EDIPIC-2D
Description of the code algorithm

Dmytro Sydorenko

Some history

- The code is being developed since 2014.
- Primary developer is Dmytro Sydorenko.
- Random number generator interface and the core of the PETSc-based field solver are written by Salomon Janhunen.
- Funding:
  - AFOSR (University of Saskatchewan, Andrei Smolyakov),
  - DOE (PPPL, Igor Kaganovich).
- Source files of the code are uploaded to *github.com/PrincetonUniversity/EDIPIC-2D* together with a sample set of input data files, description of input and output data files, several programs for processing the output, compilation instructions.
- The code will have an open source license.

Outline

- Review of main code features
- Flowchart of the algorithm
- Walk through the main program of the code

EDIPIC-2D: general code features 1

- 2d3v Particle-in-Cell
- Cartesian geometry is used, simulation domain is a rectangle (in plane x,y).
- Explicit leap-frog algorithm
- Boris scheme of particle advance
- Self-consistent electrostatic field:
  - FFT-based field solver for systems periodic along X
  - PETSc based field solver for bounded systems and systems periodic along X and Y
- Externally defined nonuniform magnetic field constant in time
- Subcycling of electrons relative to ions
- Multiple ion species
- Walls may emit particles.
- Monte-Carlo model of electron-neutral collisions:
  - Multiple neutral species with nonuniform density

Advantages and drawbacks

- Simplicity
- Minimal effects of the numerical scheme on the physics

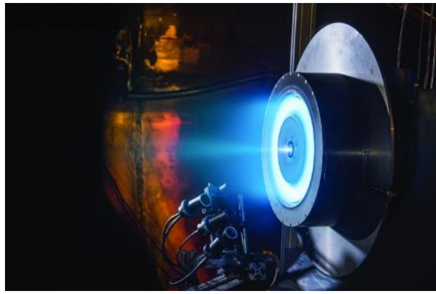
- Plasma period and Debye length must be resolved, which increases the numerical cost.

- Methods of solution: implicit algorithms and/or parallelization.

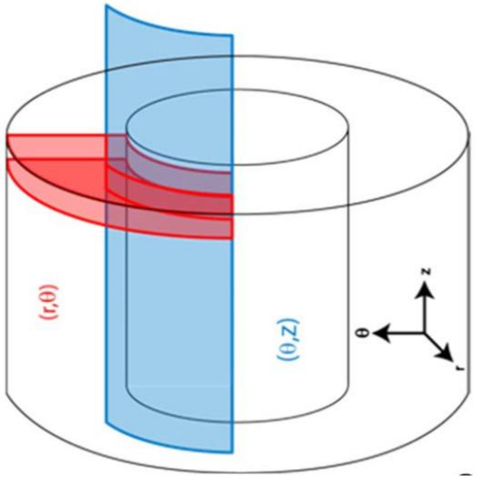
EDIPIC-2D: general code features, 2

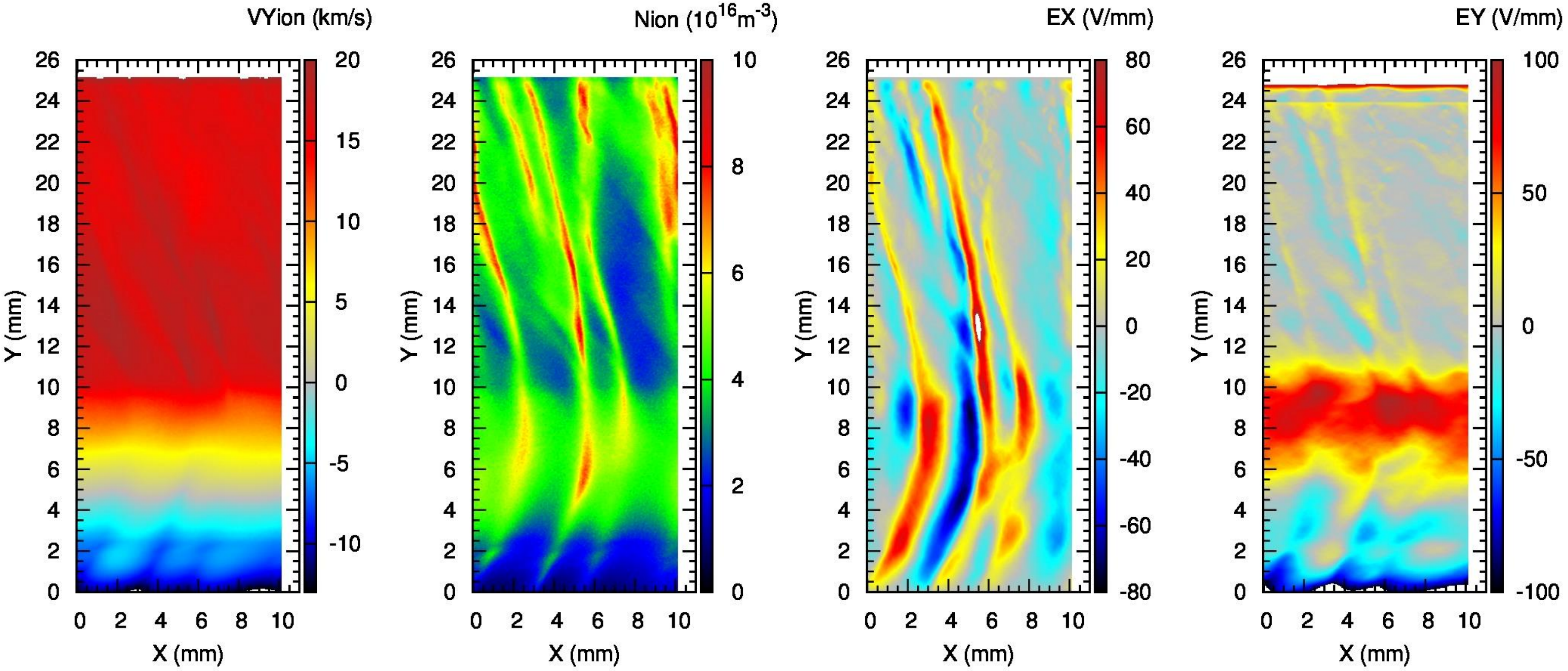
- Code works on CPUs, is written in Fortran 90, parallelized with MPI.
- The code combines domain decomposition and particle sharing.
- Special methods ensure even particle load between CPU cores.
- Abundant diagnostics output:
  - Time dependencies (potential Φ, electric fields $E_{X,Y}$, densities $n_{e,i}$) in probes;
  - 2D snapshots (potential Φ, electric fields $E_{X,Y}$, electric currents $J_{X,Y,Z;e,i,sum}$, densities $n_{e,i}$, temperatures $T_{X,Y,Z;e,i}$, energies $W_{X,Y,Z;e,i}$, flow velocities $V_{X,Y,Z;e,i}$);
  - Electron and ion velocity distribution functions over 1 velocity component $f_{e,i}(v_{X,Y,Z})$ and electron velocity distribution functions over 2 velocity components $f_e(v_X,v_Y)$
  - Electron and ion particle data (coordinates X,Y , velocities $V_{X,Y,Z}$, tags) for particles within pre-defined regions.
- An interrupted simulation may continue from a checkpoint.

Study of Hall thruster plasmas



- Azimuthal-axial (slice r=const) or radial-azimuthal (slice z=const) setup
- The code participated in 2 international benchmarks (see below)





Axial ion velocity

Ion density

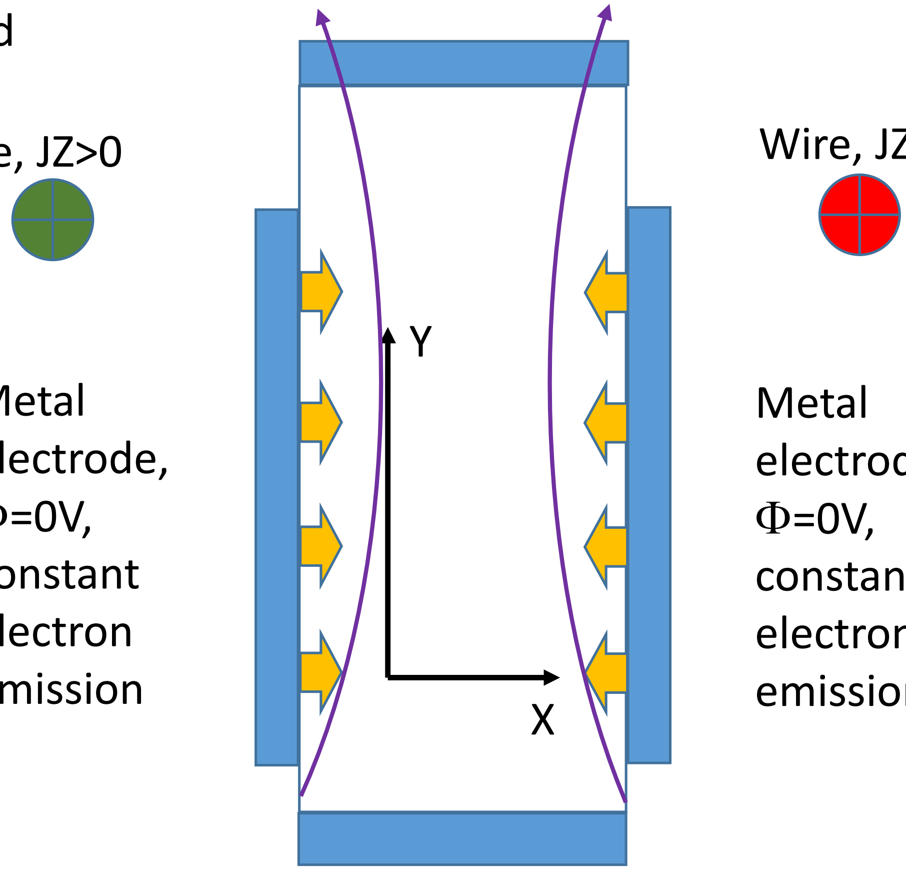
Azimuthal electric field

Axial electric field

Snapshots from a simulation of the azimuthal-axial cross-section of a Hall thruster. Anode is in the bottom, cathode in the top.

