PEG-PIC and PEG-TC simulation codes*
(a suite of plasma engineering R&D software)

Janez Krek[1], **Nikola Jelić**[1,2,3,4], **Leon Kos**[1], **Jože Duhovnik**[1]

[1]LECAD Laboratory, Faculty of Mechanical Engineering,
University of Ljubljana, SI-1000 Ljubljana, Slovenia

[2]Association EURATOM-ÖAW, Institute for Theoretical
and Computational Physics, TU Graz, A-8010 Graz, Austria

[3]Association EURATOM-ÖAW, Institute for Theoretical Physics,
University of Innsbruck, A-6020 Innsbruck, Austria

[4]TC Semto, SI-1000 Ljubljana, Slovenia

Ljubljana, december 2012

*PEG-PIC and PEG-TC are LECAD codes which are used in investigations of fusion-related, technology-oriented, laboratory and space plasmas. Particular ongoing applications of this codes are investigations on gas discharge tubes (gas arresters), supported by Slovenian Research Agency (ARRS), and fusion boundary conditions.

Contents

Abstract

For use in our current and future projects, we developed two simulation codes (software applications) to help us perform computer simulations in fields of plasmas. First is PEG-PIC (Plasma Engineering Group - Particle-In-Cell), which is based on already proven plasma PIC simulation codes [6], and PEG-TC (Plasma Engineering Group - TreeCode), which is based on treecode algorithm [1] and is presenting a grid-less method for computing parameters in various plasma-related simulations [3].

1 Introduction

In this report we present two simulation codes (PEG-PIC and PEG-TC) that were developed during our colaboration on various plasma related projects. PEG-PIC is based on variety of simulation codes, mainly on BIT1 [4] and Berkeley's XPDP1 and fall into Particle-In-Cell (PIC) category of simulation codes. Main advantages of PEG-PIC code, not yet available in other free accessible codes, are variable reflection coefficient at boundaries) and easier and faster post-processing of simulation results.

PEG-PIC code is based on stable and well proven code. With our additions and updates, it is still stable and ready for simulations. We also added additional scripts and programs that help users of PEG-PIC process results more efficiently.

New object oriented simulation code (PEG-TC) was developed to test a treecode method. Treecode was first introduced by Barnes and Hut [1] on field of astrophysics to simulate impact of gravitational forces on planets. Main reason to develop new code was ability to explore the flexibility and power of grid-less method, which is essential part of new code. Flexibility of the method is that one does not need to define ant grid in advance (before start of simulation) and to define number of levels in tree, and with this number of nodes in tree.

PEG-TC code is in early stages of development and has to go through a series of testing before it will be ready for use in (general) simulation community. PEG-TC code successfully run some simple cases with success and we are are satisfied with code behaviour. We plan to do more test and only after more complicated tests, we will be able to give final mark about code.

2 Revised and updated PEG-PIC simulation code

2.1 Enable writing history of dump (.dmp) and data (.dat) files

In addition to original functionality of program to write dump and data files in given intervals, defined in input (configuration) file, we added an option to save history of written files. Often simulations take days or even weeks to finish and it is very useful to be able to monitor changing of values during simulation. Here history of dump and data files steps-in and enable this. If values do not go in wanted direction, one could stop the simulation, change input parameters and start the simulation again and thus saving a lot of time in earlier phases of simulation where initial parameters are defined.

We implemented this functionality in two steps:

- first we added a step number to generated dump and data file
- we created break-down version of data file, separate file for each block in original file

First we added current simulation step to both dump and data file names and program generates output files with following names:

- dat file: $< input - file - name >$-000000.dat
- dmp file: $< input - file - name >$-000000.dmp

where $< input - file - name >$ is name of input file and 000000 is replaced with step number. Sequence of steps go from 0 to number of steps defined in input file.

