Beginner's guide to iPIC3D
2016 PDC Summer School: Introduction to High-Performance Computing

Elin Eriksson
Swedish Institute of Space Physics, Uppsala, Sweden
Uppsala University, Department of Physics and Astronomy, Uppsala, Sweden

November 30, 2016

Contents

When analyzing measurements from space it is important to use kinetic simulations, when kinetic effects cannot be ignored, to try to understand what is happening. iPIC3D is an implicit three-dimensional (3D) parallelized Particle-in-Cell method commonly used to simulate kinetic plasma effects in magnetic reconnection. In this report iPIC3D is explained so that a complete novice to iPIC3D should be able to understand the structure and workflow of the program and be able to run it on a supercomputer or a laptop. Results from a short realistic 3D magnetic reconnection simulation is also presented.

1 Introduction

Simulations are necessary when trying to interpret in-situ measurements from space. It is particularly important when trying to understand three-dimensional (3D) mechanisms. The most common space simulation is the fluid, magnetohydrodynamic (MHD), method. It treats plasma as a fluid and shows the macroscopic evolution of the plasma. However, when studying processes where kinetic effects are important, such as *magnetic reconnection* (MR), MHD simulations are not suitable. Instead for such processes, kinetic simulations should be used. In kinetic simulations, kinetic plasma equations are used to try to understand the behavior of the plasma and is therefore more accurate to explain for example MR. One fully kinetic simulation is iPIC3D.

1.1 iPIC3D

iPIC3D is a fully 3D implicit Particle-in-cell (PIC) parallelized method [1]. In kinetic methods the evolution of the distributions function for a species, f, is solved using the transport equation without the collisional term, the so-called Vlasov equation:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{r}} + \frac{q}{m}(\mathbf{E} + \frac{\mathbf{v} \times \mathbf{B}}{c}) \cdot \frac{\partial f}{\partial \mathbf{v}} = 0, \tag{1.1}$$

where q, m are the charge and mass of the species, respectively. $\mathbf{v}$ is the particle velocity, $\mathbf{r}$ is the particle position, and $\mathbf{B}$, $\mathbf{E}$ are the magnetic and electric fields, respectively. The Vlasov equation is solved in combination with Maxwell's equations. The links between them are the charge density,

$$\rho = \Sigma q \int f d\mathbf{v}, \tag{1.2}$$

and the current density,

$$\mathbf{J} = \Sigma q \int \mathbf{v} f d\mathbf{v}. \tag{1.3}$$

There are various ways to numerically solve the Vlasov-Maxwell system. One of those ways is the PIC method. In the PIC method the Vlasov equation is simplified by describing the distribution by a certain amount of computational particles picked at random from an initial maxwellian distribution. To simulate the evolution of the distribution function the computational particles are moved using Newton's equation of motion:

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}, \tag{1.4}$$

$$\frac{d\mathbf{v}}{dt} = \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}). \tag{1.5}$$

As can be seen in eq. 1.5 the electric and magnetic field are required to solve the acceleration of a computational particle. They are calculated on the grid, that divides the simulation box,

using Maxwell's equations. Every iPIC3D simulation starts at equilibrium. This is for example why so many kinetic simulations of MR use a Harris sheet set-up, it is in equilibrium. If the simulation does not start at equilibrium it is much harder to separate the physical from the non-physical effects.

Commonly, the amount of computational particles is only a small fraction of the actual number of particles in the system. Thus, the reconstruction of the distribution function is statistically noisy (Fig. 1). The mass ratio between the computational electron and ions is usually smaller than in reality to make the simulation less computationally expensive. However, this has been shown to not change the overall behavior of the system [1].

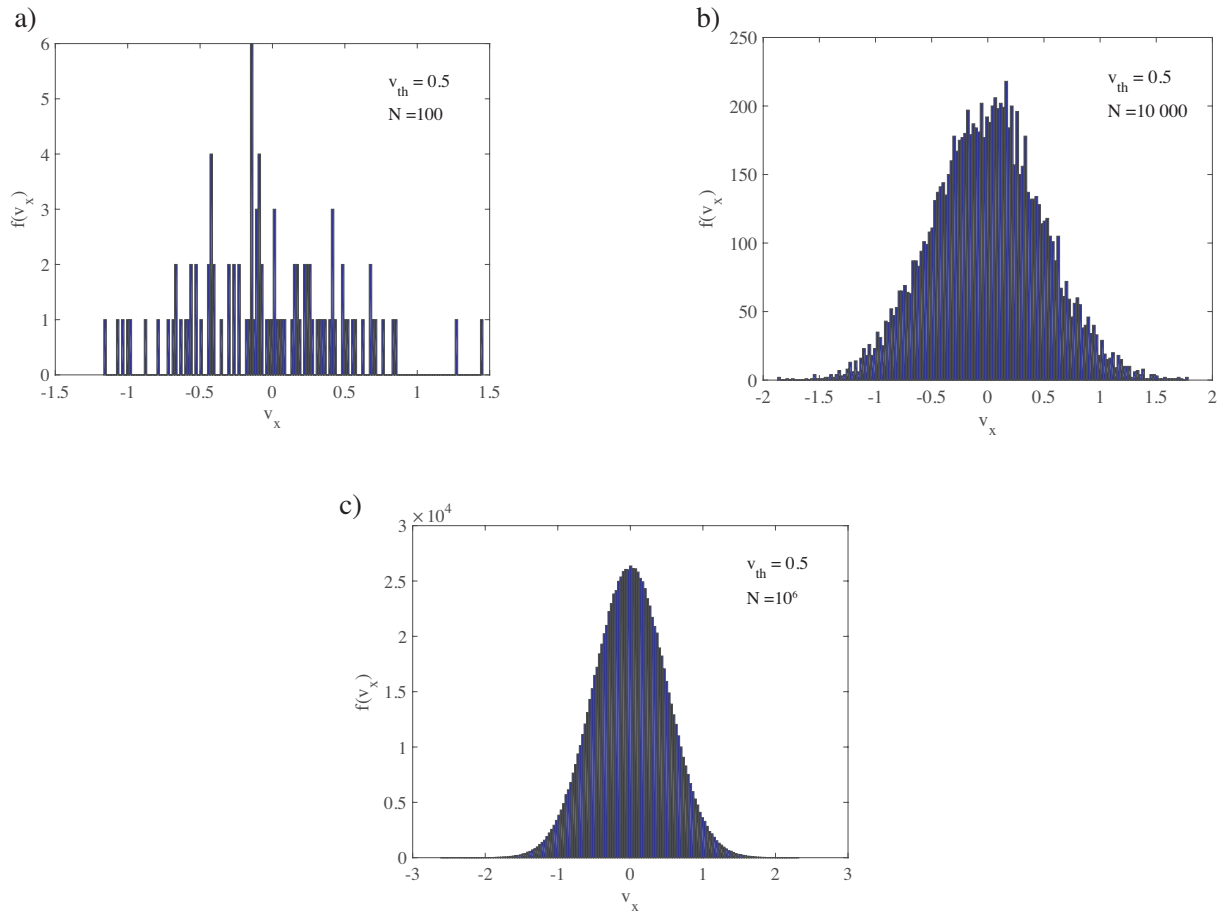

Figure 1: Initializing distribution functions with different amounts of particles. a) N=100, b) N=10 000, c) N=10^6.

