PHARE Documentation

Release 1.0

Nicolas Aunai, Roch Smets, Philip Deegan

May 15, 2024

THEORY

PHARE is a Hybrid Particle-In-Cell (PIC) code. It solves the evolution of the Vlasov equation of an arbitrary number of ion populations in a Lagrangian way. Electrons are modeled as a single fluid. Their momentum equation is used to compute the electric field, assuming quasineutrality.

Using Adaptive Mesh Refinement, provided by the library SAMRAI, PHARE aims at filling the gap between sub-ion scales and large "MHD" scales by increasing the mesh resolution wherever the solution needs it.

CHAPTER

ONE

THE HYBRID PIC FORMALISM

The Hybrid formalism consists in modeling the plasma as a combination of constituants treated with a different physical models. This usually means that ions and electrons are treated differently. A rather complete description of the different ways a code can be "hybrid" is given in The Hybrid Multiscale Simulation Technology by A.S. Lipatov. In astrophysical and space applications, the main application domains of PHARE, "hybrid" usually means that ions are considered at the kinetic level while electrons are considered as a fluid. This is the case for PHARE and this is what we mean by "hybrid" on this page.

1.1 The ion equations

The hybrid model consists in evolving in space $\mathbf{r}$ and time t, the velocity distribution function f_p of each ion populations p under the influence of the electric $\mathbf{E}$ and magnetic field $\mathbf{B}$. This is done by solving the Vlasov equation for all ion populations when collisions are negligible.

$$\frac{\partial f_p}{\partial t} + \mathbf{v} \cdot \frac{\partial f_p}{\partial \mathbf{r}} + \frac{q_p}{m_p}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f_p}{\partial \mathbf{v}} = 0 \tag{1.1}$$

Having the new distribution everywhere at $t + \Delta t$, it is easy to calculate the ion moments as the sum of the moments of all populations. Namely, for the ion density n_i and bulk velocity $\mathbf{u_i}$

$$\begin{aligned} to \\ n_i(\mathbf{r}, t) = \\ \sum_p \int f_p(\mathbf{r}, \mathbf{v}, t) d\mathbf{v} \\ \mathbf{u}_i(\mathbf{r}, t) = \\ \frac{1}{n_i} \sum_p \int \mathbf{v} f_p(\mathbf{r}, \mathbf{v}, t) d\mathbf{v} \end{aligned} \tag{1.2}$$

=

$$= \sum_p \int f_p(\mathbf{r}, \mathbf{v}, t) d\mathbf{v} \mathbf{u}_i(\mathbf{r}, t)$$

$$\frac{1}{n_i} \sum_p \int \mathbf{v} f_p(\mathbf{r}, \mathbf{v}, t) d\mathbf{v}$$

1.2 The electron momentum equation

What about the electrons? Remember? They are assumed to behave as a fluid. This is wrong of course in collisionless systems since nothing makes the density, velocity etc. of the electrons to depend on purely local physics as collisions would in a "real" fluid. But that's an approximation the hybrid formalism makes to simplify the physics (and make simulation lighter) compared to the fully kinetic system and that is already a much more realistic way of modeling the plasma and say, single fluid magnetohydrodynamics. Now there are subtleties. The electron momentum equation is:

$$m_e n_e \frac{d\mathbf{u_e}}{dt} = -\nabla \cdot \mathbf{P_e} - e n_e (\mathbf{E} + \mathbf{u_e} \times \mathbf{B})$$

1.3 Electromagnetic field equations

"Treating electrons as a fluid", you probably think we solve that equation, in contrast to the Vlasov equation we used for ions. Well not really… But let's say we did. Now we would have to wonder where the magnetic field and electric field would come from. For the magnetic field, the answer is easy. We just use the Maxwell-Faraday equation:

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

What about the electric field now? There is all the trick of Hybrid codes. We actually do not solve the electron momentum equation directly to get the new electron fluid momentum. Instead we make assumptions on the electron fluid, and use that momentum equation to calculate the electric field ! Thus, the momentum equation is re-written:

$$\mathbf{E} = -\mathbf{u_e} \times \mathbf{B} - \frac{1}{e n_e} \nabla \cdot \mathbf{P_e} + \frac{m_e}{e} \frac{d\mathbf{u_e}}{dt}$$

