A GPU-Accelerated Implementation of the MOLAR PET Reconstruction Package

W. Craig Barker, Shanthalaxmi Thada, and William Dieckmann

Abstract–**MOLAR (Motion-compensation OSEM List-mode Algorithm for Resolution-recovery reconstruction) was written to provide the best possible images from ECAT HRRT PET data. Because of computational demands, MOLAR currently requires a computer cluster for practical use. Here we have applied GPU-acceleration via CUDA to all of the computationally intensive modules of the MOLAR package. Using an NVIDIA Tesla S1070-400 GPU system hosted by an HP xw8400 workstation, we evaluated the GPU-accelerated performance of the modules that perform boundary checking, forward and backprojection, photon scatter modeling and algorithm updates. We compared their performance to CPU-only versions of MOLAR for a range of total counts (500k to 50M). We found boundary checking to be up to 35 times faster using the GPU. Forward and backprojection ran 50 and 20 times faster, respectively, and scatter modeling was 200 times faster. Algorithm updates ran up to 15 times faster. The overall performance of the entire MOLAR package was approximately 40 times faster than the CPU-only code. These results show that MOLAR can be substantially accelerated using GPUs and can thereby be practically extended for use in high count and higher resolution applications, and for 4D parametric reconstructions.**

I. Introduction

MOLAR (Motion-compensation OSEM List-mode Algorithm for Resolution-recovery reconstruction [1,2]) was originally written for the ECAT HRRT (High Resolution Research Tomograph, Siemens Medical Solutions, Knoxville, TN, USA) to obtain high-resolution, motion-corrected images. The code is nevertheless flexible enough to be adapted to any scanner. Unfortunately, it has substantial computational demands and currently requires a computer cluster to obtain reasonable reconstruction times. More rapid reconstruction is desirable and would make high-count frames and 4D parametric reconstruction [3] more accessible.

Many groups have tackled the problem of accelerating image reconstruction using a variety of technologies (GPUs, FPGAs, Cell-based systems, etc.) and a variety of languages (Cg, OpenGL, CUDA, etc.) [4-7]. We previously began accelerating parts of MOLAR using CUDA and GPUs, in particular forward projection and component-based normalization factor calculation [8]. In this study we have extended that work to all of the computationally intensive components of MOLAR and have evaluated the performance of the complete package.

Manuscript received October 20, 2009. This work was supported by the Intramural Research Program of the NIH, CC.

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).

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

W. Dieckmann is with the PET Department, NIH Clinical Center, National Institutes Health, Bethesda, MD 20892 USA (e-mail: wdieckmann @cc.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 having four GPUs, each with four GB of memory.

Software for the CPU code was written in C++ and compiled using the GNU g++ compiler (version 4.1.2). The GPU components were written using the CUDA™ SDK/compiler, version 2.2 (NVIDIA Corporation, Santa Clara, CA). We compared execution using a single CPU core to a single GPU for performance evaluation. Reconstructions were performed using two iterations and 30 subsets, which is our standard clinical practice.

The CUDA programming environment enables the execution of parallelized code in "threads" that run on a GPU. Threads are grouped in "blocks," which are activated by the invocation of a "grid," thereby providing thousands of concurrent instances of the code. Fig. 1 shows the GPU thread structure and the associated memory hierarchy. The actual number of available threads is dependent on the specifications of the particular GPU being used.

B. MOLAR Components and Test Data

Our current CPU-only cluster-based implementation of MOLAR relies on the distribution of computational tasks across several commodity PCs that communicate via the Message Passing Interface (MPI). This implementation typically requires seven dual-processor nodes for a single frame reconstruction that produces results in a few hours.

The major contributors to computation time in the CPU-only MOLAR code are the modules that perform boundary checking (identifying the spatial boundaries of the object by ray tracing), forward projection and backprojection of PET coincidence events, photon scatter modeling, and algorithm update calculations. For GPU acceleration, CUDA kernels (GPU-based programs) were written to replace these C++ modules while the predominantly serial functions of MOLAR remained in C++ code running on the CPU.

