iPIC3D Documentation

Paul Wilhelm, Pranab J Deka, & Fabio Bacchini

Centre for mathematical Plasma Astrophysics, KU Leuven, Belgium, B–3001
Leuven Gravity Institute, KU Leuven, Belgium, B–3001

May 26, 2025

This documentation has been developed
within the scope of SPACE Centre of Excellence.

Disclaimer

This documentation is provided "AS IS", without warranty of any kind, express or implied, including but not limited to warranties of merchantability, fitness for a particular approach and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages, or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

In case of any questions, comments, or suggestions, please contact:

- Pranab J Deka (pranab.deka@kuleuven.be)
- Fabio Bacchini (fabio.bacchini@kuleuven.be)
- Paul Wilhelm (paul.wilhelm@kuleuven.be)

SPACE Acknowledgement & Disclaimer

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements

We are grateful to Stefano Markidis who shared with us the new version of `iPIC3D` for SPACE CoE, that was developed by his team at KTH Sweden. We are also grateful to Andong Hu for helping us understand the technical aspects of this new version.

Citation

If you use `iPIC3D-CPU-SPACE-CoE` or `iPIC3D-GPU-SPACE-CoE` with the energy-conserving semi-implicit method, you are requested to cite the software and algorithm as "This work has been carried out using the `iPIC3D` code (Markidis et al., 2010) with the ECSIM (Lapenta, 2017) algorithm. `iPIC3D` is an open-source software that runs on both CPUs (https://github.com/Pranab-JD/iPIC3D-CPU-SPACE-CoE) and GPUs (https://github.com/Pranab-JD/iPIC3D-GPU-SPACE-CoE)." If you use the relativistic semi-implicit method, kindly cite Bacchini (2023).

Contents

Chapter 1

Introduction

iPIC3D (Markidis et al., 2010) is a PIC code developed primarily to study plasma dynamics at kinetic scales. The individual (macro)particles used to represent plasma particles are evolved in a Lagrangian framework whereas the moments (plasma density, current, etc) and the self-consistent electric and magnetic fields are tracked on an Eulerian grid. It solves the Maxwell-Vlasov equation self-consistently using the implicit moment method. The three main kernels of iPIC3D are (a) Particle Mover, (b) Moment Gatherer, and (c) Field Solver. Owing to the "implicitness" of the underlying algorithm, as opposed to explicit PIC methods, unresolved scales do not result in numerical instabilities. This allows us to choose time step sizes and spatial grid sizes that can be 10–100 times greater than those used in traditional explicit PIC codes.

In this documentation, we focus exclusively on iPIC3D-CPU-SPACE-CoE and iPIC3D-GPU-SPACE-CoE, unless stated otherwise, however, there are several versions of the code which can be found in the following list:

1. iPIC3D-CPU-SPACE-CoE
   https://github.com/Pranab-JD/iPIC3D-CPU-SPACE-CoE
   Forked from iPIC3D-CPU, includes ECSIM & RelSIM, incorporated by Pranab J Deka and Fabio Bacchini

2. iPIC3D-GPU-SPACE-CoE
   https://github.com/Pranab-JD/iPIC3D-GPU-SPACE-CoE
   Forked from iPIC3D-GPU, includes ECSIM & RelSIM, incorporated by Pranab J Deka and Fabio Bacchini

3. iPic3D (No longer maintained)
   https://github.com/CmPA/iPic3D
   This is the original code developed at CmPA by Stefano Markidis and Giovanni Lapenta

4. ECSIM
   https://bitbucket.org/CmPA/ecsim/src/new-master/
   This implements the energy-conserving method (and RelSIM) in a spin-off of iPic3D

5. ECSIM (No longer maintained)
   https://gitlab.com/valsusa/ecsimpic
   Some parts of the code has been ported to GPU using OpenMP by a South Korean group

6. iPIC3D-KTH (No longer maintained)
   https://github.com/KTH-HPC/iPIC3D
   Some optimisations have been carried out, performs better than iPic3D

7. sputniPIC (No longer maintained)
   https://github.com/KTH-HPC/sputniPIC
   Supports GPU simulations using CUDA

8. iPIC3D-CPU
   https://github.com/iPIC3D/iPIC3D-CPU
   This is a new version of the code developed by Stefano Markidis and his team

9. iPIC3D-GPU
   https://github.com/iPIC3D/iPIC3D-GPU
   A CUDA+HIP-based GPU code developed by Stefano Markidis and his team

Kindly note that you may not have access to one or more of the above repositories depending on your access level. **In this documentation, we focus exclusively on iPIC3D-CPU-SPACE-CoE and iPIC3D-GPU-SPACE-CoE, unless stated otherwise.**

1.1 Install and build iPIC3D

1.1.1 iPIC3D-CPU-SPACE-CoE

To compile the CPU version of the code, you need the following:

- gcc/g++ compiler
- MPI (OpenMPI or MPICH)
- OpenMP (optional)
- HDF5 (optional)
- VTK (optional)
- Paraview/Catalyst (optional)

Please follow these instructions to install and build the code:

