WarpX Documentation

WarpX collaboration

Mar 31, 2020

Contents

Warning: This is an **alpha release** of WarpX. The code is still in active development. Robustness and performance may fluctuate at this stage. The input and output formats may evolve.

WarpX is an advanced **electromagnetic Particle-In-Cell** code.

It supports many features including:

- Perfectly-Matched Layers (PML)
- Boosted-frame simulations
- Mesh refinement

For details on the algorithms that WarpX implements, see the section *Theoretical background*.

In addition, WarpX is a highly-parallel and highly-optimized code and features hybrid OpenMP/MPI parallelization, advanced vectorization techniques and load balancing capabilities.

In order to learn to use the code, please see the sections below:

CHAPTER 1

Building/installing WarpX

WarpX can be built with various options. This page describes the most basic build, and points to instructions for more advanced builds.

Even if you are interested in more advanced builds, we recommend reading this page first.

1.1 Downloading the source code

Clone the source codes of WarpX, and its dependencies AMReX and PICSAR into one single directory (e.g. `warpx_directory`):

```
mkdir warpx_directory
cd warpx_directory
git clone --branch dev https://github.com/ECP-WarpX/WarpX.git
git clone --branch master https://bitbucket.org/berkeleylab/picsar.git
git clone --branch development https://github.com/AMReX-Codes/amrex.git
```

1.2 Basic compilation

WarpX requires a C/C++ and Fortran compiler (e.g., GCC or Intel) and an MPI implementation (e.g., OpenMPI or MPICH). Then `cd` into the directory `WarpX` and type

```
make -j 4
```

This will generate an executable file in the `Bin` directory.

Note: The compilation options are set in the file `GNUmakefile`. The default options correspond to an optimized code for 3D geometry. You can modify the options in this file in order to (for instance):

- Use 2D geometry

- Disable OpenMP
- Profile or debug the code
- Choose a given compiler

For a description of these different options, see the corresponding page in the AMReX documentation.

Alternatively, instead of modifying the file `GNUmakefile`, you can directly pass the options in command line ; for instance:

```
make -j 4 USE_OMP=FALSE
```

In order to clean a previously compiled version:

```
make realclean
```

1.3 Advanced building instructions

1.3.1 Building WarpX with support for openPMD output

WarpX can dump data in the openPMD format. This feature currently requires to have a parallel version of HDF5 installed ; therefore we recommend to use spack in order to facilitate the installation.

More specifically, we recommend that you try installing the openPMD-api library 0.9.0a or newer using spack (first section below). If this fails, a back-up solution is to install parallel HDF5 with spack, and then install the openPMD-api library from source.

In order to install spack, you can simply do:

```
git clone https://github.com/spack/spack.git
export SPACK_ROOT=/path/to/spack
. $SPACK_ROOT/share/spack/setup-env.sh
```

(You may want to add the last 2 lines to your `.bashrc` file.)

Building openPMD support, by installing openPMD-api directly from spack

First, install the openPMD-api library:

```
spack install openpmd-api -python +adios1
```

Then, `cd` into the `WarpX` folder, and type:

```
spack load mpi
spack load openpmd-api
make -j 4 USE_OPENPMD=TRUE
```

You will also need to load the same spack environment when running WarpX, for instance:

```
spack load mpi
spack load openpmd-api

mpirun -np 4 ./warpx.exe inputs
```

Building openPMD support, by installing openPMD-api from source

First, install the openPMD-api library, and load it in your environment:

```
spack install hdf5
spack install adios
spack load -r hdf5
spack load -r adios
```

Then, in the *warpx_directory*, download and build the openPMD API:

```
git clone https://github.com/openPMD/openPMD-api.git
mkdir openPMD-api-build
cd openPMD-api-build
cmake ../openPMD-api -DopenPMD_USE_PYTHON=OFF -DCMAKE_INSTALL_PREFIX=../openPMD-
↪install/ -DCMAKE_INSTALL_RPATH_USE_LINK_PATH=ON -DCMAKE_INSTALL_RPATH='$ORIGIN'
cmake --build . --target install
```

Finally, compile WarpX:

```
cd ../WarpX
export PKG_CONFIG_PATH=$PWD/../openPMD-install/lib/pkgconfig:$PKG_CONFIG_PATH
make -j 4 USE_OPENPMD=TRUE
```

You will also need to load the same spack environment when running WarpX, for instance:

```
spack load openmpi
spack load hdf5
spack load adios

mpirun -np 4 ./warpx.exe inputs
```

1.3.2 Building the spectral solver

By default, the code is compiled with a finite-difference (FDTD) Maxwell solver. In order to run the code with a spectral solver, you need to:

- Install (or load) an MPI-enabled version of FFTW. For instance, for Debian, this can be done with

  ```
  apt-get install libfftw3-dev libfftw3-mpi-dev
  ```

- Set the environment variable `FFTW_HOME` to the path for FFTW. For instance, for Debian, this is done with

  ```
  export FFTW_HOME=/usr/
  ```

- Set `USE_PSATD=TRUE` when compiling:

  ```
  make -j 4 USE_PSATD=TRUE
  ```

Note that this is not compatible with USE_RZ yet.

1.3.3 Building WarpX to use RZ geometry

WarpX can be built to run with RZ geometry. Currently, this only allows pure axisymmetry (i.e. mode 0) with an FDTD solver.

To select RZ geometry, set the flag USE_RZ = TRUE when compiling:

```
make -j 4 USE_RZ=TRUE
```

Note that this sets DIM=2, which is required with USE_RZ=TRUE. The executable produced will have "RZ" as a suffix. Currently does not work with USE_PSATD.

1.3.4 Building WarpX with GPU support (Linux only)

Warning: In order to build WarpX on a specific GPU cluster (e.g. Summit), look for the corresponding specific instructions, instead of those on this page.

In order to build WarpX with GPU support, make sure that you have *cuda* and *mpich* installed on your system. (Compiling with *openmpi* currently fails.) Then compile WarpX with the option *USE_GPU=TRUE*, e.g.

```
make -j 4 USE_GPU=TRUE
```

1.3.5 Installing WarpX as a Python package

Type

```
make -j 4 USE_PYTHON_MAIN=TRUE
```

or edit the GNUmakefile and set *USE_PYTHON_MAIN=TRUE*, and type

```
make -j 4
```

This will compile the code, and install the Python bindings as a package (named `pywarpx`) in your standard Python installation (i.e. in your `site-packages` directory). The note on compiler options from the previous section also holds when compiling the Python package.

In case you do not have write permissions to the default Python installation (e.g. typical on computer clusters), use the following command instead:

```
make -j 4 PYINSTALLOPTIONS=--user
```

In this case, you can also set the variable *PYTHONUSERBASE* to set the folder where *pywarpx* will be installed.

1.3.6 Building WarpX with Spack

WarpX can be installed using Spack. From the Spack web page: "Spack is a package management tool designed to support multiple versions and configurations of software on a wide variety of platforms and environments."

Spack is available from github. Spack only needs to be cloned and can be used right away - there are no installation steps. The spack command, "spack/bin/spack", can be used directly or "spack/bin" can be added to your execute path.

WarpX is built with the single command

```
spack install warpx
```

This will build the 3-D version of WarpX using the master branch. At the very end of the output from build sequence, Spack tells you where the WarpX executable has been placed. Alternatively, the "spack load" command can be configured so that "spack load warpx" will put the executable in your execute path.

To build using the dev branch, the command is

```
spack install warpx@dev
```

Other variants of WarpX can be installed, for example

```
spack install warpx dims=2
```

will build the 2-D version.

```
spack install warpx debug=True
```

will build with debugging turned on.

```
spack install warpx %intel
```

will build using the intel compiler (instead of gcc).

The Python verson of WarpX is not yet available with Spack.

1.4 Building for specific platforms

1.4.1 Building WarpX for Cori (NERSC)

Standard build

For the Cori cluster at NERSC, you need to type the following command when compiling:

Note: In order to compile the code with a spectral solver, type

```
module load cray-fftw
```

before typing any of the commands below, and add `USE_PSATD=TRUE` at the end of the command containing `make`.

In order to compile for the **Haswell architecture**:

- with the Intel compiler

```
make -j 16 COMP=intel
```

- with the GNU compiler

```
module swap PrgEnv-intel PrgEnv-gnu
make -j 16 COMP=gnu
```

In order to compile for the **Knight's Landing (KNL) architecture**:

- with the Intel compiler

```
module swap craype-haswell craype-mic-knl
make -j 16 COMP=intel
```

- with the GNU compiler

```
module swap craype-haswell craype-mic-knl
module swap PrgEnv-intel PrgEnv-gnu
make -j 16 COMP=gnu
```

See *Running on specific platforms* for more information on how to run WarpX on Cori.

GPU Build

To compile on the experimental GPU nodes on Cori, you first need to purge your modules, most of which won't work on the GPU nodes.

```
module purge
```

Then, you need to load the following modules:

```
module load modules esslurm gcc/7.3.0 cuda mvapich2
```

You can also use OpenMPI-UCX instead of mvapich: openmpi/4.0.1-ucx-1.6.

Then, you need to use slurm to request access to a GPU node:

```
salloc -C gpu -N 1 -t 30 -c 10 --gres=gpu:1 -A m1759
```

This reserves 10 logical cores (5 physical), 1 GPU. The latest documentation can be found here: https://docs-dev.nersc.gov/cgpu/access Note that you can't cross-compile for the GPU nodes - you have to log on to one and then build your software.

Finally, navigate to the base of the WarpX repository and compile in GPU mode:

```
make -j 16 USE_GPU=TRUE
```

Building WarpX with openPMD support

First, load the appropriate modules:

```
module swap craype-haswell craype-mic-knl
module swap PrgEnv-intel PrgEnv-gnu
module load cmake/3.14.4
module load cray-hdf5-parallel
module load adios/1.13.1
export CRAYPE_LINK_TYPE=dynamic
```

Then, in the *warpx_directory*, download and build the openPMD API:

```
git clone https://github.com/openPMD/openPMD-api.git
mkdir openPMD-api-build
cd openPMD-api-build
cmake ../openPMD-api -DopenPMD_USE_PYTHON=OFF -DCMAKE_INSTALL_PREFIX=../openPMD-
↪install/ -DCMAKE_INSTALL_RPATH_USE_LINK_PATH=ON -DCMAKE_INSTALL_RPATH='$ORIGIN'
cmake --build . --target install
```

Finally, compile WarpX:

```
cd ../WarpX
export PKG_CONFIG_PATH=$PWD/../openPMD-install/lib64/pkgconfig:$PKG_CONFIG_PATH
make -j 16 COMP=gnu USE_OPENPMD=TRUE
```

In order to run WarpX, load the same modules again.

1.4.2 Building WarpX for Summit (OLCF)

For the Summit cluster at OLCF, use the following commands to download the source code, and switch to the correct branch:

```
mkdir warpx_directory
cd warpx_directory

git clone --branch dev https://github.com/ECP-WarpX/WarpX.git
git clone --branch master https://bitbucket.org/berkeleylab/picsar.git
git clone --branch development https://github.com/AMReX-Codes/amrex.git
```

Then, `cd` into the directory `WarpX` and use the following set of commands to compile:

```
module load gcc
module load cuda
make -j 4 USE_GPU=TRUE
```

See *Running on specific platforms* for more information on how to run WarpX on Summit.

CHAPTER 2

Running WarpX as an executable

2.1 How to run a new simulation

After compiling the code, the WarpX executable is stored in the folder `warpx/Bin`. (Its name starts with *main* but depends on the compiler options.)

In order to run a new simulation:

- Create a **new directory**, where the simulation will be run.
- Copy the **executable** to this directory:

```
cp warpx/Bin/<warpx_executable> <run_directory>/warpx.exe
```

where `<warpx_executable>` should be replaced by the actual name of the executable (see above) and `<run_directory>` by the actual path to the run directory.

- Add an **input file** in the directory.

This file contains the numerical and physical parameters that define the situation to be simulated. Example input files can be found in the section *Example input files*. The different parameters in these files are explained in the section *Input parameters*.

- **Run** the executable:

```
mpirun -np <n_ranks> ./warpx.exe <input_file>
```

where `<n_ranks>` is the number of MPI ranks used, and `<input_file>` is the name of the input file.

2.2 Example input files

This section allows you to **download input files** that correspond to different physical situations. For a definition of the different parameters that are set in these files, see the section *Input parameters*.

2.2.1 Beam-driven acceleration

- `2D case`
- `2D case in boosted frame`
- `3D case in boosted frame`

2.2.2 Laser-driven acceleration

- `2D case`
- `2D case in boosted frame`
- `3D case`

2.2.3 Plasma mirror

`2D case`

2.2.4 Uniform plasma

`2D case` `3D case`

2.3 Input parameters

Warning: This section is currently in development.

2.3.1 Overall simulation parameters

- **`max_step`** **(*integer*)** The number of PIC cycles to perform.
- **`warpx.gamma_boost`** **(*float*)** The Lorentz factor of the boosted frame in which the simulation is run. (The corresponding Lorentz transformation is assumed to be along `warpx.boost_direction`.)

  When using this parameter, some of the input parameters are automatically converted to the boosted frame. (See the corresponding documentation of each input parameters.)

  Note: For now, only the laser parameters will be converted.

- **`warpx.boost_direction`** **(string: `x`, `y` or `z`)** The direction of the Lorentz-transform for boosted-frame simulations (The direction `y` cannot be used in 2D simulations.)
- **`warpx.zmax_plasma_to_compute_max_step`** **(*float*) optional** Can be useful when running in a boosted frame. If specified, automatically calculates the number of iterations required in the boosted frame for the lower z end of the simulation domain to reach `warpx.zmax_plasma_to_compute_max_step` (typically the plasma end, given in the lab frame). The value of `max_step` is overwritten, and printed to standard output. Currently only works if the Lorentz boost and the moving window are along the z direction.
- **`warpx.verbose`** **(*0* or *1*)** Controls how much information is printed to the terminal, when running WarpX.

2.3.2 Setting up the field mesh

- **`amr.n_cell` (*2 integers in 2D*, *3 integers in 3D*)** The number of grid points along each direction (on the **coarsest level**)
- **`amr.max_level` (*integer*)** When using mesh refinement, the number of refinement levels that will be used.

  Use 0 in order to disable mesh refinement.
- **`geometry.is_periodic` (*2 integers in 2D*, *3 integers in 3D*)** Whether the boundary conditions are periodic, in each direction.

  For each direction, use 1 for periodic conditions, 0 otherwise.
- **`geometry.coord_sys` (*integer*) optional (default *0*)** Coordinate system used by the simulation. 0 for Cartesian, 1 for cylindrical.
- **`geometry.prob_lo` and `geometry.prob_hi` (*2 floats in 2D*, *3 integers in 3D*; in meters)** The extent of the full simulation box. This box is rectangular, and thus its extent is given here by the coordinates of the lower corner (`geometry.prob_lo`) and upper corner (`geometry.prob_hi`). The first axis of the coordinates is x (or r with cylindrical) and the last is z.
- **`warpx.fine_tag_lo` and `warpx.fine_tag_hi` (*2 floats in 2D*, *3 integers in 3D*; in meters) optional When using static mesh refinement with 1 level**, the extent of the refined patch. This patch is rectangular, and thus its extent is given here by the coordinates of the lower corner (`warpx.fine_tag_lo`) and upper corner (`warpx.fine_tag_hi`).
- **`warpx.n_current_deposition_buffer` (*integer*)** When using mesh refinement: the particles that are located inside a refinement patch, but within `n_current_deposition_buffer` cells of the edge of this patch, will deposit their charge and current to the lower refinement level, instead of depositing to the refinement patch itself. See the section *Mesh refinement* for more details. If this variable is not explicitly set in the input script, `n_current_deposition_buffer` is automatically set so as to be large enough to hold the particle shape, on the fine grid
- **`warpx.n_field_gather_buffer` (*integer*; 0 by default)** When using mesh refinement: the particles that are located inside a refinement patch, but within `n_field_gather_buffer` cells of the edge of this patch, will gather the fields from the lower refinement level, instead of gathering the fields from the refinement patch itself. This avoids some of the spurious effects that can occur inside the refinement patch, close to its edge. See the section *Mesh refinement* for more details. If this variable is not explicitly set in the input script, `n_field_gather_buffer` is automatically set so that it is one cell larger than `n_current_deposition_buffer`, on the fine grid.
- **`particles.deposit_on_main_grid` (*list of strings*)** When using mesh refinement: the particle species whose name are included in the list will deposit their charge/current directly on the main grid (i.e. the coarsest level), even if they are inside a refinement patch.
- **`particles.gather_from_main_grid` (*list of strings*)** When using mesh refinement: the particle species whose name are included in the list will gather their fields from the main grid (i.e. the coarsest level), even if they are inside a refinement patch.
- **`warpx.n_rz_azimuthal_modes` (*integer*; 1 by default)** When using the RZ version, this is the number of azimuthal modes.

2.3.3 Distribution across MPI ranks and parallelization

- **`amr.max_grid_size` (*integer*) optional (default *128*)** Maximum allowable size of each **subdomain** (expressed in number of grid points, in each direction). Each subdomain has its own ghost cells, and can be handled by a different MPI rank ; several OpenMP threads can work simultaneously on the same subdomain.

If `max_grid_size` is such that the total number of subdomains is **larger** that the number of MPI ranks used, than some MPI ranks will handle several subdomains, thereby providing additional flexibility for **load balancing**.

When using mesh refinement, this number applies to the subdomains of the coarsest level, but also to any of the finer level.

- **`warpx.load_balance_int`** **(*integer*) optional (default *-1*)** How often WarpX should try to redistribute the work across MPI ranks, in order to have better load balancing (expressed in number of PIC cycles inbetween two consecutive attempts at redistributing the work). Use 0 to disable load_balancing.

  When performing load balancing, WarpX measures the wall time for computational parts of the PIC cycle. It then uses this data to decide how to redistribute the subdomains across MPI ranks. (Each subdomain is unchanged, but its owner is changed in order to have better performance.) This relies on each MPI rank handling several (in fact many) subdomains (see `max_grid_size`).

- **`warpx.load_balance_with_sfc`** **(*0* or *1*) optional (default *0*)** If this is *1*: use a Space-Filling Curve (SFC) algorithm in order to perform load-balancing of the simulation. If this is *0*: the Knapsack algorithm is used instead.

- **`warpx.do_dynamic_scheduling`** **(*0* or *1*) optional (default *1*)** Whether to activate OpenMP dynamic scheduling.

2.3.4 Math parser and user-defined constants

WarpX provides a math parser that reads expressions in the input file. It can be used to define the plasma density profile, the plasma momentum distribution or the laser field (see below *Particle initialization* and *Laser initialization*).

The parser reads python-style expressions between double quotes, for instance `"a0*x**2 * (1-y*1.e2) * (x>0)"` is a valid expression where `a0` is a user-defined constant and `x` and `y` are variables. The names are case sensitive. The factor `(x>0)` is *1* where *x>0* and *0* where *x<=0*. It allows the user to define functions by intervals. User-defined constants can be used in parsed functions only (i.e., `density_function(x,y,z)` and `field_function(X,Y,t)`, see below). User-defined constants can contain only letter, numbers and character _. The name of each constant has to begin with a letter. The following names are used by WarpX, and cannot be used as user-defined constants: *x*, *y*, *z*, *X*, *Y*, *t*. For example, parameters `a0` and `z_plateau` can be specified with:

- `my_constants.a0 = 3.0`
- `my_constants.z_plateau = 150.e-6`

2.3.5 Particle initialization

- **`particles.nspecies`** **(*int*)** The number of species that will be used in the simulation.

- **`particles.species_names`** **(*strings*, separated by spaces)** The name of each species. This is then used in the rest of the input deck ; in this documentation we use *<species_name>* as a placeholder.

- **`particles.use_fdtd_nci_corr`** **(*0* or *1*) optional (default *0*)** Whether to activate the FDTD Numerical Cherenkov Instability corrector.

- **`particles.rigid_injected_species`** **(*strings*, separated by spaces)** List of species injected using the rigid injection method. The rigid injection method is useful when injecting a relativistic particle beam, in boosted-frame simulation ; see the section *Inputs and outputs* for more details. For species injected using this method, particles are translated along the $+z$ axis with constant velocity as long as their `z` coordinate verifies `z<zinject_plane`. When `z>zinject_plane`, particles are pushed in a standard way, using the specified pusher. (see the parameter `<species_name>.zinject_plane` below)

- **`<species_name>.charge`** **(*float*)** The charge of one *physical* particle of this species.

- **`<species_name>.mass`** (*float*) The mass of one *physical* particle of this species.
- **`<species_name>.injection_style`** (*string*) Determines how the particles will be injected in the simulation. The options are:
  - `NUniformPerCell`: injection with a fixed number of evenly-spaced particles per cell. This requires the additional parameter `<species_name>.num_particles_per_cell_each_dim`.
  - `NRandomPerCell`: injection with a fixed number of randomly-distributed particles per cell. This requires the additional parameter `<species_name>.num_particles_per_cell`.
  - `gaussian_beam`: Inject particle beam with gaussian distribution in space in all directions. This requires additional parameters: `<species_name>.q_tot` (beam charge), `<species_name>.npart` (number of particles in the beam), `<species_name>.x/y/z_m` (average position in *x/y/z*), `<species_name>.x/y/z_rms` (standard deviation in *x/y/z*), and optional argument `<species_name>.do_symmetrize` (whether to symmetrize the beam in the x and y directions).
- **`<species_name>.num_particles_per_cell_each_dim`** (***3 integers in 3D and RZ, 2 integers in 2D***) With the NUniformPerCell injection style, this specifies the number of particles along each axis within a cell. Note that for RZ, the three axis are radius, theta, and z.
- **`<species_name>.do_continuous_injection`** (***0* or *1***) Whether to inject particles during the simulation, and not only at initialization. This can be required whith a moving window and/or when running in a boosted frame.
- **`<species_name>.profile`** (*string*) Density profile for this species. The options are:
  - `constant`: Constant density profile within the box, or between `<species_name>.xmin` and `<species_name>.xmax` (and same in all directions). This requires additional parameter `<species_name>.density`. i.e., the plasma density in m^{-3}.
  - `parse_density_function`: the density is given by a function in the input file. It requires additional argument `<species_name>.density_function(x,y,z)`, which is a mathematical expression for the density of the species, e.g. `electrons.density_function(x,y,z) = "n0+n0*x**2*1.e12"` where `n0` is a user-defined constant, see above.
- **`<species_name>.density_min`** (*float*) **optional (default *0.*)** Minimum plasma density. No particle is injected where the density is below this value.
- **`<species_name>.density_max`** (*float*) **optional (default *infinity*)** Maximum plasma density. The density at each point is the minimum between the value given in the profile, and *density_max*.
- **`<species_name>.radially_weighted`** (*bool*) **optional (default *true*)** Whether particle's weight is varied with their radius. This only applies to cylindrical geometry. The only valid value is true.
  - `predefined`: use one of WarpX predefined plasma profiles. It requires additional arguments `<species_name>.predefined_profile_name` and `<species_name>.predefined_profile_params` (see below).
- **`<species_name>.momentum_distribution_type`** (*string*) Distribution of the normalized momentum (*u=p/mc*) for this species. The options are:
  - `constant`: constant momentum profile. This requires additional parameters `<species_name>.ux`, `<species_name>.uy` and `<species_name>.uz`, the normalized momenta in the x, y and z direction respectively.
  - `gaussian`: gaussian momentum distribution in all 3 directions. This requires additional arguments for the average momenta along each direction `<species_name>.ux_m`, `<species_name>.uy_m` and `<species_name>.uz_m` as well as standard deviations along each direction `<species_name>.ux_th`, `<species_name>.uy_th` and `<species_name>.uz_th`.

  - `radial_expansion`: momentum depends on the radial coordinate linearly. This requires additional parameter `u_over_r` which is the slope.
  - `parse_momentum_function`: the momentum is given by a function in the input file. It requires additional arguments `<species_name>.momentum_function_ux(x,y,z)`, `<species_name>.momentum_function_uy(x,y,z)` and `<species_name>.momentum_function_uz(x,y,z)`, which gives the distribution of each component of the momentum as a function of space.

- **`<species_name>.zinject_plane` (*float*)** Only read if `<species_name>` is in `particles.rigid_injected_species`. Injection plane when using the rigid injection method. See `particles.rigid_injected_species` above.
- **`<species_name>.rigid_advance` (*bool*)** Only read if `<species_name>` is in `particles.rigid_injected_species`.
  - If `false`, each particle is advanced with its own velocity `vz` until it reaches `zinject_plane`.
  - If `true`, each particle is advanced with the average speed of the species `vzbar` until it reaches `zinject_plane`.
- **`species_name.predefined_profile_name` (*string*)** Only read of `<species_name>.electrons.profile` is *predefined*.
  - If `parabolic_channel`, the plasma profile is a parabolic profile with cosine-like ramps at the beginning and the end of the profile. The density is given by

    $$n = n_0 n(x,y) n(z)$$

    with

    $$n(x,y) = 1 + 4\frac{x^2+y^2}{k_p^2 R_c^4}$$

    where k_p is the plasma wavenumber associated with density n_0. Here, $n(z)$ is a cosine-like up-ramp from 0 to $L_{ramp,up}$, constant to 1 from $L_{ramp,up}$ to $L_{ramp,up} + L_{plateau}$ and a cosine-like down-ramp from $L_{ramp,up} + L_{plateau}$ to $L_{ramp,up} + L_{plateau} + L_{ramp,down}$. All parameters are given in `predefined_profile_params`.
- **`<species_name>.predefined_profile_params` (list of *float*)** Parameters for the predefined profiles.
  - If `species_name.predefined_profile_name` is `parabolic_channel`, `predefined_profile_params` contains a space-separated list of the following parameters, in this order: $L_{ramp,up}$ $L_{plateau}$ $L_{ramp,down}$ R_c n_0
- **`<species_name>.do_backward_propagation` (*bool*)** Inject a backward-propagating beam to reduce the effect of charge-separation fields when running in the boosted frame. See examples.
- **`<species_name>.do_splitting` (*bool*) optional (default *0*)** Split particles of the species when crossing the boundary from a lower resolution domain to a higher resolution domain.
- **`<species_name>.split_type` (*int*) optional (default *0*)** Splitting technique. When *0*, particles are split along the simulation axes (4 particles in 2D, 6 particles in 3D). When *1*, particles are split along the diagonals (4 particles in 2D, 8 particles in 3D).
- **`<species>.plot_species` (*0* or *1* optional; default *1*)** Whether to plot particle quantities for this species.
- **`<species>.plot_vars` (list of *strings* separated by spaces, optional)** List of particle quantities to write to *plotfiles*. By defaults, all quantities are written to file. Choices are

  - `w` for the particle weight,
  - `ux uy uz` for the particle momentum,
  - `Ex Ey Ez` for the electric field on particles,
  - `Bx By Bz` for the magnetic field on particles.

  The particle positions are always included. Use `<species>.plot_vars = none` to plot no particle data, except particle position.

- **`<species>.do_boosted_frame_diags` (*0* or *1* optional, default *1*)** Only used when `warpx.do_boosted_frame_diagnostic=1`. When running in a boosted frame, whether or not to plot back-transformed diagnostics for this species.
- **`warpx.serialize_ics` (*0 or 1*)** Whether or not to use OpenMP threading for particle initialization.
- **`<species>.do_field_ionization` (*0* or *1*) optional (default *0*)** Do field ionization for this species (using the ADK theory).
- **`<species>.physical_element` (*string*)** Only read if *do_field_ionization* = *1*. Symbol of chemical element for this species. Example: for Helium, use `physical_element = He`.
- **`<species>.ionization_product_species` (*string*)** Only read if *do_field_ionization* = *1*. Name of species in which ionized electrons are stored. This species must be created as a regular species in the input file (in particular, it must be in *particles.species_names*).
- **`<species>.ionization_initial_level` (*int*) optional (default *0*)** Only read if *do_field_ionization* = *1*. Initial ionization level of the species (must be smaller than the atomic number of chemical element given in *physical_element*).

2.3.6 Laser initialization

- **`lasers.nlasers` (*int*) optional (default *0*)** Number of lasers pulses.
- **`lasers.names` (list of *string*. Must contain `lasers.nlasers` elements)** Name of each laser. This is then used in the rest of the input deck ; in this documentation we use <*laser_name*> as a placeholder. The parameters below must be provided for each laser pulse.
- **`` `<laser_name>`.position `` (*3 floats in 3D and 2D* ; in meters)** The coordinates of one of the point of the antenna that will emit the laser. The plane of the antenna is entirely defined by `<laser_name>.position` and `<laser_name>.direction`.

  `` `<laser_name>`.position `` also corresponds to the origin of the coordinates system for the laser tranverse profile. For instance, for a Gaussian laser profile, the peak of intensity will be at the position given by `<laser_name>.position`. This variable can thus be used to shift the position of the laser pulse transversally.

  Note: In 2D, `` `<laser_name>`.position `` is still given by 3 numbers, but the second number is ignored.

  When running a **boosted-frame simulation**, provide the value of `<laser_name>.position` in the laboratory frame, and use `warpx.gamma_boost` to automatically perform the conversion to the boosted frame. Note that, in this case, the laser antenna will be moving, in the boosted frame.

- **`<laser_name>.polarization` (*3 floats in 3D and 2D*)** The coordinates of a vector that points in the direction of polarization of the laser. The norm of this vector is unimportant, only its direction matters.

Note: Even in 2D, all the 3 components of this vectors are important (i.e. the polarization can be orthogonal to the plane of the simulation).

- **`<laser_name>.direction` (*3 floats in 3D*)** The coordinates of a vector that points in the propagation direction of the laser. The norm of this vector is unimportant, only its direction matters.

  The plane of the antenna that will emit the laser is orthogonal to this vector.

  Warning: When running **boosted-frame simulations**, `<laser_name>.direction` should be parallel to `warpx.boost_direction`, for now.

- **`<laser_name>.e_max` (*float* ; in V/m)** Peak amplitude of the laser field.

  For a laser with a wavelength $\lambda = 0.8\,\mu m$, the peak amplitude is related to a_0 by:

  $$E_{max} = a_0 \frac{2\pi m_e c}{e\lambda} = a_0 \times (4.0 \cdot 10^{12}\ V.m^{-1})$$

  When running a **boosted-frame simulation**, provide the value of `<laser_name>.e_max` in the laboratory frame, and use `warpx.gamma_boost` to automatically perform the conversion to the boosted frame.

