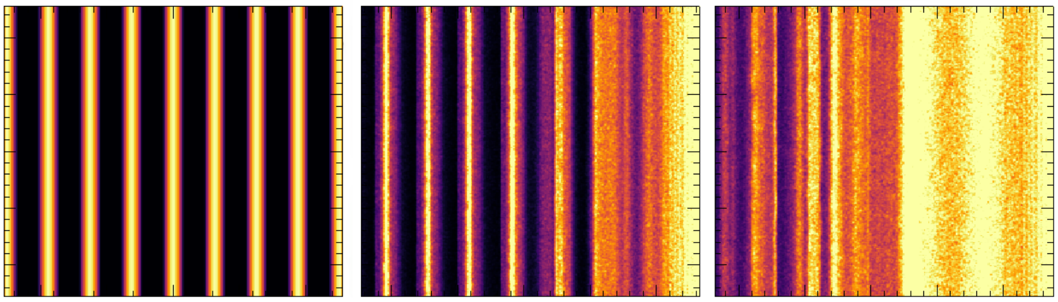
UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO



# Implementation and Development of a Workflow for Simulation Steering and Runtime Visualization of OSIRIS Simulations in HPC Environments

Luís Fernando Mendonça de Sousa Nóbrega
Licenciatura em Engenharia Física Tecnológica

PIC 1 Scientific Project

**Orientador:** Luís Oliveira e Silva
**Co-orientador:** Thales Silva

Julho de 2025

**Title:** Implementation and Development of a Workflow for Simulation Steering and Runtime Visualization of OSIRIS Simulations in HPC Environments

**Abstract:**

The use of high-performance computing (HPC) environments enables computationally intensive particle-in-cell (PIC) simulations, with codes like OSIRIS, to run at practical time scales. In this work, we develop new simulation steering and runtime visualization methods that improve simulation control and enable *in-situ* data analysis directly within HPC systems. Specifically, we demonstrate two efficient approaches for handling large data volumes using forward-porting techniques in conjunction with Singularity containers and Jupyter Notebooks, tested on the Deucalion supercomputer. Furthermore, we implement into OSIRIS options that allow runtime parameter adjustments without halting simulations, thus enabling dynamic optimization of simulation parameters, adaptive diagnostics, and corrections to plasma distributions—significantly improving the computational efficiency and reducing resource waste. These changes primarily modify diagnostic production but also extend to other aspects, such as real-time adjustment of plasma density profiles and beam parameters.

**Keywords:** HPC, Simulation Steering, Runtime Visualization, OSIRIS, Scientific workflows

# CONTENTS

# CHAPTER 1

# INTRODUCTION

## 1.1 HIGH-PERFORMANCE COMPUTING ENVIRONMENTS

HPC is a technology that utilizes clusters of processors working in parallel to handle large datasets and solve complex computational problems at significantly faster speeds than conventional computers [1]. While a typical laptop can perform billions of calculations per second, HPC systems—particularly supercomputers—can execute up to quintillions (exaflop-scale). Most modern supercomputers employ distributed-memory architectures, where computational nodes - each containing multiple processor cores with local RAM - are associated. Nodes are interconnected via a high-speed communication network, enabling data exchange and coordinated parallel processing. Two examples include the MareNostrum5 [2, 3] supercomputer in Barcelona with a peak performance of 45.9 PFlops or Deucalion in Minho [4, 5] that has around 10 PFlops. These are currently $14^{th}$ and $197^{th}$ in the top 500 list of the largest supercomputers [6]. The simulations of this project were performed in Deucalion's ARM nodes (Fig. 1.1) and compatibility testing was done in MareNostrum5 x86 nodes.

Most computer programs work by dividing a workload into a sequence of tasks and running them sequentially on a single processor. This is not ideal for large-scale complex problems, since it limits computational speed. In contrast, parallel computing consists in dividing a problem into smaller and independent tasks that can be processed simultaneously across several processors. In an environment with a large quantity of processors, much faster execution is possible. HPC environments are designed for the parallel computing, as they often consist of thousands of processors, making them ideal for data-intensive applications. When software is well parallelized, it can utilize the full potential of an HPC environment. One of the ways to do this is to rely on a standard library and protocol for parallel computing that enables processor communication such as message passing interface (MPI) [7, 8].

Given the increasing demand for HPC environments to support powerful computational models and generative AI tools, node allocation has become a competitive task. Re-submitting incorrect jobs results in longer wait times due to queue delays. Developing tools to prevent such issues is essential for researchers. Additionally, the large volumes of data typically stored in Hierarchical Data Format (HDF) [9, 10, 11] pose challenges for exporting to the local researcher devices. Thus, there is a need for methods to process this data directly within the environment.

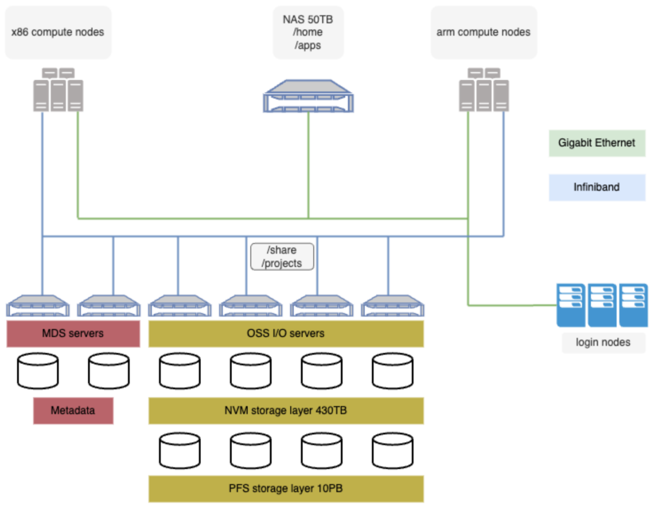


FIGURE 1.1: Deucalion architecture. There are 1632 ARM nodes, 500 x86 nodes as well as 33 GPU nodes. It runs on Rocky Linux OS, uses Singularity as the container system and slurm for resource manager and scheduling. From [4]

## 1.2 INTERACTIVE TOOLS IN HPC

Recent EU initiatives have advanced shared HPC infrastructure [12, 13] to unify cluster environments and software. With increasing architectural diversity across EuroHPC systems and limited support resources, the European Environment for Scientific Software Installations (EESSI) [14] has seen widespread adoption. EESSI harmonizes software across EuroHPC systems through a layered approach: a shared module system resolves dependencies, virtual environments enable user customization, and containerization (via Singularity/Apptainer) ensures reproducibility. This eliminates cross-site compatibility for users of different HPC environments.