1. Download the code

```
git clone https://github.com/Pranab-JD/iPIC3D-CPU-SPACE-CoE.git
```

2. Create build directory inside the folder

```
cd iPIC3D-CPU-SPACE-CoE && mkdir build && cd build
```

3. Compile the code (part 1)

```
cmake ..
ccmake ..
```

You will get a list of options (Listing 1.1) with `ccmake ..`

```
BENCH_MARK                      OFF
BUILD_SHARED_LIBS               ON
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX            /usr/local
H5HUT_LIBRARY                   /users/dekapran/H5hut/lib/libH5hut.so
H5HUT_ROOT                      /users/dekapran/H5hut
PROFILE_FIELDS                  OFF
PROFILE_MOMENTS                 OFF
PROFILING                       OFF
SITE                            default
USE_BATSRUS                     OFF
USE_CATALYST                    OFF       # Requires Python and Paraview
USE_H5HUT                       ON        # Requires HDF5 and H5hut
USE_HDF5                        ON        # Requires HDF5
USE_OPENMP                      ON
```

Listing 1.1: Making options

Should you choose to use HDF5, H5hut, OpenMP, or Catalyst, the corresponding options need to be turned on. After choosing the desired options, you need to configure (press 'c') and generate (press 'g').

4. Compile the code (part 2)

```
make -j     # -j = build with max # of threads - fast, recommended
```

Here, `-j` corresponds to using all available threads (for faster execution) to build the code. You could also choose to build with 'n' cores using `make -n`. If the code compiles correctly, you will get an executable called `iPIC3D`. We summarise the above bash commands, sequentially, in one Listing 4 for user friendliness.

```
git clone https://github.com/Pranab-JD/iPIC3D-CPU-SPACE-CoE.git

cd iPIC3D-CPU-SPACE-CoE && mkdir build && cd build

cmake ..

ccmake ..   # choose desired options

make -j
```

On a computing cluster, you may have to load modules to enable build and compilation. We provide the modules you can load to build and compile the code for a selection of clusters. Of course, you can choose other combinations or versions of modules, provided that they are compatible with each other.

Karolina	`ml CMake/3.27.6-GCCcore-13.2.0 HDF5/1.14.3-gompi-2023b`
Leonardo	`ml cmake/3.27.7 hdf5/1.14.3--openmpi--4.1.6--gcc--12.2.0`
Lumi	`ml cray-hdf5/1.12.2.11`
MareNostrum	`ml purge` `ml ucx/1.16.0-gcc cmake/3.30.5 openmpi/5.0.5-gcc hdf5/1.14.1-2-gcc-openmpi`
MeluXina	`ml CMake/3.26.3-GCCcore-12.3.0 HDF5/1.10.7-gompi-2023a`
VSC (Belgium)	`ml CMake/3.20.1-GCCcore-10.3.0 HDF5/1.10.7-gompi-2021a`

Table 1.1: List of the modules to be (explicitly) loaded on selected clusters. Note that these modules are subject to change, i.e., can be upgraded to to their latest versions.

1.1.2 iPIC3D-GPU-SPACE-CoE

To compile the GPU version of the code, you need the following:

- ◇ gcc/g++ or nvhpc compiler
- ◇ MPI (OpenMPI or MPICH)
- ◇ CUDA (NVIDIA GPUs) or HIP (AMD GPUs) (one of these is mandatory)
- ◇ OpenMP (optional)
- ◇ HDF5 (optional)
- ◇ VTK (optional)
- ◇ Paraview/Catalyst (optional)

Karolina	`ml CMake/3.24.3-GCCcore-12.2.0` `ml OpenMPI/5.0.5-NVHPC-24.3-CUDA-12.3.0`
Leonardo	`ml cmake/3.27.7 cuda/12.3` `ml hdf5/1.14.3--openmpi--4.1.6--nvhpc--24.3`
Lumi	`ml LUMI/24.03 buildtools/24.03 rocm/6.0.3`
MeluXina	`ml CMake/3.26.3-GCCcore-12.3.0` `ml OpenMPI/4.1.5-NVHPC-23.7-CUDA-12.2.0`
VSC (Belgium)	`ml foss/2023b`

Table 1.2: List of the modules to be (explicitly) loaded on selected clusters. Note that these modules are subject to change, i.e., can be upgraded to to their latest versions.

Chapter 2

Running simulations

2.1 Units

2.1.1 Input File

Everything is normalized to the ion plasma parameters. Thus in the input file L_x, L_y, L_z are in units of ion skin depth:

$$[L] = d_i. \tag{2.1}$$

The time unit is reversed ion plasma frequency:

$$[t] = 1/f_{p,i}. \tag{2.2}$$

Hence, the light speed is unity:

$$c = 1. \tag{2.3}$$

Charge is normalized to e:

$$[e] = 1. \tag{2.4}$$

The mass is normalized by the ion mass:

$$m_i = 1. \tag{2.5}$$

Furthermore, as we use Gaussian unit, we use the following (normalized) units for the charge density:

$$[\rho] = \frac{1}{4\pi} \tag{2.6}$$

The energy is in units of

$$[Energy] = m_i c^2. \tag{2.7}$$

2.1.2 Pressures

If you need to compute, e.g., plasma beta you have to compute the gas and the magnetic pressures:

$$p_{gas} = u_{th,i}^2 \frac{\rho_i}{4\pi} + \frac{m_e}{m_i} u_{th,e}^2 \frac{\rho_e}{4\pi} \tag{2.8}$$

Here the first term on the right-hand side is the ion pressure, and the second term is the electron pressure. Note, that you should take the value of a single (x, y, or z) component of the thermal velocity!

