Workshop on Numerical Modeling in MHD and Plasma Physics: methods, tools, and outcomes.

High performance collisional PIC plasma simulation with GPU-based clusters

A.V.Snytnikov,
Institute of Computational Mathematics and Mathematical Geophysics SB RAS
A.A.Romanenko
Faculty of Information Technologies, NSU
G.G.Lazareva,
Peoples' Friendship University of Russia (RUDN University), Moscow, Russian Federation

Why?

Quality of Particle-In-Cell simulation crucially depends on the number of particles so it is important to be able to process as many particles as possible

The present Particle-In-Cell performance

0.5 TFLOPS

with GPU Tesla V100 “Volta”,
6.4 million model particles processed in 0.006 sec.,

or

1 Giga-particle per second

Main equations

$$\frac{\partial f_{i,e}}{\partial t}+\vec{v}\,\frac{\partial f_{i,e}}{\partial \vec{x}}+\vec{F}\,\frac{\partial f_{i,e}}{\partial \vec{v}}=0$$

$$\nabla\times\vec{B}=4\pi\,\vec{j}+\frac{1}{c}\frac{\partial\vec{E}}{\partial t}$$

$$\nabla\times\vec{E}=-\frac{1}{c}\frac{\partial\vec{B}}{\partial t}$$

$$\nabla\vec{E}=4\pi\rho$$

$$\nabla\vec{B}=0$$

$$\vec{p}=\gamma\,\vec{v}\,,\gamma^{-1}=\sqrt{1-v^2}$$

$$\vec{F}=q_{i,e}\left(\vec{E}+\frac{1}{c}[\vec{v},\vec{B}]\right)$$

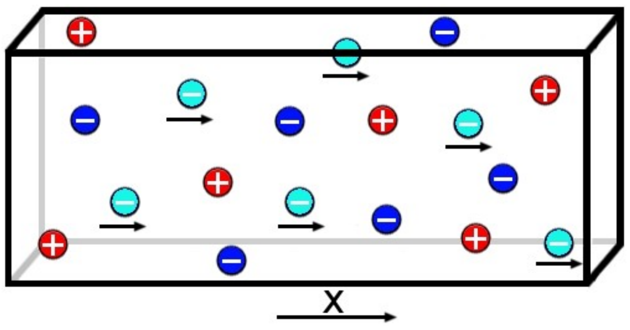
$$\vec{j}=\sum_{i,e}q_{i,e}\int f_{i,e}\vec{v}\,d\,\vec{v}$$

$$\rho=\sum_{i,e}q_{i,e}\int f_{i,e}d\,\vec{v}$$

- **Boundary conditions:** periodic

Particle-In-Cell basics

In Particle-In-Cell method plasma is simulated by means of the model particles. Motion equations of model particles are the characteristic equations of Vlasov equation.



$$\frac{\partial p_{i,e}}{\partial t} = \kappa\left(E + [v, B]\right),$$

$$\frac{\partial r_{i,e}}{\partial t} = v_{i,e}.$$

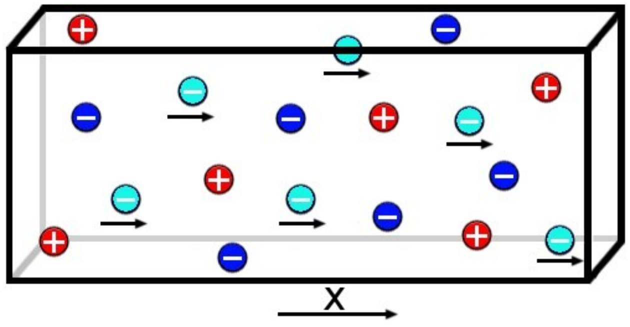
$$p_{i,e} = \frac{v_{i,e}}{\sqrt{1 - v_{i,e}^2}}, \quad \kappa_e = -1, \quad \kappa_i = m_e / m_{i.}$$

- Motion equations for model particles ar solved with leapfrog scheme
- Maxwell equations are solved with FDTD method
- Computational complexity: 500 floating point operations per particle
- Quality of the solution crucially depends on the number of particles