This increasing homogeneity lends itself to the development of tools for optimizing user experience and cross-cluster compatibility. Without native support for specific applications, the optimal practice is to encapsulate the software within a compatible container. Containers, being lightweight, portable virtual environments, enable reproducibility across different systems [15]. For instance, if the environment uses Intel compilers and some software isn't compatible or available, running it in a container eliminates compatibility issues.

A similar problem motivated part of this project. OSIRIS is a fully relativistic, electromagnetic particle-in-cell (PIC) code for kinetic plasma physics simulations [16, 17]. This PIC code provides comprehensive diagnostic capabilities, enabling detailed output of simulation data - including charged particle distributions, electromagnetic fields, and plasma currents.

These diagnostics are essential for studying diverse physical systems, from astrophysical plasmas [18] to high-intensity particle and laser beam–plasma interactions [19], where reproducing extreme laboratory conditions would be either prohibitively expensive or physically impossible.

To process this HDF-formatted data, researchers rely on visXD, an internally developed visualization suite [20]. However, since VisXD requires Interface Description Language (IDL) - unavailable on systems like Deucalion and MareNostrum5 - researchers have to download the data locally. During my internship, I experienced first-hand how this created bottlenecks: transferring multi-gigabyte datasets at 15-50 Mbps often took hours to complete. This led to the idea of containerizing the software to simplify researchers' workflows. Although Docker [21] is the most widely used container platform, we opted for Singularity [22, 23], as it is the standard in EESSI.

Despite innovation in remote GUI solutions in EuroHPC, such as Open OnDemand[24], graphical interfaces are not yet supported universally on other HPC systems. As a result, web-based utilities such as Jupyter Notebooks[25] remain essential for interactive data exploration. Notebooks follow a server-client model and require computational back-ends. By reserving compute nodes exclusively for the processing of data, users can execute notebooks while capitalizing on HPC resources.

## 1.3 Steering in OSIRIS

OSIRIS simulations require an input-deck specifying particle parameters, boundary conditions, and resolution [19, 26]. While restarts with modified name-lists are possible, key parameters like plasma frequency remain fixed during execution. Runtime steering could significantly reduce queue wait times for large node allocations, saving hundreds of computational hours.

For uninterruptible large-scale simulations, the ability to adjust parameters such as diagnostic frequency or the scheduling of restart files at runtime is highly beneficial. This enhancement would open a new pathway for runtime control, enabling adaptations to simulation behaviour in response to data as it is being produced. It would also of relevance for data driven/machine learning applications.

## 1.4 Goals of this work

This work has two primary objectives aimed at plasma physics in HPC environments:

1. **Real-time Data Visualization**: Develop a solution for generating real-time diagnostic visualizations directly from simulation data while processing is running on HPC systems instead of the researcher's local machines. This eliminates the need for transferring large datasets by enabling visualization through either an Interactive command-line prompt (visXD) or a Web-based interface (Jupyter notebook);

2. **Dynamic Simulation Control**: Implement runtime steering capabilities for OSIRIS PIC simulations, including: On-demand simulation halting and restarting and *In-situ* modification of physical parameters, including previously restricted quantities. This will enable simulation details to be optimized at runtime, saving thousands of CPU-hours in the process.

Both these advancements aim to significantly simplify the user's workflows and expedite research as well as open a door for a robust steering method in OSIRIS.

# CHAPTER 2

# METHODS AND RESULTS

## 2.1 REAL-TIME DATA VISUALIZATION

### 2.1.1 VISXD CONTAINERIZATION

In a laptop with an intel 7 ultra compiler, compatible with x64-86 architecture, a Docker container was created having Ubuntu:22.04 as the operating system (OS). Even tough the final container will be deployed in Singularity form as described before, the installation was done in Docker due to how much easier it is to run a container interactively. Singularity is commonly chosen in HPC environment due to security concerns, as unlike Docker, Singularity does not require root privileges to run containers [22] which restricts escalation to host privileges.

Several packages were incorporated to allow for compatibility handling HDF [11] files and ssh applications (e.g. forward porting and python dependencies). An isolated user was added under the name **swe** as to circumvent the inevitable mounting of the host directory when converting the docker container into Singularity.

After interactively entering the environment, IDL 7.1 version was installed, along with its dependencies [27]. After manually transferring the VisXD source files and setting environment variables to link the IDL and VisXD installation paths, the container was saved and converted into a Singularity compatible form (.sif) [28] and transferred to the HPC environment. A bash script **graphics.sh** was also developed (Appendix A) to automate the activation of the container.

The process of setting up and using the interface is shown in Fig. 2.1 and as follows:

1. Login into the HPC environment using a Secure Shell (SSH) with the -X option (ex: *ssh -X Deucalion*) to forward the local graphical interface (ex: X11) to the HPC login nodes;

2. Guarantee that the image file (**.sif**) as well as **graphics.sh** file is present in the same folder;

3. Enter the container in interactive mode by using **./graphics.sh container_name**;

4. Type the wanted VisXD command, such as **idl vis2d**;

5. Use the interactive window normally to process data. The refresh rate is limited by the internet connection.

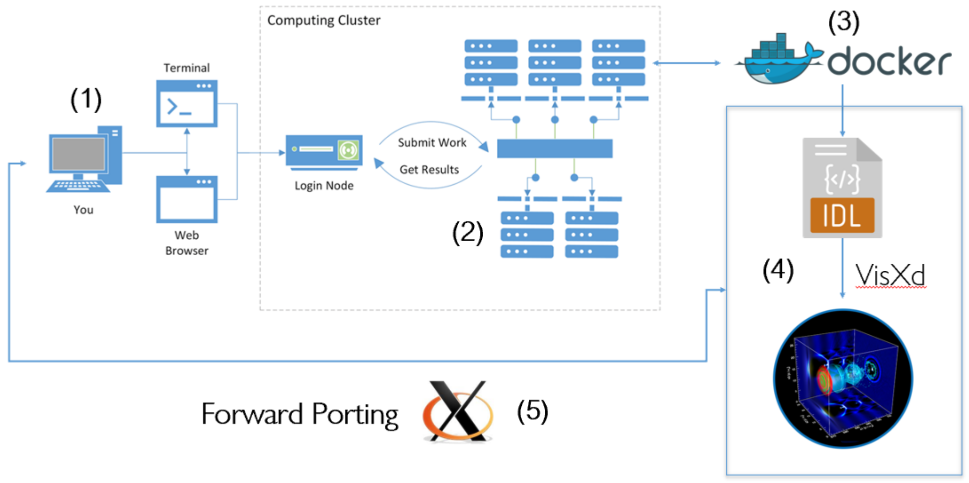


FIGURE 2.1: Schematic description of the algorithm behind using the visualization tool. Each of the steps 1-5 is described above. Figure adapted from: https://hpc.uwec.edu/about/what-is-high-performance-computing

