Strategies for Accelerating Forward and Backprojection in List-mode OSEM PET Reconstruction using GPUs

William Dieckmann, Shanthalaxmi Thada, and W. Craig Barker

Abstract–**Image reconstruction for the ECAT HRRT PET scanner with MOLAR is computationally demanding and requires a computer cluster for reasonable run times. Parallel computing using GPUs and CUDA offers a means to accelerate MOLAR. However, forward and backprojection operations present unique challenges that must be overcome to achieve acceptable speedup. In this study we implement GPU-accelerated versions of MOLAR's forward projection, backprojection and algorithm update modules and compare their performance to CPU-only versions. During this implementation we optimized the GPU thread configurations for each of these modules separately, along with a hybrid forward-backprojection module that is used for algorithm updates. We also numerically evaluated the reconstruction results to assess the impact of floating-point to integer conversions dictated by the GPU architecture. We found forward projection to be 41 times faster than the CPU-only code, while backprojection was 20 times faster. We found the optimal thread configurations always assigned 64 threads to a thread block, but with different distributions across the nested indexing loops within each module. These results show that MOLAR's forward and backprojection modules can be adequately accelerated to make the MOLAR reconstruction package much more efficient.**

I. Introduction

MOLAR (**M**otion-compensation **O**SEM **L**ist-mode **A**lgorithm for **R**esolution-recovery reconstruction [1,2]) was developed to extract high-resolution images from list-mode Siemens ECAT HRRT (High Resolution Research Tomograph, Siemens Medical Solutions, Knoxville, TN, USA) PET data. Due to computational demands, seven dual-processor clustered computers are used to generate a typical image volume in about three hours. Speeding up the reconstruction process would bring many benefits. Images would be available sooner after acquisition, particularly for dynamic studies; improvements in image quality through the use of finer sampling grids than are currently practical would become more accessible; and the more computationally intensive 4D parametric modeling implementation of MOLAR [3] would become more usable. This paper investigates the challenges related specifically to accelerating the forward projection and backprojection operations that represent a large part of computation in MOLAR. For an overview of accelerating the complete MOLAR package using GPUs, see [4].

Manuscript received October 21, 2009. This work was supported by the Intramural Research Program of the NIH, CC.

W. Dieckmann is with the PET Department, NIH Clinical Center, National Institutes Health, Bethesda, MD 20892 USA (e-mail: wdieckmann @cc.nih.gov).

S. Thada is with the PET Department, NIH Clinical Center, National Institutes Health, Bethesda, MD 20892 USA (e-mail: sthada @cc.nih.gov).

W. C. Barker is with the PET Department, NIH Clinical Center, National Institutes Health, Bethesda, MD 20892 USA (telephone: 301-451-3558, e-mail: cbarker @nih.gov).

II. Methods

A. Hardware and Programming Environment

We used a Hewlett-Packard xw8400 workstation having two dual-core Intel Xeon 5130 CPUs running at 2.0 GHz with 12 GB of RAM. The workstation was running Red Hat Linux WS 5 (64-bit). For acceleration we chose an NVIDIA Tesla S1070-400 system with four GPUs, each having four GB of memory.

MOLAR's C++ code was compiled for the workstation CPU using the GNU g++ compiler (version 4.1.2). GPU code was written using NVIDIA's CUDA™ SDK/compiler, version 2.2 (NVIDIA Corporation, Santa Clara, CA). Although the workstation has multiple CPU cores and the Tesla S1070 has four independent GPUs that can be used in parallel, we used only one core with one GPU to evaluate relative performance.

The CUDA programming environment facilitates the concurrent execution of multiple instances of code in "threads" that run on a GPU. These threads are organized into "blocks," which are invoked by a "grid" (Fig. 1). This can result in thousands of concurrent instances of code operating on independent data elements.

B. Computation and Test Data

CUDA kernels (GPU-based programs) were written to replace the C++ modules in MOLAR that perform coincidence event forward projection, backprojection, and the combined forward-backprojection in the OSEM algorithm update.

To evaluate relative performance, the GPU-accelerated and CPU-only versions of the MOLAR code were run on phantom data acquired on our HRRT PET scanner. A cylindrical phantom containing 1.3 mCi of ^{68}Ge was scanned for five minutes, yielding approximately 164M events. The image volume dimensions were 256 x 256 x 207. Once the kernel configurations were optimized, we evaluated performance by separately reconstructing five million events using the GPU-accelerated code and the CPU-only code.