Physical validation of the code:

Two-stream instability simulation,

thanks to Dr. Ekaterina Genrikh



$$x \in [0, L], \quad y, z \in [0, n h_x]$$

$$k = 2\pi / L \longrightarrow W \sim e^{2\gamma t}, \gamma = \frac{1}{2} \frac{\partial \ln W}{\partial t}$$

$$f(v) = \frac{1}{\Delta v \sqrt{2\pi}} \exp - \frac{(v - v_0)^2}{2 \Delta v^2}$$

$n_b -$ Beam density

$2(\Delta v)^2 -$ Beam temperature

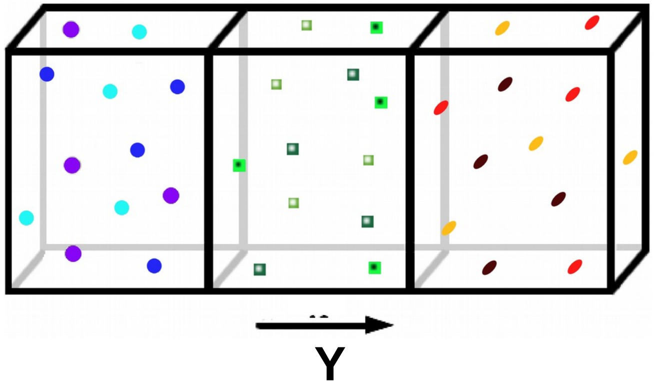
Hydrodynamical mode $(k \Delta v \ll \gamma)$ $n_b = 2 \cdot 10^{-3}, \ \Delta v = 0.035$

Transition mode $n_b = 10^{-3}, \quad \Delta v = 0.14$

Kynetic mode $(k \Delta v \gg \gamma)$ $n_b = 2 \cdot 10^{-4}, \ \Delta v = 0.14$

Computation parameters

Domain length	$L = 1.2566, \ 1.1424$
Spatial mesh	100x4x4
Timestep	$\tau = 0.001$
Particles per cell	$lp = 50...20000$

