ACCELERATED PARTICLE TRACKING USING GPULIB

V. Ranjbar, I. Pogorelov, P. Messmer, K. Amyx,
Tech-X Corporation, Boulder, Colorado, USA

Abstract

A 4D version of BNL's spin tracking code SPINK [1] with limited elements has been successfully ported to a C++/GPU platform using GPULib [2]. This prototype used only quadrupoles, simple snakes, dipoles and drifts. We present the approach used to track spin and orbit degrees of freedom of polarized proton beams simultaneously. We also present recent results of prototyping a general-purpose particle tracking on GPUs, discussing our CUDA implementation of maps for single-particle dynamics in the Argonne National Lab's accelerator code ELEGANT [3] where a 40x speedup was achieved for single-particle-dynamics elements.

GENERAL PURPOSE COMPUTING ON GPUS

In recent years, general purpose computing on graphics processing units (GPUs) has attracted significant interest from the scientific computing community because these devices offer a large amount of computing power at very low cost. Unlike general purpose processors, which are designed to address a variety of tasks ranging from control flow and bit manipulation operations to floating-point operations, GPUs are optimized to perform floating-point operations on large data sets. Instead of allocating a large amount of the on-chip real estate for large cache memory and control flow logic, GPUs dedicate a lot of resources to floating-point units. A GPU like the one found in the NVIDIA Tesla C2050 serias consists of 15 vector processors with a vector length of 32 elements. Using predicated execution enables each vector element to execute its own flow through the program, providing the impression of 448 independent execution units.

The introduction of the Compute Unified Device Architecture (CUDA) by NVIDIA has made it possible for computational scientistst without a deep knowledge of graphics oriented programming interfaces like OpenGL to take advantage of the high processing power offered by GPUs.

CUDA enables users to develop algorithms in C++ with a small set of language extensions. The developers write so-called *kernels*, code that executes on the GPU defining the behavior of a single thread of execution. Typically, a kernel is executed by thousands of threads concurrently and the GPU's thread manager maps them to the physical thread processors. The kernel is invoked on the host side, at which time it is determined how many threads will be executed. Memory management, data transfer and kernel invocations are all controlled by the host CPU. A special compiler, nvcc, translates kernels and host programs into code that executes on the CPU and on the GPU. This architecture simplifies significantly the software development process for CUDA-enabled GPUs, but it still requires a detailed knowledge of the GPU's architecture in order to obtain good performance. For example, while the threads can be treated independently of each other, they are in fact executed on a Single-Instruction-Multiple-Data (SIMD) type architecture. E.g. on a C2050 card, 32 threads are executed using the same instruction stream, which means that diverging threads can lead to a large amount of stalled threads thus degrading performance. Also, one of the benefits of GPUs is their large memory bandwidth, but in order to take advantage of it, memory access of different threads has to be carefully aligned. Finally, many inherently sequential algorithms, such as cumulative sums of a vector, are straightforward to implement on a serial processor. However, optimization on a massively parallel system like GPUs requires carefully crafted routines. In order to free developers of the burden of low-level GPU code development, Tech-X developed GPULib, a library of GPU vector operations (http://GPULib.txcorp.com) [2]. While mainly designed to be used from within high-level languages, GPULib can also be used from C or Fortran.

GOALS OF THE CURRENT WORK

Our goal is to provide a set of fast orbit tracking kernels for ELEGANT and spin-orbit tracking classes for the Unified Accelerator Library (UAL) [4]. Using the UAL paradigm will make these classes usable by a wide variety of codes. The core spin transport function in SPINK, Sprot(), will be extracted translated to Templated C++ and turned into a self-contained class. These classes will wrap and self-contain core GPU kernels which will drive the numerically expensive particle pushes. These GPU kernels will be made available in future GPULib packages.