The workflow of iPIC3D can be summarized in two steps: 1) $\mathbf{E}$, $\mathbf{B}$, $\mathbf{v}, \mathbf{r}$ are initialized on the grid using the set-up defined in the input file (see section 3). 2) The Maxwell equation and equation of motion are calculated simultaneously on the grid for several cycles. The amount of cycles to run the simulation for is given in the input file (see section 3). After the amount of cycles has run out, or the time given for the simulation is at an end, the simulation ends.

1.2 MPI

The parallelization of the iPIC3D code is based on MPI (Message Passing Interface) libraries. MPI is a message passing library standard that is decided based on group consensus from over 40 participating organization. It is in itself not a library, but a specification on what should be included in a MPI library. As the name suggest MPI is based on parallel message-passing. That is data is moved from one address space related to a specific process to another process

using message passing coding. Because MPI is a standard specification, you should be able to take your program and run it on any high performance computer (HPC) without having to change it. In fact, MPI is popular due to its portability. Applications using MPI do not need to be adapted before moving to a different machine. However, each HPC machine might have a vendor-specific version of MPI implementation. These vendor provided implementations can provide machine-specific optimization. Thus, you might have to adapt the program to increase the performance of your program when moving to a different machine.

This report is a guide directed towards space scientists with little to no previous knowledge of iPIC3D on how to use iPIC3D. It is divided into specific sections dealing with how to install iPIC3D on different computers (section 2), explaining common terms in an input file (section 3), important physical interpretations of the simulation (section 4), step by step guide on how to use the program (section 5), results from a simple realistic 3D reconnection simulation (section 6), and a short discussion of those results (section 6.2).

2 Installation

To run iPIC3D with both particle and field data output the following programs are needed:

1. iPIC3D,
2. MPI,
3. HDF5,
4. cmake.

MPI and cmake libraries are needed to compile iPIC3D, while the HDF5 library is only required during compilation if you want particle data output. These three libraries are usually pre-installed on most HPC platforms. You only need to load their respective module on the platform before compiling iPIC3D.

2.1 Beskow

Beskow, the high performance computer at KTH, is a Cray XC40 system based on Cray Aries interconnect technology and Intel Haswell processors. It has 1676 computing nodes and 26 service nodes, where each node contains 32 cores. Beskow is designed for running large parallel jobs, so when asking for time allocation large running jobs are preferred. The programs MPI, HDF5, and cmake are already installed on Beskow. Thus, to run iPIC3D on Beskow you only need to install iPIC3D on your username and load the cray-hdf5 module before making an executable file (compile the program). During the writing of this report the compiler (MPI) used for Beskow was cray version 8.3.4.

iPIC3D can be found on bitbucket (https://bitbucket.org/bopkth/ipic3d-klm). It can be downloaded using the same syntax as github. That is you can use the terminal command:

```
git clone https://bitbucket.org/bopkth/ipic3d-klm.git
```

If this is not working when you try to install it, the new location of the depository can be found at the clone button on the left-side of the website or you might need to install git (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git). After successfully executing the terminal code a folder called ipic3d-klm will be installed at the location you used the terminal command at. Go into the ipic3d-klm folder and create a folder called build. This is the folder where you will always compile your executable file. For easier use it is recommended that you also keep your input file (see section 3) in the build folder. Go into the build folder and write cmake. This will create all the files needed to make an executable file. Make all the necessary changes to the parameters in the input file and load the cray-hdf5 module using the

terminal code:

```
module load cray-hdf5
```

You can now create an executable iPIC3D file with the simulation set-up given in the input file using the terminal command:

```
make
```

An executable file named iPIC3D has now been created. It is this file that will be sent to Beskow to run the simulation. Note, please remember to create a folder in the build folder called data, otherwise no data will be saved. If you want a different name for your data folder you can change this in the input file (see section 3) and create that folder instead.

2.1.1 Running

There are two ways to run a simulation on Beskow:

1. interactive mode,
2. sbatch.

In interactive mode you directly ask Beskow for a specific amount of nodes (that can run several processes) and time for the simulation. Once it is granted you can run your simulation. However, if you leave Beskow while the simulation is running the simulation will be cancelled. In sbatch you write a shell script, a so-called job script, in sbatch code where you ask for the same things as in the interactive mode. However, in this case Beskow will run the script and you do not have to stay logged in for the simulation to run. An example of a job script can be found in appendix B.

When using iPIC3D on Beskow it is recommended to follow the steps given below:

1) To enter Beskow (or any super computer on KTH) you need to create a ticket using the terminal code:

```
kinit -l 2d -forwardable user@NADA.KTH.SE
```

This ticket makes it easier to log in and out of Beskow without having to always give your password. In the code given above the ticket for the account user will be valid for 2 days. To get the ticket you will need to give the password of your account, which you can apply for at the PDC center (https://www.pdc.kth.se/support/accounts).

2) Log in to Beskow using:

```
ssh -Y user@beskow.pdc.kth.se
```

3) When you are logged in change to the klemmings folder:

```
cd /cfs/klemming/nobackup/u/user
```

The reason why you should change to the klemmings folder is because there your data will not be deleted without notifying you.

4) Install iPIC3D in this folder following the steps in section 2.1

5) Go into the build folder inside the ipic3d-klm folder:

```
cd ipic3d-klm/build
```

6) All the compilation of the files is done in the build folder, while the input file folder has several examples of input files for different physical set-ups. Easiest is to just move the input file you are using to the build folder so that everything is in the same place:

```
cp /cfs/klemming/nobackup/u/user/ipic3d-klm/inputfiles/testGEM3Dsmall.inp
/cfs/klemming/nobackup/u/user/ipic3d-klm/build/
```

7) Edit the input file to suit your needs (see section 3) using any installed text editors such as vim, gedit etc.:

```
gedit testGEM3Dsmall.inp
```

8) Load the HDF5 module (for saving particle data):

```
module load cray-hdf5
```

9) Compile the file so you get the executable file called iPIC3D using:

```
make
```

10) Now you need to ask for allocation of nodes for your run either by submitting a sbatch file (shell script with sbatch code, see example in appendix B) or just use the interactive mode:

```
salloc -A accountname -t h:min:s -N 4
```

-N stands for the number of nodes, -A is the account that will be charged for the time, -t is the time you want to use for the simulation. Each node on Beskow have 32 cores. For best performance, it is recommended to only use one process per core. Thus, for the code given above 128 processes should be used when running the simulation.