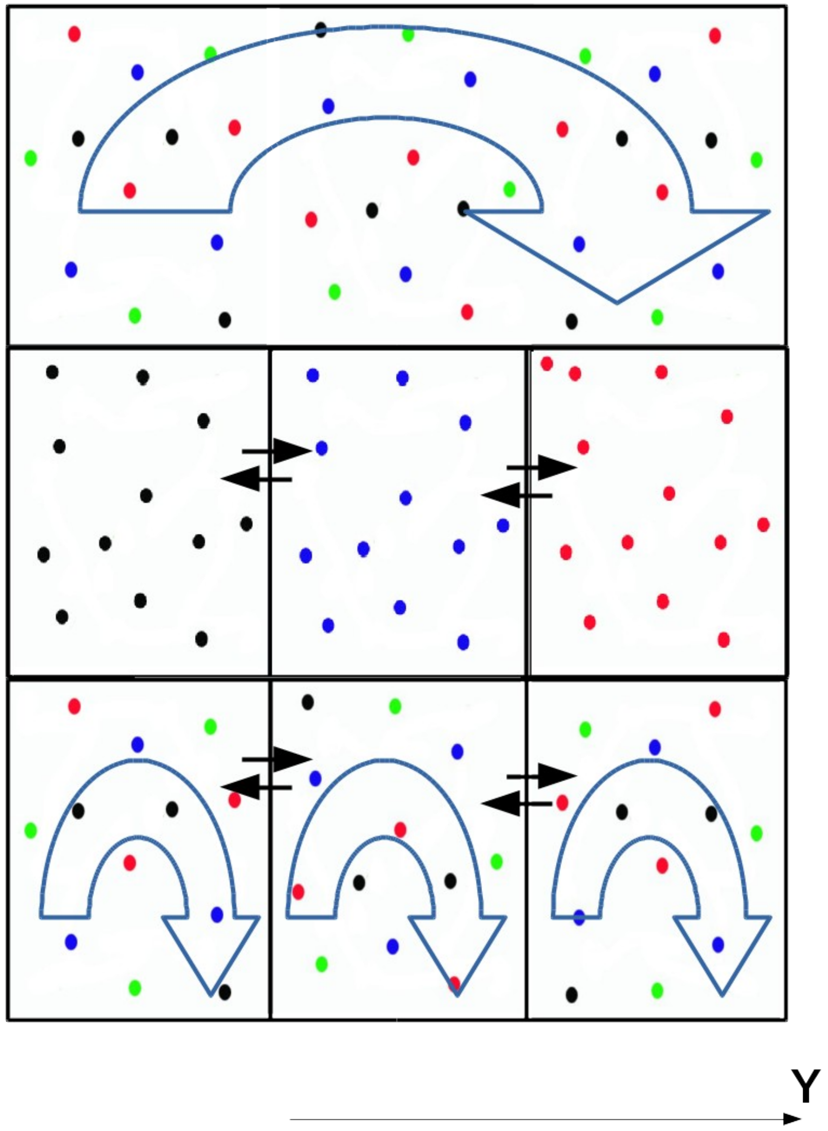


Domain Decomposition

Lagrange decomposition

Эйлерова декомпозиция

Mixed Lagrange-Euler decomposition



- Particles belonging to different