

PIC Algorithm, iPIC3D Code and its Application to Study Magnetic Reconnection

Stefano Markidis

High Performance Computing and Visualization Department
KTH Royal Institute of Technology



Oct 1 2014 - KULeuven

Outline

- What are PIC Simulations ?
- Particle and Particle-in-Cell methods
 - Number of particles, Δx , Δt
- The iPIC3D code
 - Installation, running , inputfiles and output
- Exercises in class

What are PIC Simulations ?

- Solving numerically a large number of ODE equations (particle) and PDE (field). Coupling between ODEs and PDE equations are provided by an operation called interpolation
- Variables are (time/space) differenced/discretized.
 - Space \rightarrow Dx, Time \rightarrow DT
 - $U(\text{time}) \rightarrow U^n(\text{space}) \rightarrow U_j^n$
- Discretization introduces errors proportional to Dx and Dt (Accuracy defines how big is this error)
- If this error remains bounded in time then the scheme is stable, if it grows it is unstable.
- If conservation laws (Energy, momentum) are not satisfied, some spurious effect is introduced in the simulation

Particle Algorithms

- Use particle to represent plasma particles: electrons (negative small particles) and ions (positive charged larger particles).
- Difference in mass between electron and ion is typically reduced, i.e. $m_i/m_e = 64$, or 256, ... → this allows to complete “smaller” simulation.
- Several studies to assess the use of “unphysical” mass ratios between electrons and ions → details changes but same overall behaviors.

Particle Algorithms

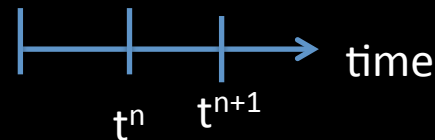
All particle methods solve the equation of motion (ODE) for each particle:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}$$

$$\frac{d\mathbf{v}}{dt} = \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B})$$



DISCRETIZATION



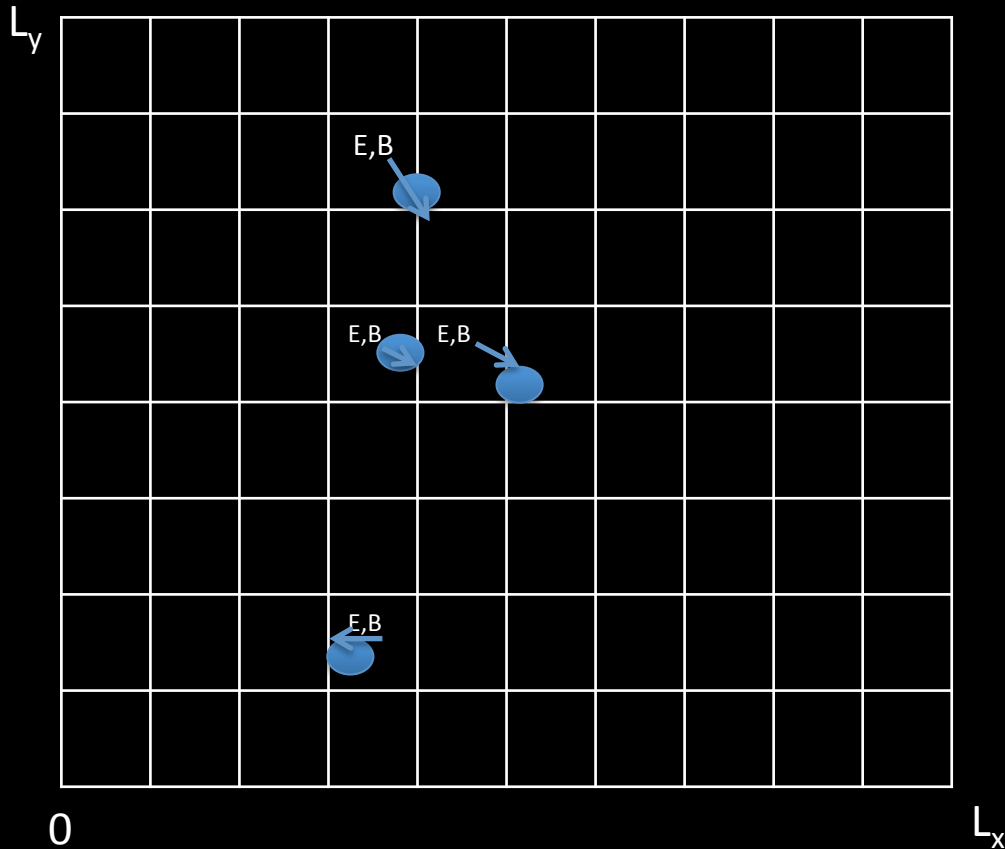
$$\frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t} = \mathbf{v}^n$$

This stage is called
mover

$$\frac{\mathbf{v}^{n+1} - \mathbf{v}^n}{\Delta t} = \frac{q}{m}(\mathbf{E}^n + \mathbf{v}^n \times \mathbf{B}^n)$$

How do we
calculate the
electric and
magnetic field
acting on
particles ?!

Particle-in-Cell Algorithms



The electric and magnetic fields are defined on each center (or node) of the cells of a grid.

Depending on which cell the particle is located, the E and B acting on the particle are the ones defined at the center cell (or a combination of the values of neighbor cells).

This step is called **interpolation grid** → particles and is typically part of the **mover** stage

How do we Calculate E and B on the grid?

We discretize in space and time the Maxwell's equations on the grid and solve them using a linear solver (typically iterative Krylov solvers, i.e. GMRes or CG).

