Mini-PIC: 2D Particle-In-Cell with tiling load balance

Advanced Topics in Computational Physics

Bianca Kolling Machado (98431)
João André Matias (103572)
Supervisor: Marija Vranic

Physics Department
Instituto Superior Técnico

April 2025

Abstract

The aim of this work was to develop a 2D PIC (Particle In Cell) code with tiling load balance capable of performing plasma simulations. The greatest focus was to take advantage of the MPI (Message Passing Interface) parallelizing structure to distribute the workload among computer cores while maintaining consistency in the spatial grid and in the time evolution of electric and magnetic fields. This was successfully achieved by building an MPI code based on tiling and following Yee's algorithm for field updates. Lastly, a numerical error analysis was conducted to assess the accuracy and stability of the code. The next step in this work would be to include particle species.

1 Introduction

The Particle-In-Cell (PIC) algorithm is a well-established computational technique used for simulating the self-consistent dynamics of plasmas and charged particles interacting with electromagnetic fields [1]. In the PIC approach, the simulation domain is discretized into a spatial grid, in which the electromagnetic fields, the particles, and the current densities are stored and evolved. The algorithm operates by alternating between the field and particle updates. At each time step, the fields are calculated by direct interpolation based on the position of each particle. This interpolation is necessary since the fields are only defined at discrete grid points while the particles move continuously through the space. The interpolated fields are then used to compute the Lorentz force acting on each particle, and the relativistic set of the equations of motion is integrated to update the positions and velocities of the particles. The updated particle positions and velocities are later used to calculate the charge and current densities, which are deposited back onto the grid through a second interpolation step. Using these updated quantities, the Maxwell equations are used to evolve the fields, completing one iteration of the PIC code. These processes are then repeated for the following time steps. This loop, illustrated in Fig. 1, defines the four key stages of the PIC algorithm: Field Interpolation, Particle Advance (Push), Current Deposition, and Field Advance.

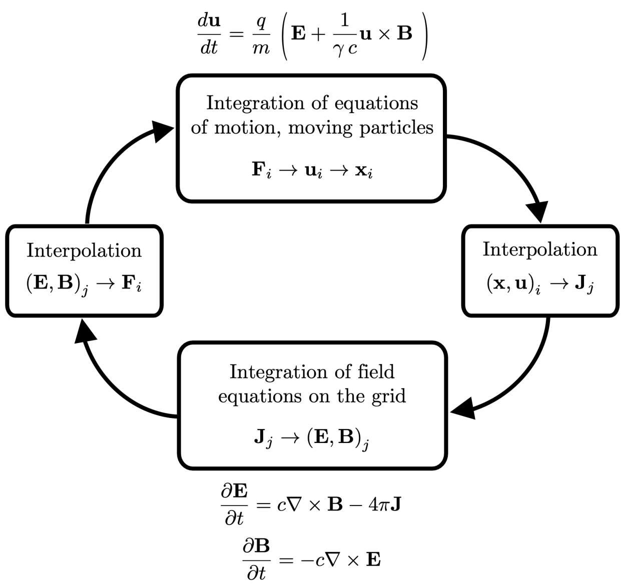

$\frac{d\mathbf{u}}{dt} = \frac{q}{m}\left(\mathbf{E} + \frac{1}{\gamma c}\mathbf{u}\times\mathbf{B}\right)$
Integration of equations of motion, moving particles
$\mathbf{F}_i \to \mathbf{u}_i \to \mathbf{x}_i$
Interpolation
$(\mathbf{x},\mathbf{u})_i \to \mathbf{J}_j$
Integration of field equations on the grid
$\mathbf{J}_j \to (\mathbf{E},\mathbf{B})_j$
Interpolation
$(\mathbf{E},\mathbf{B})_j \to \mathbf{F}_i$
$\frac{\partial \mathbf{E}}{\partial t} = c\nabla\times\mathbf{B} - 4\pi\mathbf{J}$
$\frac{\partial \mathbf{B}}{\partial t} = -c\nabla\times\mathbf{E}$

Figure 1: The four major steps of the PIC algorithm loop. The index i = 1, 2, ..., N denotes the particles, while the index j, which may be multidimensional, refers to the coordinates of the spatial grid. Image taken from [2].

By coupling the particles and the fields through the grid, this method avoids the need to compute every interaction between the particles. Instead, the interactions are contained in the field interpolations, which are influenced collectively by all particles. This reduces the computational complexity significantly, making these code structures highly efficient for large scale simulations. Additionally, many of the existing PIC codes, like OSIRIS [3], are parallelized using domain decomposition with MPI (Message Passing Interface) and/or OpenMP. This allows them to efficiently run on distributed memory architectures and high-performance computing systems.