processors (or cores, or GPUs) are shown with different colours

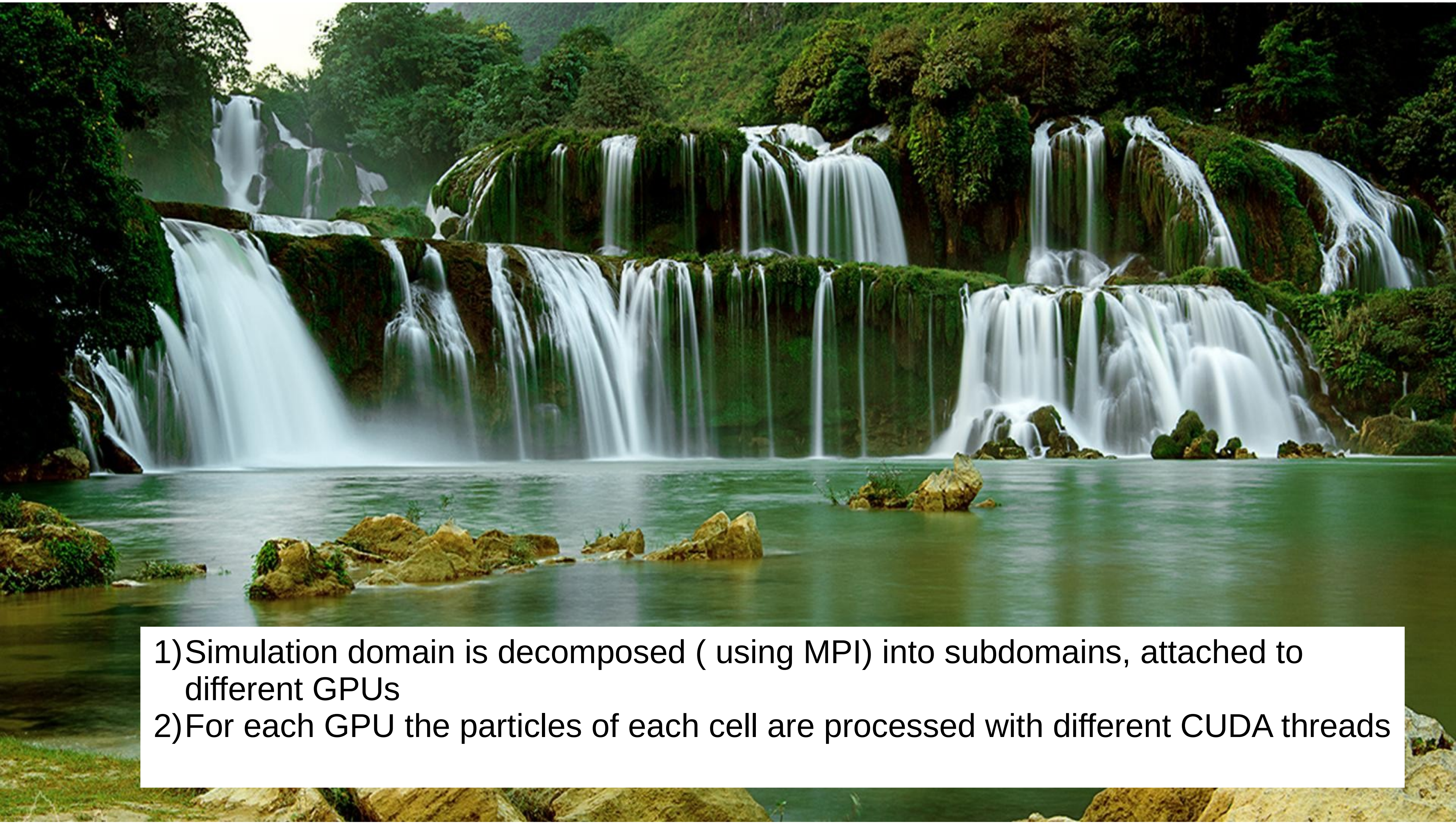
- Collective MPI operations (MPI_Allreduce are shown with large hollow arrows)

