Adding support for GPUs to PVTOL: The Parallel Vector Tile Optimizing Library

James Brock (brock.james@gmail.com)*, Miriam Leeser (mel@ece.neu.edu)*, Mark Niedre (mniedre@ece.neu.edu)*
*Department of Electrical and Computer Engineering
Northeastern University, Boston, MA

I. Introduction

Architecturally diverse systems comprised of any variety and number of processing elements (multicore processors, FPGAs, GPUs, etc.) have become common in high performance computing. Each of these architectures presents its own programming challenges and complexities. Each often has its own programming language, development environment, and processing constraints. MIT Lincoln Laboratory developed the Parallel Vector Tile Optimizing Library (PVTOL) as a means of writing high-performance signal and image processing code that is portable across a large number of multicore general purpose computing architectures [1]. This work extends the PVTOL Tasks and Conduits framework to include support for Graphics Processing Units (GPUs), alleviating the difficulty of interfacing with different GPU architectures and easing the creation of portable applications. The PVTOL Tasks and Conduits API provides a consistent, portable programming model that hides the complexity of the underlying processor configuration and memory hierarchies as well as supporting both task and data parallelism. *Tasks* are hierarchical, modular abstractions for data processing that contain single program multiple data (SPMD) code that can execute on one or more processing units. *Conduits* synchronize and transfer data between tasks. With this framework, it is easy for a programmer to construct an application pipeline; a simple example is depicted in Figure 1. We have extended the PVTOL framework to include graphics processing units [2], making use of NVIDIA's Compute Unified Device Architecture (CUDA).

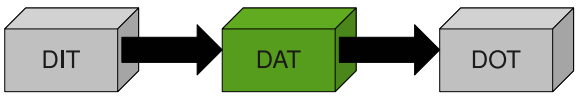

Fig. 1. Structure of a simple PVTOL pipeline application.

II. GPU Tasks and Conduits

Recently, there has been a proliferation of GPUs as co-processors for accelerating computationally intensive applications. Extending the PVTOL Tasks and Conduits framework to include these devices allows programmers to exploit the processing power of GPUs in their applications with minimal effort. A version of Tasks and Conduits for GPU coprocessors has been developed to provide abstractions for intelligently allocating memory, transferring data between the host and device, and executing kernels. Support for GPUs is seamlessly integrated with existing PVTOL Task and Conduit API using the Compute Unified Device Architecture (CUDA). Swapping the DAT Task and connected Conduits shown in Figure 1 with a GPU Task and GPU Conduits, the DAT will execute it's core computation on a GPU device. The code for each Task runs as a separate host thread, and follows the same rules of modularity and independence as regular PVTOL Tasks. The GPU Task maintains the same API as a regular PVTOL Task, and has similar responsibilities. They are responsible for querying their conduits to obtain access to data buffers, for setting up kernel parameters, and for initiating execution of kernels on the device.

The host Tasks and GPU Tasks communicate via GPU Conduits, which manage data buffers in the host's shared memory and the GPU's global memory. Conduits are also responsible for the synchronization of access to those buffers. PVTOL GPU Conduits execute their functionality on the GPU co-processor by interfacing with the NVIDIA CUDA Runtime API through a set of GPU Utility functions and can handle any data type or size in up to three dimensions. The GPU Utility functions provide means for initializing the GPU, allocating data, moving data, checking parameters, and executing kernels. The primary goal of GPU Conduits is to isolate all platform dependent code, allowing all versions of PVTOL Tasks and Conduits to maintain the same API and remain completely portable. The GPU Utility functions take in parameters related to the dimensions, type, and location of the data and configure the appropriate arguments for function calls to the CUDA API. These utility functions perform parameter checking to ensure operations the user wants to perform are supported by the hardware; error checking to ensure that the operations execute on the hardware without errors; and interfacing to platform-dependent third-party libraries such as CUFFT and CUBLAS. GPU Conduits perform the exact same forms of data storage, movement, and synchronization as regular PVTOL Conduits, but with one set of buffers located on the host and another located on the GPU device. Currently, GPU Conduits exist for the following modes of communication:

- Host ⇒ GPU
- GPU ⇒ Host
- GPU1 ⇒ Host ⇒ GPU2
- GPU1 ⇒ GPU1 (shared buffer)

The extension of the PVTOL Task and Conduit framework to include GPUs successfully abstracts the interaction and use of GPU coprocessors from the user. The code that the user writes to interact with the GPU is limited to the kernels that run on the GPU; such kernels are frequently available from third parties. The extension of PVTOL to include GPUs further realizes the goal of providing a portable, easy to program abstraction of the underlying hardware for high-performance computing applications.