The GPU-accelerated and CPU-only versions of the MOLAR code were run separately on phantom data acquired on our HRRT scanner. A cylindrical phantom containing 1.3

mCi of ^{68}Ge was scanned for five minutes, acquiring approximately 164M counts. Reconstructions were performed for 500k to 50M total coincidence events. Compute times for each of the major components were isolated, event processing rates were computed, and relative speedup was determined. For numerical accuracy evaluation under clinical imaging conditions, we also reconstructed patient data obtained after a 7.4 mCi injection of ^{18}F-fluorodopa. The reconstructed frame was 90-second duration, with 16M total counts.

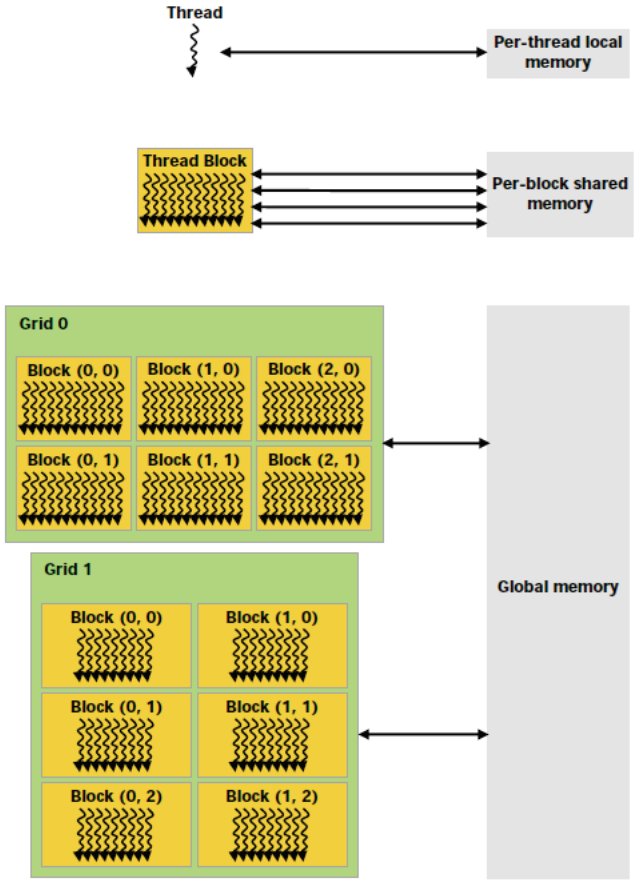

Fig. 1. The relationship between GPU threads, blocks and grids and the associated memory hierarchy [9].

C. Boundary Checking

Boundary checking (Fig. 2) is a one-time operation that is performed for two reasons: to exclude coincidence events that are detected by the scanner but do not contribute to the volume of interest being reconstructed, and to define the range of voxels along an event's line-of-response (LOR) that contribute to the reconstruction volume of interest. Both purposes serve to minimize the amount of computation required for the reconstruction and were originally implemented as a time-saving strategy for the cluster implementation of MOLAR.

To accomplish boundary checking, each event LOR is ray-traced from one side of the image matrix to the other. The first voxel that intercepts the attenuation map, allowing for a pre-defined margin outside the map, defines the starting position for forward projection. Conversely, the last voxel intercepting the attenuation map defines the end position. Determining this range of voxels along an LOR can typically reduce the relevant portion of the LOR to roughly half its original length.

Boundary checking is accelerated using the GPU by assigning the threads within each 16 x 8 thread block to a single coincidence event. The event LOR is divided among the threads for simultaneous ray tracing. The end points are stored in fast, on-chip, shared memory within the GPU for rapid inter-thread communication. Once the entire LOR has been traced, the boundary points are determined from the intermediate results stored in shared memory. All coincidence events are streamed through the GPU by invoking 5000 blocks (events) per grid.

