PARALLEL COMPUTATION OF A SPECT PROJECTION OPERATOR FOR A CONTENT ADAPTATIVE MESH MODEL

Francesc Massanes, Jovan G. Brankov

Illinois Institute of Technology

ABSTRACT

In this paper we explore a parallel implementation for fast calculation of a tomographic projection operator for content-adaptive mesh model (CAMM) image reconstruction. Previously we introduced 2D and 3D tomographic image reconstruction using a CAMM for single positron emission computed tomography (SPECT). The proposed parallel method is fast and allows incorporation of a non-uniform attenuation and distance-dependent spatial resolution of the imaging system. This implementation establishes a necessary step for the future development and practical use of the CAMM tomographic reconstruction.

Index Terms— Image reconstruction, Emission Tomography, Parallel computing, SPECT

1. INTRODUCTION

In emission tomography, it is customary to represent volumetric data using uniform sampling and voxel basis functions. In [1] and [2] we proposed a content-adaptive mesh model (CAMM) framework for reconstruction in which the volumetric data to be reconstructed is represented by a non-uniform sampling. This sampling is performed on a spatially adaptive, non-uniform grid (mesh nodes, see Figure 1), followed by partitioning into a collection of non-overlapping tetrahedrons (mesh elements). The image function is then determined over each mesh element by interpolation (usually linear) from its nodal values. In CAMM the nodes are placed most densely in image regions that have fine details by using pre-reconstructed images using filtered-back projection [2].

Previously we compared a CAMM iterative maximum likelihood reconstruction approach with commonly used iterative voxel based methods [2, 3]. These results show that CAMM achieves the best performance in terms of peak signal to noise ratio (PSNR), defect detection measured by channelized Hotelling Observer (CHO) and, since a CAMM is a compact image representation, the shortest computation time as well. However, in the listed work we did not develop a methodology for direct and fast calculation of a projection operator. In [4] and [5] we proposed an accurate model for the calculation of the projection operator (PO), using ray casting methods, but it was prohibitively computational intensive for practical use (for an alternative method one can see [6])

To achieve practical PO calculation time we presented parallel implementation of the proposed ray casting method in 2D case [7] and performed comparison with [6]. The results suggested that the ray casting method has a shorter execution time if a parallel implementation is considered. In this work we go one step forward and present a full 3D parallel implementation of the ray casting method presented in [4, 7].

Our parallel approach is implemented in OpenCL [8], a standard that has been lately used for its incredible resourcefulness and versatility. OpenCL is a royalty-free standard for cross-platform, parallel programming, allowing full utilization of both classical multi core CPUs or powerful graphical processing units (GPU) cards like NVIDIA Tesla without the need to rewrite the computer code. Other successful applications of OpenCL in the field of image reconstruction like in [9, 10, 11] suggest that OpenCL is a perfect candidate for our task.

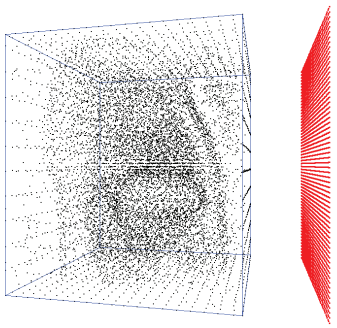


Fig. 1. Non-uniform sampling (nodes) generated using NCAT [12] in black and bins on a collimator camera in red.

2. PROJECTION OPERATOR

In this section we will describe the basics of projection operator calculation, using a ray casting method, followed by a brief description of the proposed OpenCL implementation.

This work was supported by NIH grant HL091017. The first author has received financial support through the "la Caixa" Fellowship Grant for Post-Graduate Studies, Caixa d'Estalvis i Pensions de Barcelona "la Caixa", Barcelona, Spain.

More details on the mathematical development of the operator can be found in [4].

