GPU implementation of the Q-VLPL3D plasma simulation code

A.V.Snytnikov[1],A.M.Pukhov[2], M.M.Lavrentiev[3,4]

[1]Kaliningrad State Technical University

[2]University of Duesseldorf

[3]Novosibirsk State University

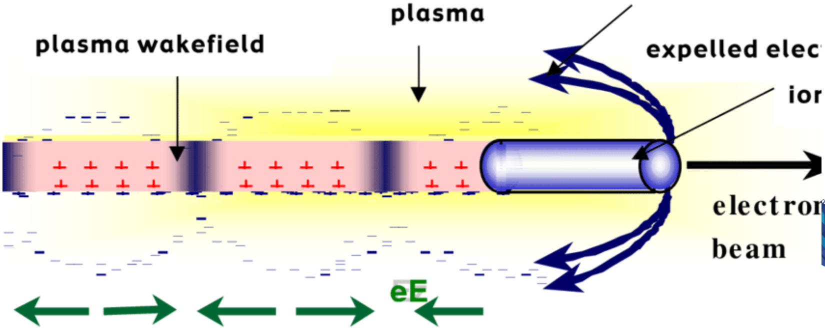
[4]Institute of Automation and Electrometry SB RAS

VLPL code

- Virtual Laser-Plasma Lab
  Pukhov, A. Three-dimensional electromagnetic relativistic particle-in-cell code VLPL (virtual laser plasma lab). Journal of Plasma Physics 61, 425 (1999)
- particle-in-cell platform
- multi-dimensional simulations of relativistic plasmas.
- the full electromagnetic code VLPL
- the hybrid version and the
- the quasi-static version Q-VLPL3D

Physical probem: Wakefield acceleration

- Wakefield acceleration is widely studied both numerically and experimentally due to the possibility of building small but powerful particle accelerators.
- What's going on: a powerful beam of ions drives out electrons while passing through plasma.
- The result are the extremely high local electric fields
- the fields may be (and are) used for particle accelerators.



The model

$$\begin{aligned}\nabla \times \vec{B} &= \frac{1}{c}\frac{\partial \vec{E}}{\partial t} + \frac{4\pi}{c}\vec{j} \\ \nabla \times \vec{E} &= -\frac{1}{c}\frac{\partial \vec{B}}{\partial t}\end{aligned} \tag{1}$$

And particle equations

$$\begin{aligned}\tfrac{\partial \vec{p}}{\partial t} &= q\vec{E} + \frac{q}{c\gamma} p \times \vec{B} \\ \gamma &= \sqrt{1 + \frac{p^2}{(mc)^2}}\end{aligned} \tag{2}$$

Particle motion equations: Quasistatic method

In the quasistatic model the beam is assumed to be changing very slowly, so the equations of motion for model particles of plasma have the following form:

$$\begin{aligned} \frac{\partial \vec{p}}{\partial t} &= \frac{\vec{E} + \vec{v} \times \vec{B}}{1 - v_x} \\ \frac{\partial y}{\partial t} &= \frac{-v_y}{1 - v_x} \\ \frac{\partial z}{\partial t} &= \frac{-v_z}{1 - v_x} \end{aligned} \tag{3}$$

Sequence of computations

The general sequence of computations in PIC codes is the following:

1. Evaluate the mesh values of charge density ρ and current $\vec{j}$
2. Compute the eletromagnetic fields, $\vec{E}$ and $\vec{H}$ for the next moment of time.
3. Push the particles.

For **quasistatic PIC codes** step 3 is performed in two substeps.

1. there are two sorts of particles: beam particles and plasma particles.
2. The pushing of beam particles is performed in the common PIC way, as described in e.g. .
3. The equations of motion for beam particles are the usual Newtonian motion equations.
4. For plasma particles the different motion equations are used in the form presented above.

Layers of plasma particles

$$\begin{aligned}\frac{\partial \vec{p}}{\partial t} &= \frac{\vec{E} + \vec{v} \times \vec{B}}{1 - v_x} \\ \frac{\partial y}{\partial t} &= \frac{-v_y}{1 - v_x} \\ \frac{\partial z}{\partial t} &= \frac{-v_z}{1 - v_x}\end{aligned} \tag{4}$$

One can notice that there's no equation for X coordinate. This is because in the quasistatic model X is the time-space coordinate, and the particles just cannot move in the negative direction because it means moving backward in time. So plasma particles are grouped in layers, each layer having some X coordinate. Number of cells in a layer is equal to $N_Y \times N_Z$.