- **`<laser_name>.wavelength` (*float*; in meters)** The wavelength of the laser in vacuum.

  When running a **boosted-frame simulation**, provide the value of `<laser_name>.wavelength` in the laboratory frame, and use `warpx.gamma_boost` to automatically perform the conversion to the boosted frame.

- **`<laser_name>.profile` (*string*)** The spatio-temporal shape of the laser. The options that are currently implemented are:

  – `"Gaussian"`: The transverse and longitudinal profiles are Gaussian.

  – `"Harris"`: The transverse profile is Gaussian, but the longitudinal profile is given by the Harris function (see `<laser_name>.profile_duration` for more details)

  – `"parse_field_function"`: the laser electric field is given by a function in the input file. It requires additional argument `<laser_name>.field_function(X,Y,t)`, which is a mathematical expression , e.g. `<laser_name>.field_function(X,Y,t) = "a0*X**2 * (X>0) * cos(omega0*t)"` where `a0` and `omega0` are a user-defined constant, see above. The profile passed here is the full profile, not only the laser envelope. `t` is time and `X` and `Y` are coordinates orthogonal to `<laser_name>.direction` (not necessarily the x and y coordinates of the simulation). All parameters above are required, but none of the parameters below are used when `<laser_name>.parse_field_function=1`. Even though `<laser_name>.wavelength` and `<laser_name>.e_max` should be included in the laser function, they still have to be specified as they are used for numerical purposes.

- **`<laser_name>.profile_t_peak` (*float*; in seconds)** The time at which the laser reaches its peak intensity, at the position given by `<laser_name>.position` (only used for the `"gaussian"` profile)

  When running a **boosted-frame simulation**, provide the value of `<laser_name>.profile_t_peak` in the laboratory frame, and use `warpx.gamma_boost` to automatically perform the conversion to the boosted frame.

- `<laser_name>.profile_duration` (*float* ; in seconds)

  The duration of the laser, defined as τ below:

  – For the `"gaussian"` profile:

$$E(\boldsymbol{x},t) \propto \exp\left(-\frac{(t-t_{peak})^2}{\tau^2}\right)$$

  – For the `"harris"` profile:

$$E(\boldsymbol{x},t) \propto \frac{1}{32}\left[10 - 15\cos\left(\frac{2\pi t}{\tau}\right) + 6\cos\left(\frac{4\pi t}{\tau}\right) - \cos\left(\frac{6\pi t}{\tau}\right)\right]\Theta(\tau - t)$$

  When running a **boosted-frame simulation**, provide the value of `<laser_name>.profile_duration` in the laboratory frame, and use `warpx.gamma_boost` to automatically perform the conversion to the boosted frame.

- **`<laser_name>.profile_waist` (*float* ; in meters)** The waist of the transverse Gaussian laser profile, defined as w_0 :

$$E(\boldsymbol{x},t) \propto \exp\left(-\frac{\boldsymbol{x}_\perp^2}{w_0^2}\right)$$

- **`<laser_name>.profile_focal_distance` (*float*; in meters)** The distance from `laser_position` to the focal plane. (where the distance is defined along the direction given by `<laser_name>.direction`.)

  Use a negative number for a defocussing laser instead of a focussing laser.

  When running a **boosted-frame simulation**, provide the value of `<laser_name>.profile_focal_distance` in the laboratory frame, and use `warpx.gamma_boost` to automatically perform the conversion to the boosted frame.

- **`<laser_name>.stc_direction` (*3 floats*) optional (default *1. 0. 0.*)** Direction of laser spatio-temporal couplings. See definition in Akturk et al., Opt Express, vol 12, no 19 (2014).

- **`<laser_name>.zeta` (*float*; in meters.seconds) optional (default *0.*)** Spatial chirp at focus in direction `<laser_name>.stc_direction`. See definition in Akturk et al., Opt Express, vol 12, no 19 (2014).

- **`<laser_name>.beta` (*float*; in seconds) optional (default *0.*)** Angular dispersion (or angular chirp) at focus in direction `<laser_name>.stc_direction`. See definition in Akturk et al., Opt Express, vol 12, no 19 (2014).

- **`<laser_name>.phi2` (*float*; in seconds**2) optional (default *0.*)** Temporal chirp at focus. See definition in Akturk et al., Opt Express, vol 12, no 19 (2014).

- **`<laser_name>.do_continuous_injection` (*0* or *1*) optional (default *0*).** Whether or not to use continuous injection. If the antenna starts outside of the simulation domain but enters it at some point (due to moving window or moving antenna in the boosted frame), use this so that the laser antenna is injected when it reaches the box boundary. If running in a boosted frame, this requires the boost direction, moving window direction and laser propagation direction to be along *z*. If not running in a boosted frame, this requires the moving window and laser propagation directions to be the same (*x*, *y* or *z*)

- **`<laser_name>.min_particles_per_mode` (*int*) optional (default *4*)** When using the RZ version, this specifies the minimum number of particles per angular mode. The laser particles are loaded into radial spokes, with the number of spokes given by min_particles_per_mode*(warpx.n_rz_azimuthal_modes-1).

- **`warpx.num_mirrors` (*int*) optional (default *0*)** Users can input perfect mirror condition inside the simulation domain. The number of mirrors is given by `warpx.num_mirrors`. The mirrors are orthogonal to the *z* direction. The following parameters are required when `warpx.num_mirrors` is >0.

- **`warpx.mirror_z` (list of*float*) required if `warpx.num_mirrors>0`** `z` location of the front of the mirrors.

- **`warpx.mirror_z_width` (list of*float*) required if `warpx.num_mirrors>0`** `z` width of the mirrors.

- **`warpx.mirror_z_npoints` (list of *int*) required if `warpx.num_mirrors>0`** In the boosted frame, depending on *gamma_boost*, `warpx.mirror_z_width` can be smaller than the cell size, so that the mirror would not work. This parameter is the minimum number of points for the mirror. If `mirror_z_width < dz/cell_size`, the upper bound of the mirror is increased so that it contains at least `mirror_z_npoints`.

2.3.7 Numerics and algorithms

- **`warpx.cfl` (*float*)** The ratio between the actual timestep that is used in the simulation and the Courant-Friedrichs-Lewy (CFL) limit. (e.g. for *warpx.cfl=1*, the timestep will be exactly equal to the CFL limit.)
- **`warpx.use_filter` (*0 or 1*)** Whether to smooth the charge and currents on the mesh, after depositing them from the macroparticles. This uses a bilinear filter (see the sub-section **Filtering** in *Theoretical background*).
- **`warpx.filter_npass_each_dir` (*3 int*) optional (default *1 1 1*)** Number of passes along each direction for the bilinear filter. In 2D simulations, only the first two values are read.
- **`algo.current_deposition` (*string*, optional)** The algorithm for current deposition. Available options are:
  - `esirkepov`: the charge-conserving Esirkepov algorithm (see Esirkepov, Comp. Phys. Comm. (2001))
  - `direct`: simpler current deposition algorithm, described in the section *The electromagnetic Particle-In-Cell method*. Note that this algorithm is not strictly charge-conserving.

  If `algo.current_deposition` is not specified, the default is `esirkepov`.
- **`algo.charge_deposition` (*string*, optional)** The algorithm for the charge density deposition. Available options are:
  - `standard`: standard charge deposition algorithm, described in the section *The electromagnetic Particle-In-Cell method*.
- **`algo.field_gathering` (*string*, optional)** The algorithm for field gathering. Available options are:
  - `standard`: gathers directly from the grid points (either staggered or nodal gridpoints depending on `warpx.do_nodal`).
- **`algo.particle_pusher` (*string*, optional)** The algorithm for the particle pusher. Available options are:
  - `boris`: Boris pusher.
  - `vay`: Vay pusher (see Vay, Phys. Plasmas (2008))

  If `algo.particle_pusher` is not specified, `boris` is the default.
- **`algo.maxwell_fdtd_solver` (*string*, optional)** The algorithm for the FDTD Maxwell field solver. Available options are:
  - `yee`: Yee FDTD solver.
  - `ckc`: (not available in `RZ` geometry) Cole-Karkkainen solver with Cowan coefficients (see Cowan, PRSTAB 16 (2013))

  If `algo.maxwell_fdtd_solver` is not specified, `yee` is the default.
- **`interpolation.nox`, `interpolation.noy`, `interpolation.noz` (*integer*)** The order of the shape factors for the macroparticles, for the 3 dimensions of space. Lower-order shape factors result in faster simulations, but more noisy results,

Note that the implementation in WarpX is more efficient when these 3 numbers are equal, and when they are between 1 and 3.

- **`warpx.do_dive_cleaning` (*0* or *1* ; default: 0)** Whether to use modified Maxwell equations that progressively eliminate the error in $div(E) - \rho$. This can be useful when using a current deposition algorithm which is not strictly charge-conserving, or when using mesh refinement. These modified Maxwell equation will cause the error to propagate (at the speed of light) to the boundaries of the simulation domain, where it can be absorbed.
- **`warpx.do_nodal` (*0* or *1* ; default: 0)** Whether to use a nodal grid (i.e. all fields are defined at the same points in space) or a staggered grid (i.e. Yee grid ; different fields are defined at different points in space)
- **`warpx.do_subcycling` (*0* or *1*; default: 0)** Whether or not to use sub-cycling. Different refinement levels have a different cell size, which results in different Courant–Friedrichs–Lewy (CFL) limits for the time step. By default, when using mesh refinement, the same time step is used for all levels. This time step is taken as the CFL limit of the finest level. Hence, for coarser levels, the timestep is only a fraction of the CFL limit for this level, which may lead to numerical artifacts. With sub-cycling, each level evolves with its own time step, set to its own CFL limit. In practice, it means that when level 0 performs one iteration, level 1 performs two iterations. Currently, this option is only supported when `amr.max_level = 1`. More information can be found at https://ieeexplore.ieee.org/document/8659392.
- **`psatd.nox`, `psatd.noy`, `pstad.noz` (*integer*) optional (default *16* for all)** The order of accuracy of the spatial derivatives, when using the code compiled with a PSATD solver.
- **`psatd.hybrid_mpi_decomposition` (*0* or *1*; default: 0)** Whether to use a different MPI decomposition for the particle-grid operations (deposition and gather) and for the PSATD solver. If *1*, the FFT will be performed over MPI groups.
- **`psatd.ngroups_fft` (*integer*)** The number of MPI groups that are created for the FFT, when using the code compiled with a PSATD solver (and only if *hybrid_mpi_decomposition* is *1*). The FFTs are global within one MPI group and use guard cell exchanges in between MPI groups. (If `ngroups_fft` is larger than the number of MPI ranks used, than the actual number of MPI ranks is used instead.)
- **`psatd.fftw_plan_measure` (*0* or *1*)** Defines whether the parameters of FFTW plans will be initialized by measuring and optimizing performance (`FFTW_MEASURE` mode; activated by default here). If `psatd.fftw_plan_measure` is set to `0`, then the best parameters of FFTW plans will simply be estimated (`FFTW_ESTIMATE` mode). See this section of the FFTW documentation for more information.
- **`warpx.override_sync_int` (*integer*) optional (default *10*)** Number of time steps between synchronization of sources (*rho* and *J*) on grid nodes at box boundaries. Since the grid nodes at the interface between two neighbor boxes are duplicated in both boxes, an instability can occur if they have too different values. This option makes sure that they are synchronized periodically.

2.3.8 Boundary conditions

- **`warpx.do_pml` (*0* or *1*; default: 1)** Whether to add Perfectly Matched Layers (PML) around the simulation box, and around the refinement patches. See the section *Boundary conditions* for more details.
- **`warpx.pml_ncells` (*int*; default: 10)** The depth of the PML, in number of cells.
- **`warpx.pml_delta` (*int*; default: 10)** The characteristic depth, in number of cells, over which the absorption coefficients of the PML increases.
- **`warpx.do_pml_in_domain` (*int*; default: 0)** Whether to create the PML inside the simulation area or outside. If inside, it allows the user to propagate particles in PML and to use extended PML
- **`warpx.do_pml_has_particles` (*int*; default: 0)** Whether to propagate particles in PML or not. Can only be done if PML are in simulation domain, i.e. if *warpx.do_pml_in_domain = 1*.

- **`warpx.do_pml_j_damping` (*int*; default: 0)** Whether to damp current in PML. Can only be used if particles are propagated in PML, i.e. if *warpx.do_pml_has_particles = 1*.
- **`warpx.do_pml_Lo` (*2 ints in 2D*, *3 ints in 3D*; default: *1 1 1*)** The directions along which one wants a pml boundary condition for lower boundaries on mother grid.
- **`warpx.do_pml_Hi` (*2 floats in 2D*, *3 floats in 3D*; default: *1 1 1*)** The directions along which one wants a pml boundary condition for upper boundaries on mother grid.

2.3.9 Diagnostics and output

- **`amr.plot_int` (*integer*)** The number of PIC cycles inbetween two consecutive data dumps. Use a negative number to disable data dumping.
- **`warpx.dump_plotfiles` (*0* or *1*) optional** Whether to dump the simulation data in AMReX plotfile format. This is `1` by default, unless WarpX is compiled with openPMD support.
- **`warpx.dump_openpmd` (*0* or *1*) optional** Whether to dump the simulation data in openPMD format. When WarpX is compiled with openPMD support, this is `1` by default.
- **`warpx.openpmd_backend` (`h5`, `bp` or `json`) optional** I/O backend for openPMD dumps. When WarpX is compiled with openPMD support, this is `h5` by default. `json` only works with serial/single-rank jobs.
- **`warpx.do_boosted_frame_diagnostic` (*0* or *1*)** Whether to use the **back-transformed diagnostics** (i.e. diagnostics that perform on-the-fly conversion to the laboratory frame, when running boosted-frame simulations)
- **`warpx.lab_data_directory` (*string*)** The directory in which to save the lab frame data when using the **back-transformed diagnostics**. If not specified, the default is is *lab_frame_data*.
- **`warpx.num_snapshots_lab` (*integer*)** Only used when `warpx.do_boosted_frame_diagnostic` is `1`. The number of lab-frame snapshots that will be written.
- **`warpx.dt_snapshots_lab` (*float*, in seconds)** Only used when `warpx.do_boosted_frame_diagnostic` is `1`. The time interval inbetween the lab-frame snapshots (where this time interval is expressed in the laboratory frame).
- **`warpx.dz_snapshots_lab` (*float*, in meters)** Only used when `warpx.do_boosted_frame_diagnostic` is `1`. Distance between the lab-frame snapshots (expressed in the laboratory frame). `dt_snapshots_lab` is then computed by `dt_snapshots_lab = dz_snapshots_lab/c`. Either *dt_snapshots_lab* or *dz_snapshot_lab* is required.
- **`warpx.do_boosted_frame_fields` (*0 or 1*)** Whether to use the **back-transformed diagnostics** for the fields.
- **`warpx.boosted_frame_diag_fields` (space-separated list of *string*)** Which fields to dumped in back-transformed diagnostics. Choices are 'Ex', 'Ey', Ez', 'Bx', 'By', Bz', 'jx', 'jy', jz' and 'rho'. Example: `warpx.boosted_frame_diag_fields = Ex Ez By`. By default, all fields are dumped.
- **`warpx.plot_raw_fields` (*0* or *1*) optional (default *0*)** By default, the fields written in the plot files are averaged on the nodes. When ``warpx.plot_raw_fields` is *1*, then the raw (i.e. unaveraged) fields are also saved in the plot files.
- **`warpx.plot_raw_fields_guards` (*0* or *1*)** Only used when `warpx.plot_raw_fields` is `1`. Whether to include the guard cells in the output of the raw fields.
- **`warpx.plot_finepatch` (*0* or *1*)** Only used when mesh refinement is activated and `warpx.plot_raw_fields` is `1`. Whether to output the data of the fine patch, in the plot files.
- **`warpx.plot_crsepatch` (*0* or *1*)** Only used when mesh refinement is activated and `warpx.plot_raw_fields` is `1`. Whether to output the data of the coarse patch, in the plot files.

- **`warpx.plot_coarsening_ratio`** (*int* **; default:** *1*) Reduce size of the field output by this ratio in each dimension. (This is done by averaging the field.) `plot_coarsening_ratio` should be an integer divisor of `blocking_factor`.
- **`amr.plot_file`** (*string*) Root for output file names. Supports sub-directories. Default *diags/plotfiles/plt*
- **`warpx.fields_to_plot`** (*list of strings*) Fields written to plotfiles. Possible values: `Ex` `Ey` `Ez` `Bx` `By` `Bz` `jx` `jy` `jz` `part_per_cell` `rho` `F` `part_per_grid` `part_per_proc` `divE` `divB`. Default is `warpx.fields_to_plot = Ex Ey Ez Bx By Bz jx jy jz part_per_cell`.
- **`slice.dom_lo` and `slice.dom_hi`** (*2 floats in 2D*, *3 floats in 3D*; **in meters similar to the units of the simulation box.)** The extent of the slice are defined by the co-ordinates of the lower corner (`slice.dom_lo`) and upper corner (`slice.dom_hi`). The slice could be 1D, 2D, or 3D, aligned with the co-ordinate axes and the first axis of the coordinates is x. For example: if for a 3D simulation, an x-z slice is to be extracted at y = 0.0, then the y-value of slice.dom_lo and slice.dom_hi must be equal to 0.0
- **`slice.coarsening_ratio`** (*2 integers in 2D*, *3 integers in 3D*; **default** *1*) The coarsening ratio input must be greater than 0. Default is 1 in all directions. In the directions that is reduced, i.e., for an x-z slice in 3D, the reduced y-dimension has a default coarsening ratio equal to 1.
- **`slice.plot_int`** (*integer*) The number of PIC cycles inbetween two consecutive data dumps for the slice. Use a negative number to disable slice generation and slice data dumping.

2.3.10 Checkpoints and restart

WarpX supports checkpoints/restart via AMReX.

- **`amr.check_int`** (*integer*) The number of iterations between two consecutive checkpoints. Use a negative number to disable checkpoints.
- **`amr.restart`** (*string*) Name of the checkpoint file to restart from. Returns an error if the folder does not exist or if it is not properly formatted.

2.4 Profiling the code

2.4.1 Profiling with AMREX's built-in profiling tools

See this page in the AMReX documentation.

2.4.2 Profiling the code with Intel Advisor on NERSC

Follow these steps:

- Instrument the code during compilation

  ```
  module swap craype-haswell craype-mic-knl
  make -j 16 COMP=intel USE_VTUNE=TRUE
  ```

  (where the first line is only needed for KNL)
- In your SLURM submission script, use the following lines in order to run the executable. (In addition to setting the usual `OMP` environment variables.)

```
module load advisor
export ADVIXE_EXPERIMENTAL=roofline
srun -n <n_mpi> -c <n_logical_cores_per_mpi> --cpu_bind=cores advixe-cl -collect␣
↪survey -project-dir advisor -trace-mpi -- <warpx_executable> inputs
srun -n <n_mpi> -c <n_logical_cores_per_mpi> --cpu_bind=cores advixe-cl -collect␣
↪tripcounts -flop -project-dir advisor -trace-mpi -- <warpx_executable> inputs
```

where `<n_mpi>` and `<n_logical_cores_per_mpi>` should be replaced by the proper values, and `<warpx_executable>` should be replaced by the name of the WarpX executable.

- Launch the Intel Advisor GUI

```
module load advisor
advixe-gui
```

(Note: this requires to use `ssh -XY` when connecting to Cori.)

2.5 Parallelization in WarpX

When running a simulation, the domain is split into independent rectangular sub-domains (called **grids**). This is the way AMReX, a core component of WarpX, handles parallelization and/or mesh refinement. Furthermore, this decomposition makes load balancing possible: each MPI rank typically computes a few grids, and a rank with a lot of work can transfer one or several **grids** to their neighbors.

A user does not specify this decomposition explicitly. Instead, the user gives hints to the code, and the actual decomposition is determined at runtime, depending on the parallelization. The main user-defined parameters are `amr.max_grid_size` and `amr.blocking_factor`.

2.5.1 AMReX `max_grid_size` and `blocking_factor`

- `amr.max_grid_size` is the maximum number of points per **grid** along each direction (default `amr.max_grid_size=32` in 3D).
- `amr.blocking_factor`: The size of each **grid** must be divisible by the *blocking_factor* along all dimensions (default `amr.blocking_factor=8`). Note that the `max_grid_size` also has to be divisible by `blocking_factor`.

These parameters can have a dramatic impact on the code performance. Each **grid** in the decomposition is surrounded by guard cells, thus increasing the amount of data, computation and communication. Hence having a too small `max_grid_size`, may ruin the code performance.

On the other hand, a too-large `max_grid_size` is likely to result in a single grid per MPI rank, thus preventing load balancing. By setting these two parameters, the user wants to give some flexibility to the code while avoiding pathological behaviors.

For more information on this decomposition, see the Gridding and Load Balancing page on AMReX documentation.

For specific information on the dynamic load balancer used in WarpX, visit the Load Balancing page on the AMReX documentation.

The best values for these parameters strongly depends on a number of parameters, among which numerical parameters:

- Algorithms used (Maxwell/spectral field solver, filters, order of the particle shape factor)
- Number of guard cells (that depends on the particle shape factor and the type and order of the Maxwell solver, the filters used, *etc.*)

- Number of particles per cell, and the number of species

and MPI decomposition and computer architecture used for the run:

- GPU or CPU
- Number of OpenMP threads
- Amount of high-bandwidth memory.

Because these parameters put additional contraints on the domain size for a simulation, it can be cumbersome to calculate the number of cells and the physical size of the computational domain for a given resolution. This `Python script` does it automatically.

2.6 Running on specific platforms

2.6.1 Running on Cori KNL at NERSC

The batch script below can be used to run a WarpX simulation on 2 KNL nodes on the supercomputer Cori at NERSC. Replace descriptions between chevrons <> by relevant values, for instance `<job name>` could be `laserWakefield`.

```
#!/bin/bash -l

#SBATCH -N 2
#SBATCH -t 01:00:00
#SBATCH -q regular
#SBATCH -C knl
#SBATCH -S 4
#SBATCH -J <job name>
#SBATCH -A <allocation ID>
#SBATCH -e error.txt
#SBATCH -o output.txt

export OMP_PLACES=threads
export OMP_PROC_BIND=spread

# KNLs have 4 hyperthreads max
export CORI_MAX_HYPETHREAD_LEVEL=4
# We use 64 cores out of the 68 available on Cori KNL,
# and leave 4 to the system (see "#SBATCH -S 4" above).
export CORI_NCORES_PER_NODE=64

# Typically use 8 MPI ranks per node without hyperthreading,
# i.e., OMP_NUM_THREADS=8
export WARPX_NMPI_PER_NODE=8
export WARPX_HYPERTHREAD_LEVEL=1

# Compute OMP_NUM_THREADS and the thread count (-c option)
export CORI_NHYPERTHREADS_MAX=$(( ${CORI_MAX_HYPETHREAD_LEVEL} * ${CORI_NCORES_PER_
↪NODE} ))
export WARPX_NTHREADS_PER_NODE=$(( ${WARPX_HYPERTHREAD_LEVEL} * ${CORI_NCORES_PER_
↪NODE} ))
export OMP_NUM_THREADS=$(( ${WARPX_NTHREADS_PER_NODE} / ${WARPX_NMPI_PER_NODE} ))
export WARPX_THREAD_COUNT=$(( ${CORI_NHYPERTHREADS_MAX} / ${WARPX_NMPI_PER_NODE} ))

srun --cpu_bind=cores -n $(( ${SLURM_JOB_NUM_NODES} * ${WARPX_NMPI_PER_NODE} )) -c $
↪{WARPX_THREAD_COUNT} <path/to/executable> <input file>
```

(continues on next page)

(continued from previous page)

```
```

To run a simulation, copy the lines above to a file `batch_cori.sh` and run

```
sbatch batch_cori.sh
```

to submit the job.

For a 3D simulation with a few (1-4) particles per cell using FDTD Maxwell solver on Cori KNL for a well load-balanced problem (in our case laser wakefield acceleration simulation in a boosted frame in the quasi-linear regime), the following set of parameters provided good performance:

- `amr.max_grid_size=64` and `amr.blocking_factor=64` so that the size of each grid is fixed to `64**3` (we are not using load-balancing here).
- **8 MPI ranks per KNL node**, with `OMP_NUM_THREADS=8` (that is 64 threads per KNL node, i.e. 1 thread per physical core, and 4 cores left to the system).
- **2 grids per MPI**, *i.e.*, 16 grids per KNL node.

2.6.2 Running on Summit at OLCF

The batch script below can be used to run a WarpX simulation on 2 nodes on the supercomputer Summit at OLCF. Replace descriptions between chevrons <> by relevalt values, for instance `<input file>` could be `plasma_mirror_inputs`. Note that the only option so far is to run with one MPI rank per GPU.

```
#!/bin/bash
#BSUB -P <allocation ID>
#BSUB -W 00:10
#BSUB -nnodes 2
#BSUB -J WarpX
#BSUB -o WarpXo.%J
#BSUB -e WarpXe.%J

module load pgi
module load cuda

omp=1
export OMP_NUM_THREADS=${omp}

num_nodes=$(( $(printf '%s\n' ${LSB_HOSTS} | sort -u | wc -l) - 1 ))
jsrun -n ${num_nodes} -a 6 -g 6 -c 6 --bind=packed:${omp} <path/to/executable> <input␣
↪file> > output.txt
```

To run a simulation, copy the lines above to a file `batch_summit.sh` and run

```
bsub batch_summit.sh
```

to submit the job.

For a 3D simulation with a few (1-4) particles per cell using FDTD Maxwell solver on Summit for a well load-balanced problem (in our case laser wakefield acceleration simulation in a boosted frame in the quasi-linear regime), the following set of parameters provided good performance:

- `amr.max_grid_size=256` and `amr.blocking_factor=128`.
- **One MPI rank per GPU** (e.g., 6 MPI ranks for the 6 GPUs on each Summit node)
- **Two '128x128x128' grids per GPU**, or **one '128x128x256' grid per GPU**.

A batch script with more options regarding profiling on Summit can be found at `Summit batch script`

CHAPTER 3

Running WarpX from Python

3.1 How to run a new simulation

After installing WarpX as a Python package, you can use its functionalities in a Python script to run a simulation.

In order to run a new simulation:

- Create a **new directory**, where the simulation will be run.
- Add a **Python script** in the directory.

This file contains the numerical and physical parameters that define the situation to be simulated. Example input files can be found in the section *Example input files*.

- **Run** the script with Python:

```
mpirun -np <n_ranks> python <python_script>
```

where `<n_ranks>` is the number of MPI ranks used, and `<python_script>` is the name of the script.

3.2 Example input files

This section allows you to **download Python scripts** that correspond to different physical situations.

3.2.1 Beam-driven acceleration

- `Without mesh refinement`
- `With mesh refinement`

3.2.2 Laser-driven acceleration

- `Without mesh refinement`

CHAPTER 4

Visualizing the simulation results

WarpX can write data either in *plotfile* format (AMReX's native format), or in openPMD format (a common data format for Particle-In-Cell codes).

Note: This is controlled by the parameters `warpx.dump_plotfiles` and `warpx.dump_openpmd` & `warpx.openpmd_backend` in the section *Input parameters*.

This section describes some of the tools available to visualize the data:

4.1 Visualization with yt (for plotfiles)

yt is a Python package that can help in analyzing and visualizing WarpX data (among other data formats). It is convenient to use yt within a Jupyter notebook.

4.1.1 Installation

From the terminal:

```
pip install yt jupyter
```

or with the Anaconda distribution of python (recommended):

```
conda install -c conda-forge yt
```

The latest version of *yt* can be required for advanced options (e.g., rigid injection for particles). To built *yt* directly from source, you can use

```
pip install git+https://github.com/yt-project/yt.git
```

4.1.2 Visualizing the data

Once data ("plotfiles") has been created by the simulation, open a Jupyter notebook from the terminal:

```
jupyter notebook
```

Then use the following commands in the first cell of the notebook to import yt and load the first plot file:

```
import yt
ds = yt.load('./diags/plotfiles/plt00000/')
```

The list of field data and particle data stored can be seen with:

```
ds.field_list
```

For a quick start-up, the most useful commands for post-processing can be found in our Jupyter notebook `Visualization.ipynb`

Field data

Field data can be visualized using `yt.SlicePlot` (see the docstring of this function here)

For instance, in order to plot the field `Ex` in a slice orthogonal to `y (1)`:

```
yt.SlicePlot( ds, 1, 'Ex', origin='native' )
```

Note: *yt.SlicePlot* creates a 2D plot with the same aspect ratio as the physical size of the simulation box. Sometimes this can lead to very elongated plots that are difficult to read. You can modify the aspect ratio with the *aspect* argument ; for instance:

```
yt.SlicePlot( ds, 1, 'Ex', aspect=1./10 )
```

Alternatively, the data can be obtained as a numpy array.

For instance, in order to obtain the field *jz* (on level 0) as a numpy array:

```
ad0 = ds.covering_grid(level=0, left_edge=ds.domain_left_edge, dims=ds.domain_
↪dimensions)
jz_array = ad0['jz'].to_ndarray()
```

Particle data

Particle data can be visualized using `yt.ParticlePhasePlot` (see the docstring here).

For instance, in order to plot the particles' `x` and `y` positions:

```
yt.ParticlePhasePlot( ds.all_data(), 'particle_position_x', 'particle_position_y',
↪'particle_weight')
```

Alternatively, the data can be obtained as a numpy array.

For instance, in order to obtain the array of position *x* as a numpy array:

```
ad = ds.all_data()
x = ad['particle_position_x'].to_ndarray()
```

4.1.3 Read back-transformed diagnotics

When running a simulation in a boosted frame, WarpX has the capability to back-transform the simulation results to the laboratory frame of reference, which is often useful to study the physics. A set of function can be found in the python file `read_raw_data.py`.

Alternatively, the main commands can be found in our example jupyter notebook for postprocessing `Visualization.ipynb`.

4.1.4 Further information

A lot more information can be obtained from the yt documentation, and the corresponding notebook tutorials here.