Example of a hollow cathode DC system

- The whole system is 241x481 nodes or 8mmx16mm
- 32 MPI processes
- Elastic, excitation, ionization collisions with neutrals (Ar)
- Full set of input data files on GitHub



External constant (in time) magnetic field

Metal electrode, Φ=+80V, no emission

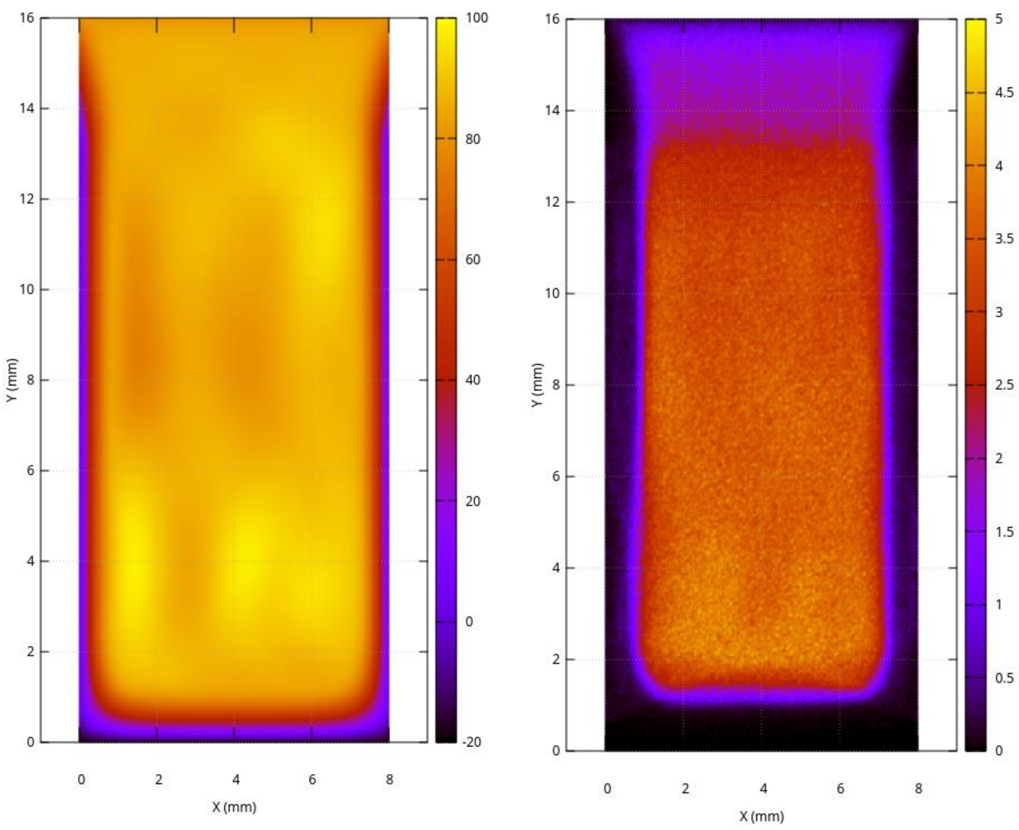
Wire, JZ>0

Wire, JZ<0

Metal electrode, Φ=0V, constant electron emission

Metal electrode, Φ=0V, constant electron emission

Metal electrode, Φ=-20V, no emission



Potential [V]

Electron density [10^{16} m^{-3}]

Snapshots are at t=100ns

Balancing of particle load between MPI processes

- The whole simulation domain splits into relatively large particle subdomains.
- Particles from one subdomain are processed by several MPI processes (CPU cores).
- The number of cores processing particles in a subdomain is proportional to the total number of particles in the subdomain.
- This number is periodically updated during simulation.

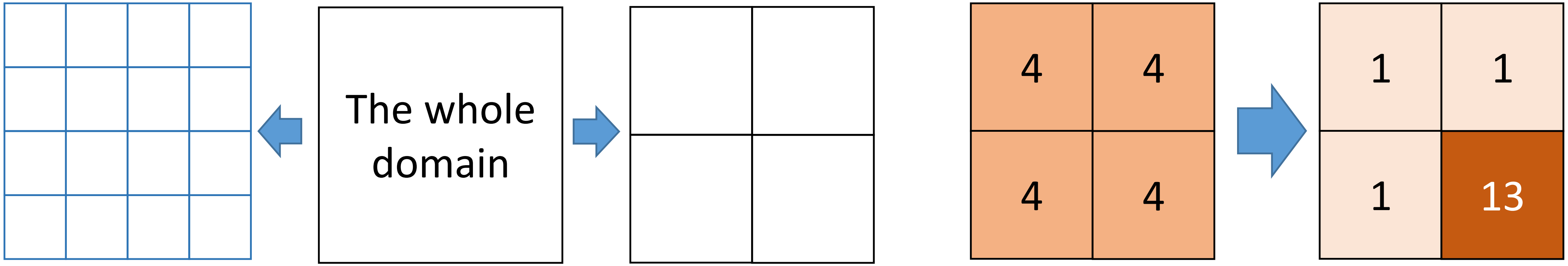
Here is an example for 16 CPU cores and 4 particle subdomains:

For comparison, smaller subdomains for field solver (blocks)

Subdomains for particle processing (clusters)

Numbers of CPU cores per particle subdomain for uniform density

Numbers of CPU cores per particle subdomain for **density peaking in the bottom right corner** (extreme case)



4	4
4	4

1	1
1	13

Scaling of the code performance is close to linear for up to 128 CPU cores

Simulation #	Number of electron particles N_{part} (millions)	Number of CPU cores N_{CPU}	Time per electron cycle τ (s)	Scale time α (s)
1	74.7	32	0.31	**0.133**
2	151.4	64	0.31	**0.131**
3	293.9	128	0.31	**0.135**

The scale time α is the time per computational cycle when there is 1 million particles per CPU core.

The scale time barely changes as the number of cores quadruples.

If the numerical cost of advancing particles is dominant and the scaling of the code performance is linear, time per computational cycle satisfies $\tau = \alpha\, N_{part}/N_{CPU}$, where the scale time α is a constant.

Analysis of the code scaling is based on simulations performed on the *Cedar* cluster of *computecanada.ca* . The axial-azimuthal Hall thruster discharge setup periodic in the azimuthal direction was used. An FFT based Poisson's equation solver was used which has minimal numerical cost.

Code Benchmarking

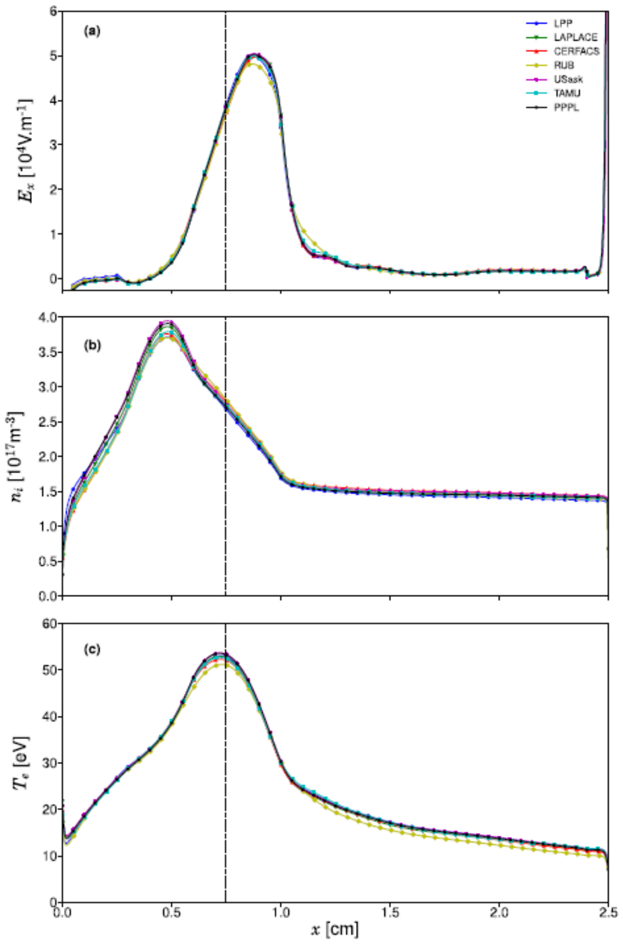
- Code participated in two benchmarks for Hall thruster plasmas (carried out at the University of Saskatchewan):
  - T. Charoy et al., “2D **axial-azimuthal** particle-in-cell benchmark for low-temperature partially magnetized plasmas”, Plasma Sources Sci. Technol., vol.28, 105010 (2019).
  - W. Villafana et al., “2D **radial-azimuthal** Particle-In-Cell benchmark for ExB discharges”, submitted to Plasma Sources Sci. Technol. (2021).
- Very good agreement with other 6 codes (1st benchmark) and 5 codes (2nd benchmark).

The profiles are averaged azimuthally and over time.