The main goal of this project was to develop a two-dimensional (2D) PIC code with load balance based on tiles - spatial subdivisions of the simulation domain composed of a fixed number of grid cells. The code, written in C++, was designed to support periodic boundary conditions and aimed to incorporate key numerical techniques used in plasma simulations. Moreover, a central objective of this work was to gain experience with parallel programming using MPI, as well as numerical methods such as the Finite-difference time-domain (FDTD) and Leapfrog methods. Although the original aim of the project was to fully implement the complete PIC cycle (Fig. 1), the final implementation corresponds to an intermediate stage. Due to time constraints and the steep learning curve associated with numerical methods and parallel computing, the development was focused on the field solver component. The current version of the code, available on [4], successfully initializes the electromagnetic fields on the spatial grid and evolves them in time using the discretized Maxwell's equations. However, particle dynamics and current deposition have not yet been implemented. Despite this limitation, the project lays a solid foundation for future implementation of the full PIC algorithm.

This report is structured as follows: Section 2 presents a detailed description of the tiling strategy and the communication mechanisms implemented to ensure proper data exchange between the tiles, particularly in a parallel computing environment. Section 3 describes the field solver, including the discretization of Maxwell's equations and their integration within the main simulation loop using a FDTD scheme. Section 4 discusses the validation tests perform to ensure the correctness of the implementation, along with the corresponding results. Finally, Section 5 summarizes the main conclusions of the project and discusses potential directions for future work.

2 Tiling and Communication Mechanisms

2.1 Main Structure of the Code

This 2D PIC code employs a two-level domain decomposition strategy to distribute the workload across multiple MPI processes and provide flexibility for dynamic tile balancing. The global simulation domain is first divided among MPI ranks in a near-square layout, and each rank then subdivides its local region into multiple Tiles, also in a near-square arrangement. By keeping track of how many rows and columns of ranks and Tiles exist, each individual Tile is uniquely identified by (global Row, global Column). This allows us to define a unique global Tile ID (GID) in row-major order:

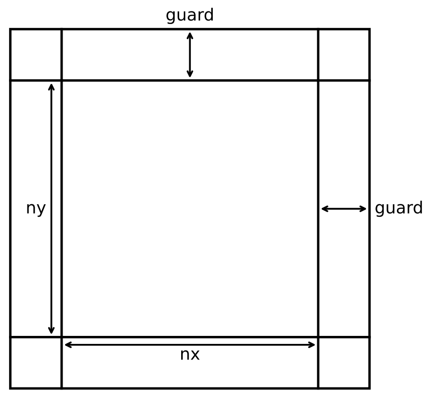
$$\mathrm{GID}(\mathrm{gR}, \mathrm{gC}, \mathrm{gTC}) = \mathrm{gR} \times \mathrm{gTC} + \mathrm{gC}\,, \tag{1}$$

where gR and gC denote the global Row and Column of the Tile, respectively, and gTC is the total number of global tile columns. By maintaining a consistent row-major mapping, each tile can be easily identified or located in the global domain, regardless of how it is distributed among different MPI ranks. The conversion from GID to global Row and global Column is done by

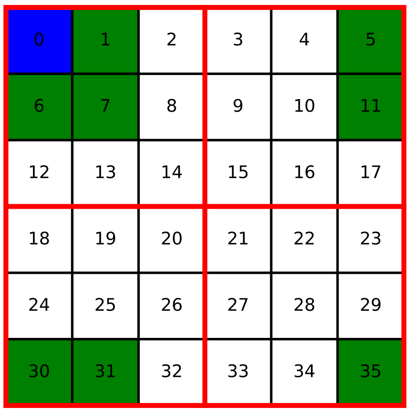
$$\mathrm{gR} = \mathrm{int}(\mathrm{GID}\ /\ \mathrm{gTC})\ \ ,\ \ \mathrm{gC} = \mathrm{remainder}(\mathrm{GID}\ /\ \mathrm{gTC})\,. \tag{2}$$

Each Tile of the physical domain holds an array of size $(\mathrm{nx} + 2 \times \mathrm{guard}) \times (\mathrm{ny} + 2 \times \mathrm{guard})$, where nx and ny refer to the interior cell counts (along the x and y directions, respectively), and guard denotes the number of guard cells (also known as ghost or halo cells) on each boundary (as illustrated in Fig. 2a). Thus, each Tile is assigned to a (C++) structure called TileInfo that stores its GID, global row and column, interior dimensions (nx and ny), and the current rank to which it belongs. Similarly, every rank has a RankInfo structure that contains information regarding its placement on the global rank grid (the rank's row and column), how many Tiles it holds, and which Tiles it holds.

Since every Tile has a unique GID, we use it to establish the communication between Tiles. Because every Tile has 8 neighbors (left, right, top, down, top left, top right, bottom left, and bottom right), the first step is to get their GIDs while ensuring a periodic wrapping. This means that if the neighbor is out of bounds, the code must be able to wrap around the opposite side of the spatial domain. Once a Tile knows the global ID of its neighbor, it must exchange boundary (guard cell) data with that neighbor via MPI. For this purpose, it is crucial to gather the subset of data that needs to be sent from one side (or corner) of a Tile to its neighbor in a specific direction. From the sketch shown in Fig. 2a, it follows that each Tile needs to pack 8 "blocks"(buffers) of guard cells. As an example, for the communication with the top and bottom left neighbors, blocks of dimensions nx $\times$ guard and guard $\times$ guard must be packed, respectively. This same logic applies in reverse when receiving data from neighbors: a Tile must also unpack each incoming buffer and use it to update its local boundary regions.