$$p_{mag} = \frac{B^2}{8\pi} \tag{2.9}$$

Plasma β is defined as

$$\beta = \frac{p_{gas}}{p_{mag}}. \tag{2.10}$$

2.1.3 Miscellaneous

The magnetic permittivity (permeability) of vacuum is set to

$$\mu_0 = 1. \tag{2.11}$$

The Alfven speed equals to the B0 in the inputfile

$$v_A = B_0. \tag{2.12}$$

The currents in iPIC3D (exactly Faraday's law):

$$j = \frac{c\nabla \times B}{4\pi}. \tag{2.13}$$

Finally for the velocity distribution

$$\frac{kT}{m} = v_{th} \tag{2.14}$$

where k is Boltzmann constant, T is temperature, m is particle mass for the respective species; v_{th} is initial thermal speed for species.

So the distribution of the absolute value of velocities should follow the Maxwellian:

$$M(u) = 4\pi \left(\frac{1}{2\pi u_{th}^2}\right)^{\frac{3}{2}} \exp\left(-\tfrac{1}{2}\tfrac{u^2}{u_{th}^2}\right), \tag{2.15}$$

where u_{th} is the thermal speed for the current species.

2.2 Inputfiles

A simulation is started by providing an input file to the the executable `iPic3D`. All input files have use ".inp" extension. In the git-repository you can also find a number of predefined input files for some common simulation setups. As a case-study we will discuss the input file for a simple ion-electron plasma with Maxwellian velocity distribution functions and effective dimension of 2x2v. The full input file can be found under the name `Maxwell2d.inp` in the `inputfiles` subfolder.

2.2.1 Initialising problem from Input Files

The initialization of the simulation is done via calling

```
int c_Solver::Init(int argc, char **argv)
```

Internally this calls

```
col = new Collective(argc, argv);
```

which after performing a couple checks on the format of the input calls

```
ReadInput(inputfile);
init_derived_parameters();
```

to initialize the simulation parameters.

2.2.2 Sample Input File

Test Case

First we define folders where the output data is supposed to be written to. In case one plans to store the simulation results at a certain point and restart from that later on, e.g., due to run-time constraints on a cluster, a restart directory to write the corresponding data can be also defined at this point. At this point we can also define how the simulation is to be called in the output.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                 TEST CASE                                     %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Case                        = DoubleHarris              # Simulation Case
SimulationName              = Double_Harris             # Simulation name for the output
SaveDirName                 = data_double_H             # Output directory
```

Data Output

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                    OUTPUT                                      %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
WriteMethod                = H5hut      # Output method [ shdf5 | phdf5 | H5hut |pvtk | nbcvtk ]

DiagnosticsOutputCycle     = 1          # Diagnostics cycle
FieldOutputCycle           = 20         # Output for moments & fields (if <= 0, data is NOT saved)
ParticlesOutputCycle       = 10         # Output for particles (if <= 0, data is NOT saved)

# Select quantites to be written to files
FieldOutputTag       = E + B + J + rho + rho_s + J_s + E_flux + pressure
```

The iPIC3D library employs either the HDF5 or VTK format to store output data and this can be defined by setting WriteMethod. The available options[1] are:

- shdf5: Serial output in HDF5 format (works, choose this for HDF5 files)
- phdf5: Parallel output in HDF5 format using parallel HDF5 (does not work, needs debugging)
- nbcvtk: Non-blocking MPI-parallelized output in VTK format (only electric and magnetic fields)
- pvtk: Blocking MPI-parallelized output in VTK format (only electric and magnetic fields)
- H5hut: Parallel output in .h5 format using H5hut

shdf5 and phdf5 will create 'N' HDF5 files, where 'N' corresponds to the total number of MPI tasks defined by the user. All field and particle data, at their respective output cycles, will be written to these files. Note that VTK files are preferred if the user is exclusively interested in data visualisation. If you choose either pvtk or nbcvtk, field data will be written to VTK files, whilst particle data will be written to HDF5 files. Two VTK files will be created at every FieldOutputCycle - one for electric and the other for magnetic field.

If you want to restart your simulations, it is crucial to choose H5hut as the WriteMethod. One .h5 file will be created every FieldOutputCycle - this will store the fields and moments. Another .h5 file will be created every ParticleOutputCycle - this will store the charge, position, and velocity of every particle. Note that all data are written without ghost cells.

If FieldOutputCycle or ParticleOutputCycle is set to a value <= 0, **NO data will be written to files**. The user has the flexibility of choosing the data to be written to files using FieldOutputTag - the available options are listed below. Leaving this options empty will result in **NO data being written to files**. Fields and moments are a part of FieldOutputTag and their frequency is set by FieldOutputCycle.

Additionally, two diagnostic text files, ConservedQuantities and SpeciesQuantities, are created to store simulaion diagnostics at every DiagnosticsOutputCycle. ConservedQuantities stores the electric, magnetic, kinetic (all particles), and total energy, along with the energy difference with respect to the initial total energy of the system and the total momentum of the particles. SpeciesQuantities stores the number of particles, charge, momentum, and kinetic energy of each of the species involved in the simulations.

```
###? Options for output tags
###! NOTE: If you DO NOT set the desired tags, NO DATA will be written to files!
##* Fields
    # B                 --> Bx, By, and Bz
    # E                 --> Ex, Ey, and Ez

##* Moments
    # J                 --> Jxh, Jyh, and Jzh: overall current density (all species combined)
    # rho               --> rho: overall charge density (all species combined)
    # J_s               --> current densities for each species
    # rho_s             --> charge densities for each species
    # pressure          --> pressure tensor for each species
    # E_flux            --> energy flux for each species
    # H_flux            --> heat flux tensor for each species
```

By default, the code always outputs the electric, magnetic, kinetic, and total energies at every diagnostic cycle.