A detailed user/installation guide is provided in Appendix A and a small file description in Table 2.1. For quick data analysis it proved to be very effective, as it allowed for a quick and easy way of analysing diagnostics as the simulation was running. This is useful as it allows to take decisions on running the simulation or making changes. The tool was 2-3 times slower than the locally installed version, due to slowness while connecting to login nodes but it proved to be much faster than downloading large amounts of data. For small data, there was not always an advantage, and we thus determined that the prime uses are for run-time heavy data production runs.

This approach was designed for using the login nodes for convenience, as analysing data is a lightweight operation. However, login nodes can become crowded and some servers are very strict with its usage. This problem can be addressed by allocating a node compatible with x86 architecture and redirecting the graphical interface connection to the node as described in Appendix A. For other architectures, such as ARM, the solution (yet to be tested) is to convert the Docker container into Singularity inside the ARM node and repeat the steps described above.

The implementation of this tool demonstrated significant impact, with researchers reporting improved workflow efficiency and substantial time savings in data analysis tasks. User feedback indicated unanimous approval of the new methodology.

TABLE 2.1: Scripts and files used in the containerization and visualization workflow

| File/Script Name | Function |
|---|---|
| `Dockerfile` | Defines base image and installs packages (e.g., IDL 7.1, HDF5) for container build. |
| `graphics.sh` | Forwards graphical interface (X11) to container environment for VisXD usage. |
| `Image.sif` | Finalized Singularity container (converted from Docker). |

### 2.1.2 JUPYTER NOTEBOOK SERVER HOSTING

Given that not all servers have an IDL collective license and some impose script restrictions on login node usage, containerization may not always be feasible. However, most EuroHPC servers support virtual environments for installing dependencies, such as Python packages [12, 13]. This allows hosting a Jupyter server on an allocated HPC node, enabling interactive notebook execution [25].

A Bash script, **Run_venv.sh** (Appendix A), was developed to initiate an interactive queue using the **salloc** method (Table 2.2). It loads a Python module (Fig. 2.2), creates/activates a virtual environment, and installs Jupyter, numpy, matplotlib, and ipykernel. The latter is essential for defining the notebook architecture—in this case, ARM, as Deucalion was used for testing [4]. The Jupyter notebook is then hosted on port 8888 within the virtual environment. Although the notebook is openly hosted, security remains intact as long as the node remains exclusive (which is typically the case).

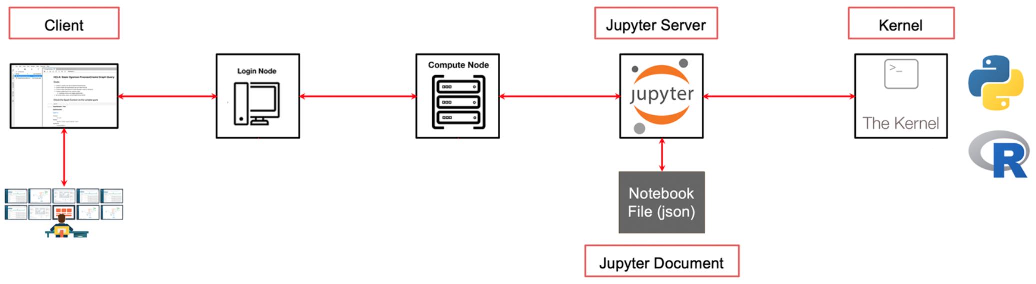


FIGURE 2.2: Illustrative scheme of Jupyter server hosting. The server runs on an allocated node, which communicates with the login node and the user's computer. The notebook can then be accessed via a browser. Edited from: https://threathunterplaybook.com/tutorials/jupyter/introduction.html

A second script, **Tunnel_venv.sh** (Appendix A), facilitates connecting a local machine to the allocated node (Table 2.2). This allows access to the execution directory of the script by entering *http://localhost:8888* in a browser or clicking the pop-up window in Visual Studio Code. The browser interface functions like a standard Jupyter notebook, making it far more practical for debugging and quick analysis than converting notebooks to Python scripts for cluster execution.

The workflow is as follows:

1. On the HPC: Execute **./Run_venv.sh** with the parameters described in Appendix A. This initializes the virtual environment and starts the server;
2. Locally: Adjust the parameters in **Tunnel_venv.sh** and run it from the local machine;
3. Open a browser and navigate to **http://localhost:8888**;
4. Terminate the connection with **Ctrl + C** when finished.

While some EuroHPC clusters already support browser-based tools like Jupyter, MLflow, TensorBoard, and RStudio via Open OnDemand [12, 24], this solution remains a convenient alternative for systems where such features are still unavailable.

TABLE 2.2: Scripts and files used in the Jupyter notebook workflow

| File/Script Name | Function |
|---|---|
| `Run_venv.sh` | Initiates an exclusive node session using `salloc`, sets up a Python virtual environment, and launches a Jupyter server on port 8888. |
| `Tunnel_venv.sh` | Establishes an SSH tunnel from the local machine to the allocated cluster node, enabling browser access to the Jupyter notebook interface. |

## 2.2 Dynamic Simulation Control

The steering module enables runtime parameter adjustments in OSIRIS without halting simulations or relinquishing HPC resources. A general visualization of the operating method is depicted in Fig. 2.3. Key features include:

— **Dynamic Control**: Modify frequency of diagnostics, simulation duration (**tmax**), or trigger checkpoints;

— **HPC Resource Retention**: Maintains node allocation during steering operations;

— **Fail-Safe Mechanism**: Some critical errors trigger restart file creation before exit while others are ignored.