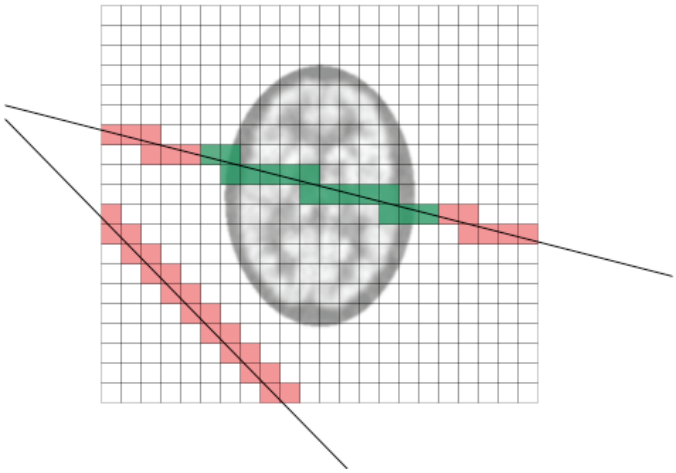

Fig. 2. Boundary checking determines which voxels to include in forward projection (green) and which voxels, or coincidence events, to exclude (red). Voxels shown are not to scale.

D. Forward Projection and Backprojection

Forward projection involves every voxel that lies in the neighborhood of an event LOR, as defined by the scanner's resolution function. It is done for the attenuation map to determine the overall impact of attenuation on a given LOR, and iteratively for the activity distribution to determine the estimated count rate observed by the detector pair associated with that LOR. Backprojection, on the other hand, generates an activity distribution image according to the current iterate of the estimated detector count rates.

Both forward and backprojection operations involve frequent reads of the activity distribution image array. Each read suffers from memory access latency that makes these reads much slower than computational instructions. Fortunately forward projection operations can be organized by voxel neighborhoods, and thereby make use of fast, cached memory accesses. Backprojection, however, is also unable to take advantage of caching, which is a read-only option. It also requires atomic writes to avoid consistency conflicts between concurrent threads. The respective acceleration strategies for forward and backprojection therefore necessitate different approaches. A detailed discussion of the challenges and

strategies involved in accelerating these modules is discussed in [10].

The algorithm update portion of MOLAR is a hybrid module in that it involves both forward and backprojection operations, along with the OSEM update factor calculations that are only performed on the CPU. Because this mixed CPU-GPU implementation has a substantial overall impact on MOLAR, the performance of this module was evaluated separately.

E. Scatter Modeling

A 3D version of the single-scatter simulation (SSS) model [11] was implemented in MOLAR for scatter correction. The implementation computes the scatter contribution for "scatter points" that are distributed throughout the subject volume. For each scatter point, MOLAR computes the scatter observed by virtual "detectors" that mimic real scanner detectors but are positioned in a cylindrical geometry with coarser spatial sampling. Fig. 3 shows the spatial distribution of the virtual detectors and the scatter points for a cylinder phantom. Coarse sampling is acceptable because scatter varies slowly spatially and can therefore be readily interpolated. Even so, there are thousands of scatter point-virtual detector combinations to compute. For each combination, the contribution of all image voxels in the neighborhood of an event's LOR must be computed. Because these computations are independent, this task is readily parallelized.

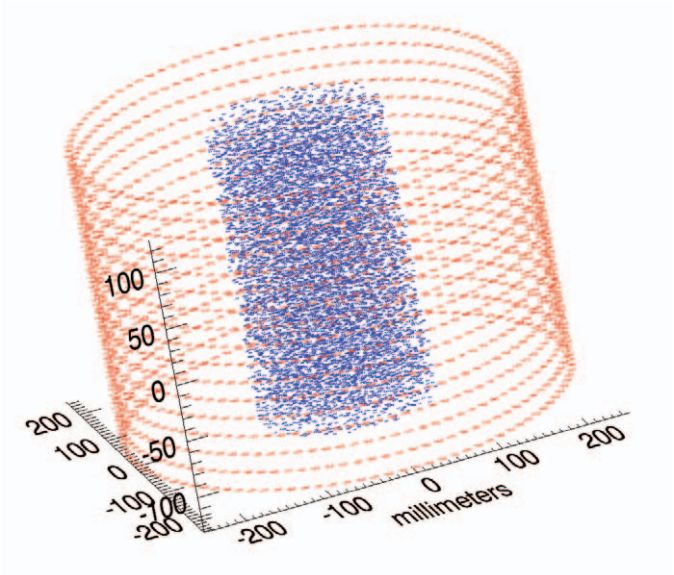

Fig. 3. The distribution of "virtual detectors" (red) in a cylindrical arrangement and "scatter points" (blue) randomly distributed throughout the object being imaged, in this case a cylindrical phantom.

