SYMPLECTIC MULTI-PARTICLE TRACKING USING CUDA*

Z. Liu[1†], J. Qiang[‡], Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA
[1]also at Key Laboratory of Particle Acceleration Physics and Technology,
Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China

Abstract

The symplectic tracking model can preserve phase space structure and reduce non-physical effects in long term simulation. Though this model is computationally expensive, it is very suitable for parallelization and can be accelerated significantly by using Graphic Processing Units (GPUs). Using a single GPU, the code achieves a speedup of more than 400 compared with the time on a single CPU core. It also shows good scalability on a GPU cluster at Oak Ridge Leadership Computing Facility. In this paper, we report on the GPU code implement, the performance test on both single-GPU and multi-GPU cluster, and an application of beam dynamics simulation.

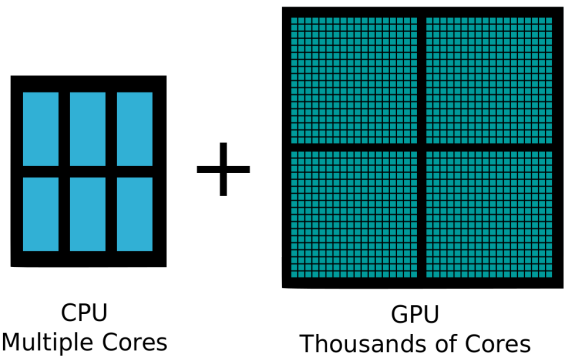


Figure 1: A schematic plot of CPU and GPU.

INTRODUCTION

Numerical simulation plays an important role in the beam dynamics study and design of high intensity accelerators, where space charge effects dominate. Most simulation codes at the accelerator community use Particle-In-Cell (PIC) method as the space charge solver [1–7]. The PIC method is an efficient algorithm to include self-consistent space charge effects. However, there are still some arguments that if the PIC algorithm could keep the symplectic condition during the particle tracking.

Symplectic integrators were constructed for conserving the symplectic condition of Hamiltonian systems [8, 9]. Recently, a gridless symplectic particle tracking model was introduced and proved to be effective in serving as symplectic Poisson solver in long-term simulation [10]. It can effectively reduce the emittance growth associated with numerical grid heating compared with the PIC algorithm. However, this model is much slower compared with the PIC method. Fortunately, it is very suitable for parallelism and can achieve very good scalability, especially by using the Graphics Processing Unit (GPU).

In contrast to CPU computer, one GPU contains several hundreds or even thousands of cores, as shown in Fig. 1. It uses high-bandwidth bus (~200Gb/s) connecting the memory on chip to the computing cores and is optimized for parallel calculations, particularly for single instruction multiple data (SIMD) operations [11]. The Compute Unified Device Architecture (CUDA) library is a parallel computing platform and GPU programming model developed by NVIDIA [12]. It enables dramatic increase of computing performance by harnessing the power of GPUs. By using the CUDA library, the gridless symplectic multi-particle tracking code can be speeded up significantly.

In this paper, firstly, the symplectic tracking theory and the GPU implements are introduced in Section 2. Then, the performance of the tracking code using CUDA is presented in Section 3. After that, an application example using this code is presented in Section 4. Finally, conclusions are drawn in Section 5.

CODE IMPLEMENTATION

For a 3D bunch, the space charge transfer map in one dircction (X) direction can be expressed as [10]:

$$\begin{cases} x_i(\tau) = x_i(0) \\ p_{xi}(\tau) = p_{xi}(0) - \tau \dfrac{1}{\varepsilon_0}\dfrac{8}{abc}\omega\kappa\gamma_0 \\ \times \sum\limits_{j=1}^{N_j}\sum\limits_{l=1}^{N_l}\sum\limits_{m=1}^{N_m}\sum\limits_{n=1}^{N_n}\left(\dfrac{\alpha_l \sin(\alpha_l x_j)\sin(\beta_m y_j)\sin(\gamma_n z_j)}{(\alpha_l^2+\beta_m^2+\gamma_n^2)}\right. \\ \left.\times \cos(\alpha_l x_i)\sin(\beta_m y_i)\sin(\gamma_n z_i)\right) \end{cases} \quad (1)$$

where $\alpha_l = \frac{l\pi}{a}, \beta_m = \frac{m\pi}{b}, \gamma_n = \frac{n\pi}{c}$. a, b and c are the boundary length.