This stage is called

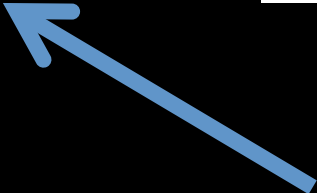
field solver

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}$$



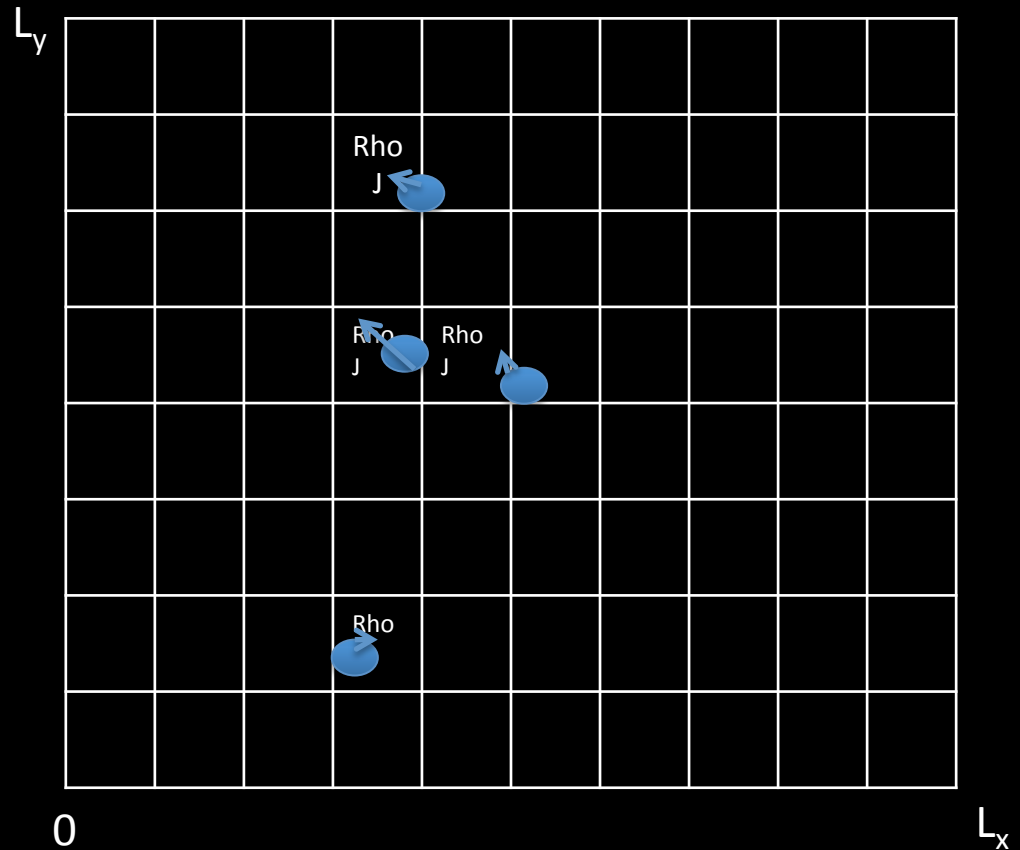
But before that, we need the \mathbf{J} and the ρ defined on the grid ?!

J and rho on the Grid ?

We use the trick we used to calculate the E and B acting on the particles (interpolation grid particle).

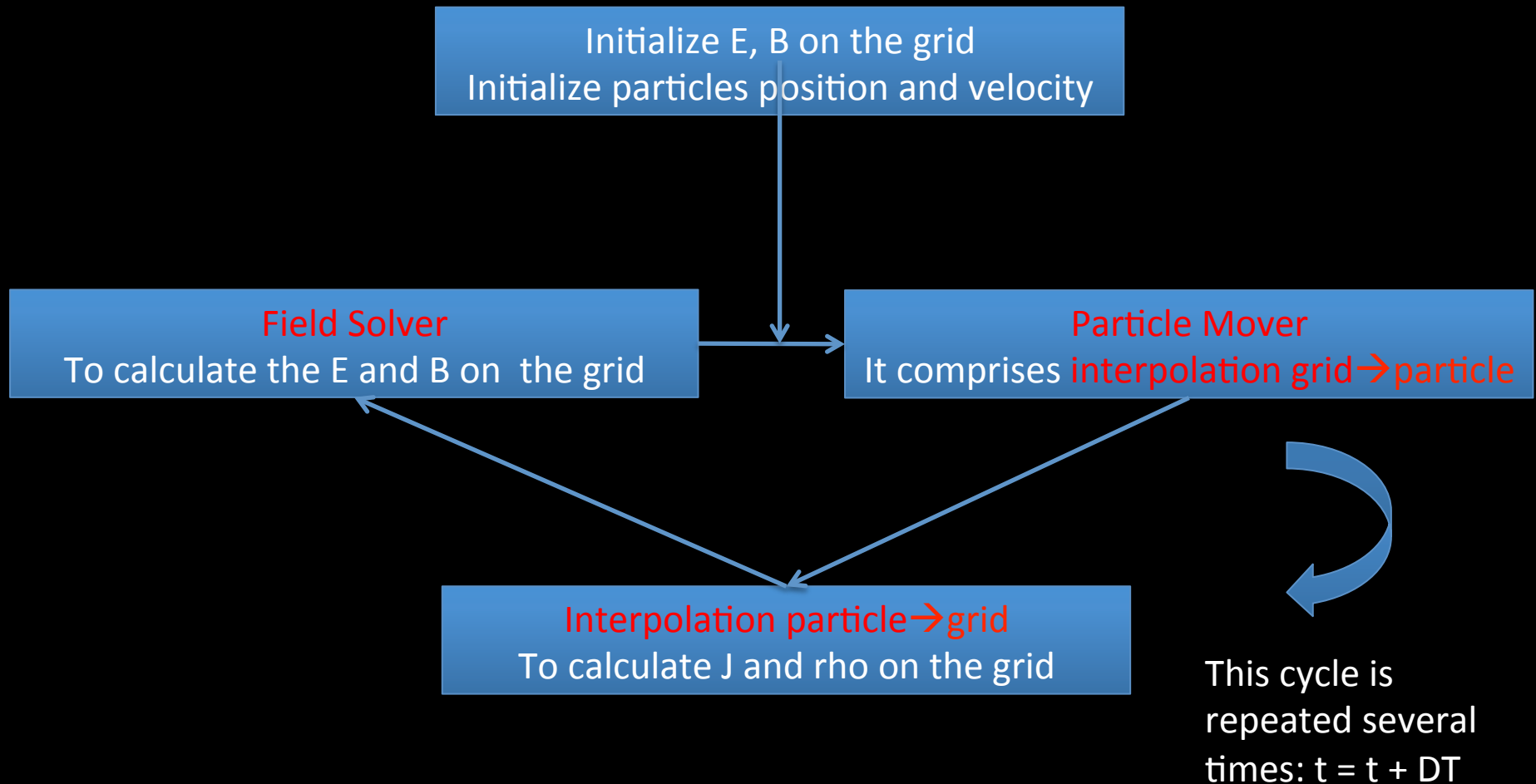
Each particle in a cell contributes to rho in the cell as q/Vol and to J in the cell as $q*v/Vol$