ELEGANT is an open-source, multi-platform code used for design, simulation, and optimization of FEL driver linacs, ERLs, and storage rings [3, 5]. The parallel version, PELEGANT [6, 7], uses MPI for parallelization and shares all source code with the serial version. Several new "direct" methods of simultaneously optimizing the dynamic and momentum aperture of storage ring lattices have recently been developed at Argonne [8]. These powerful new methods typically require various forms of tracking the distribution for over a thousand turns, and so can benefit significantly from faster tracking capabilities. Because the ability to create fully scripted simulations is essential in this approach, ELEGANT is used for these optimization computations.

Orbit Tracking

Machines are defined in terms of a 'lattice' of elements which act to 'steer' and accelerate the beam. For example, in SPINK Particles are tracked through the lattice in terms of their 6D phase space coordinates.

$$\vec{r} = x, p_x, y, p_y, z, p_z \tag{1}$$

In many codes this is accomplished by applying a transport map to propagate the particles from one element to the next:

$$\vec{r}^{n+1} = M^{n}(\vec{r}^{n}) \tag{2}$$

where *M* is the map and n is the element number. In the simplest cases these maps are just 6x6 matrices however for higher order tracking they can take the form of Taylor maps or include 'space-charge' effects which require solving for the self fields.

ELEGANT is fundamentally a lumped-element particle accelerator tracking code utilizing 6D phase space, and is written entirely in C. A variety of numerical techniques are used for particle propagation, including transport matrices (up to third order), symplectic integration, and adaptive numerical integration. Collective effects are also available, including CSR, wakefields, and resonant impedances. Presently, we are working on prototyping key ELEGANT particle tracking algorithms on NVIDIA GPUs and showing that such accelerated implementations can be incorporated into ELEGANT. To achieve this goal, we focus on one element described by a transfer map (a quadrupole), and one collective-effect element (a drift with 1D longitudinal space charge). Our longer-term goal is to expand the kernel library to include optimized implementation on GPUs of most of the ELEGANT elements, starting with the most time consuming ELEGANT kernels.

Spin Tracking

Existing codes usually track both orbital and spin coordinates of a beam of spin-1/2 particles through the lattice of a circular accelerator. The spin vector $\vec{S}$ in the particle rest frame precesses in the machines electric and magnetic fields according to the Thomas-BMT equation [9].

$$\frac{d\vec{S}}{dt} = \vec{S} \times \vec{\Omega}, \tag{3}$$

where

$$\vec{\Omega} = \frac{e}{m\gamma}\Big[(1+G\gamma)\vec{B}_{\perp} + (1+G)\vec{B}_{\parallel} + \tag{4}$$

$$\frac{1}{c}(G\gamma + \frac{\gamma}{1+\gamma})\vec{E}\times\vec{\beta}\Big] \tag{5}$$

For s based tracking purposes $\vec{\Omega}$ is usually expressed in terms of an expansion in magnetic field in terms of the Frenet-Serret coordinate system and Magnetic fields to obtain,

$$\vec{\Omega} = \frac{h}{B\rho}\Big((1+G\gamma)\vec{B} - G(\gamma-1)(\vec{r'}\cdot\vec{B})r'\Big) \tag{6}$$

where

$$h = h(x, x', y') = \sqrt{x'^2 + y'^2 + (1+x/\rho)^2} \tag{7}$$

and

$$r' = \frac{\vec{v}}{v} \tag{8}$$

where v and $\vec{v}$ is the velocity magnitude and vector respectively. Following [10] , over an infinitesimal step size δs solutions to the T-BMT equation result in a spin transport map,

$$\begin{pmatrix} 1-(B^2+C^2)c & ABc+Cs & ACc-Bs \\ ABc-Cs & 1-(A^2+C^2)c & BCc+As \\ ACc+Bs & BCc-As & 1-(A^2+B^2)c \end{pmatrix} \tag{9}$$

with,

$$\begin{aligned} c &= 1-cos(\omega\delta s) && (10)\\ s &= sin(\omega\delta s) && (11)\\ A &= \frac{\Omega_x}{\omega} && (12)\\ B &= \frac{\Omega_y - 1/\rho}{\omega} && (13)\\ C &= \frac{\Omega_z}{\omega} && (14)\\ \omega &= \sqrt{\Omega_x^2 + (\Omega_y - 1/\rho)^2 + \Omega_z^2} && (15) \end{aligned}$$

The terms in the spin transport matrix Eq. 9 contain up to fourth order terms in $\vec{r}$. Thus unlike the orbit push where the same transport map is applied to all the particles, the spin transport maps are unique to each phase space point and require calculation on the fly.