- Peer-to-peer MPI operations (send/recv) are shown with small black arrows

GPU implemenation

- The particles are stored in small array attached to cells
- Particle push is performed with shared memory
- The transit of particles from one cell to another is performed with no synchronization

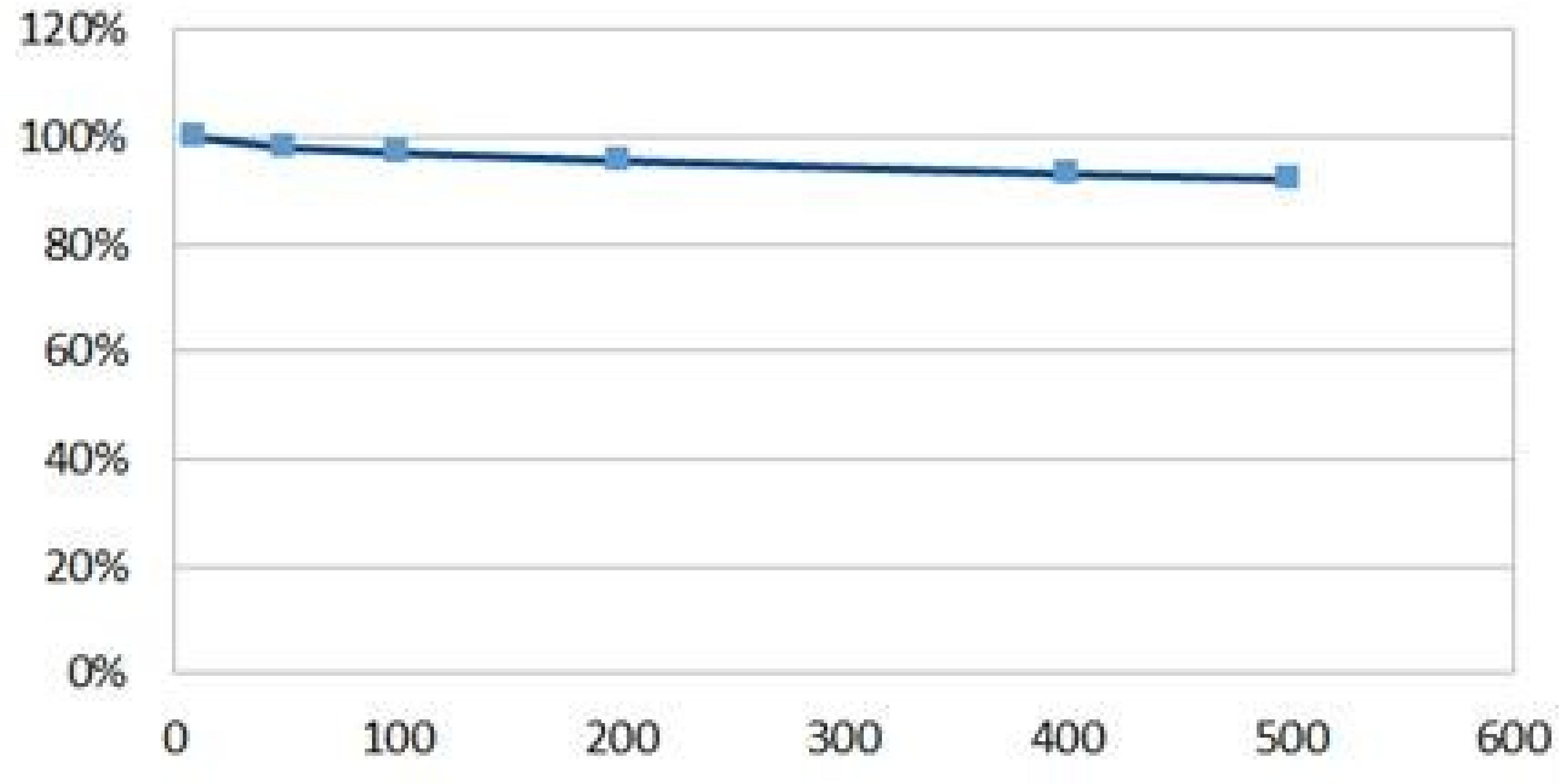
The two-level multi-flow computation



1) Simulation domain is decomposed (using MPI) into subdomains, attached to different GPUs
2) For each GPU the particles of each cell are processed with different CUDA threads

