Documentation

ContaminationFlow on Linux and Windows

Hoai My Van, Rudolf Schönmann

Contents

1. General Structure **1**

2. ContaminationFlow Linux **2**
2.1. Call of Application from Command line . . . 3
2.2. Communication . . . 4
2.3. Usage of *boost* Library . . . 5
2.4. New Quantities . . . 6
2.5. Iterative Algorithm . . . 7
2.5.1. Initialization of simulation . . . 7
2.5.2. Simulation on subprocesses . . . 8
2.5.3. Update main process . . . 10
2.6. Summary . . . 12

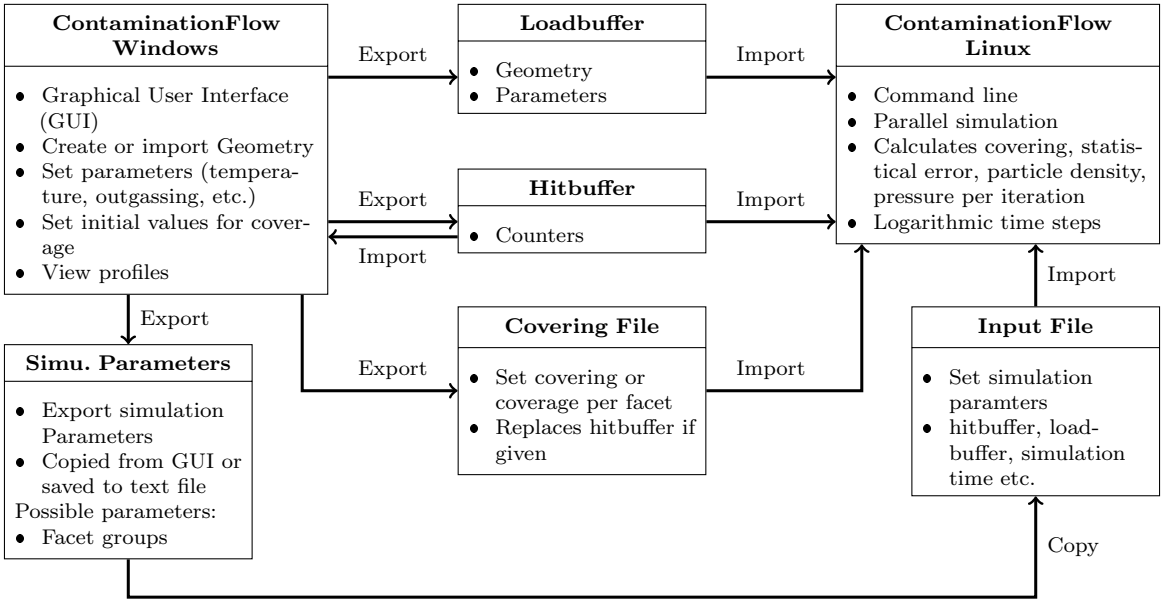
3. ContaminationFlow Windows **14**
3.1. Graphical User Interface . . . 14
3.2. Communication . . . 15
3.3. New Quantities . . . 15

A. Formulas for new Quantities **17**

B. Datatypes **19**
B.1. Boost . . . 19
B.2. Class Members . . . 19
B.3. Functions . . . 19

C. Overview of new Classes and Functions **20**
C.1. New Classes . . . 20
C.2. New Functions . . . 23
C.2.1. molflowlinux_main.cpp . . . 23
C.2.2. SimulationLinux.cpp . . . 24
C.2.3. Iteration.cpp . . . 24
C.2.4. Buffer.cpp . . . 25
C.2.5. Calculations in SimulationCalc.cpp etc. . . . 25
C.2.6. UpdateSubProcess.cpp . . . 26
C.2.7. UpdateMainProcess.cpp . . . 26

1. General Structure



General structure for ContaminationFlow simulation

- Code adapted from Molflow
- ContaminationFlow Windows primary used to create Geometry and to view simulation results through the GUI
- ContaminationFlow Linux primary used for simulation and calculation of counters, profiles, etc.
- Loadbuffer contains information of geometry
- Hitbuffer contains information such as hit counters, profiles, etc.
- Import and export of buffer files for communication between ContaminationFlow Windows and ContaminationFlow Linux
- Optional: export/import of covering text file that replaces covering in hitbuffer
- Import input file to define simulation parameters
- Some input file parameters can be exported/copied from ContaminationFlow Windows
- Export of SimulationHistory for ContaminationFlow Linux

2. ContaminationFlow Linux

- Parallel simulation on several sub processes
- Processing and control of data in main process
- Update and accumulation of hit counters and other information such as profiles
- `SimulationHistory`, simulation parameters, console output exported to results folder