Axial electric field

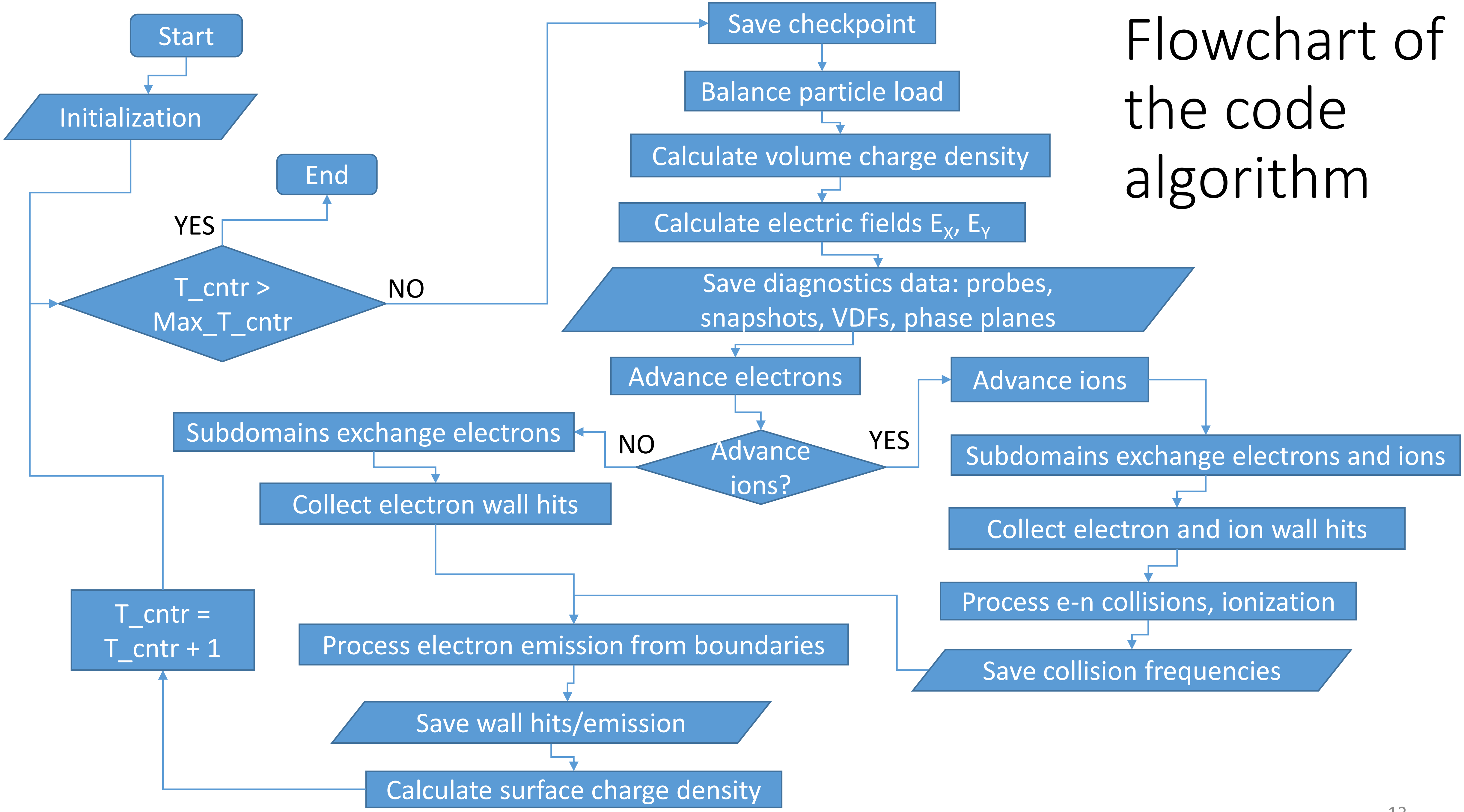
Ion density

Electron temperature



From T.Charoy et al., PSST, 28, 105010 (2019)

Flowchart of the code algorithm



Start

Initialization

T_cntr > Max_T_cntr

YES

End

NO

Save checkpoint

Balance particle load

Calculate volume charge density

Calculate electric fields E_X, E_Y

Save diagnostics data: probes, snapshots, VDFs, phase planes

Advance electrons

Advance ions?

YES

Advance ions

Subdomains exchange electrons and ions

Collect electron and ion wall hits

Process e-n collisions, ionization

Save collision frequencies

NO

Subdomains exchange electrons

Collect electron wall hits

Process electron emission from boundaries

Save wall hits/emission

Calculate surface charge density

T_cntr = T_cntr + 1

- The following slides contain essential parts of the pic2d_MainProgram.f90 file .
- For the sake of clarity, most comment lines, commented calls, MPI synchronization commands (MPI_BARRIER), timer calls (MPI_WTIME) are omitted.

Initialization (1)

```
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, Rank_of_process, ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, N_of_processes, ierr)

CALL PrepareMaxwellDistribIntegral

Start_T_cntr = 0
T_cntr_global_load_balance = Start_T_cntr
T_cntr_cluster_load_balance = Start_T_cntr

CALL INITIATE_PARAMETERS

CALL INITIATE_ELECTRON_NEUTRAL_COLLISIONS

CALL INITIATE_PROBE_DIAGNOSTICS

CALL INITIATE_WALL_DIAGNOSTICS

CALL INITIATE_en_COLL_DIAGNOSTICS

CALL INITIATE_SNAPSHOTS
```

Standard MPI initialization procedures

Prepares arrays required by functions returning velocities corresponding to a Maxwellian distribution and a Maxwellian*V distribution used for particle injection.
See file pic2d_ElectronWallCollisions.f90

Initial value of the time step counter, may be modified if simulation starts from checkpoint

Value of the time step counter when the global balancing procedure should be applied

Value of the time step counter when load balancing procedure within a cluster only is applied

Initialization (2)

```
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, Rank_of_process, ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, N_of_processes, ierr)

CALL PrepareMaxwellDistribIntegral

Start_T_cntr = 0
T_cntr_global_load_balance = Start_T_cntr
T_cntr_cluster_load_balance = Start_T_cntr

CALL INITIATE_PARAMETERS

CALL INITIATE_ELECTRON_NEUTRAL_COLLISIONS

CALL INITIATE_PROBE_DIAGNOSTICS

CALL INITIATE_WALL_DIAGNOSTICS            !

CALL INITIATE_en_COLL_DIAGNOSTICS

CALL INITIATE_SNAPSHOTS
```

This subroutine does the following:

- reads file init_configuration.dat which defines scale values, simulation domain dimensions, grid resolution, splitting of the domain between processes, boundaries;
- reads file init_simcontrol.dat which in particular defines duration of simulation, subcycling, use of checkpoints and balance loading;
- organizes the cluster structure required for balance loading – selects processes which will be masters of clusters or just particle movers, establishes communications between cluster masters;
- sets boundary objects;
- prepares field solvers – FFT based for semi-periodic rectangular domains, PETSc based otherwise;
- prepares external fields;
- prepares initial particle coordinates and velocities, if necessary from a previously saved checkpoint data file.

See file pic2d_CurProblemValues.f90

Initialization (3)

```
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, Rank_of_process, ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, N_of_processes, ierr)

CALL PrepareMaxwellDistribIntegral

Start_T_cntr = 0
T_cntr_global_load_balance = Start_T_cntr
T_cntr_cluster_load_balance = Start_T_cntr

CALL INITIATE_PARAMETERS

CALL INITIATE_ELECTRON_NEUTRAL_COLLISIONS

CALL INITIATE_PROBE_DIAGNOSTICS

CALL INITIATE_WALL_DIAGNOSTICS           !

CALL INITIATE_en_COLL_DIAGNOSTICS

CALL INITIATE_SNAPSHOTS
```

This subroutine does the following:

- reads file init_neutrals.dat which defines content, densities, and temperatures of neutral species;
- for each neutral species AAAAAA reads file init_neutral_AAAAAA.dat defining which collisional processes between electrons and this neutral species are activated;
- for each activated collisional process reads file init_neutral_AAAAAA_crsect_coll_id_NN_type_MM.dat with cross sections vs energy;
- prepares probability arrays for the null-collision algorithm.

See file pic2d_enCollisionsGeneralProc.f90

Initialization (4)

```
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, Rank_of_process, ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, N_of_processes, ierr)

CALL PrepareMaxwellDistribIntegral

Start_T_cntr = 0
T_cntr_global_load_balance = Start_T_cntr
T_cntr_cluster_load_balance = Start_T_cntr

CALL INITIATE_PARAMETERS

CALL INITIATE_ELECTRON_NEUTRAL_COLLISIONS

CALL INITIATE_PROBE_DIAGNOSTICS

CALL INITIATE_WALL_DIAGNOSTICS

CALL INITIATE_en_COLL_DIAGNOSTICS

CALL INITIATE_SNAPSHOTS
```

This subroutine reads file init_probes.dat and prepares locations (probes) where time dependencies of Φ, $E_{X,Y}$, $N_{e,i}$ will be saved. If simulation starts from a checkpoint, trims existing time dependence data files.
See file pic2d_TimeDependences.f90

This subroutine prepares files where numbers of particles that collided with or were emitted by the boundary object are saved [at each time step].
See file pic2d_ElectronWallCollisions.f90

This subroutine prepares files where numbers of electron-neutral collision events of each kind for each neutral species are saved [at each ion time step].
See file pic2d_enCollisionsGeneralProc.f90

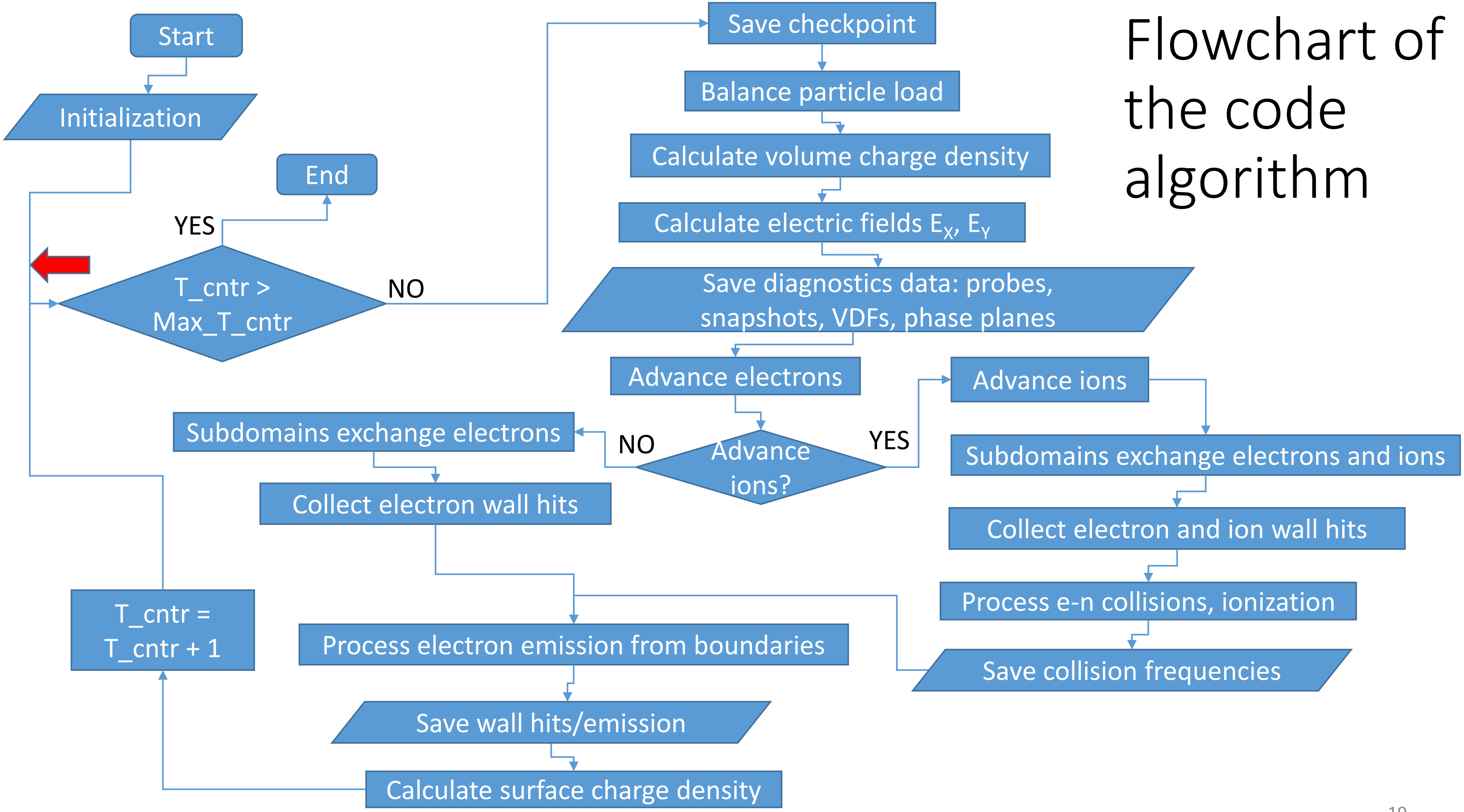
Initialization (5)

```
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, Rank_of_process, ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, N_of_processes, ierr)

CALL PrepareMaxwellDistribIntegral

Start_T_cntr = 0
T_cntr_global_load_balance = Start_T_cntr
T_cntr_cluster_load_balance = Start_T_cntr

CALL INITIATE_PARAMETERS

CALL INITIATE_ELECTRON_NEUTRAL_COLLISIONS

CALL INITIATE_PROBE_DIAGNOSTICS

CALL INITIATE_WALL_DIAGNOSTICS

CALL INITIATE_en_COLL_DIAGNOSTICS

CALL INITIATE_SNAPSHOTS
```