11) Run the file using:

```
aprun -n numberofprocesses ./iPIC3D testGEM3Dsmall.inp
```

either in the job script or directly in the terminal. Please remember to add the name of the input file in the end otherwise the simulation will not run. To submit the job script to Beskow write:

```
sbatch ./myjob.sh
```

where myjob is the name of the jobscript.

12) Just because the simulation is running without any errors does not mean that unexpected boundary effects are not occurring. That is why step 12 is always to verify that the interesting parts of the simulation is not being affected by the simulation set-up. This can be done using Paraview, a free simulation program (http://www.paraview.org). Paraview is not installed on Beskow. However, you can copy a specific data set to your own computer and do the Paraview analysis there:

```
scp user@beskow.pdc.kth.se:/cfs/klemming/nobackup/u/user/ipic3d-klm/build/data/*250.vtk
```

.

This example copies cycle 250 of field data (.vtk) from Beskow to the folder you are in when executing the code.

2.2 Laptop

If you want to use iPIC3D on your own laptop you will most likely need to install all of the programs listed in the beginning of section 2. MPI can be installed using homebrew (http://www.howtogeek.com/211541/homebrew-for-os-x-easily-installs-desktop-apps-and-terminal-utilities/) with the terminal code:

```
brew install mpich
```

HDF5 can be downloaded from https://support.hdfgroup.org/HDF5/release/obtain5.html and built using cmake. This is done by first downloading the tar file for Unix (for MAC) under source code. Untar the downloaded file and make a build folder inside it. Then write the terminal code:

```
cmake
```

inside the build folder. This will configure HDF5 so that the program can be installed by writing:

```
make
```

and then:

```
make install
```

If you do not have cmake on your computer you can download it at https://cmake.org/download/. iPIC3D is installed in the same manner as described for Beskow (section 2.1).

To get everything to work you might have to update your bash_profile with the locations of cmake, mpirun, and h5dump using the terminal code:

```
emacs /.bash_profile
alias cmake="/Applications/CMake.app/Contents/bin/cmake"
alias mpirun="/usr/local/Cellar/mpich2/3.1.4/bin/mpirun"
```

```
alias h5dump="/usr/local/HDF_Group/HDF5/1.8.17/bin/h5dump"
```
To make sure that all open terminal windows knows about the new changes use the terminal code:
```
source  /.bash_profile
```
By default iPIC3D should use the static links in your installed MPI library. However, if the compilation does not work due to an error with MPI the problem could be that for some reason iPIC3D is only seeing the dynamic links. There is a work around to handle this. Go into the library folder for the MPICH program (or whichever MPI library you have installed):
```
cd /usr/local/Cellar/mpich2/3.1.4/lib
```
and move all dynamical files (.dylib) into a new folder you create there:
```
mkdir dynamiclibrary
mv *.dylib dynamiclibrary/
```
This way iPIC3D can only "see" the static links and everything will work as it should. To successfully compile iPIC3D you will also need to update the CMakelist.txt inside the ipic3d-klm folder with the locations of the installed programs under "Find third class libraries" (line 48 and 56 in appendix C). You also need to remove -DNO_HDF5 in line 35 in CMakelist.txt (appendix C), if it is there. Otherwise, there is no output of particle data. An example of a CMakelist.txt edited for my laptop (OS 10.9.5 with MPI version 3.1.4) can be found in appendix C. All the codes in section 2.2 will need to be adjusted based on where the programs are installed on your own laptop and which version of MPI you are using. If you have done cmake before doing all these changes you need to re-do the cmake and make before trying to run the executable program.

2.2.1 Running

When running on your own laptop the code is instead:
```
mpirun -n 8 ./iPIC3D testGEM3Dsmall.inp
```
Unless you want to freeze and/or crash your laptop, iPIC3D should only be done for 2D models with ion resolution. You should also not use more than 8 processors. If you are unsure if the simulation is going to freeze your computer you can see how much memory it takes by opening a terminal window when running the simulation and writing:
```
top
```
This will show you the top programs drawing the most memory on your computer. If PhysMem in the top window jumps over your computer's memory limit, your computer will slow down and start to freeze.

3 Input file

An example of an input file can be found in appendix A. The first thing you need to do when creating the set-up for your simulation is to define your simulation box. This is done by splitting the simulation box into different processors (the MPI domain) and in inertial lengths to resolve different particle scales (size of box and number of cells). The size of the box in ion inertial length, if the charge per mass is normalized to ions, is given by L_x, L_y, and L_z (Fig. 2b). The box is then split into a number of cells in each direction (nxc, nyc, nzc), Fig. 2b. The size of a cell in any direction is given by:

$$\Delta\alpha = \frac{L_\alpha}{n_\alpha c} \ , \quad \alpha = x, y, z.$$

The $X/Y/Z_{LEN}$ parameters gives the number of MPI processes handling all the cells in each direction (Fig. 2a). Thus, the number of cells of the simulation box in each direction needs to be

dividable by the length of the box in MPI processes ($X/Y/Z_{LEN}$). This is to make the domains even so that the cells are evenly distributed and the "boundaries" between the MPI domains are fully defined. Thus, the cells can be split and calculated separately on different processors. At each MPI boundary there will be an MPI_send (data passing) and MPI_wait (synchronization so the domain does not work with "old" data), so each MPI domain knows how many particles they have and where they are. Note, if you use less processors when running a simulation you either need to lower the amount of cells in the simulation box (resolution) or lower the number of particles in each cell (less accurate distribution and interaction). Otherwise, the simulation will be extremely long (on super computer) or freeze your laptop. Please remember that to successfully compile iPIC3D $X_{LEN}*Y_{LEN}*Z_{LEN}$ = number of processors.