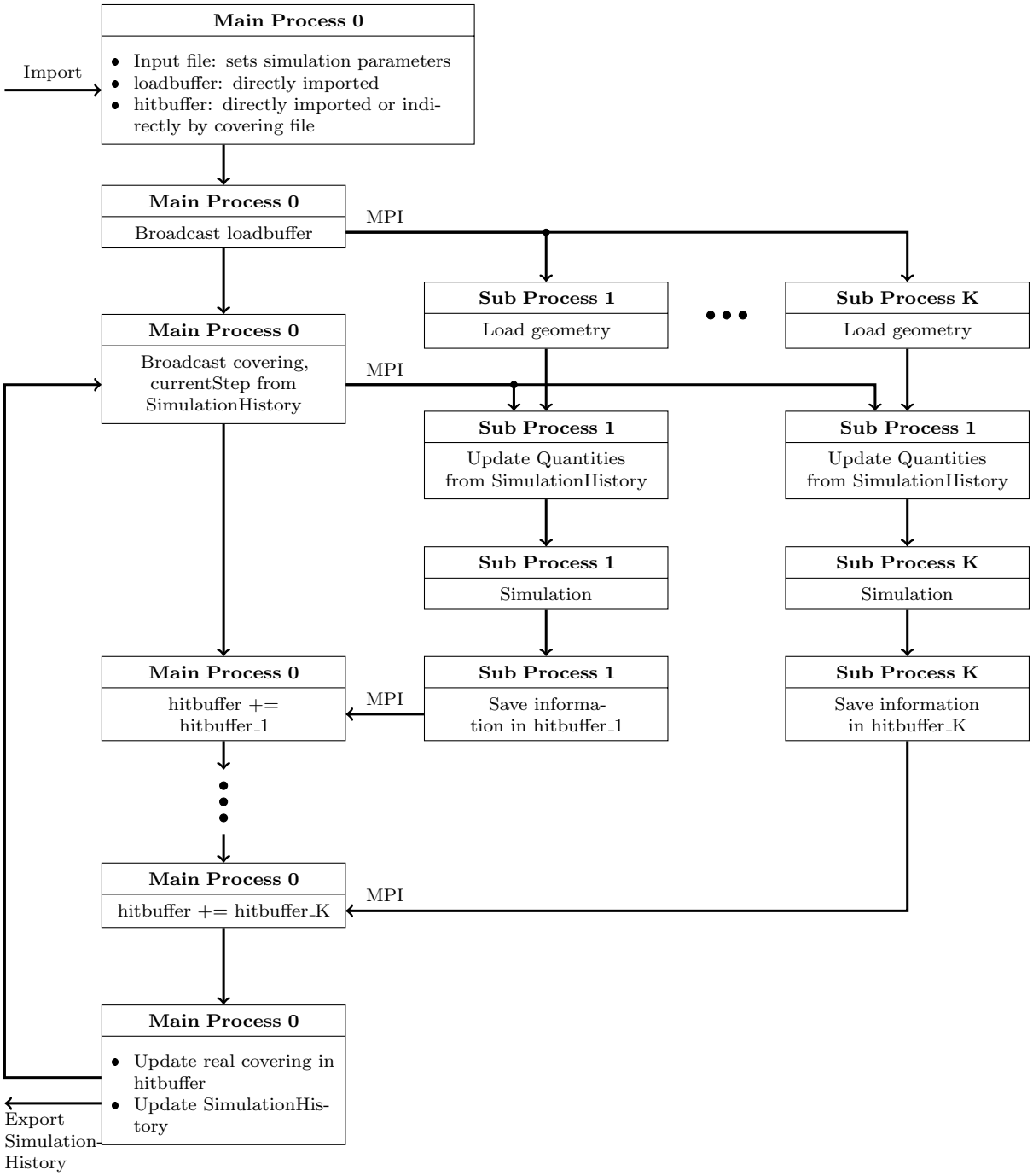


Figure 2.1.: Processing of data in main and sub processes

2.1. Call of Application from Command line

New class ProblemDef

- Defines parameters used for simulation
- Possible adaptation of default paramaters through input file
- Creates result folder for simulation if desired:
  - Final covering, error, pressure, particle density as text files
  - Input file and console output as text files

Application with custom parameters using input file

Requirements: Input and buffer files readable, N $\geq$ 2, zero moments in loadbuffer

Call of ContaminationFlow Linux application in the command line:

```
$ module load mpi
$ mpirun -n N --options  MolflowLinux inputfile save
```

with the following MPI `--options`:

- (No option): use processor cores (recommended if possible)
- `--use-hwthread-cpus`: use hardware threads instead processor cores (recommended if N > number cores)
- `--oversubscribe`: ignore available slots, for any number of MPI processes

and command line parameters:

- `N`: desired number of worker processes; simulation on K=N-1 worker processes
- `MolflowLinux`: path to application, e.g. `~/MolflowLinux/Debug/MolflowLinux`
- `inputfile`: path to file that defines simulation parameters
- `save`: determines whether result directory is created (1: true, 0: false); default: `1`

and the input file defining the following parameters:

- *loadbufferPath*: Path to loadbuffer file, contains geometry, e.g. `~/loadbuffer`
- *hitbufferPath*: Path to hitbuffer file, contains counters, etc., e.g. `~/hitbuffer`
- *coveringPath*: Path to covering file, contains either covering or coverage per facet, file can be exported from ContaminationFlow Windows, e.g. `~/covering.txt`; default: `""`
- *simulationTime*: Simulation time resolution. After each *simulationTime* the error and particle number will be checked; default: `10.0`
- *unit*: Simulation time unit; default: `s`
- *maxTime*: Max. simulated time; default: `10.0`
- *maxUnit*: Max. simulated time unit; default: `y`
- *iterationNumber*: Number of iterations; default: `43200`
- *usePCMethod*: Use predictor-corrector method or not; default: `0`
- *particleDia*: Diameter of particles; default: `2.76E-10`
- E_{de}: Binding energy of a particle on pure substrate; default: `1.6E-19`
- H_{vap}: Vaporization enthalpy of a particle if multilayer contamination; default: `0.8E-19`
- *sticking*: Constant sticking coefficient for all facets; default: `0`
- *errorMode*: Type of error monitored, "covering" or "event"; default: `covering`
- *targetParticles*: Min. number of desorbed particles per iteration; default: `1000`

- *targetError* : Avg. statistical uncertainty (error) to be achieved for each iteration, calculated as the avg. (weighted with the facets area) of the normalized standard deviation of events per facet; default: `0.001`
- *noupdateError* : Error value above which the covering will not be updated; default: `0.1`
- *hitRatioLimit* : Ratio (facet/all), e.g., number events or covering change, at which facet is ignored for error calculation; default: `0`
- *t_min* : Min. time for step size; default: `1E-4`
- *t_max* : Max. time for step size; default: `max`
- *maxTimePerIt* : Max. simulation time [s] per iteration; default: `max`
- *coveringMinThresh* : Min. covering (through multiplication); default: `1000000`
- *histSize* : Size of history lists; default: `max`
- *vipFacets* : Very important facets: facets with own target error. Alternating sequence of facet numbers and respective target errors separated via blanks; default: `[]`
- *outgassingTimeWindow* : Duration [s] of outgassing impulse; default: `0.0`
- *counterWindowPercent* : Percentage of step size (posterior) at which velocity counters are increased; default: `0.1`
- *desWindowPercent* : Percentage of step size (anterior) at which desorption occurs; default: `1.0`
- *rollingWindowSize* : Number of iterations for avg. statistics; default: `10`
- *convergenceTarget* : Target for avg. statistics to indicate convergence; default: `1`
- *stopConverged* : Stop simulation if avg. statistics indicate convergence; default: `1`
- *convergenceTime* : Min. simulated time to stop simulation if converged; default: `0`
- *facetGroups* : Indices of facets belonging to a group, groups divided by -; default: `[]`
- *focusGroup* : Indices of facet groups to be monitored; default: `[]`
- *doFocusGroupOnly* : Determines if only focusGroup facets are monitored facets; default: `1`

Optional covering file to replace hitbuffer. Hitbuffer will not be imported if covering file is given. There are two options (that can both be exported from ContaminationFlow Windows):

- set covering: `covering` followed by covering values per facet, seperated via blanks
- set coverage: `coverage` followed by coverage values per facet, seperated via blanks

Terminology

- Simulation time: desired computation time until check if target is reached for iteration
- Simulated time: physical time in the simulated system, e.g. flight time of particle
- Maximum simulated time: desired total simulated time
- Step size: desired simulated time per particle for iteration

2.2. Communication

Import and export of buffer files

- New Databuff struct that replaces Dataport struct from MolFlow Windows

```
typedef unsigned char BYTE;
typedef struct {
  signed int size;
  BYTE *buff;
} Databuff;
```

- New function `initBuffSize(·)` to initialize buffer size
- New functions `importBuff(·)` / `exportBuff(·)` to import buffer file/export Databuff struct
- New functions `checkReadable(·)` / `checkWriteable(·)` to check if file is readable/writeable

Communication between worker processes via MPI

- Main process 0 sends Databuff structs containing loadbuffer/hitbuffer and required SimulationHistory values to sub processes using `MPI_Bcast(·)`
- Sub processes send updated Databuff struct containing hitbuffer and required SimulationHistory values to main process 0 using `MPI_Send(·)` and `MPI_Recv(·)`

2.3. Usage of *boost* Library

Multiprecision

- Increase precision for variables if required (float128, uint_128t)
- Avoid overflow for integer and underflow for floating point numbers

2.4. New Quantities

New counter `covering`

- Number of carbon equivalent particles on facet
- Increases with adsorption, decreases with desorption
- Extracted from new FacetHitBuffer counter in `getCovering(·)` (`SimulationCalc.cpp`)

Coverage

- Number of monolayers of adsorbed particles
- Calculated from covering, particle diameter (previously gas mass) and facet area
- Coverage computed in `calcCoverage(·)` (`SimulationCalc.cpp`)