2.1. Operator Definition

In SPECT tomographic imaging, the acquisition process is performed by a gamma camera (detector array) on an orbit around patient (it can be circular, elliptical or any other) to acquire multiple 2-dimensional images (projections) from multiple angles. In [1], we show that the CAMM tomographic model can be expressed as:

$$E[\mathbf{g}] = \mathbf{H} \cdot \mathbf{f_m}$$

where $\mathbf{f_m}$ denotes a vector formed from the nodal values of the mesh structure representing the emission image, $E[\cdot]$ is the expectation operator, $\mathbf{g}$ denotes the projections measured by a gamma camera and $\mathbf{H}$ is the projection matrix. Note that this projection matrix depends on the attenuation image, the detector-collimator spatial response and the geometry of the imaging system. The main purpose of this paper is to show a fast method to compute projection (system) matrix.

2.2. Integral decomposition

A single element of $\mathbf{g}$ at for a given detector bin x_i,y_i, with an altitude of z_j and a projection angle of θ_k can be computed by the following integral:

$$E[g_{i,j,k}] = \int_0^\infty f\left(\gamma_{i,j,k}(u)\right) e^{-\int_0^u \mu(\gamma_{i,j,k}(v))\,\mathrm{d}v}\,\mathrm{d}u$$

where $\gamma_{i,j,k}$ is the arc-length parametrization of the casted ray path that starts at bin x_i,y_i with altitude z_j from projection θ_k and goes through and emission volume $f(x,y,z)$ with attenuation map $\mu(x,y,z)$. The use of the CAMM model to represent $f(x,y,z)$ and $\mu(x,y,z)$ allow us to convert the integral above into the integral below:

$$E[g_{i,j,k}] \cong \sum_{\{l|T_l \cap \gamma_{i,j,k} \neq \emptyset\}} \int_{p_{l,i,j,k}}^{q_{l,i,j,k}} f\left(\gamma_{i,j,k}(u)\right) e^{-\int_0^{r_{l,i,j,k}} \mu\left(\gamma_{i,j,k}(v)\right)\,\mathrm{d}v}\,\mathrm{d}u$$

where T_l are the mesh elements intersected by the path of $\gamma_{i,j,k}$, $p_{l,i,j,k}$ is the entry point of the ray $\gamma_{i,j,k}$ into the element T_l, $q_{l,i,j,k}$ is the exit point of the ray $\gamma_{i,j,k}$ from the element T_l, and $r_{l,i,j,k}$ is the middle point of the intersection, see Figure 2.

2.3. Mesh Interpolation

By using linear interpolation and CAMM the value of a function $f(x,y,z)$ inside an element T_l can be interpolated from the values of the nodes of T_l. Essentially, if the nodes of T_l can be denoted by $\mathbf{p}_i$, $i = 1,2,3,4$ and the values of $f(x,y,z)$ on the nodes of T_l are denoted by $\hat{f}_i = f(\mathbf{p}_i)$, $i = 1,2,3,4$, we can approximate the value of $f(x,y,z)$ at

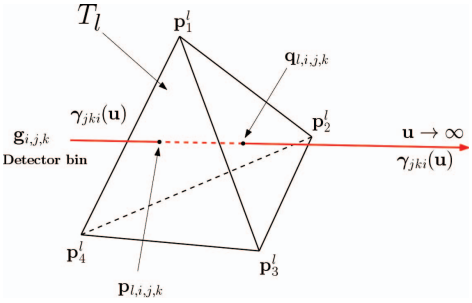


Fig. 2. Intersection of ray $\gamma_{i,j,k}$ with element T_l.

a point $\mathbf{s}$ inside T_l by using its barycentric coordinates with respect to the nodes of T_l:

$$\hat{f}(\mathbf{s}) = [\lambda_1(\mathbf{s}), \lambda_2(\mathbf{s}), \lambda_3(\mathbf{s}), \lambda_4(\mathbf{s})] \cdot \begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \end{bmatrix} = \lambda_l(s)^T \cdot \hat{\mathbf{f}}_l$$

where $\lambda(s)$ satisfies the linear equation $s = \lambda(s)^T \cdot [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]^T$ and $\lambda_1(\mathbf{s}) + \lambda_2(\mathbf{s}) + \lambda_3(\mathbf{s}) + \lambda_4(\mathbf{s}) = 1$.