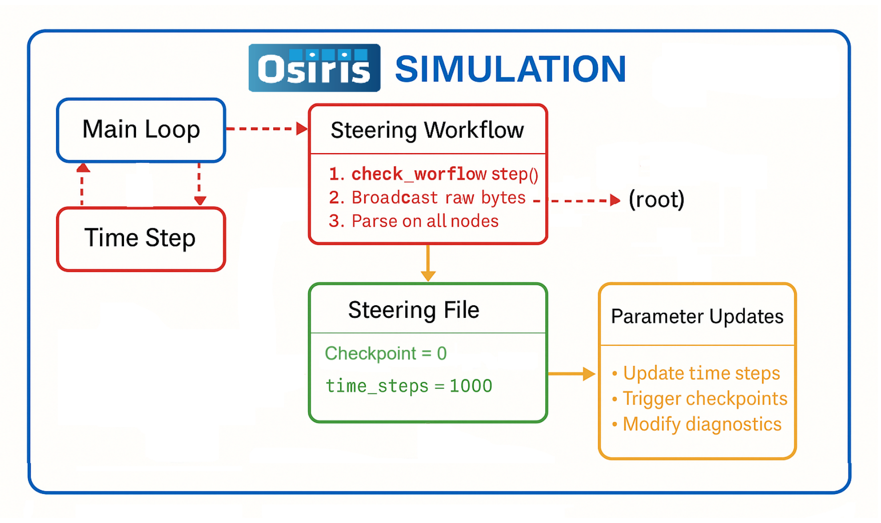


FIGURE 2.3: Steering implementation scheme. After the user defines a steering frequency, OSIRIS checks the existence of the steering file through the root node, and, if readable, broadcasts the information as raw bytes. This is done to prevent using complex MPI structures that may not be supported by every compiler. Each node then parses the information and updates parameters based on the given instructions. Figure made with Chatgpt-4o model.

### 2.2.1 Source Code Modifications

Two new modules (Table 2.3) were added to ensure proper parsing, initialization and modification of simulation variables.

It was decided to separate the two such that **m_workflow.f03** held all processes that directly changed simulation paramethers and **m_workflow_reader.f03** contained all the parsers and data type converters. As the number of functionalities grow, **m_workflow.f03** will eventually be divided into smaller files, each tasked with a part of the changes, such as diagnostics, phase spaces, etc. Some existing files, such as **os-main.f03** and **os-tstep.f90** received minor changes, for example, to allow for the user to submit the steering frequency in the input-list. An input list in OSIRIS is a structured text file defining simulation parameters in mandatory/optional sections. All changes were identified with a !*! symbol, which can be found by entering the source directory and executing **m_workflow.f03**. For more detailed information on the modules, refer to the user/developer guide in Appendix A.

TABLE 2.3: Steering modules and descriptions

| Module | Functionality |
|---|---|
| **m_workflow.f03** | Workflow control, command execution, diagnostics handling |
| **m_workflow_reader.f03** | Steering file parsing, key-value storage, diagnostics decoding |

### 2.2.2 WORKFLOW ARCHITECTURE

The main module consists of three types of routines, responsible for detecting the user input, parsing the information and conducting changes. There is also a series of dummy subroutines, responsible for intermediate processes of these routines (Table 2.4):

TABLE 2.4: Key routines for the workflow module

| Routine | Function |
|---|---|
| **check_workflow_step()** | Orchestrates file checks and MPI broadcast operations at **steering_step** intervals. |
| **parse_workflow_diagnostic()** | Decodes **[identifier; command; values...]** formatted commands into executable instructions. |
| **steering_<TYPE>_diag()** | Modifies diagnostics (EMF/current/species/neutrals/particles) during runtime execution. |

All routines responsible for active simulation modifications were implemented as separate functions called within **check_workflow_step()** to maintain code simplicity. Variable modifications can be achieved either by:

1. Directly accessing and updating values through the appropriate class;
2. Utilizing re-initialization routines when the changes affect dependent classes that require proper initialization.

A critical example involves the diagnostic data structure, which contains an internal file structure for storing diagnostic data. Improper initialization of this structure will cause simulation failure, necessitating careful handling through lengthy re-initialization procedures.

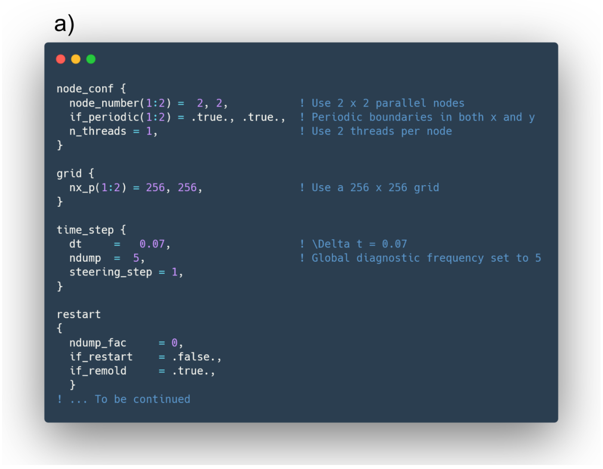
### 2.2.3 STEERING FILE SPECIFICATION

The steering file is named `steering_input_deck` by default. It shoul not be confused with the default `input_deck` as both have different structures and functions (Fig. 2.4). After processing, it is renamed to `steering_input_deck.used` and will no longer be read by the system. Nonetheless, this file follows these rules:

— Comments: Lines starting with !

— There are two command formats: **key = value** OR **diag_type_n = [name; ident; opt; val1; val2; val3]**

— Duplicate keys: Last command takes precedence a)