Sticking factor

- Ratio adsorbed particles to impinging particles
- Set to value specified in input file in `calcSticking(·)` (`SimulationCalc.cpp`)

Binding energy

- Either E_{de} or H_{vap}, depending on the how many layers of particles are adsorbed.
- If coverage is smaller than a monolayer, it will be decided at random.
- Used in `StartFromSource(·)` & `PerformBounce(·)` (`SimulationMC.cpp`) and `calcDesorption(·)` & `calcStartTime(·)` (`SimulationCalc.cpp`)

Desorption

- Number of particles desorbing
- Calculated from binding energy, covering, temperature and step size
- Desorption computed in `calcDesorption(·)` (`SimulationCalc.cpp`)

Outgassing

- Number of particles from outgassing
- Calculated from facet outgassing, temperature, and outgassing time
- Outgassing computed in `CalcTotalOutgassingWorker()` (`Worker.cpp`)

$K_{\text{real/virtual}}$

- Number of real particles represented by test particles
- Calculated from desorption & outgassing and number of desorbed molecules
- $K_{\text{real/virtual}}$ computed in `GetMoleculesPerTP(·)` (`SimulationCalc.cpp`)

Statistical error

- Event error: calculated from hits and desorbed particles (of facet and total)
- Covering error: calculated from adsorbed and desorbed particles (of facet and total)
- Used to determine significance of simulation results of iteration
- Error calculated in `UpdateErrorList(·)` for facet error & `UpdateErrorAll(·)` for average error weighted with area (`UpdateSubProcess.cpp`)

Step size

- Minimum time between adsorption and desorption
- Step size computed in `getStepSize()` (`UpdateMainProcess.cpp`)

Particle density

- Calculated from sum over reciprocal of orthogonal velocity, facet area and $K_{\text{real/virtual}}$
- Particle density computed in `calcParticleDensity(·)` (`SimulationCalc.cpp`)

Pressure

- Calculated from sum over orthogonal velocity, facet area, gas mass and $K_{\text{real/virtual}}$
- Pressure computed in `calcPressure(·)` (`SimulationCalc.cpp`)

Start time

- Determines time of desorption/outgassing for particle based on the distribution
- Desorption rate: exponential distribution for whole iteration
- Outgassing: uniform distribution of limited time for whole simulation
- Start time computed in `calcStartTime(·)` (`SimulationCalc.cpp`)

2.5. Iterative Algorithm

2.5.1. Initialization of simulation

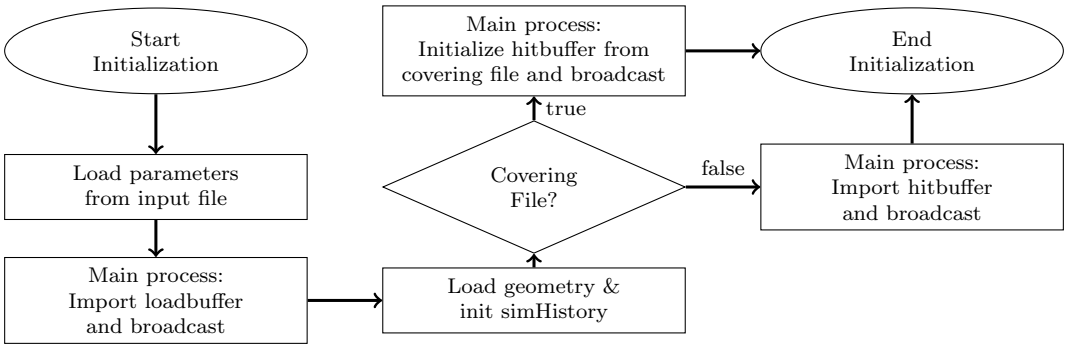


Figure 2.2.: Overview: Initialize simulation

New class to store Simulation History

- HistoryList and SimulationHistory class in `SimulationLinux.h` and `SimulationLinux.cpp` file

```
template <typename T> class HistoryList {
public:
 HistoryList();
 pair<vector<double>,vector<vector<T>>> historyList;//(vec(time), vec(facets))
 vector<pair<float128,float128>> statisticsList;//vec(mean, std)
 vector<T> currentList;//facets
 vector<T> predictList;//used for predictor-corrector method };
```

```
class SimulationHistory {
public:
 SimulationHistory();
 SimulationHistory(Databuff *hitbuffer);

 HistoryList<uint_128t> coveringList;//covering
 HistoryList<double> hitList;//MC hits
 HistoryList<llong> desorbedList;//Number of desorbed particles
 HistoryList<llong> adsorbedList;//Number of adsorbed particles
 HistoryList<double> errorList_event;//error event:  hits & desorbed particles
 HistoryList<double> errorList_covering;//error covering:  desorbed & adsorbed
 HistoryList<double> particleDensityList;
 HistoryList<double> pressureList;

 double lastTime;
 int currentStep;
 int pcStep;//Loop variable for predictor-corrector method
};
```

- Updated at the end of each iteration in `UpdateParticleDensityAndPressure(·)`, `UpdateCovering(·)`, `UpdateErrorMain(·)` (`UpdateMainProcess.cpp`)
- Recorded quantities: covering, error (event and covering), particle density and pressure for each facet and iteration, total hits and desorbed particles for each facet
- lastTime: simulated time (accumulated time steps) instead of computation time

2.5.2. Simulation on subprocesses

Calculate step size

- Calculation from `simHistory→currentStep` in `getStepSize()` (`UpdateMainProcess.cpp`)

Calculate covering threshold

- Set lower threshold for covering for each facet to prevent covering getting negative
- Stop simulation once threshold is reached
- Threshold set in `setCoveringThreshold(·)` (`Iteration.cpp`)

Multiply small covering

- Multiply covering so that smallest covering $\geq$ `ProblemDef::coveringThreshMin`
- Multiply covering threshold with same factor
- Calculation in `checkSmallCovering(·)` (`SimulationLinux.cpp`)

Calculate desorption and outgassing

- Desorption from covering in `UpdateDesorption(·)` (`UpdateSubProcess.cpp`)
- Outgassing in `CalcTotalOutgassingWorker()` (`Worker.cpp`)

Create particle and calculate start time

- Facet randomly selected based on total desorption and outgassing
- Desorption or outgassing randomly selected based on ratio on facet
- Start time randomly generated based on distribution of desorption or outgassing