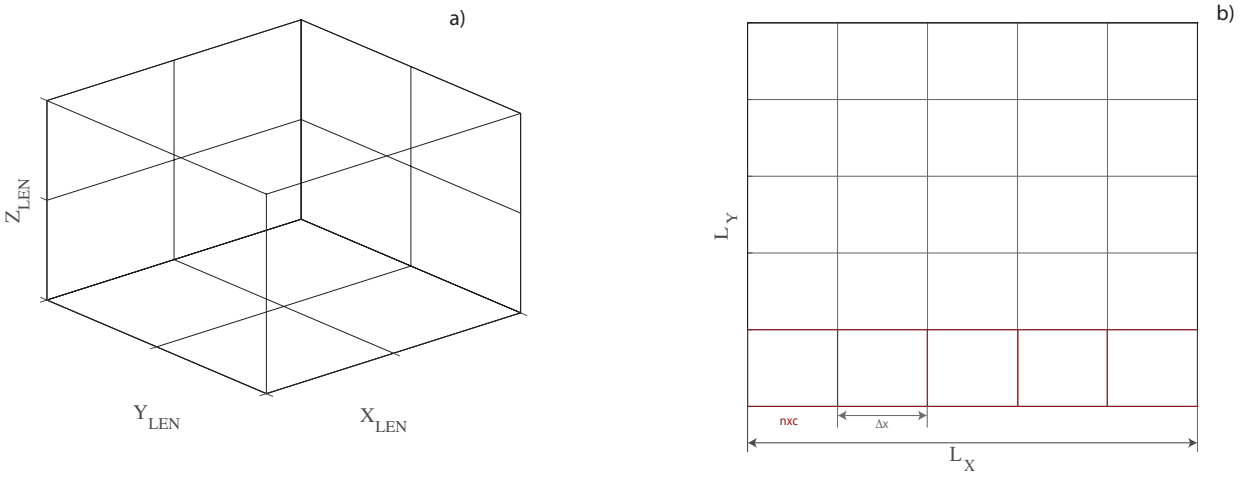

Figure 2: Illustration of the simulation box split into MPI domains and inertial length and cells. a) Box in MPI domains, b) 2D cut of simulation box showing the definition of cells, size of cell, and length of box in inertial lengths.

In the absolute beginning of the input file two different folders is defined where data is to be saved. The reason why two data sets is saved is because long simulations are not allowed to run indefinitely on Beskow because of fair use. Therefore, data needs to be saved at certain checkpoints so when the simulation can be re-started it can begin from the cycle where the last data was saved. Flags tell you which environment file should be used. You can create your own case if necessary. Here you can also set the method of writing data to disk. This should only be changed if you know the computer has a better system available than the one defined. The magnetic and electric field are also initialized here. The delta parameters gives the thickness of the current sheet in the beginning. The time section is where you decide how big each time step should be and the number of cycles the simulation should run. The smaller time step you choose, the better you can see the electrons. However, the simulation will then need to run for more cycles to observe the evolution of different physical interactions.

In iPIC3D simulations, all parameters are normalized to be equal to 1 (integers). This is because less floating points are better for the precision of the computation. All computers have rounding errors when saving floating point data. If the value of the data point is too small, the rounding errors could affect the simulation result. All velocity outputs are normalized to the speed of light, c, charge is normalized to e, density is normalized to 4π, and time is normalized over the ion plasma frequency, w_{pi}. Coordinates and lengths are usually normalized to the ion skin depth. This can be changed to electron skin depth by changing the charge per mass ratio to electrons having 1 instead of the ions. All remaining parameters are normalized to a combination of the ones just mentioned. The charge per mass given in an input file is just a simulation value to make the simulation run faster. Generally, it is assumed that all physical

effects are scalable to the plasma frequency so changing this ratio (64, 256 etc.) should not affect the result. The real physics in the charge to mass ratio has to do with the resolution of the separation of the electrons and ions ($\Delta\alpha$ =L$_\alpha$/nαc $< d_{i,e}$, $\alpha = x, y, z$). If you do not have enough separation between the electrons and ions it is hard to determine if what you are observing are due to the electron or ion physics. If you are for example interested in the electron diffusion region in MR you need to have enough grid points ($\Delta x/y/z < d_e < d_i$). If normalized to ions, L$_{x/y/z}$ is in ion inertial length. Thus, $\Delta x/y/z < 1$ tells you that the simulation box is divided enough that you can resolve ion scale effects. This can be re-calculated in electron skin depth for each cell size from $d_e = d_i/\sqrt{q/me}$ as $\Delta\alpha = (\mathrm{L}_\alpha/\mathrm{n}\alpha\mathrm{c}) * \sqrt{q/me}, \alpha = x, y, z$. If $\Delta\alpha$ is smaller than one, then the grid size is also smaller than the electron inertial length. Thus, the simulation can resolve electron scale features. Note, if you change the input file to normalize to the electron inertial length you have to change the thermal velocity in the input file as well.

The smoothing parameter determined how much data is from each specific cell and not averaged ("smoothed") with its neighbors. To see microphysics you need as high a smooth parameter as possible (smooth=1 means no smoothing at all. Everything is from its own cell). PERIODIC defines the boundary conditions. If PERIODIC is on, this means you have a continuous boundary (the last cell just continues into the first and vice versa). NiterMover is used to slow down the electrons. This is done since the electrons are usually much faster than the chosen time step, thus you would not see any detailed movements. In the particle section you define the number of species, ns, of protons and electrons. For example: ns=4 means that you have: 1. electrons in the current sheet, 2. ions in the current sheet, 3. background electrons, 4. background ions. Each of the other particle parameters such as charge per mass, qom, are defined for each particle species. Total charge is split between all particles in the box, so the particles themselves are not "real". Instead a blob of them represent a "real" particle.

Rhoinject gives the charge of particles injected from the boundary. If the boundary is chosen as periodic then injection is ignored. TrackID is used to give an unique id to each particle so if you want to trace a specific particle you can. npcelx/y/z gives the number of particles in each cell direction. The number of particles in each cell can be calculated from npcelx∗npcely∗npcelz. Total number of particles in the simulation box can be calculated from (npcelx∗npcely∗npcelz∗nxc∗nyc∗nzc)∗ns. For accuracy it is better to have larger amounts of particles. However, this results in a larger (=longer) simulation run and is therefore more expensive. Each plasma population is initialized with a maxwellian distribution by giving a thermal velocity. The drift velocity gives the velocity of the main population in the distribution function, while the thermal velocity gives the width of the distribution function (the temperature). There are different boundary conditions for fields (bcEM) and particles (bcP). If PERIODIC is chosen then the boundary conditions (bcEM and bcP) are ignored. If riemission for particles is chosen then this means that particles will be created and injected from the boundary. Exit means that particles leaving the boundaries are gone. The tolerance parameters gives the percentage of solutions to stop solving the Vlasov-Maxwell system.

In the output section you define how often you save data and which parameters should be saved. This is where you can define that no particles are saved when starting to test new simulations (see section 5). CallFinalize=1 tells the program to save the last cycle, if the program manages to reach it in the time given for the simulation. Note, make sure when saving particle data that the frequency match the field data so they can be visualized together.

For users interested in experimenting on running iPIC3D on different HPC's there is a specific file named Parameter.h (appendix D where different parallization parameters can be set to optimize the parallization. However, this report is focused on the basic set-up of iPIC3D and the physical understanding of iPIC3D so I will not go into any details about this file. Instead a short summary of its use can be found in appendix D.