In the GPU implement, we define:

$$S_j^{l,m,n} = \sin(\alpha_{l,m,n} x_j), C_j^{l,m,n} = \cos(\alpha_{l,m,n} x_j) \quad (2)$$

where j is the index of the particles, and l, m and n are the indices of spectral mode in X, Y, and Z direction. The trigonometric functions are calculated firstly.

Noticing that in the transfer map 1, the summation by index j is for every particle, so the sequence for summing can be changed to save computational complexity. Defining that:

$$Phi^{lmn} \equiv \sum_{j=1}^{N_j} S_j^l S_j^m S_j^n \quad (3)$$

* Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and the Ministry of Science and Technology of China under Grant No.2014CB845501.
† zhicongliu@lbl.gov
‡ jqiang@lbl.gov

Copyright © 2017 CC-BY-3.0 and by the respective authors

and if the calculation of Phi^{lmn} for $N_l \times N_m \times N_n$ modes is done first, the transfer map 1 can be rewritten as:

$$p_{x,i}(\tau) = p_{x,i}(0) - ep_{x,i} \quad (4)$$

$$ep_{x,i} \equiv \tau \frac{1}{\varepsilon_0} \frac{8}{abc} \omega\kappa\gamma_0 \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{Phi^{lmn} \alpha_l C_i^l S_i^m S_i^n}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)} \quad (5)$$

In this way, the computation complexity is reduced from $\alpha N_p^2 * N_{modes}$ to $\alpha N_p * N_{modes}$, which makes the symplectic particle tracking algorithm feasible.

The calculation of ep_i and pushing particles are executed separated in three directions to achieve high GPU occupancy. In the subroutine for each direction, a particle takes one thread. The subroutine of calculating ep_i has two branches depending on the problem size. It fully take advantage of the GPU constant memory, which is used for data not change within a kernel and specially optimized for broadcast, and the GPU shared memory, which is a fast memory residing on chip and can only be access by the threads within a block.

PERFORMANCE

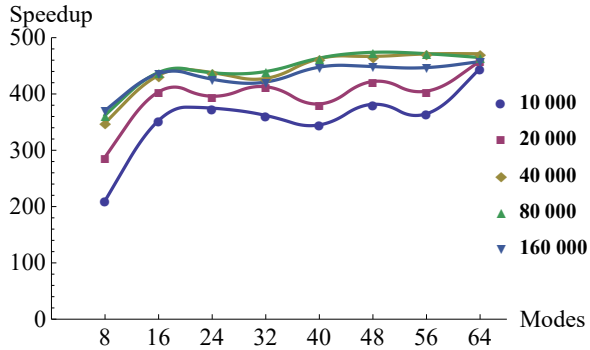
We have done two tests for measuring the efficiency and scalability of the GPU code. The first one ran the code on a common home-use GPU and the efficiency is compared with that running on a CPU core. The second test ran the code on a GPU cluster, Titan, a hybrid-architecture supercomputer located at Oak Ridge Leadership Computing Facility (OLCF), to show how the speedup change with the number of GPUs.

Single GPU Speedup

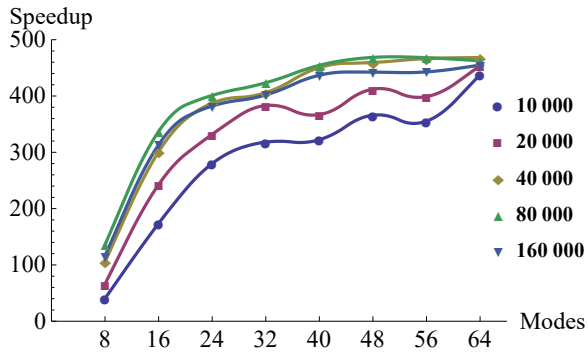
Firstly, the performance of the GPU code is compared with the CPU code. The GPU code runs on the GeForce GTX 1060 6GB (Pascal architecture), with CUDA version 8.0, while the CPU code runs serially on a single core of AMD Opteron(tm) Processor 6134. The speedup is calculated by using the CPU verion runtime divided by the GPU version runtime. In this measurement, two comparisons are made separately for only space charge kicker and the entire code, as shown in Fig. 2. Each line denotes the result of different number of particles.

The top plot of Fig. 2 shows the speedup of the space charge kicker with different problem sizes. It meets the expectation that the larger number of modes, the higher speedup it can achieve. When the number of modes is small, the speedup is low because it does relatively few computation. As the number of modes increasing, the computation becomes larger and more balanced, which results in a higher speedup. On the other hand, the particle number affects the speedup relatively slightly, especially with situation of large mode number. The reason is that the particle number exceeds the number of cores on GTX 1060(1280 cores). It can make fully use of all computing cores at the beginning.