Electromagnetic Fields

We define the strength of the initial magnetic (and electric, if any) field. By default, they are set to 0. External fields have not yet been implemented.

[1] Note that choosing an option which is not supported on your system or which you did not link during compilation will result in an error.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                          ELECTRIC and MAGNETIC FIELD                          %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

B0x          = 0.001            # Initial magnetic field strength along X
B0y          = 0.001            # Initial magnetic field strength along Y
B0z          = 0.0              # Initial magnetic field strength along Z
```

The initial field configuration and particle distribution can be chosen by parameter `Case`. Some of the available options are:

- `Maxwellian`: Initialise a uniform spatial distribution of particles with Maxwellian velocities in 1D/2D/3D
- `DoubleHarris`: Initialise a uniform spatial distribution of particles with two antiparallel current sheets

Remark 1. *There is also the option `BATSRUS`, which is the option to be chosen when coupling with the fluid code BATSRUS.*

Simulation Domain and Numerical Parameters

The time step size is defined by `dt`, the total number of time steps (cycles) by `ncycles`, the decentering parameter, `th` (θ), and the speec of light by `c`. `GMREStol` is the user-defined tolerance of the field solver. It must be $< 10^{-15}$ to conserve energy exactly.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                    TIME                                       %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dt             = 0.03         # Time-step size
ncycles        = 50           # Number of time steps
th             = 0.5          # Decentering parameter
c              = 1.0          # Speed of light
GMREStol       = 1E-8         # GMRES tolerance (must be <= 1e-15 for exact energy conservation)
```

Lx, Ly, and Lz correspond to the domain size whilst nxc, nyc, and nzc correspond to the grid resolution.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                   BOX SIZE                                    %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Lx              = 40.0                  # Simulation box length along X
Ly              = 40.0                  # Simulation box length along Y
Lz              = 0.1                   # Simulation box length along Z

nxc             = 1024                  # Number of cells along X
nyc             = 1024                  # Number of cells along Y
nzc             = 1                     # Number of cells along Z
```

For the case of magnetic reconnection initialised using a standard Harris sheet or a double Harris sheet (GEM, DoubleHarris, etc.), you can set the thickness of the current sheet by setting

```
delta = 0.5     # Thickness of the current sheet
```

Smoothing

The parameter `Smooth` is to used to adjust smoothing of the fields. `num_smoothings` corresponds to how many times due we smooth a dataset at a given time and `SmoothCycle` corresponds to how often (or after how many time cycles) do we wish to smooth the data. If `Smooth = 0`, no data will be smoothed and the params, `num_smoothings` and `SmoothCycle`, will be ignored.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                  SMOOTHING                                    %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Smooth          = 1    # If 0 (1 = true OR 0 = false), the following 2 parameters will be ignored
num_smoothings  = 2    # Number of smoothings
SmoothCycle     = 1    # After how many time cycles is the data smoothed
```

Parallelisation

We have to provide how many MPI subdomains we desire along each direction. XLEN x YLEN x ZLEN must equal to the total number of MPI tasks.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                    MPI TOPOLOGY                                 %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

XLEN          = 32                    # Number of MPI subdomains along X
YLEN          = 32                    # Number of MPI subdomains along Y
ZLEN          = 1                     # Number of MPI subdomains along Z
```

Particles

Next we set up how many particle species we want and how many particles each species is supposed to have. With `ns` we set the number of particle species and with `rhoINIT` how the charge density is distributed over the different particle species. If you want more than one particle species you have to provide the corresponding parameters for the different particle species in a list separated by blank spaces as is show-cased below. The first species are electrons, followed by protons and any other desired ion species.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                    PARTICLES                                    %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#*    ns = number of species
#*    0 = electrons; 1 = protons/positrons; 2,3,4,5,... = electrons/positrons/ions

ns                  = 2                                 # Number of species of particles

rhoINIT             = 0.5           0.5                 # Initial density
rhoINJECT           = 0.0           0.0                 # Injection density

npcelx              = 50            50                  # Particles per cell along X
npcely              = 50            50                  # Particles per cell along Y
npcelz              = 1             1                   # Particles per cell along Z
NpMaxNpRatio        = 5.0                               # Maximum number of particles allocated

qom                 = -1.0          1.0                 # Charge/mass ratio

uth                 = 0.001         0.001               # Thermal velocity along X
vth                 = 0.001         0.001               # Thermal velocity along Y
wth                 = 0.000         0.000               # Thermal velocity along Z

u0                  = 0.02          0.02                # Drift/bulk velocity along X
v0                  = 0.0           0.0                 # Drift/bulk velocity along Y
w0                  = 0.0           0.0                 # Drift/bulk velocity along Z
```

Boundary conditions: Only periodic boundaries are correctly implemented, for both particles and fields. Other boundary types may or may not work! We suggest sticking to periodic boundaries.

```
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# %                                BOUNDARY CONDITION                               %
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

PERIODICX                       = 1                 # Along X (1 = true OR 0 = false)
PERIODICY                       = 1                 # Along Y (1 = true OR 0 = false)
PERIODICZ                       = 1                 # Along Z (1 = true OR 0 = false)
```

We can also set the boundary condition for the particles to periodic by setting variables `PERIODICX_P`, `PERIODICY_P` and `PERIODICZ_P` to 1. If these are not set explicitly, by default, boundaries for the particles are assumed to be the same as that for the fields.