4 Physical Interpretations

In iPIC3D you go from a continuous function to a discrete one (number of grid points, cells). That is you take a distribution function and give a statistical expression for it. Thus, to go between them you just have to choose a region of particles and make a histogram. However, there is not enough particles in each cell that you will have a smooth reversal (Fig. 1). The major assumption in iPIC3D is that you take a few discrete particles at random from a maxwellian distribution when initializing the simulation. Thus, the randomly picked points has a higher probability to be around the thermal population since more particles are there. Therefore, iPIC3D (and iPIC overall) has a problem when it comes to simulating wave particle interaction. Wave particle interaction occurs when particle velocity is close to the wave phase velocity. With iPIC you have a statistical sample of the distribution function and therefore there can be a lack of particles at this velocity where the acceleration due to waves would occur. This means that while in space you would see wave particle acceleration, in the iPIC simulation you would not. The trick to avoid this is to make sure that enough particles are chosen when initializing the distribution so that you have enough particles in the tails where the interactions with waves occurs. In addition, when looking at other iPIC3D simulations make sure that the cell size, $\Delta x/y/z$, is small enough that the particle physics analyzed is well resolved.

5 Simulation steps

When using iPIC3D it is advisable to follow these steps:
1) Change the input file to create the simulation environment for your study case. Most important is that: X_{LEN}*Y_{LEN}*Z_{LEN} = number of processors. This is what typically will slow down your own computer, since it does not have access to as many processors.
2) Only save field data in the beginning to speed up the run.
3) Compile using make. This can give several errors due to your parameter choices. Debug until it works.
4) Run the program for 1 hour. This will help you determine how many cycles it takes to run the whole simulation and thus help lessen the time you ask Beskow for. The less time you ask for the less time you spend in the queue before your simulation is run. If you have access to a supercomputer, use a job script, so you do not have to stay logged in during the simulation run.
5) Look at the results from the field data in Paraview to make sure that the simulation is working as it should. If you see boundary effects near the interesting regions increase the box size to avoid non-physical effects muddling the results and re-run. If the box size is increased make sure that the number of cells still give you the resolution you are after.
6) When the simulation is done, visualize the field data again and determine from the box size which processors are handling interesting physics. There is not enough disk space to save particle data for every cycle, so we need to pick a certain amount of cycles to save particle data for. Re-run the simulation, this time saving particle data.
7) Use the id's for the processors containing the interesting physical regions to post-process particle data.

5.1 Post-Processing

Post-processing particle data takes time and usually there is just too much data to post-process it all. Therefore, the id of a processor and specific cycles are commonly used to pick which data to post-process. The HDF5 files in the postprocessing_tools/hdf5 folder do different post-processing. Here you can for example change the particle data from .hdf5 to .vtk allowing you

to visualize it in Paraview. You can also use Matlab to import both sets of data. The fields data (.vtk) can be read using common text readers such as fscanf. The particle data (.hd5f) data can be read using Matlab's high level HDF5 functions: h5disp, h5read etc. or you can just use the post-processing tool to change the particle data into .vtk files and use the fscanf. The easiest way to illustrate your simulation results in physical units is to change the parameters in Matlab by removing the normalization (see section 3).

6 3D Magnetic reconnection simulation

To illustrate the capabilities of iPIC3D a 3D magnetotail reconnection simulation was performed according to the GEM challenge. In the simulation a typical Harris current sheet configuration is initially imposed with no guide field. The z-direction is along the Harris current sheet and the x-y plane is the reconnection plane. The simulation box size is L_x x L_y x L_z=20 d_i x 10 d_i x 10 d_i. The thickness of the current sheet in the beginning is 0.5 d_i. Furthermore, to reduce the computational time of the simulation, an electron mass of $m_e = m_i/64$ is used. The drift velocities of the ion and electron current sheet is -0.0325c and 0.0065c in the z-direction, respectively. The electron temperature is $T_e = T_i/5$. The computational particles are initialized with a maxwellian velocity distribution. The thermal velocity for the electrons and ions are 0.045c and 0.0126c, respectively. The grid of the simulation box is composed of 160 x 80 x 80 cells. In total, 5.1 10^8 computational particles are used. The simulation was run with 128 processors on Beskow supercomputer at KTH Royal Institute of Technology. The simulation time step was chosen as $\Delta t = 0.2 w_{pi}^{-1}$. Both particle and field boundaries are periodic in the x- and z-direction. In the y-direction, the field boundary acts as a perfect conductor and the boundary for particles is perfectly reflecting.

6.1 Results

The simulation show all the characteristic signatures of reconnection: the quadrupole structure of the Hall magnetic field (B_z), Fig 3, and a reconnection electric field near the x-line, Fig. 4. Figure 3 shows the quadrupole Hall magnetic field structure (B_z) around z=$L_z/2$ at three different times. It shows how the quadrupole structure expands from cycle 1300 to 3000 as is expected from previous simulations [1, 2]. Figure 4 shows the z-component of the electric field about y = $L_y/2$ for the same times as in Fig 3. E_z shows signs of lower hybrid waves (blue and red stripes) initially propagating in z-direction at cycle 1300. When reconnection occurs a reconnection electric field is created due to the temporal change in the magnetic field along the x-line in the z-direction at the center of the simulation box. This reconnection electric field can be seen in cycle 3000 as the more focused blue color in the middle of Fig. 4b.

6.2 Discussion

The results from the simulation is in agreement with other simulation results [1, 2]. The simulation showed the quadrupole hall field and reconnection electric field. This 3D reconnection simulation had to be re-done several times. Each time the simulation box had to be increased, because non-physical results were occurring due to boundary effects. The simulation box in the final simulation results have a cell size of 0.125 d_i = 1 d_e. Thus, the simulation has enough resolution that ion scale effects can be resolved. Furthermore, in the diffusion region the density is actually smaller (~ 0.1) than the reference density (= 1). Thus, the electron inertial length is much larger than the one used when calculating the cell size with the reference density. Therefore, electron scale effects can still be resolved with this cell size, since the cell size is actually smaller than 1 d_e.