4.2 Visualization with Visit (for plotfiles)

Note: The openPMD format can also be visualized with Visit, but requires the installation of a specific plugin: see this link.

WarpX results can also be visualized by VisIt, an open source visualization and analysis software. VisIT can be downloaded and installed from https://wci.llnl.gov/simulation/computer-codes/visit.

Assuming that you ran a 2D simulation, here are instructions for making a simple plot from a given plotfile:

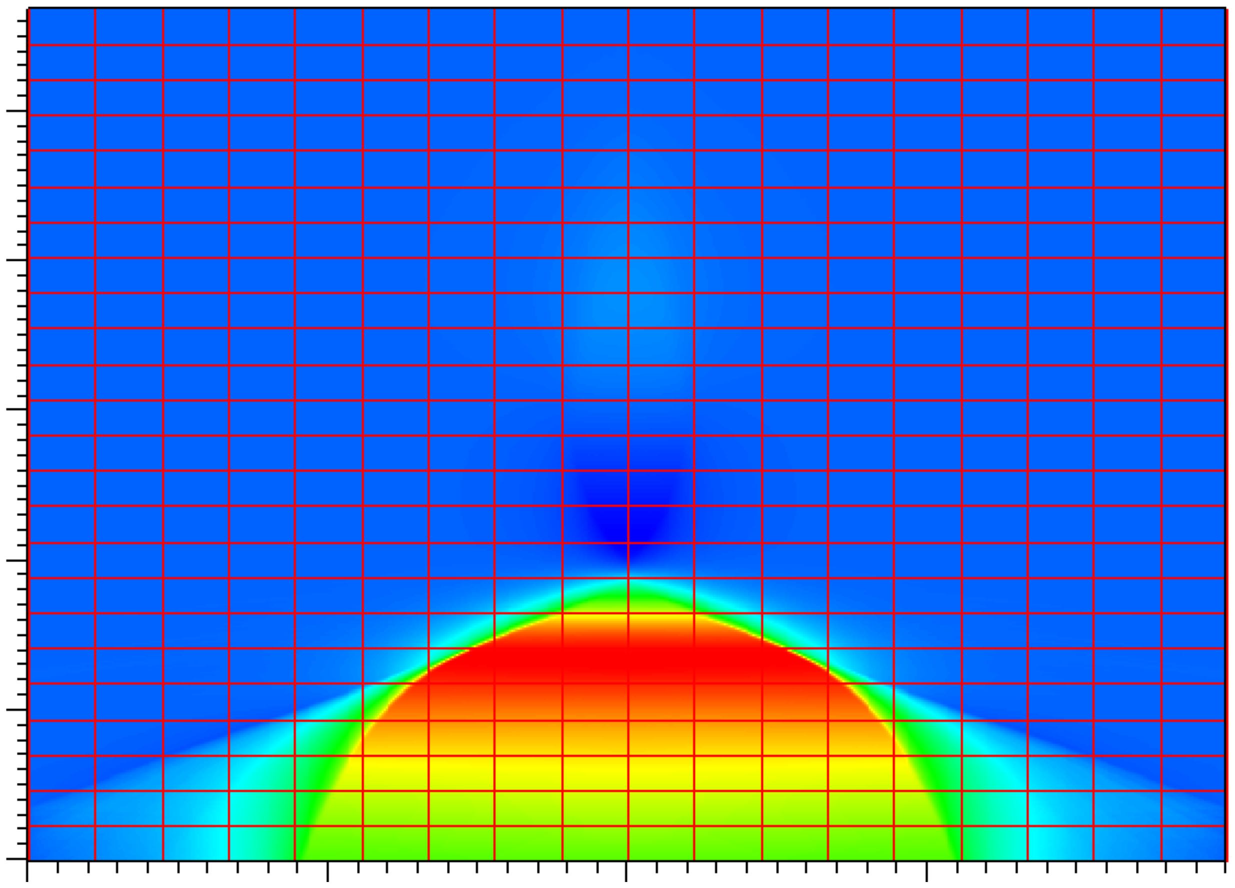
- Open the header file: Run VisIt, then select "File" -> "Open file ...", then select the Header file associated with the plotfile of interest (e.g., `plt10000/Header`).
- View the data: Select "Add" -> "Pseudocolor" -> "Ez" and select "Draw". You can select other variable to draw, such as `jx`, `jy`, `jz`, `Ex`, ...
- View the grid structure: Select "Subset" -> "levels". Then double clik the text "Subset-levels", enable the "Wireframe" option, select "Apply", select "Dismiss", and then select "Draw".
- Save the image: Select "File" -> "Set save options", then customize the image format to your liking, then click "Save".

Your image should look similar to the one below

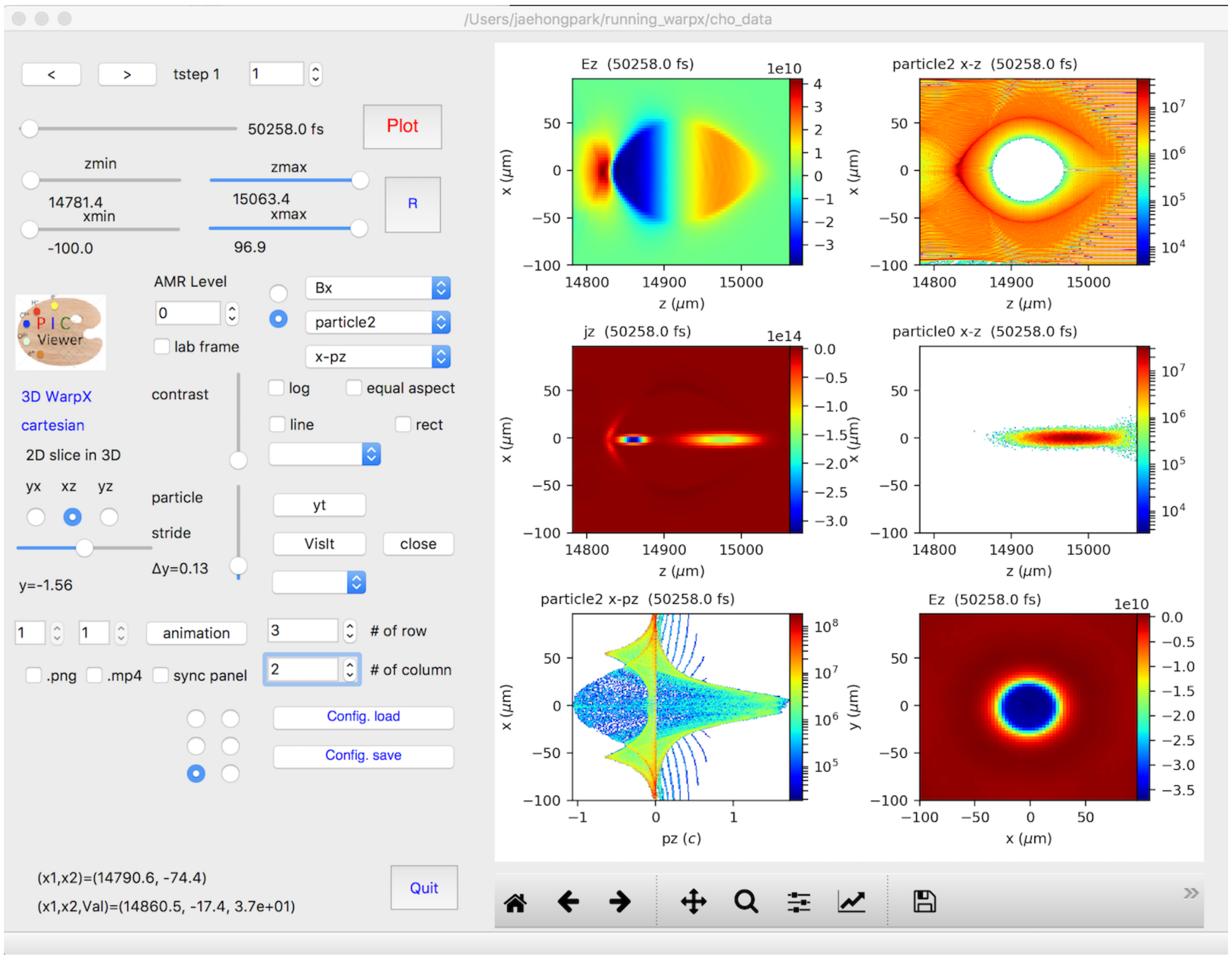
In 3D, you must apply the "Operators" -> "Slicing" -> "ThreeSlice", You can left-click and drag over the image to rotate the image to generate image you like.

To make a movie, you must first create a text file named `movie.visit` with a list of the Header files for the individual frames.

The next step is to run VisIt, select "File" -> "Open file ...", then select `movie.visit`. Create an image to your liking and press the "play" button on the VCR-like control panel to preview all the frames. To save the movie, choose "File" -> "Save movie ...", and follow the instructions on the screen.



4.3 PyQt-based visualization GUI: PICViewer (for both plotfiles and openPMD)



The toolkit provides various easy-to-use functions for data analysis of Warp/WarpX simulations.

4.3.1 Main features

- 2D/3D openPMD or WarpX data visualization,
- Multi-plot panels (up to 6 rows x 5 columns) which can be controlled independently or synchronously
- Interactive mouse functions (panel selection, image zoom-in, local data selection, etc)
- Animation from a single or multiple panel(s)
- Saving your job configuration and loading it later
- Interface to use VisIt, yt, or mayavi for 3D volume rendering (currently updating)

4.3.2 Required software

- python 2.7 or higher: http://docs.continuum.io/anaconda/install.

- PyQt5

  ```
  conda install pyqt
  ```

- h5py
- matplotlib
- numpy
- yt

  ```
  pip install git+https://github.com/yt-project/yt.git --user
  ```

- numba

4.3.3 Installation

```
pip install picviewer
```

You need to install yt and PySide separately.

You can install from the source for the latest update,

```
pip install git+https://bitbucket.org/ecp_warpx/picviewer/
```

4.3.4 To install manually

- Clone this repository

  ```
  git clone https://bitbucket.org/ecp_warpx/picviewer/
  ```

- Switch to the cloned directory with *cd picviewer* and type *python setup.py install*

4.3.5 To run

- You can start PICViewer from any directory. Type *picviewer* in the command line. Select a folder where your data files are located.
- You can directly open your data. Move on to a folder where your data files ae located (*cd [your data folder]*) and type *picviewer* in the command line.

4.4 Visualization with openPMD-viewer (for openPMD data)

openPMD-viewer is an open-source Python package to access openPMD data.

It allows to: - Quickly browse through the data, with a GUI-type interface in the Jupyter notebook - Have access to the data numpy array, for more detailed analysis

4.4.1 Installation

openPMD-viewer can be installed via `conda` or `pip`:

```
conda install -c rlehe openpmd_viewer
```

```
pip install openPMD-viewer
```

4.4.2 Usage

openPMD-viewer can be used either in simple Python scripts, or in a Jupyter notebook. In both cases, you can import openPMD-viewer, and load the data with the following commands:

```
from opmd_viewer import OpenPMDTimeSeries
ts = OpenPMDTimeSeries('./diags/hdf5')
```

Note: If you are using the Jupyter notebook, then you can start a pre-filled notebook, which already contains the above lines, by typing in a terminal:

```
openPMD_notebook
```

When using the Jupyter notebook, you can quickly browse through the data by using the command:

```
ts.slider()
```

You can also access the particle and field data as numpy arrays with the methods `ts.get_field` and `ts.get_particle`. See the openPMD-viewer tutorials here for more info.

4.5 Advanced yt visualization, for developers (for plotfiles)

This sections contains yt commands for advanced users. The Particle-In-Cell methods uses a staggered grid (see *The electromagnetic Particle-In-Cell method*), so that the x, y, and z components of the electric and magnetic fields are all defined at different locations in space. Regular output (see the *Visualization with yt (for plotfiles)* page, or the notebook at `WarpX/Tools/Visualization.ipynb` for an example) returns cell-centered data for convenience, which involves an additional operation. It is sometimes useful to access the raw data directly. Furthermore, the WarpX implementation for mesh refinement contains a number of grids for each level (coarse, fine and auxilary, see ../theory/warpx_theory for more details), and it is sometimes useful to access each of them (regular output return the auxiliary grid only). This page provides information to read raw data of all grids.

4.5.1 Dump additional data

In order to dump additional data in WarpX (mostly for debugging purpose), run the simulation with parameters

```
warpx.plot_raw_fields = 1
warpx.plot_finepatch = 1
warpx.plot_crsepatch = 1
warpx.plot_dive = 1
warpx.plot_rho = 1
```

see *Input parameters* for more information on these parameters.

4.5.2 Read raw data

Meta-data relevant to this topic (number and locations of grids in the simulation) are accessed to with

```
import yt
# get yt dataset
ds = yt.load( './plotfiles/plt00004' )
# Index of data in the plotfile
ds_index = ds.index
# Print the number of grids in the simulation
ds_index.grids.shape
# Left and right physical boundary of each grid
ds_index.grid_left_edge
ds_index.grid_right_edge
# List available fields
ds.field_list
```

When `warpx.plot_raw_fields=1` and `warpx.plot_finepatch=1`, here are some useful commands to access properties of a grid and the Ex field on the fine patch:

```
# store grid number 2 into my_grid
my_grid = ds.index.grids[2]
# Get left and right edges of my_grid
my_grid.LeftEdge
my_grid.RightEdge
# Get Level of my_grid
my_grid.Level
# left edge of the grid, in number of points
my_grid.start_index
```

Return the `Ex` field on the fine patch of grid `my_grid`:

```
my_field = my_grid['raw', 'Ex_fp'].squeeze().v
```

For a 2D plotfile, `my_field` has shape `(nx,nz,2)`. The last component stands for the two values on the edges of each cell for the electric field, due to field staggering. Numpy function `squeeze` removes empty components. While `yt` arrays are unit-aware, it is sometimes useful to extract the data into unitless numpy arrays. This is achieved with `.v`. In the case of `Ex_fp`, the staggering is on direction `x`, so that `my_field[:,:-1,1] == my_field[:,1:,0]`.

All combinations of the fields (`E` or `B`), the component (`x`, `y` or `z`) and the grid (`_fp` for fine, `_cp` for coarse and `_aux` for auxiliary) can be accessed in this way, i.e., `my_grid['raw', 'Ey_aux']` or `my_grid['raw', 'Bz_cp']` are valid queries.

4.6 Out-of-the-box plotting script

A ready-to-use python script for plotting simulation results is available at `plot_parallel.py`. Feel free to use it out-of-the-box or to modify it to suit your needs.

4.6.1 Dependencies

Most of its dependencies are standard Python packages, that come with a default Anaconda installation or can be installed with `pip` or `conda`: *os, matplotlib, sys, argparse, matplotlib, scipy*.

Additional dependencies are `yt >= 3.5` (or `yt >= 3.6` if you are using rigid injection, see section *Visualization with yt (for plotfiles)* on how to install `yt`), and `mpi4py`.

4.6.2 Run serial

Executing the script with

```
python plot_parallel.py
```

will loop through plotfiles named `plt?????` (e.g., `plt00000`, `plt00100` etc.) and save one image per plotfile. For a 2D simulation, a 2D colormap of the Ez field is plotted by default, with 1/20 of particles of each species (with different colors). For a 3D simulation, a 2D colormap of the central slices in *y* is plotted, and particles are handled the same way.

The script reads command-line options (which field and particle species, rendering with *yt* or *matplotlib*, etc.). For the full list of options, run

```
python plot_parallel.py --help
```

In particular, option `--plot_Ey_max_evolution` shows you how to plot the evolution of a scalar quantity over time (by default, the max of the Ey field). Feel free to modify it to plot the evolution of other quantities.

4.6.3 Run parallel

To execute the script in parallel, you can run for instance

```
mpirun -np 4 python plot_parallel.py --parallel
```

In this case, MPI ranks will share the plotfiles to process as evenly as possible. Note that each plotfile is still processed in serial. When option `--plot_Ey_max_evolution` is on, the scalar quantity is gathered to rank 0, and rank 0 plots the image.

If all dependencies are satisfied, the script can be used on Summit or Cori. For instance, the following batch script illustrates how to submit a post-processing batch job on Cori haswell with some options:

```
#!/bin/bash
#SBATCH --job-name=postproc
#SBATCH --time=00:20:00
#SBATCH -C haswell
#SBATCH -N 8
#SBATCH -q regular
#SBATCH -e postproce.txt
#SBATCH -o postproco.txt
#SBATCH --mail-type=end
#SBATCH --account=m2852

export OMP_NUM_THREADS=1

# Requires python3 and yt > 3.5
srun -n 32 -c 16 python plot_parallel.py --path <path/to/plotfiles> --plotlib=yt --
↪parallel
```

In addition, WarpX also has In-Situ Visualization capabilities (i.e. visualizing the data directly from the simulation, without dumping data files to disk).

4.7 In situ Visualization with SENSEI

SENSEI is a light weight framework for in situ data analysis. SENSEI's data model and API provide uniform access to and run time selection of a diverse set of visualization and analysis back ends including VisIt Libsim, ParaView Catalyst, VTK-m, Ascent, ADIOS, Yt, and Python.

SENSEI uses an XML file to select and configure one or more back ends at run time. Run time selection of the back end via XML means one user can access Catalyst, another Libsim, yet another Python with no changes to the code.

4.7.1 System Architecture

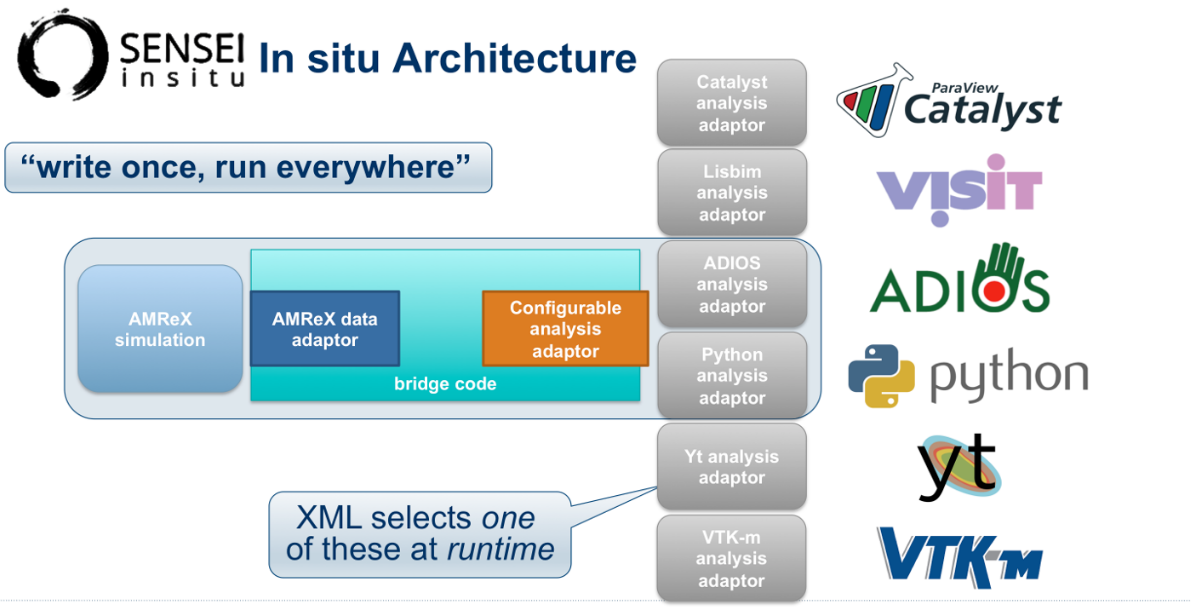


Fig. 4.1: SENSEI's in situ architecture enables use of a diverse of back ends which can be selected at run time via an XML configuration file

The three major architectural components in SENSEI are *data adaptors* which present simulation data in SENSEI's data model, *analysis adaptors* which present the back end data consumers to the simulation, and *bridge code* from which the simulation manages adaptors and periodically pushes data through the system. SENSEI comes equipped with a number of analysis adaptors enabling use of popular analysis and visualization libraries such as VisIt Libsim, ParaView Catalyst, Python, and ADIOS to name a few. AMReX contains SENSEI data adaptors and bridge code making it easy to use in AMReX based simulation codes.

SENSEI provides a *configurable analysis adaptor* which uses an XML file to select and configure one or more back ends at run time. Run time selection of the back end via XML means one user can access Catalyst, another Libsim, yet another Python with no changes to the code. This is depicted in figure Fig. 4.1. On the left side of the figure AMReX produces data, the bridge code pushes the data through the configurable analysis adaptor to the back end that was selected at run time.

4.7.2 Compiling with GNU Make

For codes making use of AMReX's build system add the following variable to the code's main `GNUmakefile`.

```
USE_SENSEI_INSITU = TRUE
```

When set, AMReX's make files will query environment variables for the lists of compiler and linker flags, include directories, and link libraries. These lists can be quite elaborate when using more sophisticated back ends, and are best set automatically using the `sensei_config` command line tool that should be installed with SENSEI. Prior to invoking make use the following command to set these variables:

```
source sensei_config
```

Typically, the `sensei_config` tool is in the users PATH after loading the desired SENSEI module. After configuring the build environment with `sensei_config`, proceed as usual.

```
make -j4 -f GNUmakefile
```

4.7.3 ParmParse Configuration

Once an AMReX code has been compiled with SENSEI features enabled, it will need to be enabled and configured at runtime. This is done using ParmParse input file. The supported parameters are described in the following table.

parameter	description	default
`insitu.int`	turns in situ processing on or off and controls how often data is processed.	0
`insitu.start`	controls when in situ processing starts.	0
`insitu.config`	points to the SENSEI XML file which selects and configures the desired back end.	
`insitu.pin_mesh`	when 1 lower left corner of the mesh is pinned to 0.,0.,0.	0

A typical use case is to enabled SENSEI by setting `insitu.int` to be greater than 1, and `insitu.config` to point SENSEI to an XML file that selects and configures the desired back end.

```
insitu.int = 2
insitu.config = render_iso_catalyst.xml
```

4.7.4 Back-end Selection and Configuration

The back end is selected and configured at run time using the SENSEI XML file. The XML sets parameters specific to SENSEI and to the chosen back end. Many of the back ends have sophisticated configuration mechanisms which SENSEI makes use of. For example the following XML configuration was used on NERSC's Cori with WarpX to render 10 iso surfaces, shown in figure Fig. 4.2, using VisIt Libsim.

```
<sensei>
  <analysis type="libsim" frequency="1" mode="batch"
    session="beam_j_pin.session"
    image-filename="beam_j_pin_%ts" image-width="1200" image-height="900"
    image-format="png" enabled="1"/>
</sensei>
```

The *session* attribute names a session file that contains VisIt specific runtime configuration. The session file is generated using VisIt GUI on a representative dataset. Usually this data set is generated in a low resolution run of the desired simulation.

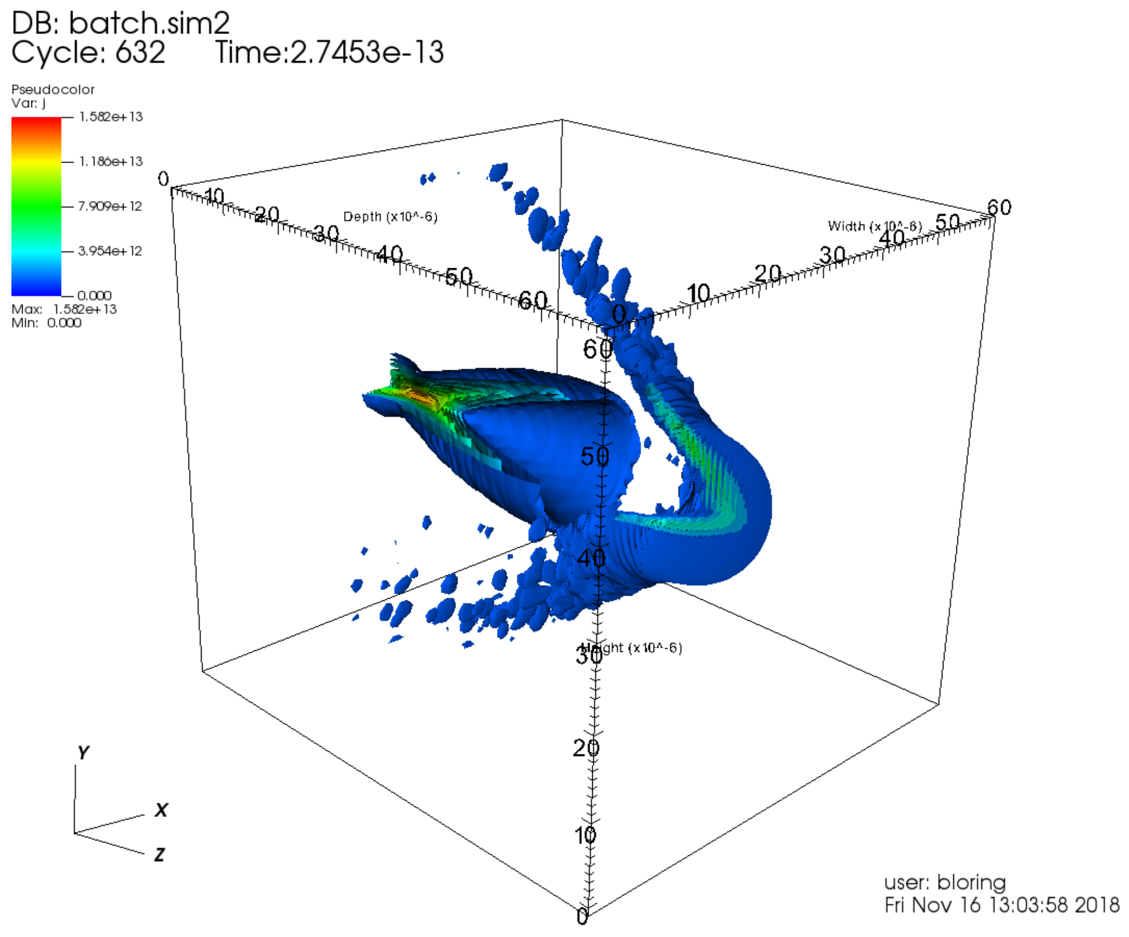


Fig. 4.2: Rendering of 10 3D iso-surfaces of j using VisIt libsim. The upper left quadrant has been clipped away to reveal innner structure.

The same run and visualization was repeated using ParaView Catalyst, shown in figure Fig. 4.3, by providing the following XML configuration.

```
<sensei>
  <analysis type="catalyst" pipeline="pythonscript"
    filename="beam_j.py" enabled="1" />
</sensei>
```

Here the *filename* attribute is used to pass Catalyst a Catalyst specific configuration that was generated using the ParaView GUI on a representative dataset.

The renderings in these runs were configured using a representative dataset which was obtained by running the simulation for a few time steps at a lower spatial resolution. When using VisIt Libsim the following XML configures the VTK writer to write the simulation data in VTK format. At the end of the run a `.visit` file that VisIt can open will be generated.

```
<sensei>
  <analysis type="PosthocIO" mode="visit" writer="xml"
```

(continues on next page)

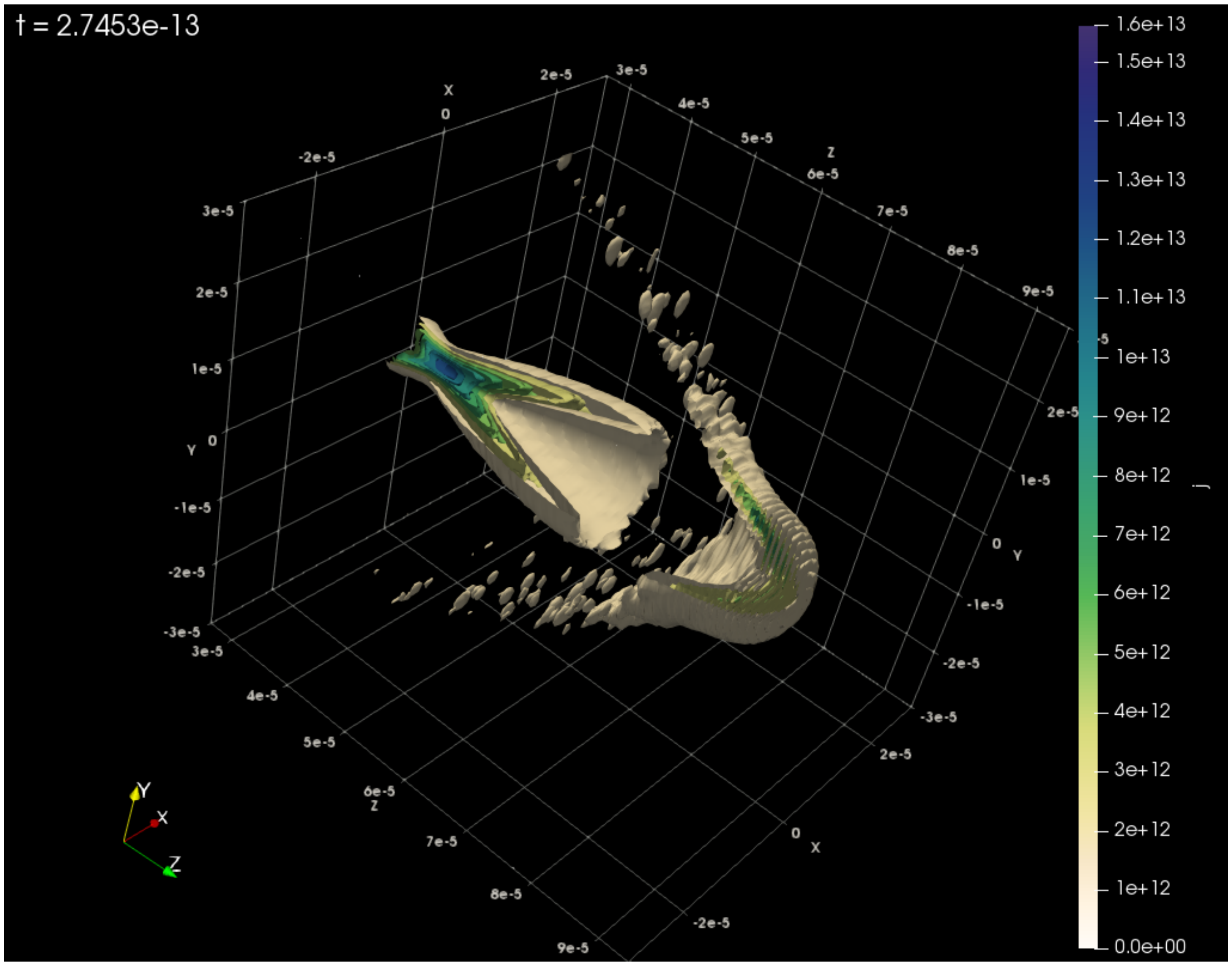


Fig. 4.3: Rendering of 10 3D iso-surfaces of j using ParaView Catalyst. The upper left quadrant has been clipped away to reveal innner structure.