Remark

There are some (advanced) setups that cannot be obtained through the input files, such as providing custom initial distribution functions via a closed expression. These have to be hardcoded and the code has to be recompiled. Further information on how such cases can be treated can be found in chapter **??**.

Chapter 3

Scaling Plots

We have implemented ECSIM in `iPIC3D-CPU-SPACE-CoE`. For a test case of a double Harris sheet leading to magnetic reconnection, we show the conversion of magnetic energy to kinetic energy of particles whilst conserving energy exactly up to machine precision (Fig. 3.1). This test case was run on Karolina CPU on 8 nodes with 128 cores per node for a total of 10000 time steps.

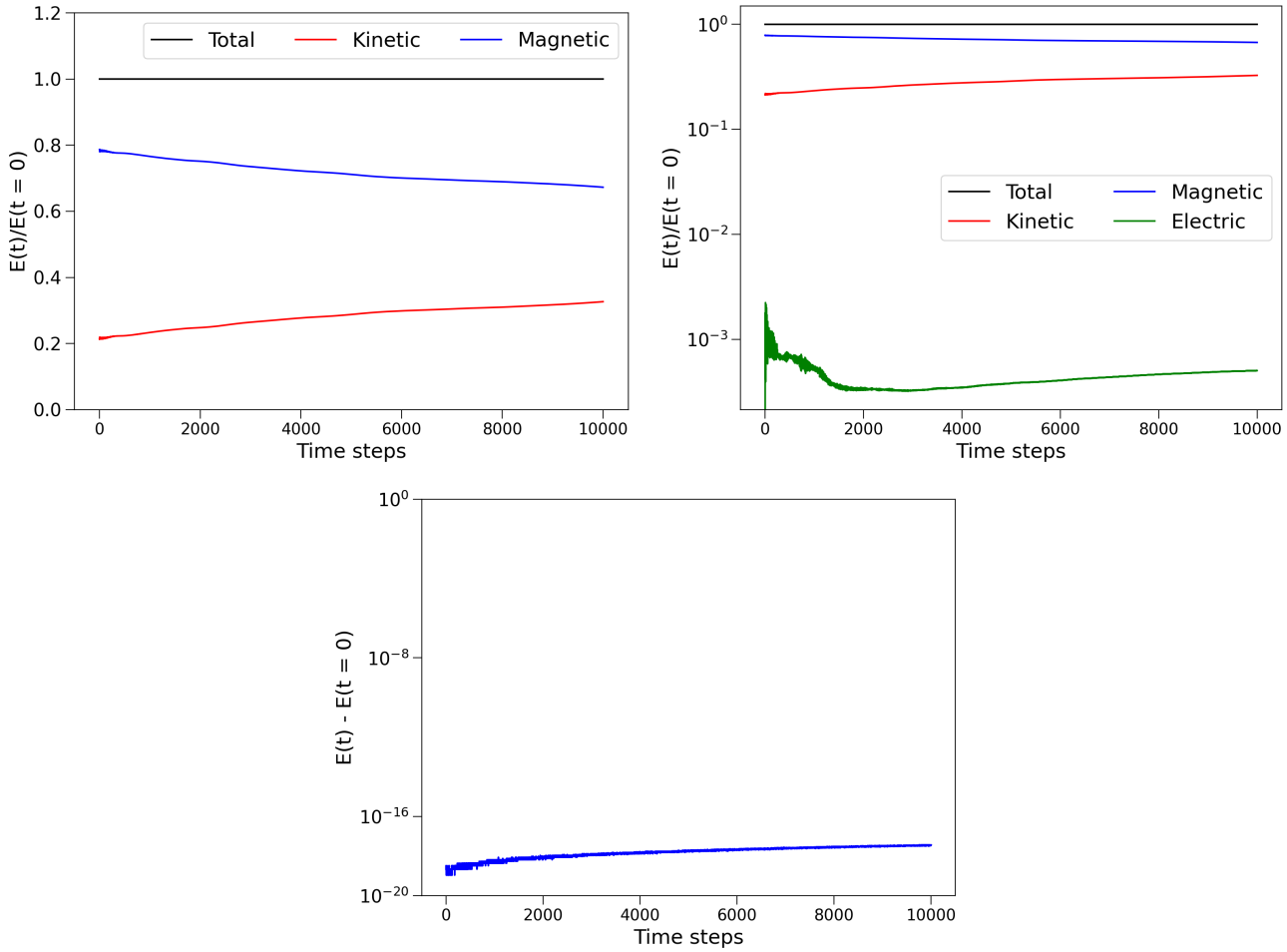
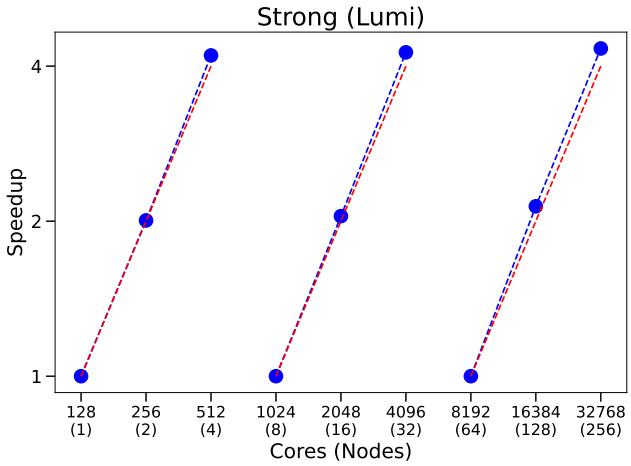
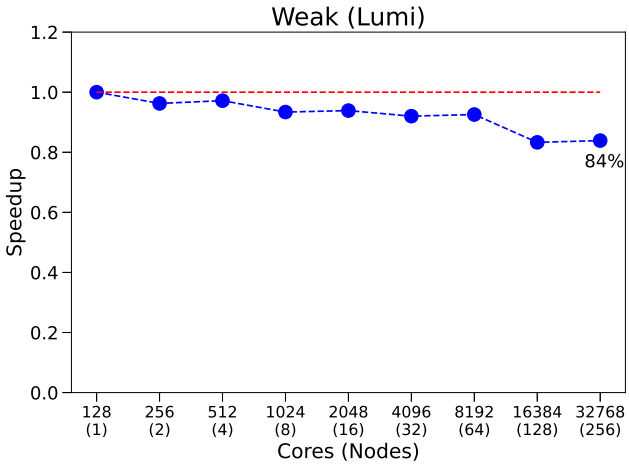