This subroutine reads file init_snapshots.dat which controls the set of data to save: 2d maps of various parameters, velocity distribution functions, particle data. It also defines when to create snapshots.

See file pic2d_Snapshots.f90

Flowchart of the code algorithm


Start
Initialization
End
YES
T_cntr > Max_T_cntr
NO
Save checkpoint
Balance particle load
Calculate volume charge density
Calculate electric fields E_X, E_Y
Save diagnostics data: probes, snapshots, VDFs, phase planes
Advance electrons
Advance ions?
NO
YES
Subdomains exchange electrons
Collect electron wall hits
Advance ions
Subdomains exchange electrons and ions
Collect electron and ion wall hits
Process e-n collisions, ionization
Save collision frequencies
Process electron emission from boundaries
Save wall hits/emission
Calculate surface charge density
T_cntr = T_cntr + 1

Main cycle

```
n_sub = 0

DO T_cntr = Start_T_cntr, Max_T_cntr
.........................
END DO
```

This counter is used to define when it is necessary to move ions (remember that electrons are subcycled relative to ions).

This is the main cycle. When it is done, ths simulation is over. What is inside is described below.

Save checkpoint

Checkpoint is a file which contains all data required to restore a system state and continue a simulation at some point in time. To minimize amount of data saved into the checkpoint, the checkpoint is created immediately after the ions are advanced (in this case the accumulated electric field used in ion motion equations is zero). The IF ensures that the checkpoint is saved at the proper time step.

This subroutine saves data into a checkpoint file (in binary format). Variable n_sub is used for an additional safety check, if it is nonzero this is an error.
See file pic2d_Checkpoints.f90

Here the time step for the next checkpoint is updated. The value of the interval between checkpoints dT_save_checkpoint is an integer number times the number of electron subcycles per ion time step, see file pic2d_CurProblemValues.f90 .

```
IF (T_cntr.EQ.T_cntr_save_checkpoint) THEN
   CALL SAVE_CHECKPOINT_MPIIO_2(n_sub)
   T_cntr_save_checkpoint = T_cntr_save_checkpoint + dT_save_checkpoint
END IF

call report_total_number_of_particles
```

This is a handy run-time diagnostics procedure which prints a line with the total number of particles, X,Y,Z momentum, and energy for electrons and each ion species.
See file pic2d_Diagnostics.f90

Balance particle load (1)

```
IF (T_cntr.EQ.T_cntr_global_load_balance) THEN
   IF (n_sub.NE.0) THEN
      PRINT '("Process ",i5," :: ERROR-1 in MainProg :: GLOBAL_LOAD_BALANCE is about to be
called at wrong time :: T_cntr = ",i8," n_sub = ",i8)', Rank_of_process, T_cntr, n_sub
      STOP
   END IF
   CALL GLOBAL_LOAD_BALANCE  ! includes calls to SET_COMMUNICATIONS
                             !                   DISTRIBUTE_CLUSTER_PARAMETERS
   T_cntr_global_load_balance = T_cntr_global_load_balance + dT_global_load_balance
END IF

IF (T_cntr.EQ.T_cntr_cluster_load_balance) THEN
   CALL BALANCE_LOAD_WITHIN_CLUSTER
   T_cntr_cluster_load_balance = T_cntr_cluster_load_balance + dT_cluster_load_balance
END IF
```

This subroutine performs global balancing of particle load. This procedure may require transfer of large amount of data between MPI processes which is why it should not be called too often. Numbers of particles per process are compared between all clusters, then clusters with lower load release some processes which are reassigned to clusters with higher load [if necessary]. A process departing a cluster leaves its particles to the cluster master process. A process joining a cluster initially has zero particles. Redistribution of particles within the cluster occurs in the subsequent call of BALANCE_LOAD_WITHIN_CLUSTER.
See file pic2d_LoadBalancing.f90

Balance particle load (2)

```
IF (T_cntr.EQ.T_cntr_global_load_balance) THEN
   IF (n_sub.NE.0) THEN
      PRINT '("Process ",i5," :: ERROR-1 in MainProg :: GLOBAL_LOAD_BALANCE is about to be
called at wrong time :: T_cntr = ",i8," n_sub = ",i8)', Rank_of_process, T_cntr, n_sub
      STOP
   END IF
   CALL GLOBAL_LOAD_BALANCE  ! includes calls to SET_COMMUNICATIONS
                             !                   DISTRIBUTE_CLUSTER_PARAMETERS
   T_cntr_global_load_balance = T_cntr_global_load_balance + dT_global_load_balance
END IF

IF (T_cntr.EQ.T_cntr_cluster_load_balance) THEN
   CALL BALANCE_LOAD_WITHIN_CLUSTER
   T_cntr_cluster_load_balance = T_cntr_cluster_load_balance + dT_cluster_load_balance
END IF
```

This subroutine performs balancing of particle load within a cluster (sub-domain). It redistributes particles between processes working on the same cluster, but it does not change the list of these processes. This procedure requires fewer communications than the GLOBAL_LOAD_BALANCE and is called more frequently.

See file pic2d_LoadBalancing.f90

Balance particle load (3)

```
IF (T_cntr.EQ.T_cntr_global_load_balance) THEN
   IF (n_sub.NE.0) THEN
      PRINT '("Process ",i5," :: ERROR-1 in MainProg :: GLOBAL_LOAD_BALANCE is about to be
called at wrong time :: T_cntr = ",i8," n_sub = ",i8)', Rank_of_process, T_cntr, n_sub
      STOP
   END IF
   CALL GLOBAL_LOAD_BALANCE  ! includes calls to SET_COMMUNICATIONS
                             !                   DISTRIBUTE_CLUSTER_PARAMETERS
   T_cntr_global_load_balance = T_cntr_global_load_balance + dT_global_load_balance
END IF

IF (T_cntr.EQ.T_cntr_cluster_load_balance) THEN
   CALL BALANCE_LOAD_WITHIN_CLUSTER
   T_cntr_cluster_load_balance = T_cntr_cluster_load_balance + dT_cluster_load_balance
END IF
```

Calls of balancing procedures occur at time steps satisfying the following rules:
ions were advanced at the previous time step (n_sub is zero);
BALANCE_LOAD_WITHIN_CLUSTER is called more often than GLOBAL_LOAD_BALANCE;
a call of GLOBAL_LOAD_BALANCE is always followed by a call of BALANCE_LOAD_WITHIN_CLUSTER.
This is ensured by dT_cluster_load_balance = an_integer_number * N_subcycles (number of electron subcycles per ion step) and dT_global_load_balance = an_integer_number * dT_cluster_load_balance, see file pic2d_CurProblemValues.f90 .

Calculate volume charge density

```
IF (n_sub.EQ.0) CALL GATHER_ION_CHARGE_DENSITY

CALL GATHER_ELECTRON_CHARGE_DENSITY
```

This subroutine calculates charge density of ions in the grid nodes. It is called at the time step immediately after the time step when the ions were advanced to account for ions produced in ionization. This density is used in the Poisson's equation until the next ion move.
See file pic2d_IonDynamics.f90

This subroutine calculates charge density of electrons in the grid nodes. It is called at each time step.
See file pic2d_ElectronDynamics.f90

Charge density arrays are stored in cluster master processes for semi-periodic domains which use FFT-based field solver. For non-periodic or double-periodic domains, where the PETSc-based solver is applied, the densities are transferred to field solver processes (blocks).

Calculate electric fields E_X, E_Y (1)

Domain boundaries may be electrodes with the potential being given functions of time, $\Phi=\Phi_0+\Phi_{var}\sin(\omega t+\phi)$. This subroutine calculates potentials of all boundary objects where the potential is given.
See file pic2d_WallPotentials.f90

```
CALL UPDATE_WALL_POTENTIALS(T_cntr)

IF ((periodicity_flag.EQ.PERIODICITY_NONE).OR.(periodicity_flag.EQ.PERIODICITY_X_Y)) THEN
   CALL SOLVE_POTENTIAL_WITH_PETSC
   CALL CALCULATE_ELECTRIC_FIELD
ELSE IF (periodicity_flag.EQ.PERIODICITY_X) THEN
   CALL SOLVE_POISSON_FFTX_LINSYSY
   CALL CALCULATE_ELECTRIC_FIELD_FFTX_LINSYSY
END IF
```

This subroutine solves Poisson’s equation using a PETSc based solver. The electrostatic potential is stored in the field calculator processes.
See file pic2d_ElectricFieldCalc_PETSc.F90

This subroutine calculates components of the electric field vector $E_{X,Y}$ inside a whole cluster when the electrostatic potential is split between the cluster field calculators (blocks).
See file pic2d_ElectricFieldCalc_PETSc.F90

This branch is involved for non-periodic or double-periodic domains.