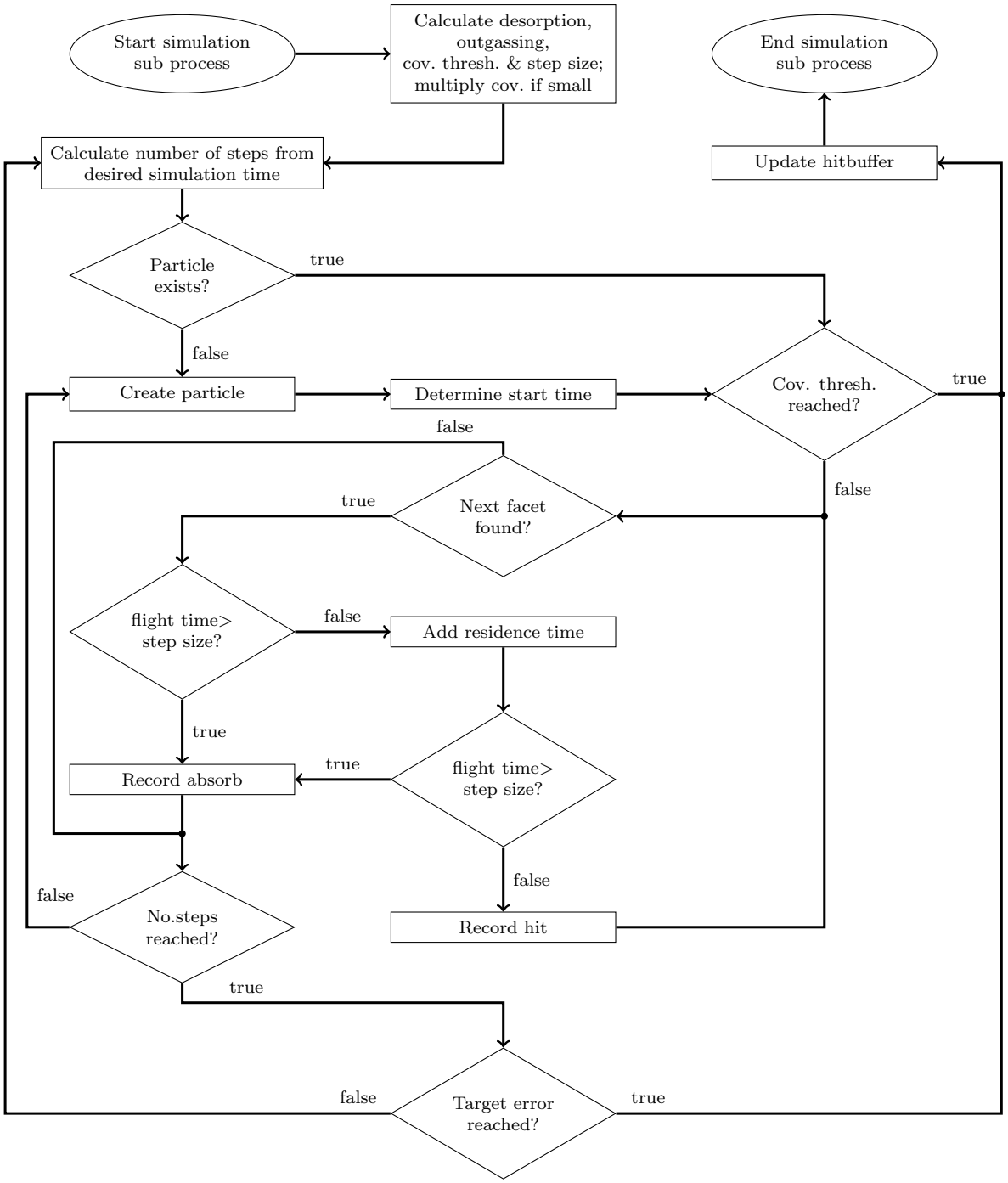


Figure 2.3.: Overview: simulation on sub processes

- Calculation in `StartFromSource(·)` (`SimulationMC.cpp`)

Calculate residence time

- Sojourn time randomly calculated from binding energy, facet temperature and sojourn frequency in `PerformBounce(·)` (`SimulationMC.cpp`)

Increase facet counters in case of desorb, absorb or hit

- Increase hit, desorb or absorb counter according to event

- Increase velocity counters only if event within `p→counterWindowPercent`
- Facet counters increased in `IncreaseFacetCounter(·)` (`SimulationMC.cpp`)

Target error reached?

- Calculate statistical error in `UpdateError()` from `UpdateSubProcess.cpp` file
- Avg. error calculated from summing facet error weighted with facet area
- Error to check can be either covering or event error (currently covering)
  - Check if vip facets reached their own target error
  - Check if avg. error reached target error
- Facets with error=inf are not considered
  - Facets that reached `ProblemDef::hitRatioLimit`
  - Facets with no events or covering change
  - If vip facet: own target error automatically reached
  - If normal facet: facet error and area not used for calculation
- Check in `checkErrorSub(·)` (`UpdateSubProcess.cpp`)

2.5.3. Update main process

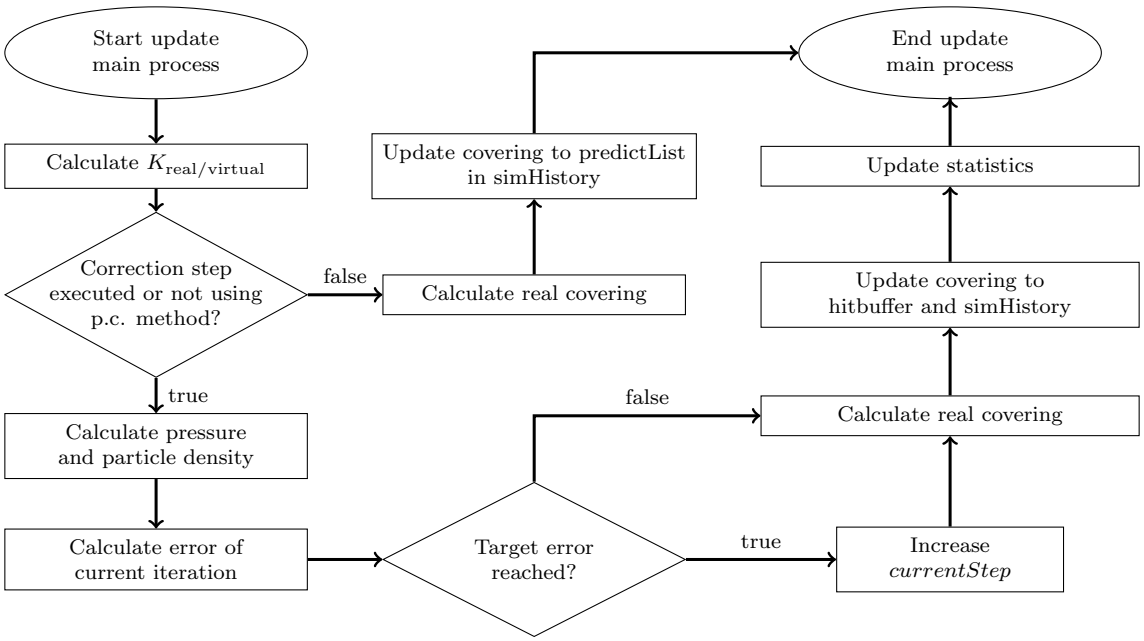


Figure 2.4.: Overview: update of covering in hitbuffer and simHistory

Before summation of subprocesses

- Calculate step size in `getStepSize()` (`UpdateMainProcess.cpp`)
- Multiply covering in `hitbuffer` of main process analogous to sub processes in `checkSmallCovering(·)` (`SimulationLinux.cpp`)

Predictor-corrector method

- Analogous to Heun's method for solving ordinary differential equations numerically

- In the prediction step, the iteration for the current time step is executed and the resulting covering values are saved in `simHistory→coveringList.predictList`
- In the correction step, the iteration for the current time step is repeated. In `PerformBounce(·)` (`SimulationMC.cpp`), reading the covering values from `currentList` if `sHandle→currentParticle.flightTime <= 1/2*simHistory→stepSize` , otherwise reading them from `predictList`
- At the end of the correction step, the main process is updated as usual and continuing with the next iteration
- Used only, if `p->usePCMethod` is set to anything other than `0` explicitly via the input file
- If `usePCMethod` is set to `0` : not using the p.c. method; `1` : using the first approach for the p.c. method; `2` : using the second approach for the p.c. method
- The second approach for the implementation of the p.c. method works better than the first one