In second step, we generated separate files for each block in original history data file. Source file changed: `dmprest.c`. Now program writes each section of DAT files into separate file to enable easier creation of plots from files. Files that are created are:

- **<X>-000000-all.dat**
  cumulative file
- **<X>-000000-block1.dat**
  block 1 (t(/dt) = 120000, n_grid, V_par (if B>0), ...)
- **<X>-000000-block2.dat**
  block 2 (t/4_pow_x, N(t), LHS_flux, RHS_flux, LHS_Eflu.....)
- **<X>-000000-block3-sp00a.dat**
  block 3 for particle 1 (E, f(E) LHS Wall, f(E) RHS Wall, f1(E), f2(E))
- **<X>-000000-block3-sp00b.dat**
  block 3 for particle 1 (v, f1(v), f2(v), ...)
- **<X>-000000-block3-sp01a.dat**
  block 3 for particle 2 (E, f(E) LHS Wall, f(E) RHS Wall, f1(E), f2(E))
- **<X>-000000-block3-sp01b.dat**
  block 3 for particle 2 (v, f1(v), f2(v), ...)
- **<X>-000000-block4.dat**
  block 4 (Freq., Amp_LHS_Pot, Amp_RHS_Pot)

where <X> is name of input file name. Putting all files `<input-file-name>-block*` together results in fie `<input-file-name>-000000-all.dat`. With dividing file into many files, cumulative file (file that hold all data) is renamed to `<input-file-name>-000000-all.dat`.

2.2 Switch "origdmp" with new values fot output files

We added new values for configuration parameter **origdmp**: in addition to existing values 0 and 1, values 2 and 3 were added to support writing additional values into output file for drawing phase-space diagrams.

Switch **origdmp** in input file controls type of dump file (1=binary, 0=ASCII). All possible parameter values are:

0 = create ASCII dump file

1 = create binary dump file

2 = create ASCII dump files with values (coordinates) with x, Vx, Vy, Vz values that can be used directly for plot drawing

3 = create binary dump file and ASCII files for ph-space diagram

2.3 Output of data for phase-space diagrams

New values for switch **origdmp** were added to enable of adding values to output files for drawing phase-space diagrams. PEG-PIC can create ASCII files with values for plotting phase-space diagrams for step. Files are created at the same steps as dump files (controlled by dmpstep).

Files are named as `<input-file-name>-sp00-000000-ph_space.dat`, where:

- sp00 - defined species (00-first species, 01-second, etc.)
- 000000 - current step (sequence number for file)

Program also creates cumulative file for each species. File is named `<input-file-name>-sp00-all-ph_spa` Putting all files `<input-file-name>-sp00-00*-ph_space.dat` together results in file `<input-file-name>-sp00-all-ph_space.dat`.

2.4 Variable reflection coefficient at boundaries

PEG-PIC enables user to define variable reflection value on left and right boundary through new switches in input file. New switches define reflection coefficient for left and right boundary as follows:

- `gridref_l0` - initial value for left side
- `gridref_lsramp` - how coefficient is changed with time - left side
- `gridref_r0` - initial value for right side
- `gridref_rsramp` - how coefficient is changed with time - right side

Value of coefficient is calculated at given point in time using following equation: `coeff = coeff0 + sizeramp * t`, where:

- **coeff0**: is initial value
- **sizeramp**: defines how coefficient is changed in relation to time
- **t**: time

Values for coefficients are written into data files along with other values for species, into "block2" DAT file.

2.4.1 Program source

To implement needed functionality and options added to configuration file, we changed source code in function `adjust()` that is part of file `padjus.c`. Following source code demonstrates usage of reflection parameters in simulation.

Initialization of values is done first time function is executed.

```
for (isp=0; isp<nsp; isp++) {
    if (inject[isp]) {
        i+=1;
    }

    /* define initial values for grid reflection on left and right side */
    gridref_l[isp] = gridref_l0[isp];
    gridref_r[isp] = gridref_r0[isp];
}
```

Calculating current value for reflection coefficient:

```
for (isp=0; isp<nsp; isp++) {
    if (dinj[isp]) {
        jwall[isp] =0.0;

        if (slow) {           /* initialize f() diagnostic arrays */
            for (s=0; s<NDP; s++) {
                for (ii=0; ii<nbin[isp]; ii++) fe_mid[isp][s][ii] = 0;
                for (ii=0; ii<nbin_fvx[isp]; ii++) f_vpar[isp][s][ii] =0;
            }
        }
    }

    /* calculate current values for grid reflection on left and right side */
    gridref_l[isp] = gridref_l0[isp] + gridref_lsramp[isp] * t;
    gridref_r[isp] = gridref_r0[isp] + gridref_rsramp[isp] * t;
}
```