III. Fluorescence Mediated Tomography

We have used PVTOL for applying Monte Carlo methods in Fluorescence Mediated Tomography. FMT is emerging as an important molecular imaging modality which, in contrast to simple planar imaging, allows visualization of 3-dimensional distributions of fluorophores in live animals non-invasively. Combined with fluorescent probes directed to specific molecular targets, FMT has important potential applications in, for example, studying disease development in small animals, novel drug discovery, and monitoring molecular responses of disease to novel therapeutics [3]. Like many other medical imaging techniques, FMT consists of a forward problem, accurately modeling light propagation from source to detector through the medium being studied; as well as an inverse problem, solving the system of equations from the forward problem to reconstruct the final image. Major challenges in FMT are the high degree of light scatter through biological tissue which limits the potential imaging resolution of the technique [4], as well as the large amount of computation required for both the forward and inverse problems. Monte Carlo methods for computing the photon propagation in biological tissue yield accurate results with few, if any, assumptions [5]. We have chosen this algorithm to include in a prototype application utilizing PVTOL and extending it to GPU Tasks and Conduits for performance improvement.

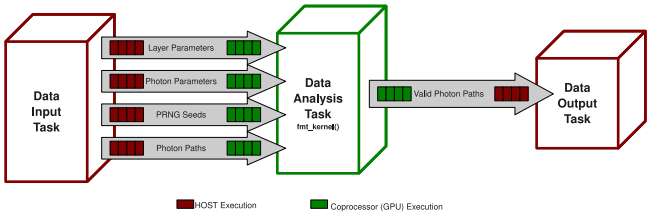

Fig. 2. PVTOL FMT Application pipeline

We have developed a C version of the FMT algorithm based off the MCML software package [5]. A PVTOL Tasks and Conduits application, shown in Figure 2 has been developed. In addition, a NVIDIA CUDA version of the algorithm was developed so that the DAT Task will run either the C or CUDA version of the algorithm. Our CUDA implementation has 512 threads arranged into a single thread block. The algorithms track photon propagation between a single source to a single detector. Once the pipeline is constructed, porting the application from the C version to the CUDA version only requires a change of 12 SLOC.

IV. Experimental Results

We simulated photon propagation of 10M photons, with the source and detector separated by 0.5cm of diffusive media, with optical properties matching those of biological tissue. Regardless of the time it takes a photon packet to travel through the media from source to detector, the two implementations provide nearly identical results, confirming the accuracy of our new GPU-based time-domain Monte Carlo algorithm. Figure 3 shows overall performance results for simulating 10^7 photon packets propagating from source to detector through the media. These are end-to-end results, including all initialization, data transfer, file I/O operations, as well as core computation. Our complete GPU implementation demonstrated a speedup of 6.777x versus the complete C implementation.

	Execution Time	**Speedup**
C	1:01 (hr:min)	1.000x
CUDA	0:09 (hr:min)	6.777x

Fig. 3. Performance statistics for simulating 10^7 photon packets propagating through 0.5 cm of turbid media.

V. Future Directions

The CUDA version of the Monte Carlo algorithm is currently unoptimized. We expect to achieve even more pronounced speedups. In addition, we are developing a version that processes photons from multiple detectors. For GPU PVTOL, we are investigating adding direct GPU to GPU communication to our conduits.

VI. Acknowledgements

We would like to recognize and thank Hahn Kim, Sanjeev Mohindra, Jeremy Kepner, Robert Bond, and MIT Lincoln Laboratory for their support in providing us the PVTOL source code.

References

[1] H. Kim, E. Rutledge, S. Sacco, S. Mohindra, M. Marzilli, J. Kepner, R. Haney, J. Daly, and N. Bliss, "PVTOL: providing productivity, performance and portability to DoD signal processing applications on multicore processors," in *DoD HPCMP Users Group Conference, 2008. DOD HPCMP UGC*, 2008, pp. 327–333.

[2] S. Mohindra, J. Daly, R. Haney, and G. Schrader, "Task and conduit framework for multi-core systems," in *DoD HPCMP Users Group Conference, 2008. DOD HPCMP UGC*, 2008, pp. 506–513.

[3] R. Weissleder and M. J. Pittet, "Imaging in the era of molecular oncology," *Nature*, vol. 452, no. 7187, pp. 580–589, Apr. 2008.

[4] A. H. Hielscher, "Optical tomographic imaging of small animals," *Current Opinion in Biotechnology*, vol. 16, no. 1, pp. 79–88, Feb. 2005.

[5] S. L. Jacques, L. Zheng, and L. Wang, "MCML–Monte carlo modeling of light transport in multi-layered tissues." *Computer Methods and Programs in Biomedicine*, vol. 47, no. 2, pp. 131–146, 1995.