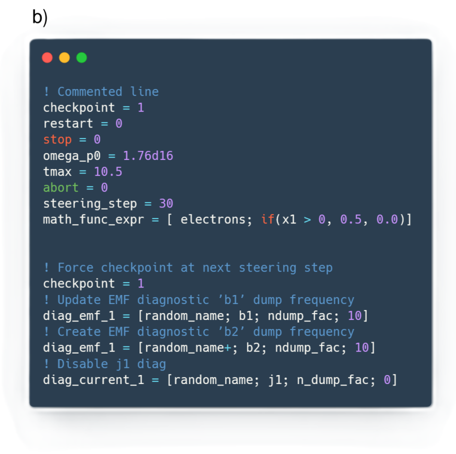
```
node_conf {
  node_number(1:2) =  2, 2,             ! Use 2 x 2 parallel nodes
  if_periodic(1:2) = .true., .true.,    ! Periodic boundaries in both x and y
  n_threads = 1,                        ! Use 2 threads per node
}

grid {
  nx_p(1:2) = 256, 256,                 ! Use a 256 x 256 grid
}

time_step {
  dt      =   0.07,                     ! \Delta t = 0.07
  ndump   =  5,                         ! Global diagnostic frequency set to 5
  steering_step = 1,
}

restart
{
  ndump_fac     = 0,
  if_restart    = .false.,
  if_remold     = .true.,
  }
! ... To be continued
```

b)



```
! Commented line
checkpoint = 1
restart = 0
stop = 0
omega_p0 = 1.76d16
tmax = 10.5
abort = 0
steering_step = 30
math_func_expr = [ electrons; if(x1 > 0, 0.5, 0.0)]


! Force checkpoint at next steering step
checkpoint = 1
! Update EMF diagnostic 'b1' dump frequency
diag_emf_1 = [random_name; b1; ndump_fac; 10]
! Create EMF diagnostic 'b2' dump frequency
diag_emf_1 = [random_name+; b2; ndump_fac; 10]
! Disable j1 diag
diag_current_1 = [random_name; j1; n_dump_fac; 0]
```

FIGURE 2.4: a) Typical input deck structure. The input deck Responsible for submitting data at the begging of the simulation and at restart points b) Steering deck structure. This file is responsible for all runtime steering.

### 2.2.4 VALID COMMANDS

A summarised list of base commands is depicted in Table 2.5. It does not include all developed commands as these are analysed in depth in Appendix A. The command selection was driven by researchers' requirements, including restart file generation to enhance productivity, diagnostic editing capabilities for general control, and precise parameter adjustments such as modifying the maximum simulation time or natural plasma frequency.

TABLE 2.5: Summary of steering file commands

| **Command** | **Value** | **Action** |
|---|---|---|
| checkpoint | 0 or 1 | Force restart file creation |
| stop | 0 or 1 | Create restart files + graceful shutdown |
| abort | 0 or 1 | Immediate termination (no restart files) |
| steering_step | Integer $\geq 0$ | Update steering frequency |
| tmax | Real > current t | Extend/Reduce simulation duration |
| math_func_expr | [NAME; CMD] | Change species/neutral distribution function |
| diag_TYPE_N | [NAME; ID; CMD; V1; V2...] | Modify diagnostics (see Table 2.6) |

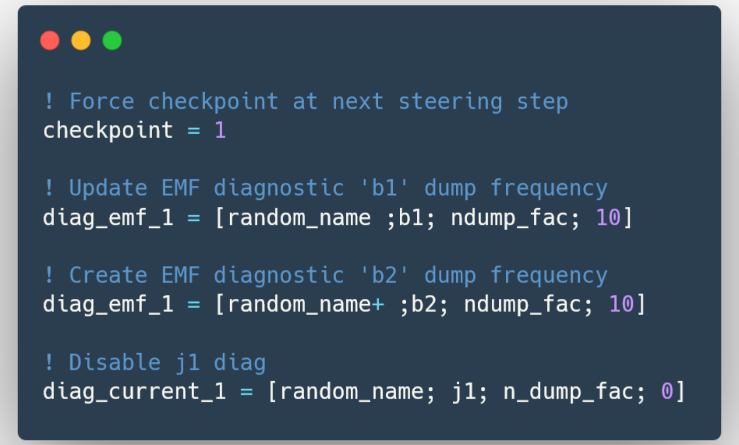
### 2.2.5 DIAGNOSTIC COMMAND STRUCTURE

The main focus here was developing the tools to allow modification and addition of the 4 diagnostic types [26]: emf, current, species and neutrals. Table 2.6 lists the purpose of each segment of the allowed instruction. Minor novelties, such as the use of "+" in the name attribute are described in the user guide in Appendix A.

TABLE 2.6: Structure of `diag_TYPE_N` steering command

| Component | Example | Description |
|---|---|---|
| TYPE | emf, species | Diagnostic type |
| N | 1, 2 | Unique ID (prevents command overwrite) |
| NAME | "Neutral+" | Name of species if needed |
| ID | "b1", "j2" | Report identifier (e.g., field component) |
| CMD | ndump_fac | Parameter to modify |
| V1, V2... | 5, 10 | New values (dimension must match parameter) |

The example in Fig. 2.5 writes checkpoint restart data, as well as modifies `n_dump_fac`, not only for the magnetic field in the X-direction **b1**, but also for all **b1** dependent diagnostics such as **b1-tavg** (time average). If **b1** was not properly initialized, the module will start writing new data. The example also shows how to disable the **j1** current diagnostic. For a detailed view on two valid diagnostics, refer to the user/developer guide.



```
! Force checkpoint at next steering step
checkpoint = 1

! Update EMF diagnostic 'b1' dump frequency
diag_emf_1 = [random_name ;b1; ndump_fac; 10]

! Create EMF diagnostic 'b2' dump frequency
diag_emf_1 = [random_name+ ;b2; ndump_fac; 10]

! Disable j1 diag
diag_current_1 = [random_name; j1; n_dump_fac; 0]
```

FIGURE 2.5: Reference instructions for steering diagnostics.

### 2.2.6 MOVING WINDOW

In PIC simulations, the moving window technique consists of moving the computational domain along with the interaction front at the speed of light so that only the physically relevant region is simulated over time. This reduces the required spatial domain, and therefore, computational cost [29]. As the window moves, new particles enter the simulation domain according to a distribution function. The ability to change this distribution during simulation, not only allows for correcting input errors, as well as the introduction of variable density regions only when they become relevant in the simulation. At the moment custom math expressions can be submitted using **math_func_expr**. An example is shown in Fig. 6. The original name-list is in Appendix B.