For each particle, we check in which cell is located, and deposit its charge and current densities on the cell. This stage is called interpolation particle \rightarrow grid.



PIC Algorithm

Putting all together ...



iPIC3D

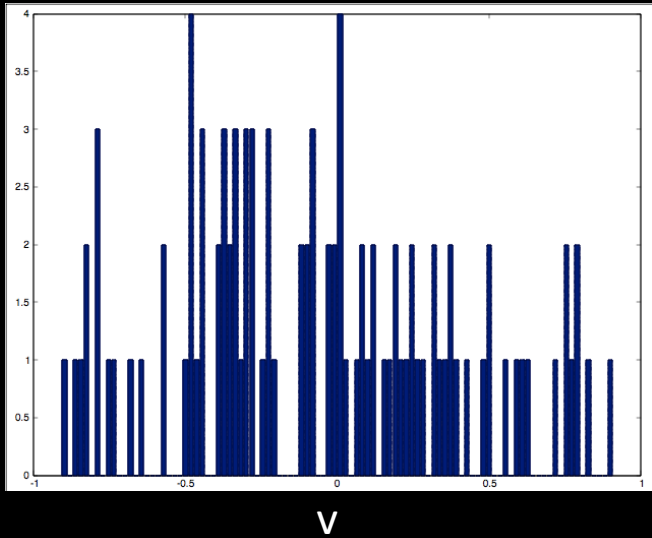
- iPIC3D is a PIC code implementing the PIC algorithm.
- The “i” refers to the discretization in time of the Maxwell’s equations (implicit in time).
- Designed for running on supercomputers but I am providing a version of the code that doesn’t need libraries for supercomputers
- It is in 3D but used in a 2D configuration today.

How Many Particles ?

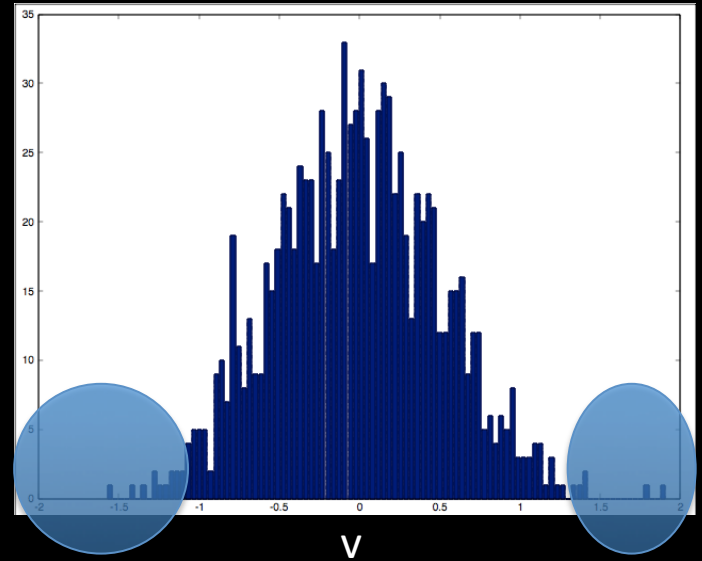
- To solve the PIC discretized equations is equivalent to solve the discretized Vlasov equation (equation for f) + Maxwell's equation
- Particles provides a statistical representation of f . More particles \rightarrow better representation of f .
- Difficulties of representing f at high velocities (called “tails” of f) because we have few particles at high velocities. Resonance phenomena might not be present if not enough particles.

Maxwellian with Particles ($v_0=0$, $v_{th}=0.5$)