To assess numerical accuracy and precision, we also chose two clinical brain scans and reconstructed them using the

CPU-only and GPU-accelerated versions of MOLAR. The first scan had ^{18}F-fluorodopa as the tracer and represents a spatially varying activity distribution, while the other scan utilized ^{11}C-leucine, giving a more uniform distribution. From the fluorodopa scan, emission images for frame durations of six, 15, 60, and 90 seconds were created. Frame durations for the leucine scan were all 90 seconds, but five different frame start times were chosen so that tracer kinetics and physical decay caused the number of events incorporated per frame to vary widely. Reconstructed images were then subtracted voxel-by-voxel for direct comparison.

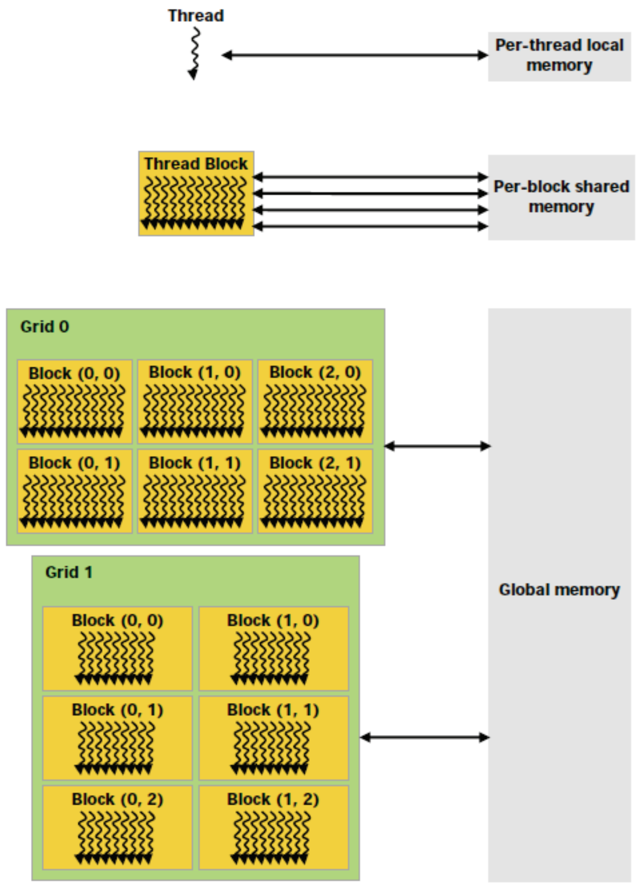

Fig. 1. GPU threads, blocks and grids and the memory hierarchy associated with them [5].

C. *Kernel Strategies*

Projection of coincidence events consists largely of tracing rays through an image volume along an event's line-of-response (LOR). Image voxels in the neighborhood of the LOR, as defined by the scanner's spatial resolution function, are included in the computations that determine their relative contribution to the projection value. For the datasets used in this study, a 7 x 7 voxel neighborhood orthogonal to the LOR's primary axis was used.

Each coincidence event typically requires access to thousands of image voxels for each forward projection, so memory transaction efficiency is essential to good performance on the GPU. We sought to maximize this efficiency in three ways: a) the storage and retrieval of intermediate values were replaced with on-the-fly calculation of the values as they are needed, b) read-only variables, such as voxel weight functions, were stored in cacheable memory on the GPU, and c) kernel configurations were optimized by varying the total number of threads per block, the number of threads assigned to voxel indexing, and the total number of thread blocks in each grid. The goal of this optimization was to find the configuration that most effectively balanced memory access with computation.

D. *Forward Projection*

Because forward projection only requires read access to an image, image volumes were stored in 3D CUDA arrays, which provide cached read-only access to the GPU device memory. Low-latency, on-chip, shared memory available to each thread block was used to accumulate intermediate results across threads. For kernel design, we tried two separate approches: one in which a single block handles multiple coincidence events, and another that dedicates a thread block to a single event. The single-event kernel gave us more flexibility in the assignment of threads to the image array indexing loops.

E. *Backprojection*

Image voxels are modified during backprojection, so we could not use the GPU's read-only array cache for efficiency gains. Instead, we made use of CUDA's memory transaction coalescing by replicating the image volume in two versions, each having the image voxels arranged in different order. One version was x-dominant, the other y-dominant. The version used by any given coincidence event was determined by the angle of the event's LOR to the scanner's transaxial plane. After all the events are projected, the two image volumes are summed to obtain the complete backprojection result. This approach is similar to that used in the original MOLAR code.

With many parallel threads simultaneously updating an image volume, it was necessary to use the atomicAdd() function to ensure that all events contributed to the result without conflicting. As an aside, because the CUDA atomicAdd() function only operates on unsigned integers, we had to scale and truncate the floating-point image values to 32-bit integers. In doing so, we had to be careful to choose scale factors that would preserve as much precision as possible while avoiding integer overflow.