There are two main tasks in the scatter model that are computationally intensive: ray summing and scatter calculation. Ray sums are computed along lines that connect every scatter point-virtual detector pair for both the attenuation map and for the estimated activity distribution. This is similar to event forward projection except one end point lies within the subject volume.

Our GPU-accelerated ray sums are parallelized using a single grid invocation that encompasses all scatter point-virtual detector combinations. The attenuation map and activity distribution are stored in 3D read-only, cached arrays that are efficiently accessed. A single thread block is assigned to each scatter point with the individual threads of that block each assigned to a virtual detector. All threads execute the same code, but some threads complete sooner than others due to differences in the distances the ray sums must traverse in their respective paths. The threads therefore do not suffer from code divergence, which degrades overall performance, so that fast performance is maintained.

Scatter calculation is performed for every possible virtual detector pair (e.g., detectorA and detectorB), mimicking the physical process of PET coincidence detection. For each detector pair, the contribution from all the scatter points are combined to get an estimate of the total scatter observed by that pair of detectors. For the HRRT, the virtual detectors are located at and indexed by 17 axial positions and 100 azimuthal angles. To parallelize scatter calculation, the GPU-accelerated code invokes a grid for one of the virtual detector's (detectorA) angle. Blocks, which can have two indices, are defined by detectorA's axial position and by detectorB's angle. Threads are then assigned by detectorB's axial position so that 16 scatter points can be computed concurrently. This arrangement results in 272 (16 x 17) active threads per block, 1700 blocks, and 100 grids.

F. Numerical Accuracy

Images generated by the CPU-only and GPU-accelerated code were compared for numerical accuracy by direct subtraction of the final reconstructed images. Maximum absolute differences and mean differences within a large region of interest (ROI) were obtained from a single central slice and from a summed image (slices 50-150). The images were also evaluated visually for artifacts.

G. Overall Package Performance

Lastly, we evaluated the net event processing rates for the complete GPU-accelerated MOLAR package.

III. Results

A. Boundary Checking

As seen in Fig. 4, GPU-accelerated boundary checking was nine to 35 times faster than for the CPU-only code. The best performance occurred when large numbers of counts were processed, overcoming the startup costs inherent in initiating the GPU kernels. The GPU processing rate levels out at more than 1M events/sec for 10M total counts.

B. Forward Projection and Backprojection

Forward projection on the GPU (Fig. 5) was found to be 38 to 50 times faster over the range of count totals used. Indeed, performance gains were fairly consistent over the entire range of number of events evaluated.

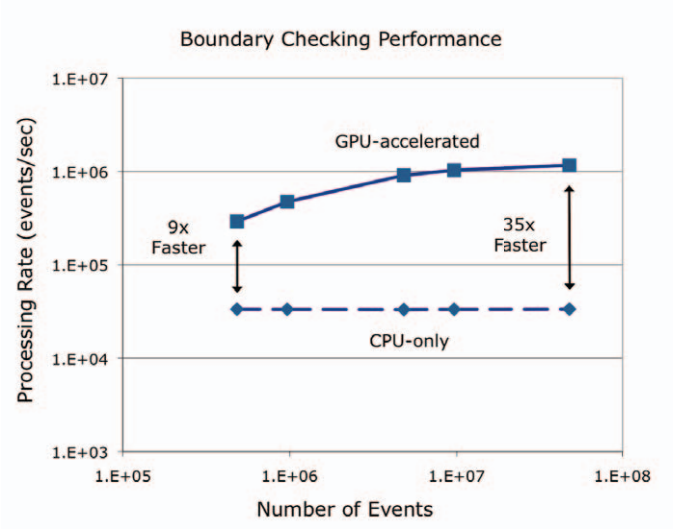

Fig. 4. Boundary checking performance as a function of total events processed.

