EVOLUTION OF PYTHON TOOLS FOR THE SIMULATION OF ELECTRON CLOUD EFFECTS

G. Iadarola*, E. Belli, K. Li, L. Mether, A. Romano, G. Rumolo, CERN, Geneva, Switzerland

Abstract

PyECLOUD was originally developed as a tool for the simulation of electron cloud build-up in particle accelerators. Over the last five years the code has become part of a wider set of modular and scriptable Python tools that can be combined to study different effects of the e-cloud in increasingly complex scenarios. The Particle In Cell solver originally included in PyECLOUD later developed into a stand-alone general purpose library (PyPIC) that now includes advanced features like a refined modeling of curved boundaries and optimized resolution based on the usage of nested grids. The effects of the e-cloud on the beam dynamics can be simulated interfacing PyECLOUD with the PyHEADTAIL code. These simulations can be computationally very demanding due to the multi-scale nature of this kind of problems. Hence, a dedicated parallelization layer (PyPARIS) has been recently developed to profit from parallel computing resources in order to significantly speed up the computation.

INTRODUCTION

Electron cloud effects pose several challenges to the operation of the Large Hadron Collider (LHC) at CERN when operating with the nominal bunch spacing of 25 ns. The machine configuration had to be carefully optimized in order to avoid e-cloud induced instabilities, while the power deposited by the electrons on the beam screens of the cryogenic magnets constitutes a major load for the LHC cryogenic system [1].

As the understanding of these phenomena heavily relies on macroparticle simulations, in the latest years a significant effort has been devoted to the development of numerical tools to model the formation of the e-cloud and its effect on the machine systems and on the beam dynamics.

A PYTHON SIMULATION TOOLKIT

Experience has shown that these simulation tools need to be very flexible in order to cover the variety of simulation scenarios that are of interest for the LHC and its injector chain. In this respect we found it convenient to abandon the idea of a monolithic code with a rigid user interface in favor of a set of tools in the form of Python [2] libraries. In this way the user can use the power of Python scripting to define arbitrarily complex simulation setups and use the multitude of freely available Python packages to perform post-processing, data manipulation and storage, plotting etc.

The Python language is also used for a large fraction of the implementation, which proved to ease significantly the process of development and maintenance. Compiled languages, namely C and FORTRAN, are used to program computationally intensive tasks, using cython [3] and f2py [4] to interface these parts to the Python code.

PyECLOUD

PyECLOUD [5] is the core of our e-cloud simulation toolkit. It is a 2D macroparticle (MP) code for the simulation of the electron cloud formation in accelerator structures. It is developed and maintained at CERN since 2011 [6] following the legacy of the ECLOUD code [7].

In the classical "e-cloud buildup" simulation mode, the code generates “primary” electrons due to ionization of the residual gas and “photoemission” driven by synchrotron radiation, evaluates the forces acting on the electrons due to the interaction with the particle beam and with the electrons themselves (the latter is done using a Particle-In-Cell – PIC – algorithm) and tracks the electrons accounting for the presence of externally applied magnetic fields. When an electron impacts on the chamber’s walls, secondary electrons can be generated according to the implemented secondary emission models. One of the peculiarities of these simulations is that the number of electrons grows exponentially during the buildup process, spanning several orders of magnitude. For the calculation to remain computationally affordable, the number of MPs cannot stay proportional to the number of electrons. Instead, the number and the mean size of the MPs need to be dynamically adapted during the simulation. Details about the implemented models and algorithms can be found in [5, 8].

Over the years several features were added to the code allowing buildup simulations for increasingly complex scenarios: possibility of simulating multiple beams, accurate tracking routines for the simulation of non-uniform magnetic fields (e.g. quadrupoles, combined function magnets), arbitrarily shaped polygonal chamber and non-uniform surface properties.

Figure 1 shows an example of a simulation where a non-convex chamber shape was used to model a pumping slot in the LHC beam screen with the baffle shield installed behind the opening [9].