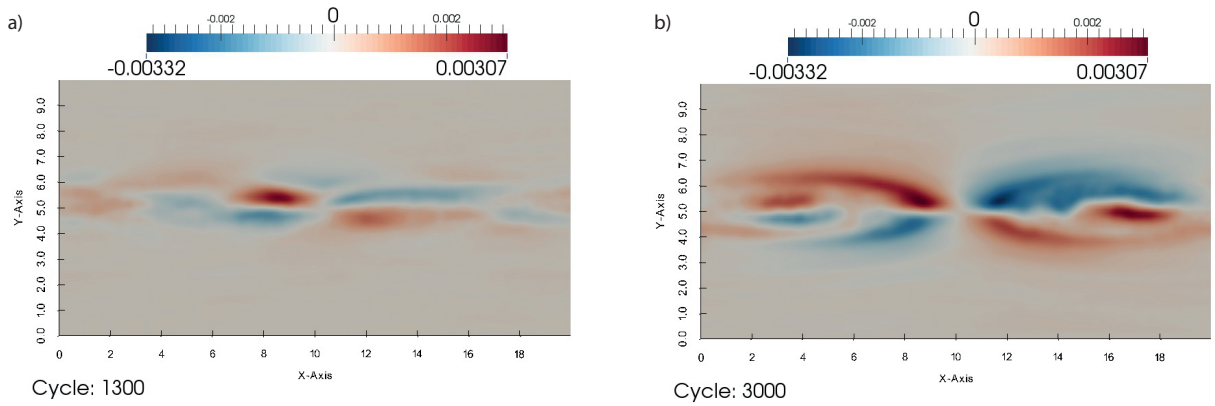

Figure 3: Contourplot of the Hall magnetic field at three different cycle stages of the simulation. a) cycle 1300 b) cycle 3000.

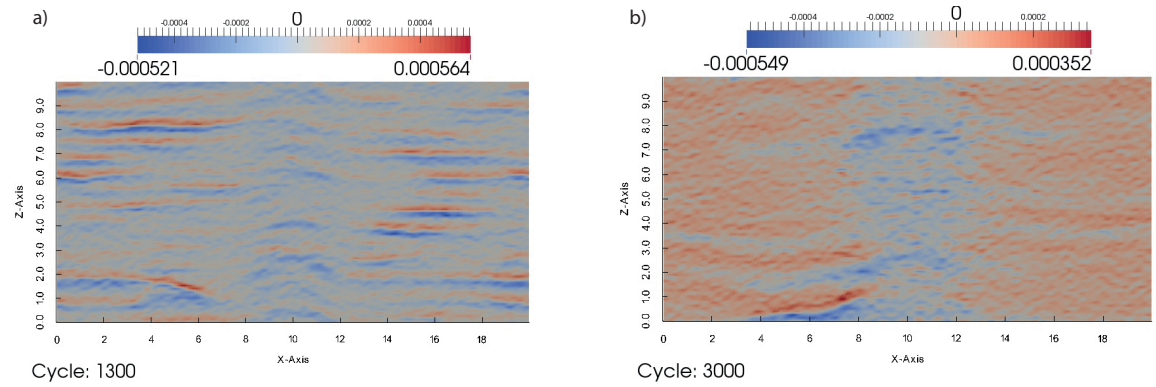

Figure 4: Contourplot of the the electric field in z-direction at three different cycle stages of the simulation. a) cycle 1300 b) cycle 3000.

The pictures given in the results section all show the full simulation box. If the full simulation box is not shown in at least one figure in a simulation paper there is no way to determine if the results from the simulations are from potential boundary effects or real. Thus, if you read a simulation paper where the figures only shows cut-outs of interesting results ask the authors of the paper for more figures showing the full simulation box before using their result in your own paper.

Appendix A Input file

Listing 1: Input File for Simulation Parameters

```
SaveDirName = data
RestartDirName = data

# New flags:
Case            = GEM      # Select the case
PoissonCorrection = no    # Poisson correction
WriteMethod     = pvtk    #parallel VTK
SimulationName  = testGEM3D # Simulation name for the output

#  %%%%%%%%%%%%%%%%%%% Magnetic Reconnection %%%%%%%%%%%%%%%%%%
B0x = 0.0195
B0y = 0.0
B0z = 0.0
delta = 0.5


#  %%%%%%%%%%%%%%%%%%% TIME %%%%%%%%%%%%%%%%%%
dt = 0.15        # dt = time step
ncycles = 5001  # cycles
th = 1.0        # th =  decentering parameter
c = 1.0

#  %%%%%%%%%%%%%%%%%%% SMOOTH %%%%%%%%%%%%%%%%%%
Smooth = 0.2    # Smoothing value (5-points stencil)

Lx = 10  # Lx = simulation box length - x direction
Ly = 10  # Ly = simulation box length - y direction
Lz = 10   # Lz = simulation box length - z direction
nxc = 20 # nxc = number of cells - x direction
nyc = 20 # nyc = number of cells - y direction
nzc = 20 # nzc = number of cells - z direction

# %%%%%%%%%%%%%%% MPI TOPOLOGY %%%%%%%%%%%%%%%
# number of MPI subdomains in each direction
XLEN = 2
YLEN = 2
ZLEN = 2
# topology of subdomains in each dimension (1=true, 0=false)
PERIODICX = 1
PERIODICY = 0
PERIODICZ = 1

# mover
NiterMover = 3 # number of predictor-corrector iterations for electrons

# %%%%%%%%%%%%%%% PARTICLES %%%%%%%%%%%%%%%%%%
#    ns = number of species
#    0 = electrons
#    1 = protons
#    ...
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
rhoINJECT = 0.0 0.0 0.0 0.0

# TrackParticleID[species] = 1=true, 0=false --> Assign ID to particles
TrackParticleID= 1 1 1 1
# npcelx = number of particles per cell - Direction X
npcelx = 2 2 2 2
# npcely = number of particles per cell - Direction Y */
npcely = 2 2 2 2
# npcelz = number of particles per cell - Direction Z */
npcelz = 2 2 2 2


# uth = thermal velocity for different species - Direction X */
uth  = 0.045 0.0126 0.045 0.0126
# vth = thermal velocity for different species - Direction Y */
vth  = 0.045 0.0126 0.045 0.0126
# wth = thermal velocity for different species - Direction Z */
wth  = 0.045 0.0126 0.045 0.0126
# u0 = drift velocity - Direction X  */
u0 = 0.0 0.0 0.0 0.0
# v0 = drift velocity - Direction Y  */
v0 = 0.0 0.0 0.0 0.0
# w0 = drift velocity - Direction Z  */
w0 = 0.0065 -0.0325 0.0 0.0

# &&&&&&&&&&&& boundary conditions &&&&&&&&&&&&&&&
# PHI Electrostatic Potential
# 0,1 = Dirichilet boundary condition
# 2  = Neumann boundary condition
bcPHIfaceXright = 1
bcPHIfaceXleft = 1
bcPHIfaceYright = 1
bcPHIfaceYleft = 1
bcPHIfaceZright = 1
bcPHIfaceZleft = 1

#    EM field boundary condition
#    0 = perfect conductor
#    1 = magnetic mirror