(a) Schematic representation of a single Tile. It contains an inner region of size nx $\times$ ny, plus an outer layer of guard cells. The total dimensions are then (nx + 2 $\times$ guard) $\times$ (ny + 2 $\times$ guard).



(b) Global indexing of Tiles in row-major order in a 2$\times$2 rank grid (from left to right, the first row contains ranks 0 and 1, and the second row contains ranks 2 and 3). Red lines denote rank boundaries, and each rank contains 3$\times$3 Tiles (black lines), each associated with a unique GID. The 8 neighbors of GID 0 are shown in green.

Figure 2: Overview of tile structure and domain decomposition.

To prevent message collisions in MPI, we generate a unique integer tag for each tile–direction pair. This way, messages from a particular GID and direction cannot be confused with any other messages, ensuring that every send and receive call properly matches up with its intended neighbor. Within each communication phase, the following steps occur for each Tile:

1. Post MPI_Irecv calls, one for each of the eight directions, to prepare for receiving boundary data from each neighbor;
2. Pack the local boundary blocks into buffers and send them to the corresponding neighbors using MPI_Isend;
3. Wait on all sends and receives (with MPI_Waitall) so that the data exchange completes;
4. Unpack the newly arrived buffers and update the Tile's boundary regions.

2.2 Examples and Tile Exchange Between Ranks

A succinct way to visualize this code structure is illustrated in Fig. 2b. In this example, the global domain is a 2×2 grid of ranks, each containing 9 Tiles, labeled from GID 0 (top-left corner) to 35 (bottom-right corner). For instance, the Tile with GID = 0 will exchange guard cell data with its eight neighbors, marked by the green colours, via periodic boundary conditions. Given that the code relies on a Tile's GID for all communication and neighbor identification, each Tile can be reassigned to a different rank at runtime without losing its identity. This design choice enables a straightforward implementation of dynamic load balancing.

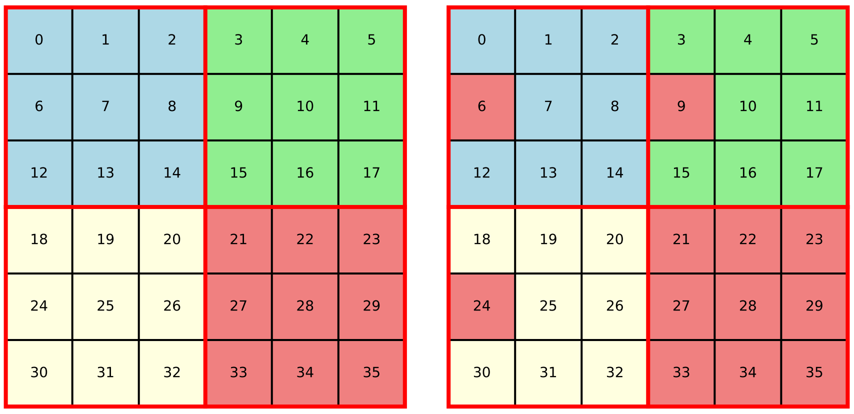


Figure 3: Example of Tile mitigation between ranks. Initially, each rank contains 9 Tiles marked by the same colours. Tiles with GID 6, 9, and 24 are later moved to rank 3 (at the bottom-right corner), maintaining their identities (GID and neighbours).

When a tile is migrated for load balancing, its GID and grid data are sent via MPI to the destination rank. The source rank updates the global owner array and removes the tile from its RankInfo, while the destination rank receives the data, reconstructs the tile (keeping its unchanged GID), updates its currentRank, and appends it to its own RankInfo. This process is depicted in Fig. 3.

3 Advancing the Fields using Maxwell's Equations

With a solid foundation in the code of discretized space, tiles, tile communication, and tile exchange among ranks, the next step is to implement electric and magnetic fields. In order to achieve this, Yee's model [5] was followed for field discretization and its time evolution. Yee's scheme is a numerical model widely used for simulating electric and magnetic fields through Maxwell's equations. It specifically explores the nature of Maxwell's equations and how they evolve in space and time to build a spatial grid that naturally satisfies Gauss' laws and avoids numerical instability. It also adapts the finite difference time evolution calculation using the leapfrog method, ensuring that Maxwell's equations are solved self-consistently.

3.1 Spatial Grid

In this scheme, a staggered grid is used in which the electric field and the magnetic field are set at different points of the spatial domain. This allows the central differences for spatial derivatives to be calculated directly with neighboring values without the need for field interpolation. In Yee's original publication, a 3D model was set with an image of a cubic cell showing where each field point would be (electric field points at the cell edges and magnetic field points at cell faces). In the work here developed, a 2D model was chosen instead, and some field points were switched, but still maintained self-consistency. A schematic drawing of a unitary square cell is shown below, in which point (i,j), which corresponds to the cell origin, is located in the lower left corner.