PyECLOUD-PyHEADTAIL simulation setup

To simulate the impact of the e-cloud on the beam dynamics we decided to drop the traditional approach of having a tool that is completely separated from the buildup simulator. We decided instead to build an interface which would allow combining PyECLOUD with the PyHEADTAIL beam dynamics code [10, 11].

* Giovanni.Iadarola@cern.ch

ISBN 978-3-95450-182-3

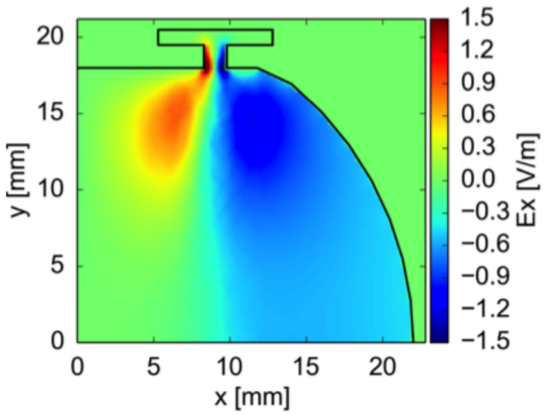

Figure 1: Horizontal component of the electric field generated by the electron cloud in the region of a pumping slot of the LHC arc beam screen [9].

In PyHEADTAIL the accelerator is modeled as a list of Python objects arranged in a ring structure, which perform different actions on the beam (longitudinal and transverse tracking, wake fields, feedbacks, space charge, etc.). PyECLOUD allows the definition of electron cloud objects which can be inserted in a PyHEADTAIL ring. When receiving a bunch from PyHEADTAIL the e-cloud object performs the simulation of the electron dynamics as for the buildup simulations with two notable differences: instead of using a predefined rigid distribution, the beam field at each time-step is computed by applying a PIC algorithm to the PyHEADTAIL MPs in the corresponding beam slice; the electric forces from the e-cloud evaluated by the PIC are applied not only to the e-cloud MPs but also to the beam MPs [12].

This solution was possible thanks to the highly modular structure of both PyECLOUD and PyHEADTAIL and allowed for a significant reduction of the work needed for development and maintenance. Moreover, all the advanced features available in PyECLOUD for the simulation of the e-cloud buildup became automatically available also for the simulation of the beam dynamics in the presence of e-cloud. Furthermore, this setup could be generalized for the simulation for multi-bunch Fast Beam Ion Instabilities [13].

PyPIC

Initially PyECLOUD included a simple PIC solver. Later on we decided to provide it as a separated Python library, called PyPIC [14], in order to make it available to other applications. By now the library contains different solvers, using both FFT methods and Finite Difference (FD) methods. The latter are more frequently used for e-cloud simulations as they are more suited to model arbitrarily shaped chambers. To improve the field calculation accuracy in the presence of curved boundaries (as this is critical in multipacting simulations), the "Shortley-Weller" refinement of the boundary conditions was implemented [15, 16].

As the PIC solution needs to be performed at each time step, its execution has a strong impact on the overall computational time. Therefore a significant effort was put into the optimization of this component. As the grid and the shape of the boundary stay constant during the simulation, it is possible to compute and store the LU factorization [17] of the FD matrix and apply the back-substitution at each step. It was found that, as for circuit simulation problems, the KLU library outperforms the more common SuperLU for these very sparse matrices [18]. A cython wrapper [19] of the KLU library [20] was written to interface the library to the PyPIC code.

