ILMATIETEEN LAITOS
METEOROLOGISKA INSTITUTET
FINNISH METEOROLOGICAL INSTITUTE

# Six-dimensional Vlasov simulations with GPUs

QuESpace team

Daldorff, L.K.S., Honkonen, I., Palmroth, M., Sandroos, A.

# Introduction

The Vlasov equation describes the dynamics of charged particle motion with a six-dimensional distribution function *f*:

$$\partial_t f(\boldsymbol{x},\boldsymbol{v},t)+\boldsymbol{v}\cdot\nabla_x f(\boldsymbol{x},\boldsymbol{v},t)+\boldsymbol{a}\cdot\nabla_v f(\boldsymbol{x},\boldsymbol{v},t)=0$$

The acceleration **a** is given by the Lorentz force:

$$\boldsymbol{a}=(q/m)(\boldsymbol{E}(\boldsymbol{x},t)+\boldsymbol{v}\times\boldsymbol{B}(\boldsymbol{x},t))$$

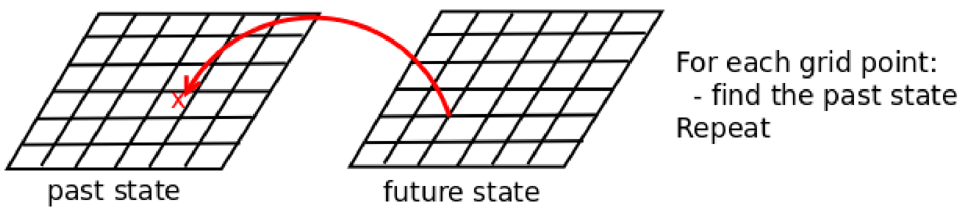
Vlasov eqn. is coupled with Maxwell's equations, which are solved using a separate field solver. Particle moments and Ohm's law are the necessary source terms:

$$\nabla\cdot\boldsymbol{E}(\boldsymbol{x},t)=0$$
$$\nabla\cdot\boldsymbol{B}(\boldsymbol{x},t)=0$$
$$\nabla\times\boldsymbol{E}(\boldsymbol{x},t)=-\partial_t\boldsymbol{B}(\boldsymbol{x},t)$$
$$\nabla\times\boldsymbol{B}(\boldsymbol{x},t)=\mu_0\boldsymbol{j}(\boldsymbol{x},t)$$

$$n(\boldsymbol{x},t)=\int f(\boldsymbol{x},\boldsymbol{v},t)\,d^3v$$
$$\boldsymbol{j}=q\,n(\boldsymbol{x},t)\int f(\boldsymbol{x},\boldsymbol{v},t)\boldsymbol{v}\,d^3v$$
$$\boldsymbol{E}=-\boldsymbol{V}\times\boldsymbol{B}$$

# Solving Vlasov: semi-Lagrangian

The Vlasov eq. is an equation for the motion of incompressible fluid. Often it is solved by using a semi-Lagrangian solver, which follows the characteristic curves (=”particle trajectories”):




In order to get good results, 6D cubic splines should be used for interpolation. Memory requirements are quite heavy, 4096 spline coefficients per grid point..

Lars Daldorff took a look into these algorithm, and in principle they should work with GPUs (matrix methods).

# Solving Vlasov: FVM

The plan is to use a finite volume method (FVM) which are also used, e.g. for hydrodynamics. FVM schemes use volume averages instead of point values:

$$\partial_t f^{i,j,\ldots} = -[F_x^{i+1} - F_x^i]/(\Delta x) - [F_y^{j+1} - F_y^j]/(\Delta y) - \ldots$$

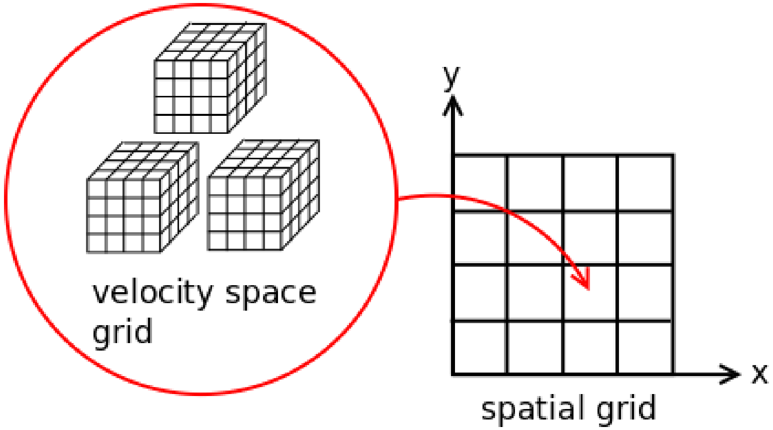
The fluxes, evaluated at cell faces, take on simple expressions:

$$F_x = v_x f(\boldsymbol{x}, \boldsymbol{v}, t)$$
$$F_{v_x} = a_x(\boldsymbol{x}, \boldsymbol{v}, t) f(\boldsymbol{x}, \boldsymbol{v}, t)$$

However, FVM schemes are plagued by diffusion, and eventually adaptive mesh refinement is needed. In preparation the FVM scheme is implemented using a block cartesian grid.

# Solving Vlasov: FVM

Spatial and velocity grids are separated: each spatial cell contains its own block-based velocity grid. Grids can adapted separately if needed.




A leapfrog-type splitting scheme: first calculate the acceleration, then spatial translation.

# Solving Vlasov: FVM

Acceleration:

1. Derivatives
2. Fluxes
3. Propagation
   - Send avgs to spat. Nbrs.

Neighbor data in same spatial cell but in different blocks → data available at every step.

Translation:

1. Derivatives
   - Send derivatives
2. Fluxes
   - Send fluxes
3. Propagation
   - Send averages

Neighbor data in corresponding velocity blocks but in different spatial cells.

# On the problem

We estimate that the smallest possible velocity grid is $40^3$ cells, i.e. $10^3$ 4x4x4 blocks.

GUMICS-4 uses about 300 000 spatial cells for the magnetosphere, we may need more (need to resolve gyro motion).

Few hours (in physical time) need to be simulated: time step is 0.1 s or less → 180 000 time steps.

These numbers add up to ~ $10^{15}$ calculated cells (5 hours). If a GPU propagates $20 \cdot 10^6$ cells per sec, 2000 GPU days are needed.

Memory requirement is ~200 GB.

# Present status

A working single-GPU CUDA code exists:

- Rigorous testing on numerical algorithm need to be done
- Parallelization + I/O
- Can be optimized a bit more
- AMR