Figure 3.1: Conversion of magnetic energy to kinetic energy of particles whilst conserving energy exactly.

LUMI: We present results of strong and weak scaling for `iPIC3D-CPU-SPACE-CoE` (with ECSIM) on LUMI-C for a 2D (Table 3.1) and 3D (Table 3.2) test case. Strong scaling tests for 2D are initialised with 512×256 grid cells along X and Y directions for 1 to 4 nodes, 1024×512 cells for 8 to 32 nodes, and 4096×2048 cells for 64 to 256 nodes. We consider three baselines (1 node, 8 nodes, and 64 nodes) for strong scaling as increasing the number of processors without increasing the system size would result in very few cells per processor leading to a significant overhead of the internode communications (which is seldom the case for realistic astrophysical PIC simulations). Weak scaling tests are initialised with 256×256 grid cells on 1 node, following which the system size is increased proportional to the number of cores/nodes. For each of these simulations, we consider 80×80 particles per cell for

Table 3.1: Strong (left) and weak (right) scaling of `iPIC3D-CPU-SPACE-CoE` on LUMI-C for a 2D Maxwellian. Runtimes are measured in seconds.





nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Speedup
128 (1)	482.04	482.04	1.00
256 (2)	240.25	241.02	2.00
512 (4)	114.89	120.51	4.20
1024 (8)	244.66	244.66	1.00
2048 (16)	119.62	122.33	2.04
4096 (32)	57.51	61.16	4.25
8192 (64)	549.61	549.61	1.00
16384 (128)	257.16	274.80	2.13
32768 (256)	126.99	137.40	4.32

nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Efficiency
128 (1)	230.44	230.44	100%
256 (2)	239.40	230.44	96%
512 (4)	237.14	230.44	97%
1024 (8)	246.77	230.44	93%
2048 (16)	245.48	230.44	94%
4096 (32)	250.42	230.44	92%
8192 (64)	248.96	230.44	92%
16384 (128)	276.65	230.44	83%
32768 (256)	274.70	230.44	84%

2 species of particles. We obtain around 84% efficiency for weak scaling and almost ideal strong scaling on 32768 cores.

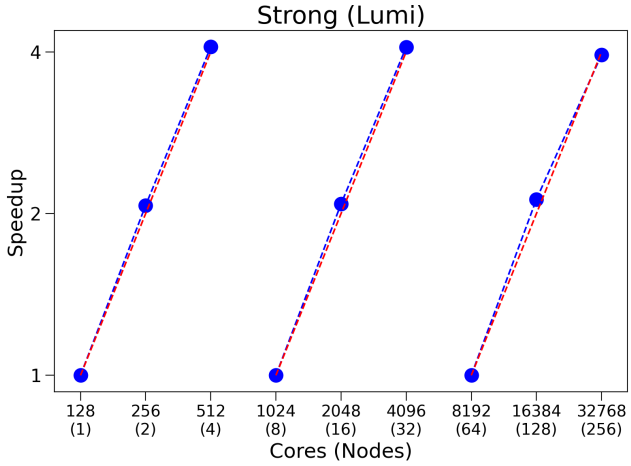
Strong scaling tests for 3D are initialised with $64 \times 32 \times 32$ grid cells along X, Y, and Z directions, respectively, for 1 to 4 nodes, $128 \times 64 \times 64$ cells for 8 to 32 nodes, and $256 \times 128 \times 128$ cells for 64 to 256 nodes. We consider three baselines — 1 node, 8 nodes, and 64 nodes. Weak scaling tests are initialised with $64 \times 32 \times 32$ grid cells on 1 node, following which the system size is increased proportional to the number of cores/nodes. For each of these simulations, we consider $20 \times 20 \times 20$ particles per cell for 2 species of particles. We obtain around 80% efficiency for weak scaling and almost ideal strong scaling on 32768 cores.