Absorb or reflect current particle is calculated in following code excerption - for left and right boundary,

```
if (frand() > gridref_r[isp]) {
    /* on RIGHT boundary: particle is absorbed */
    nnp = np[isp] -1;
    x[isp][i] = x[isp][nnp];
    vx[isp][i] = vx[isp][nnp]
    vy[isp][i] = vy[isp][nnp];
    vz[isp][i] = vz[isp][nnp];
    np[isp] -=1;
} else {
    /* particle is reflected back from boundary */
    x[isp][i] = 2*xnc - x[isp][i];
    vx[isp][i] = -vx[isp][i];
}

. . .
. . .
. . .

if (frand() > gridref_l[isp]) {
    /* on LEFT boundary: particle is absorbed */
    nnp = np[isp] - 1;
    x[isp][i] = x[isp][nnp];
    vx[isp][i] = vx[isp][nnp];
    vy[isp][i] = vy[isp][nnp];
    vz[isp][i] = vz[isp][nnp];
    np[isp] -= 1;
    jwall[isp] += q[isp];
} else {
    /* particle is reflected back from boundary */
    x[isp][i] = -x[isp][i];
    vx[isp][i] = -vx[isp][i];
}
```

2.5 Two values in output file for creating movies

In file for movies (...movies.txt), two values are added to each line in file. These values are potential, first is averaged (as all other values), second values is nonaveraged value of potential. Values for potential are written to the end of line, to columns 12 and 13.

3 Set of programs/scripts for post-processing tasks

We developed set of programs/scripts to make post-processing of results faster and easier. For larger simulations, simulation codes usually run on cluster computers, e.g. HPC Prelog (that is installed on Faculty of mechancal engineering, University of Ljubljana), and intermediate and final results are save into files. These files are processed during simulation or at the end of simulation.

3.1 Drawing/displaying distribution functions

We developed wrote short program for preparing data for drawing distribution functions. Program is written in C and should compile and run on any platform that have C compiler. Program was tested on Linux OS. Program automatically defines lower and upper limit for velocity in defined direction (vx, vy or vz).

3.1.1 Input parameters

All parameters for running the program are given via parameters in command line. Input parameters are:

input file - input file from which to read data

vel - which velocity to use for making histogram: v_x, v_y, v_z

K - number of cells in velocity direction

x_1 - lower limit of area of interest

x_2 - upper limit of area of interest

M - number of strips in area of interest, each is Δx wide

Δx - width of strip

3.1.2 Format of input file

Data in input file is arranged in a way that x coordinate is running along rows, data for each value of x is written in columns. Columns are expected to have following values:

Example of input file (few lines):

```
1 0.007865 -5215.25 -0 0
1 0.028205 5236.34 0 0
1 0.015475 1199.52 0 0
1 0.014722 -2238.04 -0 0
```