(continued from previous page)

```
    ghost_array_name="avtGhostZones" output_dir="./"
    enabled="1">
  </analysis>
</sensei>
```

When using ParaView Catalyst the following XML configures the VTK writer to write the simulation data in VTK format. At the end of the run a `.pvd` file that ParaView can open will be generated.

```
<sensei>
  <analysis type="PosthocIO" mode="paraview" writer="xml"
    ghost_array_name="vtkGhostType" output_dir="./"
    enabled="1">
  </analysis>
</sensei>
```

4.7.5 Obtaining SENSEI

SENSEI is hosted on Kitware's Gitlab site at https://gitlab.kitware.com/sensei/sensei It's best to checkout the latest release rather than working on the master branch.

To ease the burden of wrangling back end installs SENSEI provides two platforms with all dependencies pre-installed, a VirtualBox VM, and a NERSC Cori deployment. New users are encouraged to experiment with one of these.

SENSEI VM

The SENSEI VM comes with all of SENSEI's dependencies and the major back ends such as VisIt and ParaView installed. The VM is the easiest way to test things out. It also can be used to see how installs were done and the environment configured.

The SENSEI VM can be downloaded here.

The SENSEI VM uses modules to manage the build and run environment. Load the SENSEI modulefile for the back-end you wish to use. The following table describes the available installs and which back-ends are supported in each.

modulefile	back-end(s)
sensei/2.1.1-catalyst-shared	ParaView Catalyst, ADIOS, Python
sensei/2.1.1-libsim-shared	VisIt Libsim, ADIOS, Python
sensei/2.1.1-vtk-shared	VTK-m, ADIOS, Python

NERSC Cori

SENSEI is deployed at NERSC on Cori. The NERSC deployment includes the major back ends such as ADIOS, ParaView Catalyst, VisIt Libsim, and Python.

The SENSEI installs uses modules to manage the build and run environment. Load the SENSEI modulefile for the back-end you wish to use. The following table describes the available installs and which back-ends are supported in each.

modulefile	back-end(s)
sensei/2.1.0-catalyst-shared	ParaView Catalyst, ADIOS, Python
sensei/2.1.0-libsim-shared	VisIt Libsim, ADIOS, Python
sensei/2.1.0-vtk-shared	VTK-m, ADIOS, Python

To access the SENSEI modulefiles on cori first add the SENSEI install to the search path:

```
module use /usr/common/software/sensei/modulefiles
```

4.7.6 3D LPA Example

This section shows an example of using SENSEI and three different back ends on a 3D LPA simulation. The instructions are specifically for NERSC cori, but also work with the SENSEI VM. The primary difference between working through the examples on cori or the VM are that different versions of software are installed.

Rendering with VisIt Libsim

First, log into cori and clone the git repo's.

```
cd $SCRATCH
mkdir warpx
cd warpx/
git clone --branch dev https://github.com/ECP-WarpX/WarpX.git WarpX-libsim
git clone --branch development https://github.com/AMReX-Codes/amrex
git clone --branch master https://bitbucket.org/berkeleylab/picsar.git
cd WarpX-libsim
vim GNUmakefile
```

Next, edit the makefile to turn the SENSEI features on.

```
USE_SENSEI_INSITU=TRUE
```

Then, load the SENSEI VisIt module, bring SENSEI's build requirements into the environment, and compile WarpX.

```
module use /usr/common/software/sensei/modulefiles/
module load sensei/2.1.0-libsim-shared
source sensei_config
make -j8
```

Download the WarpX input deck, SENSEI XML configuration and and VisIt session files. The inputs file configures WarpX, the xml file configures SENSEI, and the session file configures VisIt. The inputs and xml files are written by hand, while the session file is generated in VisIt gui on a representative data set.

```
wget https://data.kitware.com/api/v1/item/5c05d48e8d777f2179d22f20/download -O inputs.
↪3d
wget https://data.kitware.com/api/v1/item/5c05d4588d777f2179d22f16/download -O beam_j_
↪pin.xml
wget https://data.kitware.com/api/v1/item/5c05d4588d777f2179d22f0e/download -O beam_j_
↪pin.session
```

To run the demo, submit an interactive job to the batch queue, and launch WarpX.

```
salloc -C haswell -N 1 -t 00:30:00 -q debug
./Bin/main3d.gnu.TPROF.MPI.OMP.ex inputs.3d
```

Rendering with ParaView Catalyst

First, log into cori and clone the git repo's.

```
cd $SCRATCH
mkdir warpx
cd warpx/
git clone --branch dev https://github.com/ECP-WarpX/WarpX.git WarpX-catalyst
git clone --branch development https://github.com/AMReX-Codes/amrex
git clone --branch master https://bitbucket.org/berkeleylab/picsar.git
cd WarpX-catalyst
vim GNUmakefile
```

Next, edit the makefile to turn the SENSEI features on.

```
USE_SENSEI_INSITU=TRUE
```

Then, load the SENSEI ParaView module, bring SENSEI's build requirements into the environment, and compile WarpX.

```
module use /usr/common/software/sensei/modulefiles/
module load sensei/2.1.0-catalyst-shared
source sensei_config
make -j8
```

Download the WarpX input deck, SENSEI XML configuration and and ParaView session files. The inputs file configures WarpX, the xml file configures SENSEI, and the session file configures ParaView. The inputs and xml files are written by hand, while the session file is generated in ParaView gui on a representative data set.

```
wget https://data.kitware.com/api/v1/item/5c05b3fd8d777f2179d2067d/download -O inputs.
↪3d
wget https://data.kitware.com/api/v1/item/5c05b3fd8d777f2179d20675/download -O beam_j.
↪xml
wget https://data.kitware.com/api/v1/item/5c05b3fc8d777f2179d2066d/download -O beam_j.
↪py
```

To run the demo, submit an interactive job to the batch queue, and launch WarpX.

```
salloc -C haswell -N 1 -t 00:30:00 -q debug
./Bin/main3d.gnu.TPROF.MPI.OMP.ex inputs.3d
```

In situ Calculation with Python

SENSEI's Python back-end loads a user provided script file containing callbacks for `Initialize`, `Execute`, and `Finalize` phases of the run. During the execute phase the simulation pushes data through SENSEI. SENSEI forwards this data to the user provided Python function. SENSEI's MPI communicator is made available to the user's function via a global variable `comm`.

Here is a template for the user provided Python code.

```
# YOUR IMPORTS HERE

# SET DEFAULTS OF GLOBAL VARIABLES THAT INFLUENCE RUNTIME BEHAVIOR HERE

def Initialize():
  """ Initialization code """
  # YOUR CODE HERE
  return

def Execute(dataAdaptor):
  """ Use sensei::DataAdaptor instance passed in
      dataAdaptor to access and process simulation data """
  # YOUR CODE HERE
  return

def Finalize():
  """ Finalization code """
  # YOUR CODE HERE
  return
```

`Initialize` and `Finalize` are optional and will be called if they are provided. `Execute` is required. SENSEI's DataAdaptor API is used to obtain data and metadata from the simulation. Data is through VTK Object's. In WarpX the vtkOverlappingAMR VTK dataset is used.

The following script shows a simple integration of a scalar quantity over the valid cells of the mesh. The result is saved in a CSV format.

```
import numpy as np, matplotlib.pyplot as plt
from vtk.util.numpy_support import *
from vtk import vtkDataObject
import sys

# default values of control parameters
array = ''
out_file = ''

def Initialize():
  # rank zero writes the result
  if comm.Get_rank() == 0:
    fn = out_file if out_file else 'integrate_%s.csv'%(array)
    f = open(fn, 'w')
    f.write('# time, %s\n'%(array))
    f.close()
  return

def Execute(adaptor):
  # get the mesh and arrays we need
  dobj = adaptor.GetMesh('mesh', False)
  adaptor.AddArray(dobj, 'mesh', vtkDataObject.CELL, array)
  adaptor.AddGhostCellsArray(dobj, 'mesh')
  time = adaptor.GetDataTime()

  # integrate over the local blocks
  varint = 0.
  it = dobj.NewIterator()
  while not it.IsDoneWithTraversal():
    # get the local data block and its props
```

(continues on next page)

(continued from previous page)

```
    blk = it.GetCurrentDataObject()

    # get the array container
    atts = blk.GetCellData()

    # get the data array
    var =  vtk_to_numpy(atts.GetArray(array))

    # get ghost cell mask
    ghost = vtk_to_numpy(atts.GetArray('vtkGhostType'))
    ii = np.where(ghost == 0)[0]

    # integrate over valid cells
    varint = np.sum(var[ii])*np.prod(blk.GetSpacing())

    it.GoToNextItem()

  # reduce integral to rank 0
  varint = comm.reduce(varint, root=0, op=MPI.SUM)

  # rank zero writes the result
  if comm.Get_rank() == 0:
    fn = out_file if out_file else 'integrate_%s.csv'%(array)
    f = open(fn, 'a+')
    f.write('%s, %s\n'%(time, varint))
    f.close()
  return
```

The following XML configures SENSEI's Python back-end.

```
<sensei>
  <analysis type="python" script_file="./integrate.py" enabled="1">
    <initialize_source>
array='rho'
out_file='rho.csv'
     </initialize_source>
  </analysis>
</sensei>
```

The `script_file` attribute sets the file path to load the user's Python code from, and the `initialize_source` element contains Python code that controls runtime behavior specific to each user provided script.

4.8 In situ Visualization with Ascent

Ascent is a system designed to meet the in-situ visualization and analysis needs of simulation code teams running multi-physics calculations on many-core HPC architectures. It provides rendering runtimes that can leverage many-core CPUs and GPUs to render images of simulation meshes.

4.8.1 Compiling with GNU Make

After building and installing Ascent according to the instructions at Building Ascent, you can enable it in WarpX by changing the line

```
USE_ASCENT_INSITU = FALSE
```

in GNUmakefile to

```
USE_ASCENT_INSITU = TRUE
```

Furthermore, you must ensure that either the `ASCENT_HOME` shell environment variable contains the directory where Ascent is installed or you must specify this location when invoking make, i.e.,

```
make -j 8 ASCENT_HOME = /path/to/ascent/install
```

4.8.2 ParmParse Configuration

Once an AMReX code has been compiled with Ascent enabled, it will need to be enabled and configured at runtime. This is done using ParmParse input file. The supported parameters are described in the following table.

parameter	description	default
`insitu.int`	turns in situ processing on or off and controls how often data is processed.	0
`insitu.start`	controls when in situ processing starts.	0

A typical use case is setting `insitu.int` to a value of one or greater and `insitu.start` to the first time step where in situ analyswhere in situ analysis should be performed.

4.8.3 Visualization/Analysis Pipeline Configuration

Ascent uses the file `ascent_actions.json` to configure analysis and visualization pipelines. For example, the following `ascent_actions.json` file extracts an isosurface of the field Ex for 15 levels and saves the resulting images to `levels_<nnnn>.png`. Ascent Actions provides an overview over all available analysis and visualization actions.

```
[
  {
    "action": "add_pipelines",
    "pipelines":
    {
      "p1":
      {
        "f1":
        {
          "type" : "contour",
          "params" :
          {
            "field" : "Ex",
            "levels": 15
          }
        }
      }
    }
  },
  {
    "action": "add_scenes",
    "scenes":
    {
```

(continues on next page)

(continued from previous page)

```
      "s1":
      {
        "image_prefix": "levels_%04d",
        "plots":
        {
        "p1":
          {
            "type": "pseudocolor",
            "pipeline": "p1",
            "field": "Ex"
          }
        }
      }
    }
  },

  {
    "action": "execute"
  },

  {
    "action": "reset"
  }
]
```

If you like the 3D rendering of laser wakefield acceleration on the WarpX documentation frontpage (which is also the avatar of the ECP-WarpX organization), you can find the serial analysis script `video_yt.py` as well as a parallel analysis script `video_yt.py` used to make a similar rendering for a beam-driven wakefield simulation, running parallel.

CHAPTER 5

Theoretical background

This page contains information on the algorithms that are used in WarpX.

Topics:

5.1 Introduction

Computer simulations have had a profound impact on the design and understanding of past and present plasma acceleration experiments (Tsung et al. 2006; Geddes et al. 2008; C. Geddes et al. 2009; Huang et al. 2009), with accurate modeling of wake formation, electron self-trapping and acceleration requiring fully kinetic methods (usually Particle-In-Cell) using large computational resources due to the wide range of space and time scales involved. Numerical modeling complements and guides the design and analysis of advanced accelerators, and can reduce development costs significantly. Despite the major recent experimental successes(Leemans et al. 2014; Blumenfeld et al. 2007; Bulanov S V and Wilkens J J and Esirkepov T Zh and Korn G and Kraft G and Kraft S D and Molls M and Khoroshkov V S 2014; Steinke et al. 2016), the various advanced acceleration concepts need significant progress to fulfill their potential. To this end, large-scale simulations will continue to be a key component toward reaching a detailed understanding of the complex interrelated physics phenomena at play.

For such simulations, the most popular algorithm is the Particle-In-Cell (or PIC) technique, which represents electromagnetic fields on a grid and particles by a sample of macroparticles. However, these simulations are extremely computationally intensive, due to the need to resolve the evolution of a driver (laser or particle beam) and an accelerated beam into a structure that is orders of magnitude longer and wider than the accelerated beam. Various techniques or reduced models have been developed to allow multidimensional simulations at manageable computational costs: quasistatic approximation (Sprangle, Esarey, and Ting 1990; Antonsen and Mora 1992; Krall et al. 1993; Mora and Antonsen 1997; Huang et al. 2006), ponderomotive guiding center (PGC) models (Antonsen and Mora 1992; Krall et al. 1993; Huang et al. 2006; Benedetti et al. 2010; Cowan et al. 2011), simulation in an optimal Lorentz boosted frame (Vay 2007; Bruhwiler et al. 2009; Vay et al. 2009, 2010; Vay et al. 2009; Martins et al. 2009; Martins, Fonseca, Lu, et al. 2010; Martins, Fonseca, Vieira, et al. 2010; S. F. Martins et al. 2010; J L Vay et al. 2011; J. Vay et al. 2011; J -L. Vay et al. 2011; Yu et al. 2016), expanding the fields into a truncated series of azimuthal modes (Godfrey 1985; Lifschitz et al. 2009; Davidson et al. 2015; Lehe et al. 2016; Andriyash, Lehe, and Lifschitz 2016), fluid approximation (Krall et al. 1993; Shadwick, Schroeder, and Esarey 2009; Benedetti et al. 2010) and scaled parameters (Cormier-Michel et al. 2009; C. G. R. Geddes et al. 2009).

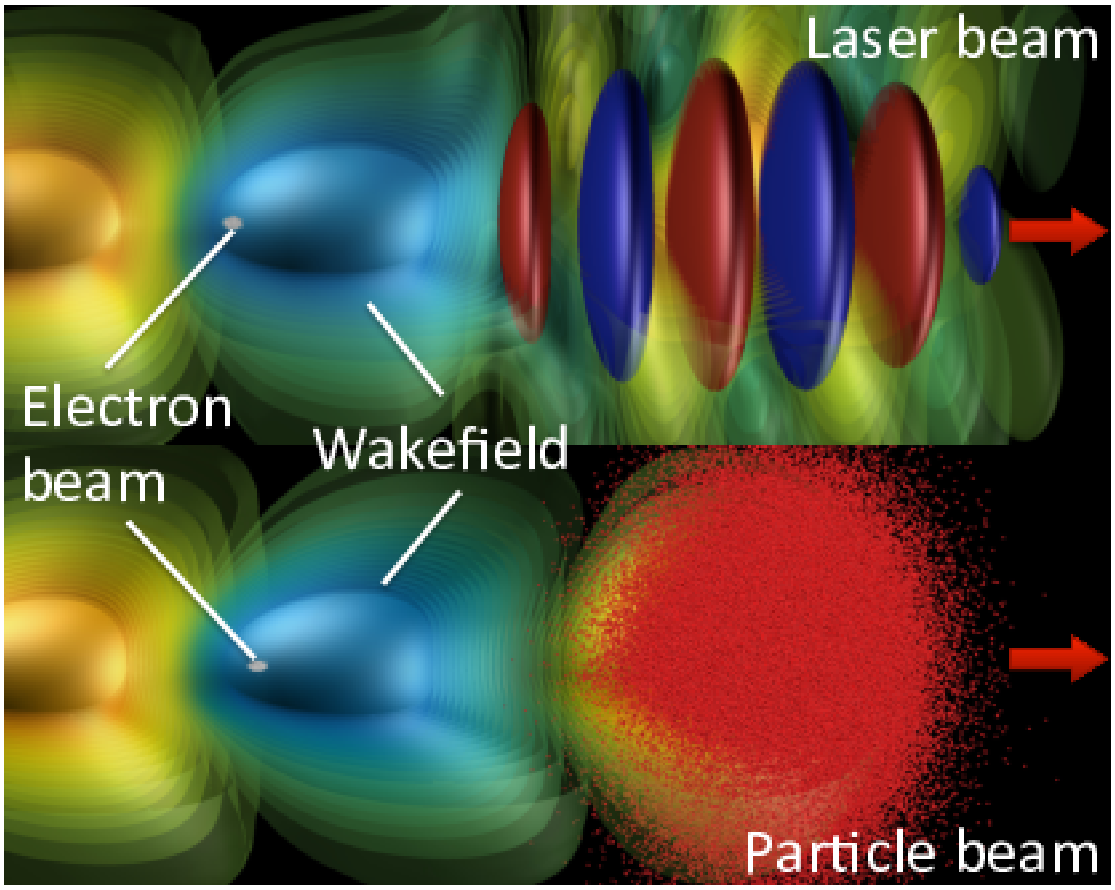


Fig. 5.1: [fig:Plasma_acceleration_sim] Plasma laser-driven (top) and charged-particles-driven (bottom) acceleration (rendering from 3-D Particle-In-Cell simulations). A laser beam (red and blue disks in top picture) or a charged particle beam (red dots in bottom picture) propagating (from left to right) through an under-dense plasma (not represented) displaces electrons, creating a plasma wakefield that supports very high electric fields (pale blue and yellow). These electric fields, which can be orders of magnitude larger than with conventional techniques, can be used to accelerate a short charged particle beam (white) to high-energy over a very short distance.

Andriyash, Igor A., Remi Lehe, and Agustin Lifschitz. 2016. “Laser-Plasma Interactions with a Fourier-Bessel Particle-in-Cell Method.” *Physics of Plasmas* 23 (3). https://doi.org/http://dx.doi.org/10.1063/1.4943281.

Antonsen, T M, and P Mora. 1992. “Self-Focusing and Raman-Scattering of Laser-Pulses in Tenuous Plasmas.” *Physical Review Letters* 69 (15): 2204–7. https://doi.org/10.1103/Physrevlett.69.2204.

Benedetti, C, C B Schroeder, E Esarey, C G R Geddes, and W P Leemans. 2010. “Efficient Modeling of Laser-Plasma Accelerators with Inf&Rno.” *Aip Conference Proceedings* 1299: 250–55. https://doi.org/10.1063/1.3520323.

Blumenfeld, Ian, Christopher E Clayton, Franz-Josef Decker, Mark J Hogan, Chengkun Huang, Rasmus Ischebeck, Richard Iverson, et al. 2007. “Energy doubling of 42[thinsp]GeV electrons in a metre-scale plasma wakefield accelerator.” *Nature* 445 (7129): 741–44. http://dx.doi.org/10.1038/nature05538.

Bruhwiler, D L, J R Cary, B M Cowan, K Paul, C G R Geddes, P J Mullowney, P Messmer, et al. 2009. “New Developments in the Simulation of Advanced Accelerator Concepts.” In *Aip Conference Proceedings*, 1086:29–37.

Bulanov S V and Wilkens J J and Esirkepov T Zh and Korn G and Kraft G and Kraft S D and Molls M and Khoroshkov V S. 2014. “Laser ion acceleration for hadron therapy.” *Physics-Uspekhi* 57 (12): 1149. http://stacks.iop.org/1063-7869/57/i=12/a=1149.

Cormier-Michel, E, C G R Geddes, E Esarey, C B Schroeder, D L Bruhwiler, K Paul, B Cowan, and W P Leemans. 2009. “Scaled Simulations of A 10 Gev Accelerator.” In *Aip Conference Proceedings*, 1086:297–302.

Cowan, Benjamin M, David L Bruhwiler, Estelle Cormier-Michel, Eric Esarey, Cameron G R Geddes, Peter Messmer, and Kevin M Paul. 2011. “Characteristics of an Envelope Model for Laser-Plasma Accelerator Simulation.” *Journal of Computational Physics* 230 (1): 61–86. https://doi.org/Doi: 10.1016/J.Jcp.2010.09.009.

Davidson, A., A. Tableman, W. An, F.S. Tsung, W. Lu, J. Vieira, R.A. Fonseca, L.O. Silva, and W.B. Mori. 2015. “Implementation of a hybrid particle code with a PIC description in r–z and a gridless description in into OSIRIS.” *Journal of Computational Physics* 281: 1063–77. https://doi.org/10.1016/j.jcp.2014.10.064.

Geddes, C G R, D L Bruhwiler, J R Cary, W B Mori, J.-L. Vay, S F Martins, T Katsouleas, et al. 2008. “Computational Studies and Optimization of Wakefield Accelerators.” In *Journal of Physics: Conference Series*, 125:012002 (11 Pp.).

Geddes et al., Cgr. 2009. “Laser Plasma Particle Accelerators: Large Fields for Smaller Facility Sources.” In *Scidac Review 13*, 13.

Geddes et al., C G R. 2009. “Scaled Simulation Design of High Quality Laser Wakefield Accelerator Stages.” In *Proc. Particle Accelerator Conference*. Vancouver, Canada.

Godfrey, B.B. 1985. *The Iprop Three-Dimensional Beam Propagation Code*. Defense Technical Information Center. https://books.google.com/books?id=hos_OAAACAAJ.

Huang, C, W An, V K Decyk, W Lu, W B Mori, F S Tsung, M Tzoufras, et al. 2009. “Recent Results and Future Challenges for Large Scale Particle-in-Cell Simulations of Plasma-Based Accelerator Concepts.” *Journal of Physics: Conference Series* 180 (1): 012005 (11 Pp.).

Huang, C, V K Decyk, C Ren, M Zhou, W Lu, W B Mori, J H Cooley, T M Antonsen Jr., and T Katsouleas. 2006. “Quickpic: A Highly Efficient Particle-in-Cell Code for Modeling Wakefield Acceleration in Plasmas.” *Journal of Computational Physics* 217 (2): 658–79. https://doi.org/10.1016/J.Jcp.2006.01.039.

Krall, J, A Ting, E Esarey, and P Sprangle. 1993. “Enhanced Acceleration in A Self-Modulated-Laser Wake-Field Accelerator.” *Physical Review E* 48 (3): 2157–61. https://doi.org/10.1103/Physreve.48.2157.

Leemans, W P, A J Gonsalves, H.-S. Mao, K Nakamura, C Benedetti, C B Schroeder, Cs. Tóth, et al. 2014. “Multi-GeV Electron Beams from Capillary-Discharge-Guided Subpetawatt Laser Pulses in the Self-Trapping Regime.” *Phys. Rev. Lett.* 113 (24). American Physical Society: 245002. https://doi.org/10.1103/PhysRevLett.113.245002.

Lehe, Rémi, Manuel Kirchen, Igor A. Andriyash, Brendan B. Godfrey, and Jean-Luc Vay. 2016. “A spectral, quasi-cylindrical and dispersion-free Particle-In-Cell algorithm.” *Computer Physics Communications* 203: 66–82. https://doi.org/10.1016/j.cpc.2016.02.007.

Lifschitz, A F, X Davoine, E Lefebvre, J Faure, C Rechatin, and V Malka. 2009. "Particle-in-Cell modelling of laser{â}plasma interaction using Fourier decomposition." *Journal of Computational Physics* 228 (5): 1803–14. https://doi.org/http://dx.doi.org/10.1016/j.jcp.2008.11.017.

Martins, Samuel F, Ricardo A Fonseca, Luis O Silva, Wei Lu, and Warren B Mori. 2010. "Numerical Simulations of Laser Wakefield Accelerators in Optimal Lorentz Frames." *Computer Physics Communications* 181 (5): 869–75. https://doi.org/10.1016/J.Cpc.2009.12.023.

Martins, S F, R A Fonseca, W Lu, W B Mori, and L O Silva. 2010. "Exploring Laser-Wakefield-Accelerator Regimes for Near-Term Lasers Using Particle-in-Cell Simulation in Lorentz-Boosted Frames." *Nature Physics* 6 (4): 311–16. https://doi.org/10.1038/Nphys1538.

Martins, S F, R A Fonseca, J Vieira, L O Silva, W Lu, and W B Mori. 2010. "Modeling Laser Wakefield Accelerator Experiments with Ultrafast Particle-in-Cell Simulations in Boosted Frames." *Physics of Plasmas* 17 (5): 56705. https://doi.org/10.1063/1.3358139.

Martins et al., S F. 2009. "Boosted Frame Pic Simulations of Lwfa: Towards the Energy Frontier." In *Proc. Particle Accelerator Conference*. Vancouver, Canada.

Mora, P, and Tm Antonsen. 1997. "Kinetic Modeling of Intense, Short Laser Pulses Propagating in Tenuous Plasmas." *Phys. Plasmas* 4 (1): 217–29. https://doi.org/10.1063/1.872134.

Shadwick, B A, C B Schroeder, and E Esarey. 2009. "Nonlinear Laser Energy Depletion in Laser-Plasma Accelerators." *Physics of Plasmas* 16 (5): 56704. https://doi.org/10.1063/1.3124185.

Sprangle, P, E Esarey, and A Ting. 1990. "Nonlinear-Theory of Intense Laser-Plasma Interactions." *Physical Review Letters* 64 (17): 2011–4.

Steinke, S, J van Tilborg, C Benedetti, C G R Geddes, C B Schroeder, J Daniels, K K Swanson, et al. 2016. "Multistage coupling of independent laser-plasma accelerators." *Nature* 530 (7589). Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.: 190–93. http://dx.doi.org/10.1038/nature16525 http://10.1038/nature16525.

Tsung, Fs, W Lu, M Tzoufras, Wb Mori, C Joshi, Jm Vieira, Lo Silva, and Ra Fonseca. 2006. "Simulation of Monoenergetic Electron Generation via Laser Wakefield Accelerators for 5-25 Tw Lasers." *Physics of Plasmas* 13 (5): 56708. https://doi.org/10.1063/1.2198535.

Vay, J.-L. 2007. "Noninvariance of Space- and Time-Scale Ranges Under A Lorentz Transformation and the Implications for the Study of Relativistic Interactions." *Physical Review Letters* 98 (13): 130405/1–4.

Vay, J.-L., D L Bruhwiler, C G R Geddes, W M Fawley, S F Martins, J R Cary, E Cormier-Michel, et al. 2009. "Simulating Relativistic Beam and Plasma Systems Using an Optimal Boosted Frame." *Journal of Physics: Conference Series* 180 (1): 012006 (5 Pp.).

Vay, J -. L, C G R Geddes, C Benedetti, D L Bruhwiler, E Cormier-Michel, B M Cowan, J R Cary, and D P Grote. 2010. "Modeling Laser Wakefield Accelerators in A Lorentz Boosted Frame." *Aip Conference Proceedings* 1299: 244–49. https://doi.org/10.1063/1.3520322.

Vay, J L, C G R Geddes, E Cormier-Michel, and D P Grote. 2011. "Numerical Methods for Instability Mitigation in the Modeling of Laser Wakefield Accelerators in A Lorentz-Boosted Frame." *Journal of Computational Physics* 230 (15): 5908–29. https://doi.org/10.1016/J.Jcp.2011.04.003.

Vay, Jl, C G R Geddes, E Cormier-Michel, and D P Grote. 2011. "Effects of Hyperbolic Rotation in Minkowski Space on the Modeling of Plasma Accelerators in A Lorentz Boosted Frame." *Physics of Plasmas* 18 (3): 30701. https://doi.org/10.1063/1.3559483.

Vay, J -L., C G R Geddes, E Esarey, C B Schroeder, W P Leemans, E Cormier-Michel, and D P Grote. 2011. "Modeling of 10 Gev-1 Tev Laser-Plasma Accelerators Using Lorentz Boosted Simulations." *Physics of Plasmas* 18 (12). https://doi.org/10.1063/1.3663841.

Vay et al., J.-L. 2009. "Application of the Reduction of Scale Range in A Lorentz Boosted Frame to the Numerical Simulation of Particle Acceleration Devices." In *Proc. Particle Accelerator Conference*. Vancouver, Canada.

Yu, Peicheng, Xinlu Xu, Asher Davidson, Adam Tableman, Thamine Dalichaouch, Fei Li, Michael D. Meyers, et al. 2016. “Enabling Lorentz boosted frame particle-in-cell simulations of laser wakefield acceleration in quasi-3D geometry.” *Journal of Computational Physics*. https://doi.org/10.1016/j.jcp.2016.04.014.