This spatial arrangement had to be considered for the spatial and time derivatives, as well as later on in the code when there needed to be a connection between the scheme and the real physical domain.

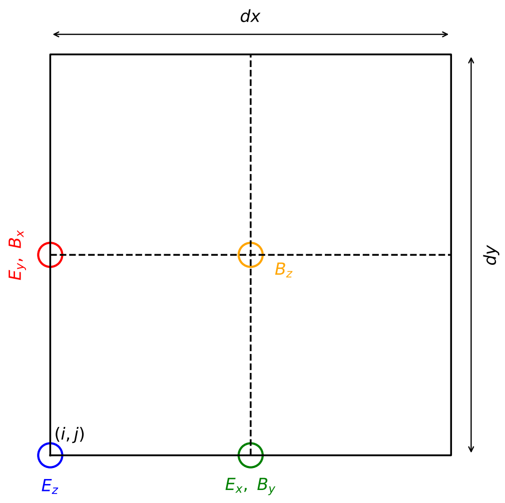


Figure 4: Schematic representation of each field component on a grid cell according to Yee's layout.

3.2 Maxwell's Equations

The evolution for this system is described by Maxwell's equations. Since the goal was to make a simulation in two dimensions (x and y) of a three-dimensional system, the usual expansion in three dimensions is simplified when the derivatives in "z"go to zero. The equations that then describe the system are listed below, using $B = \mu H$, $D = \epsilon E$.

$$\frac{\partial B_x}{\partial t} = -\frac{\partial E_z}{\partial y} \ , \frac{\partial B_y}{\partial t} = \frac{\partial E_z}{\partial x} \ , \frac{\partial B_z}{\partial t} = \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}$$

$$\frac{\partial D_x}{\partial t} = -\frac{\partial H_z}{\partial y} - J_x \ , \frac{\partial D_y}{\partial t} = -\frac{\partial H_z}{\partial x} - J_y \ , \frac{\partial D_z}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - J_z$$

These equations were then discretized through finite difference, also following Yee's algorithm.

3.3 Leapfrog Method

When dealing with two connected fields, like the electric and magnetic fields, one can advance them in time in a simulation through the leapfrog method. It consists of advancing each field one at a time, usually involving evolution in half-time steps. In this work, the magnetic field was always evolved in time first, in one half-time step. Then, the electric field went through a full-time step evolution, followed by another half-time step evolution of the magnetic field. In the code, this all happens in one time loop, ensuring that both fields advance one full time step at the end of each time iteration. To illustrate how this works in the algorithm, bellow are two time evolution equations that also account for the staggered grid in the indices i and j.

$$B_x^{n+1/2}(i, j+1/2) = B_x^{n-1/2}(i, j+1/2) - \frac{\Delta t}{\Delta y}(E_z^n(i, j+1) - E_z^n(i, j))$$

$$D_x^n(i+1/2, j) = D_x^{n-1}(i+1/2, j) - \frac{\Delta t}{\Delta y}(H_z^{n-1/2}(i+1/2, j+1/2) - H_z^{n-1/2}(i+1/2, j-1/2)) - \Delta t J_x^{n-1/2}$$

Finally, since in this case we are dealing with an MPI code in which the guard cells are responsible for field propagation between tiles, the guard cells need to be carefully updated as well. So, the full-time update is as follows:

1. Half time step $\vec{B}$ update
2. Update guard cells in $\vec{B}$ grid
3. Full time step $\vec{E}$ update
4. Update guard cells in $\vec{E}$ grid
5. Half time step $\vec{B}$ update
6. Update guard cells in $\vec{B}$ grid

4 Tests and Benchmarks

In order to validate our PIC code's Field Solver implementation, we carried out a set of controlled tests using analytically defined electromagnetic waveforms that satisfy Maxwell's equations. These tests included both sinusoidal waves and localized pulses, allowing us to assess field initialization, propagation, and the code's ability to handle different wave geometries. The test cases were:

- **Test 1:** Electromagnetic sinusoidal wave propagating along $\hat{\boldsymbol{x}}$:

$$\begin{aligned} E_x &= A \cdot \sin(k_y \cdot y) \\ E_y &= 0 \\ B_z &= -A \cdot \sin(k_y \cdot y) \end{aligned}$$

- **Test 2:** Obliquely propagating Electromagnetic sinusoidal wave in the xy-plane:

$$\begin{aligned} E_x &= \frac{A}{\sqrt{2}} \cdot \sin(k_x \cdot x + k_y \cdot y) \\ E_y &= -\frac{A}{\sqrt{2}} \cdot \sin(k_x \cdot x + k_y \cdot y) \\ B_z &= -A \cdot \sin(k_x \cdot x + k_y \cdot y) \end{aligned}$$

