Software design of the educational 1d BEPS1 PIC codes

Viktor K. Decyk and Joshua Kelly
Department of Physics and Astronomy
University of California, Los Angeles
Los Angles, California 90095-1547

Introduction

The purpose of this interactive Particle-in-Cell (PIC) code is to illustrate important concepts in the teaching of plasma physics. Plasmas are ionized gases interacting with electromagnetic fields they generate. It is an example of a many body system described by statistical mechanics. PIC codes model plasmas as discrete particles and this code serves as a lab to perform numerical experiments. It is designed to be used either in conjunction with a textbook or for individual self-study. There are two major modes of operation. The basic mode provides a set of input parameters which are provided for a given concept and users are encouraged to vary a limited number of parameters to better understand the concept. For advanced students, one can perform homework problems or basic research for problems where input parameters have not been supplied or are unknown. This would likely involve modifying the code. The software is designed to support both modes of operation. It is a modern version of a code originally developed in the 1980s [1].

This software consists of three separate Particle-in-Cell codes: an electrostatic, and electromagnetic, and a darwin code. They are based on the 1d OpenMP skeleton codes (mpic1, mbpic1, mdpic1) available at: http://picksc.idre.ucla.edu/software/skeleton-code/, with additional diagnostics, initial conditions, and a Graphical User Interface (GUI) added. The codes are written in layers. The target platform are student laptops with multicore processors. The lowest, most compute intensive layer is written in a Fortran77 subset of Fortran90. This layer uses only basic types, without array syntax and compiles to very fast code. It is easy to replace this layer with a C language layer. It contains about 100 procedures and 8,000 lines of code. The middle layer consists of Fortran90 wrappers, which simplifies the argument lists, introduces some polymorphism with case statements, adds safety checks, and is designed to be easily called from Python. It consists of 100 procedures and 4,000 lines of code organized in 10 libraries. The upper layer consists of 3 high level libraries which are replicated in Fortran and Python. The Fortran version is written to conform to the Fortran2003 standard which allows implementation of such features as object oriented programming, interoperability with C, and stream IO. These features are also available in Python. The high level libraries, however, are not interoperable. Two of the high

level libraries contain about 50 procedures and 2,000 lines of code, the third contains 30 procedures and 1,000 lines of code. The main code in each language primarily calls the high level libraries. The Python main script contains the GUI elements. The Fortran main code is intended to be run in non-interactive mode.

Low level libraries

The libraries are organized according the type of code, electrostatic, electromagnetic, and darwin. Seven of the libraries are used by all 3 codes:

```
libminit1.f: initializes particles
libmpush1.f: pushes electrostatic particles, deposits charge,
             and provides utility functions
libmsort1.f: reorders particles for parallelization
libmgard1.f: provides functions to process guard cells
libmfield1.f: provides spectral field solvers and diagnostics
libmfft1.f: provides 1D FFTs for scalar and vector arrays
libmdiag1.f: provides diagnostic procedures such as distribution
             functions, frequency analysis, and others
```

Two of the libraries are used by the electromagnetic and darwin codes:

```
libmcurd1.f: deposits current density
libmbpush1.f: pushes electromagnetic particles
```

One library is used by the darwin code:

```
libmdpush1.f: deposits time derivative of current density
```

Comments at the beginning of each library describe what each individual procedure does and comments at the beginning of each function give additional details as well as information about the input and output variables.

Middle level libraries

The ten middle level libraries provide an easier to use interface to the low level libraries and can be called by Python. They provide error checking but do not process errors (which may need to be processed in another language.) They also provide some level of polymorphism, such as whether a relativistic version of procedures should be used.

The middle level wrapper libraries have the same structure as the low level libraries. They are written to conform to the Fortran 90 standard. The names are similar, except that the start with `mod...` and end in `...f90` instead of `lib...` and `...f`. For

example, the wrapper for `libminit1.f` is `modminit1.f90`. In addition, there are libraries that provide interfaces to the low level procedures to support argument checking for procedures (similar to header files in C). Their names are the same as the low level libraries, except they terminate with `_h.f90` instead of `.f`.

Comments at the beginning of each library describe what each individual procedure does and what low level procedures are called.

The Python wrappers are created automatically from the middle layer by the numpy tool f2py. This required that the middle layer avoid certain Fortran90 language features, such as derived types and function overloading, and required that they provide the `intent` attribute for dummy arguments. The attribute `intent(inout)` was used whenever a variable or array was modified, and `intent(in)` otherwise. All communication between Python and Fortran occurs only in the middle layer,

Input to the codes currently consists of about 100 `namelist` variables in 4 different `namelists`. The variables are defined and default values are given in the Fortran90 file `input1mod.f90`, which defines a module called `in1`. The `namelist input1` is used by all three codes. The `namelist input1b` is used by the electromagnetic and darwin codes, and the `namelist input1d` is used only by the darwin code. There is also a `namelist ions1` which is used by all 3 codes if ions are mobile. Comments in the module `input1mod.f90` give short descriptions of each input variable and its usage.