Performance

Profiling shows that the most of the worktime is taken by the following code fragments:

- Beam particles push
- Plasma particles push
- Solving Poisson equation by means of 2D FFT

GPU implementation

In order to build GPU implementation of Q-VLPL3D, the above mentioned fragments of code were rewritten in a way more suitable for GPU:

- Data format has been changed, which allowed to ensure the localization of data. Q-VLPL3D code uses classes to store particle data, as shown in listing. This is not the best choice for GPU memory. An array or list of class intstance will lead to frequent cache misses within GPU warps.
- In order to facilitate the GPU memory use, we organize the particle data in the so called CUDA surface (a data format originally introduced to optimize 2D graphical data processing).
- Performance limitations were investigated for the beam pushing kernel with Nvdia Performance analysis tools. The limitations are: bad data locality, non-optimal data access pattern and the main constraint is the number of registers.

GPU implementation of Poisson solver

- Another problem of GPU implementation of Q-VLPL3D code was the need to implement real-to-real half-integer FFT.
- There is no such transform in the cuFFT library.
- A real implementation of this transform was performed using CUDA surface memory, which is faster then the global GPU memory.
- Since cuFFT library lacks 2D real-to real transform.

GPU performance

FFT kernel		Beam push kernel		Mesh size			beam particles per cell
% of total time	Runtime, ns	% of total time	Runtime, ns	N_X	N_Y	N_Z	
25.9	96032	74	272894	120	64	64	19
12	52160	65.3	284254	120	128	128	19
0.7	51996	97.5	5954866	500	128	128	500
1.2	84130	97.5	5988467	1000	128	128	500
0.5	93694	99.8	103857564	1000	128	128	10000

Comparing GPU and CPU performance

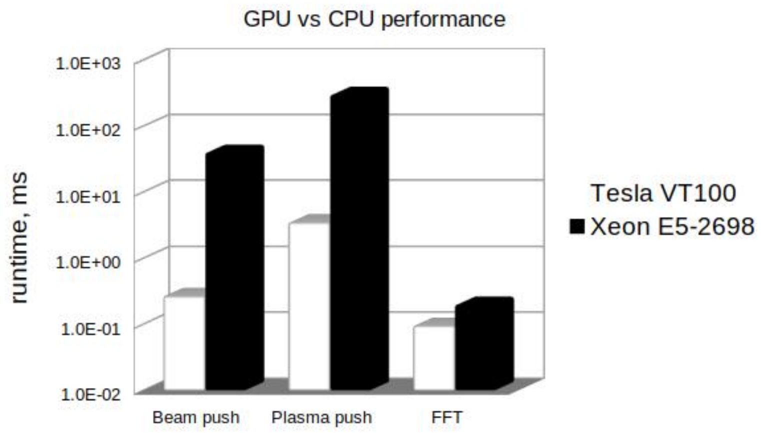


Рис.: Runtime (in milliseconds) for main operations in the code.

Volta and Ampere

- The performance of the GPU implementation was tested
- with recent Volta and Ampere GPUs resulting in the following performance (for Volta):
- 277 microseconds for pushing beam particles,
- for 19190 particles total, 3.5 ms is the duration of plasma particles push for 16384 particles total per one layer,
- and the 2D FFT is performed in 80 microseconds for 64 $\times$ 64 array.

Maximal size of the model

- The largest model used in computations with GPU has the mesh size $1000 \times 128 \times 128$ nodes and 10,000 beam particles in a cell.
- Let us point it once more that it is for a single Tesla VT100 Volta GPU with 16Gb RAM onboard.
- For the newest Ampere GPU with 80 Gb RAM one can use even larger models.
- In such a way, the coarse-grain MPI parallelization is not necessary any more for this code.

Conclusion

- One can see than there is some speedup for the GPU version of VLPL code compared to CPU performance.
- But the speedup itself is not very important. GPU implementation enables to perform large scale simulation in reasonable time without using MPI parallelization.
- One can also consider the spatial size of a CPU cluster necessary for MPI program.
- the spatial size of Tesla VT100 as well as the energy consumption of both.