Particle deposition in grid according to Particle-In-Cell Scheme using Graphical Processing Unit

Tunazzina Islam, Kamesh Arumugam, Mohammad Zubair, Desh Ranjan, Balša Terzić, Alexander Godunov
Interdisciplinary Research Team from Dept. Of Computer Science & Dept. Of Physics, Old Dominion University, VA, USA

Introduction

Particle-in-Cell (PIC) scheme has important applications in areas like computational physics, plasma physics, computational fluid dynamics, quantum chemistry and molecular dynamics. Among the most challenging and heretofore unsolved problems in accelerator physics is accurate simulation of the collective effects in electron beams. When electron bunches traveling at nearly the speed of light are forced by accelerator magnets to traverse a curved trajectory, they emit bright ultraviolet or x-ray radiation [1]. If the radiation wavelength is larger than the electron bunch itself, coherent synchrotron radiation (CSR) is produced. CSR leads to a host of deleterious effects, such as emittance degradation and micro-bunching instability, thereby degrading or entirely erasing the electron beam's experimental usefulness. For simulation of collective effects in electron beams that severely degrade beam quality, first step in mitigating the damaging effects of CSR. One of the important parts for simulating CSR and other collective effects in an electron beam using state-of-the art computing platforms is Particle Deposition. This is computationally intensive step of the simulation. We implemented parallel algorithm for particle deposition using Graphical Processing Unit (GPU) with CUDA. Speed up was calculated by comparing parallel method with sequential method.

Method

The beam bunch was sampled by point-particles. The Particle-in-cell method treats each point-particle as a collection of computer particles. As (Number of particles) >> (Number of grids), the execution time was dominated by the charge deposition of the particles.

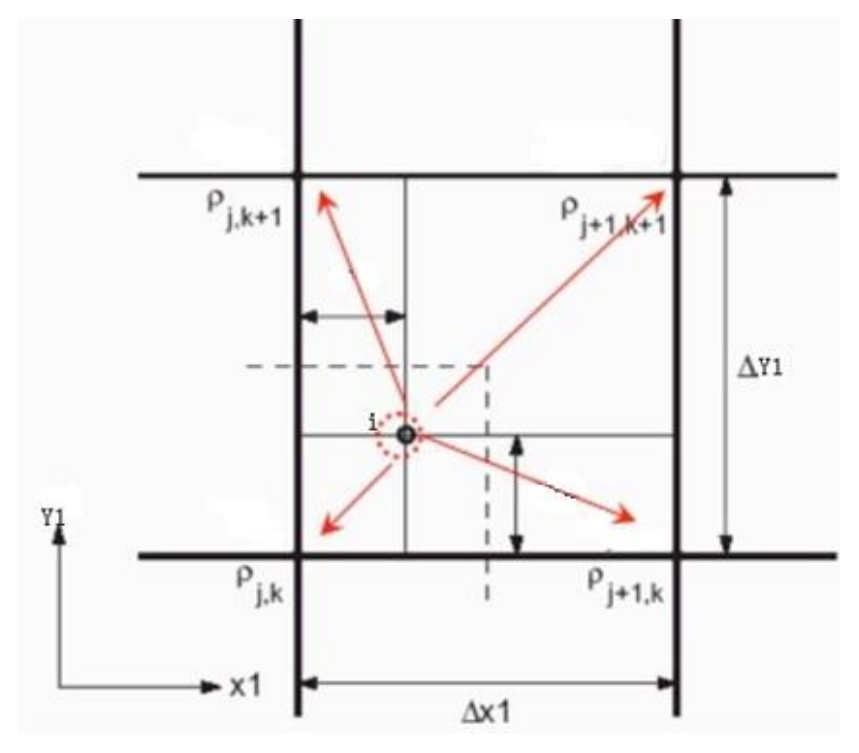

Figure 1. Depositing Charge to the Grids

In Figure1 particle "i" is at position (x1, y1) which belongs within the grid (Δx1, Δy1). The charge density of the particle "i" will be deposited on neighboring grid points (j, k), (j+1, k), (j, k+1) and (j+1, k+1) according to PIC.