- **Test 3:** Localized Electromagnetic pulse with cosine-squared envelope varying slowly coupled with the wave frequency propagating in $\hat{\boldsymbol{x}}$:

$$\begin{aligned} E_x &= 0 \\ E_y &= A \cdot \sin(k_x \cdot x) \cdot \cos^2\left(\frac{x - x_c}{\tau} \cdot \frac{\pi}{2}\right) \cdot \mathcal{H}\left(1 - \left|\frac{x - x_c}{\tau}\right|\right) \\ B_z &= A \cdot \sin(k_x \cdot x) \cdot \cos^2\left(\frac{x - x_c}{\tau} \cdot \frac{\pi}{2}\right) \cdot \mathcal{H}\left(1 - \left|\frac{x - x_c}{\tau}\right|\right) \end{aligned}$$

Every other field component was initialized as zero. Here, A denotes the wave amplitude, and k_x and k_y represent the wave vectors along the x and y directions, respectively. Additionally, x_c and y_c correspond to the center of the pulse in space, and τ symbolizes the pulse width. Moreover, the Heaviside function $\mathcal{H}$ ensures that the pulse is spatially localized. As mentioned earlier, the field components are staggered in space according to the Yee grid shown in Fig. 4, meaning that different components are defined at different spatial positions within each cell. For this reason, each field component is initialized based on its unique grid. Each grid has its own set of x and y coordinates (local to each grid).

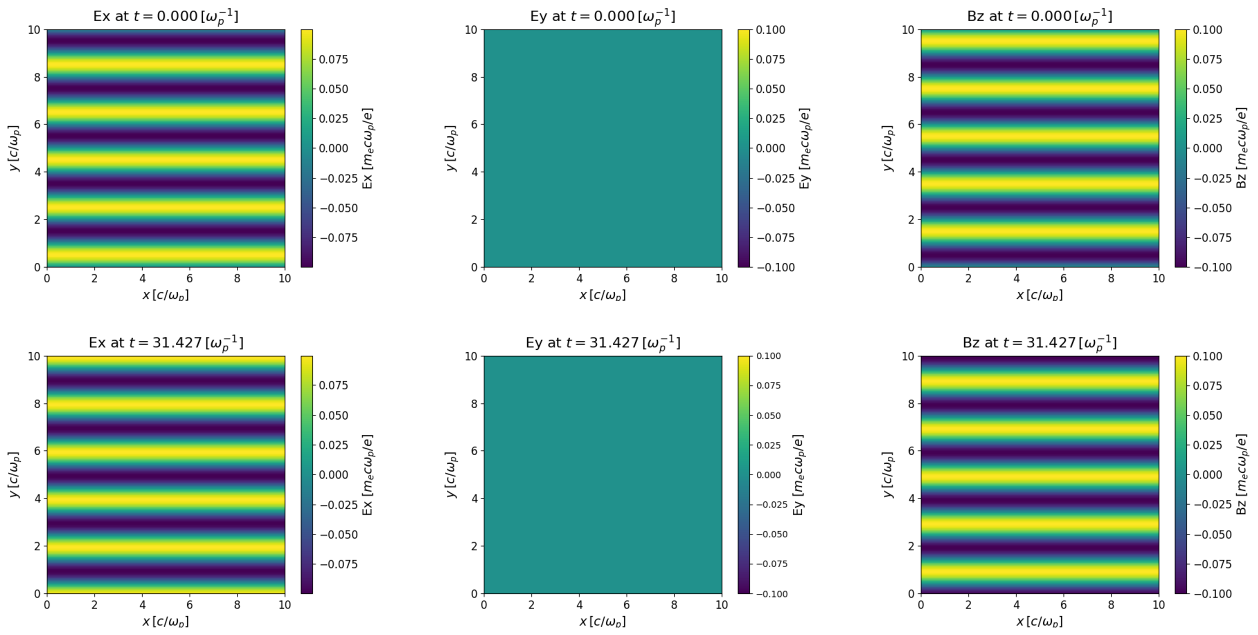


Figure 5: Test 1: Electromagnetic sinusoidal wave propagating along the y direction.