```

```
bcEMfaceXright = 0
bcEMfaceXleft = 0
bcEMfaceYright = 0
bcEMfaceYleft = 0
bcEMfaceZright = 0
bcEMfaceZleft = 0
#    Particles Boundary condition
#    0 = exit
#    1 = perfect mirror
#    2 = riemission
#    Caveat: if your processor topology is set to be periodic in a direction automatically the boundary condition in that direction will be periodic*/
bcPfaceXright = 1
bcPfaceXleft = 1
bcPfaceYright = 1
bcPfaceYleft = 1
bcPfaceZright = 1
bcPfaceZleft = 1

#  print to video results */
verbose = 1

# velocity of the injection from the wall
Vinj= 0.0

# CG solver stopping criterium tolerance
CGtol = 1E-3
# GMRES solver stopping criterium tolerance
GMREStol = 1E-3
# mover predictor corrector iteration
NiterMover = 3
# Output for field
FieldOutputCycle = 25
FieldOutputTag  = B+E+Je+Ji
MomentsOutputTag = rho+PXX+PXY+PXZ+PYY+PYZ+PZZ
# Output for particles if 0 it doesnt save particles data
ParticlesOutputCycle = 45
ParticlesOutputTag = position+velocity+q
# restart cycle
RestartOutputCycle = 30
CallFinalize=1
```

Appendix B Sbatch Jobscript

Listing 2: Job Script on Beskow Supercomputer

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# Set the allocation to be charged for this job. For my summer course we have an account named summer-2016 not required if you have set a default allocation
#SBATCH -A your-project-account

# The name of the script is myjob. You can pick this at random. This is the name that will show up if you look in Beskows queue on job being performed or waiting to run.
#SBATCH -J myjob

# Only 3 hours will be given for this simulation, so if the simulation is not done by this time it will be cancelled.
#SBATCH -t 3:00:00

# Number of nodes. Each node has 32 cores and.
#SBATCH --nodes=4
# Number of MPI processes per node (the following is actually the default). It is best to keep to one process per core otherwise the performance due to several processes trying to access the same memory space will limit the performace.
#SBATCH --ntasks-per-node=32

# If there is any errors put them in error_file
#SBATCH -e error_file.e
# Save all output such as warnings, errors etc. in the output_file
#SBATCH -o output_file.o

# Run the executable named iPIC3D with input file testGEM3Dsmall.inp with 128 processes and write the output into my_output_file

aprun -n 128 ./iPIC3D testGEM3Dsmall.inp > my_output_file 2>\&1\\
```

Appendix C CMakeList.txt

Listing 3: Cmake file for Compilation

```
cmake_minimum_required(VERSION 2.8.8)
# compiler set in ../cmake/cmake_template.cmake.XeonPhi
#message ("for Xeon Phi:")
#message ("cmake .. -DCMAKE_TOOLCHAIN_FILE=../cmake/cmake_template.cmake.XeonPhi")
#message ("for Xeon:")
#message ("cmake .. -DCMAKE_TOOLCHAIN_FILE=../cmake/cmake_template.cmake.Xeon")
#
# Project declaration
#

project(iPic3D)

#
# Set compiler flags per system
#
if (${CMAKE_SYSTEM_PROCESSOR} STREQUAL "k1om") ## Xeon Phi
option(IPIC_XEONPHI "ipic xeon phi standard compile flags" on)
if(IPIC_XEONPHI)
set(CMAKE_CXX_FLAGS "-O3 -openmp -fno-exceptions -fp-model fast=2 -vec-report -mmic")
else()
set(CMAKE_CXX_FLAGS "-mmic")
endif()
#set(CMAKE_CXX_FLAGS "$(CMAKE_CXX_FLAGS) -mmic")
#set(CMAKE_CXX_FLAGS "-O3 -openmp -fno-exceptions -vec-report -mmic")
#set(CMAKE_CXX_FLAGS "-openmp -g -mmic") # set flags for Xeon Phi, totalview
elseif (${CMAKE_SYSTEM_PROCESSOR} STREQUAL "x86_64") ## Xeon
option(IPIC_XEON "optimisation with icpc" OFF)
if(IPIC_XEON)
#reporting: -g -vec-report
#optimization: -O3 -xHost -fno-exceptions
set(CMAKE_CXX_FLAGS "-openmp -fno-exceptions -O3 -xHost -vec-report")
set(CMAKE_CXX_COMPILER "icpc")
endif()
#set(CMAKE_CXX_FLAGS "-O3 -openmp -g -xHost -fno-exceptions -vec-report")
set(CMAKE_CXX_FLAGS "-O3 -DPRINTPCL") # -DPRINTPCL for printing particle subcycles -DNO_HDF5 to disable output
else()
set(CMAKE_CXX_FLAGS "-O3")
endif()

#
# Find third class libraries
#
if (${CMAKE_SYSTEM_PROCESSOR} STREQUAL "k1om") ## Xeon Phi
set(CMAKE_PREFIX_PATH /opt/hdf5/1.8.10-patch1-mic)
set(VARIOUS_LIB /opt/intel/lib/mic)
set (EXTRA_LIBS ${VARIOUS_LIB}/libimf.so ${VARIOUS_LIB}/libsvml.so ${VARIOUS_LIB}/libirng.so ${VARIOUS_LIB}/libintlc.so ${MPELIB})
else()
set(CMAKE_CXX_COMPILER /usr/local/Cellar/mpich2/3.1.4/bin/mpicxx)
set(EXTRA_LIBS "")
endif()

SITE_NAME(myhostname)
if (${myhostname} MATCHES "(.*)pdc(.*)")
set(HDF5_INCLUDE_DIRS /opt/cray/hdf5/1.8.7/cray/73/include)
else()
set(CMAKE_PREFIX_PATH /usr/local/HDF_Group/HDF5/1.8.17)
find_package(HDF5 COMPONENTS HL C )
endif()


#
# include and lib directories
#

# include_directories: files there are accessible to the project
include_directories(
include
${HDF5_INCLUDE_DIRS}
${HDF5_HL_INCLUDE_DIRS}
${MPI_INCLUDE_PATH}
)

link_directories(
${HDF5_LIBRARY_DIRS}
${HDF5_hdf5_hl_LIBRARY_DIRS}
${MPI_LIBRARY_DIRS}
${EXTRA_LIBS}
)

#
# Header file list
#

file(
GLOB
inc_files
include/*.h
)

#
# Source file list
#

file(
GLOB
src_files
ConfigFile/src/*.cpp
```

```
PSKOutput3D/*.cpp
bc/*.cpp
communication/*.cpp
fields/*.cpp
grids/*.cpp
inputoutput/*.cpp
mathlib/*.cpp
mpidata/*.cpp
particles/*.cpp
performances/*.cpp
processtopology/*.cpp
solvers/*.cpp
utility/*.cpp
main/*.cpp
)