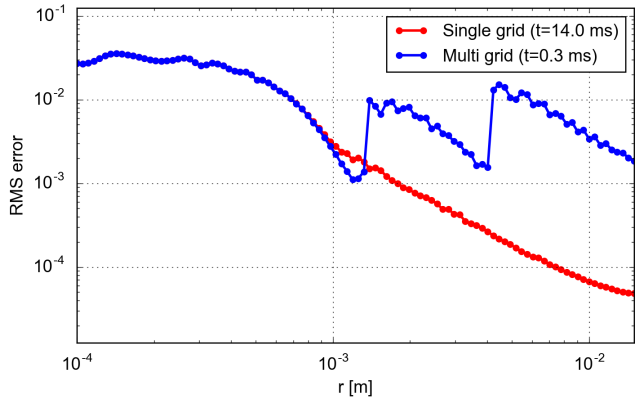

Figure 2: Error on the computation of the electric field for a charge distribution corresponding to the LHC beam at 6.5 TeV. The PyPIC single grid and multigrid solvers are compared as a function of the distance from the bunch center. In the legend: computation time the field map evaluation.

An important feature of this kind of simulations is that the required PIC resolution is not uniform over the simulation domain. In fact a strong gradient in the charge distribution is observed at the beam location (pinch effect) while the distribution is much smoother elsewhere. It is therefore convenient to employ nested grids with different resolution to refine the PIC accuracy only where needed. This was implemented following the approach described in [21] and was found to have a dramatic impact on the processing time [22].

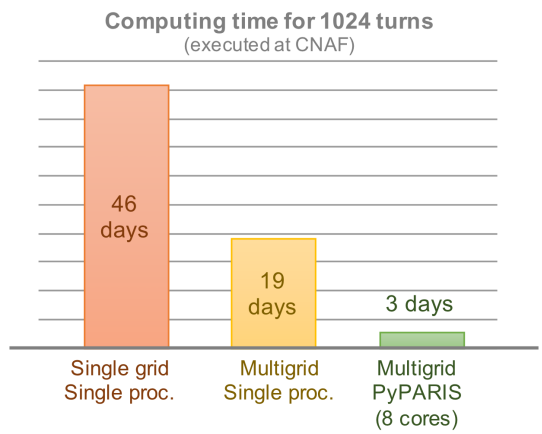

Figure 3: Required time to simulate the interaction of the LHC beam at high energy with the e-cloud in dipoles and quadrupole magnets, using different simulation setups. For details on the simulated scenario see [23].

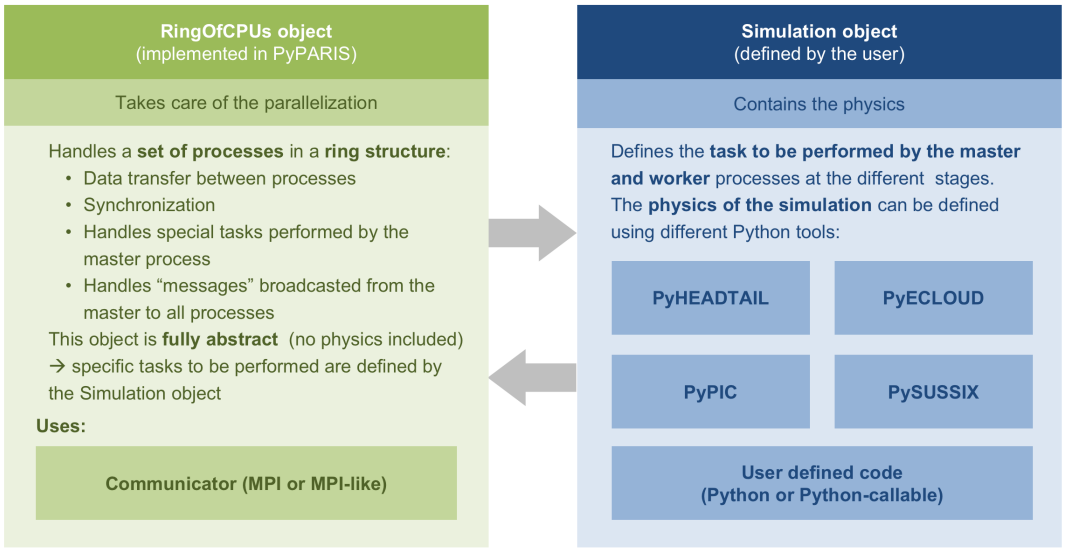

Figure 4: Schematic representation of the two main objects used for the parallel simulation of e-cloud driven instabilities.