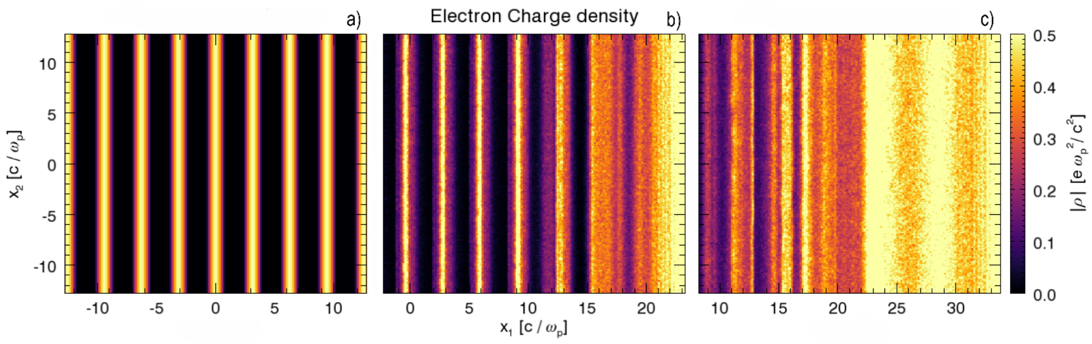


FIGURE 2.6: We simulate counter-streaming electron and positron plasmas using a 2D PIC code with a moving window in the x-direction, open boundaries in x, periodic in y, and a density profile defined by a mathematical function. Electron charge density shown at a) t = 0 $\omega_p^{-1}$ b) $t = 10.5\omega_p^{-1}$ and $t = 21\omega_p^{-1}$. The initial distribution $\rho$ = 0.5 × cos($2x_1$) is shown on the left at t = 0 $\omega_p^{-1}$. After submitting the new distribution $\rho$ = i f ($x_1$ > 0, 0.5, 0.0) the new particles are injected.

### 2.2.7 Computational impact

To assess the impact of the module, the input deck in Appendix C was used, and no data/diagnostics were written. The Steering frequency was varied and the module only checked for the presence of the steering file. Then, two of the most impactful cases were performed with a simple namelist reading. Three repetitions were conducted for each frequency. Results are shown in in Table 2.7.

TABLE 2.7: Elapsed time while using different steering step frequencies

| **Steering Step Frequency** | **Elapsed Time** |
|---|---|
| 50 | 5′56" |
| 20 | 5′56" |
| 10 | 5′56" |
| 5 | 5′56" |
| 2 | 5′56" |
| 1 | 5′58" |

To evaluate the impact of conducting changes, the process was repeated, now with a valid namelist present for steering step frequencies of 2 and 1. The simulation times were 5′57" and 5′59" respectively, showing a 1-second difference. Finally, the same input deck was run without the steering module, yielding a simulation time of 5′56". This negligible variation demonstrates that the steering module introduces no significant computational overhead to the overall simulation time.

### 2.2.8 Real life example

Recent work on radiative cooling in relativistic plasmas [30] proposed using runtime steering to optimize simulations by dynamically adjusting the reference plasma frequency ($\omega_0$). As in the example (Fig. 2.7) the plasma response is not important for (t < 5000 $\omega_p^{-1}$) increasing ($\omega_0$) during this phase speeds up the simulation, as coarsening the simulation decreases the required number of particle pushes per cycle, field solver iterations and MPI communications. This reduces computational cost by 100K CPU-hours. Steering enables these adjustments during execution and permits fine-tuning if needed. The technique is particularly valuable for long simulations where early-stage dynamics does not require high resolution, allowing computational resources to focus on physically critical later phases when new effects become relevant [16].

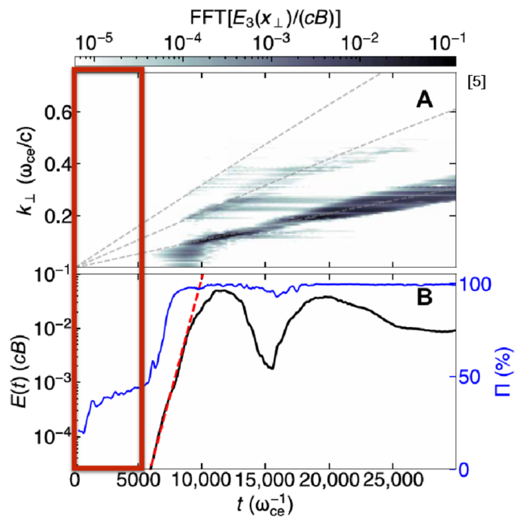


FIGURE 2.7: Figure modified with author's permission from [30]. Red Rectangle: Region of interest where quantum effects are negligible and that can be accelerated.

### 2.2.9 A "cheaper" cluster alternative

Preemptible HPC clusters with QoS-based priority scheduling are HPC environments where priority queues dictate who allocates resources and who has to abdicate of theirs in case of reaching operational capacity [31, 32]. This is becoming more common with the rise of cloud computing [33].

In most systems, it is up to the user to start a job after cancellation, which for codes such as OSIRIS, requires a manual process that is dependent on the frequency of restart files [19].

A script, **Preemptible_HPC_OSIRIS.sh**, was developed to mitigate this issue. The user can configure and run this script, which then initializes OSIRIS via **sbatch job_name** and tracks its running status. If stopped due to unexpected cancellation, it resubmits the job again to the queue up to a user defined maximum amount of times. Due to policies regarding login node usage, the script should always be run from an allocated node.

Altough this technology is not yet widely used, with Google's predictions of 60-91% discounts [33] on these types of nodes, this will become relevant very soon.

## 2.3 Future Work

Very significant results have already been achieved here, but there are many additional opportunities for steering in OSIRIS. The developed code still needs refining, especially when it comes to expanding the diagnostic options to include raw diagnostics, particle tracking and phase-spaces for the species type. The current code also needs maintenance when it comes to memory allocation for the new diagnostics and improved error handling and recovery to prevent the user from accidentally initializing an incomplete structure. Moreover, there are new features that have been discussed and can be implemented. These include the possibility of adding or removing laser pulses as well as modification of intrinsic parameters related to spatial/temporal resolution. This project holds potential for important advances, including user-triggered adaptive mesh refinement and neural network-based predictive steering to pre-empt numerical instabilities. These capabilities would further enhance OSIRIS, combining high-performance computing with intelligent automation for advanced plasma physics research.

# CHAPTER 3

# CONCLUSION

This work successfully developed two visualization tools and a dynamic steering mechanism for the OSIRIS particle-in-cell code, addressing relevant challenges in high-performance computing environments. By enabling real-time interaction with running simulations and *in-situ* data analysis through containerized tools and Jupyter Notebooks, the proposed workflow improves usability and resource efficiency. Modifications to OSIRIS introduced runtime parameter control without halting execution, valuable in competitive or preemptible HPC systems. The implementation proved feasible and expandable. Future work will focus on extending steering capabilities, improving diagnostic flexibility, and enhancing support for preemptible HPC clusters.