Weak efficiency

Lomonosov supercomputer, MSU

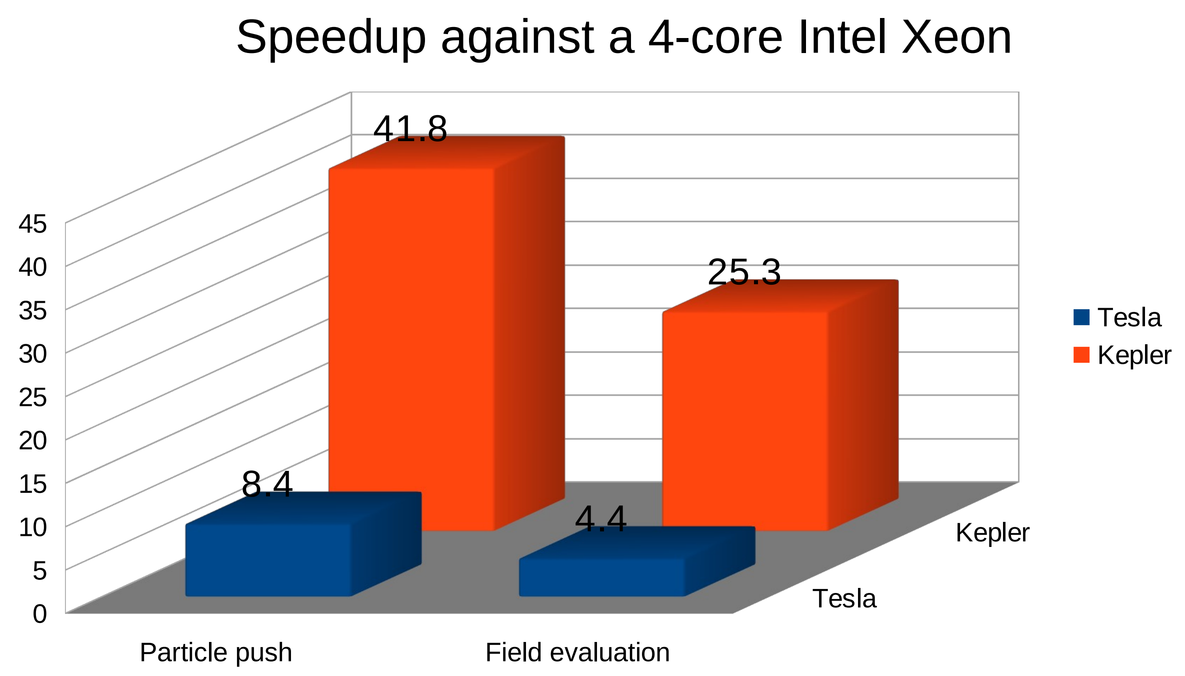


Number of GPUs

Mesh: 100x4x4 nodes, 6.4 million particles, MPI_Allreduce

Particle push

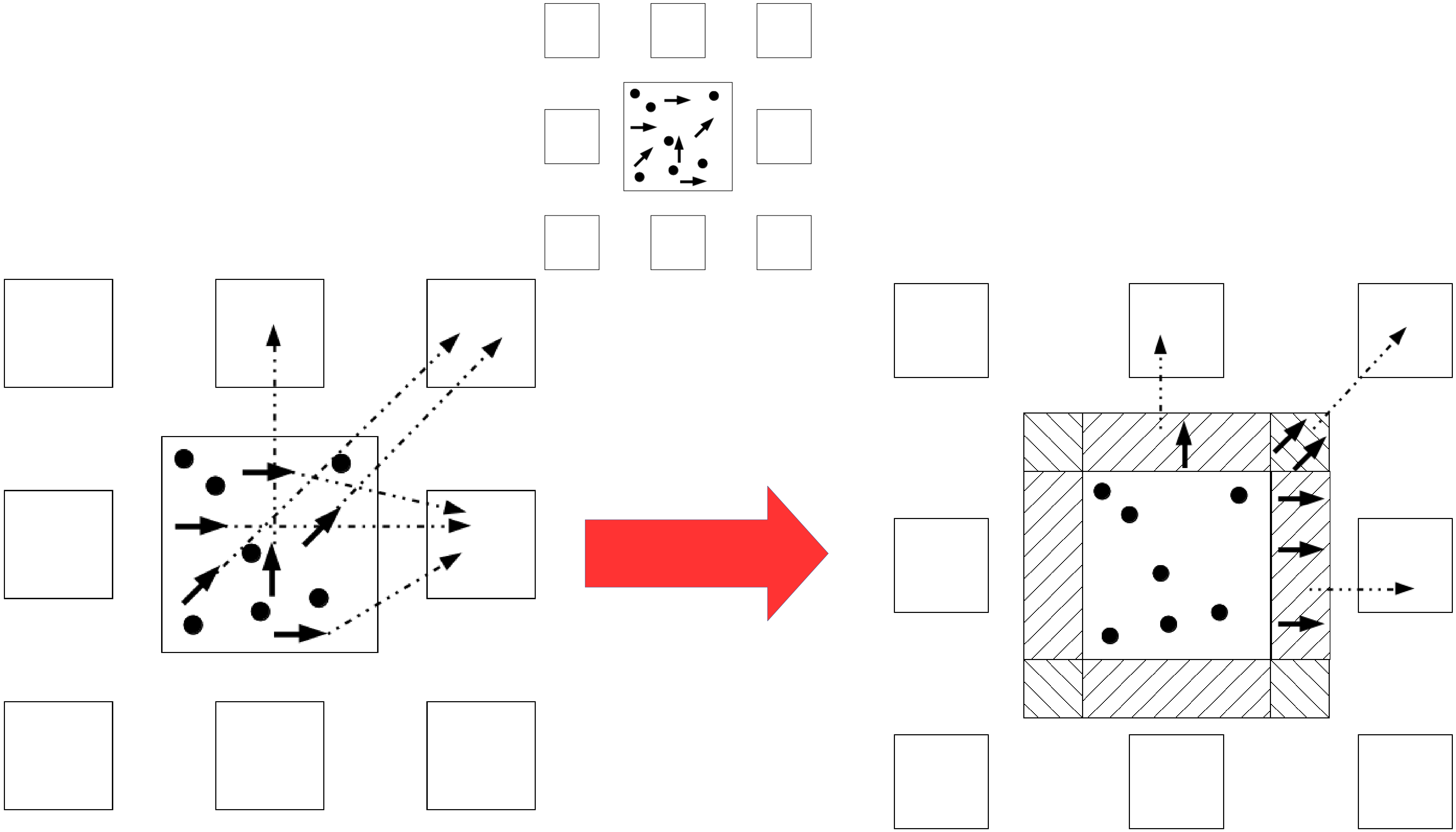
- Particle push is the most expensive part of the code (up to 90 % of the total time)
- Fortunately, particle push is accelerated the best with GPU



