CATEGORY: COMPUTATIONAL PHYSICS - CY06
POSTER P4196
CONTACT NAME Jutta Fitzek: j.fitzek@gsi.de

GPU TECHNOLOGY CONFERENCE

Parallelization of the Particle-in-cell-Code PATRIC with GPU-Programming

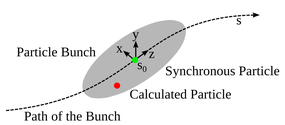
J. Fitzek, S. Appel, O. Boine-Frankenheim *GSI, Darmstadt, Germany*

Abstract

The particle-in-cell code PATRIC (Particle Tracking Code) is used to simulate particles in a circular particle accelerator at the GSI Helmholtz Center for Heavy Ion in Darmstadt, Germany. Parallelization of PIC codes is an open research field and solutions depend very much on the specific problem.
As topic of a diploma thesis, the possibilities and limits for a GPU integration into the existing simulation codes is evaluated. The focus lies on general GPU aspects and on the problems arising from collective particle effects. Aim is clear and maintainable code as well as reuse of existing code where possible. As GPU, the NVIDIA® Tesla C2075 was used. This contribution summarizes the findings.

Beam Dynamic Simulations with PATRIC

- Simulates particle motion, used e.g. to study collective effects in the accelerators
- Inhouse development, modern C++ code with additional python scripts



Particle Motion

Particle Tracking

Background:

- Describes, how particles move through the accelerator
- Magnets represented as matrices, calculation consists of matrix-vector-multiplications and store operations
- Here: linear optics used

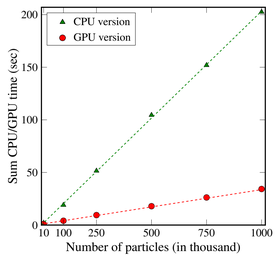
$$v' = M \cdot v = \begin{pmatrix} M_{11} & M_{12} & 0 & 0 & 0 & M_{16} \\ M_{21} & M_{22} & 0 & 0 & 0 & M_{26} \\ 0 & 0 & M_{33} & M_{34} & 0 & 0 \\ 0 & 0 & M_{43} & M_{44} & 0 & 0 \\ M_{51} & M_{52} & 0 & 0 & 1 & M_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ x' \\ y \\ y' \\ z \\ v \end{pmatrix}$$

Parallelization:

- Particles are independent, good parallelization possible
- CGMA=3 ! ⇒ Code is memory-bound
- Structure of Arrays (one array per coordinate)
- One thread per particle or per particle coordinate?

Results for parallelized tracking:

- Solution with each transport step separately on the GPU:
  - One particle per thread slightly better than one coordinate per thread (latter gives more load on the GPU)
  - Synchronization has almost no impact
  - Data copy dominates with 1/4 of the execution time
- Solution with particles as long as possible on the GPU:
  - Speedup of 6 compared to the original CPU version
  - Only copy particles back, when needed for output. Additional array and procedures needed on the GPU to be able to loose particles.

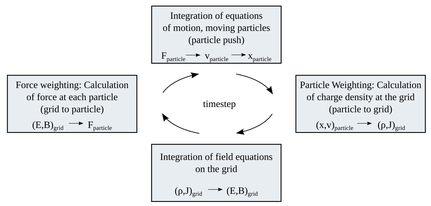


Particle Tracking on the GPU, Speedup: 6x

Collective Effects

Background:

- Describe particle interaction
- Forces are calculated on a grid and act back on the particles
- Parameters used:
  1d grid, 512 grid points,
  250000 particles



PIC Calculation Cycle (see [2])

- Runtime of the serial version: 85% interpolation, 4% particle push, 1% calculation of forces

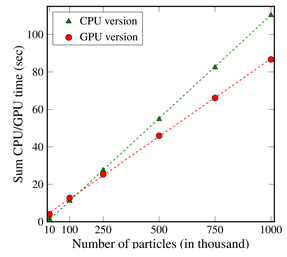
Parallelization of each algorithm step (see also [1]):

1. Interpolation particles → grid (scatter): scales with number of particles, problem that many particles need to update the same grid point (most problematic step for parallelization!)
2. Caculation of fields (field solver): scales with number of grid points, FFT library can be used
3. Interpolation grid → particles (gather): Scales with number of particles, but easier, since each particle is only updated once. Better data locality if the same grid information can be used for several particles.
4. Determine new particle positions (particle push): Scales with number of particles, simple parallelization.

⟹ General problem: load balancing vs. data locality

Results for parallelization of collective effects:

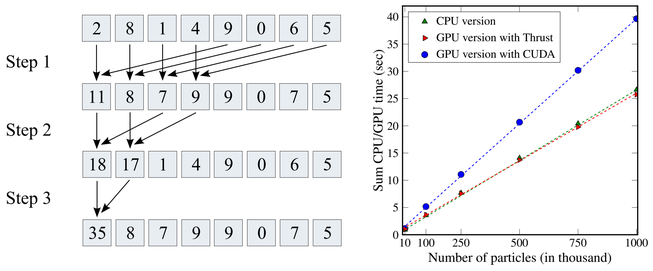
- Solution with presorting of particles:
  - Simple sorting in each algorithm cycle is much too slow (9x) (`thrust::sort_by_key`, `thrust::lower_bound`, `thrust::transform`)
  - More sophistiacted sorting would be needed
- Solution with atomic updates:
  - Update grid information from each particle (`atomicAdd`)
  - Speedup of 1.19
  - Basis for future development



Collective effects on the GPU

General Aspects

- Particle loss / max. branch divergency : only small performance loss !
- Floating point single vs. double: only 5% performance loss, keep double calculations to not run into numerical problems
- Usage of streams: no performance gain, because code is memory bound
- Output of intermediate data: very costly, since data needs to be copied back to the host. Avoide, if possible.
- Calculation of beam emittance $\epsilon_x = \sqrt{\langle x^2 \rangle \langle x'^2 \rangle - \langle x x' \rangle^2}$
  Reduction operations are costly ⇒ keep those calculations on the CPU!



Beam Emittance: CUDA vs. Thrust vs. CPU, Thrust slightly faster

Summary

- Good results for particle tracking (speedup 6x), since particles can be treated independently.
- Collective effects are more difficult to accelerate, only slightly faster than CPU. Version with atomic updates seems more promising for the future.
- Since the code is memory bound, path divergence due to particle loss is not an issue. Also, double precision can be used and is only 5% slower.

Outlook

- Integration of MPI and GPUs in the future HPC system
- Keep track of newest GPU development, esp. possibilities to eliminate the data copy between host/device

[1] Carmona, E.A.; Chandler, L.J.: On parallel PIC versatility and the structure of parallel PIC approaches. In: Concurrency: Practice and Experience 9 (1997), Nr. 12.

[2] Birdsall, C.K.; Langdon, A.B.: Plasma Physics via Computer Simulation. New York, NY, USA : Taylor and Francis Group, 2005