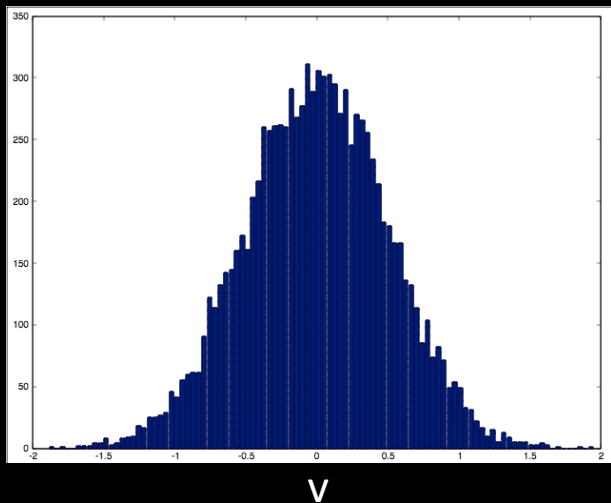
N=100



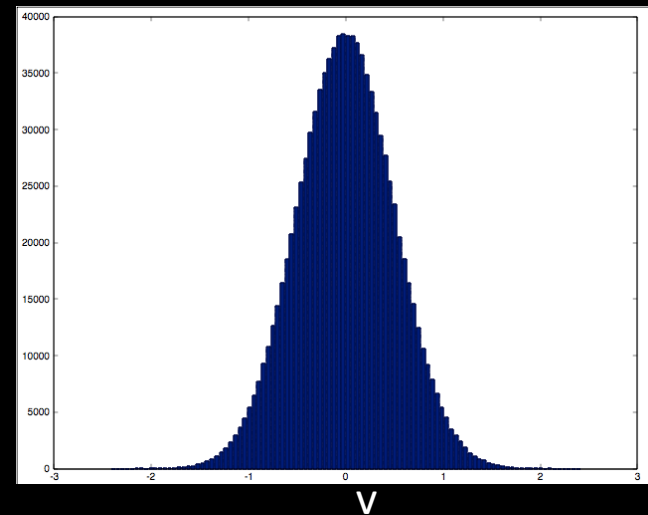
N=1,000



N=10,000



N=1,000,000

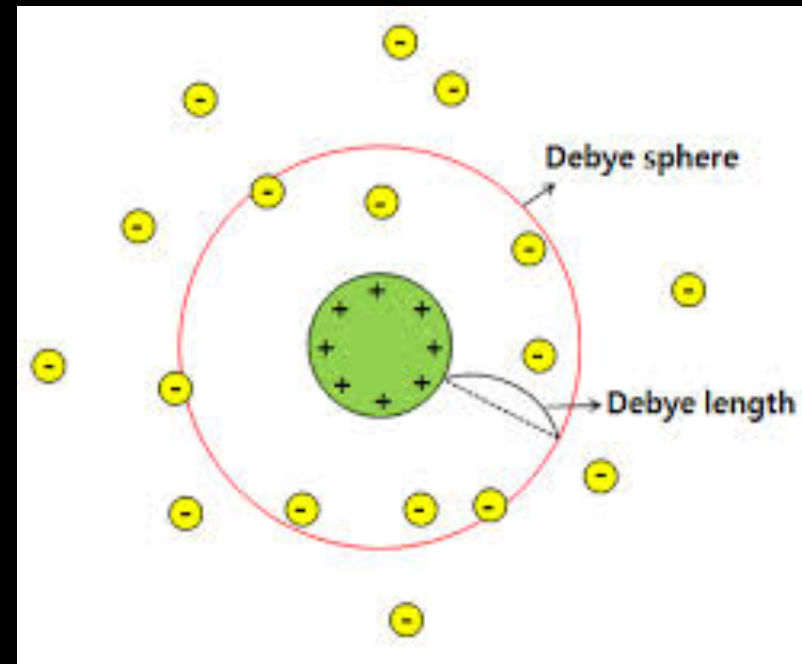


How long DT ?

- For numerical stability reasons, DT must be smaller than $\frac{1}{4}$ of the plasma period.
- Higher density \rightarrow lower plasma period
- Implicit Scheme (like iPIC) allows us to avoid this stability, DT \rightarrow 2-3 plasma periods
- Keep in mind that it is a local quantity!
- Implicit schemes damp in time high frequency component of the field ($<$ plasma frequency)

How Large Dx?

- Dx should be less than the Debye length (characteristic screening distance) = v_{the}/w_p
- Implicit schemes allows us to use approx $Dx = 10$ Debye lengths
- Keep in mind that it is a local quantity!
- Perfect BC on boundaries problematic because we should resolve sheath of typically tens Debye lengths



The Divergence Cleaning in PIC methods

- The field solver in PIC methods solves the Ampere and Faraday laws, **what about the divergence equations ?**
- In PIC methods, $\text{div}(\mathbf{B})=0$ is always conserved but not $\text{div}(\mathbf{E}) = \rho/\epsilon_0$!
- To enforce the Gauss a law an extra equation (divergence cleaning) is solved, the solution of this equation corrects the results obtained by solving Ampere and Faraday law



Variables in iPIC3D

- x, y, z = particle positions
 - u, v, w = particle velocity
 - E = electric field
 - B = magnetic field
 - ρ = charge density
 - J = current density
 - hat variables = intermediate variables used for estimating other variables
- $x \text{ nop} = \text{number of particles}$
- $x \text{ Nxc} \times \text{Nyc} \times \text{Nzc} = \text{number of grid points}$

Main file iPIC3D.cpp

```
EMf->initGEM(vct,grid); // init simulation for reconnection
for (int i = 0; i < ns; i++) // Maxwellian
    part[i].maxwellian(grid, EMf, vct);
...
for (int cycle = first_cycle; cycle < (col->getNcycles() + first_cycle); cycle++) {
    cout << "  cycle = " << cycle + 1 << endl;
    // interpolation
    EMf->setZeroDensities(); // set to zero the densities
    for (int i = 0; i < ns; i++)
        part[i].interpP2G(EMf, grid, vct); // interpolate Particles to Grid(Nodes)
    EMf->sumOverSpecies(vct); // sum all over the species
    EMf->interpDensitiesN2C(vct, grid); // calculate densities on centers from nodes
    EMf->calculateHatFunctions(grid, vct); // calculate the hat quantities for the implicit
    // MAXWELL'S SOLVER
    EMf->calculateE(grid, vct); // calculate the E field
    // PARTICLE MOVER
    for (int i = 0; i < ns; i++) // move each species
        part[i].mover_PC(grid, vct, EMf);
    EMf->calculateB(grid, vct); // calculate the B field
    ...
}
```

Software Requirements

- g++ or any other c++ compiler
- If g++ not available, we provide the results from a previous simulation.
- Autotool make available. Otherwise compilation from command line possible.
- Paraview software for visualization
- (Eventually Visit program can substitute Paraview)