The bottom plot shows the speedup of the total computing time, including external transfer map, coordinate conversion, space charge kicker, diagnose, and output. The speedups



(a) Speed up of the space charge kicker



(b) Speed up of the total time

Figure 2: Speedup of the symplectic code using a single GPU.

fall compared with that of only space charge kicker, but the trend keeps the same.

In general, we achieved a very good speedup. For the total runtime, the GPU code runs 450 times faster than the CPU code with a certain size of problem, while if we only consider the space charge kicker, the maximum speedup reached 460.

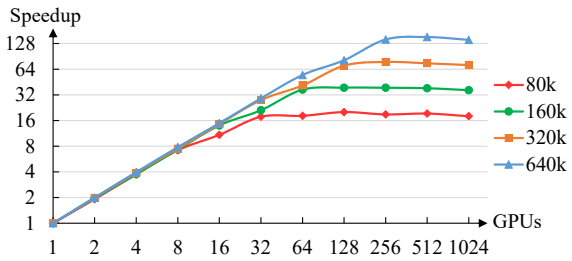
GPU Cluster Speedup - Titan

A strong scaling of the ImpactZ symplectic code was done on Titan to check how this algorithm performs with increasing number of GPUs. On the cluster Titan, each computing node contains one GPU. The way to transfer data among GPUs is to copy it back to CPU and then to communicate through Message Passing Interface(MPI). In the scaling test, we used 16×16×16 modes, which is the typical configuration in a real simulation. Figure 3 shows the speedup of the ImpactZ symplectic code running on different number of nodes compared with that running on a single node with the given problem size.

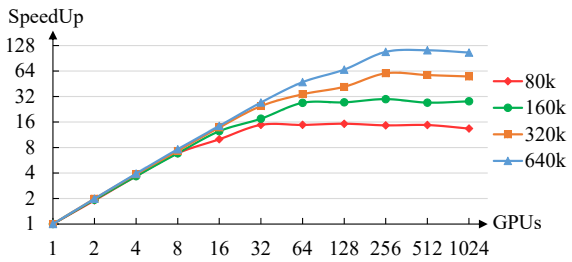
As shown at the top plot of Fig. 3, the speedup of space charge kicker increases almost linearly with the number of GPUs at the beginning and then reaches a limit gradually. The linear increasing is due to very small amount of data exchange, which is a great advantage of this gridless symplectic tracking model. On the other hand, the maximum speedup it can achieve is mainly limited by particle number.

Copyright © 2017 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-182-3



(a) speedup of space charge kicker



(b) speedup of total time

Figure 3: Strong scaling of symplectic code at GPU cluster.

Taking the situation of 160k particles using 64 nodes for example, with each GPU containing 2688 cores, totally we use 64×2688≈172k cores. It is even more than particle number. As a result, the speedup would not be linear as before.

The bottom plot shows the speedup of total time, including the transfer map, conversion from Z coordinates to T coordinates, diagnose, and output. The functions listed above are also parallelized, but it's more difficult to achieve a high parallelism due to its intrinsic less computation demanding. So the speedup of total time goes down a little compared with that of only space charge kicker.

APPLICATION SIMULATION

Several application simulations had been done using a periodic focusing channel and different currents using the GPU symplectic code. We set the phase advance per turn at 0 current to be 2.3979. With increasing current, the tune will be depressed and cross the third order resonance line of 2.3333 around 0.6 A. There is a sextupole at the end of each turn to excite the resonance.

Figure 4 shows the emittance growth with different currents. The emittance almost keeps constant with 0.1A and 0.2A, where the tune is about 2.40. However, it keeps growing with 0.4A, 0.6A, and 0.8A, where the tune is depressed below 2.33. It's shown that the emittance growth is due to the space charge driving 3rd order resonance.

The Poincaré map of the phase space coordinates of particles near 2.3333 is plotted in Fig. 5. In this contour plot, dark means large particle density. Different plots are from the results of particles starting from different initial position. Drived by the space charge force, the Poincaré map would be distorted and shaped into a triangle. The particles affected by resonance would gradually move towards outside. Finally, it would become part of beam halo and lost.