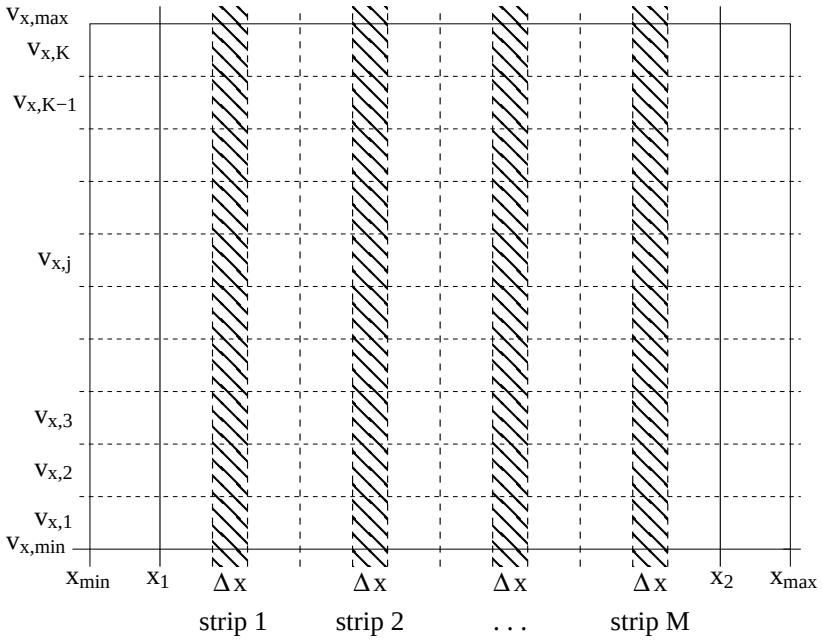

Figure 1: Explanation of input parameters.

column num	mark	description
1	seq	not used; must be present, integer number
2	x	position in x direction
3	vx	velocity in x direction
4	vy	velocity in y direction
5	vz	velocity in z direction

```
1 0.004000 -8702.17 -0 0
1 0.008804 3171.99 0 0
1 0.023986 -755.435 -0 0
```

3.1.3 Output results

Program displays results to standard output (screen, console). The amount of output is small and results can be copied from screen or redirected to file for further use. Histogram results are displayed in table (array) form, where each row represent data for one cell in velocity direction. First column define velocity value in the middle of the cell and other columns show number of values at given velocity in each strip - each strip is displayed in separate column.

Example of output:

```
Results:
-39309.7     7      1
-30570.8   156    111
-21831.9  2429   1588
-13093 24028  13811
-4354.06  7493  24069
4384.86  1069   3177
13123.8   146    413
21862.7     9     34
30601.6     0      0
39340.5     0      0
```

3.2 Comparing (analysis) of values from two or more time steps

Values of variables in simulation at different time steps are written in separate files. To be able to compare values, they should be in the same file or one should import values to some statistical or spreadsheet program and combine them into same sheet/working file. With a few scripts and standard Linux programs, drawing graphs can be dome automatically, without any manual work.

3.2.1 Preparing data before analysis

After simulation data is written into result files (data files), first step is to extract values (columns) of interest from each dump file.

Steps:

1. first extract columns of interest from dump files into separate file. Example: dump file is test.inp-sp01-000000-ph_space.dat and we are interested in columns 3 and 7.
   Commend is (write in same line):

   ```
   cat varrefl.inp-sp01-000000-ph_space.dat |
   awk '{ print $1 " " $7 " " $3}' > file_000.dat

   cat varrefl.inp-sp01-000100-ph_space.dat |
   awk '{ print $1 " " $7 " " $3}' > file_100.dat
   ```

```
cat varrefl.inp-sp01-000200-ph_space.dat |
awk '{ print $1 " " $7 " " $3}' > file_200.dat
```

2. now join columns in three files into one (write in same line):

   ```
   join file_000.dat file_100.dat file_200.dat > file_0-100-200.dat
   ```

   Joined file (file_0-100-200.dat) has now 7 columns, which are as follows:

Column no.	value in column
1	column no. 1 in all files (it is the same)
2	column no. 3 from file file_000.dat
3	column no. 7 from file file_000.dat
4	column no. 3 from file file_100.dat
5	column no. 7 from file file_100.dat
6	column no. 3 from file file_200.dat
7	column no. 7 from file file_200.dat

Now it is possible to import final file (file_0-100-200.dat) into any program for producing graphs and analyze or display differences between columns, which represent same variables at different points in time (run).

3.2.2 Preparing graph in gnuplot

From joined file (file_0-100-200.dat) it is straightforward to display graph(s) using gnuplot tool.

```
plot "a.dat" using 1:3 with lines , "a.dat" using 1:4 with lines
```

3.2.3 Example of preparing file for plotting

Here is an example of preparing data file for drawing graphs for case with 250 time steps. Values from each time step is written into separate ASCII dump file.

First we join all block1 dump files (from each time step) into single large file. Each file is appended to previous one "from side" - columns are added to previous file. Final file has same number of rows as source files and number of columns is increased.