Calculate pressure and particle density

- Calculation in `UpdateParticleDensityAndPressure(·)` `(UpdateMainProcess.cpp)`
- Values per facet saved in `simHistory→pressureList` / `simHistory→particleDensityList`

Calculate error

- Calculation analogous to sub processes in `UpdateErrorMain(·)` (`UpdateMainProcess.cpp`)
- Save error per facet in `simHistory→errorList_event` / `simHistory→errorList_covering`
  ⇒ Increase `simHistory→currentStep` if target errors reached

Calculate & update covering

- $K_{\text{real/virtual}}$ computed in `GetMoleculesPerTP(·)` (`Simulationcalc.cpp`)
- Divide covering in `hitbuffer` if previously multiplied
- Use $K_{\text{real/virtual}}$ to calculate new covering
- Save new covering in `currentList` in `simHistory→coveringList` and `hitbuffer`
- Calculation in `UpdateCovering(·)` from `UpdateMainProcess.cpp` file
- Update buffers in `UpdateCoveringPhys(·)` (`UpdateMainProcess.cpp`)

Calculate & update statistics

- Calculate mean and std of covering over last `p→rollingWindowSize` iterations
- Update statistics in `HistoryList::updateStatistics(·)` (`SimulationLinux.h`)
- End simulation if `p→convergenceTarget` is reached by avg. statistics weighted with area which is calculated in `HistoryList::getAverageStatistics(·)` (`SimulationLinux.h`)

2.6. Summary

General Pipeline

- Initialize MPI, `ProblemDef p` and `SimulationHistory simHistory`
- Check if all simulation parameters are valid
- Load geometry into `Simulation sHandle` and check if values are valid using `loadAndCheckSHandle()`
  - Load geometry using `LoadSimulation`
  - Hitbuffer: import using `importBuff(·)` or initialize size using `initBuffSize(·)`
  - Check for correct hitbuffer size
  - Check for zero moments
  - Check for no two-sided facets with opacity
  - Check for valid covering file and import covering values
- Iteration until `p→maxTimeS` or `p→convergenceTarget` (`p→convergenceTime` , `p→stopConverged`) is reached. If `p→usePCMethod` is not set to `0` , repeat same iteration twice:
  - Reset hitbuffer counters using `initbufftozero(·)`
  - Broadcast `simHistory→coveringList` using `MPI_Bcast(·)`
  - Set covering threshold `covthresh` using `setCoveringThreshold(·)`
  - Update relevant simulation values using `simHistory→updateStepSize(·)` , `UpdateSticking(·)` , `CalcTotalOutgassingWorker(·)` , `UpdateDesorption(·)` . If using p.c. method: only in prediction step.
  - Multiply covering and `covthresh` with `simHistory→smallCoveringFactor` if covering is small
  - Simulation in sub processes:
    - Simulate until `targetParticles` and `targetError` or `covthresh` reached
    - Update hitbuffers of sub processes from `sHandle` using `UpdateMCSubHits(·)` (`UpdateSubProcess.cpp`)
  - Update main process:
    - Send hitbuffer to main process using `MPI_Send(·)` and `MPI_Recv(·)`
    - Update of hitbuffer in `UpdateMCMainHits(·)` (`UpdateMainProcess.cpp`)
  - If using p.c. method and in prediction step:
    - Calculate real covering in main process using $K_{\text{real/virtual}}$ in `UpdateCovering(·)` (`UpdateMainProcess.cpp`), save in `simHistory` : `predictList`
  - If not using p.c. method or otherwise in correction step:
    - Update pressure and particle density using `UpdateParticleDensityAndPressure(·)` `(UpdateMainProcess.cpp)` , save in `simHistory`
    - Update error of iteration using `UpdateErrorMain(·)` (`UpdateMainProcess.cpp`), save in `simHistory`
    - Calculate real covering in main process using $K_{\text{real/virtual}}$ in `UpdateCovering(·)` (`UpdateMainProcess.cpp`), save in `simHistory` : `historyList` and `currentList`
    - Update real covering in hitbuffer of main process in `UpdateCoveringphys(·)` (`UpdateMainProcess.cpp`)
    - Update statistics using `simHistory→coveringList.updateStatistics(·)`
- Export final results (`simHistory` lists) to results folder
- Close MPI