F. *OSEM Algorithm Update*

To achieve computational efficiency, the original implementation of MOLAR's OSEM algorithm update module combines, event-by-event, a forward projection with a subsequent backprojection. We initially created an equivalent GPU kernel to perform this combined forward-backprojection in addition to the stand-alone forward and backprojection modules described above. Ultimately, however, we separated these operations into two separate kernels to allow independent performance tuning of the forward and backward parts. As before, GPU caching of 3D CUDA arrays were used for forward projection, and two image arrays were used for backprojection.

III. Results

A. Performance Comparison

Our performance evaluations found that the GPU-accelerated code for forward projection using the single event per block kernel was 41 times faster than the CPU-only code (Table I). The multiple-event per block kernel was better, running 50 times faster, but it could not be paired with a fully tunable version of backprojection in the same kernel.

Backprojection using the GPU was 20 times faster than the CPU, limited by the lack of caching during image array writes. The OSEM algorithm updates in which forward and backprojection were combined in a single kernel ran nine times faster on the GPU. When forward and backprojection within the algorithm updates were in separate kernels, performance improved to 11 times faster.

Tuning of the various kernel block configurations varied in effectiveness. Forward projection showed the largest changes in performance, with run-times differing by almost a factor of three (Table II). The best performing configuration is shown in bold type. The three values designated “thread distribution” represent the number of threads assigned to three nested code loops that process the voxels within an LOR’s neighborhood.

Backprojection run-times varied by about 40% across the different block configurations (Table III), and forward-backprojection changed by a factor of two (Table IV). Kernel grid size (number of thread blocks activated per kernel invocation) had a negligible effect any of the observed run-times.

B. Numerical Accuracy

Subtracting images from the CPU-only reconstructions from those obtained with the GPU-accelerated code revealed minor numerical issues. Fig. 2 (top row) shows a central slice from the fluorodopa (left column) and leucine (right column) reconstructions along with outlines of the regions-of-interest (ROI) used for numerical comparison. The middle row shows difference images (GPU minus CPU) and the bottom row shows percent differences. The fluorodopa reconstruction included 16M events, while the leucine run had 43M events.

For the fluorodopa and leucine central slices, the mean values were 8,427 Bq/ml and 28,783 Bq/ml, respectively. Approximate measurement noise, as estimated by the voxel standard deviation within the ROIs, were 52% and 41%. The maximum absolute differences between the GPU and CPU images were 194 and 529, which is 4.4 and 4.5% of the approximate measurement noise.

The mean ROI values for the difference images were -0.07 and -14.45, or -0.0008% and 0.05% of the mean values. Standard deviations of the difference regions were 15.8 and 51.8, which is 0.36 and 0.44% of the measurement noise.

Subtle artifact structures in the difference images are clearly visible in the bottom row of Fig. 2. We suspect that the primarily diagonal features are caused by floating-point rounding differences in the voxel indexing code, which we expect to be able to correct after further investigation. There are also other small round-off differences, similar in magnitude to differences arising from the floating-point to integer conversions used in our GPU-based backprojection module.

Table I. Event processing rates for 5M events

Module	Events/sec CPU-only	Events/sec GPU-accelerated
Forward Proj. (single-event/block)	9,770	396,000
Backprojection	8,561	167,001
OSEM Fwd-Back (1 kernel)	4,550	39,062
OSEM Fwd-Back (2 kernels)	4,550	48,543

Table II. Tuning the forward projection kernel, 1M events

Threads per Block	Thread Distribution	Relative Run-time
256	8 / 8 / 4	2.79
128	8 / 8 / 2	2.37
64	8 / 8 / 1	1.91
128	4 / 4 / 8	1.44
32	2 / 2 / 8	1.52
64	**2 / 2 / 16**	**1.00**
128	2 / 2 / 32	1.30

Table III. Tuning the backprojection kernel, 200K events

Threads per Block	Thread Distribution	Relative Run-time
256	8 / 8 / 4	1.09
128	8 / 8 / 2	1.04
64	**8 / 8 / 1**	**1.00**
128	16 / 4 / 2	1.31
128	4 / 16 / 2	1.37

Table IV. Tuning the forward/backprojection kernel, 5M events

Threads per Block	Thread Distribution	Relative Run-time
128	8 / 8 / 2	1.08
64	8 / 8 / 1	1.09
128	4 / 4 / 8	1.06
64	**4 / 4 / 4**	**1.00**
64	2 / 2 / 16	1.47
8	2 / 2 / 2	1.98

IV. Discussion

It is difficult to know precisely why particular thread distributions perform best for a given kernel. There appear to be many factors. Memory access timing varies greatly for forward projection and backprojection due to the presence of low-latency cache for data reading. Each module performs different calculations between ray-tracing operations. Cached image access is slightly more efficient for one dominant axis compared to the other. In short, aside from the observation that the optimal configurations all had 64 threads in a block, it appears that finding those optimal configurations will continue to be an experimental exercise.