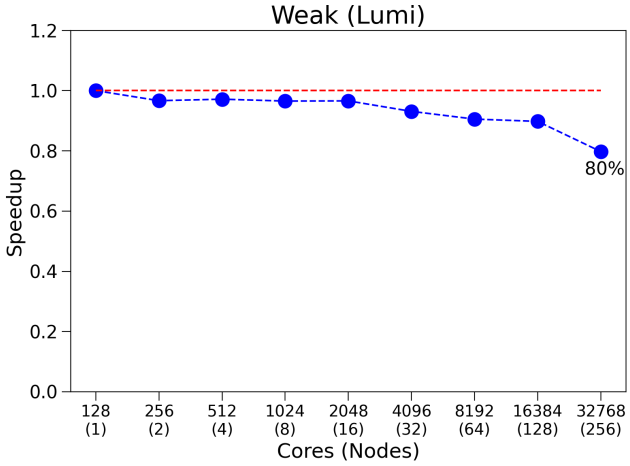
Karolina: We present results of strong and weak scaling for `iPIC3D-CPU-SPACE-CoE` (with ECSIM) on Karolina CPU (Table 3.3) for a 2D test case. Strong scaling tests for 2D are initialised with 256×256 grid cells along X and Y directions for 1 to 8 nodes and 1024×1024 cells for 16 to 128 nodes. We consider two baselines — 1 node and 16 nodes. Weak scaling tests are initialised with 256×256 grid cells on 1 node, following which the system size is increased proportional to the number of cores/nodes. For each of these simulations, we consider 120×120 particles per cell for 2 species of particles. We obtain around 83% efficiency for weak scaling and excellent strong scaling on 16384 cores.

MareNostrum 5: We present results of strong and weak scaling for `iPIC3D-CPU-SPACE-CoE` (with ECSIM) on MareNostrum GPP (Table 3.4) for a 2D test case. Strong scaling tests for 2D are initialised with 256×224 grid cells along X and Y directions for 1 to 8 nodes, 1024×896 cells for 8 to 64 nodes, and 2048×1792. cells for 64 to 512 nodes. We consider three baselines — 1 node, 8 nodes, and 64 nodes. Weak scaling tests are initialised with 256×224 grid cells on 1 node, following which the system size is increased proportional to the number of cores/nodes. For each of these simulations, we consider 120×120 particles per cell for 2 species of particles. We obtain around 83% efficiency for weak scaling and excellent strong scaling on 57334 cores.

`iPIC3D-CPU-SPACE-CoE` shows excellent scalability on LUMI, Karolina, and MareNostrum 5 making it highly efficient and desirable for astrophysical plasma simulations on these clusters. In the near future, we will perform scaling tests for RelSIM.

Table 3.2: Strong (left) and weak (right) scaling of `iPIC3D-CPU-SPACE-CoE` on LUMI-C for a 3D Maxwellian. Runtimes are measured in seconds.

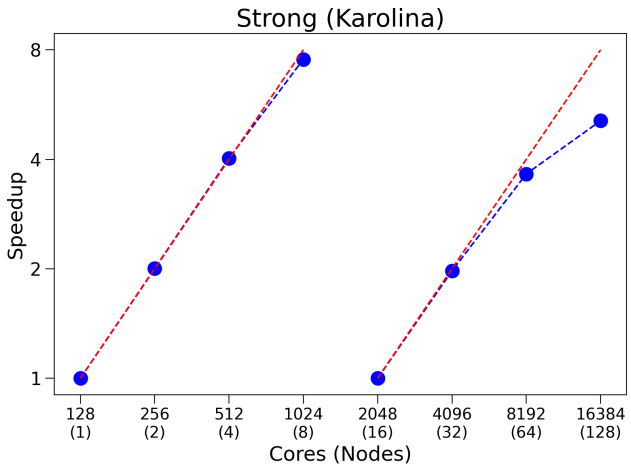


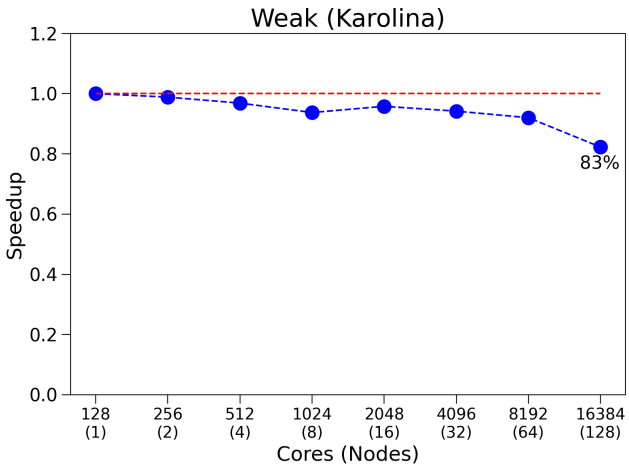


nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Speedup
128 (1)	243.48	243.48	1.00
256 (2)	117.51	121.74	2.07
512 (4)	59.56	60.87	4.09
1024 (8)	252.92	252.92	1.00
2048 (16)	121.33	126.46	2.08
4096 (32)	61.97	63.23	4.08
8192 (64)	272.64	272.64	1.00
16384 (128)	128.19	136.32	2.12
32768 (256)	68.97	68.16	3.95

nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Efficiency
128 (1)	248.05	248.05	100%
256 (2)	256.64	248.05	97%
512 (4)	255.31	248.05	97%
1024 (8)	256.96	248.05	96%
2048 (16)	256.78	248.05	96%
4096 (32)	266.41	248.05	93%
8192 (64)	273.96	248.05	91%
16384 (128)	276.27	248.05	90%
32768 (256)	310.98	248.05	80%

Table 3.3: Strong (left) and weak (right) scaling of `iPIC3D-CPU-SPACE-CoE` on Karolina CPU for a 2D Maxwellian. Runtimes are measured in seconds.