1.4 Quasineutrality

At this point, the equation for the electric field still has unknowns. The most obvious perhaps is n_e the electron particle density. This is where the hybrid formalism makes the assumption that at the scale we solve the equations, the plasma is quasineutral, and thus we can neglect the difference between n_i and n_e and have only one variable n: the plasma density. Since we have the total ion density already, that's our n. Quasineutrality enable us to get the electron bulk velocity from the known ion bulk velocity and the electric current density:

$$\mathbf{u_e} = \mathbf{u_i} - \frac{\mathbf{j}}{en}$$

The total current density is obtained from the static Maxwell-Ampere equation, meaning we neglect the displacement current:

$$\mu_0 \mathbf{j} = \nabla \times \mathbf{B}$$

The electric field is now equal to

$$\mathbf{E} = -\mathbf{u_e} \times \mathbf{B} - \frac{1}{en} \nabla \cdot \mathbf{P_e} + \frac{m_e}{e} \frac{d\mathbf{u_e}}{dt}$$

1.5 Massless electrons

The next assumption usually made in Hybrid codes, that is also made in PHARE, is that the spatial and time scales at which we are interested in are much larger and longer that the scales at which the electron bulk inertia matters. The electrons being so light compare to even protons, that it is mostly ok to neglect the last term of, which now reads:

$$\mathbf{E} = -\mathbf{u_e} \times \mathbf{B} - \frac{1}{en} \nabla \cdot \mathbf{P_e}$$

1.6 Electron closure

Since we do not have an electron distribution function in hand, the pressure is not known a priori. Hybrid codes thus have to come with a so-called closure equation which role is to give us the pressure everywhere at time t, based on some assumption on the system. Usually, unless in very specific conditions, there is no rigorous way of getting such equation and most hybrid code assume a closure that is "reasonable" and above all "simple" to calculate.

Perhaps the simplest and most used electron closure is the isothermal one. This simply say that the electron pressure P_e is given by the product of the density by some scalar constant that we call "the electron temperature".

$$P_e = nT_e$$

1.7 Dissipative terms

Using above equations to calculate the electric field would result in current sheets to collapse at grid scale in the absence of an intrinsic dissipation scale in the system. Too ways are typically employed in Hybrid codes to include such a dissipation. Joule resistivity well known to be used already in MHD codes. It is a simple term $\eta\mathbf{j}$ to add on the right hand side of the electric field equation. This term adds diffusion of magnetic flux. However there is no scale at which this terms dominate over the electron ideal term $-\mathbf{u_e} \times \mathbf{B}$, unless η is so large that ion scale structures are diffused away too.

Another term that can be employed is the so-called hyper-resistivity (sometimes called hyper-viscosity) that takes the form $-\nu\nabla^2\mathbf{j}$ In contrast to classical resistivity, this terms (due to the second order derivative) comes with an intrinsic scale at which it is dominant over electron convection term and efficiently adds sub-ion scale dissipation.

PHARE include these two terms and the electric field is obtained via :

$$\mathbf{E} = -\mathbf{u_e} \times \mathbf{B} - \frac{1}{en}\nabla P_e + \eta\mathbf{j} - \nu\nabla^2\mathbf{j}$$

CHAPTER

TWO

THE PARTICLE-IN-CELL FORMALISM

There are two ways to solve the Vlasov equation for ion populations. It can be solved calculating eulerian derivatives, i.e. discretizing velocity and spatial dimensions and solving the equation at those fixed locations. This is called a "Vlasov Hybrid code". It is generally complex and require lots of computational resources. The other way consists in adopting a Lagrangian viewpoint. That is, cutting the initial distribution function in N weighted bins and follow the dynamics of those chunks in phase space. The little pieces of distributions are called "macro-particles". Decomposing the distribution function of the population into the contribution of macro-particles in the following way is the base of the so-called "Particle-in-Cell" (PIC) method.

$$f_p(\mathbf{r}, \mathbf{v}, t) = \sum_m^{N_p} w_m S(\mathbf{r} - \mathbf{r}_m(t)) \delta(\mathbf{v} - \mathbf{v}_m(t))$$

where $\mathbf{r}_m$ and $\mathbf{v}_m$ are the position and velocity of the m_{th} macro-particle. w_m represents the weight of that macro-particle, i.e. how much it counts in the evaluation of f_p. δ is the Dirac function, which says that a macro-particle represent a specific velocity in the distribution. In contrast, the function S is a finite support function representing the "shape" of the macro-particle in the spatial dimension. This function tells us how far from the macro-particle a local distribution sees its influence. In PHARE we use b-splinefunctions to model S. PHARE uses b-splines of the first, second and third order. The higher the order the further a macro-particle influences the distribution, but the longer it takes to compute it.