5.2 The electromagnetic Particle-In-Cell method

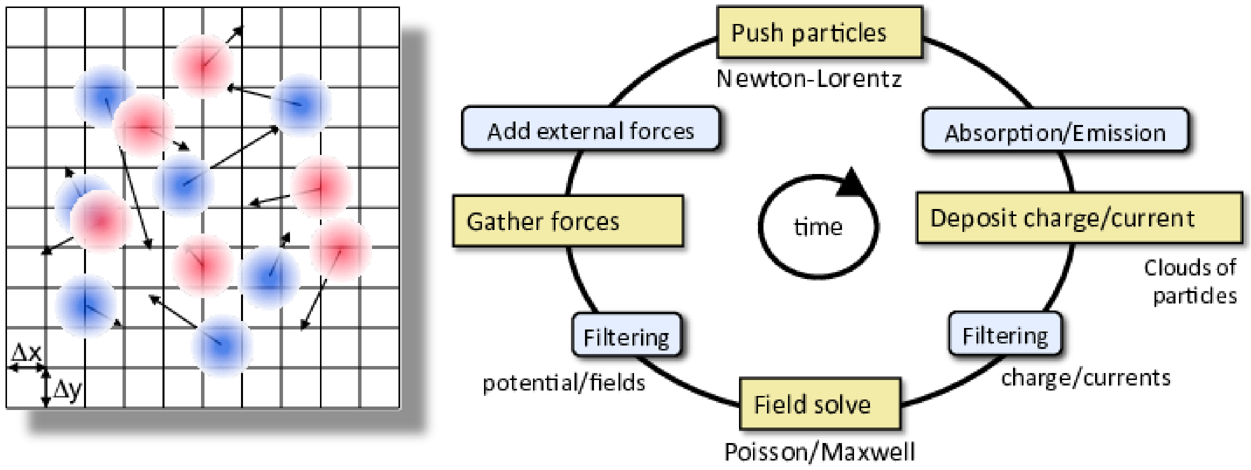


Fig. 5.2: [fig:PIC] The Particle-In-Cell (PIC) method follows the evolution of a collection of charged macro-particles (positively charged in blue on the left plot, negatively charged in red) that evolve self-consistently with their electromagnetic (or electrostatic) fields. The core PIC algorithm involves four operations at each time step: 1) evolve the velocity and position of the particles using the Newton-Lorentz equations, 2) deposit the charge and/or current densities through interpolation from the particles distributions onto the grid, 3) evolve Maxwell’s wave equations (for electromagnetic) or solve Poisson’s equation (for electrostatic) on the grid, 4) interpolate the fields from the grid onto the particles for the next particle push. Additional “add-ons” operations are inserted between these core operations to account for additional physics (e.g. absorption/emission of particles, addition of external forces to account for accelerator focusing or accelerating component) or numerical effects (e.g. smoothing/filtering of the charge/current densities and/or fields on the grid).

In the electromagnetic Particle-In-Cell method (Birdsall and Langdon 1991), the electromagnetic fields are solved on a grid, usually using Maxwell’s equations

$$\begin{aligned}\frac{\partial \mathbf{B}}{\partial t} &= -\nabla\times\mathbf{E}\\ \frac{\partial \mathbf{E}}{\partial t} &= \nabla\times\mathbf{B}-\mathbf{J}\\ \nabla\cdot\mathbf{E} &= \rho\\ \nabla\cdot\mathbf{B} &= 0\end{aligned}$$

given here in natural units ($\epsilon_0 = \mu_0 = c = 1$), where t is time, $\mathbf{E}$ and $\mathbf{B}$ are the electric and magnetic field components, and ρ and $\mathbf{J}$ are the charge and current densities. The charged particles are advanced in time using the Newton-Lorentz equations of motion

$$\begin{aligned}\frac{d\mathbf{x}}{dt} =& \mathbf{v},\\ \frac{d\left(\gamma\mathbf{v}\right)}{dt} =& \frac{q}{m}\left(\mathbf{E}+\mathbf{v}\times\mathbf{B}\right),\end{aligned}$$

where m, q, $\mathbf{x}$, $\mathbf{v}$ and $\gamma = 1/\sqrt{1-v^2}$ are respectively the mass, charge, position, velocity and relativistic factor of the particle given in natural units ($c = 1$). The charge and current densities are interpolated on the grid from the particles' positions and velocities, while the electric and magnetic field components are interpolated from the grid to the particles' positions for the velocity update.

5.2.1 Particle push

A centered finite-difference discretization of the Newton-Lorentz equations of motion is given by

$$\begin{aligned} \frac{\mathbf{x}^{i+1}-\mathbf{x}^{i}}{\Delta t} =& \mathbf{v}^{i+1/2}, \\ \frac{\gamma^{i+1/2}\mathbf{v}^{i+1/2}-\gamma^{i-1/2}\mathbf{v}^{i-1/2}}{\Delta t} =& \frac{q}{m}\left(\mathbf{E}^{i}+\mathbf{\bar{v}}^{i}\times\mathbf{B}^{i}\right). \end{aligned}$$

In order to close the system, $\mathbf{\bar{v}}^{i}$ must be expressed as a function of the other quantities. The two implementations that have become the most popular are presented below.

Boris relativistic velocity rotation

The solution proposed by Boris (Boris 1970) is given by

$$\mathbf{\bar{v}}^{i} = \frac{\gamma^{i+1/2}\mathbf{v}^{i+1/2}+\gamma^{i-1/2}\mathbf{v}^{i-1/2}}{2\bar{\gamma}^{i}}.$$

where $\bar{\gamma}^{i}$ is defined by $\bar{\gamma}^{i} \equiv (\gamma^{i+1/2}+\gamma^{i-1/2})/2$.

The system (*[Eq:leapfrog_v]*,*[Eq:boris_v]*) is solved very efficiently following Boris' method, where the electric field push is decoupled from the magnetic push. Setting $\mathbf{u}=\gamma\mathbf{v}$, the velocity is updated using the following sequence:

$$\begin{aligned} \mathbf{u^{-}} =& \mathbf{u}^{i-1/2}+\left(q\Delta t/2m\right)\mathbf{E}^{i} \\ \mathbf{u'} =& \mathbf{u}^{-}+\mathbf{u}^{-}\times\mathbf{t} \\ \mathbf{u}^{+} =& \mathbf{u}^{-}+\mathbf{u'}\times2\mathbf{t}/(1+t^{2}) \\ \mathbf{u}^{i+1/2} =& \mathbf{u}^{+}+\left(q\Delta t/2m\right)\mathbf{E}^{i} \end{aligned}$$

where $\mathbf{t}=\left(q\Delta t/2m\right)\mathbf{B}^{i}/\bar{\gamma}^{i}$ and where $\bar{\gamma}^{i}$ can be calculated as $\bar{\gamma}^{i}=\sqrt{1+(\mathbf{u}^{-}/c)^{2}}$.

The Boris implementation is second-order accurate, time-reversible and fast. Its implementation is very widespread and used in the vast majority of PIC codes.

Vay Lorentz-invariant formulation

It was shown in (Vay 2008) that the Boris formulation is not Lorentz invariant and can lead to significant errors in the treatment of relativistic dynamics. A Lorentz invariant formulation is obtained by considering the following velocity average

$$\mathbf{\bar{v}}^{i} = \frac{\mathbf{v}^{i+1/2}+\mathbf{v}^{i-1/2}}{2},$$

This gives a system that is solvable analytically (see (Vay 2008) for a detailed derivation), giving the following velocity update:

$$\begin{aligned} \mathbf{u^{*}} =& \mathbf{u}^{i-1/2}+\frac{q\Delta t}{m}\left(\mathbf{E}^{i}+\frac{\mathbf{v}^{i-1/2}}{2}\times\mathbf{B}^{i}\right), \\ \mathbf{u}^{i+1/2} =& \left[\mathbf{u^{*}}+\left(\mathbf{u^{*}}\cdot\mathbf{t}\right)\mathbf{t}+\mathbf{u^{*}}\times\mathbf{t}\right]/\left(1+t^{2}\right), \end{aligned}$$

where $\mathbf{t} = \boldsymbol{\tau}/\gamma^{i+1/2}$, $\boldsymbol{\tau} = (q\Delta t/2m)\,\mathbf{B}^i$, $\gamma^{i+1/2} = \sqrt{\sigma + \sqrt{\sigma^2 + (\tau^2 + w^2)}}$, $w = \mathbf{u}^* \cdot \boldsymbol{\tau}$, $\sigma = \left(\gamma'^2 - \tau^2\right)/2$ and $\gamma' = \sqrt{1 + (\mathbf{u}^*/c)^2}$. This Lorentz invariant formulation is particularly well suited for the modeling of ultra-relativistic charged particle beams, where the accurate account of the cancellation of the self-generated electric and magnetic fields is essential, as shown in (Vay 2008).

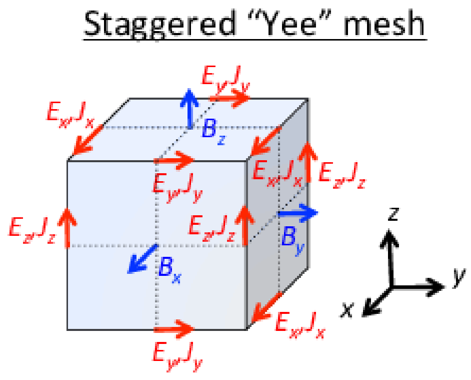
5.2.2 Field solve

Various methods are available for solving Maxwell's equations on a grid, based on finite-differences, finite-volume, finite-element, spectral, or other discretization techniques that apply most commonly on single structured or unstructured meshes and less commonly on multiblock multiresolution grid structures. In this chapter, we summarize the widespread second order finite-difference time-domain (FDTD) algorithm, its extension to non-standard finite-differences as well as the pseudo-spectral analytical time-domain (PSATD) and pseudo-spectral time-domain (PSTD) algorithms. Extension to multiresolution (or mesh refinement) PIC is described in, e.g. (Vay et al. 2012; Vay, Adam, and Heron 2004).

Finite-Difference Time-Domain (FDTD)

The most popular algorithm for electromagnetic PIC codes is the Finite-Difference Time-Domain (or FDTD) solver

$$\begin{aligned} D_t\mathbf{B} &= -\nabla\times\mathbf{E} \\ D_t\mathbf{E} &= \nabla\times\mathbf{B}-\mathbf{J} \\ [\nabla\cdot\mathbf{E} &= \rho] \\ [\nabla\cdot\mathbf{B} &= 0]\,. \end{aligned}$$



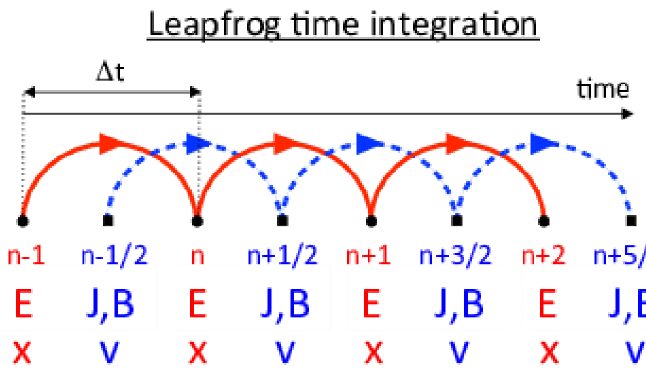


Fig. 5.3: [fig:yee_grid](left) Layout of field components on the staggered "Yee" grid. Current densities and electric fields are defined on the edges of the cells and magnetic fields on the faces. (right) Time integration using a second-order finite-difference "leapfrog" integrator.

The differential operator is defined as $\nabla = D_x\hat{\mathbf{x}} + D_y\hat{\mathbf{y}} + D_z\hat{\mathbf{z}}$ and the finite-difference operators in time and space are defined respectively as

$$D_tG|^n_{i,j,k} = \left(G|^{n+1/2}_{i,j,k} - G|^{n-1/2}_{i,j,k}\right)/\Delta t$$

and $D_xG|^n_{i,j,k} = \left(G|^n_{i+1/2,j,k} - G|^n_{i-1/2,j,k}\right)/\Delta x$, where Δt and Δx are respectively the time step and the grid cell size along x, n is the time index and i, j and k are the spatial indices along x, y and z respectively. The difference operators along y and z are obtained by circular permutation. The equations in brackets are given for completeness, as they are often not actually solved, thanks to the usage of a so-called charge conserving algorithm, as explained below. As shown in Figure *[fig:yee_grid]*, the quantities are given on a staggered (or "Yee") grid (Yee 1966), where the electric field components are located between nodes and the magnetic field components are located in the center of the cell faces. Knowing the current densities at half-integer steps, the electric field components are updated alternately with the magnetic field components at integer and half-integer steps respectively.

Non-Standard Finite-Difference Time-Domain (NSFDTD)

In (Cole 1997, 2002), Cole introduced an implementation of the source-free Maxwell's wave equations for narrow-band applications based on non-standard finite-differences (NSFD). In (Karkkainen et al. 2006), Karkkainen *et al.* adapted it for wideband applications. At the Courant limit for the time step and for a given set of parameters, the stencil proposed in (Karkkainen et al. 2006) has no numerical dispersion along the principal axes, provided that the cell size is the same along each dimension (i.e. cubic cells in 3D). The "Cole-Karkkainnen" (or CK) solver uses the non-standard finite difference formulation (based on extended stencils) of the Maxwell-Ampere equation and can be implemented as follows (Vay et al. 2011):

$$\begin{aligned} D_t\mathbf{B} = &\quad -\nabla^*\times\mathbf{E} \\ D_t\mathbf{E} = &\quad \nabla\times\mathbf{B}-\mathbf{J} \\ [\nabla\cdot\mathbf{E} = &\quad \rho] \\ [\nabla^*\cdot\mathbf{B} = &\quad 0] \end{aligned}$$

Eq. *[Eq:Gauss]* and *[Eq:divb]* are not being solved explicitly but verified via appropriate initial conditions and current deposition procedure. The NSFD differential operators is given by $\nabla^* = D_x^*\hat{\mathbf{x}} + D_y^*\hat{\mathbf{y}} + D_z^*\hat{\mathbf{z}}$ where $D_x^* = \left(\alpha + \beta S_x^1 + \xi S_x^2\right) D_x$ with $S_x^1 G|_{i,j,k}^n = G|_{i,j+1,k}^n + G|_{i,j-1,k}^n + G|_{i,j,k+1}^n + G|_{i,j,k-1}^n$, $S_x^2 G|_{i,j,k}^n = G|_{i,j+1,k+1}^n + G|_{i,j-1,k+1}^n + G|_{i,j+1,k-1}^n + G|_{i,j-1,k-1}^n$. G is a sample vector component, while α, β and ξ are constant scalars satisfying $\alpha + 4\beta + 4\xi = 1$. As with the FDTD algorithm, the quantities with half-integer are located between the nodes (electric field components) or in the center of the cell faces (magnetic field components). The operators along y and z, i.e. D_y, D_z, D_y^*, D_z^*, S_y^1, S_z^1, S_y^2, and S_z^2, are obtained by circular permutation of the indices.

Assuming cubic cells ($\Delta x = \Delta y = \Delta z$), the coefficients given in (Karkkainen et al. 2006) ($\alpha = 7/12$, $\beta = 1/12$ and $\xi = 1/48$) allow for the Courant condition to be at $\Delta t = \Delta x$, which equates to having no numerical dispersion along the principal axes. The algorithm reduces to the FDTD algorithm with $\alpha = 1$ and $\beta = \xi = 0$. An extension to non-cubic cells is provided by Cowan, *et al.* in 3-D in (Cowan et al. 2013) and was given by Pukhov in 2-D in (Pukhov 1999). An alternative NSFDTD implementation that enables superluminous waves is also given by Lehe et al. in (Lehe et al. 2013).

As mentioned above, a key feature of the algorithms based on NSFDTD is that some implementations (Karkkainen et al. 2006; Cowan et al. 2013) enable the time step $\Delta t = \Delta x$ along one or more axes and no numerical dispersion along those axes. However, as shown in (Vay et al. 2011), an instability develops at the Nyquist wavelength at (or very near) such a timestep. It is also shown in the same paper that removing the Nyquist component in all the source terms using a bilinear filter (see description of the filter below) suppresses this instability.

Pseudo Spectral Analytical Time Domain (PSATD)

Maxwell's equations in Fourier space are given by

$$\begin{aligned} \frac{\partial\tilde{\mathbf{E}}}{\partial t} = &\quad i\mathbf{k}\times\tilde{\mathbf{B}}-\tilde{\mathbf{J}} \\ \frac{\partial\tilde{\mathbf{B}}}{\partial t} = &\quad -i\mathbf{k}\times\tilde{\mathbf{E}} \\ [i\mathbf{k}\cdot\tilde{\mathbf{E}} = &\quad \tilde{\rho}] \\ [i\mathbf{k}\cdot\tilde{\mathbf{B}} = &\quad 0] \end{aligned}$$

where $\tilde{a}$ is the Fourier Transform of the quantity a. As with the real space formulation, provided that the continuity equation $\partial\tilde{\rho}/\partial t + i\mathbf{k}\cdot\tilde{\mathbf{J}} = 0$ is satisfied, then the last two equations will automatically be satisfied at any time if satisfied initially and do not need to be explicitly integrated.

Decomposing the electric field and current between longitudinal and transverse components $\tilde{\mathbf{E}} = \tilde{\mathbf{E}}_L + \tilde{\mathbf{E}}_T = \hat{\mathbf{k}}(\hat{\mathbf{k}}\cdot\tilde{\mathbf{E}}) - \hat{\mathbf{k}}\times(\hat{\mathbf{k}}\times\tilde{\mathbf{E}})$ and $\tilde{\mathbf{J}} = \tilde{\mathbf{J}}_L + \tilde{\mathbf{J}}_T = \hat{\mathbf{k}}(\hat{\mathbf{k}}\cdot\tilde{\mathbf{J}}) - \hat{\mathbf{k}}\times(\hat{\mathbf{k}}\times\tilde{\mathbf{J}})$ gives

$$\begin{aligned}
\frac{\partial\tilde{\mathbf{E}}_T}{\partial t} &= i\mathbf{k}\times\tilde{\mathbf{B}} - \tilde{\mathbf{J}}_{\mathbf{T}}\\
\frac{\partial\tilde{\mathbf{E}}_L}{\partial t} &= -\tilde{\mathbf{J}}_{\mathbf{L}}\\
\frac{\partial\tilde{\mathbf{B}}}{\partial t} &= -i\mathbf{k}\times\tilde{\mathbf{E}}
\end{aligned}$$

with $\hat{\mathbf{k}} = \mathbf{k}/k$.

If the sources are assumed to be constant over a time interval Δt, the system of equations is solvable analytically and is given by (see (Haber et al. 1973) for the original formulation and (Jean-Luc Vay, Haber, and Godfrey 2013) for a more detailed derivation):

[Eq:PSATD]

$$\begin{aligned}
\tilde{\mathbf{E}}_T^{n+1} &= C\tilde{\mathbf{E}}_T^n + iS\hat{\mathbf{k}}\times\tilde{\mathbf{B}}^n - \frac{S}{k}\tilde{\mathbf{J}}_T^{n+1/2}\\
\tilde{\mathbf{E}}_L^{n+1} &= \tilde{\mathbf{E}}_L^n - \Delta t\tilde{\mathbf{J}}_L^{n+1/2}\\
\tilde{\mathbf{B}}^{n+1} &= C\tilde{\mathbf{B}}^n - iS\hat{\mathbf{k}}\times\tilde{\mathbf{E}}^n\\
&+ i\frac{1-C}{k}\hat{\mathbf{k}}\times\tilde{\mathbf{J}}^{n+1/2}
\end{aligned}$$

with $C = \cos(k\Delta t)$ and $S = \sin(k\Delta t)$.

Combining the transverse and longitudinal components, gives

$$\begin{aligned}
\tilde{\mathbf{E}}^{n+1} &= C\tilde{\mathbf{E}}^n + iS\hat{\mathbf{k}}\times\tilde{\mathbf{B}}^n - \frac{S}{k}\tilde{\mathbf{J}}^{n+1/2}\\
&+ (1-C)\hat{\mathbf{k}}(\hat{\mathbf{k}}\cdot\tilde{\mathbf{E}}^n)\\
&+ \hat{\mathbf{k}}(\hat{\mathbf{k}}\cdot\tilde{\mathbf{J}}^{n+1/2})\left(\frac{S}{k} - \Delta t\right),\\
\tilde{\mathbf{B}}^{n+1} &= C\tilde{\mathbf{B}}^n - iS\hat{\mathbf{k}}\times\tilde{\mathbf{E}}^n\\
&+ i\frac{1-C}{k}\hat{\mathbf{k}}\times\tilde{\mathbf{J}}^{n+1/2}.
\end{aligned}$$

For fields generated by the source terms without the self-consistent dynamics of the charged particles, this algorithm is free of numerical dispersion and is not subject to a Courant condition. Furthermore, this solution is exact for any time step size subject to the assumption that the current source is constant over that time step.

As shown in (Jean-Luc Vay, Haber, and Godfrey 2013), by expanding the coefficients S_h and C_h in Taylor series and keeping the leading terms, the PSATD formulation reduces to the perhaps better known pseudo-spectral time-domain (PSTD) formulation (Dawson 1983; Liu 1997):

$$\begin{aligned}
\tilde{\mathbf{E}}^{n+1} &= \tilde{\mathbf{E}}^n + i\Delta t\mathbf{k}\times\tilde{\mathbf{B}}^{n+1/2} - \Delta t\tilde{\mathbf{J}}^{n+1/2},\\
\tilde{\mathbf{B}}^{n+3/2} &= \tilde{\mathbf{B}}^{n+1/2} - i\Delta t\mathbf{k}\times\tilde{\mathbf{E}}^{n+1}.
\end{aligned}$$

The dispersion relation of the PSTD solver is given by $\sin(\frac{\omega\Delta t}{2}) = \frac{k\Delta t}{2}$. In contrast to the PSATD solver, the PSTD solver is subject to numerical dispersion for a finite time step and to a Courant condition that is given by $\Delta t \leq \frac{2}{\pi}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta x^2}\right)^{-1/2}$.

The PSATD and PSTD formulations that were just given apply to the field components located at the nodes of the grid. As noted in (Ohmura and Okamura 2010), they can also be easily recast on a staggered Yee grid by multiplication of the field components by the appropriate phase factors to shift them from the collocated to the staggered locations. The choice between a collocated and a staggered formulation is application-dependent.

Spectral solvers used to be very popular in the years 1970s to early 1990s, before being replaced by finite-difference methods with the advent of parallel supercomputers that favored local methods. However, it was shown recently that standard domain decomposition with Fast Fourier Transforms that are local to each subdomain could be used effectively with PIC spectral methods (Jean-Luc Vay, Haber, and Godfrey 2013), at the cost of truncation errors in the guard cells that could be neglected. A detailed analysis of the effectiveness of the method with exact evaluation of the magnitude of the effect of the truncation error is given in (Vincenti and Vay 2016) for stencils of arbitrary order (up-to the infinite "spectral" order).

5.2.3 Current deposition

The current densities are deposited on the computational grid from the particle position and velocities, employing splines of various orders (Abe et al. 1986).

$$\begin{aligned}\rho &= \frac{1}{\Delta x\Delta y\Delta z}\sum_n q_n S_n \\ \mathbf{J} &= \frac{1}{\Delta x\Delta y\Delta z}\sum_n q_n \mathbf{v_n} S_n\end{aligned}$$

In most applications, it is essential to prevent the accumulation of errors resulting from the violation of the discretized Gauss' Law. This is accomplished by providing a method for depositing the current from the particles to the grid that preserves the discretized Gauss' Law, or by providing a mechanism for "divergence cleaning" (Birdsall and Langdon 1991; Langdon 1992; Marder 1987; Vay and Deutsch 1998; Munz et al. 2000). For the former, schemes that allow a deposition of the current that is exact when combined with the Yee solver is given in (Villasenor and Buneman 1992) for linear splines and in (Esirkepov 2001) for splines of arbitrary order.

The NSFDTD formulations given above and in (Pukhov 1999; Vay et al. 2011; Cowan et al. 2013; Lehe et al. 2013) apply to the Maxwell-Faraday equation, while the discretized Maxwell-Ampere equation uses the FDTD formulation. Consequently, the charge conserving algorithms developed for current deposition (Villasenor and Buneman 1992; Esirkepov 2001) apply readily to those NSFDTD-based formulations. More details concerning those implementations, including the expressions for the numerical dispersion and Courant condition are given in (Pukhov 1999; Vay et al. 2011; Cowan et al. 2013; Lehe et al. 2013).

In the case of the pseudospectral solvers, the current deposition algorithm generally does not satisfy the discretized continuity equation in Fourier space $\tilde{\rho}^{n+1} = \tilde{\rho}^n - i\Delta t\mathbf{k}\cdot\tilde{\mathbf{J}}^{n+1/2}$. In this case, a Boris correction (Birdsall and Langdon 1991) can be applied in k space in the form $\tilde{\mathbf{E}}_c^{n+1} = \tilde{\mathbf{E}}^{n+1} - \left(\mathbf{k}\cdot\tilde{\mathbf{E}}^{n+1} + i\tilde{\rho}^{n+1}\right)\hat{\mathbf{k}}/k$, where $\tilde{\mathbf{E}}_c$ is the corrected field. Alternatively, a correction to the current can be applied (with some similarity to the current deposition presented by Morse and Nielson in their potential-based model in (Morse and Nielson 1971)) using $\tilde{\mathbf{J}}_c^{n+1/2} = \tilde{\mathbf{J}}^{n+1/2} - \left[\mathbf{k}\cdot\tilde{\mathbf{J}}^{n+1/2} - i\left(\tilde{\rho}^{n+1} - \tilde{\rho}^n\right)/\Delta t\right]\hat{\mathbf{k}}/k$, where $\tilde{\mathbf{J}}_c$ is the corrected current. In this case, the transverse component of the current is left untouched while the longitudinal component is effectively replaced by the one obtained from integration of the continuity equation, ensuring that the corrected current satisfies the continuity equation. The advantage of correcting the current rather than the electric field is that it is more local and thus more compatible with domain decomposition of the fields for parallel computation (Jean Luc Vay, Haber, and Godfrey 2013).

Alternatively, an exact current deposition can be written for the pseudospectral solvers, following the geometrical interpretation of existing methods in real space (Morse and Nielson 1971; Villasenor and Buneman 1992; Esirkepov 2001), thereby averaging the currents of the paths following grid lines between positions (x^n, y^n) and (x^{n+1}, y^{n+1}), which is given in 2D (extension to 3D follows readily) for $k \neq 0$ by (Jean Luc Vay, Haber, and Godfrey 2013):

$$\tilde{\mathbf{J}}^{k\neq 0} = \frac{i\tilde{\mathbf{D}}}{\mathbf{k}}$$

with

$$
\begin{aligned}
D_x = & \frac{1}{2\Delta t}\sum_i q_i[\Gamma(x_i^{n+1},y_i^{n+1})-\Gamma(x_i^n,y_i^{n+1}) \\
& +\Gamma(x_i^{n+1},y_i^n)-\Gamma(x_i^n,y_i^n)], \\
D_y = & \frac{1}{2\Delta t}\sum_i q_i[\Gamma(x_i^{n+1},y_i^{n+1})-\Gamma(x_i^{n+1},y_i^n) \\
& +\Gamma(x_i^n,y_i^{n+1})-\Gamma(x_i^n,y_i^n)],
\end{aligned}
$$

where Γ is the macro-particle form factor. The contributions for $k=0$ are integrated directly in real space (Jean Luc Vay, Haber, and Godfrey 2013).

5.2.4 Field gather

The current densities are deposited on the computational grid from the particle position and velocities, employing splines of various orders (Abe et al. 1986).

$$
\begin{aligned}
\rho = & \frac{1}{\Delta x\Delta y\Delta z}\sum_n q_n S_n \\
\mathbf{J} = & \frac{1}{\Delta x\Delta y\Delta z}\sum_n q_n \mathbf{v_n} S_n
\end{aligned}
$$

In most applications, it is essential to prevent the accumulation of errors resulting from the violation of the discretized Gauss' Law. This is accomplished by providing a method for depositing the current from the particles to the grid that preserves the discretized Gauss' Law, or by providing a mechanism for "divergence cleaning" (Birdsall and Langdon 1991; Langdon 1992; Marder 1987; Vay and Deutsch 1998; Munz et al. 2000). For the former, schemes that allow a deposition of the current that is exact when combined with the Yee solver is given in (Villasenor and Buneman 1992) for linear splines and in (Esirkepov 2001) for splines of arbitrary order.

The NSFDTD formulations given above and in (Pukhov 1999; Vay et al. 2011; Cowan et al. 2013; Lehe et al. 2013) apply to the Maxwell-Faraday equation, while the discretized Maxwell-Ampere equation uses the FDTD formulation. Consequently, the charge conserving algorithms developed for current deposition (Villasenor and Buneman 1992; Esirkepov 2001) apply readily to those NSFDTD-based formulations. More details concerning those implementations, including the expressions for the numerical dispersion and Courant condition are given in (Pukhov 1999; Vay et al. 2011; Cowan et al. 2013; Lehe et al. 2013).

In the case of the pseudospectral solvers, the current deposition algorithm generally does not satisfy the discretized continuity equation in Fourier space $\tilde{\rho}^{n+1}=\tilde{\rho}^n-i\Delta t\mathbf{k}\cdot\mathbf{\tilde{J}}^{n+1/2}$. In this case, a Boris correction (Birdsall and Langdon 1991) can be applied in k space in the form $\mathbf{\tilde{E}}_c^{n+1}=\mathbf{\tilde{E}}^{n+1}-\left(\mathbf{k}\cdot\mathbf{\tilde{E}}^{n+1}+i\tilde{\rho}^{n+1}\right)\mathbf{\hat{k}}/k$, where $\mathbf{\tilde{E}}_c$ is the corrected field. Alternatively, a correction to the current can be applied (with some similarity to the current deposition presented by Morse and Nielson in their potential-based model in (Morse and Nielson 1971)) using $\mathbf{\tilde{J}}_c^{n+1/2}=\mathbf{\tilde{J}}^{n+1/2}-\left[\mathbf{k}\cdot\mathbf{\tilde{J}}^{n+1/2}-i\left(\tilde{\rho}^{n+1}-\tilde{\rho}^n\right)/\Delta t\right]\mathbf{\hat{k}}/k$, where $\mathbf{\tilde{J}}_c$ is the corrected current. In this case, the transverse component of the current is left untouched while the longitudinal component is effectively replaced by the one obtained from integration of the continuity equation, ensuring that the corrected current satisfies the continuity equation. The advantage of correcting the current rather than the electric field is that it is more local and thus more compatible with domain decomposition of the fields for parallel computation (Jean Luc Vay, Haber, and Godfrey 2013).