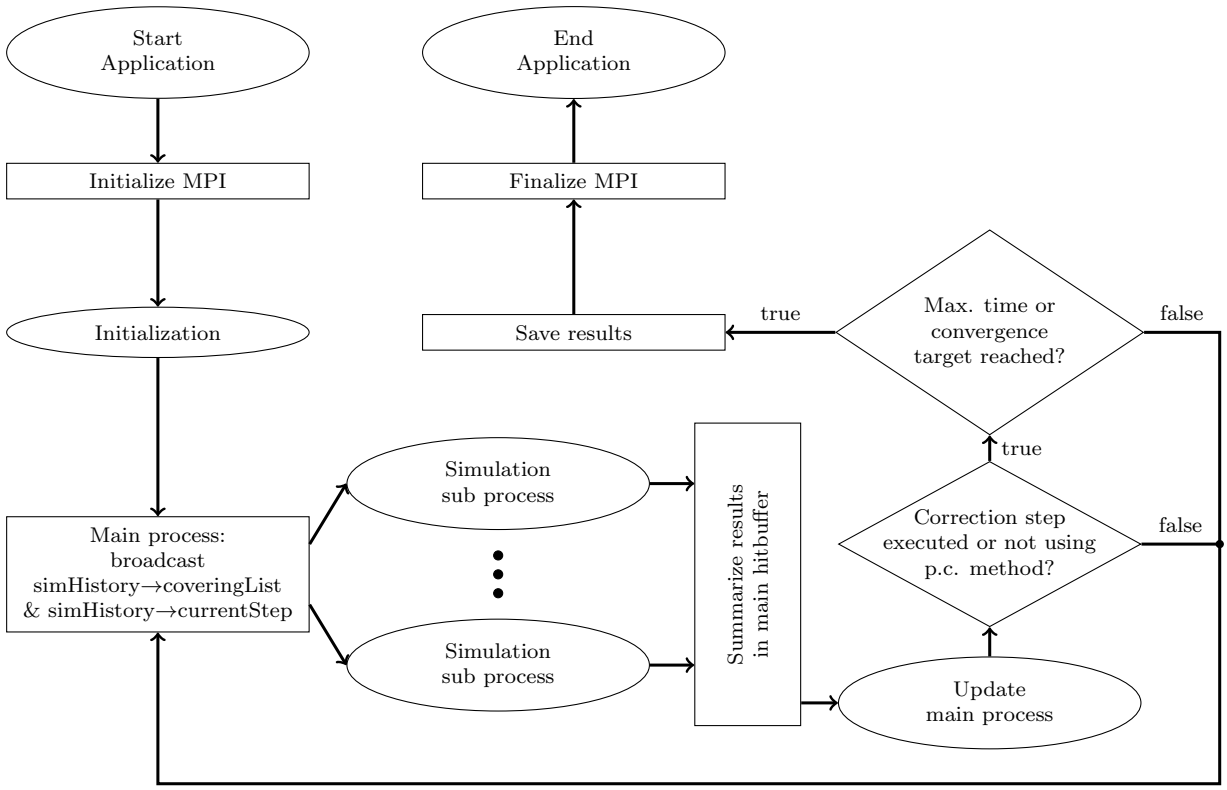
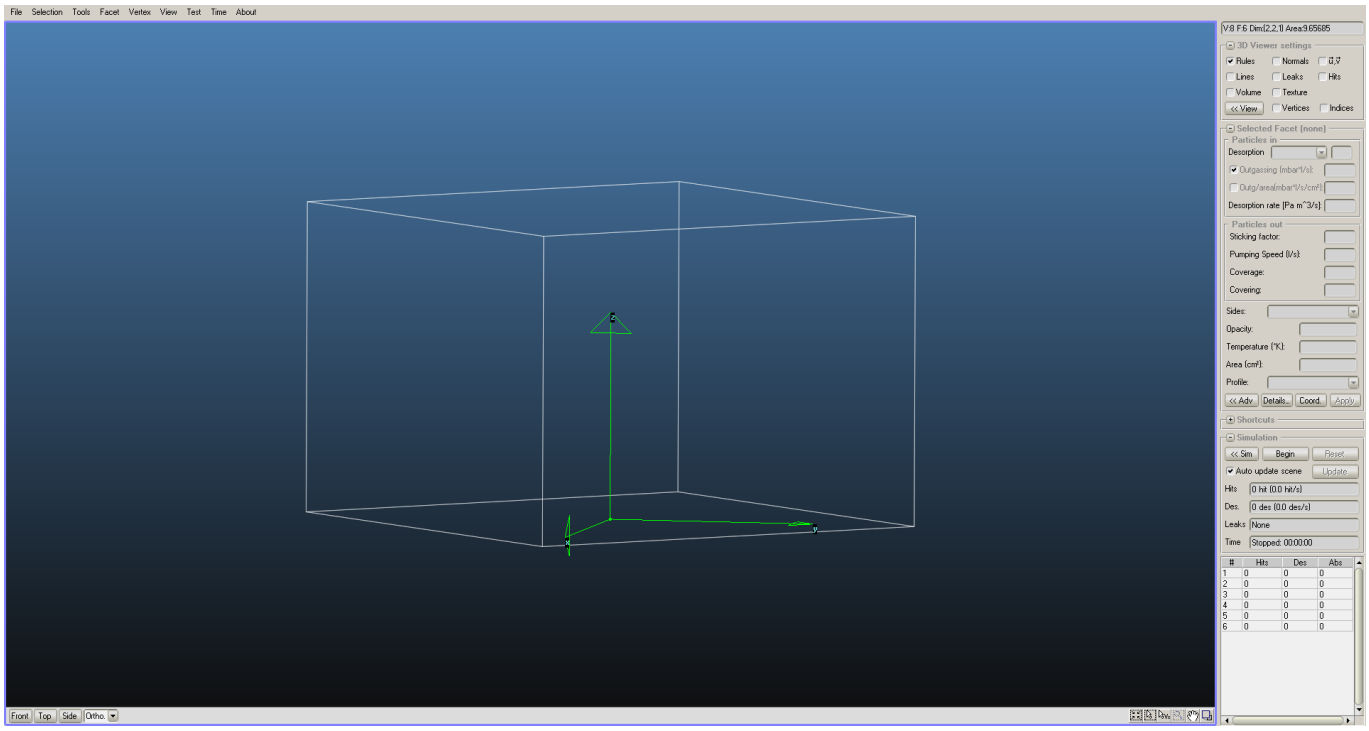


Figure 2.5.: Overview: ContaminationFlow application

3. ContaminationFlow Windows

- Expansion of original Molflow for contamination
- Create Geometry and set parameters such as initial coverage and temperature
- Import and export of buffer files
- Export of facet groups and covering file

3.1. Graphical User Interface



New GUI elements

- "Particles out" renamed to Contamination level
  - Text field for covering
  - Text field for coverage
- New facet properties
  - Effective surface factor
  - Facet depth and facet volume
  - Diffusion coefficient
  - Concentration and gas mass
- New menu options
  - File: Export buffer, Import buffer, Export Cov.
  - Selection: Export Selections

3.2. Communication

Import and export of buffer files via GUI

- New Databuff struct

```
typedef unsigned char BYTE;
typedef struct
  signed int size;
  BYTE *buff;
Databuff;
```

- New functions `importBuff(·)` and `exportBuff(·)` for import and export of buffer files/Databuff
- New options in `File` menu: `Export buffer` and `Import buffer`

Export of Facet Groups

- New functions to output (text file or text field line) correct formating of facet groups for input file for ContaminationFlow Linux
- New options in `Selection` menu: `Export Selections`

Export of Covering/Coverage File

- Two output options: covering or coverage per facet
- New functions to output (text file or text field line) correct formating of covering/coverage file for ContaminationFlow Linux
- New options in `File` menu: `Export Cov.`

3.3. New Quantities

New counter `covering`

- Added covering counter to hitbuffer
- Added covering to GUI, can be defined through textfield
- Covering increased at adsorb

New facet property `effetiveSurfaceFactor`

- Defines increase of facet area due to texture

New facet property `facetDepth`

- Defines depth of facet

New facet property `diffusionCoefficient`

- Defines diffusion coefficient

New facet property `concentration`

- Defines concentration = mass of particles in volume

Removal of irrelevant quantities

- Sticking factor and pumping speed removed from GUI
- `calcSticking()` and `calcFlow()` in `Molflow.cpp` file not used anymore
- Flow not needed for iterative Algorithm

A. Formulas for new Quantities

Constants

$$\begin{aligned} k_b &= 1.381 \times 10^{-23}\, JK^{-1} \\ h &= 6.626 \times 10^{-34}\, Js \\ N_A &= 6.022 \times 10^{23}\, mol^{-1} \end{aligned} \tag{A.1}$$

Variables

$$T = \text{Facet temperature} \tag{A.2}$$

Number of carbon equivalent particles of one monolayer

$$N_{mono} = \frac{\text{Area of Facet } [\text{m}^2]}{\text{ProblemDef::particleDia}^2\ [\text{m}^2]} \tag{A.3}$$

Covering θ^*

$$\theta^* = N_{\text{particles on facet}} \tag{A.4}$$

Coverage θ

$$\theta = \theta^* / N_{mono} \tag{A.5}$$

Sticking factor $sticking$

$$sticking = \begin{cases} \text{ProblemDef::sticking}, & \text{if } \theta^* > 0 \\ 0, \text{otherwise} \end{cases} \tag{A.6}$$

Binding energy E

$$E = \begin{cases} E_{de}, & \text{if particle binds with substrate} \\ H_{vap}, & \text{if particle binds with adsorbate} \end{cases} \tag{A.7}$$

Residence time τ

$$\begin{aligned} A &= \exp\big(-E/(k_b T)\big),\ \tau_0 = \frac{h}{k_b\, T} \\ \tau &= \frac{-\ln(rnd) \cdot \tau_0}{A} \end{aligned} \tag{A.8}$$