The input values stored in `input1mod.f90` are directly accessible in Python when the module is wrapped by f2py. For example, the integer input variable idrun in the Fortran module in1 is accessible in Python by the name in1.idrun. A Fortran function encapsulating the reading of the `namelists` is then called by Python to update the defaults. Some of these inputs can be interactively modified by Python.

`High Level libraries`

Three Fortran2003 high level libraries are defined. The main purpose of these libraries is to encapsulate the data and the main steps of the PIC code and the major diagnostics that will be visualized by the GUI so that users of the code could easily add their own diagnostics and special initial conditions.

The library `msimul1.f03` is used by all three codes. This library defines a module `f1` which provides support for the following:

1. The allocation and deallocation of scalar field, particle, and diagnostic arrays
2. Particle initialization, electrostatic particle push, and particle reordering.
3. Support for time reversal for electrostatic code
4. Support for restart and reset files for electrostatic code

It also provides encapsulation of the following diagnostics:

1. Energy and energy conservation
2. Electron and ion density and ion spectrum
3. Potential and potential spectrum
4. Longitudinal electric field
5. Electron and ion velocity distributions and entropy (one velocity component)
6. Test particle trajectories and distributions (one velocity component)

The library `mbsimul1.f03` is used by the electromagnetic and darwin codes. This library defines a module `fb1` which provides support for the following:

1, The allocation and deallocation of vector field and diagnostic arrays
2. Particle initialization, electromagnetic particle push, and current deposit.
3. Support for time reversal for electromagnetic code
4. Support for restart and reset files for electromagnetic code

It also provides encapsulation of the following diagnostics:

1. Energy and energy conservation
2. Electron and ion current density and ion current spectrum
3. Radiative vector potential and spectrum
4. Vector potential spectrum
5. Transverse electric field and spectrum
6. Magnetic field
7. Electron and ion velocity distributions and entropy (three velocity components)
8. Test particle trajectories and distributions (three velocity components)

The library `mdsimul1.f03` is used by the darwin code. This library defines a module `fd1` which provides support for the following:

1, The allocation and deallocation of darwin field and diagnostic arrays
2. Darwin particle push
3. Support for time reversal for darwin code
4. Support for restart and reset files for darwin code

It also provides encapsulation of the following diagnostics:

1. Energy and energy conservation
2. Vector potential spectrum
3. Transverse electric field and spectrum
4. Magnetic field

Comments at the beginning of each library describe what each individual procedure does. Currently, the data in the modules are public and there are few if any arguments to the procedures. This was done to facilitate access to data which users might want to modify. It is possible, however, to create objects and encapsulate the data itself.

The three high level Fortran modules f1, fb1, and fd1, were first translated line by line to three python scripts entitled s1.py, sb1.py, and sd1.py, respectively, using numpy arrays and procedures to replace Fortran90 array syntax. The Fortran middle layer is called directly by Python, with surprisingly little overhead cost. Comments in the Python script conform to conventions so they are readable by standard Python help facilities. There are no GUI elements in these Python scripts.

Main codes

Three Fortran90 main codes were written for each code, `mbeps1.f90`, `mbbeps1.f90`, and `mdbeps1.f90`. They are small in size (500-800 lines) because they mainly call the high level procedures. These main scripts were written by translating the Fortran line by line, creating mbeps1.py, mbbeps1.py, and mdbeps1.py. The GUI elements were then added to these scripts. The libraries and main codes are compiled by a Makefile.

The Python GUI uses wxPython and matplotlib for interactive graphics. Both libraries are cross-platform, and matplotlib is familiar to many scientists.. It was originally designed as a stand alone application, however it evolved to a library that can be used in multiple levels of complexity. The advantage is the user can write their Python however they chose, without having to learn an application framework.

In its most basic form, the library will initialize an interactive window that will display plots as requested by the simulation code. The initialization is around 5 lines of code, and plotting is a single line. In this mode, the user has control of the layout of plots on the screen, and can interact with the plots.

The library allows for a high level of interactivity between the GUI and host Python via callbacks. For example, there is a simulation Reset button, that by default does not do anything. However by hooking in to the reset callback, the host Python can chose how it resets itself. The GUI also provides a way to change simulation parameters on the fly, such as reversing time. The host python is however not obligated to use these features.

There many creative uses for the library. For example, if the host python and Fortran was written in such a way that the simulation state was encapsulated in a python object, it would be straightforward to create new simulation instances and run a single time step to generate a simulation preview. Another option would be to run two simulations simultaneously in the host python, and display the difference between them as the simulation runs.

No matter how it is used, the library always looks like a stand-alone application, and provides a certain minimum set of features.

References

[1] Viktor K. Decyk and Joan E. Slottow, "Supercomputers in the classroom," Computers in Physics, March/April, p. 50 (1989).