Alternatively, an exact current deposition can be written for the pseudospectral solvers, following the geometrical interpretation of existing methods in real space (Morse and Nielson 1971; Villasenor and Buneman 1992; Esirkepov 2001), thereby averaging the currents of the paths following grid lines between positions (x^n,y^n) and (x^{n+1},y^{n+1}), which is given in 2D (extension to 3D follows readily) for $k\neq0$ by (Jean Luc Vay, Haber, and Godfrey 2013):

$$
\mathbf{\tilde{J}}^{k\neq0}=\frac{i\mathbf{\tilde{D}}}{\mathbf{k}}
$$

with

$$D_x = \frac{1}{2\Delta t}\sum_i q_i[\Gamma(x_i^{n+1}, y_i^{n+1}) - \Gamma(x_i^n, y_i^{n+1}) \\ +\Gamma(x_i^{n+1}, y_i^n) - \Gamma(x_i^n, y_i^n)],$$

$$D_y = \frac{1}{2\Delta t}\sum_i q_i[\Gamma(x_i^{n+1}, y_i^{n+1}) - \Gamma(x_i^{n+1}, y_i^n) \\ +\Gamma(x_i^n, y_i^{n+1}) - \Gamma(x_i^n, y_i^n)],$$

where Γ is the macro-particle form factor. The contributions for $k = 0$ are integrated directly in real space (Jean Luc Vay, Haber, and Godfrey 2013).

5.3 Filtering

It is common practice to apply digital filtering to the charge or current density in Particle-In-Cell simulations as a complement or an alternative to using higher order splines (Birdsall and Langdon 1991). A commonly used filter in PIC simulations is the three points filter $\phi_j^f = \alpha\phi_j + (1-\alpha)(\phi_{j-1} + \phi_{j+1})/2$ where ϕ^f is the filtered quantity. This filter is called a bilinear filter when $\alpha = 0.5$. Assuming $\phi = e^{jkx}$ and $\phi^f = g(\alpha, k)\, e^{jkx}$, the filter gain g is given as a function of the filtering coefficient α and the wavenumber k by $g(\alpha, k) = \alpha + (1-\alpha)\cos(k\Delta x) \approx 1 - (1-\alpha)\frac{(k\Delta x)^2}{2} + O(k^4)$. The total attenuation G for n successive applications of filters of coefficients $\alpha_1 \ldots \alpha_n$ is given by $G = \prod_{i=1}^{n} g(\alpha_i, k) \approx 1 - (n - \sum_{i=1}^{n}\alpha_i)\frac{(k\Delta x)^2}{2} + O(k^4)$. A sharper cutoff in k space is provided by using $\alpha_n = n - \sum_{i=1}^{n-1}\alpha_i$, so that $G \approx 1 + O(k^4)$. Such step is called a "compensation" step (Birdsall and Langdon 1991). For the bilinear filter ($\alpha = 1/2$), the compensation factor is $\alpha_c = 2 - 1/2 = 3/2$. For a succession of n applications of the bilinear factor, it is $\alpha_c = n/2 + 1$.

It is sometimes necessary to filter on a relatively wide band of wavelength, necessitating the application of a large number of passes of the bilinear filter or on the use of filters acting on many points. The former can become very intensive computationally while the latter is problematic for parallel computations using domain decomposition, as the footprint of the filter may eventually surpass the size of subdomains. A workaround is to use a combination of filters of limited footprint. A solution based on the combination of three point filters with various strides was proposed in (Vay et al. 2011) and operates as follows.

The bilinear filter provides complete suppression of the signal at the grid Nyquist wavelength (twice the grid cell size). Suppression of the signal at integer multiples of the Nyquist wavelength can be obtained by using a stride s in the filter $\phi_j^f = \alpha\phi_j + (1-\alpha)(\phi_{j-s} + \phi_{j+s})/2$ for which the gain is given by $g(\alpha, k) = \alpha + (1-\alpha)\cos(sk\Delta x) \approx 1 - (1-\alpha)\frac{(sk\Delta x)^2}{2} + O(k^4)$. For a given stride, the gain is given by the gain of the bilinear filter shifted in k space, with the pole $g = 0$ shifted from the wavelength $\lambda = 2/\Delta x$ to $\lambda = 2s/\Delta x$, with additional poles, as given by $sk\Delta x = \arccos\left(\frac{\alpha}{\alpha-1}\right) \pmod{2\pi}$. The resulting filter is pass band between the poles, but since the poles are spread at different integer values in k space, a wide band low pass filter can be constructed by combining filters using different strides. As shown in (Vay et al. 2011), the successive application of 4-passes + compensation of filters with strides 1, 2 and 4 has a nearly equivalent fall-off in gain as 80 passes + compensation of a bilinear filter. Yet, the strided filter solution needs only 15 passes of a three-point filter, compared to 81 passes for an equivalent n-pass bilinear filter, yielding a gain of 5.4 in number of operations in favor of the combination of filters with stride. The width of the filter with stride 4 extends only on 9 points, compared to 81 points for a single pass equivalent filter, hence giving a gain of 9 in compactness for the stride filters combination in comparison to the single-pass filter with large stencil, resulting in more favorable scaling with the number of computational cores for parallel calculations.

Abe, H, N Sakairi, R Itatani, and H Okuda. 1986. "High-Order Spline Interpolations in the Particle Simulation." *Journal of Computational Physics* 63 (2): 247–67.

Birdsall, C K, and A B Langdon. 1991. *Plasma Physics via Computer Simulation*. Adam-Hilger.

Boris, Jp. 1970. "Relativistic Plasma Simulation-Optimization of a Hybrid Code." In *Proc. Fourth Conf. Num. Sim. Plasmas*, 3–67. Naval Res. Lab., Wash., D. C.

Cole, J. B. 1997. "A High-Accuracy Realization of the Yee Algorithm Using Non-Standard Finite Differences." *Ieee Transactions on Microwave Theory and Techniques* 45 (6): 991–96.

———. 2002. "High-Accuracy Yee Algorithm Based on Nonstandard Finite Differences: New Developments and Verifications." *Ieee Transactions on Antennas and Propagation* 50 (9): 1185–91. https://doi.org/10.1109/Tap.2002.801268.

Cowan, Benjamin M, David L Bruhwiler, John R Cary, Estelle Cormier-Michel, and Cameron G R Geddes. 2013. "Generalized algorithm for control of numerical dispersion in explicit time-domain electromagnetic simulations." *Physical Review Special Topics-Accelerators and Beams* 16 (4). https://doi.org/10.1103/PhysRevSTAB.16.041303.

Dawson, J M. 1983. "Particle Simulation of Plasmas." *Reviews of Modern Physics* 55 (2): 403–47. https://doi.org/10.1103/RevModPhys.55.403.

Esirkepov, Tz. 2001. "Exact Charge Conservation Scheme for Particle-in-Cell Simulation with an Arbitrary Form-Factor." *Computer Physics Communications* 135 (2): 144–53.

Haber, I, R Lee, Hh Klein, and Jp Boris. 1973. "Advances in Electromagnetic Simulation Techniques." In *Proc. Sixth Conf. Num. Sim. Plasmas*, 46–48. Berkeley, Ca.

Karkkainen, M, E Gjonaj, T Lau, and T Weiland. 2006. "Low-Dispersionwake Field Calculation Tools." In *Proc. Of International Computational Accelerator Physics Conference*, 35–40. Chamonix, France.

Langdon, A B. 1992. "On Enforcing Gauss Law in Electromagnetic Particle-in-Cell Codes." *Computer Physics Communications* 70 (3): 447–50.

Lehe, R, A Lifschitz, C Thaury, V Malka, and X Davoine. 2013. "Numerical growth of emittance in simulations of laser-wakefield acceleration." *Physical Review Special Topics-Accelerators and Beams* 16 (2). https://doi.org/10.1103/PhysRevSTAB.16.021301.

Liu, Qh. 1997. "The PSTD Algorithm: A Time-Domain Method Requiring Only Two Cells Per Wavelength." *Microwave and Optical Technology Letters* 15 (3): 158–65. **'https://doi.org/10.1002/(Sici)1098-2760(19970620)15:3<158::Aid-Mop11>3.3.Co;2-T <https://doi.org/10.1002/(Sici)1098-2760(19970620)15:3<158::Aid-Mop11>3.3.Co;2-T>'__**.

Marder, B. 1987. "A Method for Incorporating Gauss Law into Electromagnetic Pic Codes." *Journal of Computational Physics* 68 (1): 48–55.

Morse, Rl, and Cw Nielson. 1971. "Numerical Simulation of Weibel Instability in One and 2 Dimensions." *Phys. Fluids* 14 (4): 830 –&. https://doi.org/10.1063/1.1693518.

Munz, Cd, P Omnes, R Schneider, E Sonnendrucker, and U Voss. 2000. "Divergence Correction Techniques for Maxwell Solvers Based on A Hyperbolic Model." *Journal of Computational Physics* 161 (2): 484–511. https://doi.org/10.1006/Jcph.2000.6507.

Ohmura, Y, and Y Okamura. 2010. "Staggered Grid Pseudo-Spectral Time-Domain Method for Light Scattering Analysis." *Piers Online* 6 (7): 632–35.

Pukhov, A. 1999. "Three-dimensional electromagnetic relativistic particle-in-cell code VLPL (Virtual Laser Plasma Lab)." *Journal of Plasma Physics* 61 (3): 425–33. https://doi.org/10.1017/S0022377899007515.

Vay, Jean-Luc, Irving Haber, and Brendan B Godfrey. 2013. "A domain decomposition method for pseudo-spectral electromagnetic simulations of plasmas." *Journal of Computational Physics* 243 (June): 260–68. https://doi.org/10.1016/j.jcp.2013.03.010.

Vay, Jean Luc, Irving Haber, and Brendan B. Godfrey. 2013. "A domain decomposition method for pseudo-spectral electromagnetic simulations of plasmas." *Journal of Computational Physics* 243: 260–68.

Vay, J L. 2008. "Simulation of Beams or Plasmas Crossing at Relativistic Velocity." *Physics of Plasmas* 15 (5): 56701. https://doi.org/10.1063/1.2837054.

Vay, J.-L., J.-C. Adam, and A Heron. 2004. “Asymmetric Pml for the Absorption of Waves. Application to Mesh Refinement in Electromagnetic Particle-in-Cell Plasma Simulations.” *Computer Physics Communications* 164 (1-3): 171–77. https://doi.org/10.1016/J.Cpc.2004.06.026.

Vay, J.-L., and C Deutsch. 1998. “Charge Compensated Ion Beam Propagation in A Reactor Sized Chamber.” *Physics of Plasmas* 5 (4): 1190–7.

Vay, J L, C G R Geddes, E Cormier-Michel, and D P Grote. 2011. “Numerical Methods for Instability Mitigation in the Modeling of Laser Wakefield Accelerators in A Lorentz-Boosted Frame.” *Journal of Computational Physics* 230 (15): 5908–29. https://doi.org/10.1016/J.Jcp.2011.04.003.

Vay, J.-L., D P Grote, R H Cohen, and A Friedman. 2012. “Novel methods in the particle-in-cell accelerator code-framework warp.” Journal Paper. *Computational Science and Discovery* 5 (1): 014019 (20 pp.).

Villasenor, J, and O Buneman. 1992. “Rigorous Charge Conservation for Local Electromagnetic-Field Solvers.” *Computer Physics Communications* 69 (2-3): 306–16.

Vincenti, H., and J.-L. Vay. 2016. “Detailed analysis of the effects of stencil spatial variations with arbitrary high-order finite-difference Maxwell solver.” *Computer Physics Communications* 200 (March). ELSEVIER SCIENCE BV, PO BOX 211, 1000 AE AMSTERDAM, NETHERLANDS: 147–67. https://doi.org/10.1016/j.cpc.2015.11.009.

Yee, Ks. 1966. “Numerical Solution of Initial Boundary Value Problems Involving Maxwells Equations in Isotropic Media.” *Ieee Transactions on Antennas and Propagation* Ap14 (3): 302–7.

5.4 Mesh refinement

The mesh refinement methods that have been implemented in WarpX were developed following the following principles: i) avoidance of spurious effects from mesh refinement, or minimization of such effects; ii) user controllability of the spurious effects’ relative magnitude; iii) simplicity of implementation. The two main generic issues that were identified are: a) spurious self-force on macroparticles close to the mesh refinement interface (J. Vay et al. 2002; Colella and Norgaard 2010); b) reflection (and possible amplification) of short wavelength electromagnetic waves at the mesh refinement interface (Vay 2001). The two effects are due to the loss of translation invariance introduced by the asymmetry of the grid on each side of the mesh refinement interface.

In addition, for some implementations where the field that is computed at a given level is affected by the solution at finer levels, there are cases where the procedure violates the integral of Gauss’ Law around the refined patch, leading to long range errors (J. Vay et al. 2002; Colella and Norgaard 2010). As will be shown below, in the procedure that has been developed in WarpX, the field at a given refinement level is not affected by the solution at finer levels, and is thus not affected by this type of error.

5.4.1 Electrostatic

A cornerstone of the Particle-In-Cell method is that assuming a particle lying in a hypothetical infinite grid, then if the grid is regular and symmetrical, and if the order of field gathering matches the order of charge (or current) deposition, then there is no self-force of the particle acting on itself: a) anywhere if using the so-called “momentum conserving” gathering scheme; b) on average within one cell if using the “energy conserving” gathering scheme (Birdsall and Langdon 1991). A breaking of the regularity and/or symmetry in the grid, whether it is from the use of irregular meshes or mesh refinement, and whether one uses finite difference, finite volume or finite elements, results in a net spurious self-force (which does not average to zero over one cell) for a macroparticle close to the point of irregularity (mesh refinement interface for the current purpose) (J. Vay et al. 2002; Colella and Norgaard 2010).

A sketch of the implementation of mesh refinement in WarpX is given in Figure *[fig:ESAMR]* (left). Given the solution of the electric potential at a refinement level L_n, it is interpolated onto the boundaries of the grid patch(es) at the next refined level L_{n+1}. The electric potential is then computed at level L_{n+1} by solving the Poisson equation. This procedure necessitates the knowledge of the charge density at every level of refinement. For efficiency, the

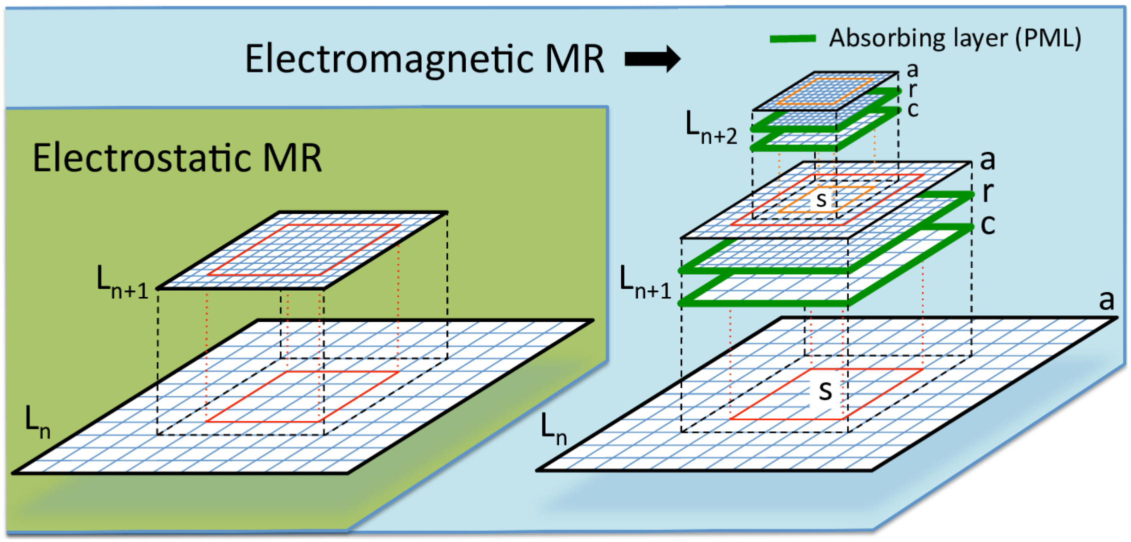


Fig. 5.4: Sketches of the implementation of mesh refinement in WarpX with the electrostatic (left) and electromagnetic (right) solvers. In both cases, the charge/current from particles are deposited at the finest levels first, then interpolated recursively to coarser levels. In the electrostatic case, the potential is calculated first at the coarsest level L_0, the solution interpolated to the boundaries of the refined patch r at the next level L_1 and the potential calculated at L_1. The procedure is repeated iteratively up to the highest level. In the electromagnetic case, the fields are computed independently on each grid and patch without interpolation at boundaries. Patches are terminated by absorbing layers (PML) to prevent the reflection of electromagnetic waves. Additional coarse patch c and fine grid a are needed so that the full solution is obtained by substitution on a as $F_{n+1}(a) = F_{n+1}(r) + I[F_n(s) - F_{n+1}(c)]$ where F is the field, and I is a coarse-to-fine interpolation operator. In both cases, the field solution at a given level L_n is unaffected by the solution at higher levels L_{n+1} and up, allowing for mitigation of some spurious effects (see text) by providing a transition zone via extension of the patches by a few cells beyond the desired refined area (red & orange rectangles) in which the field is interpolated onto particles from the coarser parent level only.

macroparticle charge is deposited on the highest level patch that contains them, and the charge density of each patch is added recursively to lower levels, down to the lowest.

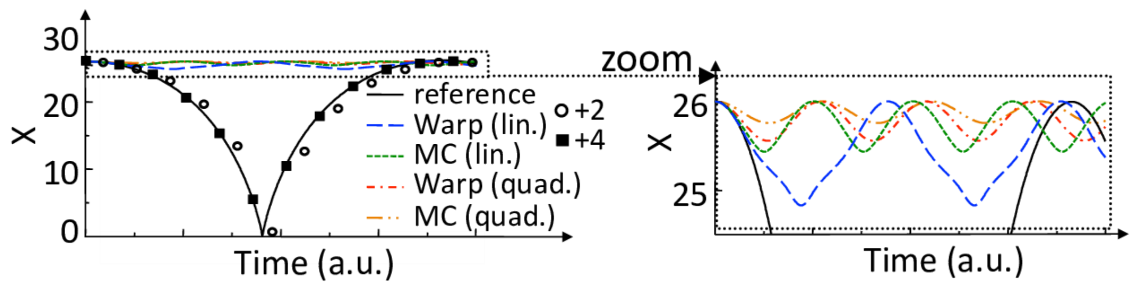


Fig. 5.5: Position history of one charged particle attracted by its image induced by a nearby metallic (dirichlet) boundary. The particle is initialized at rest. Without refinement patch (reference case), the particle is accelerated by its image, is reflected specularly at the wall, then decelerates until it reaches its initial position at rest. If the particle is initialized inside a refinement patch, the particle is initially accelerated toward the wall but is spuriously reflected before it reaches the boundary of the patch whether using the method implemented in WarpX or the MC method. Providing a surrounding transition region 2 or 4 cells wide in which the potential is interpolated from the parent coarse solution reduces significantly the effect of the spurious self-force.

The presence of the self-force is illustrated on a simple test case that was introduced in (J. Vay et al. 2002) and also used in (Colella and Norgaard 2010): a single macroparticle is initialized at rest within a single refinement patch four cells away from the patch refinement boundary. The patch at level L_1 has 32×32 cells and is centered relative to the lowest 64×64 grid at level L_0 ("main grid"), while the macroparticle is centered in one direction but not in the other. The boundaries of the main grid are perfectly conducting, so that the macroparticle is attracted to the closest wall by its image. Specular reflection is applied when the particle reaches the boundary so that the motion is cyclic. The test was performed with WarpX using either linear or quadratic interpolation when gathering the main grid solution onto the refined patch boundary. It was also performed using another method from P. McCorquodale et al (labeled "MC" in this paper) based on the algorithm given in (Mccorquodale et al. 2004), which employs a more elaborate procedure involving two-ways interpolations between the main grid and the refined patch. A reference case was also run using a single 128×128 grid with no refined patch, in which it is observed that the particle propagates toward the closest boundary at an accelerated pace, is reflected specularly at the boundary, then slows down until it reaches its initial position at zero velocity. The particle position histories are shown for the various cases in Fig. *[fig:ESselfforce]*. In all the cases using the refinement patch, the particle was spuriously reflected near the patch boundary and was effectively trapped in the patch. We notice that linear interpolation performs better than quadratic, and that the simple method implemented in WarpX performs better than the other proposed method for this test (see discussion below).

The magnitude of the spurious self-force as a function of the macroparticle position was mapped and is shown in Fig. *[fig:ESselfforcemap]* for the WarpX and MC algorithms using linear or quadratic interpolations between grid levels. It is observed that the magnitude of the spurious self-force decreases rapidly with the distance between the particle and the refined patch boundary, at a rate approaching one order of magnitude per cell for the four cells closest to the boundary and about one order of magnitude per six cells beyond. The method implemented in WarpX offers a weaker spurious force on average and especially at the cells that are the closest to the coarse-fine interface where it is the largest and thus matters most. We notice that the magnitude of the spurious self-force depends strongly on the distance to the edge of the patch and to the nodes of the underlying coarse grid, but weakly on the order of deposition and size of the patch.

A method was devised and implemented in WarpX for reducing the magnitude of spurious self-forces near the coarse-fine boundaries as follows. Noting that the coarse grid solution is unaffected by the presence of the patch and is thus free of self-force, extra "transition" cells are added around the "effective" refined area. Within the effective area, the particles gather the potential in the fine grid. In the extra transition cells surrounding the refinement patch, the force is gathered directly from the coarse grid (an option, which has not yet been implemented, would be to interpolate between the coarse and fine grid field solutions within the transition zone so as to provide continuity of the force experienced by the particles at the interface). The number of cells allocated in the transition zones is

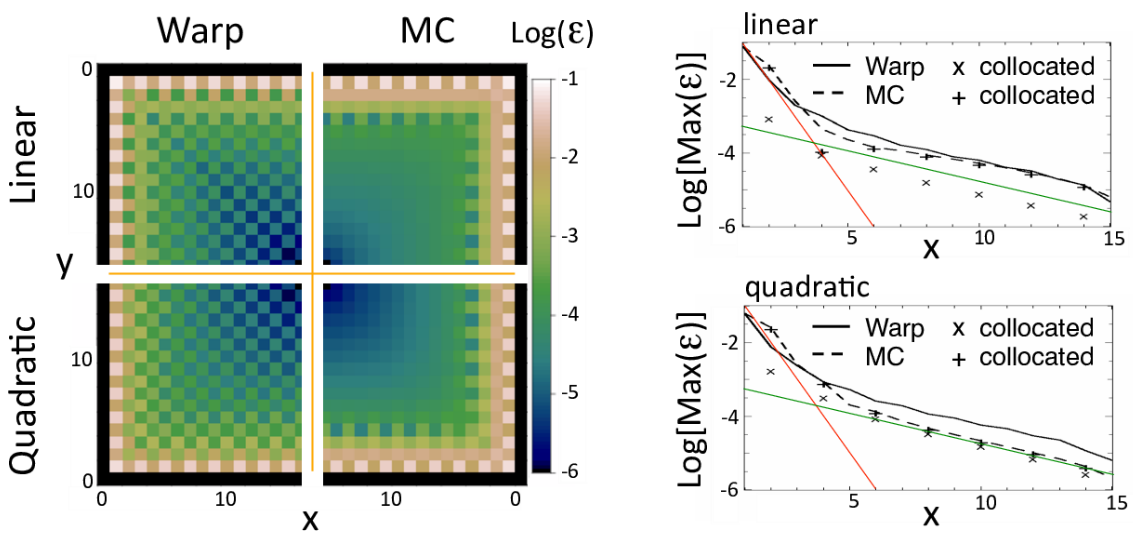


Fig. 5.6: (left) Maps of the magnitude of the spurious self-force ϵ in arbitrary units within one quarter of the refined patch, defined as $\epsilon = \sqrt{(E_x - E_x^{ref})^2 + (E_y - E_y^{ref})^2}$, where E_x and E_y are the electric field components within the patch experienced by one particle at a given location and E_x^{ref} and E_y^{ref} are the electric field from a reference solution. The map is given for the WarpX and the MC mesh refinement algorithms and for linear and quadratic interpolation at the patch refinement boundary. (right) Lineouts of the maximum (taken over neighboring cells) of the spurious self-force. Close to the interface boundary (x=0), the spurious self-force decreases at a rate close to one order of magnitude per cell (red line), then at about one order of magnitude per six cells (green line).

controllable by the user in WarpX, giving the opportunity to check whether the spurious self-force is affecting the calculation by repeating it using different thicknesses of the transition zones. The control of the spurious force using the transition zone is illustrated in Fig. *[fig:ESselfforce]*, where the calculation with WarpX using linear interpolation at the patch interface was repeated using either two or four cells transition regions (measured in refined patch cell units). Using two extra cells allowed for the particle to be free of spurious trapping within the refined area and follow a trajectory that is close to the reference one, and using four extra cells improved further to the point where the resulting trajectory becomes undistinguishable from the reference one. We note that an alternative method was devised for reducing the magnitude of self-force near the coarse-fine boundaries for the MC method, by using a special deposition procedure near the interface (Colella and Norgaard 2010).

5.4.2 Electromagnetic

The method that is used for electrostatic mesh refinement is not directly applicable to electromagnetic calculations. As was shown in section 3.4 of (Vay 2001), refinement schemes relying solely on interpolation between coarse and fine patches lead to the reflection with amplification of the short wavelength modes that fall below the cutoff of the Nyquist frequency of the coarse grid. Unless these modes are damped heavily or prevented from occurring at their source, they may affect particle motion and their effect can escalate if trapped within a patch, via multiple successive reflections with amplification.

To circumvent this issue, an additional coarse patch (with the same resolution as the parent grid) is added, as shown in Fig. *[fig:ESAMR]*-right and described in (Vay, Adam, and Heron 2004). Both the fine and the coarse grid patches are terminated by Perfectly Matched Layers, reducing wave reflection by orders of magnitude, controllable by the user (Berenger 1996; J.-L. Vay 2002). The source current resulting from the motion of charged macroparticles within the refined region is accumulated on the fine patch and is then interpolated onto the coarse patch and added onto the parent grid. The process is repeated recursively from the finest level down to the coarsest. The Maxwell equations are then solved for one time interval on the entire set of grids, by default for one time step using the time step of the finest grid. The field on the coarse and fine patches only contain the contributions from the particles that have evolved within the refined area but not from the current sources outside the area. The total contribution of the field from sources within and outside the refined area is obtained by adding the field from the refined grid $F(r)$, and adding an interpolation I of the difference between the relevant subset s of the field in the parent grid $F(s)$ and the field of the coarse grid $F(c)$, on an auxiliary grid a, i.e. $F(a) = F(r) + I[F(s) - F(c)]$. The field on the parent grid subset $F(s)$ contains contributions from sources from both within and outside of the refined area. Thus, in effect, there is substitution of the coarse field resulting from sources within the patch area by its fine resolution counterpart. The operation is carried out recursively starting at the coarsest level up to the finest. An option has been implemented in which various grid levels are pushed with different time steps, given as a fixed fraction of the individual grid Courant conditions (assuming same cell aspect ratio for all grids and refinement by integer factors). In this case, the fields from the coarse levels, which are advanced less often, are interpolated in time.

The substitution method has two potential drawbacks due to the inexact cancellation between the coarse and fine patches of : (i) the remnants of ghost fixed charges created by the particles entering and leaving the patches (this effect is due to the use of the electromagnetic solver and is different from the spurious self-force that was described for the electrostatic case); (ii) if using a Maxwell solver with a low-order stencil, the electromagnetic waves traveling on each patch at slightly different velocity due to numerical dispersion. The first issue results in an effective spurious multipole field whose magnitude decreases very rapidly with the distance to the patch boundary, similarly to the spurious self-force in the electrostatic case. Hence, adding a few extra transition cells surrounding the patches mitigates this effect very effectively. The tunability of WarpX's electromagnetic finite-difference and pseudo-spectral solvers provides the means to optimize the numerical dispersion so as to minimize the second effect for a given application, which has been demonstrated on the laser-plasma interaction test case presented in (Vay, Adam, and Heron 2004). Both effects and their mitigation are described in more detail in (Vay, Adam, and Heron 2004).

Caustics are supported anywhere on the grid with an accuracy that is set by the local resolution, and will be adequately resolved if the grid resolution supports the necessary modes from their sources to the points of wavefront crossing. The mesh refinement method that is implemented in WarpX has the potential to provide higher efficiency than the standard use of fixed gridding, by offering a path toward adaptive gridding following wavefronts.

Berenger, Jp. 1996. "Three-Dimensional Perfectly Matched Layer for the Absorption of Electromagnetic Waves." *Journal of Computational Physics* 127 (2): 363–79.

Birdsall, C K, and A B Langdon. 1991. *Plasma Physics via Computer Simulation*. Adam-Hilger.

Colella, Phillip, and Peter C Norgaard. 2010. "Controlling Self-Force Errors at Refinement Boundaries for Amr-Pic." *Journal of Computational Physics* 229 (4): 947–57. https://doi.org/10.1016/J.Jcp.2009.07.004.

Mccorquodale, P, P Colella, Dp Grote, and Jl Vay. 2004. "A Node-Centered Local Refinement Algorithm For Poisson's Equation In Complex Geometries." *Journal of Computational Physics* 201 (1): 34–60. https://doi.org/10.1016/J.Jcp.2004.04.022.

Vay, J.-L. 2001. "An Extended Fdtd Scheme for the Wave Equation: Application to Multiscale Electromagnetic Simulation." *Journal of Computational Physics* 167 (1): 72–98.

———. 2002. "Asymmetric Perfectly Matched Layer for the Absorption of Waves." *Journal of Computational Physics* 183 (2): 367–99. https://doi.org/10.1006/Jcph.2002.7175.

Vay, J.-L., J.-C. Adam, and A Heron. 2004. "Asymmetric Pml for the Absorption of Waves. Application to Mesh Refinement in Electromagnetic Particle-in-Cell Plasma Simulations." *Computer Physics Communications* 164 (1-3): 171–77. https://doi.org/10.1016/J.Cpc.2004.06.026.