Step size t_{step}

$$\begin{aligned} t_{min} &= \text{ProblemDef::t\_min} \\ t_i &= t_{min} \cdot \exp\big(\text{i} \cdot \ln(\text{ProblemDef::maxTimeS}/T_{min})/\text{ProblemDef::iterationNumber})\big) \\ t_{step} &= \min\big(t_{currentStep+1} - t_{currentStep}, \text{ProblemDef::t\_max}\big) \end{aligned} \tag{A.9}$$

Desorption *des*

$$\tau_0 = \frac{h}{k_b T},\ \tau_{subst} = \tau_0 \cdot \exp\left(\frac{E_{de}}{k_b T}\right),\ \tau_{ads} = \tau_0 \cdot \exp\left(\frac{H_{vap}}{k_b T}\right),\ t_{ads} = \tau_{ads} \cdot (\theta - 1)$$

$$des = \begin{cases} 0, & \text{if } \theta = 0 \text{ or } T = 0 \\ \theta \cdot \big(1 - \exp(-t_{step}/\tau_{subst})\big), & \text{else if } \theta \leq 1 \\ t_{step}/\tau_{ads}, & \text{else if } \theta - 1 \geq t_{step}/\tau_{ads} \\ \theta - 1 + \big(1 - \exp(-(t_{step} - t_{ads}/\tau_{subst}))\big), & \text{else if } \theta - 1 < t_{step}/\tau_{ads} \end{cases} \tag{A.10}$$

Outgassing *out*

$$out = \frac{\text{Facet outgassing}}{k_b T} \tag{A.11}$$

Particle density

$$density = \frac{\text{sum over reciprocal of orthogonal velocity}}{\text{Area of Facet } [\text{m}^2] \cdot t_{step}} \cdot K_{\text{real/virtual}} \tag{A.12}$$

Pressure [mbar]

$$density = \frac{\text{sum over orthogonal velocity}}{\text{Area of Facet } [\text{m}^2] \cdot t_{step}} \cdot \frac{\text{carbon equivalent gas mass}}{1000/N_A} \cdot 0.01 \cdot K_{\text{real/virtual}} \tag{A.13}$$

Small covering factor

$$mincov = \text{Smallest covering on a single facet that desorbs}$$

$$\text{small covering factor} = \begin{cases} 1, & \text{if } mincov \geq \text{ProblemDef::coveringMinThresh} \\ 1 + 1.1 \cdot (\text{ProblemDef::coveringMinThresh}/mincov), & \text{otherwise} \end{cases} \tag{A.14}$$

$K_{\text{real/virtual}}$

$$K_{\text{real/virtual}} = \frac{\sum\limits_{\text{facets}} \Big(out + des\Big)}{\text{number of total desorbed molecules/small covering factor}} \tag{A.15}$$

Statistical error

$$\text{error}(counter) = \begin{cases} inf & , \text{ if } (counter) \text{ on facet } = 0 \\ \sqrt{\dfrac{1}{(counter) \text{ on facet}} \cdot \left(1 - \dfrac{(counter) \text{ on facet}}{\text{total } (counter)}\right)} & , \text{else} \end{cases} \tag{A.16}$$

$$error\_covering = \text{error(adsorbed particles + desorbed particles)}$$
$$error\_event = \text{error(hits + desorbed particles)}$$

B. Datatypes

B.1. Boost

Datatype	Alias
boost::multiprecision::uint_128t	uint_128t
boost::multiprecision::float128	float128

B.2. Class Members

Name	Datatype	Note
SimulationHistory::coveringList	uint_128t	
FacetHitBuffer::covering	llong	hitbuffer & tmpcounter
FacetProperties::desorption	float128	
Simulation::coveringThreshold	llong	

B.3. Functions

Function	Output Datatype	Relevant Input
getCovering()	float128	SimulationHistory::coveringList
getCovering()	llong	FacetHitBuffer::covering
calcCoverage()	float128	getCovering()
calcDesorption()	float128	calcCoverage()
calctotalDesorption()	float128	FacetProperties::desorption
GetMoleculesPerTP()	float128	FacetProperties::desorption

C. Overview of new Classes and Functions

C.1. New Classes

HistoryList	
historyList	list containing history respective facet values
currentList	list containing facet values at current step
predictList	list containing facet values for predictor-corrector method
statisticsList	list containing facet statistics over last iterations
currIt	current iteration number
reset()	Reset lists
initCurrent()	Initialize size of currentList
initPredict()	Initialize size of predictList
initStatistics()	Initialize size of statisticsList
initList()	Initialize size of historyList
appendCurrent()	Append currentList to historyList
appendList()	Append input list to historyList
updateStatistics()	Calculate statistics per facet (mean, std), save to statisticsList
getAverageStatistics()	Calculate average ratio (std/mean) weighted with area for all facets or focusGroup only
convertTime()	Convert time for better clarity
print()	Print historyList to terminal, optinal message
printCurrent()	Print currentList as table to terminal, optional message
printPredict()	Print predictList as table to terminal, optional message
printStatistics()	Print statisticsList as table to terminal, optional message
write()	Write historyList to file
erase()	delete desired point in historyList
empty()	Check if historyList is empty
setCurrent()	Set value of desired facet in currentList
getCurrent()	Get value of desired facet in currentList
setPredict()	Set value of desired facet in predictList
getPredict()	Get value of desired facet in predictList
setLast()	Set value of desired facet from historyList
getLast()	Get value of desired facet from historyList

SimulationHistory	
coveringList	of class HistoryList, stores covering history
errorList_event	of class HistoryList, stores error history for events
errorList_covering	of class HistoryList, stores error history for covering
hitList	of class HistoryList, stores hits for each facet
desorbedList	of class HistoryList, stores desorbed particles for each facet
adsorbedList	of class HistoryList, stores adsorbed particles for each facet
particleDensityList	of class HistoryList, stores particle density for each facet
pressureList	of class HistoryList, stores pressure for each facet
numFacet	number of facets
numSubProcess	number of sub processes used for simulation
flightTime	Simulated flight time for iteration
nParticles	Simulated particles for iteration
lastTime	Total simulated time = last time in historyList
currentStep	step of logarithmic time step calculation in `getStepSize()`
pcStep	current step of predictor-corrector method
stepSize	current step size
stepSize_outgassing	current step size of outgassing impulse
smallCoveringFactor	Factor used to multiply covering to reach a minimal value
updateHistory()	Reset and update
updateStepSize()	Calculate stepSize and stepSize_outgassing
erase()	Erase desired point in history
print()	Print to terminal
write()	Write to file