2.4. Integral approximation

We can use the interpolation explained in Sec 2.3 in order to approximate the integral in Sec 2.2 as:

$$E[g_{i,j,k}] \cong \sum_{\{l|T_l \cap \gamma_{i,j,k} \neq \emptyset\}} \int_{p_{l,i,j,k}}^{q_{l,i,j,k}} \left[\lambda\left(\gamma_{i,j,k}(u)\right)\right]^T \cdot \hat{\mathbf{f}}_l \cdot \exp\left(-\int_0^{r_{l,i,j,k}} \left[\lambda\left(\gamma_{i,j,k}(v)\right)\right]^T \cdot \hat{\mu}_l\,\mathrm{d}v\right)\mathrm{d}u$$

2.5. Detector-collimator spatial response

The detector-collimator spatial response is usually modeled as a Gaussian spatially dependent blur with increasing full width at a half maximum (FWHM) linearly as a function of the distance from the detector surface [13]. This blur model approximates collimator penetration and light diffusion in the scintillator material. We account for this by tracing multiple rays in a cone, denoted by n, from each projection bin $g_{i.j,k}$, where with each ray we associate the probability of accepting a photon from that path, w_n.

Now the n-th casted ray is describe by $\gamma_{i,j,k}^n$, thus, the original formula becomes:

$$E[g_{i,j,k}] \cong \sum_n w_n \sum_{\{l|T_l \cap \gamma_{i,j,k}^n \neq \emptyset\}} \int_{p_{l,i,j,k}}^{q_{l,i,j,k}} \left[\lambda\left(\gamma_{i,j,k}^n(u)\right)\right]^T \cdot \hat{\mathbf{f}}_l \cdot \exp\left(-\int_0^{r_{l,i,j,k}} \left[\lambda\left(\gamma_{i,j,k}^n(v)\right)\right]^T \cdot \hat{\mu}_l\,\mathrm{d}v\right)\mathrm{d}u$$

At this point we are faced with a trade-off between projection operator accuracy and computational complexity, which depends on how many rays are casted. In Figure 3 we present a hexagonal ray casting pattern which, if properly dimensioned, ensures that no element inside the hexagonal cone is missed. The mathematical formulation of this is omitted due to the spatial limitation.

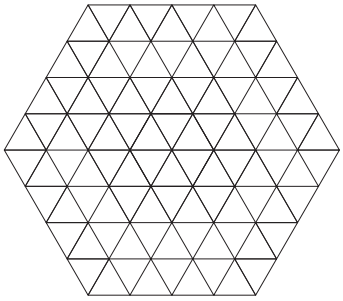


Fig. 3. Ray-Casting pattern which, if properly dimensioned, can assures that no mesh element is missed

3. IMPLEMENTATION

To fully use the parallel computation by OpenCL, the proposed implementation must observe some basic rules. These rules can be summarized as follows: One should not synchronize threads often; write on the same memory location by more then one running thread; have multiple flow paths within a thread; or use too many registers. Therefore, we divided our parallel implementation into four steps:

1. For fixed n, compute the entry point of $\gamma^n_{i,j,k}$ into T_l mesh element (i.e. $p^n_{l,i,j,k}$)
2. For fixed n, compute:

$$w_n \cdot \sum_{\left\{l|T_l \cap \gamma^n_{i,j,k} \neq \emptyset\right\}} \int_{p_{l,i,j,k}}^{q_{l,i,j,k}} \left[\lambda\left(\gamma^n_{i,j,k}(u)\right)\right]^T \cdot \hat{\mathbf{f}}_l \cdot \exp\left(-\int_0^{r_{l,i,j,k}} \left[\lambda\left(\gamma^n_{i,j,k}(v)\right)\right]^T \cdot \hat{\mu}_l \, \mathrm{d}v\right) \mathrm{d}u$$

3. Iterate over n.

Next we will give several remarks on improvement in the overall execution time.