Figure 2 shows an example in which, using three nested grids, it is possible to keep the calculation error below 4% over the entire simulation domain and to reduce the computation time required for the linear system solution by over a factor of 40 with respect to a single grid solver with the same target accuracy.

PyPARIS

Electron cloud instabilities observed at high energy in the LHC have instability rise-times of the order of 10^4 turns. Considering the necessity of having a sufficient time resolution of the electron motion during the bunch passage, this translates into over 10^7 time steps to be simulated. Figure 3 shows the computational time for a shorter simulation (10^3 turns). Even profiting from the speedup from the PyPIC multigrid, the time requirements are still prohibitive. To tackle these cases we resorted to parallel computing.

The simulation is performed by multiple processes organized in a ring structure. Each process takes care of the interaction of the beam with a fraction of the accelerator. The different bunch slices are traveling along the ring of processors as they do in the real accelerator. At the end of each turn all particles need to be recollected by a "master" process for longitudinal tracking and re-slicing. For this reason the expected speed-up is not proportional to the number of processes as shown in Fig. 5 (more details can be found in [23]). The parallelization is realized by an additional Python layer called PyPARIS (Python PAralallel RIng Simulator, [24]) independent from PyECLOUD and PyHEADTAIL. This was done in order to keep as separated as possible the physics and the parallelization code (a developer who is unaware of the parallelization details should still be able to extend the physics part) and to minimize the number of changes in pre-existing tools (to avoid extensive re-validation). This is implemented as shown in Fig. 4. The simulation is managed by two Python objects that are instantiated by each process:

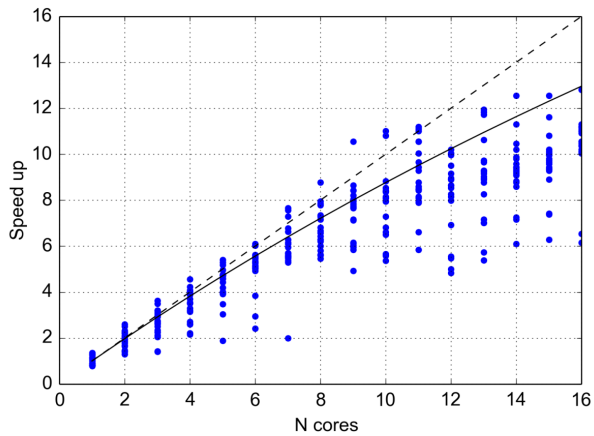

Figure 5: Observed speedup as a function of the number of cores (blue dots), compared to the theoretical expectation for the implemented parallelization strategy (solid line) and to the ideal speedup $S = N_{\text{cores}}$. For details on simulated scenario see [23].

a "RingOfCPUs" object, implemented in PyPARIS, takes care of all the tasks related to the parallelization; a "Simulation" object, which can be customized by the user, defines the physics of the simulation using the other libraries of the toolkit (PyHEADTAIL, PyECLOUD, etc.). More information on the implementation can be found in [24].

Tests performed at the INFN-CNAF cluster showed a satisfactory speedup as shown in Figs. 3 and 5. We found particularly convenient to perform our studies using 8 CPU cores per simulation, as this allows achieving a practically ideal speedup ($S = N_{\text{cores}}$) while being able to launch several simulations at the same time to study parametric dependencies. This simulation mode has been extensively used to study LHC e-cloud driven instabilities [25].

ISBN 978-3-95450-182-3

ACKNOWLEDGMENTS

The authors would like to thank the INFN-CNAF institute in Bologna (Italy) for providing support and access to their computing facilities. Research supported by the HL-LHC project.