Input File (Sample GEM2D.inp)

SaveDirName = data

B0x = 0.0195

B0y = 0.0

B0z = 0.0

delta = 0.5 #current sheet thickness

dt = 0.3 # dt = time step

ncycles = 2402 # cycles

Lx = 20 # Lx = simulation box length - x direction

Ly = 10 # Ly = simulation box length - y direction

nxc = 128 # nxc = number of cells - x direction

nyc = 64 # nyc = number of cells - y direction

ns = number of species

ns = 4

qom = charge to mass ratio for different species */

qom = -64.0 1.0 -64 1.0

Initial density (make sure that plasma is neutral)

rhoINIT = 1.0 1.0 0.1 0.1

npcelx = number of particles per cell - Direction X

npcelx = 3 3 2 2

npcely = number of particles per cell - Direction Y */

npcely = 3 3 2 2

npcelz = number of particles per cell - Direction Z */

npcelz = 3 3 2 2

4 species:

1. Electron current sheet
2. Ion Current Sheet
3. Electron background
4. Ion Background

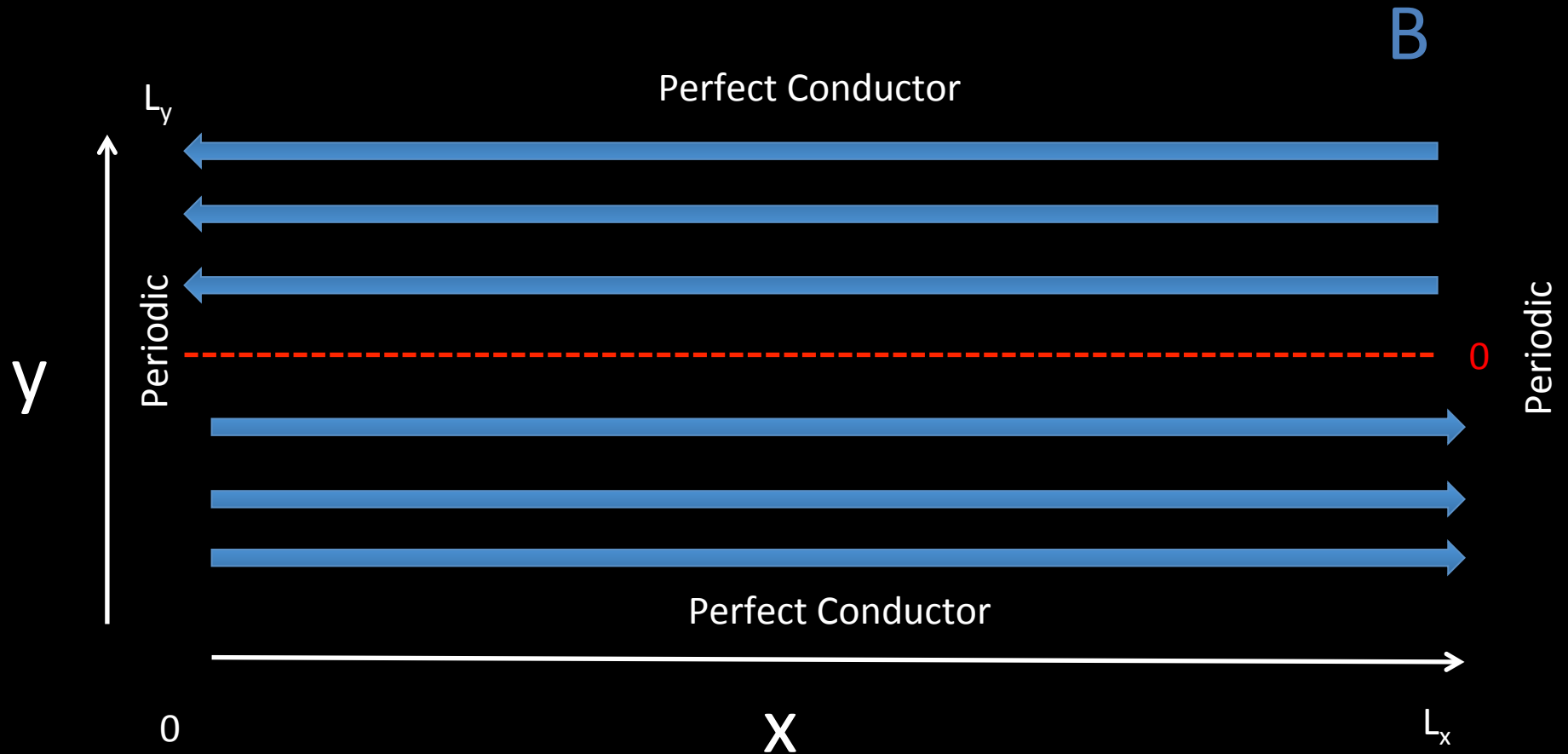
Few Words About iPIC3D Units

All variables are dimensionless. The code allows different normalization, however the most used one is:

- density is normalized by the characteristic current sheet peak density ρ_0
- Time is normalized over w_{pi}
- Velocities are normalized to the speed of light in vacuum c
- Coordinates and lengths are normalized by ion skin depth d_i
- The remaining quantities over a combination of the previous ones.