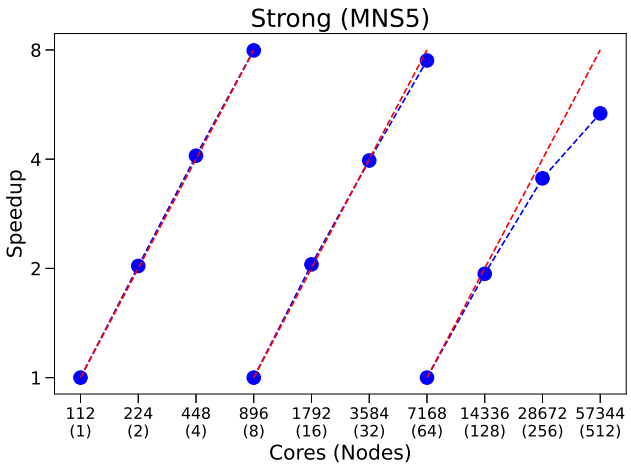


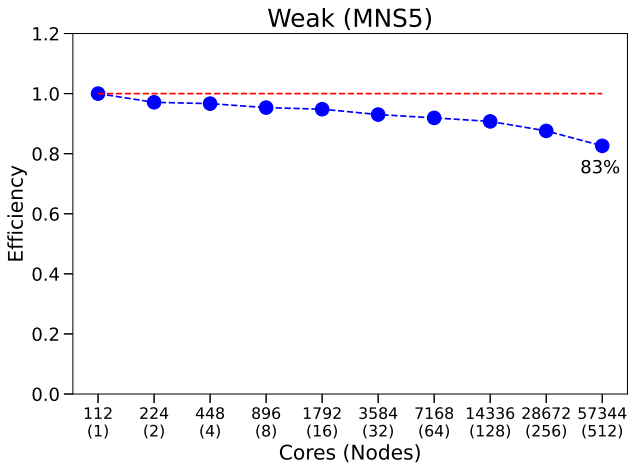


nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Speedup
128 (1)	556.96	556.96	1.00
256 (2)	277.89	278.48	2.00
512 (4)	138.30	139.24	4.02
1024 (8)	73.94	69.62	7.53
2048 (16)	576.09	576.09	1.00
4096 (32)	291.89	288.05	1.97
8192 (64)	157.95	144.02	3.64
16384 (128)	112.85	72.01	5.10

nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Efficiency
128 (1)	558.69	558.69	100%
256 (2)	565.30	558.69	99%
512 (4)	576.86	558.69	97%
1024 (8)	596.10	558.69	94%
2048 (16)	583.26	558.69	96%
4096 (32)	593.19	558.69	94%
8192 (64)	607.22	558.69	92%
16384 (128)	679.10	558.69	82%

Table 3.4: Strong (left) and weak (right) scaling of `iPIC3D-CPU-SPACE-CoE` on MareNostrum GPP for a 2D Maxwellian. Runtimes are measured in seconds.





nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Speedup
112 (1)	311.47 s	311.47 s	1.00
224 (2)	153.35 s	155.74 s	2.03
448 (4)	76.19 s	77.87 s	4.09
896 (8)	39.05 s	38.93 s	7.97
896 (8)	417.67 s	417.67 s	1.00
1792 (16)	203.59 s	208.84 s	2.05
3584 (32)	105.32 s	104.42 s	3.97
7168 (64)	55.82 s	52.21 s	7.48
7168 (64)	392.93 s	392.93 s	1.00
14336 (128)	203.29 s	196.46 s	1.93
28672 (256)	110.96 s	98.23 s	3.54
57344 (512)	73.48 s	49.12 s	5.35

nProcs (nodes)	Runtime (measured)	Runtime (ideal)	Efficiency
112 (1)	310.15 s	310.15 s	100%
224 (2)	319.39 s	310.15 s	97%
448 (4)	320.85 s	310.15 s	97%
896 (8)	325.30 s	310.15 s	95%
1792 (16)	327.13 s	310.15 s	95%
3584 (32)	333.40 s	310.15 s	93%
7168 (64)	337.42 s	310.15 s	92%
14336 (128)	341.77 s	310.15 s	91%
28672 (256)	354.12 s	310.15 s	88%
57344 (512)	375.41 s	310.15 s	83%

Bibliography

Bacchini, F. 2023, ApJS, 268, 60, doi: 10.3847/1538-4365/acefba

Lapenta, G. 2017, J. Comput. Phys., 334, 349, doi: 10.1016/j.jcp.2017.01.002

Markidis, S., Lapenta, G., & Rizwan-uddin. 2010, Math. Comput. Simul., 80, 1509, doi: 10.1016/j.matcom.2009.08.038