GPU ACCELERATED ONLINE MULTI-PARTICLE BEAM DYNAMICS SIMULATOR FOR THE LANSCE LINAC*

X. Pang†, L. Rybarcyk, S. A. Baily, Los Alamos National Laboratory, Los Alamos, NM, 87544, USA

Abstract

A GPU-accelerated online multi-particle beam dynamics simulator is being developed for use in the LANSCE linac operation. The goal is to provide new insights on the beam distribution inside the linac and to help understand the impact of set point adjustments on it. Details regarding the code structure design, GPU programming and performance, code validation and status will be presented.

INTRODUCTION

The Los Alamos Neutron Science Center (LANSCE) linac can provide H^+ and H^- beam up to 800 MeV. Due to a lack of direct beam measurements during high power operation, small adjustments to linac operating set points are made primarily on the basis of beam loss along the linac. To provide more insight on the beam distributions along the linac and to better understand the effects of machine parameter changes on the beam, we are creating a virtual beam diagnostic tool for use in the LANSCE control room [1]. This tool is a Graphics Processing Unit (GPU)-accelerated multi-particle simulator whose beam dynamics algorithms are based upon the ion linac design and simulation code PARMILA [2]. PARMILA is a z-code which uses the R matrices to transfer particles through accelerator elements. The simulator is designed using modern software design techniques and coded in C++ and Python. Using NVIDIAs CUDA technology [3], the algorithms are recast to fully harvest the computing power of a GPU. Once connected to the EPICS control system, the simulator can track the real time machine parameter changes, convert control set points to model quantities and quickly update the simulation.

CODE STRUCTURE

Since the simulator is developed for use in the control room, fast execution and ease of use are both important design criteria. To meet these criteria, a combination of a high-level scripting language, i.e. Python, and a low level compiled language, i.e. C++, were adopted. Figure 1 shows the hierarchy of code layers that allows the user to take advantage of the fast number-crunching kernels written in C++ and CUDA C without having to deal with the complex syntax and the lengthy compilation process that is associated with them. The outermost shell of the code structure that a user directly interfaces with is written in Python. Python is chosen as the high-level scripting language because of its shallow learning curve, pseudo

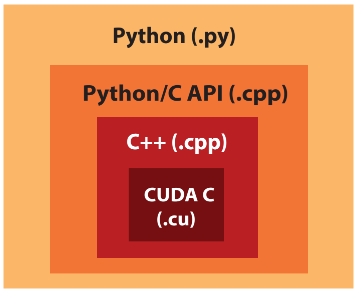

Figure 1: Code hierarchy. Lower level codes are wrapped by the higher level ones.

code like syntax, lack of compilation phase, easy integration with the compiled languages and the rich numerical and visualization libraries. Figure 2 shows the major components of the core part of the code that is written in C++ and CUDA C. Following the CUDA convention [3], we refer to the CPU as the host and the GPU as the device in the rest of this paper. The components colored in blue (left) are the ones that reside on the host while those colored in yellow (right) reside on the device. The *BeamLine* component (middle) resides in pinned, or page-locked memory on the host and the information stored is shared between the host and device and denoted by the green block. Each major component is described in detail as follows:

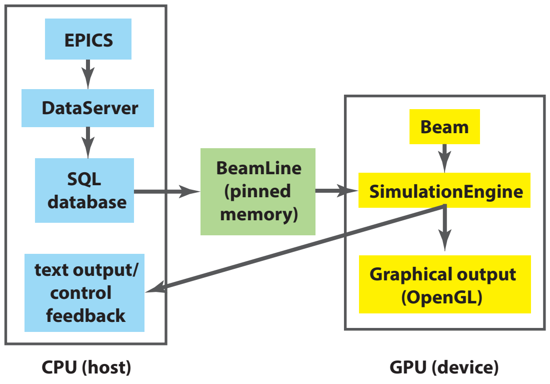

Figure 2: High level code structure and data flow indicated by the arrows.

The *DataServer* reads values from EPICS in multi-threaded fashion and stores the updated values to the SQL database.

The *SQL database* stores all the measured engineering values coming in from the *DataServer*, the conversion algorithms for every controllable accelerator element, and most importantly the converted physics quantities that are

* Work supported by U.S. DOE, NNSA under contract DE-AC52-06NA25396.

† xpang@lanl.gov

eventually used in the beam dynamics simulations. A central database repository was chosen over flat files to guarantee data consistency and to provide historical tracking of data changes. Due to its light weight, self-contained and server-less features, *SQLite* was picked as the relational database management system (RDBMS) for the simulator.