To prepare data for plotting graphs, we have to calculate average values from all time steps for all x positions in file (rows). At the same time, we will extract only portion of data.

To extract and calculate average for values v_x, n, T, E - the shell (bash/sh) script is as follows:

```
awk '{ n=400. ; a = 0; b = 0; c = 0; d = 0; e = 0; f = 0; for (i = 0; i < n; i++) {
        a += $(3+i*12); b += $(8+i*12); f +=  $(2+i*12);
        c += $(4+i*12); d += $(9+i*12); e += $(12+i*12);
     };
   print $1 " " a/n " " c/n " " b/n " " d/n " " e/n " " f/n;
}' joined_block1_file.dat > results01.txt
```

where number 400 represents number of time steps (sets of data = 12 * 400 columns) in final (joined) file.

3.3 Defining number of sets in joined file

Sometimes it is difficult to remember how many time steps were used to produce joined file (block1.dat) - specially in cases when one did not work with files for a long time.

In this case, small shell script comes in very handy. Script calculates number of sets in joined file (= number of time steps). With this information, regenerating data for graphs is easy.

```
#!/bin/sh

COLS_IN_SET=12
COLS_BEGIN=1
CHECK_ROW=2

if [ -z "$1" ]; then
  # no file was given
  echo
  echo "Usage: $0 <file to check> [in which row to check]"
  echo
  exit 1
else
  # save file name for later use
  FILE=$1
fi

if [ ! -z "$2" ]; then
  # if second parameter is given, it's row in which we should check
      number of sets
  CHECK_ROW=$2
fi

# get number of columns in given line
COLS=`head -$CHECK_ROW $FILE | tail -1 | sed -r "s/\s+/ /g" | sed "s/
      /\n/g" | wc -l `

# calc number of sets
SETS=$(( ($COLS - $COLS_BEGIN) / $COLS_IN_SET ))

echo "There are $SETS sets in line in file $FILE (checked in line
      $CHECK_ROW)."
```

4 New grid-less simulation code PEG-TC

To eliminate the need to define grid in our simulations, we decided to try to adopt treecode algorithm, developed by Barnes and Hut in 1986 [1] to field of our research. Simulation code PEG-TC was developed mainly from ground-up, but some parts were taken from other simulation codes, mainly from oopd1 [5] (Berkeley based 1D simulation code). From oopd1, we adopted structure and content of input file (text file divided into blocks).

4.1 Treecode (TC) method

Basic idea behind TC algorithm is to replace particle-to-particle (P2P) interaction with particle-to-cluster (P2C) interactions in a such way, that we reduce in number of necessary interactions (calculations) required without making to much error. With reduced number of clusters in comparison to number of particles, number of required iterations (and thus required simulation time) is reduced. Number of iterations for calculating interaction between particles is in

range of $O(N^2)$, with the use of TC method, number of interactions particle-cluster falls in range of $O(N \log N)$ - this is a huge reduction in number of required calculations. In current plasma simulations, where number of particles is in order of 10^{22} (or 10^{17} super particles), number of interactions particle-particle is large even for today's super computers, or at least computers that are usually used to run simulations.

One of the most important parts of treecode method during computation is decision when to use P2P and when to use P2C interaction. In process of force computing, cluster of particles are replaced with "virtual particle" that is positioned in center of cluster and has charge equal to sum of charges of all particles in cluster (figure 2):

$$q_C = \sum_{j=1}^{N_C} q_j \qquad ; \qquad x_C = \frac{1}{N_C} \sum_{j=1}^{N_C} x_j \tag{1}$$

Figure 2 present situation of computing force on particle i in lower left corner of the square caused by particles in blue circle (cluster). This case be done in two ways:

- **P2P**: compute and sum forces caused by all particles in cluster to particle i
- **P2C**: compute force caused by whole cluster to particle i, when whole cluster is taken as single particle.

P2P interaction is used for all clusters that are close to particle, for clusters "far away" from particle, P2C interaction is used. Limit value, when to use P2P or P2C interaction is defined in input file via parameter **p2cRatio** (listing on figure 5).