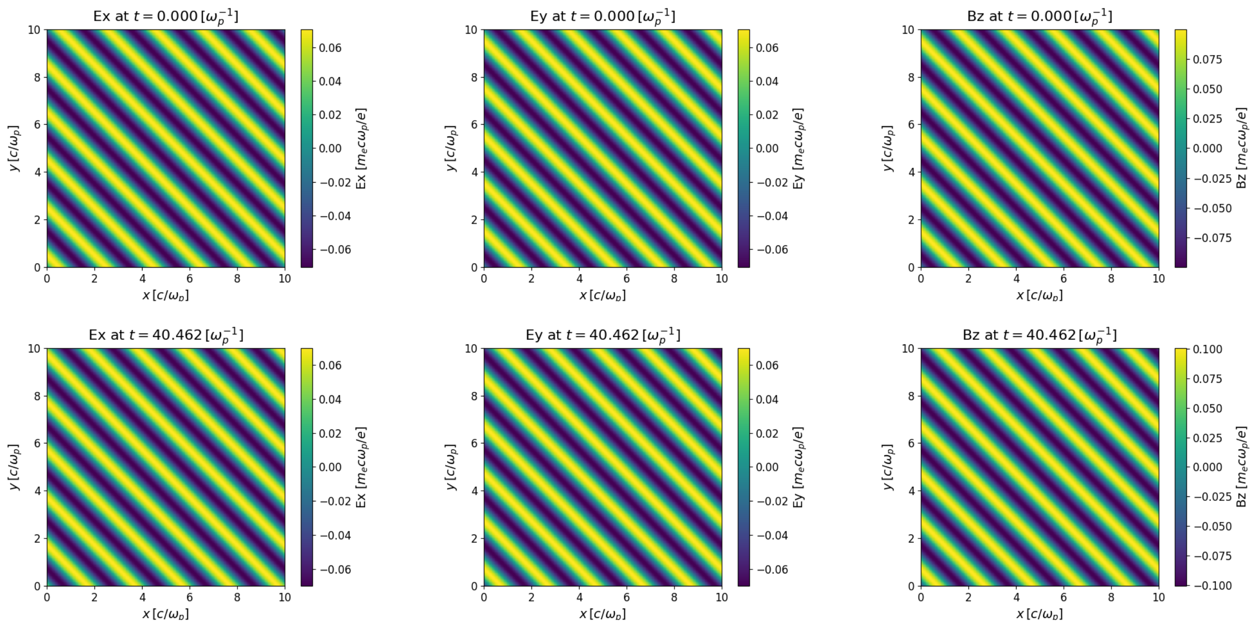


Figure 6: Test 2: Electromagnetic sinusoidal wave propagating along the x and y direction.

The simulations were carried out using 9 CPU (Central Processing Unit) cores, with each core (MPI rank) initially assigned 36 Tiles arranged in a 6×6 grid. The physical size of the simulation box is $10 \times 10\,(c/\omega_p)^2$, where c denotes the speed of light and ω_p is a normalization frequency (which may later be specified as a plasma or laser frequency). To discretize the domain, we used 450 cells in each spatial direction, resulting in Tiles containing 25×25 cells. Each Tile also includes 2 guard cells per direction to handle communication with its neighbor Tiles and boundary conditions. Furthermore, the electromagnetic fields were initialized using $A = 0.1$, $k_x = k_y = \pi\,[\omega_p/c]$, $x_c = y_c = 3.5\,[c/\omega_p]$, and $\tau = 3\,[c/\omega_p]$. Finally, the simulation was run until $t = 50\,[\omega_p^{-1}]$ with a time step $dt = 0.5 \times dt_{CFL}$, where

$$dt_{\mathrm{CFL}} = \frac{1}{\sqrt{\frac{1}{dx^2} + \frac{1}{dy^2}}}, \tag{3}$$

thus satisfying the Courant-Friedrichs-Lewy [6] condition for numerical stability. At the fifth step ($t \approx 0.04\,[\omega_p^{-1}]$), we performed a manual Tile exchange: ranks 0, 1, and 2 sent the Tiles with GID 3, 9, and 15, respectively, to rank 3.