Calculate electric fields E_X, E_Y (2)

```
CALL UPDATE_WALL_POTENTIALS(T_cntr)

IF ((periodicity_flag.EQ.PERIODICITY_NONE).OR.(periodicity_flag.EQ.PERIODICITY_X_Y)) THEN
   CALL SOLVE_POTENTIAL_WITH_PETSC
   CALL CALCULATE_ELECTRIC_FIELD
ELSE IF (periodicity_flag.EQ.PERIODICITY_X) THEN
   CALL SOLVE_POISSON_FFTX_LINSYSY
   CALL CALCULATE_ELECTRIC_FIELD_FFTX_LINSYSY
END IF
```

This subroutine solves Poisson's equation using an FFT based solver. The electrostatic potential is stored in the cluster master processes.
See file pic2d_ElectricFieldCalc_FFT_X.f90

This subroutine calculates components of the electric field vector $E_{X,Y}$ inside the cluster when the electrostatic potential inside the whole cluster is known.
See file pic2d_ElectricFieldCalc_FFT_X.f90

This branch is involved for rectangular domains periodic along the X direction.

Save diagnostics data

```
CALL DO_PROBE_DIAGNOSTICS(n_sub)

CALL CREATE_SNAPSHOT
```

This subroutine saves values of Φ, $E_{X,Y}$, $N_{e,i}$ in the probes. n_sub is necessary to avoid accumulation of ion density if the interval between saving is less than the ion time step.
See file pic2d_TimeDependences.f90

This subroutine creates snapshots of numerous plasma parameters, including

- 2d profiles of electrostatic potential, electric fields, particle densities, flow velocities, energies, temperatures, electric currents,
- velocity distributions over 1 and 2 velocity components,
- particle data.

See file pic2d_Snapshots.f90

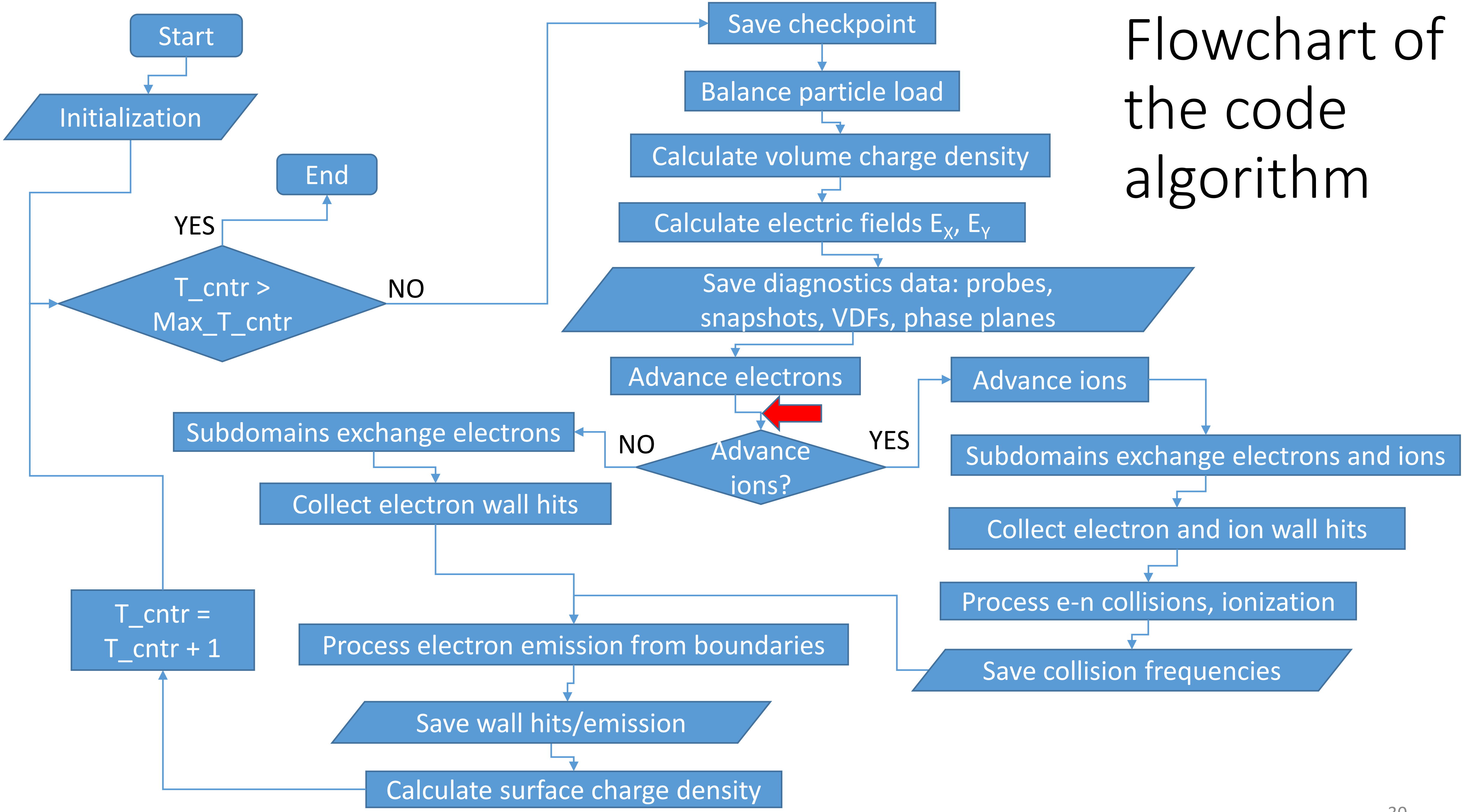
Advance electrons

```
CALL ADVANCE_ELECTRONS
```

This subroutine advances electron velocity and coordinate: $\vec{v}_e^{n-1/2} \rightarrow \vec{v}_e^{n+1/2}$, $\vec{r}_e^{n} \rightarrow \vec{r}_e^{n+1}$. Particles that cross boundary between neighbor clusters are placed into special buffers for further exchange (left/right/up/down) and removed from the main array. Particles that collide with material walls are processed and removed from the main array.

See files pic2d_ElectronDynamics.f90 and pic2d_ElectronWallCollisions.f90

Flowchart of the code algorithm


Start
Initialization
End
YES
T_cntr > Max_T_cntr
NO
Save checkpoint
Balance particle load
Calculate volume charge density
Calculate electric fields E_X, E_Y
Save diagnostics data: probes, snapshots, VDFs, phase planes
Advance electrons
Advance ions
Subdomains exchange electrons
NO
Advance ions?
YES
Subdomains exchange electrons and ions
Collect electron wall hits
Collect electron and ion wall hits
Process e-n collisions, ionization
T_cntr = T_cntr + 1
Process electron emission from boundaries
Save collision frequencies
Save wall hits/emission
Calculate surface charge density

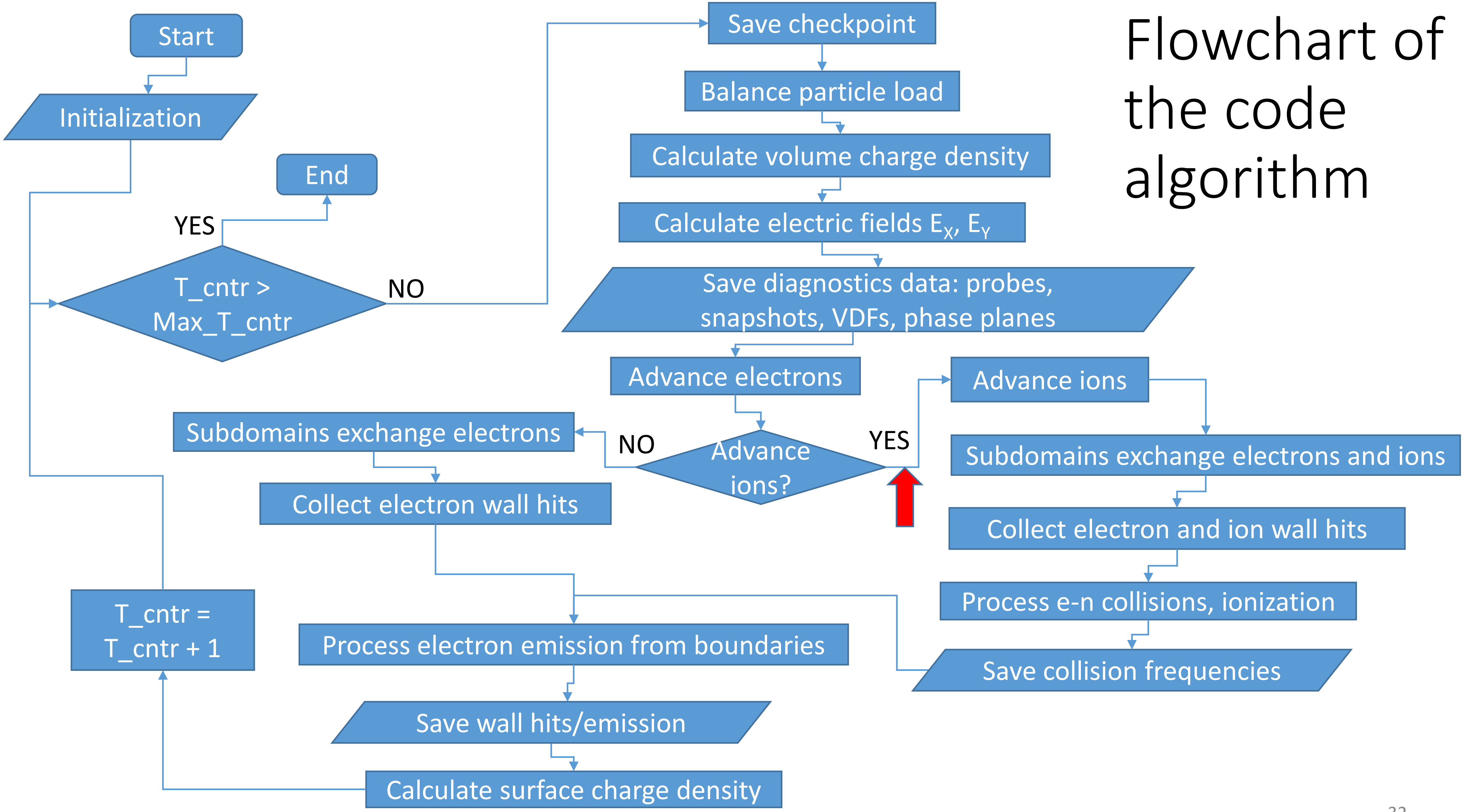
The long IF-THEN-ELSE thing related to electron subcycling

This is a counter of electron time steps. When it reaches the threshold, the ions are advanced.

```
n_sub = n_sub + 1
IF (n_sub.EQ.N_subcycles) THEN              ! N_subcycles is odd

.... Do what is required when the ions must be advanced as well [the ion branch] ...
   n_sub = 0

ELSE

... Do what is required when only the electrons are advanced [the electron branch] ...

END IF
```

Once the ion branch is done, the counter is set to zero.