This 1st Cycle Scientific Project in Technological Physics Engineering was a highly rewarding experience that significantly enhanced my expertise. It enabled me to apply theoretical knowledge to practical challenges in kinetic plasma simulations, parallel computing, and computational physics, while strengthening my technical and analytical skills.

# ACKNOWLEDGMENTS

This work would not have been possible without the support of several individuals. First, I would like to express my sincere gratitude to Luis Oliveira e Silva and Thales Silva, who, following an internship conducted in the previous semester, generously proposed and supervised a project in an under-explored research area. Their guidance and provision of essential computational resources were invaluable for testing and validating the code.

I also deeply appreciate the feedback and support from the rest of the epp team, particularly Pablo Bilbao and Filipe Cruz, who were always available to discuss ideas and provide critical insights. A special thanks goes to Ricardo Fonseca for his expertise on the internal structure of OSIRIS. His advice was instrumental in guiding the modifications to the code efficiently.

The simulations presented in this work were performed on the Deucalion supercomputer (Portugal), funded by the FCT Masters in Astrophysical Plasmas (MAPs) project 2024.11062. CPCA.A3.

# Bibliography

[1] UiT HPC Team. *UiT HPC Documentation*. UiT The Arctic University of Norway, 1.0 edition. Documentation version 0.0 (PDF available at https://hpc-uit.readthedocs.io/_/downloads/en/latest/pdf/).

[2] Barcelona Supercomputing Center. Bsc support knowledge center. https://www.bsc.es/supportkc/docs/MareNostrum5/intro/, 2023. Supercomputer documentation and user guide. Accessed: 2025-06-26.

[3] Barcelona Supercomputing Center. Patc 2024: Basics of hpc. https://www.bsc.es/support/PATC_2024/1stDay/9_45-11_00_Basics.pdf, 2024. Training materials for PATC (Parallel Applications Training Course) 2024. Accessed: 2025-06-26.

[4] MACC/FCCN Team. *Deucalion User Guide*. Minho Advanced Computing Center, May 2025. Documentação técnica atualizada em 28/05/2025, incluindo partições ARM/x86, acesso SSH e gestão de jobs via Slurm. https://hpc-uit.readthedocs.io/_/downloads/en/latest/pdf/ Accessed: 2025-06-26.

[5] EuroHPC Joint Undertaking. Eurohpc systems report: Technical and operational overview. Technical report, European High Performance Computing Joint Undertaking (EuroHPC JU), September 2021. Technical report detailing system architectures, performance metrics, and operational frameworks for EuroHPC supercomputers. https://eurohpc-ju.europa.eu/system/files/2023-07/EuroHPC%20Systems%20Report-Sep2021.pdf Accessed: 2025-06-26.

[6] TOP500.org. Top500 list - june 2025. https://top500.org/lists/top500/2025/06/, 2025. 65th edition of the biannual ranking of the world's most powerful supercomputers. Accessed: 2025-06-26.

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. MPI Forum, version 5.0 edition, 09 2024. Official specification document for MPI-5.0 standard https://www.mpi-forum.org/docs/mpi-5.0/mpi50-report.pdf Accessed: 2025-06-22.

[8] Wesley Bland. Mpi broadcast and collective communication. https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/, n.d. Accessed: 2025-06-22.

[9] The HDF Group. *HDF4 User's Guide*. The HDF Group, 2008. Documentation for HDF4 (Hierarchical Data Format version 4) `https://docs.hdfgroup.org/archive/support/release4/doc/UG_PDF.pdf` Accessed: 2025-06-26.

[10] The HDF Group. *HDF5 User's Guide*. The HDF Group, 2010. Official user manual for HDF5 version 1.8 `https://docs.hdfgroup.org/archive/support/HDF5/doc1.8/UG/HDF5_Users_Guide.pdf` Accessed:2025-06-25.

[11] The HDF Group. Hdf5: A data model, library, and file format for storing and managing data, 2024. Accessed: 2025-05-08.

[12] Council of the European Union. Council regulation (eu) 2021/1173 on establishing the european high performance computing joint undertaking. Legal basis for the EuroHPC Joint Undertaking (2021-2027) `https://eurohpc-ju.europa.eu/system/files/2022-03/uriserv_OJ.L_.2021.256.01.0003.01.ENG_EN_TXT.pdf` Accessed: 2025-06-27.

[13] FCT. Espaço europeu de investigação - eurohpc joint undertaking, 2024. `https://www.fct.pt/internacional/espaco-europeu-de-investigacao/eurohpc-ju/` Accessed: 2025-05-08.

[14] EESSI Collaboration. *EESSI Documentation*. European Environment for Scientific Software Installations, 2024. Comprehensive documentation for the EESSI software stack, including user guides, architecture specifications, and deployment procedures `https://www.eessi.io/docs/` Accessed: 2025-06-23.

[15] Microsoft Corporation. *Introduction to Containers*. Microsoft, 2021. Technical eBook covering container fundamentals, Docker, and Kubernetes (v2.0) `https://download.microsoft.com/download/a/1/3/a13a2b9e-d47c-4f93-b180-f7f9cd3382a7/introduction_to_containers_ebook.pdf` Accessed: 2025-06-24.

[16] OSIRIS Consortium. Osiris documentation. `https://osiris-code.github.io/`, 2025. Accessed: 2025-06-26.

[17] UCLA Plasma Simulation Group. Introduction to particle-in-cell simulation methods. `https://picksc.physics.ucla.edu/presentations/introduction.pdf`, 2023. Tutorial slides from the Particle-in-Cell and Kinetic Simulation Center (PICKSC). Accessed: 2025-06-26.

[18] Ken-Ichi Nishikawa, Ioana Dutan, Christoph Koehn, and Yosuke Mizuno. Pic methods in astrophysics: simulations of relativistic jets and kinetic physics in astrophysical systems. *Living Reviews in Computational Astrophysics*, 7, 12 2021. DOI: 10.1007/s41115-021-00012-0.

[19] R. A. Fonseca, L. O. Silva, R. G. Hemker, F. S. Tsung, V. K. Decyk, W. Lu, C. Ren, W. B. Mori, S. Deng, S. Lee, T. Katsouleas, and J. C. Adam. OSIRIS: A three-dimensional, fully relativistic particle-in-cell code for modeling plasma based accelerators. In Peter M. A. Sloot, C. J. Kenneth Tan, Jack Dongarra, and Alfons Hoekstra, editors, *Computational*

*Science — ICCS 2002*, volume 2331 of *Lecture Notes in Computer Science*, pages 342–351. Springer, 2002. DOI: 10.1007/3-540-47789-6$_3$6.