The *BeamLine* component of the code is a data structure that stores the physics quantities of all the accelerator elements required in the simulation. It is updated in real-time once changes to the SQL database are detected so that the physics model used in the simulation is synchronized with the real world machine parameter settings. This data is stored in the pinned memory. In this way, modifications by either the host or the device can be reflected in the memory spaces of both sides, eliminating the need to first search for the element index for which the changes were made, then explicitly copying the new value from host to device.

The *Beam* class is used to store coordinate information for all the particles and some collective properties of the beam. Particle data are allocated on the global memory of the GPU in the format of structure-of-arrays (SoAs) to facilitate coalesced global memory access in the particle tracking simulation and beam property calculations. In order to reduce the expensive memory traffic between the host and the device, the *Beam* object is allocated on the GPU before the simulation starts and is kept on the GPU until the simulation ends. This class also provides methods to calculate different properties of the beam like centroid, size, emittance and beam loss etc. on the GPU.

The *SimulationEngine* component is the core of this simulator. All the physics algorithms for particle advancing and space charge effects are implemented in the *SimulationEngine*. These algorithms are stored in several CUDA kernel functions and are glued together by two overarching C++ classes, *SimulationEngine* and *SpaceCharge*. The *SimulationEngine* class orchestrates the whole simulation while the *SpaceCharge* class only manages the parameters and functions used in space charge calculations.

The *Output* of the code can be a graphical display, a text file or specific values sent back to the EPICS control system. A graphical display system is being developed using openGL so that simulation results sitting on the GPU can be conveniently displayed without copying back to the host when a graphical output enabled GPU is used.

GPU PERFORMANCE

The performance of the GPU accelerated simulator is compared with that of the optimized single CPU version of the code. The test system used for the comparison is a custom built workstation with Intel Xeon E5520 2.27GHz, 32 Gb of RAM, and a NVIDIA GTX 580 (Fermi architecture), running an x86_64 install of Scientific Linux 6.3 and CUDA 5.0.

The physics algorithm of the code consists of two major parts: particle advancing and space charge calculations. For particle advancing, the instruction level parallelism (ILP) was explored by assigning one thread to process four particles. This allows more computing resources to be used by a thread, more data to be reused and independent calculations within a thread to be processed in parallel, therefore more latency to be hidden. An overall faster execution can be achieved. In the space charge force calculation, particles are first binned onto a mesh and stored in a charge density table. Then an electric field table is calculated based on the charge density. Space charge kicks on the particles are applied by interpolating the electric field table based on individual particle's position on the mesh. Among these steps, the charge density table calculation accounts for the most computing time and is the bottleneck of the overall performance. To avoid the race condition when the density table is updated by multiple threads, the floating point atomic addition in the CUDA is used. This gives us about 12 times speedup in the charge density calculation compared to the CPU version.

Two LANSCE beam lines were used for the overall performance test: part of the H^+ low energy beam transport (LEBT) which consists of 8 quadrupoles, 4.32 meter long drift spaces (divided into 442 segments to apply space charge kicks), a RF cavity and a dipole and the drift tube linac (DTL), which includes 135 quadrupoles, 165 RF gaps and 1.68 meter long inter-tank drift spaces (divided into 171 segments). Typically using 32K particles in the simulation is sufficient to get an accurate estimate of many of the beam properties.

Without calculating the space charge effect, transporting 32K particles through an accelerator element using GPU can be 160 times faster than on a single CPU. Table 1 summarizes the total computing time including the space charge calculations and the speedup of simulations using different particle numbers. The overall performance is dominated by the space charge calculation which consumes between 70% to 90% of the computing time. One simulation through the transport structure on the GPU using 32K particles takes less than a second, whereas on a single CPU, it takes more than half a minute. In an accelerator control room where rapid response from any system is required, a reduction from half a minute to a fraction of a second makes this simulator a viable tool for pseudo-real-time computational turn-around in this demanding environment.

RESULTS

Before relying on the simulator to provide useful insight about the beam evolution in the linac under routine operating conditions, it must first be calibrated. Details about the calibration process is described in [4]. Figure 3 shows that after the calibration, we were able to get an excellent agreement between the results from the actual phase scan measurements made with the LANSCE H^+ beam and the results from the simulated phase scan following the adjustment of the relevant calibration factors. Figure 4 is a screen shot of the EPICS control channels and the simulator outputs for the simulation of H^- beams at LANSCE accelerated from 0.75 to 100 MeV through the DTL.