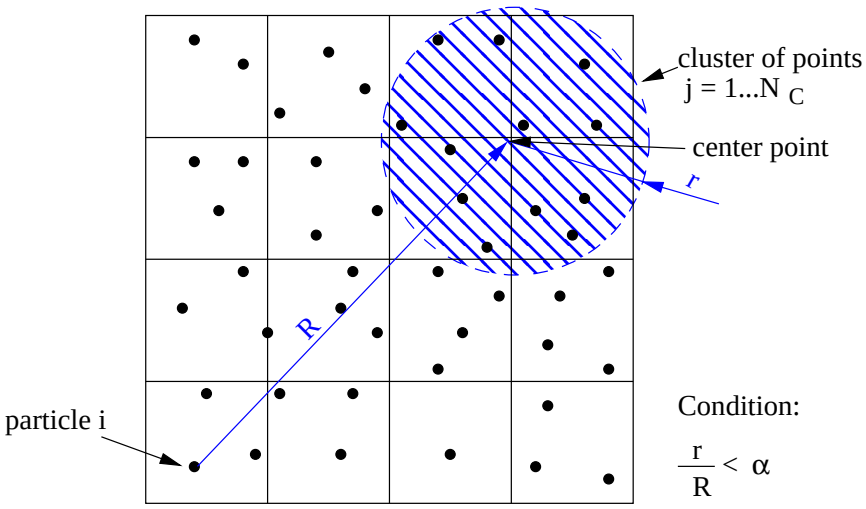

Figure 2: How clusters are defined and used in force calculation.

4.2 TC method in PIC simulation loop

TC method can be used in PIC simulation to replace some steps in main PIC simulation loop - figure 3 shows steps in main PIC loop that are replaced with TC method. New (with TC method) PIC main loop is shown on figure 4. Because in each time step (in each loop) in PIC simulation new particles are injected into simulation domain, tree has to be generated again.

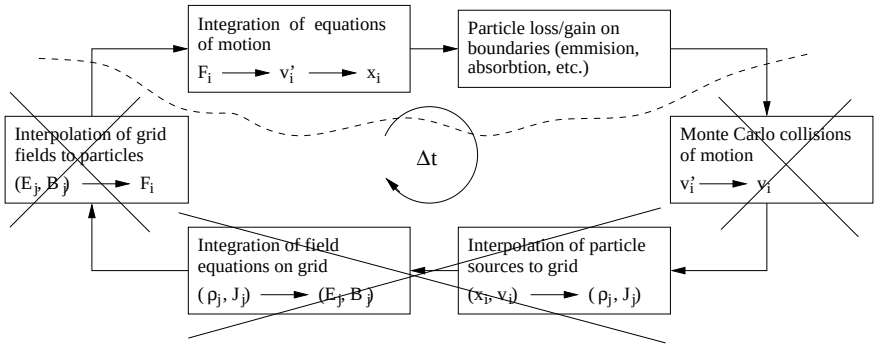

Figure 3: Steps in main PIC loop that are not necessary when using treecode.

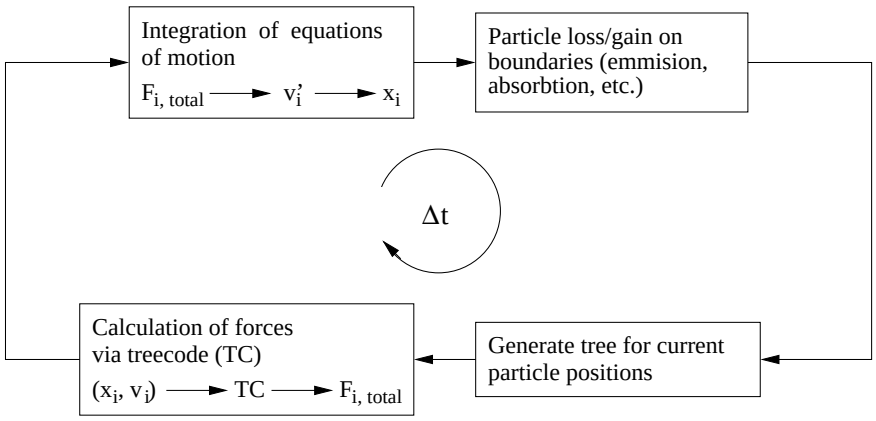

Figure 4: Simple PIC main loop with treecode.