INITIAL RESULTS

Spin-Orbit Tracking Efforts Using GPULib

A 4D version of BNL's spin tracking code SPINK [1] with limited elements has been successfully ported to a C++/GPU platform using GPULib [2]. This prototype used only quadrupoles, simple snakes, dipoles and drifts.

Our approach was as follows:

1. The 4D phase space particles are loaded from the CPU into an N sized particle array on the GPU: X[N], PX[N], Y[N], PY[N]
2. The particles are then pushed through a 4x4 orbit transport matrix defined by the lattice. This is accomplished using several calls to GPULibs' gpuAddFAT function.

3. The input and output phase space variable are average to get Xavg[N] on the GPU using both GPULibs' gpuAddF and gpuAddFAT functions.

4. These averaged phase space points together with knowledge of the magnetic fields in the elements are then used to construct the elements of the 3x3 spin transport matrix. This involves calculation of up to 4th order terms in r based on the Thomas-BMT equation. The calculations were performed on the GPU using GPULibs' gpuMultF, gpuMultFAT, gpuAddF, gpuSqrtF, gpuDivF, gpuSinF and gpuCosFAT functions.

5. Finally the spin is pushed through this spin transport map now using GPULib's gpuMultF and gpuAddF functions

Prototype Kernels for Single-Particle-Dynamics in ELEGANT

As a first step toward enabling particle tracking with ELEGANT on GPUs, we implemented in CUDA the 2nd order map for the quadrupole beamline element (QUAD in ELEGANT notation). This implementation serves as a starting point for the transition of a set of ELEGANT's single-particle-dynamics algorithms to GPUs. So far, we implemented algorithms in both single and double precision, with an emphasis on optimizing the kernels for the NVIDIA "Fermi" GPU architecture. In our implementation, we stored particles in linear memory, and investigated schemes in which one particle is computed per thread. We utilize the high-bandwidth and low-latency constant memory cache to store and access the map parameters used to update the particle information. This minimizes memory traffic and has yielded superior performance to schemes we investigated that utilized L1 caching or shared memory.

For testing purposes, we generated a realistic 6D phase space distribution function that was propagated through a lattice consisting of a small number (~ 20) of quadrupoles and drifts (without space charge). In a simulation with 100k double-precision particles, we observed a 20x speedup on a C2050 Fermi GPU compared to a single core Intel Xeon X5650 @ 2.67GHz CPU, with a comparable speedup seen when traversing a single quadrupole. (The 20x speedup becomes approximately 40x when the computation is done in single precision.) In addition to developing kernels for other beamline elements, we plan to explore additional efficient ways of accessing the map parameters. The latter becomes more important in simulations with higher order maps, as the number of Taylor series coefficients that describe the map goes up.

ONGOING DEVELOPMENT

Older codes like SPINK used an averaged phase space value based on the values at the entry and exit of an element for $\vec{r}$ in the spin matrix calculation. However it has been determined that for the high precision necessary for spin tracking in the EDM (Electric Dipole Moment) and RHIC-Spin experiments, that this was not sufficient. More recent approaches use a thin element treatment of each magnet (as in UAL-TEAPOT), thus keeping the entry and exit phase space values is no longer necessary.

We are currently developing CUDA code embedded in UAL Templated C++ classes to perform the particle push in UAL-TEAPOT. This approach while easily integratable into the UAL framework has the draw back that particle information needs to be copied back and forth to the GPU on each function call. A better approach will be to maintain particle information on the GPU only uploading the initial distribution and downloading the final distribution after the tracking is complete. An even more efficient approach would be for both the 6x6 orbital transport matrix and magnetic fields for each element in a typical lattice (i.e. for RHIC 256 - 1000 active elements depending on slicing needs) to be preloaded on the GPU, then a single GPU kernel call can track many particles over many turns. We believe the final result will be capable of performing all three approaches depending on diagnostic details required by the modeler.