#
# Macro definitions
#

set(TEST_B $ENV{BATSRUS})
if(DEFINED TEST_B)
add_definitions( -DBATSRUS )
message(" WARNING: BATSRUS flag is active.")
else(DEFINED TEST_B)
message(" INFO: BATSRUS is not active.")
endif(DEFINED TEST_B)

#
# Executable declaration
#

# Particle solver
add_executable(
iPIC3D
iPIC3D.cpp
)

option(STATIC_LINK "Choose static link" on)
if(STATIC_LINK)
add_library(
iPIC3Dlib     #name of the library
STATIC    #type of the library
${inc_files}  # stuff to build the library
${src_files}
)
else()
add_library(
iPIC3Dlib     #name of the library
SHARED    #type of the library
${inc_files}  # stuff to build the library
${src_files}
)
endif()


#
# Link external libraries
#
target_link_libraries(
iPIC3Dlib
${HDF5_LIBRARIES}
${HDF5_HL_LIBRARIES}
${MPI_LIBRARIES}
${EXTRA_LIBS}
)

target_link_libraries(
iPIC3D
iPIC3Dlib
)

## to save the executable in the folder where the CMakeLists.txt file is, i.e. CMAKE_CURRENT_SOURCE_DIR
set_target_properties(iPIC3D PROPERTIES RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR})

## debug releases have a _d appended to the executable
set_target_properties(iPIC3D PROPERTIES DEBUG_POSTFIX "_d")

message("Which system am I compiling for:")
message("MYHOSTNAME is ${myhostname}")
message("CMAKE_SYSTEM_PROCESSOR is ${CMAKE_SYSTEM_PROCESSOR}")
message("Compiler & compiler flags:")
message("CMAKE_CXX_COMPILER is ${CMAKE_CXX_COMPILER}")
message("CMAKE_CXX_FLAGS is ${CMAKE_CXX_FLAGS}")
message("HDF5_INCLUDE_DIRS is ${HDF5_INCLUDE_DIRS}")
message("HDF5_LIBRARIES" is ${HDF5_LIBRARIES})
message("HDF5_HL_LIBRARIES is ${HDF5_HL_LIBRARIES}")
message("MPI_LIBRARY_DIRS is ${MPI_LIBRARY_DIRS}")
```

Appendix D Parameters.h

Listing 4: Parameter.h

```
1  #include "Parameters.h"
2
3  using namespace Parameters;
4
5  //********** edit these parameters *********
6  // parameters relevant to parallelization
7  //
8  // This should be a multiple of the width of the vector unit
9  // as measured in double precision floating point numbers
10 int Parameters::get_multiple_of_vector_width_in_doubles(){ return 8; }
11 //
12 // parameters relevant to parallel IO
13 //bool Parameters::call_H5Block3dSetChunk() { return true; }
14 //
15 // parameters relevant to particle communication
16 //
17 // int Parameters::get_blockSize() { return 64; }
18 int Parameters::get_blockSize() { return 8192; }
19 int Parameters::get_numBlocks() { return 4; }
20 //
21 // parameters relevant to parallelization
22 //
23 bool Parameters::get_VECTORIZE_MOMENTS() { return false; }
24 // supported options: SoA AoS
25 Parameters::Enum Parameters::get_MOMENTS_TYPE() { return AoS; }
26 // supported options: SoA AoS AoSvec AoSintr AoS_vec_onesort SoA_vec_resort
27 Parameters::Enum Parameters::get_MOVER_TYPE() { return AoS; }
28 //********** derived parameters *********
29
30 static bool SORTING_PARTICLES;
31 static bool RESORTING_PARTICLES;
32 static bool USING_AOS;
33 static bool SORTING_SOA;
34
35 void Parameters::init_parameters()
36 {
37   RESORTING_PARTICLES =
38     get_MOVER_TYPE()==SoA_vec_resort
39     || get_MOVER_TYPE()==AoS_vec_resort;
40   SORTING_PARTICLES = get_VECTORIZE_MOMENTS()
41     || get_MOVER_TYPE()==SoA_vec_onesort
42     || get_MOVER_TYPE()==AoS_vec_onesort
43     || get_MOVER_TYPE()==SoA_vec_resort
44     || get_MOVER_TYPE()==AoS_vec_resort;
45   SORTING_SOA = get_VECTORIZE_MOMENTS()
46     || get_MOVER_TYPE()==SoA_vec_onesort
47     || get_MOVER_TYPE()==SoA_vec_resort;
48   USING_AOS =
49     get_MOMENTS_TYPE()==AoS
50     || get_MOVER_TYPE()==AoS
51     || get_MOVER_TYPE()==AoSintr
52     || get_MOVER_TYPE()==AoS_vec_onesort
53     || get_MOVER_TYPE()==AoS_vec_resort;
54 }
55
56 bool Parameters::get_RESORTING_PARTICLES() { return RESORTING_PARTICLES; }
57 bool Parameters::get_SORTING_PARTICLES() { return SORTING_PARTICLES; }
58 bool Parameters::get_SORTING_SOA() { return SORTING_SOA; }
59 bool Parameters::get_USING_AOS() { return USING_AOS; }
60
61 bool Parameters::get_doWriteOutput() { return true; }
```

There are three main options that can be set in Parameter.h. The default values are good for the most common simulation set-ups. It is not advisable that users change the default values unless they want to experiment on a different machine. The three main options are:

1. Line 19: $int\ Parameters :: get\_numBlocks()\ \{return 4;\}$ By changing the return value from 4 to larger values, each process will reserve more blocks for the particle communication. This change should only be used when the number of exiting particles is large and particle communication is taking significant time.
2. Line 61: $bool\ Parameters :: get\_doWriteOutput()\ \{return\ true;\}$ By switching the return value from true to false, the simulation will not write any output files during simulation. This change may be used when doing some tests that need to exclude impact from I/O.
3. Line 27: $Parameters :: EnumParameters :: get\_MOVER\_TYPE()\ \{return\ AoS;\}$ This option switches the data structure for particles. The standard one is AoS, i.e array of structures. User should choose this data representation for most cases unless they are developing new schemes using a different data structure, e.g. SoA, structure of array.

References

[1] Markidis, S., Lapenta, G., and Rizwan-uddin (2010). Multi-scale simulations of plasma with ipic3d. *Math. Comput. Simul.*, 80(7):1509–1519.

[2] Peng, I. B., Vencels, J., Lapenta, G., Divin, A., Vaivads, A., Laure, E., and Markidis, S. (2015). Energetic particles in magnetotail reconnection. *Journal of Plasma Physics*, 81(2).