1. Even though matrix $\mathbf{H}$ is highly sparse, allocating a sparse matrix used in parallel processing can diminish overall performance due to the need to synchronize running threads in memory access step. Furthermore allocating the full matrix $\mathbf{H}$ was too memory demanding so we opted for computing and storing one row of at a time.
2. Since each row of $\mathbf{H}$ is uniquely related to the value of $g_{i,j,k}$, running multiple $\gamma^n_{i,j,k}$ with fixed n guarantees that each running thread will be working on a separate memory space without the need for thread synchronization.
3. OpenCL is designed to handle geometric operations efficiently. So, operations as an addition of three-dimensional tuples may be done in just one cycle if the hardware supports it (like on CPUs but not GPUs).

3.1. Computing the entry point

In order to compute the first entry point of $\gamma^n_{i,j,k}$ into the CAMM we first intersect the ray with all the planes defining the boundaries of the CAMM. Then we use the jump-and-walk algorithm [14, 15] to approach that point. If we detect that the point is inside the triangulation, we already have the entry point $p^n_{l,i,j,k}$ and the element T_l; otherwise, if we detect that the point lies outside the triangulation, we finish the thread.

3.2. Computing the integral

As we described in Sec 2.2, the integral can be approximated by summation of a set of integrals over each element. Thus, given $p^n_{l,i,j,k}$ and T_l we only need to find $q^n_{l,i,j,k}$ and then compute the integral described on Sec 2.4 over the element T_l.

In order to compute the entries and exits points of the ray through the element, we use a numerically stable algorithm described in [16].

4. RESULTS

For the testing of our implementation we used fixed geometry as shown in Figure 1, with 10927 nodes and casting 225 rays for each projection bin. We tested the proposed parallel implementation on: 1) Intel(R) Xeon(R) CPU E5520 at 2.27GHz with four quad-cores (16 processors in total); 2) NVIDIA Tesla C2070 with 14 multiprocessors each one at 1.147GHz (448 processors in total), and as a baseline a single-threaded C/C++ implementation (with following optimization flags: `-O3`) running on the Intel(R) Xeon(R) CPU E5520 at 2.27GHz. The execution times for the 16 CPU cores are shown in Table 1 and the execution times for the NVIDIA Tesla GPU are shown in Table 2. In Table 3 we predict the execution time of using four identical GPUs.

The presented results demonstrate the OpenCL implementation's superiority over classical single thread CPU C/C++ implementation. In addition it also demonstrates that using GPU hardware one can achieve the best speedup. Here it is worth noting that the GPU hardware cost is a fraction of that of the CPU hardware.

Table 1. Execution time for OpenCL implementation executed on 16 cores CPU at 2.27GHz.

Number of bins	Number of Projections	OpenCL Execution Time	Single thread Execution Time	Speed Up
64^2	64	1.7 min	37.8 min	22.3
64^2	128	3.4 min	75.6 min	22.3
128^2	64	7.3 min	151.2 min	20.8
128^2	128	14.5 min	302.4 min	20.8

Table 2. Execution times for OpenCL implementation executed on 14 multicores NVIDIA GPU at 1.147GHz.

Number of bins	Number of Projections	OpenCL Execution Time	Single thread Execution Time	Speed Up
64^2	64	1.3 min	37.8 min	28.0
64^2	128	2.7 min	75.6 min	28.0
128^2	64	4.8 min	151.2 min	31.4
128^2	128	9.6 min	302.4 min	31.4

Table 3. Estimated execution times using four GPU cards.