CONCLUSION

A gridless symplectic particle tracking model was implemented on GPUs using the CUDA library. The gridless tracking algorithm has the advantage to keep the symplectic condition and effectively reduce the noise driving emittance growth. We achieved a maximum speedup of more than 450 using a home-use GPU card GTX 1060. This algorithm also shows good strong scalability, which was tested on the GPU cluster Titan. Several application simulations were done using this code with different currents through a periodic focusing channel. No emittance growth appears when the tune is far away from the resonance line, while it keeps growing when the tune approaches it. In the future study, we will continue to extend this code and to compare the efficiency of this code with different architecture. We also would like to port the PIC model onto GPU and compare it with the symplectic gridless algorithm.

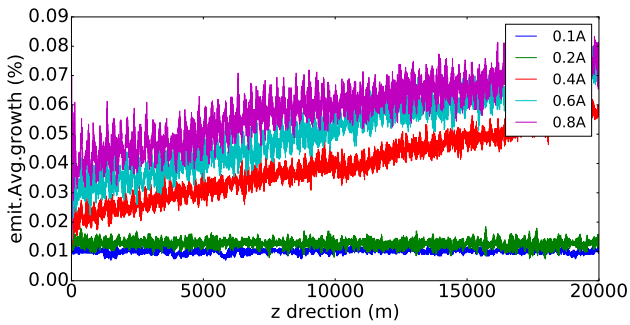


Figure 4: Emittance growth at different current.

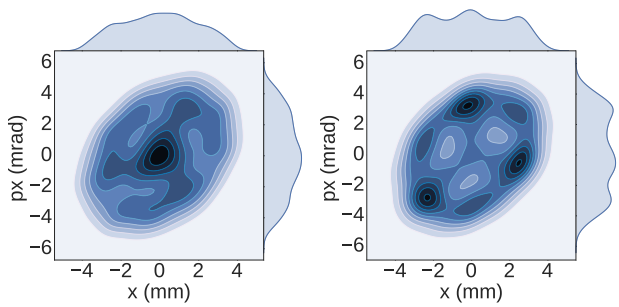


Figure 5: Poincaré map of the particles affected by 3^{rd} order resonance.

ACKNOWLEDGMENTS

One of the author, Zhicong Liu, would like to extend his thanks for the financial support from China Scholarship Council (CSC, File No. 201604910876). We have used computing resources at the Oak Ridge Leadership Computing Facility (OLCF).

REFERENCES

[1] C. K. Birdsall, "Particle-in-cell charged-particle simulations, plus monte carlo collisions with neutral atoms, pic-mcc," *IEEE Transactions on Plasma Science*, vol. 19, pp. 65–85, Apr 1991.

Copyright © 2017 CC-BY-3.0 and by the respective authors

[2] A. Friedman, D. P. Grote, and I. Haber, "Three-dimensional particle simulation of heavy-ion fusion beams," *Physics of Fluids B: Plasma Physics*, vol. 4, no. 7, pp. 2203–2210, 1992.

[3] J. Qiang, R. D. Ryne, S. Habib, and V. Decyk, "An object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators," *Journal of Computational Physics*, vol. 163, no. 2, pp. 434 – 451, 2000.

[4] J. Qiang, M. A. Furman, and R. D. Ryne, "A parallel particle-in-cell model for beam–beam interaction in high energy ring colliders," *Journal of Computational Physics*, vol. 198, no. 1, pp. 278–294, 2004.

[5] J. Amundson, P. Spentzouris, J. Qiang, and R. Ryne, "Synergia: An accelerator modeling tool with 3-d space charge," *Journal of Computational Physics*, vol. 211, no. 1, pp. 229 – 248, 2006.

[6] D. Uriot and N. Pichoff, "Tracewin," *CEA Saclay, June*, 2014.

[7] Y. K. Batygin, "Particle-in-cell code beampath for beam dynamics simulations in linear accelerators and beamlines," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 539, no. 3, pp. 455–489, 2005.

[8] P. Channell and C. Scovel, "Symplectic integration of hamiltonian systems," *Nonlinearity*, vol. 3, no. 2, p. 231, 1990.

[9] H. Yoshida, "Construction of higher order symplectic integrators," *Physics Letters A*, vol. 150, no. 5-7, pp. 262–268, 1990.

[10] J. Qiang, "Symplectic multiparticle tracking model for self-consistent space-charge simulation," *Phys. Rev. Accel. Beams*, vol. 20, p. 014203, Jan 2017.

[11] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[12] C. Nvidia, "Programming guide," 2010.

Copyright © 2017 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-182-3