Flowchart of the code algorithm


Start
Initialization
End
YES
T_cntr > Max_T_cntr
NO
Save checkpoint
Balance particle load
Calculate volume charge density
Calculate electric fields E_X, E_Y
Save diagnostics data: probes, snapshots, VDFs, phase planes
Advance electrons
Advance ions
Subdomains exchange electrons
NO
Advance ions?
YES
Subdomains exchange electrons and ions
Collect electron wall hits
Collect electron and ion wall hits
Process e-n collisions, ionization
T_cntr = T_cntr + 1
Process electron emission from boundaries
Save collision frequencies
Save wall hits/emission
Calculate surface charge density

The ion branch (1)

```
CALL ADVANCE_IONS
```

This subroutine advances ion velocity and coordinate:

$\vec{v}_i^{\,n-Nesubcycles+1/2} \rightarrow \vec{v}_i^{\,n+1/2}$,

$\vec{r}_i^{\,n-INT(Nesubcycles/2)} \rightarrow \vec{r}_i^{\,n-INT(Nesubcycles/2)+Nesubcycles}$.

The ion velocities are advanced using electric fields accumulated over N_subcycles electron time steps. Particles that cross boundary between neighbor clusters are placed into special buffers for further exchange and removed from the main array. Particles that collide with material walls are processed and removed from the main array.

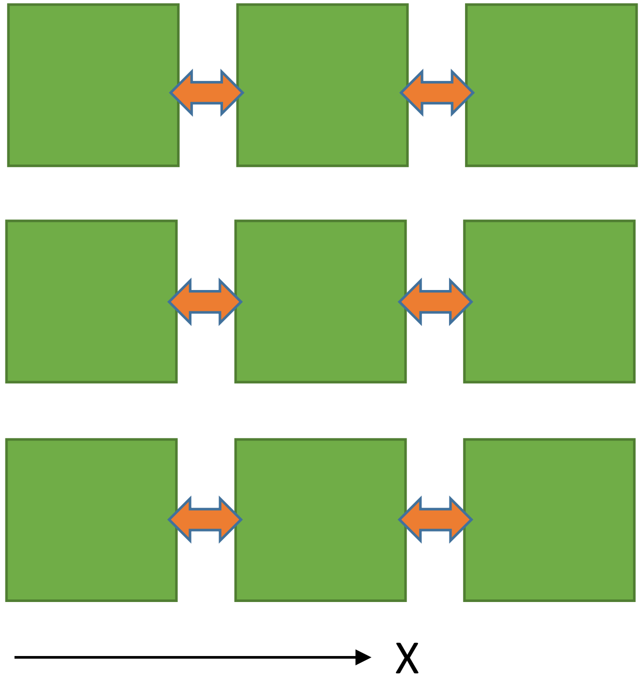
See files pic2d_IonDynamics.f90 and pic2d_IonWallCollisions.f90

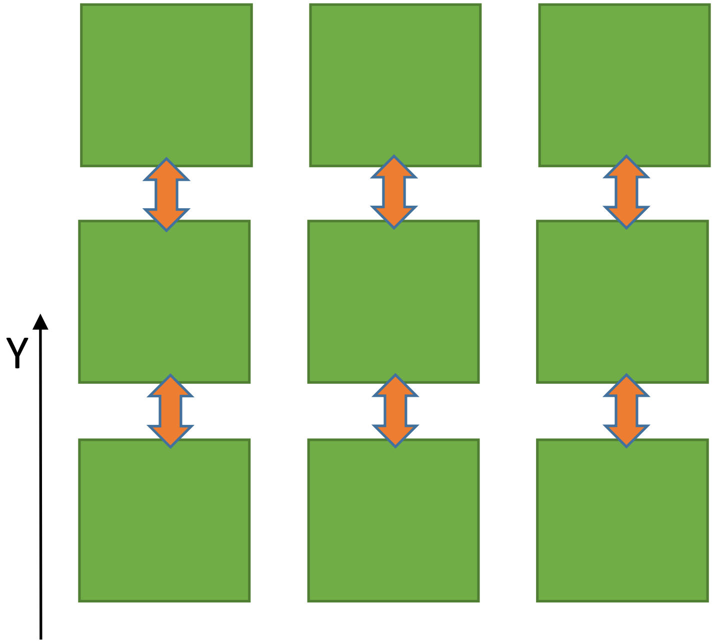
The ion branch (2a)

This subroutine performs exchange of **electrons and ions** between neighbor clusters in the X direction.
See file pic2d_ParticleExchange.f90

This subroutine performs exchange of **electrons and ions** between neighbor clusters in the Y direction.
See file pic2d_ParticleExchange.f90

```
IF (periodic_boundary_X_left.AND.periodic_boundary_X_right) THEN
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
ELSE
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
END IF
```



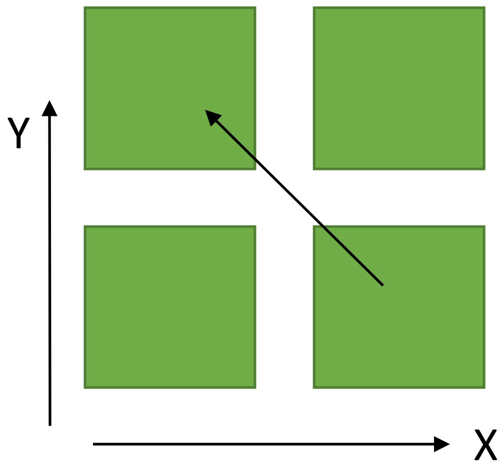


Note that in this approach there is a fixed number of non-overlapping communications between MPI processes (2 sends 2 receives) for each direction.

Here left/right is the negative/positive X-direction, below/above is the negative/positive Y-direction.

The ion branch (2b)

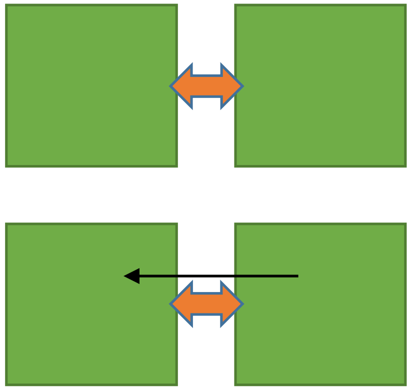
Consider a situation when a particle crosses the border of a cluster near its corner and appears in a diagonal neighbor.



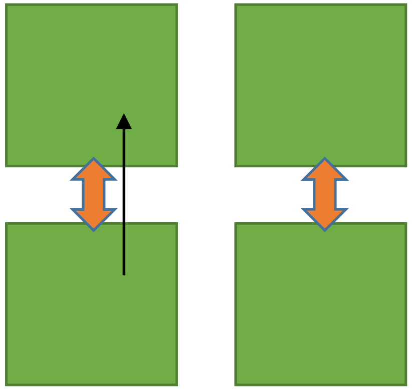
If the particle was placed in a buffer for **left-right exchange**, this particle will be transferred in **two steps**:

```
IF (periodic_boundary_X_left.AND.periodic_boundary_X_right) THEN
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
ELSE
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
END IF
```

Step 1: exchange left-right

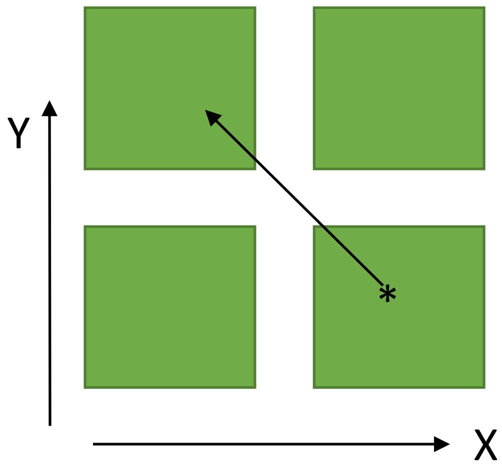


Step 2: exchange up-down



The ion branch (2c)

Consider a situation when a particle crosses the border of a cluster near its corner and appears in a diagonal neighbor.

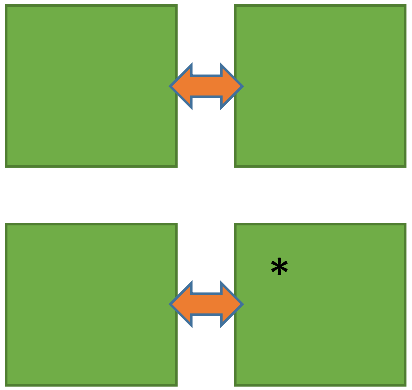


If the particle was placed in a buffer for above-below exchange, this particle will be transferred in **three** steps:

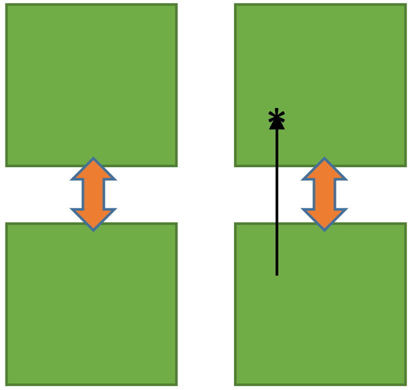
```
IF (periodic_boundary_X_left.AND.periodic_boundary_X_right) THEN
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
ELSE
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
END IF
```

This is why the additional step here is included.