Vay, Jl, P Colella, P Mccorquodale, B Van Straalen, A Friedman, and Dp Grote. 2002. "Mesh Refinement for Particle-in-Cell Plasma Simulations: Applications to and Benefits for Heavy Ion Fusion." *Laser and Particle Beams* 20 (4): 569–75. https://doi.org/10.1017/S0263034602204139.

5.5 Boundary conditions

5.5.1 Open boundary condition for electromagnetic waves

For the TE case, the original Berenger's Perfectly Matched Layer (PML) writes

$$\begin{aligned}
\varepsilon_0\frac{\partial E_x}{\partial t}+\sigma_yE_x =& \frac{\partial H_z}{\partial y}\\
\varepsilon_0\frac{\partial E_y}{\partial t}+\sigma_xE_y =& -\frac{\partial H_z}{\partial x}\\
\mu_0\frac{\partial H_{zx}}{\partial t}+\sigma^*_xH_{zx} =& -\frac{\partial E_y}{\partial x}\\
\mu_0\frac{\partial H_{zy}}{\partial t}+\sigma^*_yH_{zy} =& \frac{\partial E_x}{\partial y}\\
H_z =& H_{zx}+H_{zy}
\end{aligned}$$

This can be generalized to

$$\begin{aligned}
\varepsilon_0\frac{\partial E_x}{\partial t}+\sigma_yE_x =& \frac{c_y}{c}\frac{\partial H_z}{\partial y}+\overline{\sigma}_yH_z\\
\varepsilon_0\frac{\partial E_y}{\partial t}+\sigma_xE_y =& -\frac{c_x}{c}\frac{\partial H_z}{\partial x}+\overline{\sigma}_xH_z\\
\mu_0\frac{\partial H_{zx}}{\partial t}+\sigma^*_xH_{zx} =& -\frac{c^*_x}{c}\frac{\partial E_y}{\partial x}+\overline{\sigma}^*_xE_y\\
\mu_0\frac{\partial H_{zy}}{\partial t}+\sigma^*_yH_{zy} =& \frac{c^*_y}{c}\frac{\partial E_x}{\partial y}+\overline{\sigma}^*_yE_x\\
H_z =& H_{zx}+H_{zy}
\end{aligned}$$

For $c_x=c_y=c^*_x=c^*_y=c$ and $\overline{\sigma}_x=\overline{\sigma}_y=\overline{\sigma}^*_x=\overline{\sigma}^*_y=0$, this system reduces to the Berenger PML medium, while adding the additional constraint $\sigma_x=\sigma_y=\sigma^*_x=\sigma^*_y=0$ leads to the system of Maxwell equations in vacuum.

[Sec:analytic theory, propa plane wave]Propagation of a Plane Wave in an APML Medium

We consider a plane wave of magnitude (E_0,H_{zx0},H_{zy0}) and pulsation ω propagating in the APML medium with an angle φ relative to the x axis

$$\begin{aligned}
E_x =& -E_0\sin\varphi e^{i\omega\left(t-\alpha x-\beta y\right)}\\
E_y =& E_0\cos\varphi e^{i\omega\left(t-\alpha x-\beta y\right)}\\
H_{zx} =& H_{zx0}e^{i\omega\left(t-\alpha x-\beta y\right)}\\
H_{zy} =& H_{zy0}e^{i\omega\left(t-\alpha x-\beta y\right)}
\end{aligned}$$

where α andβ are two complex constants to be determined.

Introducing (*[Plane_wave_APML_def_1]*), (*[Plane_wave_APML_def_2]*), (*[Plane_wave_AMPL_def_3]*) and (*[Plane_wave_APML_def_4]*) into (*[APML_def_1]*), (*[APML_def_2]*), (*[APML_def_3]*) and (*[APML_def_4]*) gives

$$\begin{aligned}
\varepsilon_0E_0\sin\varphi-i\frac{\sigma_y}{\omega}E_0\sin\varphi =& \beta\frac{c_y}{c}\left(H_{zx0}+H_{zy0}\right)+i\frac{\overline{\sigma}_y}{\omega}\left(H_{zx0}+H_{zy0}\right)\\
\varepsilon_0E_0\cos\varphi-i\frac{\sigma_x}{\omega}E_0\cos\varphi =& \alpha\frac{c_x}{c}\left(H_{zx0}+H_{zy0}\right)-i\frac{\overline{\sigma}_x}{\omega}\left(H_{zx0}+H_{zy0}\right)\\
\mu_0H_{zx0}-i\frac{\sigma^*_x}{\omega}H_{zx0} =& \alpha\frac{c^*_x}{c}E_0\cos\varphi-i\frac{\overline{\sigma}^*_x}{\omega}E_0\cos\varphi\\
\mu_0H_{zy0}-i\frac{\sigma^*_y}{\omega}H_{zy0} =& \beta\frac{c^*_y}{c}E_0\sin\varphi+i\frac{\overline{\sigma}^*_y}{\omega}E_0\sin\varphi
\end{aligned}$$

Defining $Z = E_0/\left(H_{zx0}+H_{zy0}\right)$ and using (*[Plane_wave_APML_1_1]*) and (*[Plane_wave_APML_1_2]*), we get

$$\beta = \left[Z\left(\varepsilon_0 - i\frac{\sigma_y}{\omega}\right)\sin\varphi - i\frac{\overline{\sigma}_y}{\omega}\right]\frac{c}{c_y}$$
$$\alpha = \left[Z\left(\varepsilon_0 - i\frac{\sigma_x}{\omega}\right)\cos\varphi + i\frac{\overline{\sigma}_x}{\omega}\right]\frac{c}{c_x}$$

Adding H_{zx0} and H_{zy0} from (*[Plane_wave_APML_1_3]*) and (*[Plane_wave_APML_1_4]*) and substituting the expressions for α and β from (*[Plane_wave_APML_beta_of_g]*) and (*[Plane_wave_APML_alpha_of_g]*) yields

$$\begin{aligned}\frac{1}{Z} =& \frac{Z\left(\varepsilon_0 - i\frac{\sigma_x}{\omega}\right)\cos\varphi\frac{c_x^*}{c_x} + i\frac{\overline{\sigma}_x}{\omega}\frac{c_x^*}{c_x} - i\frac{\overline{\sigma}_x^*}{\omega}}{\mu_0 - i\frac{\sigma_x^*}{\omega}}\cos\varphi \\ &+ \frac{Z\left(\varepsilon_0 - i\frac{\sigma_y}{\omega}\right)\sin\varphi\frac{c_y^*}{c_y} - i\frac{\overline{\sigma}_y}{\omega}\frac{c_y^*}{c_y} + i\frac{\overline{\sigma}_y^*}{\omega}}{\mu_0 - i\frac{\sigma_y^*}{\omega}}\sin\varphi\end{aligned}$$

If $c_x = c_x^*$, $c_y = c_y^*$, $\overline{\sigma}_x = \overline{\sigma}_x^*$, $\overline{\sigma}_y = \overline{\sigma}_y^*$, $\frac{\sigma_x}{\varepsilon_0} = \frac{\sigma_x^*}{\mu_0}$ and $\frac{\sigma_y}{\varepsilon_0} = \frac{\sigma_y^*}{\mu_0}$ then

$$Z = \pm\sqrt{\frac{\mu_0}{\varepsilon_0}}$$

which is the impedance of vacuum. Hence, like the PML, given some restrictions on the parameters, the APML does not generate any reflection at any angle and any frequency. As for the PML, this property is not retained after discretization, as shown subsequently in this paper.

Calling ψ any component of the field and ψ_0 its magnitude, we get from (*[Plane_wave_APML_def_1]*), (*[Plane_wave_APML_beta_of_g]*), (*[Plane_wave_APML_alpha_of_g]*) and (*[APML_impedance]*) that

$$\psi = \psi_0 e^{i\omega\left(t\mp x\cos\varphi/c_x\mp y\sin\varphi/c_y\right)}e^{-\left(\pm\frac{\sigma_x\cos\varphi}{\varepsilon_0 c_x}+\overline{\sigma}_x\frac{c}{c_x}\right)x}e^{-\left(\pm\frac{\sigma_y\sin\varphi}{\varepsilon_0 c_y}+\overline{\sigma}_y\frac{c}{c_y}\right)y}$$

We assume that we have an APML layer of thickness δ (measured along x) and that $\sigma_y = \overline{\sigma}_y = 0$ and $c_y = c$. Using (*[Plane_wave_absorption]*), we determine that the coefficient of reflection given by this layer is

$$\begin{aligned}R_{APML}\left(\theta\right) =& e^{-\left(\sigma_x\cos\varphi/\varepsilon_0 c_x+\overline{\sigma}_x c/c_x\right)\delta}e^{-\left(\sigma_x\cos\varphi/\varepsilon_0 c_x-\overline{\sigma}_x c/c_x\right)\delta} \\ =& e^{-2\left(\sigma_x\cos\varphi/\varepsilon_0 c_x\right)\delta}\end{aligned}$$

which happens to be the same as the PML theoretical coefficient of reflection if we assume $c_x = c$. Hence, it follows that for the purpose of wave absorption, the term $\overline{\sigma}_x$ seems to be of no interest. However, although this conclusion is true at the infinitesimal limit, it does not hold for the discretized counterpart.

Discretization

$$\begin{aligned}\frac{E_x|^{n+1}_{j+1/2,k,l} - E_x|^{n}_{j+1/2,k,l}}{\Delta t} + \sigma_y\frac{E_x|^{n+1}_{j+1/2,k,l} + E_x|^{n}_{j+1/2,k,l}}{2} =& \frac{H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j+1/2,k-1/2,l}}{\Delta y} \\ \frac{E_y|^{n+1}_{j,k+1/2,l} - E_y|^{n}_{j,k+1/2,l}}{\Delta t} + \sigma_x\frac{E_y|^{n+1}_{j,k+1/2,l} + E_y|^{n}_{j,k+1/2,l}}{2} =& -\frac{H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j-1/2,k+1/2,l}}{\Delta x} \\ \frac{H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} - H_{zx}|^{n}_{j+1/2,k+1/2,l}}{\Delta t} + \sigma_x^*\frac{H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} + H_{zx}|^{n}_{j+1/2,k+1/2,l}}{2} =& -\frac{E_y|^{n+1}_{j+1,k+1/2,l} - E_y|^{n+1}_{j,k+1/2,l}}{\Delta x} \\ \frac{H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} - H_{zy}|^{n}_{j+1/2,k+1/2,l}}{\Delta t} + \sigma_y^*\frac{H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} + H_{zy}|^{n}_{j+1/2,k+1/2,l}}{2} =& \frac{E_x|^{n+1}_{j+1/2,k+1,l} - E_x|^{n+1}_{j+1/2,k,l}}{\Delta y} \\ H_z =& H_{zx} + H_{zy}\end{aligned}$$

$$
\begin{aligned}
E_x|^{n+1}_{j+1/2,k,l} &= \left(\frac{1-\sigma_y\Delta t/2}{1+\sigma_y\Delta t/2}\right)E_x|^{n}_{j+1/2,k,l} + \frac{\Delta t/\Delta y}{1+\sigma_y\Delta t/2}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j+1/2,k-1/2,l}\right)\\
E_y|^{n+1}_{j,k+1/2,l} &= \left(\frac{1-\sigma_x\Delta t/2}{1+\sigma_x\Delta t/2}\right)E_y|^{n}_{j,k+1/2,l} - \frac{\Delta t/\Delta x}{1+\sigma_x\Delta t/2}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j-1/2,k+1/2,l}\right)\\
H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} &= \left(\frac{1-\sigma^*_x\Delta t/2}{1+\sigma^*_x\Delta t/2}\right)H_{zx}|^{n}_{j+1/2,k+1/2,l} - \frac{\Delta t/\Delta x}{1+\sigma^*_x\Delta t/2}\left(E_y|^{n+1}_{j+1,k+1/2,l} - E_y|^{n+1}_{j,k+1/2,l}\right)\\
H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} &= \left(\frac{1-\sigma^*_y\Delta t/2}{1+\sigma^*_y\Delta t/2}\right)H_{zy}|^{n}_{j+1/2,k+1/2,l} + \frac{\Delta t/\Delta y}{1+\sigma^*_y\Delta t/2}\left(E_x|^{n+1}_{j+1/2,k+1,l} - E_x|^{n+1}_{j+1/2,k,l}\right)\\
H_z &= H_{zx} + H_{zy}
\end{aligned}
$$

$$
\begin{aligned}
E_x|^{n+1}_{j+1/2,k,l} &= e^{-\sigma_y\Delta t}E_x|^{n}_{j+1/2,k,l} + \frac{1-e^{-\sigma_y\Delta t}}{\sigma_y\Delta y}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j+1/2,k-1/2,l}\right)\\
E_y|^{n+1}_{j,k+1/2,l} &= e^{-\sigma_x\Delta t}E_y|^{n}_{j,k+1/2,l} - \frac{1-e^{-\sigma_x\Delta t}}{\sigma_x\Delta x}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j-1/2,k+1/2,l}\right)\\
H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} &= e^{-\sigma^*_x\Delta t}H_{zx}|^{n}_{j+1/2,k+1/2,l} - \frac{1-e^{-\sigma^*_x\Delta t}}{\sigma^*_x\Delta x}\left(E_y|^{n+1}_{j+1,k+1/2,l} - E_y|^{n+1}_{j,k+1/2,l}\right)\\
H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} &= e^{-\sigma^*_y\Delta t}H_{zy}|^{n}_{j+1/2,k+1/2,l} + \frac{1-e^{-\sigma^*_y\Delta t}}{\sigma^*_y\Delta y}\left(E_x|^{n+1}_{j+1/2,k+1,l} - E_x|^{n+1}_{j+1/2,k,l}\right)\\
H_z &= H_{zx} + H_{zy}
\end{aligned}
$$

$$
\begin{aligned}
E_x|^{n+1}_{j+1/2,k,l} &= e^{-\sigma_y\Delta t}E_x|^{n}_{j+1/2,k,l} + \frac{1-e^{-\sigma_y\Delta t}}{\sigma_y\Delta y}\frac{c_y}{c}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j+1/2,k-1/2,l}\right)\\
E_y|^{n+1}_{j,k+1/2,l} &= e^{-\sigma_x\Delta t}E_y|^{n}_{j,k+1/2,l} - \frac{1-e^{-\sigma_x\Delta t}}{\sigma_x\Delta x}\frac{c_x}{c}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j-1/2,k+1/2,l}\right)\\
H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} &= e^{-\sigma^*_x\Delta t}H_{zx}|^{n}_{j+1/2,k+1/2,l} - \frac{1-e^{-\sigma^*_x\Delta t}}{\sigma^*_x\Delta x}\frac{c^*_x}{c}\left(E_y|^{n+1}_{j+1,k+1/2,l} - E_y|^{n+1}_{j,k+1/2,l}\right)\\
H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} &= e^{-\sigma^*_y\Delta t}H_{zy}|^{n}_{j+1/2,k+1/2,l} + \frac{1-e^{-\sigma^*_y\Delta t}}{\sigma^*_y\Delta y}\frac{c^*_y}{c}\left(E_x|^{n+1}_{j+1/2,k+1,l} - E_x|^{n+1}_{j+1/2,k,l}\right)\\
H_z &= H_{zx} + H_{zy}
\end{aligned}
$$

$$
\begin{aligned}
c_x &= ce^{-\sigma_x\Delta t}\frac{\sigma_x\Delta x}{1-e^{-\sigma_x\Delta t}}\\
c_y &= ce^{-\sigma_y\Delta t}\frac{\sigma_y\Delta y}{1-e^{-\sigma_y\Delta t}}\\
c^*_x &= ce^{-\sigma^*_x\Delta t}\frac{\sigma^*_x\Delta x}{1-e^{-\sigma^*_x\Delta t}}\\
c^*_y &= ce^{-\sigma^*_y\Delta t}\frac{\sigma^*_y\Delta y}{1-e^{-\sigma^*_y\Delta t}}
\end{aligned}
$$

$$
\begin{aligned}
E_x|^{n+1}_{j+1/2,k,l} &= e^{-\sigma_y\Delta t}\left[E_x|^{n}_{j+1/2,k,l} + \frac{\Delta t}{\Delta y}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j+1/2,k-1/2,l}\right)\right]\\
E_y|^{n+1}_{j,k+1/2,l} &= e^{-\sigma_x\Delta t}\left[E_y|^{n}_{j,k+1/2,l} - \frac{\Delta t}{\Delta x}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j-1/2,k+1/2,l}\right)\right]\\
H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} &= e^{-\sigma^*_x\Delta t}\left[H_{zx}|^{n}_{j+1/2,k+1/2,l} - \frac{\Delta t}{\Delta x}\left(E_y|^{n+1}_{j+1,k+1/2,l} - E_y|^{n+1}_{j,k+1/2,l}\right)\right]\\
H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} &= e^{-\sigma^*_y\Delta t}\left[H_{zy}|^{n}_{j+1/2,k+1/2,l} + \frac{\Delta t}{\Delta y}\left(E_x|^{n+1}_{j+1/2,k+1,l} - E_x|^{n+1}_{j+1/2,k,l}\right)\right]\\
H_z &= H_{zx} + H_{zy}
\end{aligned}
$$

$$
\begin{aligned}
E_x|^{n+1}_{j+1/2,k,l} =& E_x|^{n}_{j+1/2,k,l} + \frac{\Delta t}{\Delta y}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j+1/2,k-1/2,l}\right) \\
E_y|^{n+1}_{j,k+1/2,l} =& E_y|^{n}_{j,k+1/2,l} - \frac{\Delta t}{\Delta x}\left(H_z|^{n+1/2}_{j+1/2,k+1/2,l} - H_z|^{n+1/2}_{j-1/2,k+1/2,l}\right) \\
H_{zx}|^{n+3/2}_{j+1/2,k+1/2,l} =& H_{zx}|^{n}_{j+1/2,k+1/2,l} - \frac{\Delta t}{\Delta x}\left(E_y|^{n+1}_{j+1,k+1/2,l} - E_y|^{n+1}_{j,k+1/2,l}\right) \\
H_{zy}|^{n+3/2}_{j+1/2,k+1/2,l} =& H_{zy}|^{n}_{j+1/2,k+1/2,l} + \frac{\Delta t}{\Delta y}\left(E_x|^{n+1}_{j+1/2,k+1,l} - E_x|^{n+1}_{j+1/2,k,l}\right) \\
H_z =& H_{zx} + H_{zy}
\end{aligned}
$$

5.6 Moving window and optimal Lorentz boosted frame

The simulations of plasma accelerators from first principles are extremely computationally intensive, due to the need to resolve the evolution of a driver (laser or particle beam) and an accelerated particle beam into a plasma structure that is orders of magnitude longer and wider than the accelerated beam. As is customary in the modeling of particle beam dynamics in standard particle accelerators, a moving window is commonly used to follow the driver, the wake and the accelerated beam. This results in huge savings, by avoiding the meshing of the entire plasma that is orders of magnitude longer than the other length scales of interest.

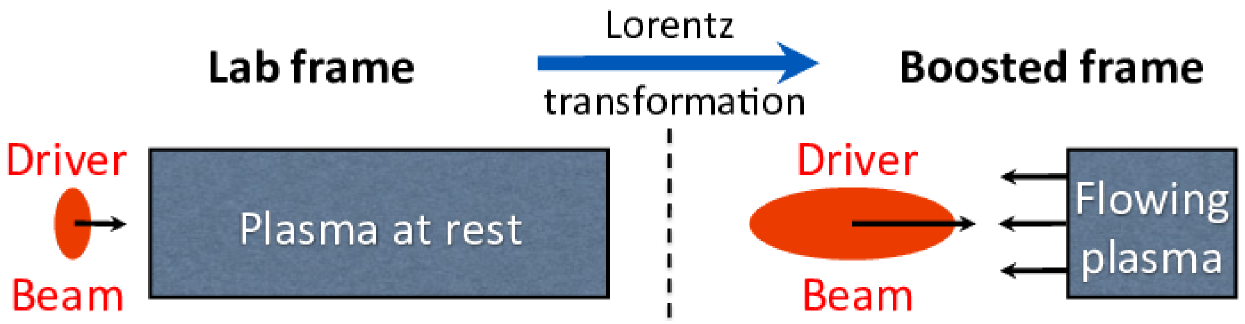


Fig. 5.7: [fig:PIC] A first principle simulation of a short driver beam (laser or charged particles) propagating through a plasma that is orders of magnitude longer necessitates a very large number of time steps. Recasting the simulation in a frame of reference that is moving close to the speed of light in the direction of the driver beam leads to simulating a driver beam that appears longer propagating through a plasma that appears shorter than in the laboratory. Thus, this relativistic transformation of space and time reduces the disparity of scales, and thereby the number of time steps to complete the simulation, by orders of magnitude.

Even using a moving window, however, a full PIC simulation of a plasma accelerator can be extraordinarily demanding computationally, as many time steps are needed to resolve the crossing of the short driver beam with the plasma column. As it turns out, choosing an optimal frame of reference that travels close to the speed of light in the direction of the laser or particle beam (as opposed to the usual choice of the laboratory frame) enables speedups by orders of magnitude (Vay 2007; J -L. Vay et al. 2011). This is a result of the properties of Lorentz contraction and dilation of space and time. In the frame of the laboratory, a very short driver (laser or particle) beam propagates through a much longer plasma column, necessitating millions to tens of millions of time steps for parameters in the range of the BELLA or FACET-II experiments. As sketched in Fig. *[fig:PIC]*, in a frame moving with the driver beam in the plasma at velocity $v = \beta c$ (where c is the speed of light in vacuum), the beam length is now elongated by $\approx (1+\beta)\gamma$ while the plasma contracts by γ (where $\gamma = 1/\sqrt{1-\beta^2}$ is the relativistic factor associated with the frame velocity). The number of time steps that is needed to simulate a "longer" beam through a "shorter" plasma is now reduced by up to $\approx (1+\beta)\gamma^2$ (a detailed derivation of the speedup is given below).

The modeling of a plasma acceleration stage in a boosted frame involves the fully electromagnetic modeling of a plasma propagating at near the speed of light, for which Numerical Cerenkov (Boris and Lee 1973; Haber et al. 1973) is a potential issue, as explained in more details below. In addition, for a frame of reference moving in the direction of the accelerated beam (or equivalently the wake of the laser), waves emitted by the plasma in the forward direction expand while the ones emitted in the backward direction contract, following the properties of the Lorentz transformation. If one had to resolve both forward and backward propagating waves emitted from the plasma, there would be no gain in selecting a frame different from the laboratory frame. However, the physics of interest for a laser wakefield is the laser driving the wake, the wake, and the accelerated beam. Backscatter is weak in the short-pulse regime, and does not interact as strongly with the beam as do the forward propagating waves which stay in phase for a long period. It is thus often assumed that the backward propagating waves can be neglected in the modeling of plasma accelerator stages. The accuracy of this assumption has been demonstrated by comparison between explicit codes which include both forward and backward waves and envelope or quasistatic codes which neglect backward waves (Geddes et al. 2008; Geddes et al. 2009; Cowan et al. 2009).

5.6.1 Theoretical speedup dependency with the frame boost

The derivation that is given here reproduces the one given in (J -L. Vay et al. 2011), where the obtainable speedup is derived as an extension of the formula that was derived earlier(Vay 2007), taking in addition into account the group velocity of the laser as it traverses the plasma.

Assuming that the simulation box is a fixed number of plasma periods long, which implies the use (which is standard) of a moving window following the wake and accelerated beam, the speedup is given by the ratio of the time taken by the laser pulse and the plasma to cross each other, divided by the shortest time scale of interest, that is the laser period. To first order, the wake velocity v_w is set by the 1D group velocity of the laser driver, which in the linear (low intensity) limit, is given by (Esarey, Schroeder, and Leemans 2009):

$$v_w/c = \beta_w = \left(1 - \frac{\omega_p^2}{\omega^2}\right)^{1/2}$$

where $\omega_p = \sqrt{(n_e e^2)/(\epsilon_0 m_e)}$ is the plasma frequency, $\omega = 2\pi c/\lambda$ is the laser frequency, n_e is the plasma density, λ is the laser wavelength in vacuum, ϵ_0 is the permittivity of vacuum, c is the speed of light in vacuum, and e and m_e are respectively the charge and mass of the electron.

In practice, the runs are typically stopped when the last electron beam macro-particle exits the plasma, and a measure of the total time of the simulation is then given by

$$T = \frac{L + \eta\lambda_p}{v_w - v_p}$$

where $\lambda_p \approx 2\pi c/\omega_p$ is the wake wavelength, L is the plasma length, v_w and $v_p = \beta_p c$ are respectively the velocity of the wake and of the plasma relative to the frame of reference, and η is an adjustable parameter for taking into account the fraction of the wake which exited the plasma at the end of the simulation. For a beam injected into the n^{th} bucket, η would be set to $n - 1/2$. If positrons were considered, they would be injected half a wake period ahead of the location of the electrons injection position for a given period, and one would have $\eta = n - 1$. The numerical cost R_t scales as the ratio of the total time to the shortest timescale of interest, which is the inverse of the laser frequency, and is thus given by

$$R_t = \frac{Tc}{\lambda} = \frac{(L + \eta\lambda_p)}{(\beta_w - \beta_p)\,\lambda}$$

In the laboratory, $v_p = 0$ and the expression simplifies to

$$R_{lab} = \frac{Tc}{\lambda} = \frac{(L + \eta\lambda_p)}{\beta_w \lambda}$$

In a frame moving at βc, the quantities become

$$\begin{aligned}
\lambda_p^* &= \lambda_p / \left[\gamma\left(1-\beta_w\beta\right)\right] \\
L^* &= L/\gamma \\
\lambda^* &= \gamma\left(1+\beta\right)\lambda \\
\beta_w^* &= \left(\beta_w-\beta\right)/\left(1-\beta_w\beta\right) \\
v_p^* &= -\beta c \\
T^* &= \frac{L^*+\eta\lambda_p^*}{v_w^*-v_p^*} \\
R_t^* &= \frac{T^* c}{\lambda^*} = \frac{\left(L^*+\eta\lambda_p^*\right)}{\left(\beta_w^*+\beta\right)\lambda^*}
\end{aligned}$$

where $\gamma = 1/\sqrt{1-\beta^2}$.

The expected speedup from performing the simulation in a boosted frame is given by the ratio of R_{lab} and R_t^*

$$S = \frac{R_{lab}}{R_t^*} = \frac{\left(1+\beta\right)\left(L+\eta\lambda_p\right)}{\left(1-\beta\beta_w\right)L+\eta\lambda_p}$$

We note that assuming that $\beta_w \approx 1$ (which is a valid approximation for most practical cases of interest) and that $\gamma << \gamma_w$, this expression is consistent with the expression derived earlier (Vay 2007) for the laser-plasma acceleration case, which states that $R_t^* = \alpha R_t/\left(1+\beta\right)$ with $\alpha = \left(1-\beta+l/L\right)/\left(1+l/L\right)$, where l is the laser length which is generally proportional to $\eta\lambda_p$, and $S = R_t/R_T^*$. However, higher values of γ are of interest for maximum speedup, as shown below.

For intense lasers ($a \sim 1$) typically used for acceleration, the energy gain is limited by dephasing (Schroeder et al. 2011), which occurs over a scale length $L_d \sim \lambda_p^3/2\lambda^2$. Acceleration is compromised beyond L_d and in practice, the plasma length is proportional to the dephasing length, i.e. $L = \xi L_d$. In most cases, $\gamma_w^2 >> 1$, which allows the approximations $\beta_w \approx 1 - \lambda^2/2\lambda_p^2$, and $L = \xi\lambda_p^3/2\lambda^2 \approx \xi\gamma_w^2\lambda_p/2 >> \eta\lambda_p$, so that Eq.(*[Eq_scaling1d0]*) becomes

$$S = \left(1+\beta\right)^2\gamma^2\frac{\xi\gamma_w^2}{\xi\gamma_w^2+\left(1+\beta\right)\gamma^2\left(\xi\beta/2+2\eta\right)}$$

For low values of γ, i.e. when $\gamma << \gamma_w$, Eq.(*[Eq_scaling1d]*) reduces to

$$S_{\gamma<<\gamma_w} = \left(1+\beta\right)^2\gamma^2$$

Conversely, if $\gamma \rightarrow \infty$, Eq.(*[Eq_scaling1d]*) becomes

$$S_{\gamma\rightarrow\infty} = \frac{4}{1+4\eta/\xi}\gamma_w^2$$

Finally, in the frame of the wake, i.e. when $\gamma = \gamma_w$, assuming that $\beta_w \approx 1$, Eq.(*[Eq_scaling1d]*) gives

$$S_{\gamma=\gamma_w} \approx \frac{2}{1+2\eta/\xi}\gamma_w^2$$

Since η and ξ are of order unity, and the practical regimes of most interest satisfy $\gamma_w^2 >> 1$, the speedup that is obtained by using the frame of the wake will be near the maximum obtainable value given by Eq.(*[Eq_scaling_gamma_inf]*).

Note that without the use of a moving window, the relativistic effects that are at play in the time domain would also be at play in the spatial domain (Vay 2007), and the γ^2 scaling would transform to γ^4. Hence, it is important to use a moving window even in simulations in a Lorentz boosted frame. For very high values of the boosted frame, the optimal velocity of the moving window may vanish (i.e. no moving window) or even reverse.

5.6.2 Numerical Stability and alternate formulation in a Galilean frame

The numerical Cherenkov instability (NCI) (Godfrey 1974) is the most serious numerical instability affecting multidimensional PIC simulations of relativistic particle beams and streaming plasmas (Martins et al. 2010; Vay et al. 2010; J L Vay et al. 2011; Sironi and Spitkovsky 2011; Godfrey and Vay 2013; Xu et al. 2013). It arises from coupling between possibly numerically distorted electromagnetic modes and spurious beam modes, the latter due to the mismatch between the Lagrangian treatment of particles and the Eulerian treatment of fields (Godfrey 1975).

In recent papers the electromagnetic dispersion relations for the numerical Cherenkov instability were derived and solved for both FDTD (Godfrey and Vay 2013; Brendan B. Godfrey and Vay 2014) and PSATD (Brendan B. Godfrey, Vay, and Haber 2014a, 2014b) algorithms.