Two approaches were followed to deposit particles:

- ❑ One thread handles one particle (Naïve Approach) [2].
  - The charge on particle was split into 4 parts, which were then deposited to the 4 nearest grid points.
- ❑ One thread handles one cell (multiple particles) [2].
  - Sort particles according to cell.
  - Assign a CUDA block to a cell block.
  - Perform a per-block, shared memory, segmented scan to compute density sum for each cell.
  - Sum cached copy to global grid.

Why Two Approaches?

The charge density has a data dependency or data hazard, since particles in different threads can attempt to simultaneously update the same grid point. There are several possible methods to deal with this data dependency.

- ❑ One way to use Atomic operation [3]. But atomic operations are considered to be very slow in the current NVIDA hardware because atomicity prevents parallel execution by stalling other threads in the code segment.
- ❑ Another way to partition memory with extra guards cells [2] so that each thread writes to a different location, then add up those locations that refer to the same grid. No need of Atomic operation.

Performance Analysis

Both sequential code and parallel code were implemented for particle deposition. First we started with 1000 particles and 32 * 32 grids. Then we fixed the grid size and incremented the number of particles up to 10000000 particles. After doing that we fixed the number of particles to 10000000 and incremented the grid size as 64*64, 128*128, 256*256. We Measured execution time for both CPU and GPU as well as the speed up. Correctness of our code was also assured by matching result got from sequential implementation and parallel implementation.

Table 1. Speed up from 1st approach: 1 thread → 1 particle

Number of Particles	Grid Size	Execution Time in CPU (milliseconds)	Execution Time in GPU (milliseconds)	Speed Up (cpu_time/gpu_time)
1000	32*32	0.06992	0.041088	1.7
10000	32*32	0.511264	0.128928	3.97
100000	32*32	4.91392	1.15616	4.25
1000000	32*32	43.0989	10.1378	4.25
10000000	32*32	436.877	99.7978	4.38
10000000	64*64	437.65	46.0646	9.5
10000000	128*128	438.743	32.88	13.34
10000000	256*256	450.719	29.822	15.11

Table 2. Speed up from 2nd approach: 1 thread → 1 cell

Number of Particles	Grid Size	Execution Time in CPU (milliseconds)	Execution Time in GPU (milliseconds)	Speed Up (cpu_time/gpu_time)
1000	32*32	0.085952	0.044032	1.95
10000	32*32	0.26848	0.075232	3.57
100000	32*32	2.15651	0.366272	5.88
1000000	32*32	21.0361	3.11658	6.75
10000000	32*32	210.061	30.2322	6.95
10000000	64*64	209.906	7.91629	26.52
10000000	128*128	218.777	2.00618	109.05
10000000	256*256	224.377	0.550848	407.33

Results and Findings

The massively parallel architecture makes GPUs very effective for algorithms where processing of large blocks of data can be executed in parallel. The computationally intensive nature of PIC requires a high-performance implementation. Various optimization techniques are applied to maximize the utilization of the GPU. For 10000000 particles and 256 * 256 grid in Naïve Approach 15 times speed up (Table 1) and for 2nd approach 407 times speed up (Table 2) have been got compare to sequential method.

Challenge

Particle deposition will occur multiple time steps. The most challenging part of implementing 2nd approach is maintaining the particle order. For first time step we did sorting as a preprocessing. Then rest of time steps we handled sorting different way. As the cost of maintaining the particle order depends on how many particles are leaving a grid, we reduced it by defining a sorting cell to contain multiple grid points. In this case to resolve data hazard Shared Memory for each thread was used.

Conclusion

The first-principle nature of the PIC model determines that PIC simulations require intense computation. Modern GPU provides a significant amount of raw compute power and bandwidth, both about an order of magnitude more than a conventional CPU. Computation of PIC was memory bound.

In future, we have plan to extend the algorithm to gain more speed up and improve performance like avoiding the slow down due to the additional usage of shared memory, to allow a thread to be responsible for more than one cell.

References

[1] B. Terzić, A. Godunov"A New 2D Particle-In-Cell Model For Coherent Synchrotron Radiation Effects on Beam Dynamics".

[2] V. K. Decyk, T. V. Singh, "Adaptable Particle-in-Cell Algorithms for Graphical Processing Units".

[3] NVIDIA CUDA TM Programming Guide Version 3.0., NVIDIA Corporation.