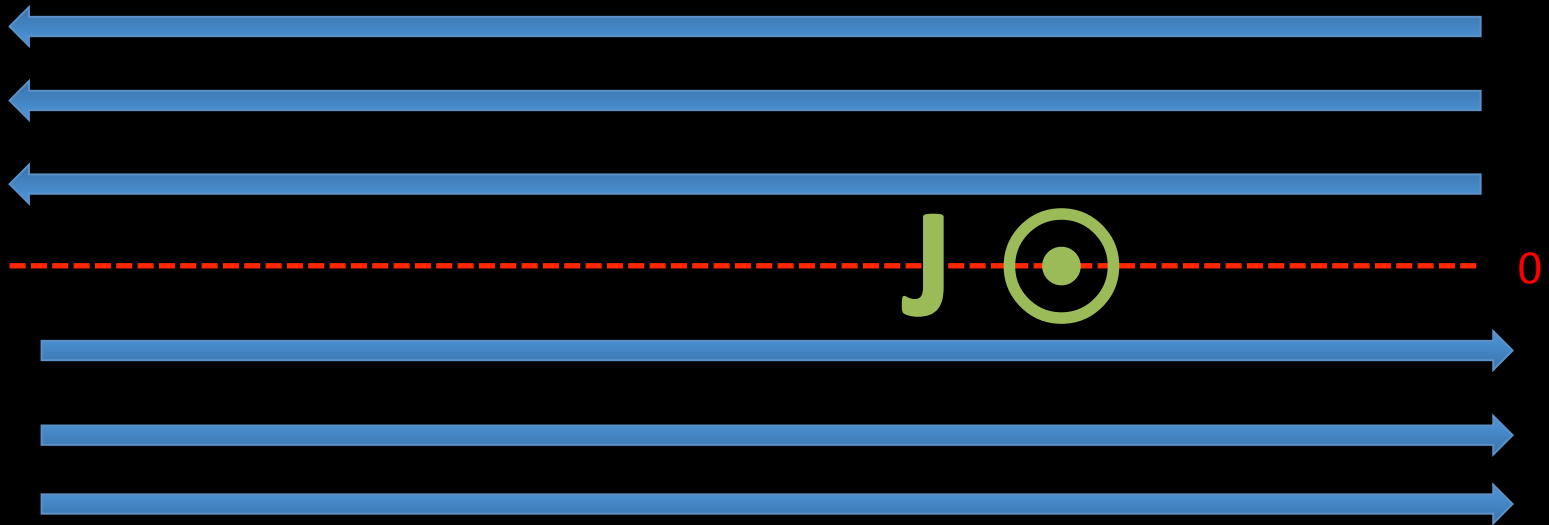
Initial Configuration: 1 current sheet

In initGEM(...) subroutine in fields/EMFields3D.h



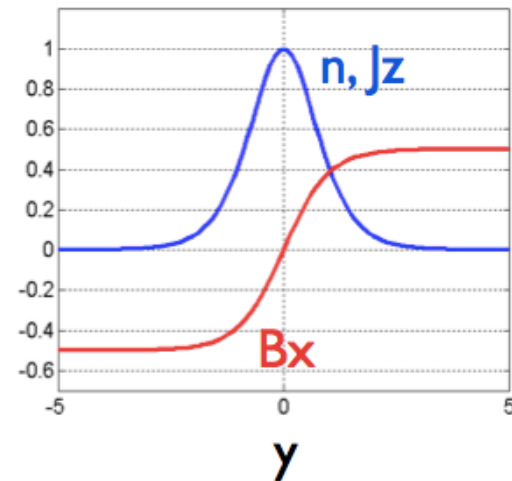
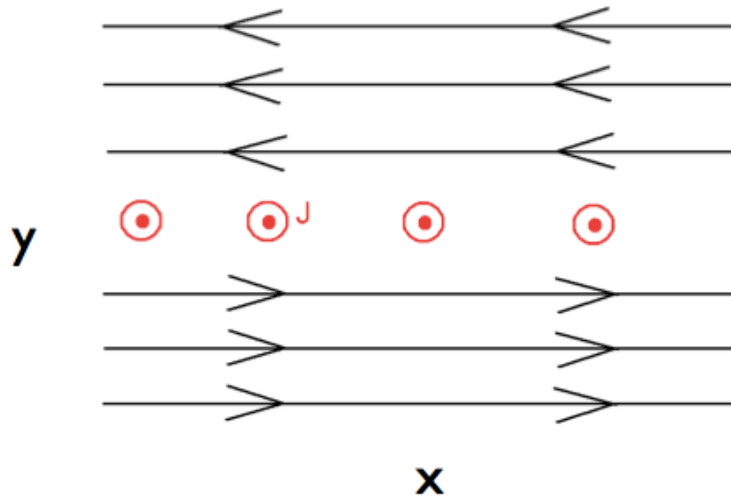
Initial Configuration

One current to satisfy Ampere's law \rightarrow we determine the v drift



+ Force balance \rightarrow initialize $p \rightarrow$ initialize ρ and v_{th}

Initial Configuration



Harris equilibrium:

$$B_x = B_0 \tanh(y/L)$$

$$\rho(y) = \rho_0 \operatorname{sech}(y/L)$$

particle velocities are sampled as a drifted Maxwellian.

A background plasma is added typically 10-20% density of the peak density.

An initial perturbation is added to the magnetic field.

Initial Configuration

We perturb the magnetic field lines only in the center of the current sheet layer.



We want to speed up reconnection introducing already an x-line in the simulation.

Installation of iPIC3D

- Unzip iPIC3D-serial-epigram.zip
- cd iPIC3D-epigram
- Type “make”

```
iPIC3D-epigram markidis$ make
```

```
g++ -O2 -c ./particles/Particles3Dcomm.cpp
```

```
g++ -O2 -c ./particles/Particles3D.cpp
```

```
g++ -O2 -c ./ConfigFile/src/ConfigFile.cpp
```

```
g++ -O2 -o iPIC3D \
```

```
    iPIC3D.cpp Particles3Dcomm.o Particles3D.o ConfigFile.o
```

Running the iPIC3Dcode

- Type “./iPIC3D inputfiles/GEM_2D.inp

```
...
• *****
•   cycle = 47
•   *****
•   *** E CALCULATION ***
•   *** DIVERGENCE CLEANING ***
•   CG Initial error: 0.383322
•   CG converged at iteration # 70 with error 0.000381786
•   *** MAXWELL SOLVER ***
•   Initial residual: 0.70326 norm b vector (source) = 0.446776
•   GMRES converged at restart # 0; iteration #14 with error: 0.000924232
•   *** MOVER with SUBCYCLING 1 - species 0 ***
•   *** MOVER with SUBCYCLING 1 - species 1 ***
•   *** MOVER with SUBCYCLING 1 - species 2 ***
•   *** MOVER with SUBCYCLING 1 - species 3 ***
•   *** B CALCULATION ***
...
```

Output

ConservedQuantities.txt

...

cycle time tot_en magnetic_en kinetic_en ...