Number of bins	Number of Projections	OpenCL Execution Time	Single thread Execution Time	Speed Up
64^2	64	0.32 min	37.8 min	114.4
64^2	128	0.67 min	75.6 min	112.2
128^2	64	1.2 min	151.2 min	124.6
128^2	128	2.4 min	302.4 min	124.6

5. CONCLUSIONS

In this paper we explored the feasibility of using OpenCL for tomographic reconstruction using content adaptive mesh model (CAMM). We demonstrated a fast mesh model based project operator calculation using the ray casting algorithm. Then, the problem of image reconstruction becomes the estimation of these model parameters. We also demonstrated approximately 22x of speed up using OpenCL and multi-core CPU and 30x of speed up using NVIDIA graphical processing units with the possibility of 120x speed up with use of four GPU cards.

6. REFERENCES

[1] Yongyi Yang, M.N. Wernick, and J.G. Brankov, "A fast algorithm for accurate content-adaptive mesh generation," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, 2001, vol. 3, pp. 868 –871 vol.3.

[2] Jovan G. Brankov, Yongyi Yang, and Miles N. Wernick, "Tomographic image reconstruction based on a content-adaptive mesh model," *IEEE Transactions on Medical Imaging*, vol. 23, no. 2, pp. 202–212, February 2004.

[3] Jovan G. Brankov, Yongyi Yang, and Miles N. Wernick, "Content-adaptive 3D mesh modeling for representation of volumetric images," in *Proc. IEEE International Conference on Image Processing*, 2002, vol. 3, pp. 849–852.

[4] Ricard G. Delgado and Jovan G. Brankov, "Mesh model based projection operator for emission tomography," in *Proc. IEEE Nuclear Science Symposium*, November 2007, vol. 4, pp. 2760–2763, ISSN=1082-3654.

[5] Jovan G. Brankov, Ricard Delgado Gonzalo, Yongyi Yang, Mingwu Jin, and Miles N. Wernick, "Progress in mesh based spatio-temporal reconstruction," in *Computational Imaging VI*, Charles A. Bouman, Eric L. Miller, and Ilya Pollak, Eds. 2008, vol. 6814, p. 68140V, SPIE.

[6] Arkadiusz Sitek, Ronald H. Huesman, and Grant T. Gullberg, "Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud," *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1172–1179, September 2006.

[7] Francesc Massanes and Jovan G. Brankov, "GPU Based Calculation of a SPECT Projection Operator for Content Adaptive Mesh Model," in *Medical Imaging Conference Record, 2011. MIC '11. IEEE*, 26 2007-nov. 3 2011, vol. 6, pp. 4546 –4548.

[8] Khronos Group, "OpenCL www.khronos.org/opencl/," .

[9] Bo Wang, Lei Zhu, Kebin Jia, and Jie Zheng, "Accelerated cone beam ct reconstruction based on opencl," in *Image Analysis and Signal Processing (IASP), 2010 International Conference on*, april 2010, pp. 291 –295.

[10] Wenyu Zhang, Li Zhang, Shangmin Sun, Yuxiang Xing, Yajie Wang, and Juan Zheng, "A preliminary study of OpenCL for accelerating CT reconstruction and image recognition," in *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE*, 2009, pp. 4059 –4063.

[11] Christian Siegl, H. G. Hofmann, B. Keck, M. Prümmer, and J. Hornegger, "OpenCL: a viable solution for high-performance medical image reconstruction?," in *Proc. SPIE 7961*, 2011.

[12] William Paul Segars, *Development and Application of the New Dynamic NURBS-based Cardiac-torso (NCAT) Phantom*, Ph.D. thesis, University of North Carolina at Chapel Hill, 2001.

[13] Miles N. Wernick and John N. Aarsvold, *Emission Tomography*, Academic Press, Boston, MA, 2004, isbn = 0127444823.

[14] Luc Devroye, Ernst P. Mucke, and Binhai Zhu, "A note on point location in Delaunay triangulations of random points," *Algorithmica*, vol. 22, pp. 477–482, 1998.

[15] Ernst P. Mucke, Isaac Saias, and Binhai Zhu, "Fast randomized point location without preprocessing in two and three-dimensional Delaunay triangulations," in *Proc. 12th ACM Symposium on Computational Geometry*, 1996, pp. 274–283.

[16] Christer Ericson, *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*, Morgan Kaufmann, 2005.