Recent optimizations

- Transit of model particles from cell to cell with no synchronization using buffers
- Get rid of buffers using particle labels
- Get rid of atomic operations when computing current within a cell

Transit of model particles from cell to cell with no synchronization using buffers



Get rid of buffers using particle labels

- Instead of moving into buffer the particle is given a label (“which direction it goes”)
- Then neighbour cells read their particles from the particle list of the present cells with no read-and write conflict
- After that the flow-away particles are removed from the list by a separate kernel
- In such a way, send buffers (quite big) are not necessary

Get rid of atomic operations when computing current within a cell

The particles are stored in small array attached to cells

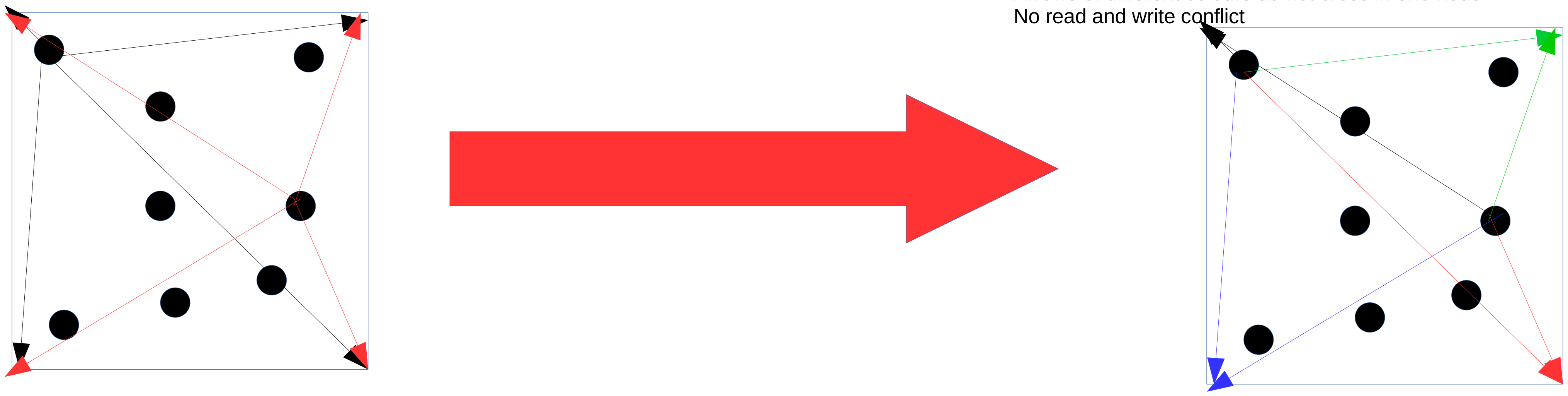
Current a computed in each cell asynchronously by CUDA threads, one thread for one particle

Each thread **reads** the present value of the current in a mesh node, adds the current of the particle and **writes** the result into the mesh node (reading and writing to different places by different threads at once)

In this case the read-and write conflict occurs and atomic operations are necessary

If we do the opposite and attach a CUDA thread to the mesh nodes instead of particles,

Then the thread reads the current value from each particle and writes to its own memory (reading from different places but writing to one place only for each thread)


Arrows of different colours do not cross in one node.
No read and write conflict