...

VTK files in the results directory every 10 cycles
(from inputfile): **B**, **E**, **rho**, **v**.

We visualize the VTK files using Paraview

Exercise – GEM Reconnection

- Install the iPIC3D code
- Run it with inputfile GEM_2D.inp (inputfiles folder) → it saves vtk files in the data folder every 10 cycles → use paraview
- Check total energy (should it change it? Is magnetic field energy decreasing? What about kinetic energy?): open the text file ConservedQuantities (the second column is the total energy)

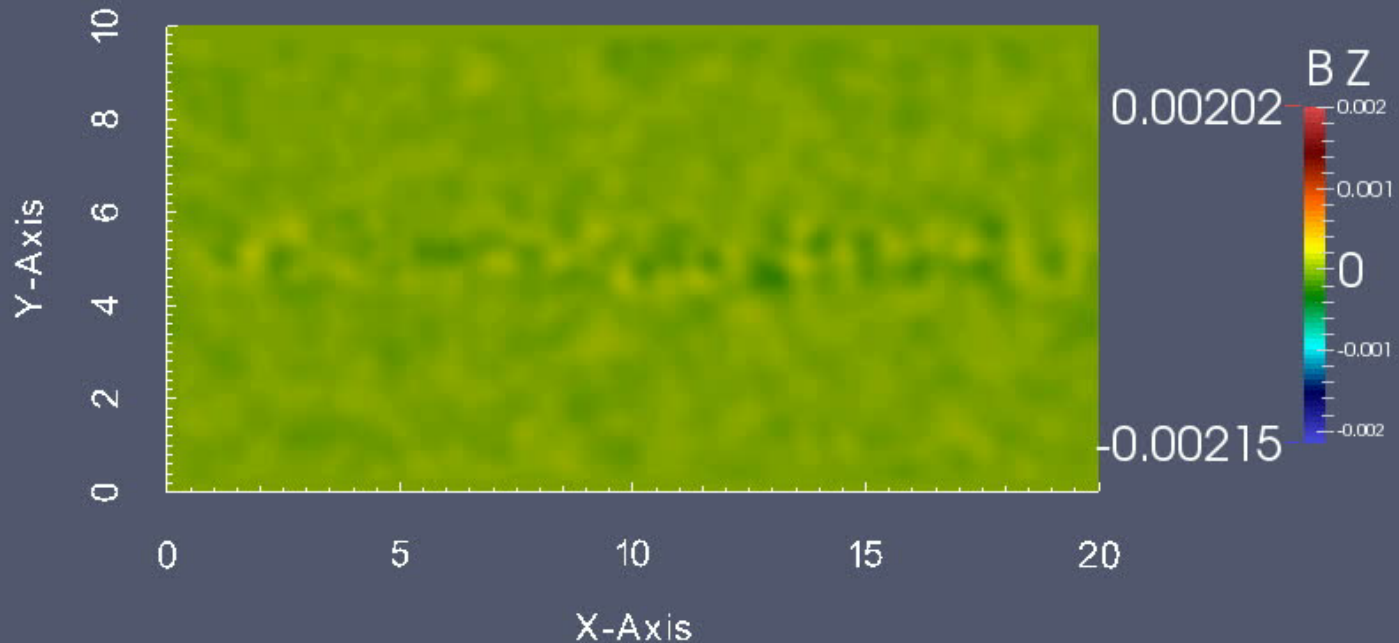
Goal of Exercise

- Looking for signatures of magnetic reconnection:
 - Which are the quantities and pattern of magnetic reconnection? i.e.: Which components of the electric or magnetic field, densities?

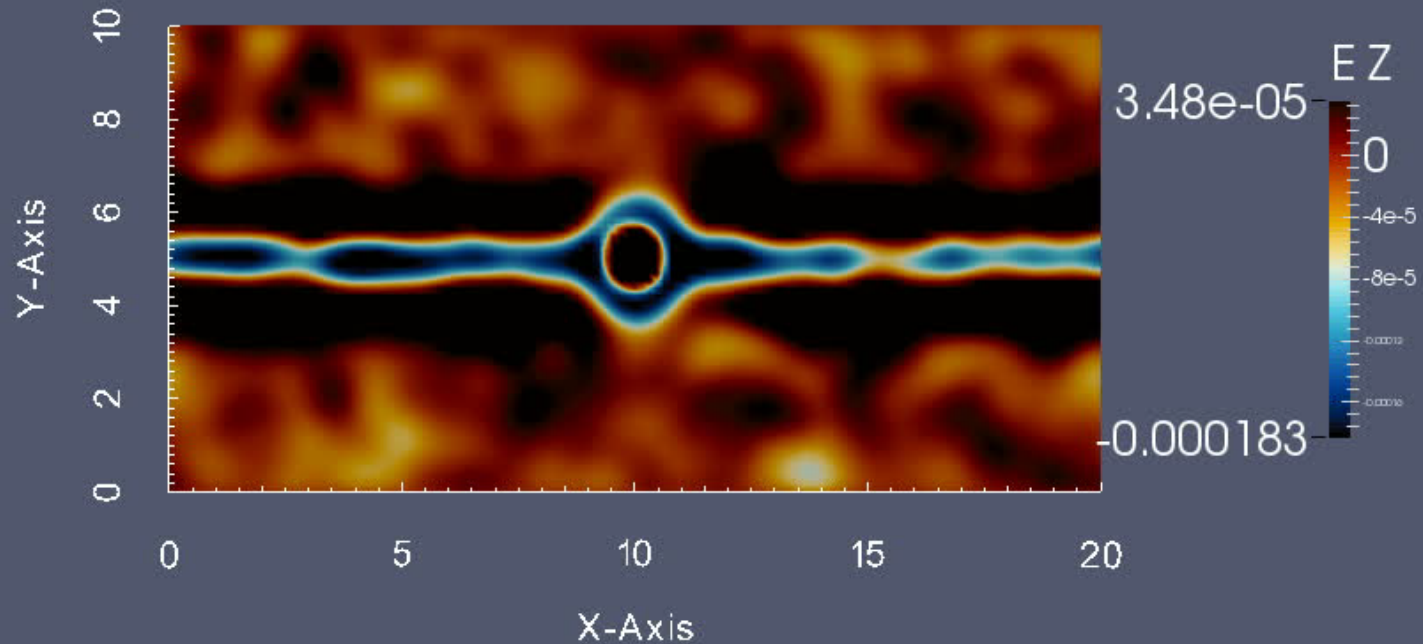
Task 1 – Movie of the Hall Field Intensity