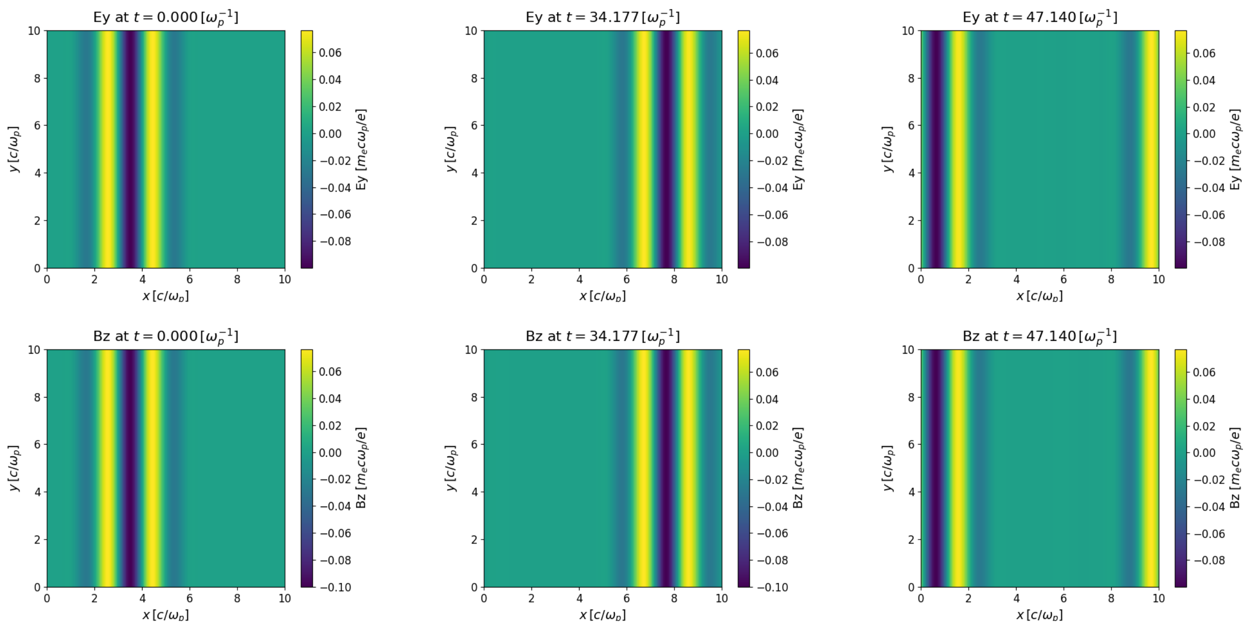


Figure 7: Test 3: Electromagnetic pulse propagating along the x direction.

The results of the three tests are shown in Figs. 5 - 7. They confirm the correct implementation of electromagnetic wave propagation in our PIC code. In all cases, the injected waves evolve as expected from Maxwell's equations, with consistent behavior across all field components. These outcomes validate the correctness of the field update routines, guard cell communication, periodic boundaries, and tile exchange mechanisms.

To further test the accuracy and stability of the Field Solver, we performed two long duration ($t = 500\,[\omega_p^{-1}]$) propagations of the pulse described in Test 3. On the first instance, we used nx = ny = 450, whereas on the second one, we utilized nx = ny = 720 (note that this means that dx and dy change, also impacting the value of $dt = 0.5 \times dt_{CFL}$). The remaining parameters were the same as the ones used in the previous tests, and an identical manual Tile exchange was performed. The results of both runs are shown in Figs. 8 and 9. In both cases, the left side plots illustrate lineouts of the B_z component of the electromagnetic pulse from Test 3, taken at $y = 5\,[c/\omega_p]$ for selected simulation steps. Each curve corresponds to a different time, arranged from left to right in increasing temporal order. The pulse structure consists of two main peaks separated by a central minimum. Over time, the amplitude of the first peak increases slightly over time, while the second peak gradually decreases, with the central valley remaining nearly unchanged. This behavior illustrates the numerical effects that result from numerically resolving for the speed of light c. The plots on the right provide a more quantitative perspective on each peak's amplitude variation over time. The key observation is that, as expected, the increase of the spatial resolution led to reduced numerical errors.

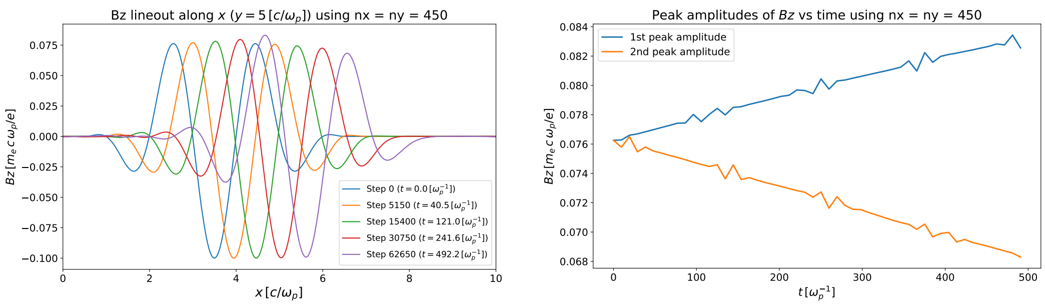


Figure 8: Lineouts of the B_z component of the electromagnetic pulse from Test 3, taken at $y = 5\,[c/\omega_p]$ for selected simulation steps. Each curve corresponds to a different time, arranged from left to right in increasing temporal order. Right: Sketch of the amplitudes of the two peaks of the pulse as functions of the simulation time. Both images correspond to the case where nx = ny = 450.

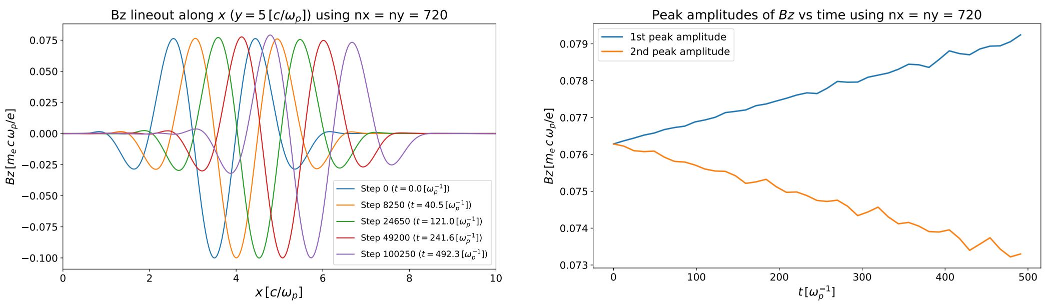


Figure 9: Lineouts of the B_z component of the electromagnetic pulse from Test 3, taken at $y = 5\,[c/\omega_p]$ for selected simulation steps. Each curve corresponds to a different time, arranged from left to right in increasing temporal order. Right: Sketch of the amplitudes of the two peaks of the pulse as functions of the simulation time. Both images correspond to the case where nx = ny = 720.