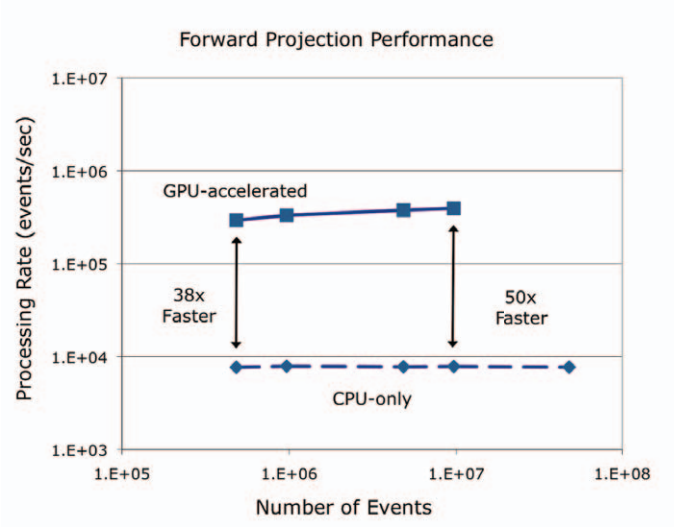

Fig. 5. Forward projection performance as a function of total events processed.

Backprojection, on the other hand, was only 16 to 20 times faster (Fig. 6). This lesser performance gain was not unexpected since backprojection requires writing to GPU memory, and array caching is a read-only feature of our GPU.

The algorithm update portion of MOLAR includes forward and backprojection operations that accompany computation of the OSEM update factors. We saw very little speedup from the GPU-accelerated code when a low number of events are processed (Fig. 7). This is largely due to the fact that the update calculations are done on the CPU and dominate low event count processing. For high-count reconstructions, performance improves as backprojection becomes more dominant. The event processing rate increases to almost 100k/sec for 20M counts, without reaching a plateau. This trend suggests that we will see better performance for higher event totals.

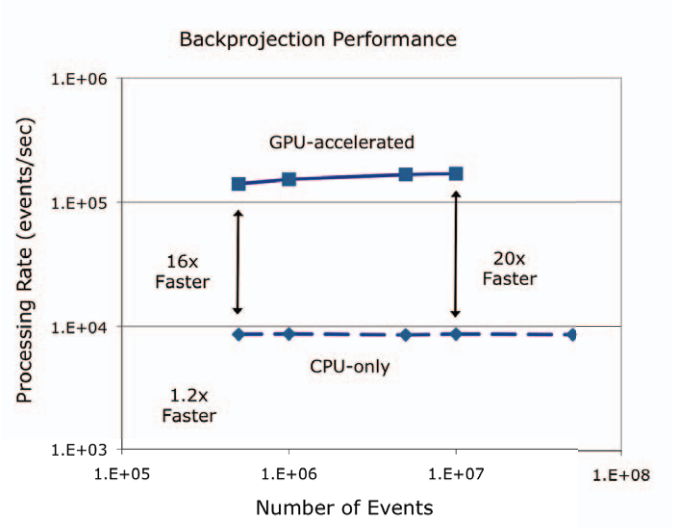

Fig. 6. Backprojection performance as a function of total events processed.

C. *Scatter Modeling*

Scatter modeling showed the most dramatic performance boost with the GPU-accelerated code being 200 times faster than the CPU-only code. Here event count totals have no impact because scatter modeling is a geometrical calculation driven by the image array dimensions, and by the number of virtual detectors and scatter points. Because modeling has much more computation relative to memory access, it is highly suited to GPU acceleration.

D. *Numerical Accuracy*

Difference images computed from the CPU-only and GPU-accelerated reconstruction results demonstrated minor quantitative concerns. Fig. 8 displays a central slice and a summed image (slices 50:150), along with the ROI used for numerical comparison.