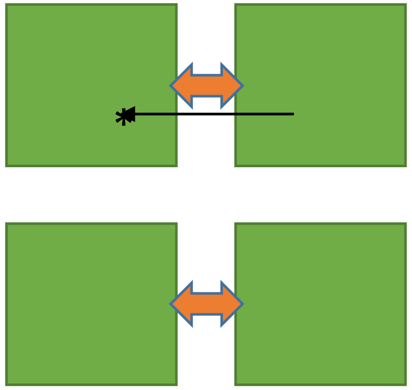
Step 1: exchange left-right, particle not included



Step 2: exchange above-below



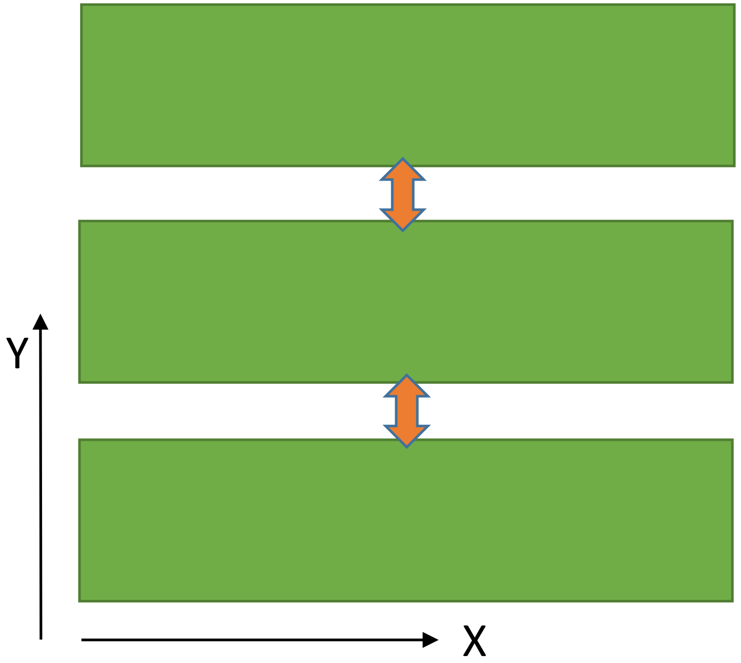
Step 3: exchange left-right



The ion branch (2d)

```
IF (periodic_boundary_X_left.AND.periodic_boundary_X_right) THEN
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
ELSE
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_ABOVE_BELOW_NEIGHBOURS
   CALL EXCHANGE_PARTICLES_WITH_LEFT_RIGHT_NEIGHBOURS
END IF
```

This branch processes a rare situation when a system periodic along the X direction is split into clusters along the Y direction only, so that particle exchange can be done in just one step.



The ion branch (3)

```
CALL PROCESS_ADDED_ELECTRONS
CALL COLLECT_PARTICLE_BOUNDARY_HITS
CALL PERFORM_ELECTRON_NEUTRAL_COLLISIONS
CALL SAVE_en_COLLISIONS
CALL PROCESS_ADDED_IONS
CALL CLEAR_ACCUMULATED_FIELDS
```

Electrons obtained after exchange with neighbor clusters are added here to the main array. This additional operation (there is one more call of this very procedure) allows to account for these electrons in the electron-neutral collision procedure.
See file pic2d_ElectronDynamics.f90

Boundary objects may be connected to multiple clusters. This subroutine collects **electron and ion** wall hit counters from all processes in a process with global MPI rank zero. This process prints a quick run-time report and later saves the accumulated counters into data files in SAVE_BOUNDARY_PARTICLE_HITS_EMISSIONS.
See file pic2d_IonWallCollisions.f90

This subroutine performs collisions between electrons and neutrals using the null-collision algorithm. The collisions are processed with the ion time step. It is done so because of the electron subcycling: if ionization collisions are included, it is not convenient to introduce new ions at intermediate time steps.
See file pic2d_enCollisionsGeneralProc.f90

The ion branch (4)

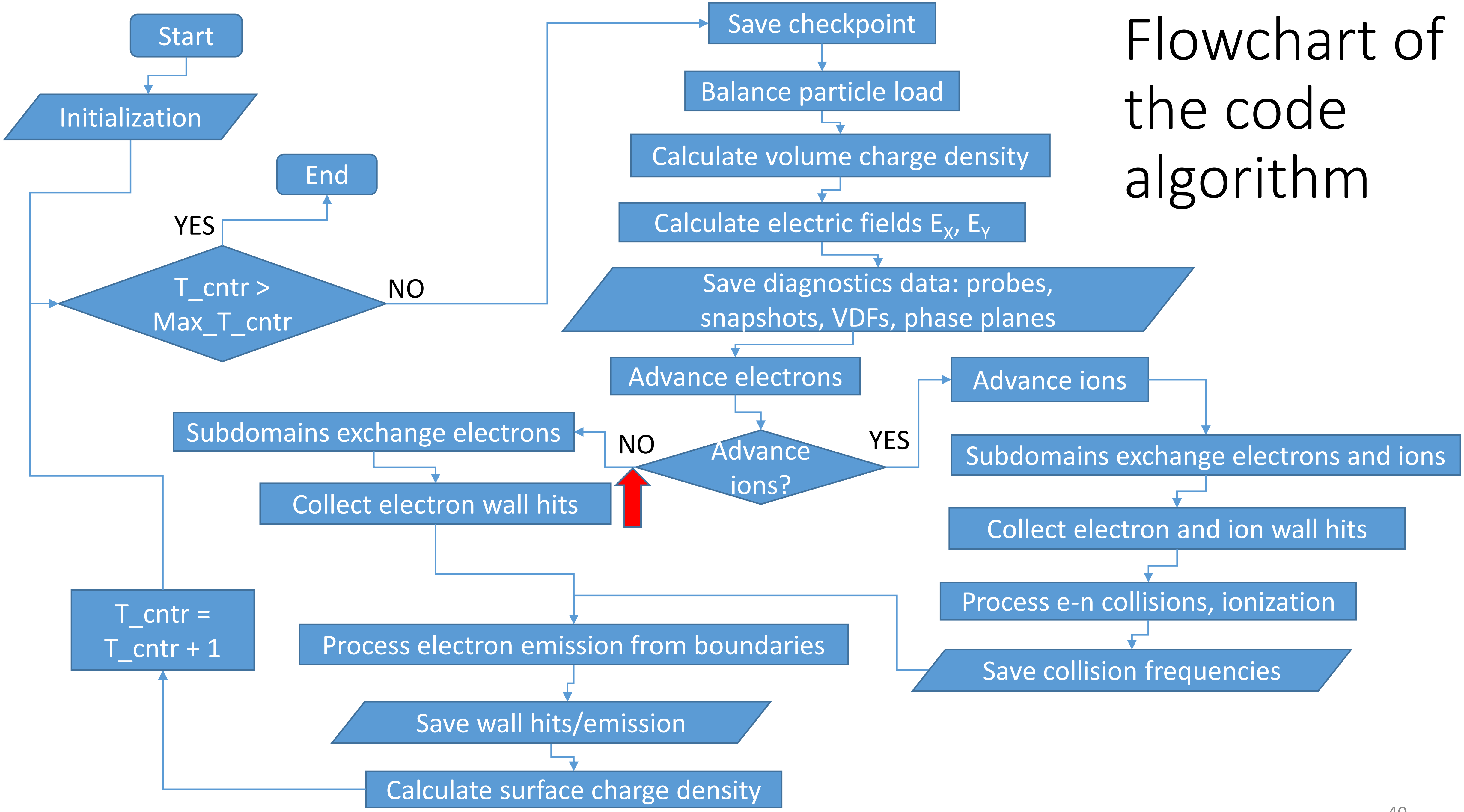
```
CALL PROCESS_ADDED_ELECTRONS
CALL COLLECT_PARTICLE_BOUNDARY_HITS
CALL PERFORM_ELECTRON_NEUTRAL_COLLISIONS
CALL SAVE_en_COLLISIONS
CALL PROCESS_ADDED_IONS
CALL CLEAR_ACCUMULATED_FIELDS
```

This subroutine collects collision event counters from all processes in the process with the global MPI rank zero. The zero rank process then saves the added counters into a file (one file history_coll_e_n_AAAAAA.dat per neutral species AAAAAA).
See file pic2d_enCollisionsGeneralProc.f90

Ions obtained after exchange with neighbor clusters and produced in ionization collisions are added here to the main array.
See file pic2d_IonDynamics.f90

Here electric field components accumulated on the grid during one ion time step are set to zero. The accumulated electric fields are used in the ion motion equations.
See file pic2d_ElectricFieldCalc_PETSc.F90

Flowchart of the code algorithm



Start
Initialization
End
YES
T_cntr > Max_T_cntr
NO
Save checkpoint
Balance particle load
Calculate volume charge density
Calculate electric fields E_X, E_Y
Save diagnostics data: probes, snapshots, VDFs, phase planes
Advance electrons
Advance ions
Subdomains exchange electrons
NO
Advance ions?
YES
Subdomains exchange electrons and ions
Collect electron wall hits
Collect electron and ion wall hits
Process e-n collisions, ionization
T_cntr = T_cntr + 1
Process electron emission from boundaries
Save collision frequencies
Save wall hits/emission
Calculate surface charge density