Table 1: Computing Time and Speedup

Initial Particle Number	Beamline Structure	Overall Time [s]	Overall Speedup
16384	Transport	0.66	45.8
	DTL	0.37	26.7
32768	Transport	0.78	44.5
	DTL	0.47	34
65536	Transport	1.02	42.3
	DTL	0.71	39
131072	Transport	1.48	40.1
	DTL	1.15	45.2
262144	Transport	2.4	39.8
	DTL	2.07	48.6

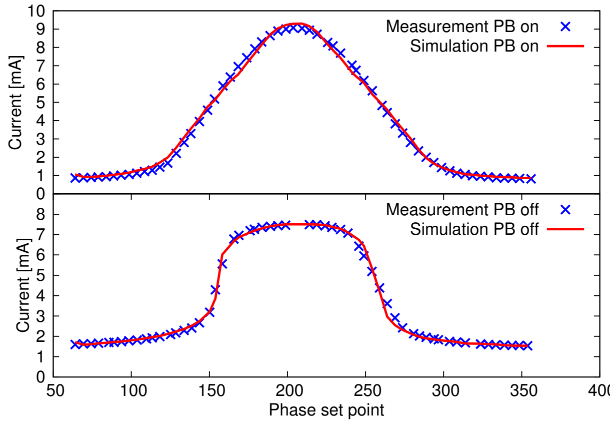

Figure 3: Phase scan measurements (blue x's) of the LANSCE DTL tank 1 with main buncher on and prebuncher on (top) and off (bottom). The simulated phase scan results following calibration factor adjustment are shown in red.

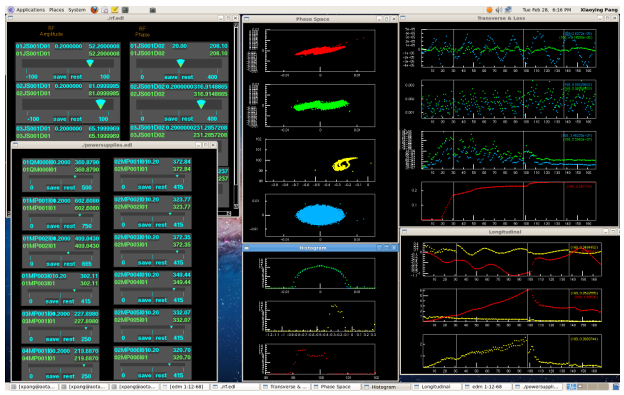

Figure 4: Screenshot of the EPICS control interfaces (left) and the simulator outputs (right).

Numerous beam properties such as phase space distributions and phase space projections at the end of the DTL as well as beam centroid, size, emittance, and fractional beam loss along the DTL are displayed. From this interface, changes that are made to relevant DTL operating parameters can be captured by the simulator and reflected in the simulation results in pseudo realtime. These 2D graphics displays were made using OpenGL, which maintains high throughput as additional communications through the CPU are minimized. This type of interface gives control room personnel additional information regarding beam performance and allows them to better understand the impact of any machine parameter changes to the overall accelerator performance.

STATUS AND FUTURE WORK

Currently, the simulator contains several standard transport elements and the DTL structure. The addition of the coupled-cavity linac (CCL) structure to the code is underway as well as improvements to the Python interface and data I/O. Following that, our near term plans include generation and calibration of the LANSCE CCL in the model and front-to-end simulations of the LANSCE linac from 0.75 to 800 MeV. More long range plans under consideration include simultaneously modeling multiple beam species, addition of other beam loss mechanism, e.g. H- stripping [5] and addition of a new space charge calculation routine based on a 3D Poisson solver.

CONCLUSION

By combining multi-particle tracking simulation algorithms with state-of-the-art GPU technology we have developed a powerful tool for use in the demanding accelerator control room environment. It has many applications not mentioned here [4][6][7]. Although more efforts are needed to further calibrate the existing model and to incorporate more accelerator structures, we believe this approach can greatly enhance operations and performance of existing and future accelerators.

REFERENCES

[1] X. Pang, L. Rybarcyk, S. A. Baily, IPAC, 2012.

[2] H. Takeda, J. H. Billen, Parmila, LA-UR-98-4478, 2005.

[3] NVIDIA Corporation, CUDA C Programming Guide, version 5.0, 2012.

[4] L. Rybarcyk, X. Pang, PAC 2013, 2013.

[5] L. Rybarcyk, C. Kelsey IV, R. McCrady, X. Pang, IPAC 12, 2012.

[6] A. Scheinker, X. Pang, and L. Rybarcyk, submitted for publication to Phys. Rev. ST-Accel. Beams.

[7] X. Pang, L. Rybarcyk, "Multi-objective Optimization for LANSCE Linac Operations", ICALEPCS 2013.