- vtk files are already available in GEM-Results-iPIC3D.zip file. You can run (full simulation approx 30 minutes)
- Open B*vtk files in paraview, select the Bz component.
- Add grid, colorbar, change colormap
- Add Annotate time filter
- Save animation in avi format

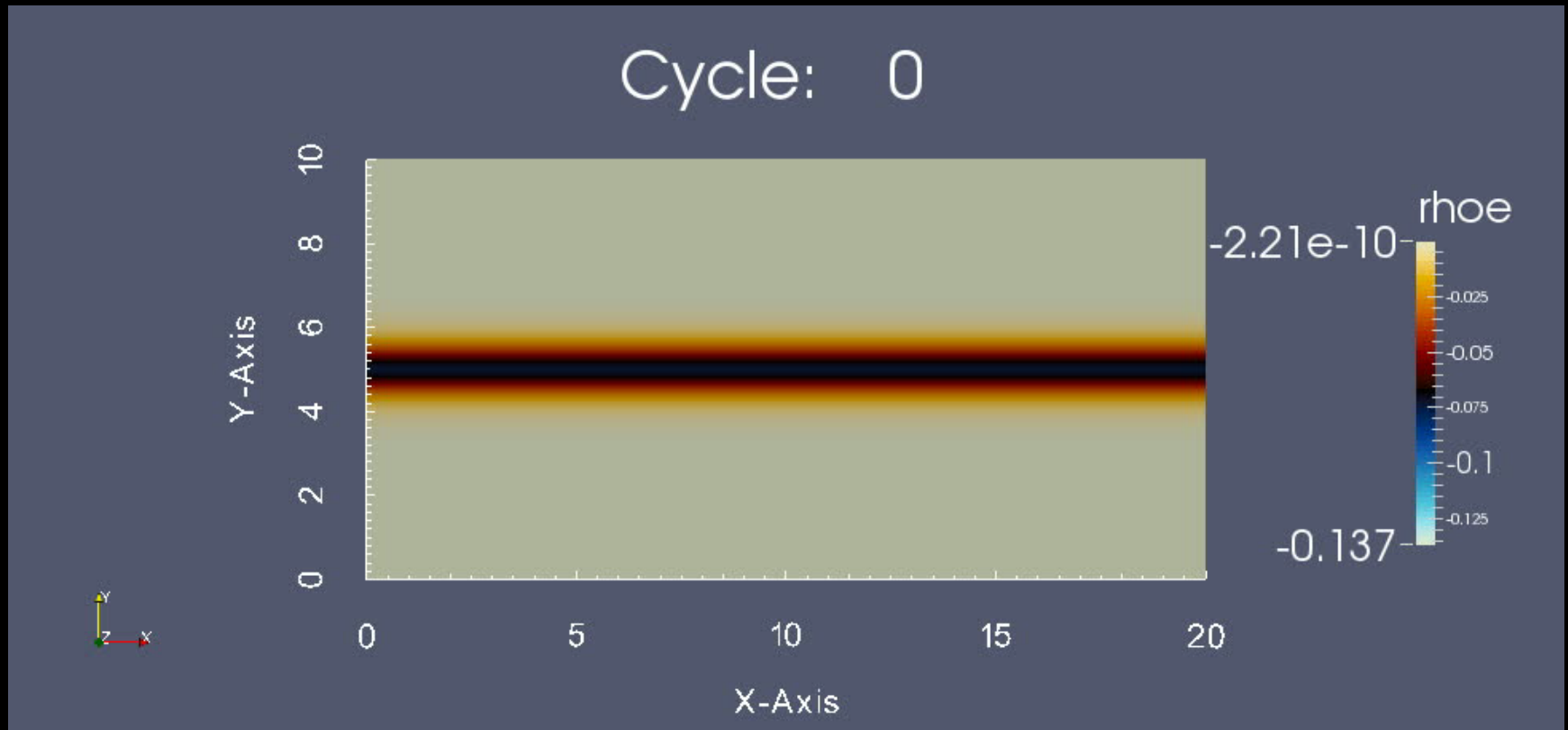
Task 1 – Something like this...



Task 2 – Reconnection Electric Field



Task 3 - Electron Density (ρ_e)



Task 4

- Calculate the parallel (to what?) electric field
- how do you that ?
 - Load E and B files
 - Select them both and use filter Append Attributes
 - Use the Calculator on the append attributes
 - Make an animation of $E_{//}$