Several solutions have been proposed to mitigate the NCI (Brendan B Godfrey, Vay, and Haber 2014; Brendan B. Godfrey, Vay, and Haber 2014b, 2014a; Godfrey and Vay 2015; Yu, Xu, Decyk, et al. 2015; Yu, Xu, Tableman, et al. 2015). Although these solutions efficiently reduce the numerical instability, they typically introduce either strong smoothing of the currents and fields, or arbitrary numerical corrections, which are tuned specifically against the NCI and go beyond the natural discretization of the underlying physical equation. Therefore, it is sometimes unclear to what extent these added corrections could impact the physics at stake for a given resolution.

For instance, NCI-specific corrections include periodically smoothing the electromagnetic field components (Martins et al. 2010), using a special time step (Vay et al. 2010; J L Vay et al. 2011) or applying a wide-band smoothing of the current components (Vay et al. 2010; J L Vay et al. 2011; J. Vay et al. 2011). Another set of mitigation methods involve scaling the deposited currents by a carefully-designed wavenumber-dependent factor (Brendan B. Godfrey and Vay 2014; Brendan B. Godfrey, Vay, and Haber 2014b) or slightly modifying the ratio of electric and magnetic fields (E/B) before gathering their value onto the macroparticles (Brendan B. Godfrey, Vay, and Haber 2014a; Godfrey and Vay 2015). Yet another set of NCI-specific corrections (Yu, Xu, Decyk, et al. 2015; Yu, Xu, Tableman, et al. 2015) consists in combining a small timestep Δt, a sharp low-pass spatial filter, and a spectral or high-order scheme that is tuned so as to create a small, artificial "bump" in the dispersion relation (Yu, Xu, Decyk, et al. 2015). While most mitigation methods have only been applied to Cartesian geometry, this last set of methods ((Yu, Xu, Decyk, et al. 2015; Yu, Xu, Tableman, et al. 2015)) has the remarkable property that it can be applied (Yu, Xu, Tableman, et al. 2015) to both Cartesian geometry and quasi-cylindrical geometry (i.e. cylindrical geometry with azimuthal Fourier decomposition (Lifschitz et al. 2009; Davidson et al. 2015; R. Lehe et al. 2016)). However, the use of a small timestep proportionally slows down the progress of the simulation, and the artificial "bump" is again an arbitrary correction that departs from the underlying physics.

A new scheme was recently proposed, in (Kirchen et al. 2016; Lehe et al. 2016), which completely eliminates the NCI for a plasma drifting at a uniform relativistic velocity – with no arbitrary correction – by simply integrating the PIC equations in *Galilean coordinates* (also known as *comoving coordinates*). More precisely, in the new method, the Maxwell equations *in Galilean coordinates* are integrated analytically, using only natural hypotheses, within the PSATD framework (Pseudo-Spectral-Analytical-Time-Domain (Haber et al. 1973; Vay, Haber, and Godfrey 2013)).

The idea of the proposed scheme is to perform a Galilean change of coordinates, and to carry out the simulation in the new coordinates:

$$\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{v}_{gal} t$$

where $\boldsymbol{x} = x\,\boldsymbol{u}_x + y\,\boldsymbol{u}_y + z\,\boldsymbol{u}_z$ and $\boldsymbol{x}' = x'\,\boldsymbol{u}_x + y'\,\boldsymbol{u}_y + z'\,\boldsymbol{u}_z$ are the position vectors in the standard and Galilean coordinates respectively.

When choosing $\boldsymbol{v}_{gal} = \boldsymbol{v}_0$, where $\boldsymbol{v}_0$ is the speed of the bulk of the relativistic plasma, the plasma does not move with respect to the grid in the Galilean coordinates $\boldsymbol{x}'$ – or, equivalently, in the standard coordinates $\boldsymbol{x}$, the grid moves along with the plasma. The heuristic intuition behind this scheme is that these coordinates should prevent the discrepancy between the Lagrangian and Eulerian point of view, which gives rise to the NCI (Godfrey 1975).

An important remark is that the Galilean change of coordinates (*[eq:change-var]*) is a simple translation. Thus, when used in the context of Lorentz-boosted simulations, it does of course preserve the relativistic dilatation of space and time which gives rise to the characteristic computational speedup of the boosted-frame technique.

Another important remark is that the Galilean scheme is *not* equivalent to a moving window (and in fact the Galilean scheme can be independently *combined* with a moving window). Whereas in a moving window, gridpoints are added and removed so as to effectively translate the boundaries, in the Galilean scheme the gridpoints *themselves* are not only translated but in this case, the physical equations are modified accordingly. Most importantly, the assumed time evolution of the current $\boldsymbol{J}$ within one timestep is different in a standard PSATD scheme with moving window and in a Galilean PSATD scheme (Lehe et al. 2016).

In the Galilean coordinates $\boldsymbol{x}'$, the equations of particle motion and the Maxwell equations take the form

$$\begin{aligned}
\frac{d\boldsymbol{x}'}{dt} &= \frac{\boldsymbol{p}}{\gamma m} - \boldsymbol{v}_{gal} \\
\frac{d\boldsymbol{p}}{dt} &= q\left(\boldsymbol{E} + \frac{\boldsymbol{p}}{\gamma m} \times \boldsymbol{B}\right) \\
\left(\frac{\partial}{\partial t} - \boldsymbol{v}_{gal}\cdot\boldsymbol{\nabla'}\right)\boldsymbol{B} &= -\boldsymbol{\nabla'} \times \boldsymbol{E} \\
\frac{1}{c^2}\left(\frac{\partial}{\partial t} - \boldsymbol{v}_{gal}\cdot\boldsymbol{\nabla'}\right)\boldsymbol{E} &= \boldsymbol{\nabla'} \times \boldsymbol{B} - \mu_0\boldsymbol{J}
\end{aligned}$$

where $\boldsymbol{\nabla'}$ denotes a spatial derivative with respect to the Galilean coordinates $\boldsymbol{x}'$.

Integrating these equations from $t = n\Delta t$ to $t = (n+1)\Delta t$ results in the following update equations (see (Lehe et al. 2016) for the details of the derivation):

$$\begin{aligned}
\mathbf{\tilde{B}}^{n+1} &= \theta^2 C\mathbf{\tilde{B}}^n - \frac{\theta^2 S}{ck} i\boldsymbol{k} \times \mathbf{\tilde{E}}^n \\
&+ \frac{\theta\chi_1}{\epsilon_0 c^2 k^2}\, i\boldsymbol{k} \times \mathbf{\tilde{J}}^{n+1/2} \\
\mathbf{\tilde{E}}^{n+1} &= \theta^2 C\mathbf{\tilde{E}}^n + \frac{\theta^2 S}{k}\, ci\boldsymbol{k} \times \mathbf{\tilde{B}}^n \\
&+ \frac{i\nu\theta\chi_1 - \theta^2 S}{\epsilon_0 ck}\,\mathbf{\tilde{J}}^{n+1/2} \\
&- \frac{1}{\epsilon_0 k^2}\left(\,\chi_2\,\hat{\rho}^{n+1} - \theta^2\chi_3\,\hat{\rho}^n\,\right) i\boldsymbol{k}
\end{aligned}$$

where we used the short-hand notations $\mathbf{\tilde{E}}^n \equiv \mathbf{\tilde{E}}(\boldsymbol{k}, n\Delta t)$, $\mathbf{\tilde{B}}^n \equiv \mathbf{\tilde{B}}(\boldsymbol{k}, n\Delta t)$ as well as:

$$\begin{aligned}
&C = \cos(ck\Delta t) \quad S = \sin(ck\Delta t) \quad k = |\boldsymbol{k}| \\
&\nu = \frac{\boldsymbol{k}\cdot\boldsymbol{v}_{gal}}{ck} \quad \theta = e^{i\boldsymbol{k}\cdot\boldsymbol{v}_{gal}\Delta t/2} \quad \theta^* = e^{-i\boldsymbol{k}\cdot\boldsymbol{v}_{gal}\Delta t/2} \\
&\chi_1 = \frac{1}{1-\nu^2}\left(\theta^* - C\theta + i\nu\theta S\right) \\
&\chi_2 = \frac{\chi_1 - \theta(1-C)}{\theta^* - \theta} \quad \chi_3 = \frac{\chi_1 - \theta^*(1-C)}{\theta^* - \theta}
\end{aligned}$$

Note that, in the limit $\boldsymbol{v}_{gal} = \boldsymbol{0}$, (*[eq:disc-maxwell1]*) and (*[eq:disc-maxwell2]*) reduce to the standard PSATD equations (Haber et al. 1973), as expected. As shown in (Kirchen et al. 2016; Lehe et al. 2016), the elimination of the NCI with the new Galilean integration is verified empirically via PIC simulations of uniform drifting plasmas and laser-driven plasma acceleration stages, and confirmed by a theoretical analysis of the instability.

Boris, Jp, and R Lee. 1973. "Nonphysical Self Forces in Some Electromagnetic Plasma-Simulation Algorithms." Note. *Journal of Computational Physics* 12 (1). 525 B St, Ste 1900, San Diego, Ca 92101-4495: Academic Press Inc Jnl-Comp Subscriptions: 131–36.

Cowan, B, D Bruhwiler, E Cormier-Michel, E Esarey, C G R Geddes, P Messmer, and K Paul. 2009. "Laser Wakefield Simulation Using A Speed-of-Light Frame Envelope Model." In *Aip Conference Proceedings*, 1086:309–14.

Davidson, A., A. Tableman, W. An, F.S. Tsung, W. Lu, J. Vieira, R.A. Fonseca, L.O. Silva, and W.B. Mori. 2015. “Implementation of a hybrid particle code with a PIC description in r–z and a gridless description in into OSIRIS.” *Journal of Computational Physics* 281: 1063–77. https://doi.org/10.1016/j.jcp.2014.10.064.

Esarey, E, C B Schroeder, and W P Leemans. 2009. “Physics of Laser-Driven Plasma-Based Electron Accelerators.” *Rev. Mod. Phys.* 81 (3): 1229–85. https://doi.org/10.1103/Revmodphys.81.1229.

Geddes, C G R, D L Bruhwiler, J R Cary, W B Mori, J.-L. Vay, S F Martins, T Katsouleas, et al. 2008. “Computational Studies and Optimization of Wakefield Accelerators.” In *Journal of Physics: Conference Series*, 125:012002 (11 Pp.).

Geddes et al., C G R. 2009. “Scaled Simulation Design of High Quality Laser Wakefield Accelerator Stages.” In *Proc. Particle Accelerator Conference*. Vancouver, Canada.

Godfrey, Bb. 1974. “Numerical Cherenkov Instabilities in Electromagnetic Particle Codes.” *Journal of Computational Physics* 15 (4): 504–21.

———. 1975. “Canonical Momenta and Numerical Instabilities in Particle Codes.” *Journal of Computational Physics* 19 (1): 58–76.

Godfrey, Brendan B, and Jean-Luc Vay. 2013. “Numerical stability of relativistic beam multidimensional {PIC} simulations employing the Esirkepov algorithm.” *Journal of Computational Physics* 248 (0): 33–46. https://doi.org/http://dx.doi.org/10.1016/j.jcp.2013.04.006.

Godfrey, Brendan B., and Jean Luc Vay. 2014. “Suppressing the numerical Cherenkov instability in FDTD PIC codes.” *Journal of Computational Physics* 267: 1–6.

———. 2015. “Improved numerical Cherenkov instability suppression in the generalized PSTD PIC algorithm.” *Computer Physics Communications* 196. Elsevier: 221–25.

Godfrey, Brendan B., Jean Luc Vay, and Irving Haber. 2014a. “Numerical stability analysis of the pseudo-spectral analytical time-domain PIC algorithm.” *Journal of Computational Physics* 258: 689–704.

———. 2014b. “Numerical stability improvements for the pseudospectral EM PIC algorithm.” *IEEE Transactions on Plasma Science* 42 (5). Institute of Electrical; Electronics Engineers Inc.: 1339–44.

Godfrey, Brendan B, Jean-Luc Vay, and Irving Haber. 2014. “Numerical stability analysis of the pseudo-spectral analytical time-domain {PIC} algorithm.” *Journal of Computational Physics* 258 (0): 689–704. https://doi.org/http://dx.doi.org/10.1016/j.jcp.2013.10.053.

Haber, I, R Lee, Hh Klein, and Jp Boris. 1973. “Advances in Electromagnetic Simulation Techniques.” In *Proc. Sixth Conf. Num. Sim. Plasmas*, 46–48. Berkeley, Ca.

Kirchen, M., R. Lehe, B. B. Godfrey, I. Dornmair, S. Jalas, K. Peters, J.-L. Vay, and A. R. Maier. 2016. “Stable discrete representation of relativistically drifting plasmas.” *arXiv:1608.00215*.

Lehe, Rémi, Manuel Kirchen, Igor A. Andriyash, Brendan B. Godfrey, and Jean-Luc Vay. 2016. “A spectral, quasi-cylindrical and dispersion-free Particle-In-Cell algorithm.” *Computer Physics Communications* 203: 66–82. https://doi.org/10.1016/j.cpc.2016.02.007.

Lehe, R., M. Kirchen, B. B. Godfrey, A. R. Maier, and J.-L. Vay. 2016. “Elimination of Numerical Cherenkov Instability in flowing-plasma Particle-In-Cell simulations by using Galilean coordinates.” *arXiv:1608.00227*.

Lifschitz, A F, X Davoine, E Lefebvre, J Faure, C Rechatin, and V Malka. 2009. “Particle-in-Cell modelling of laser{â}plasma interaction using Fourier decomposition.” *Journal of Computational Physics* 228 (5): 1803–14. https://doi.org/http://dx.doi.org/10.1016/j.jcp.2008.11.017.

Martins, Samuel F, Ricardo A Fonseca, Luis O Silva, Wei Lu, and Warren B Mori. 2010. “Numerical Simulations of Laser Wakefield Accelerators in Optimal Lorentz Frames.” *Computer Physics Communications* 181 (5): 869–75. https://doi.org/10.1016/J.Cpc.2009.12.023.

Schroeder, C B, C Benedetti, E Esarey, and W P Leemans. 2011. “Nonlinear Pulse Propagation and Phase Velocity of Laser-Driven Plasma Waves.” *Physical Review Letters* 106 (13): 135002. https://doi.org/10.1103/Physrevlett.106.135002.

Sironi, L, and A Spitkovsky. 2011. “No Title.”

Vay, Jean Luc, Irving Haber, and Brendan B. Godfrey. 2013. “A domain decomposition method for pseudo-spectral electromagnetic simulations of plasmas.” *Journal of Computational Physics* 243: 260–68.

Vay, J.-L. 2007. “Noninvariance of Space- and Time-Scale Ranges Under A Lorentz Transformation and the Implications for the Study of Relativistic Interactions.” *Physical Review Letters* 98 (13): 130405/1–4.

Vay, J -. L, C G R Geddes, C Benedetti, D L Bruhwiler, E Cormier-Michel, B M Cowan, J R Cary, and D P Grote. 2010. “Modeling Laser Wakefield Accelerators in A Lorentz Boosted Frame.” *Aip Conference Proceedings* 1299: 244–49. https://doi.org/10.1063/1.3520322.

Vay, J L, C G R Geddes, E Cormier-Michel, and D P Grote. 2011. “Numerical Methods for Instability Mitigation in the Modeling of Laser Wakefield Accelerators in A Lorentz-Boosted Frame.” *Journal of Computational Physics* 230 (15): 5908–29. https://doi.org/10.1016/J.Jcp.2011.04.003.

Vay, Jl, C G R Geddes, E Cormier-Michel, and D P Grote. 2011. “Effects of Hyperbolic Rotation in Minkowski Space on the Modeling of Plasma Accelerators in A Lorentz Boosted Frame.” *Physics of Plasmas* 18 (3): 30701. https://doi.org/10.1063/1.3559483.

Vay, J -L., C G R Geddes, E Esarey, C B Schroeder, W P Leemans, E Cormier-Michel, and D P Grote. 2011. “Modeling of 10 Gev-1 Tev Laser-Plasma Accelerators Using Lorentz Boosted Simulations.” *Physics of Plasmas* 18 (12). https://doi.org/10.1063/1.3663841.

Xu, Xinlu, Peicheng Yu, Samual F Martins, Frank S Tsung, Viktor K Decyk, Jorge Vieira, Ricardo A Fonseca, Wei Lu, Luis O Silva, and Warren B Mori. 2013. “Numerical instability due to relativistic plasma drift in EM-PIC simulations.” *Computer Physics Communications* 184 (11): 2503–14. https://doi.org/http://dx.doi.org/10.1016/j.cpc.2013.07.003.

Yu, Peicheng, Xinlu Xu, Viktor K. Decyk, Frederico Fiuza, Jorge Vieira, Frank S. Tsung, Ricardo A. Fonseca, Wei Lu, Luis O. Silva, and Warren B. Mori. 2015. “Elimination of the numerical Cerenkov instability for spectral EM-PIC codes.” *Computer Physics Communications* 192 (July). ELSEVIER SCIENCE BV, PO BOX 211, 1000 AE AMSTERDAM, NETHERLANDS: 32–47. https://doi.org/10.1016/j.cpc.2015.02.018.

Yu, Peicheng, Xinlu Xu, Adam Tableman, Viktor K. Decyk, Frank S. Tsung, Frederico Fiuza, Asher Davidson, et al. 2015. “Mitigation of numerical Cerenkov radiation and instability using a hybrid finite difference-FFT Maxwell solver and a local charge conserving current deposit.” *Computer Physics Communications* 197 (December). ELSEVIER SCIENCE BV, PO BOX 211, 1000 AE AMSTERDAM, NETHERLANDS: 144–52. https://doi.org/10.1016/j.cpc.2015.08.026.

5.7 Inputs and outputs

Initialization of the plasma columns and drivers (laser or particle beam) is performed via the specification of multidimensional functions that describe the initial state with, if needed, a time dependence, or from reconstruction of distributions based on experimental data. Care is needed when initializing quantities in parallel to avoid double counting and ensure smoothness of the distributions at the interface of computational domains. When the sum of the initial distributions of charged particles is not charge neutral, initial fields are computed using generally a static approximation with Poisson solves accompanied by proper relativistic scalings (Vay 2008; Cowan et al. 2013).

Outputs include dumps of particle and field quantities at regular intervals, histories of particle distributions moments, spectra, etc, and plots of the various quantities. In parallel simulations, the diagnostic subroutines need to handle additional complexity from the domain decomposition, as well as large amount of data that may necessitate data reduction in some form before saving to disk.

Simulations in a Lorentz boosted frame require additional considerations, as described below.

5.7.1 Inputs and outputs in a boosted frame simulation

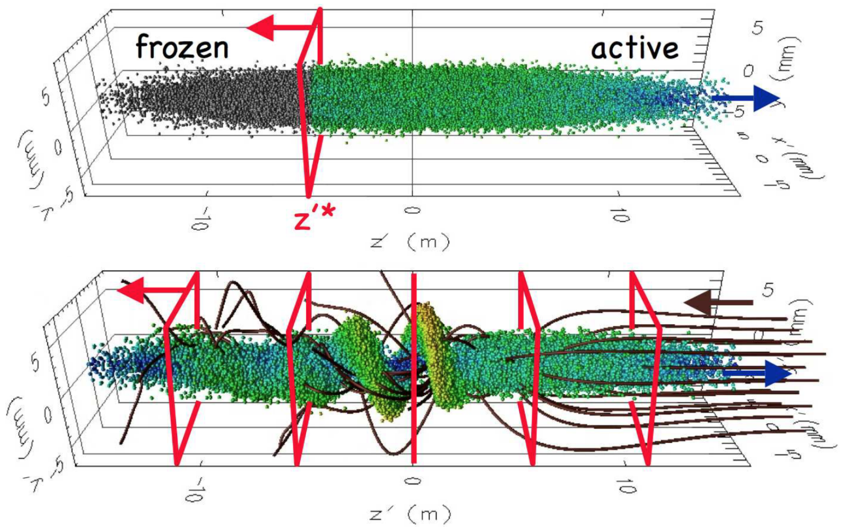


Fig. 5.8: (top) Snapshot of a particle beam showing “frozen” (grey spheres) and “active” (colored spheres) macroparticles traversing the injection plane (red rectangle). (bottom) Snapshot of the beam macroparticles (colored spheres) passing through the background of electrons (dark brown streamlines) and the diagnostic stations (red rectangles). The electrons, the injection plane and the diagnostic stations are fixed in the laboratory plane, and are thus counter-propagating to the beam in a boosted frame.

[Fig_inputoutput]

The input and output data are often known from, or compared to, experimental data. Thus, calculating in a frame other than the laboratory entails transformations of the data between the calculation frame and the laboratory frame. This section describes the procedures that have been implemented in the Particle-In-Cell framework Warp (Grote et al. 2005) to handle the input and output of data between the frame of calculation and the laboratory frame (Vay et al. 2011). Simultaneity of events between two frames is valid only for a plane that is perpendicular to the relative motion of the frame. As a result, the input/output processes involve the input of data (particles or fields) through a plane, as well as output through a series of planes, all of which are perpendicular to the direction of the relative velocity between the frame of calculation and the other frame of choice.

Input in a boosted frame simulation

Particles -

Particles are launched through a plane using a technique that is generic and applies to Lorentz boosted frame simulations in general, including plasma acceleration, and is illustrated using the case of a positively charged particle beam propagating through a background of cold electrons in an assumed continuous transverse focusing system, leading to a well-known growing transverse “electron cloud” instability (Vay 2007). In the laboratory frame, the electron background is initially at rest and a moving window is used to follow the beam progression. Traditionally, the beam macroparticles are initialized all at once in the window, while background electron macroparticles are created continuously in front of the beam on a plane that is perpendicular to the beam velocity. In a frame moving at some fraction of the beam velocity in the laboratory frame, the beam initial conditions at a given time in the calculation frame are generally unknown and one must initialize the beam differently. However, it can be taken advantage of the fact that the beam initial conditions are often known for a given plane in the laboratory, either directly, or via simple calculation or projection from the conditions at a given time in the labortory frame. Given the position and velocity $\{x, y, z, v_x, v_y, v_z\}$ for each beam macroparticle at time $t = 0$ for a beam moving at the average velocity $v_b = \beta_b c$ (where c is the speed of light) in the laboratory, and using the standard synchronization ($z = z' = 0$ at $t = t' = 0$) between the laboratory and the calculation frames, the procedure for transforming the beam quantities for injection in a boosted frame moving at velocity βc in the laboratory is as follows (the superscript $'$ relates to quantities known in the boosted frame while the superscript * relates to quantities that are know at a given longitudinal position z^* but different times of arrival):

1. project positions at $z^* = 0$ assuming ballistic propagation

$$\begin{aligned} t^* &= (z - \bar{z})/v_z \\ x^* &= x - v_x t^* \\ y^* &= y - v_y t^* \\ z^* &= 0 \end{aligned}$$

   the velocity components being left unchanged,

2. apply Lorentz transformation from laboratory frame to boosted frame

$$\begin{aligned} t'^* &= -\gamma t^* \\ x'^* &= x^* \\ y'^* &= y^* \\ z'^* &= \gamma\beta c t^* \\ v'^*_x &= \frac{v^*_x}{\gamma(1-\beta\beta_b)} \\ v'^*_y &= \frac{v^*_y}{\gamma(1-\beta\beta_b)} \\ v'^*_z &= \frac{v^*_z - \beta c}{1-\beta\beta_b} \end{aligned}$$

   where $\gamma = 1/\sqrt{1-\beta^2}$. With the knowledge of the time at which each beam macroparticle crosses the plane into consideration, one can inject each beam macroparticle in the simulation at the appropriate location and time.

3. synchronize macroparticles in boosted frame, obtaining their positions at a fixed $t' = 0$ (before any particle is injected)

$$z' = z'^* - \bar{v}'^*_z t'^*$$

   This additional step is needed for setting the electrostatic or electromagnetic fields at the plane of injection. In a Particle-In-Cell code, the three-dimensional fields are calculated by solving the Maxwell equations (or static approximation like Poisson, Darwin or other (Vay 2008)) on a grid on which the source term is obtained from the macroparticles distribution. This requires generation of a three-dimensional representation of the beam distribution of macroparticles at a given time before they cross the injection plane at z'^*. This is accomplished by expanding the beam distribution longitudinally such that all macroparticles (so far known at different times of arrival at the injection plane) are synchronized to the same time in the boosted frame. To keep the beam shape constant, the particles are "frozen" until they cross that plane: the three velocity components and the two position components perpendicular to the boosted frame velocity are kept constant, while the remaining position component is advanced at the average beam velocity. As particles cross the plane of injection, they become regular "active" particles with full 6-D dynamics.

Figure *[Fig_inputoutput]* (top) shows a snapshot of a beam that has passed partly through the injection plane. As the frozen beam macroparticles pass through the injection plane (which moves opposite to the beam in the boosted frame), they are converted to "active" macroparticles. The charge or current density is accumulated from the active and the frozen particles, thus ensuring that the fields at the plane of injection are consistent.

Laser -

Similarly to the particle beam, the laser is injected through a plane perpendicular to the axis of propagation of the laser (by default z). The electric field $E_\perp$ that is to be emitted is given by the formula

$$E_\perp\left(x,y,t\right)=E_0 f\left(x,y,t\right)\sin\left[\omega t+\phi\left(x,y,\omega\right)\right]$$

where E_0 is the amplitude of the laser electric field, $f\left(x,y,t\right)$ is the laser envelope, ω is the laser frequency, $\phi\left(x,y,\omega\right)$ is a phase function to account for focusing, defocusing or injection at an angle, and t is time. By default, the laser envelope is a three-dimensional gaussian of the form

$$f\left(x,y,t\right)=e^{-\left(x^2/2\sigma_x^2+y^2/2\sigma_y^2+c^2t^2/2\sigma_z^2\right)}$$

where σ_x, σ_y and σ_z are the dimensions of the laser pulse; or it can be defined arbitrarily by the user at runtime. If $\phi\left(x,y,\omega\right)=0$, the laser is injected at a waist and parallel to the axis z.

If, for convenience, the injection plane is moving at constant velocity $\beta_s c$, the formula is modified to take the Doppler effect on frequency and amplitude into account and becomes

$$\begin{aligned}E_\perp\left(x,y,t\right)= & \left(1-\beta_s\right)E_0 f\left(x,y,t\right)\\ \times & \sin\left[\left(1-\beta_s\right)\omega t+\phi\left(x,y,\omega\right)\right].\end{aligned}$$

The injection of a laser of frequency ω is considered for a simulation using a boosted frame moving at βc with respect to the laboratory. Assuming that the laser is injected at a plane that is fixed in the laboratory, and thus moving at $\beta_s=-\beta$ in the boosted frame, the injection in the boosted frame is given by

$$\begin{aligned}E_\perp\left(x',y',t'\right)= & \left(1-\beta_s\right)E_0' f\left(x',y',t'\right)\\ \times & \sin\left[\left(1-\beta_s\right)\omega' t'+\phi\left(x',y',\omega'\right)\right]\\ = & \left(E_0/\gamma\right)f\left(x',y',t'\right)\\ \times & \sin\left[\omega t'/\gamma+\phi\left(x',y',\omega'\right)\right]\end{aligned}$$

since $E_0'/E_0=\omega'/\omega=1/\left(1+\beta\right)\gamma$.

The electric field is then converted into currents that get injected via a 2D array of macro-particles, with one positive and one dual negative macro-particle for each array cell in the plane of injection, whose weights and motion are governed by $E_\perp\left(x',y',t'\right)$. Injecting using this dual array of macroparticles offers the advantage of automatically including the longitudinal component that arises from emitting into a boosted frame, and to automatically verify the discrete Gauss' law thanks to using charge conserving (e.g. Esirkepov) current deposition scheme (Esirkepov 2001).

Output in a boosted frame simulation

Some quantities, e.g. charge or dimensions perpendicular to the boost velocity, are Lorentz invariant. Those quantities are thus readily available from standard diagnostics in the boosted frame calculations. Quantities that do not fall in this category are recorded at a number of regularly spaced "stations", immobile in the laboratory frame, at a succession of time intervals to record data history, or averaged over time. A visual example is given on Fig. *[Fig_inputoutput]* (bottom). Since the space-time locations of the diagnostic grids in the laboratory frame generally do not coincide with the space-time positions of the macroparticles and grid nodes used for the calculation in a boosted frame, some interpolation is performed at runtime during the data collection process. As a complement or an alternative, selected particle or field quantities can be dumped at regular intervals and quantities are reconstructed in the laboratory frame during a post-processing phase. The choice of the methods depends on the requirements of the diagnostics and particular implementations.

Cowan, Benjamin M, David L Bruhwiler, John R Cary, Estelle Cormier-Michel, and Cameron G R Geddes. 2013. "Generalized algorithm for control of numerical dispersion in explicit time-domain electromagnetic simulations." *Physical Review Special Topics-Accelerators and Beams* 16 (4). https://doi.org/10.1103/PhysRevSTAB.16.041303.

Esirkepov, Tz. 2001. “Exact Charge Conservation Scheme for Particle-in-Cell Simulation with an Arbitrary Form-Factor.” *Computer Physics Communications* 135 (2): 144–53.

Grote, D P, A Friedman, J.-L. Vay, and I Haber. 2005. “The Warp Code: Modeling High Intensity Ion Beams.” In *Aip Conference Proceedings*, 55–58. 749.

Vay, J L. 2008. “Simulation of Beams or Plasmas Crossing at Relativistic Velocity.” *Physics of Plasmas* 15 (5): 56701. https://doi.org/10.1063/1.2837054.

Vay, J.-L. 2007. “Noninvariance of Space- and Time-Scale Ranges Under A Lorentz Transformation and the Implications for the Study of Relativistic Interactions.” *Physical Review Letters* 98 (13): 130405/1–4.

Vay, J -L., C G R Geddes, E Esarey, C B Schroeder, W P Leemans, E Cormier-Michel, and D P Grote. 2011. “Modeling of 10 Gev-1 Tev Laser-Plasma Accelerators Using Lorentz Boosted Simulations.” *Physics of Plasmas* 18 (12). https://doi.org/10.1063/1.3663841.