To investigate whether the observed numerical errors could be attributed to numerical dispersion, we conducted a final test. Using the data from the run with nx = ny = 450, we analysed the propagation of the pulse's first peak. Fig. 10, displays the position of the pulse's first peak (unwrapped to account for the periodic boundary conditions) as a function of the simulation time. A linear fit of the data yields $x_{\text{1st\_peak}} = 0.9977\,t + x_c$, indicating a nearly constant propagation velocity. To compare this with the theoretical expectations, we also computed the velocity predicted by the numerical dispersion of the FDTD scheme for a wave propagating along x:

$$v_{\text{numerical}} = \frac{\omega}{k_x} = \frac{2}{\Delta t}\arcsin\left[\frac{c\,\Delta t}{\Delta x}\sin\left(\frac{k_x\,\Delta x}{2}\right)\right] = 0.99982\,[c]. \tag{4}$$

The close agreement between the theoretical and simulation values confirm that the small errors observed in Figs. 8 and 9 are consistent with the inherent dispersion of the discretization scheme.

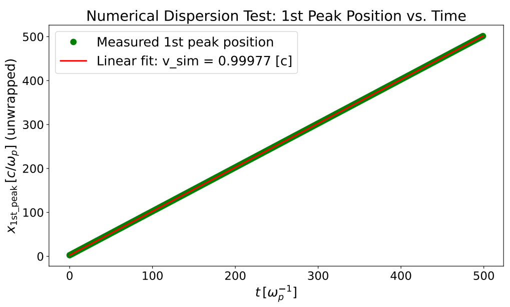


Figure 10: Position of the pulse's first peak as a function of the simulation time, used to fit the propagation velocity of the pulse.

5 Conclusions

In this study, an MPI-based Field Solver was developed, and the main goal was met: to build a fully functioning parallelized code based on dynamic tiling that is capable of working seamlessly when computing electric and magnetic fields. In the time frame available for this work, introducing particles and current deposition for a full PIC code was not possible. However, with the working structure that was achieved, expanding the code for particles should come easily and with minimal requirements on editing the parallelization.

The code works by distributing tiles across different computer cores, with each tile containing a set number of cells as well as guard cells for communication. Space is discretized by the cells and information exchange between tiles occurs through the guard cells. The user can alter the number of cores to be used, as well as the number of cells in each tile, altering the resolution of the simulation. The electric and magnetic fields are also discretized in a stagerred grid, meaning the code works with six different grids: one for each spatial component (x,y,z) of each field. This allows for a direct application of central difference finite difference methods for spatial and temporal derivatives, following Yee's algorithm for Maxwell's equations. To check the validity of the code, a few tests for electromagnetic waves were conducted and showed physically consistent results. Additionally, tests for numerical stability were performed and showed increasing numerical instability for increasing simulation time. It was also shown that higher resolutions lead to lower numerical instability, as expected, and that the observed errors are consistent with the numerical dispersion inherent to the discretization scheme.

Finally, future work includes the implementation of particle dynamics and current deposition routines. Once the full PIC algorithm loop is implemented, new dynamic load balance criteria can be tested. One promising idea is to utilize the MPI_Wtime temporal counter to order the ranks in a list by ascending "work time". This way, the ranks that are overloaded with particles can dynamically send Tiles to the ranks with lower computational loads, improving overall efficiency and minimizing idle time across the simulation domain. Finally, it would be important to perform further benchmark tests and comparisons with other PIC codes, such as Osiris [3].

References

[1] C.K. Birdsall and A.B. Langdon. *Plasma Physics via Computer Simulation.* Taylor & Francis, 1991.

[2] Eduardo Paulo Jorge da Costa Alves. Magnetic field generation via the kelvin-helmholtz instability. Dissertação de mestrado, Instituto Superior Técnico, Lisboa, Portugal, 2010.

[3] R. A. Fonseca et al. Osiris: A three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators. In *Lecture Notes in Computer Science, vol. 2331*, pages 342–351. Springer, 2002.

[4] Bianca Machado André Matias. Mini-PIC: 2D PIC code with tiling and load balancing. `https://github.com/JACM57/Mini-PIC--2D-PIC-code-with-tiling-load-balance`, 2025. Accessed: 2025-04-02.

[5] K. S. Yee. Numerical Solution of Initial Boundary value Problems Involving Maxwell's Equations in Isotropic Media. *IEEE Transactions on Antennas and Propagation*, Ap-14, n. 3:303–307, May 1966.

[6] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100:32–74, January 1928.