Compared to original treecode method we added one extra parameter with which one can limit tree depth. In input file, it is possible to define maximal number of particles in cluster - not necessary one (how it is defined in original algorithm).

4.3 Running the code and processing results

PEG-TC simulation code does not have any graphical user interface and utilize input files for defining simulation parameters. Input files are ordinary textual files that can be created and manipulated with various text editors. Structure

of input file looks a lot like C program and is based on the input file of another Berkeley based simulation code - OOPD1. With original input file, it shares most of main blocks, but also introduces one new block - block to define treecode parameters (see example on figure 6.

PIC-TC recognize following blocks in input file:

- **Variables**
  Defines variables and theirs values for use later in input file.
- **Species**
  Defines properties of species. Input file can have multiple **Species** blocks.
- **Treecode**
  Defines parameters for computation via treecode method.
- **SpatialRegion**
  Defines properties of simulation partial region, such at simulation **dt**, number of steps, cross section area, boundary charges, etc.
- **Grid**
  Defines grid parameters for computation. Grid is only used to compute potential values on grid. Block **Grid** has to be inside block **SpatialRegion**.
- **InitLoad** and **Source**
  Blocks define particle initial load and source of particles for use inside each computational step. Blocks can have multiple blocks **Particles**. Blocks are not mandatory.
- **Particles**
  Defines properties of particles inside **InitLoad** or **Source** blocks. Block define particle species, inject velocities and positions, their distribution, etc. Block has to be inside **InitLoad** or **Source**.
- **Diagnostics**
  Defines where to save simulation results and holds blocks that define which simulation parameters to write to output files.
- **PhaseSpace**
  Defines parameters for writing phase space data to output file: when to write it (at which simulation times), name of file, etc.
- **PotentialProfile**
  Defined parameters for writing potential values to output file.

Newly introduced block **Treecode** enables one to define following values:

- **N0**
  Parameter **N0** defines maximal number of particles in each tree node. To follow original treecode algorithm, one can define $N0 = 1$.
- **nodeLengthRatio**
  Parameter **nodeLengthRatio** defines maximal tree node size compared to system size. Value of parameter can range from $[0 \rightarrow 1]$. Value of 0 disabled checking of node size.

```
TreeCode
{
  N0 = 10;
  nodeLengthRatio = 0.1;
  p2cRatio = 0.02;
}
```

Figure 5: Block in input file for defining parameters for treecode method.

- **p2cRatio**
  Parameter **p2cRatio** defines limit value for ratio between tree node size (also cluster size) and distance between certain point and center of node (cluster) (see figure 2). During computation, if ratio for given particle and cluster is below parameter value, particle-to-particle is used, otherwise particle-to-cluster interaction is used for computation of force.

Two parameters (N0 and nodeLengthRatio) limit tree node size, p2cRatio influence computational speed and simulation error. Added block in in input file is displayed on figure 5.

Simulation results are saved into series of output files, as defined in input file in block Diagnostics. Post-processing of results is done with series of Bash scripts, tailored to output format generated by PEG-TC. Scripts enable users to generate images and movies of potential profiles, phase-space diagrams, comparison of values at different simulation steps, etc. More sophisticated results analysis could be done using one of professional chart or statistical programs.

4.4 PEG-TC source code

Source code for PEG-TC simulation code is not available for download due to it's young development stage, far from "production ready" state and the fact that requires some "fine-tuning" of parameters to produce usable results. Because of this, it is available upon request - sent to email janez.krek@lecad.fs.uni-lj.si.

5 Discussion

In this work we present revisions and updates we made on existing PEG-PIC code to implement functionality we need for current project. Because simulation code is stable, our focus was on adding missing functionality and development if miscellaneous tools that make use of PEG-PIC code easier and time effective. With this in mind, we developed few simple but effective scripts (support programs) to enable users of PEG-PIC easy post-processing. Post-processing is getting more and more time demanding as simulation codes usually run on clusters and can produce large quantity of results, usually save in text or binary files.