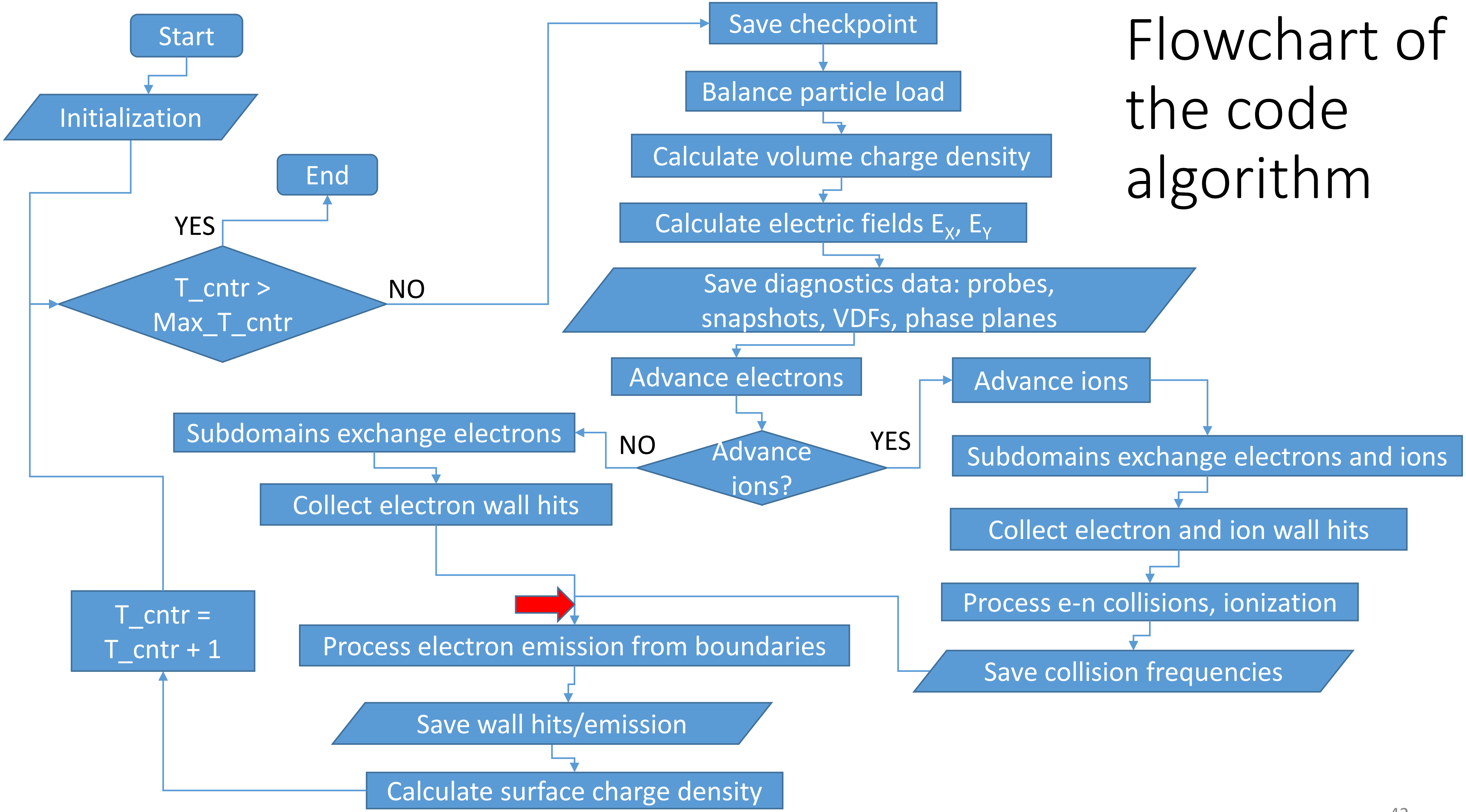
The electron branch

```
IF (periodic_boundary_X_left.AND.periodic_boundary_X_right) THEN
   CALL EXCHANGE_ELECTRONS_WITH_ABOVE_BELOW_NEIGHBOURS
ELSE
   CALL EXCHANGE_ELECTRONS_WITH_LEFT_RIGHT_NEIGHBOURS
   CALL EXCHANGE_ELECTRONS_WITH_ABOVE_BELOW_NEIGHBOURS
   CALL EXCHANGE_ELECTRONS_WITH_LEFT_RIGHT_NEIGHBOURS
END IF

CALL COLLECT_ELECTRON_BOUNDARY_HITS
```

Here the exchange of electrons between neighbor clusters takes place.
Everything is similar to the corresponding piece from the ion branch, except the ions are not processed.
See file pic2d_ElectronExchange.f90

Boundary objects may be connected to multiple clusters. This subroutine collects electron wall hit counters from all processes in a process with global MPI rank zero. This process prints a quick run-time report and later saves the accumulated counters into data files in SAVE_BOUNDARY_PARTICLE_HITS_EMISSIONS.
See file pic2d_ElectronWallCollisions.f90

Flowchart of the code algorithm


Start
Initialization
End
YES
T_cntr > Max_T_cntr
NO
Save checkpoint
Balance particle load
Calculate volume charge density
Calculate electric fields E_X, E_Y
Save diagnostics data: probes, snapshots, VDFs, phase planes
Advance electrons
Advance ions
Subdomains exchange electrons
NO
Advance ions?
YES
Subdomains exchange electrons and ions
Collect electron wall hits
Collect electron and ion wall hits
Process e-n collisions, ionization
T_cntr = T_cntr + 1
Process electron emission from boundaries
Save collision frequencies
Save wall hits/emission
Calculate surface charge density

Process electron emission from boundaries

```
CALL PERFORM_ELECTRON_EMISSION_SETUP
CALL PROCESS_ADDED_ELECTRONS
```

Emission of electrons from material surfaces is performed here. Presently it is emission with pre-defined current, like emission from a thermal cathode.

See file pic2d_Setup.f90

Electrons produced in ionization collisions and emitted from walls are added here to the main array.

See file pic2d_ElectronDynamics.f90

Save wall hits/emissions

```
CALL SAVE_BOUNDARY_PARTICLE_HITS_EMISSIONS
```

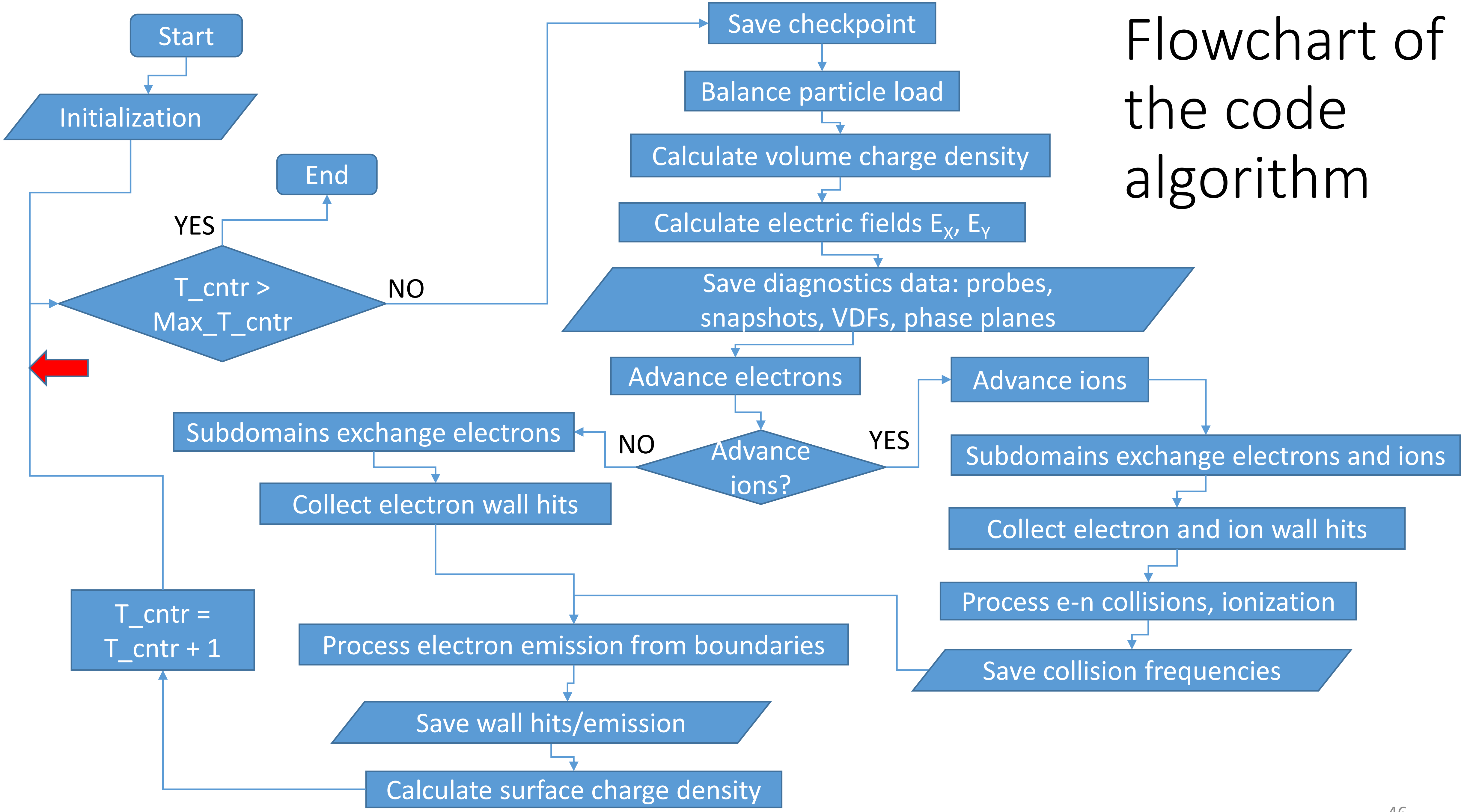
This subroutine saves counters of particles (electrons and ions) collided with and emitted (electrons only) by boundary objects into a data file (one file history_bo_NN.dat per boundary object (NN is the id number of the object)).
See file pic2d_ElectronWallCollisions.f90

Calculate surface charge density

```
CALL GATHER_SURFACE_CHARGE_DENSITY
```

This subroutine calculates surface charge density along dielectric boundary objects. Note that presently a dielectric boundary object can be included only as a semi-infinite dielectric bounding at $y=0$ and $y=L_Y$ a rectangular domain periodic along the X-direction.
See file pic2d_IonWallCollisions.f90

Flowchart of the code algorithm



Start
Initialization
End
YES
T_cntr > Max_T_cntr
NO
Save checkpoint
Balance particle load
Calculate volume charge density
Calculate electric fields E_X, E_Y
Save diagnostics data: probes, snapshots, VDFs, phase planes
Advance electrons
Advance ions?
NO
YES
Advance ions
Subdomains exchange electrons
Subdomains exchange electrons and ions
Collect electron wall hits
Collect electron and ion wall hits
Process e-n collisions, ionization
Save collision frequencies
Process electron emission from boundaries
Save wall hits/emission
Calculate surface charge density
T_cntr = T_cntr + 1