ProblemDef	
contaminationFlowPath	Path of github directory
resultPath	Path of result folder
outFile	Path of file that contains terminal output
loadbufferPath	Path of loadbuffer file
hitbufferPath	Path of hitbuffer file
coveringPath ⇒doCoveringFile	Path of covering file, hitbuffer not imported if given
saveResults	1: save all results, 0: do not save results
simulationTime, unit ⇒simulationTimeMS	Computation time of each iteration in milliseconds
maxTime, maxUnit ⇒maxTimeS	Maximal total simulated time in seconds
iterationNumber	Number of iterations of simulation
usePCMethod	0: do not use-, 1 or 2: use predictor-corrector-method v1 or v2
particleDia	Diameter of particles
E_de, H_vap	Parameters to calculate binding energy, see eq. A.7
sticking	Sticking factor for all facets
targetParticles/-Error	Target values for each iteration
hitRatioLimit	Threshold of hit ratio (facet/total) at which hits are ignored
coveringMinThresh	Minimum covering, multiplication to this if covering low
t_min, t_max	Minimum/ Maximum step size
maxTimePerIt	Maximun simulation time [s] per iteration
histSize	Size of historyList objects (most recent values in memory)
vipFacets	Alterning: vip facet and target error, e.g. 1 0.001 3 0.002
outgassingTimeWindow	Duration of outgassing impulse
counterWindowPercent	[%] of step size (posterior) at which velocity counters are increased
desWindowPercent	[%] of step size (anterior) for desorption
rollingWindowSize	Number of iterations over which statistics are calculated
convergenceTarget	Target for average ratio (std/mean) for convergence
stopConverged	1: stop simulation at convergence, 0: continue simulation
facetGroups	Indices of facets belonging to a group, groups divided by -
focusGroup	Indices of facet groups to be monitored
doFocusGroupOnly	1: only monitor focus group, 0: monitor all facets
createOutput()	Create output directory and file
readInputfile()	Initialization from input file, checks if parameters are valid
printInputfile()	Print to terminal
writeInputfile()	Write to text file
SetFocusGroup()	Converts focusGroup indices to facet indices

C.2. New Functions

C.2.1. molflowlinux_main.cpp

Preprocessing	
parametercheck()	Checks validity of input parameters from input file Defines values for ProblemDef object `p`
importBuff()	Import load- and hitbuffer to main process
MPI_Bcast()	Send loadbuffer to sub processes
loadAndCheckSHandle()	Load geometry from loadbuffer and check values
initCoveringThresh()	Initialize covering threshold
UpdateSojourn()	Enable sojourn time for each facet
`simHistory`	Initialize SimulationHistory object

Simulation Loop	
initbufftozero()	Reset all hitbuffer counters except covering
MPI_Bcast()	Send `simHistory→coveringList` and `simHistory→currentStep` to sub processes
setCoveringThreshold()	Sets covering threshold for each facet
updateStepSize()	Calculates step sizes for desorption and outgassing
CalcTotalOutgassingWorker()	Calculates total outgassing for iteration
UpdateDesorption()	Sets desorption for each facet
checkSmallCovering()	multiplies covering to reach threshold if necessary
simulateSub2()	Simulation on sub processes
MPI_Send(), MPI_Recv()	Send sub hitbuffer to main process
UpdateMCMainHits()	Add simulation results to main hitbuffer
UpdateParticleDensityAndPressure()	Calculate and save particle density and pressure
UpdateErrorMain()	Calculate and save error of iteration to simHistory
UpdateCovering()	Calculate and save new covering to simHistory
UpdateCoveringphys()	Save current covering to hitbuffer
`simHistory→erase()`	Adapt historyList size of to `p→histSize`
updateStatistics(), getAverageStatistics()	Statistics over `p→rollingWindowSize` iterations
End simulation if `p→maxTimeS` or `p→convergenceTarget` is reached	

Postprocessing	
simHistory→write()	Export simulation history

C.2.2. SimulationLinux.cpp

simulateSub2()	
`simHistory->updateHistory()`	Update SimulationHistory object from `sHandle`
smallCoveringFactor	If covering is small: multiplied by smallCoveringFactor to be able to have statistics without overflow of covering variable
targetParticles, targetError	Calculate target values from overall target /# sub processes
SimulationRun()	Simulate for desired simulation time
UpdateError()	Calculate current error of sub process
CheckErrorSub()	Checks if total error reached targetError and if vip facets reached own target
UpdateMCSubHits()	Save simulation results to hitbuffer

Small covering	
checkSmallCovering()	Find smallCoveringFactor to reach `p→coveringMinThresh`
Undo multiplication	In `UpdateCovering()`

Others	
readCovering()	Reads covering or covereage values, save to buffer
get_path()	Get path of executable
printStream()	Print input string to terminal and file
tilde_to_home(),home_to_tilde()	Exchange ~ and home directory
convert_to/from_contflowdir()	Exchange CONTFLOWDIR and `p→contaminationFlowPath`

C.2.3. Iteration.cpp

Set Covering Threshold to avoid negative covering	
initCoveringThresh()	Initialize size of covering threshold vector
setCoveringThreshold()	Set covering threshold for each facet

Error calculations	
getErrorList()	get pointer to list corresponding to `simHistory→errorMode`
getErrorVariables()	get number hits, adsorbed, desorbed particles
UpdateErrorList()	Calculate error per facet, see eq. A.16. Save to `simHistory`
CalcErrorAll()	Sum up facet errors & weight by area for all error modes
CheckError()	Check if total error and vip facet error reached target

C.2.4. Buffer.cpp

Buffer functions	
Databuff struct()	signed int size BYTE *buff
initBuffSize()	Initialize size of buffer (without content)
checkReadable()	Check if file can be opened for reading
checkWriteable()	Check if file can be openend or created for writing
importBuff()	Import buffer file to Databuff struct
exportBuff()	Export Databuff struct to buffer file

C.2.5. Calculations in SimulationCalc.cpp etc.

SimulationCalc.cpp	
getCovering()	Get covering from hitbuffer or `simHistory`
getHits()	Get number of hits from hitbuffer
getnbDesorbed()	Get number of total desorbed molecules from hitbuffer
getnbAdsorbed()	Get number of total adsorbed molecules from hitbuffer
calcNmono()	see eq. A.3
calcCoverage()	see eq. A.5
calcSticking()	see eq. A.6
calcDesorption()	see eq. A.10
GetMoleculesPerTP()	see eq. A.15
calctotalDesorption	Calculate desorption for `startFromSource()`
calcOutgassingFactor()	Calculate factor to determine outgassing particles
calcPressure()	see eq. A.13
calcParticleDensity()	see eq. A.12
calcStartTime()	Calculate start time of particle depending on desorption/outgassing distribution

worker.cpp	
CalcTotalOutgassingWorker()	see eq. A.11, calculate outgassing distribution for `startFromSource()`

SimulationLinux.cpp	
convertunit()	Convert simutime · unit to milliseconds

C.2.6. UpdateSubProcess.cpp

Update sHandle paramters from hitbuffer	
UpdateSticking()	Update sticking
UpdateDesorption()	Update desorption
UpdateSojourn()	Enable residence time for all facets

Error calculations	
UpdateErrorSub()	UpdateErrorList()
CalcErrorSub()	CalcErrorAll() for only one error quantity in sub process

Update hitbuffer	
initbufftozero()	Set hitbuffer except covering to zero
UpdateMCSubHits()	Save simulation results from sHandle into hitbuffer

C.2.7. UpdateMainProcess.cpp

Update main hitbuffer from sub hitbuffer	
UpdateMCMainHits()	Add simulation results from sub hitbuffer to main hitbuffer

Update real covering in hitbuffer	
getStepSize()	Calculate step size for current step, see eq. A.9
UpdateCovering()	Use Krealvirt to calculate new covering Save to `simHistory→coveringList`
UpdateCoveringphys()	Save current real covering to hitbuffer
UpdateErrorMain()	UpdateErrorList(), adapt time entries
UpdateParticleDensityAndPressure()	Calculate pressure and particle density, see eq. A.12, A.13
CalcPerIteration()	Calculate total error (covering and event) and covering over all facets per iteration