REFERENCES

[1] K. Li *et al.*, "Electron Cloud Observations during LHC Operation with 25 ns Beams", in Proceedings of IPAC2016, Busan, Korea, paper TUPMW017, 2016.

[2] https://www.python.org

[3] https://www.cython.org

[4] https://docs.scipy.org/doc/numpy-dev/f2py/

[5] https://github.com/PyCOMPLETE/PyECLOUD/wiki

[6] G. Iadarola and G. Rumolo, "PyECLOUD and build-up simulations at CERN", Proceedings of the ECLOUD12 Workshop, CERN-2013-002, pp. 189-194.

[7] F. Zimmermann, "A Simulation Study of Electron-Cloud Instability and Beam-Induced Multipacting in the LHC", CERN LHC Project Report 95, SLAC-PUB-7425 (1997).

[8] G. Iadarola, "Electron cloud studies for CERN particle accelerators and simulation code development", https://cds.cern.ch/record/1705520CERN-THESIS-2014-047, 2014.

[9] A. Romano *et al.*, "Effect of the LHC Beam Screen Baffle on the Electron Cloud Buildup", Proceedings of IPAC2016, Busan, Korea, paper TUPMW016, 2016.

[10] https://github.com/PyCOMPLETE/PyHEADTAIL/wiki

[11] K. Li *et al.*, "Code Development for Collective Effects", *in Proc. 57th ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams (HB16)*, Malmo, Sweden, http://jacow.org/hb2016/papers/weam3x01.pdf, paper WEAM3X01, 2016.

[12] G. Iadarola, "PyHEADTAIL-PyECLOUD development", presentation at theElectron Cloud Meeting, 27 May 2015, CERN, Geneva. https://indico.cern.ch/event/394530/

[13] L. Mether *et al.*, "Numerical Modeling of Fast Beam Ion Instabilities", in Proc. 57th ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams (HB16), Malmo, Sweden, paper WEAM4X01, 2016.

[14] https://github.com/PyCOMPLETE/PyPIC/wiki

[15] G. H. Shortley and R. Weller, "The numerical solution of Laplace's equation" Journal of Applied Physics 9.5 (1938): 334-348.

[16] G. Iadarola, "PyECLOUD development: accurate space charge module", presentation at the Electron Cloud Meeting, 27 Jun 2014, CERN, Geneva. https://indico.cern.ch/event/320287/

[17] I. S. Duff, A. M. Erisman, and J. K. Reid. Direct methods for sparse matrices. Oxford: Clarendon press, 1986.

[18] T. A. Davis and E. P. Natarajan, "Algorithm 907: KLU, A Direct Sparse Solver for Circuit Simulation Problems", ACM Trans. Math. Softw. 37, 3, Article 36, 2010.

[19] https://github.com/PyCOMPLETE/PyKLU

[20] http://faculty.cse.tamu.edu/davis/suitesparse.html

[21] J.-L. Vay *et al.*, "Mesh refinement for particle-in-cell plasma simulations: Applications to and benefits for heavy ion fusion", Laser Particle Beams, vol. 20, no. 4, pp. 569–575, 2002.

[22] E. Belli *et al.*, "Multigrid solver in PyPIC", presentation at the Electron Cloud Meeting, 2 Sep 2016, CERN, Geneva. https://indico.cern.ch/event/547910/

[23] G. Iadarola *et al.*, "PyPARIS: parallelisation strategy for PyECLOUD-PyHEADTAIL simulations", presentation at the Electron Cloud Meeting 2 Sep 2016, CERN, Geneva. https://indico.cern.ch/event/547910/.

[24] https://github.com/PyCOMPLETE/PyPARIS/wiki

[25] A. Romano *et al.*, "Macroparticle Simulation Studies of LHC Beam Dynamics in the Presence of the E-Cloud", presented at IPAC'17 Copenhagen, Denmark 2017, paper TUPVA018, this conference.