CHAPTER

THREE

SPATIAL DISCRETIZATION

CHAPTER

FOUR

TEMPORAL DISCRETIZATION

CHAPTER

FIVE

ADAPTIVE MESH REFINEMENT

CHAPTER

SIX

GETTING PHARE

6.1 Lastest commit

```
git clone --recursive https://github.com/PHAREHUB/PHARE
```

6.2 Latest release

6.3 Previous releases

CHAPTER

SEVEN

BUILD PHARE

7.1 Build for production

```
cd path/to/dir/containing/PHARE
mkdir build
cd build
cmake -DCMAKE_CXX_FLAGS="-O3 -march=native -mtune=native" -DCMAKE_BUILD_TYPE=Release   ..
↪/PHARE
make -j
```

7.2 Build for debugging

```
cd path/to/dir/containing/PHARE
mkdir build
cd build
cmake -DCMAKE_CXX_FLAGS="-g3 -O0 -march=native -mtune=native" -DCMAKE_BUILD_TYPE=Debug  -
↪DCMAKE_CXX_FLAGS="-DPHARE_DIAG_DOUBLES=1"  ../PHARE
make -j
```

CHAPTER

EIGHT

RUN PHARE FROM PYTHON

PHARE can run as a python script.

8.1 Python dependencies

PHARE requires a minimmum version of python 3.8 to run properly. Make sure *python3* shows the version is at least 3.8. Python package dependencies are listed in *requirements.txt* file. Install dependencies for the user:

```
pip3 install --user -r requirements.txt
```

Install dependencies in a virtual environment:

```
python3 -m venv phare_venv
source phare_venv/bin/activate
pip3 install -r requirements.txt
```

8.2 Running PHARE

First, make sure it is accessible to python. Assuming PHARE source directory is in */path/to/PHARE*, and the build directory is

/path/to/build/, then use the following to let python know where to find PHARE:

```
export PYTHONPATH=/path/to/PHARE/pyphare:/path/to/build:$PYTHONPATH
```

Write a [simulation input script](../simulation_inputs.md) and run the following command:

```
python3 /path/to/my_script.py
```

CHAPTER

NINE

RUN FROM THE PHARE BINARY

CHAPTER

TEN

SIMULATION INPUTS

10.1 Python script structure

PHARE takes python scripts as inputs. They consists in declaring various blocks as follows.

```
# ------------ MANDATORY BLOCKS

Simulation(
    # some parameters
    # configuring numerical and AMR
    # parameters
)


MawellianFluidModel(
   # some parameters
   # configuring the magnetic field profile
   # and ion initial condition
   # as fluid moment profiles
   # ion particles are assumed to follow
   # locally Maxwellian distributions with these moments
   )

# ------------ END OF MANDATORY BLOCKS


# ------------ OPTIONAL BLOCKS

ElectronModel(
    # configures electron fluid properties
)



ElectromagDiagnostics(
   # parameters configuring outputs
   # of E and B
)
```

(continues on next page)

(continued from previous page)

```
FluidDiagnostics(
    # parameters configuring ion moment outputs
)


ParticleDiagnostics(
    # some parameters configuring particle outputs
)


# ------------ END OF OPTIONAL BLOCKS
```

10.2 The Simulation block

The Simulation class is used to set general parameters to the simulation like the integration time, domain size, interpolation order, or adaptive meshing. The `Simulation` must be the first block defined in an input script

10.3 Magnetic field and ions

10.4 Electron model

10.5 Diagnostics

10.5.1 Electromagnetic Diagnostics

10.5.2 Moment Diagnostics

10.5.3 Particle Diagnostics

10.5.4 Meta-data Diagnostics

CHAPTER

ELEVEN

EXAMPLES

CHAPTER

TWELVE

DATA ANALYSIS

12.1 Getting Data

12.2 Python Patch Hierarchy

12.3 Using the finest field available

CHAPTER

THIRTEEN

PLOTTING FIELDS

CHAPTER

FOURTEEN

PLOTTING PARTICLE DISTRIBUTIONS

CHAPTER

FIFTEEN

PHARE TESTS

15.1 Continuous integration

15.2 Unit tests

15.3 Functional tests

- genindex
- modindex
- search