We are also working on developing prototype kernels for collective effects in ELEGANT, using as our test case the LSCDRIFT element (drift with longitudinal space charge). CUFFT library will be used to implement a DFT of the binned longitudinal space charge which is then multiplied by a known impedance function, the inverse DFT applied thereupon to transform the result back to the original space. A nontrivial aspect of this work is efficient implementation of charge binning algorithms, which are challenging to implement on a GPU due to the possibility of memory contention between individual threads attempting to deposit charge to the same bins. We will investigate two approaches: atomic memory updates and data-parallel primitives. The Fermi architecture allows for fast floating-point atomic updates to memory locations that avoid thread contention issues. This can be utilized to perform a one-dimensional charge binning. The second approach is to apply a generic sorting algorithm (such as the radix sort implemented in the CUDPP library) to sort particles based on bin index; then perform a modified segmented prefix sum operation (also implemented in the CUDPP library) to calculate the average longitudinal position of particles based on bin index, as well as count the number of particles based on bin index; and finally calculate the charge contribution to first the lower charge bins, then the upper charge bins, based on the average longitudinal position and number of particles between bins. Depositing charge first to the "lower bins", synchronizing (in CUDA this is equivalent to launching a second kernel), and then depositing charge to the "upper bins" will allow us to avoid memory conflicts and perform the charge binning in a data-parallel manner.

ACKNOWLEDGEMENTS

This work is funded by the DOE/BES Grant No. DE-SC0004585, DOE/NP Grant No. DE-SC0004432 and Tech-X Corp. We would also like to thank NERSC for use of their computing resources.

REFERENCES

[1] A. U. Luccio, Spin tracking in RHIC code SPINK,"in Proceedings of the Adriatico Research Conference on Trends in Collider Spin Physics, (Singapore), p. 235, World Scientific, 1995.

[2] P. Messmer, P. J. Mullowney, and B. E. Granger, "GPULib: GPU computing in high-level languages," Comput. Sci. Eng., vol. 10, no. 5, pp. 70-73, 2008. (`http://GPULib.txcorp.com`)

[3] M. Borland, "elegant: A Flexible SDDS-compliant Code for Accelerator Simulation", APS LS-287, September 2000.

[4] N. Malitsky and R. Talman, "The Framework of Unified Accelerator Libraries" `http://www.slac.stanford.edu/econf/C980914 /papers/C-We13.pdf`.

[5] M. Borland, V. Sajaev, H. Shang, R. Soliday, Y. Wang, A. Xiao, W. Guo, "Recent Progress and Plans for the Code ELEGANT," in Proceedings of 2009 International Computational Accelerator Physics conference, San Francisco, CA, WE3IOpk02 (2009).

[6] Y. Wang, M. Borland. "Implementation and Performance of Parallelized Elegant", in Proceedings of PAC07, THPAN095 (2007).

[7] H. Shang, M. Borland, R. Soliday, Y. Wang, "Parallel SDDS: A Scientific High-Performance I/O Interface," in Proceedings of 2009 International Computational Accelerator Physics conference, San Francisco, CA, THPsc050 (2009).

[8] M. Borland, V. Sajaev, L. Emery, and A. Xiao, "Direct Methods of Optimization of Storage Ring Dynamic and Momentum Aperture", in Proceedings of PAC09, TH6PFP062 (2009).

[9] V Bargmann and Louis Michel and V. L. Tegegdi, "Precession of the polarization of particles moving in a homogenous electromagnetic field", PREVL. **2** pp.435–436 (1959).

[10] A. Luccio, "Spin Rotation Matrices for Spin Tracking", AGS/RHIC/SN No. 013 (1996).