For the central slice, the maximum absolute relative difference observed within the ROI was 2.3%, well below the image standard deviation (approximate measurement noise) of

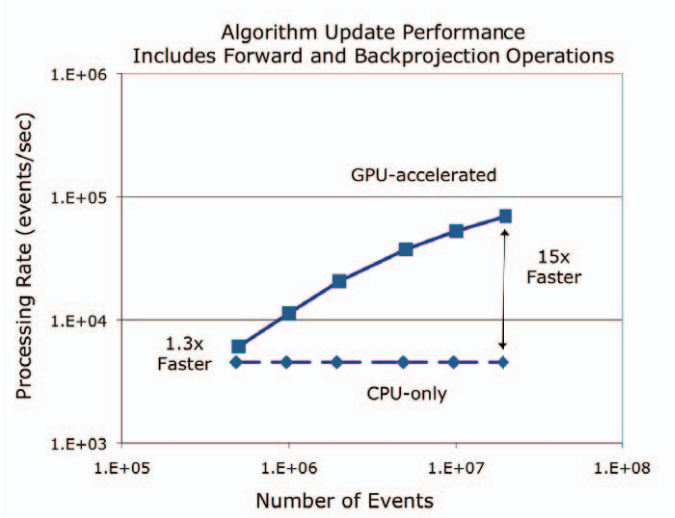

Fig. 7. Performance of the OSEM algorithm update module as a function of total events processed. This module has forward and backprojection operations.

50%. The summed image gave 0.5%, less than the image standard deviation of 12%. This equates to numerical inaccuracies that are approximately 4% of the image measurement noise.

The difference images (bottom row, Fig. 8) reveal subtle artifacts that are we believe are due to array indexing and integer rounding discrepancies. The magnitude of these artifacts (inaccuracies) are well below the statistical measurement noise inherent in these data, and can be ignored in many cases. Nevertheless, it is better that they are resolved, and we anticipate that these discrepancies can be corrected upon further investigation.

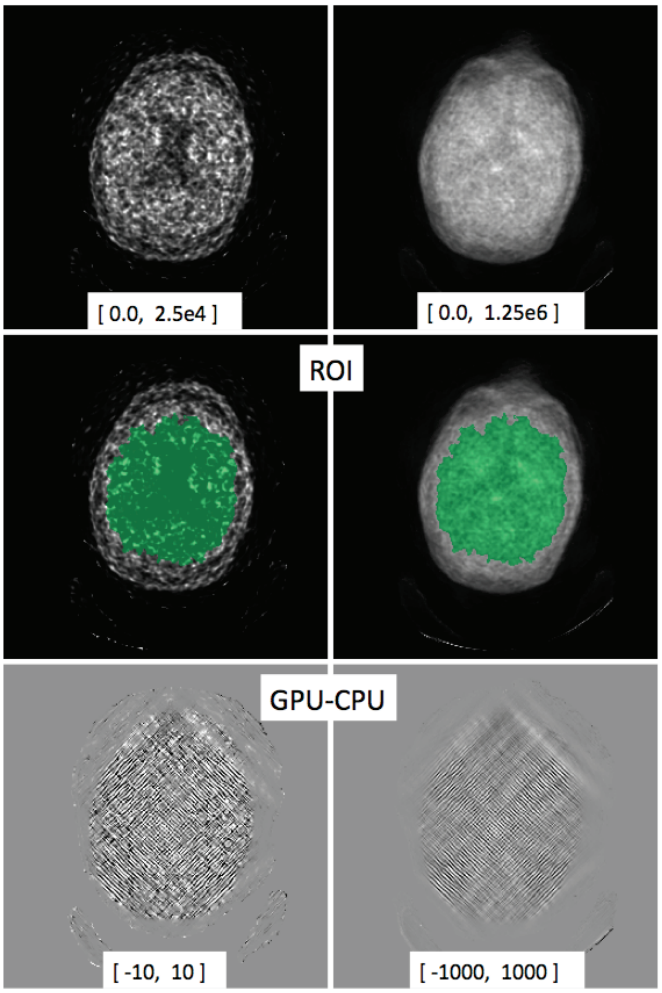

Fig. 8. Numerical evaluation of GPU-accelerated code compared to CPU-only code. A central slice (left column) and the sum of slices 50 to 150 (right column) are displayed: the activity distribution (top row), along with the region of interest (middle row), and the GPU-CPU difference image (bottom row). The numbers in brackets indicate the minimum and maximum grayscale voxel values.

E. Overall Performance

We observed an overall acceleration of the entire MOLAR package that was 31 to 49 times faster than the CPU-only code (Fig. 9). It is notable that within the range of events evaluated here, the slope of the event processing rate curve is positive. This trend suggests that overall performance will continue to improve for 100M or more total events.

