaging 1987; 6: 185-192.

[10] Huesman RH, Gullberg GT, Greenberg WL and Budinger TF. Users manual-donner algorithms for reconstruction tomography. Lawrence Berkeley Laboratory Publication PUB-214, 1977.

[11] Winant CD, Aparici CM, Zelnik YR, Reutter BW, Sitek A, Bacharach SL, Gullberg GT. Investigation of dynamic SPECT measurements of the arterial input function in human subjects using simulation, phantom and human studies. Phys Med Biol 2012; 57: 375-93.

[12] Mitra D, Pan H, Alhassen F and Seo Y. Parallelization of iterative reconstruction algorithms in multiple modalities. IEEE Nucl Sci Symp Conf Rec (1997) 2014; 2014.

[13] Segars WP and Tsui BMW. Study of the efficacy of respiratory gating in myocardial SPECT using the new 4D NCAT phantom. IEEE Trans Nucl Sci 2002; 49: 675-679.

[14] Reutter BW, Botvinick E, Boutchko R, Huesman RH and Gullberg GT. Quantitative rest/stress myocardial perfusion imaging with dynamic SPECT. J Nucl Med 2010; 51: 241.

[15] Dice LR. Measures of the amount of ecologic association between species ecology. Ecology 1945; 26: 297-302.

[16] Whitehead N, Fit-Florea A. Precision & performance: floating point and IEEE 754 compliance for NVIDIA GPUs. Florea 2011; 21.

[17] Pan H. Accelerated tomographic image reconstruction of SPECT-CT using GPU parallelization. Ph.D. Dissertation 2015; Florida Institute of Technology, Melbourne, Florida.

[18] Gullberg GT, Huesman RH, Malko JA, Pelc NJ, Budinger TF. An attenuated projector-backprojector for iterative SPECT reconstruction. Phy Med Biol 1985; 30: 799-816.

[19] Pratx G, Chinn G, Olcott PD and Levin CS. Fast, accurate and shift-varying line projections for iterative reconstruction using the GPU. IEEE Trans Med Imag 2009; 28: 435-445.

[20] Yang,L, Zhou J, Ferrero A, Badawi RD and Qi J. Fast and efficient fully 3D PET image reconstruction using sparse system matrix factorization with GPU acceleration. Phys Med Bio 2014; 59: 403-419.

[21] Hudson HM, Larkin RS. Accelerated image reconstruction using ordered subsets of projection data. IEEE Trans Med Imag 1994; 13: 601-609.