Re-optimizing configurations is likely to useful if the number of neighboring voxels associated with an LOR is changed, or even for seemingly minor kernel changes. Different versions of GPUs also have different computational and memory I/O characteristics, and therefore should benefit from customized optimization.

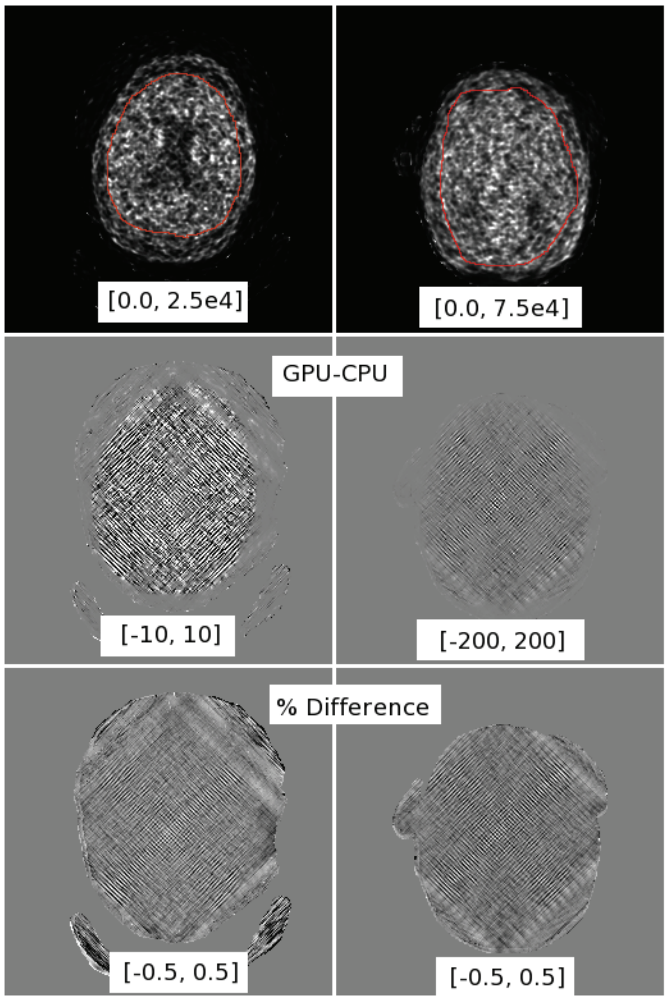

Fig. 2. Sample images obtained from GPU-accelerated code compared with CPU-only results. The left column is from a fluorodopa study, the right column is from a leucine study. Top row: central slices, with the outlines of the regions-of-interest used. Middle row: GPU-CPU difference images. Bottom row: percent difference images. The numbers indicate the minimum and maximum grayscale voxel values.

V. Conclusion

Forward projection and backprojection in OSEM list-mode PET reconstruction can be substantially accelerated when assisted by a GPU co-processor. Tuning of CUDA thread block dimensions to alter global memory access patterns yields worthwhile benefits.

References

[1] R. E. Carson, W. C. Barker, J-S. Liow, and C. A. Johnson, “Design of a motion-compensation OSEM list-mode algorithm for resolution-recovery reconstruction for the HRRT,” *IEEE Nuclear Science Symposium and Medical Imaging Conference*, Portland, 2003.

[2] C. A. Johnson, S. Thada, M. Rodriquez, Y. Zhao, A. R. Iano-Fletcher, J-S. Liow, W. C. Barker, R. L. Martino, and R. E. Carson, “Software architecture of the MOLAR-HRRT reconstruction engine,” *IEEE Nuclear Science Symposium and Medical Imaging Conference*, Rome, 2004.

[3] J. Yan, B. Planeta-Wilson, J-D. Gallezot, R. E. Carson, “Initial evaluation of direct 4D parametric reconstruction with human PET data,” *IEEE Nuclear Science Symposium and Medical Imaging Conference*, Orlando, 2009

[4] W. C. Barker, S. Thada, and W. Dieckmann, "A GPU-accelerated implementation of the MOLAR PET reconstruction package," *Workshop on High Performance Medical Imaging IEEE Nuclear Science Symposium and Medical Imaging Conference,* Orlando, 2009.

[5] CUDA Programming Guide Version 2.3, NVIDIA Corporation, Santa Clara, CA.