Second part of our work on field of development simulation codes was development of new PEG-TC simulation code which is based on not so commonly used tree-code method for computing forces between particles in system. Tree-code method was first presented in 1986 and is well proven in multi-body systems [2] (e.g. astrophysics). With most noticeable property of method is grid-less model - there is no need to define grid in advance which saves some time in

cases where we do many simulations. Tree-code method is specially effective in case with low density of simulation gas and small number of particles - in this cases we expect substantial speed increase compared to PEG-PIC. Disadvantage of the code at the moment is its development stage. Code is in early development stage and is practically untested and unproven when compared to other simulation codes, e.g. PEG-PIC, BIT1, xpdp1, oopd1, etc.

Next steps in development of both simulation codes will be to test the codes on Prelog HPC regarding speed and code stability. Despite early development stage of PEG-TC, we believe that code will be stable and will bring considerable speed gains in certain simulation cases.

6 Acknowledgments

This work is supported by Slovenian Ministry of Science and Technology by grant No L2-3652 "Raziskava in razvoj integriranih prenapetostnih zaščitnih naprav na osnovi plinskega odvodnika (GDT) v smeri zanesljive miniaturizirane tehnične rešitve (MINIGDT)".

This work was supported by the European Commission under the Contract of Association between EURATOM and the Austrian Academy of Sciences. It was carried out within the framework of the European Fusion Development Agreement. The views and opinions expressed herein do not necessarily reflect those of the European Commission. This work was also supported by the Austrian Science Fund (FWF).

References

[1] J. Barnes and P. Hut. A Hierarchical O(NlogN) Force-Calculation Algorithm. *Nature*, 324:446–449, December 1986.

[2] A. J. Christlieb, R. Krasny, and J. P. Verboncoeur. A treecode algorithm for simulating electron dynamics in a penning-malmberg trap. *Computer Physics Communications*, 164(1-3):306–310, 2004.

[3] K. Matyash, R. Schneider, R. Sydora, and F. Taccogna. Application of a grid-free kinetic model to the collisionless sheath. *Contributions to Plasma Physics*, 48(1-3):116–120, 2008.

[4] D. Tskhakaya and R. Schneider. Optimization of pic codes by improved memory management. *J. of Comp. Phys.*, 225:829–839, 2007.

[5] J. P. Verboncoeur, A. B. Langdon, and N. T. Gladd. An object-oriented electromagnetic PIC code. *Computer Physics Communications 87*, 1995.

[6] John P. Verboncoeur. Partice-in-cell techniques. Technical report, Department of Nuclear Engineeringn, University of California, Berkeley, CA-94720-1730, May 2007.

7 Appendix A

Input file for PIC-TC simulation code is regulat textual file that enables to define all simulation parameters. Example of input file is shown on figure

```
test_matlab_PTSG_case.inp ---
{
Species // the first species
{
   name = electrons
   q = -1.6E-19  // [C] species charge
   m = 9.1E-31  // [kg] species mass; can be variable
}

SpatialRegion
{
  dt = 1E-11  // [s] simulation time step
  ns = 2000  // [-] number of steps to run
  area = 1e-4 // [m2] cross section of region
  p2c = 1  // [-] number of real particles per computer particle 1 comp
      . particle = p2c * real particles
  V0 = 0.0  // [V] charge on left boundary
  V1 = 0.0  // [V] charge on right boundary

  Grid
  {
  x0 = 0.  // [m] dimension of region -- start len
  x1 = 0.06 // [m] dimension of region -- end len
  ng = 500  // [-] number of grid cells
  }
} // ends SpatialRegion

TreeCode
{
  N0 = 30;
  nodeLengthRatio = 1e-6;
  p2cRatio = 0.2;
}

Diagnostics
{
  directory = ./data

  PhaseSpace
  {
    file = phase-space_%07d.dat
    n_step = 10 // write data every 10 steps
  }

  PotentialProfile
  {
    file = potential_%07d.dat
    files = 50
    disabled  // disable diagnostics
  }
}
}
```

Figure 6: Small example of input file.