[20] R A Fonseca, S F Martins, L O Silva, J W Tonge, F S Tsung, and W B Mori. One-to-one direct modeling of experiments and astrophysical scenarios: pushing the envelope on kinetic plasma simulations. *Plasma Physics and Controlled Fusion*, 50(12):124034, nov 2008. DOI: 10.1088/0741-3335/50/12/124034.

[21] Docker, Inc. *Dockerfile Reference*, 2024. Version 1.8 of Dockerfile specification `https://docs.docker.com/engine/reference/builder/` Accessed: 2025-06-26.

[22] Sylabs Inc. *SingularityCE User Guide*. Sylabs, 2023. Legacy documentation for SingularityCE v3.6 (pre-Apptainer fork) `https://docs.sylabs.io/guides/3.6/user-guide.pdf` Accessed: 2025-06-23.

[23] Apptainer Community. *Apptainer User Documentation*. Linux Foundation, 2024. Official documentation for Apptainer (formerly Singularity) v1.3+ `https://apptainer.org/docs/user/main/` Accessed: 2025-06-24.

[24] Open OnDemand Team. *Open OnDemand Documentation*. Ohio Supercomputer Center (OSC), 2025. Latest documentation for Open OnDemand web portal (v3.0+) `https://osc.github.io/ood-documentation/latest/` Accessed: 2025-06-27.

[25] Project Jupyter. *Jupyter Notebook Documentation*. Project Jupyter, version 6.4.5 edition, 08 2021. Official PDF documentation for Jupyter Notebook (classic interface) `https://jupyter-notebook.readthedocs.io/_/downloads/en/v6.4.5/pdf/` Accessed: 2025-06-27.

[26] OSIRIS Development Team. *OSIRIS Input Parameters Reference*. OSIRIS Consortium, 2025. Complete specification of input deck parameters for OSIRIS 4.0+ `https://osiris-code.github.io/osiris/input/` Accessed: 2025-06-27.

[27] David W. Fanning. *IDL Programming Techniques*. Coyote Book, 2000. `https://www.astro.sunysb.edu/fwalter/AST443/pdf/idl_manual.pdf` Accessed: 2025-06-14.

[28] Pavlin Mitev. Converting docker containers for hpc with singularity. `https://pmitev.github.io/UPPMAX-Singularity-workshop/docker2singularity/`, March 2021. Workshop materials covering Docker-to-Singularity conversion for HPC systems. Accessed: 2025-06-26.

[29] A. Pukhov. Particle–in–cell codes for plasma-based particle acceleration. In B. Holzer, editor, *Proceedings of the CAS–CERN Accelerator School: Plasma Wake Acceleration*, pages 181–206, Geneva, Switzerland, 2016. CERN. DOI:10.5170/CERN-2016-001.181.

[30] Pablo Bilbao, Thales Silva, and Luís O. Silva. Radiative cooling induced coherent maser emission in relativistic plasmas. *Science Advances*, 11(15):eadt8912, 2025. DOI:10.1126/sciadv.adt8912.

[31] Yandex Cloud. Running hpc workloads on preemptible virtual machines. `https://yandex.cloud/en/docs/tutorials/testing/hpc-on-preemptible`, n.d. Accessed: 2025-06-22.

[32] Pablo Martínez, Jeremy Tonge, Ricardo A. Fonseca, and Luís O. Silva. Dynamic control of the number of particles per cell in PIC simulations. *arXiv preprint*, 2019. 10.48550/arXiv.1905.03094.

[33] Google Cloud. Preemptible vm instances. `https://cloud.google.com/compute/docs/instances/preemptible`, n.d. Accessed: 2025-06-22.

# APPENDIX

## APPENDIX A: USER GUIDES

Visualization software user guide and Steering software user/developer guide

## APPENDIX B: MOVING WINDOW CONFIGURATION

```
simulation{n0=3.2e16}
node_conf{
 node_number(1:2)=2,2, if_periodic(1:2)=.false.,.true., n_threads=1}
grid{nx_p(1:2)=256,256}
time_step{dt=0.07, ndump=5, steering_step=100}
restart{ndump_fac=0, if_restart=.false., if_remold=.false.}
space{
 xmin(1:2)=-12.8,-12.8, xmax(1:2)=12.8,12.8, if_move(1:2)=.true.,.false.}
time{tmax=30.0}
emf_bound{
 type(1:2,1)="open","open", type(1:2,2)="conducting","conducting"}
particles{num_species=2}
species{name="electrons", num_par_x(1:2)=8,8, rqm=-1.0}
udist{uth(1:3)=0.1,0.1,0.1, ufl(1:3)=0.6,0.0,0.6}
profile{
 profile_type="math func", math_func_expr="0.5*cos(2*x1)", density=1.0}
spe_bound{
 type(1:2,1)="open","open", type(1:2,2)="open","open"}
diag_species{ndump_fac=2, reports="charge"}
species{name="positrons", num_par_x(1:2)=8,8, rqm=+1.0}
udist{uth(1:3)=0.1,0.1,0.1, ufl(1:3)=0.0,0.0,-0.6}
spe_bound{
 type(1:2,1)="open","open", type(1:2,2)="open","open"}
current{}
diag_emf{ndump_fac=0, reports="b1","b2","b3"}
diag_current{ndump_fac=0, reports="j3"}
```

## APPENDIX C: BENCHMARK CONFIGURATION

```
node_conf{node_number(1:2)=2,2,if_periodic(1:2)=.true.,.true.,n_threads=1}
grid{nx_p(1:2)=256,256}
time_step{dt=0.07,ndump=5,steering_step=1}
restart{ndump_fac=0,if_restart=.false.,if_remold=.true.}
space{xmin(1:2)=-12.8,-12.8,xmax(1:2)=12.8,12.8}
time{tmax=30.0}
emf_bound{}
particles{num_species=2}
species{name="electrons",num_par_x(1:2)=8,8,rqm=-1.0}
udist{uth(1:3)=0.1,0.1,0.1,ufl(1:3)=0.0,0.0,0.6}
species{name="positrons",num_par_x(1:2)=8,8,rqm=+1.0}
udist{uth(1:3)=0.1,0.1,0.1,ufl(1:3)=0.0,0.0,-0.6}
current{}
```