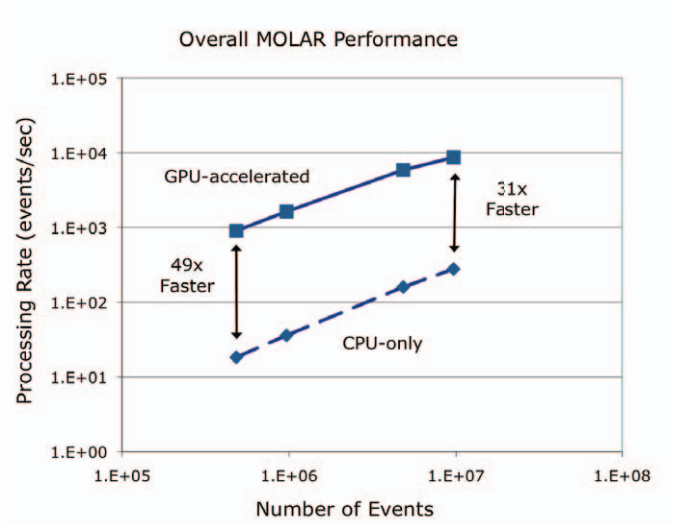

Fig. 9. Overall performance of MOLAR as a function of total events processed.

IV. Discussion

In general, GPU-accelerated code demonstrated substantial performance improvement over the CPU-only code. Best performance was typically seen for count totals above 10M.

Because of the performance gains achieved here, the relative time requirements of other portions of MOLAR have become more pronounced. In particular, disk I/O now occupies a greater fraction of overall run-time, particularly when large dynamic datasets are involved, or when disk caching is needed. In the reconstructions performed here, we found that disk I/O accounted for 1% of CPU-only reconstruction time, but 30% of the GPU-assisted reconstruction time. These performance bottlenecks can be alleviated by using newer, faster disk drives, perhaps even solid state drives. We also anticipate that the recently released Intel Nehalem architecture will provide much better CPU memory performance for the CPU-based components of MOLAR that are not readily parallelized, particularly those that have high memory requirements. These factors partially explain why we observed a speedup factor of only 31 for 10M events as compared to 49 for 500k events in Fig. 9.

One of our other goals is to run MOLAR in a multi-core, multi-GPU environment to see if a large reconstruction job can be adequately achieved within a single server. For example, an 8-core server with 64 GB of RAM coupled to 8 GPUs may make it possible to run a 4D reconstruction job entirely within a single server, eliminating inter-computer communication limitations.

V. Conclusion

This study has demonstrated the benefit of using GPUs to accelerate MOLAR. We anticipate additional improvements as we move toward better data I/O support. This acceleration positions us well for rapid reconstruction of high count datasets, higher resolution finely-sampled images, and perhaps even 4D parametric reconstructions.

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] G. Pratx, G. Chinn, F. Habte, P. Olcott, and C. Levin, "Fully 3-D list-mode OSEM accelerated by graphics processing units," *IEEE Nuclear Science Symposium and Medical Imaging Conference*, San Diego, 2006.

[5] 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.

[6] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 3, pp. 654-663, Jun. 2005.

[7] O. Bockenbach, S. Schuberth, M. Knaup, and M. Kachelrieß, "High performance 3D image reconstruction platforms; state of the art, implications and compromises," *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, Lindau, 2007.

[8] W. C. Barker and S. Thada, "GPU acceleration of MOLAR for HRRT list-mode OSEM reconstructions," *IEEE Nuclear Science Symposium and Medical Imaging Conference*, Honolulu, 2007.

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

[10] W. Dieckmann, S. Thada, and W. C. Barker, "Strategies for accelerating forward and backprojection in list-mode OSEM PET reconstruction using GPUs," *Workshop on High Performance Medical Imaging IEEE Nuclear Science Symposium and Medical Imaging Conference,* Orlando, 2009.

[11] C. C. Watson, "New, faster, image-based scatter correction for 3D PET," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 4, pp. 1587-1594, Aug. 2000.