

Documentation

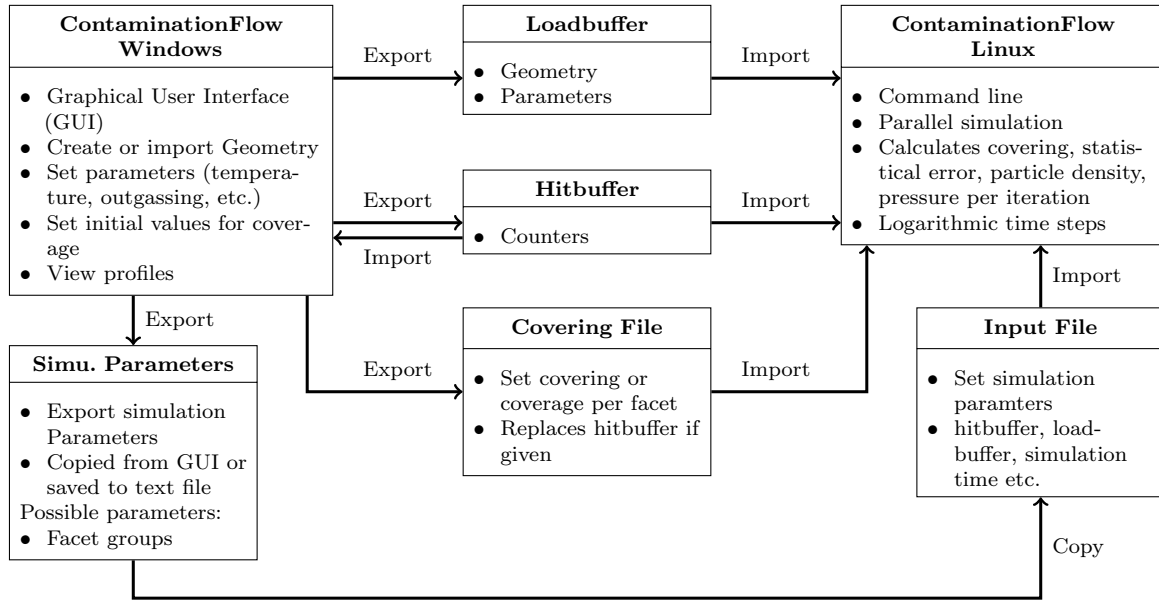
ContaminationFlow on Linux and Windows

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 <i>boost</i> 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 | 14 |
| 3.3. New Quantities | 15 |
| A. Formulas for new Quantities | 16 |
| B. Datatypes | 18 |
| B.1. Boost | 18 |
| B.2. Class Members | 18 |
| B.3. Functions | 18 |
| C. Overview of new Classes and Functions | 19 |
| C.1. New Classes | 19 |
| C.2. New Functions | 22 |
| C.2.1. molflowlinux_main.cpp | 22 |
| C.2.2. SimulationLinux.cpp | 23 |
| C.2.3. Iteration.cpp | 23 |
| C.2.4. Buffer.cpp | 24 |
| C.2.5. Calculations in SimulationCalc.cpp etc. | 24 |
| C.2.6. UpdateSubProcess.cpp | 25 |
| C.2.7. UpdateMainProcess.cpp | 25 |

1. General Structure



General structure for ContaminationFlow simulation

- Code adapted from Molflow
- The Windows executable of ContaminationFlow is used to create Geometry, define the initial value problem and export it via two files (Loadbuffer & Hitbuffer). It cannot simulate the contamination transfer.
- The Linux version of ContaminationFlow is used for simulation.
- Loadbuffer contains information of geometry.
- Hitbuffer contains information such as hit counters, profiles, etc.
- Import and export of buffer files are used for communication between ContaminationFlow on Windows and ContaminationFlow on Linux. The import function of the loadbuffer does not work properly yet.
- Optional: export/import of covering text file that replaces covering in hitbuffer

2. ContaminationFlow Linux

- Parallel simulation on several subprocesses
- Control and of data processing in main process

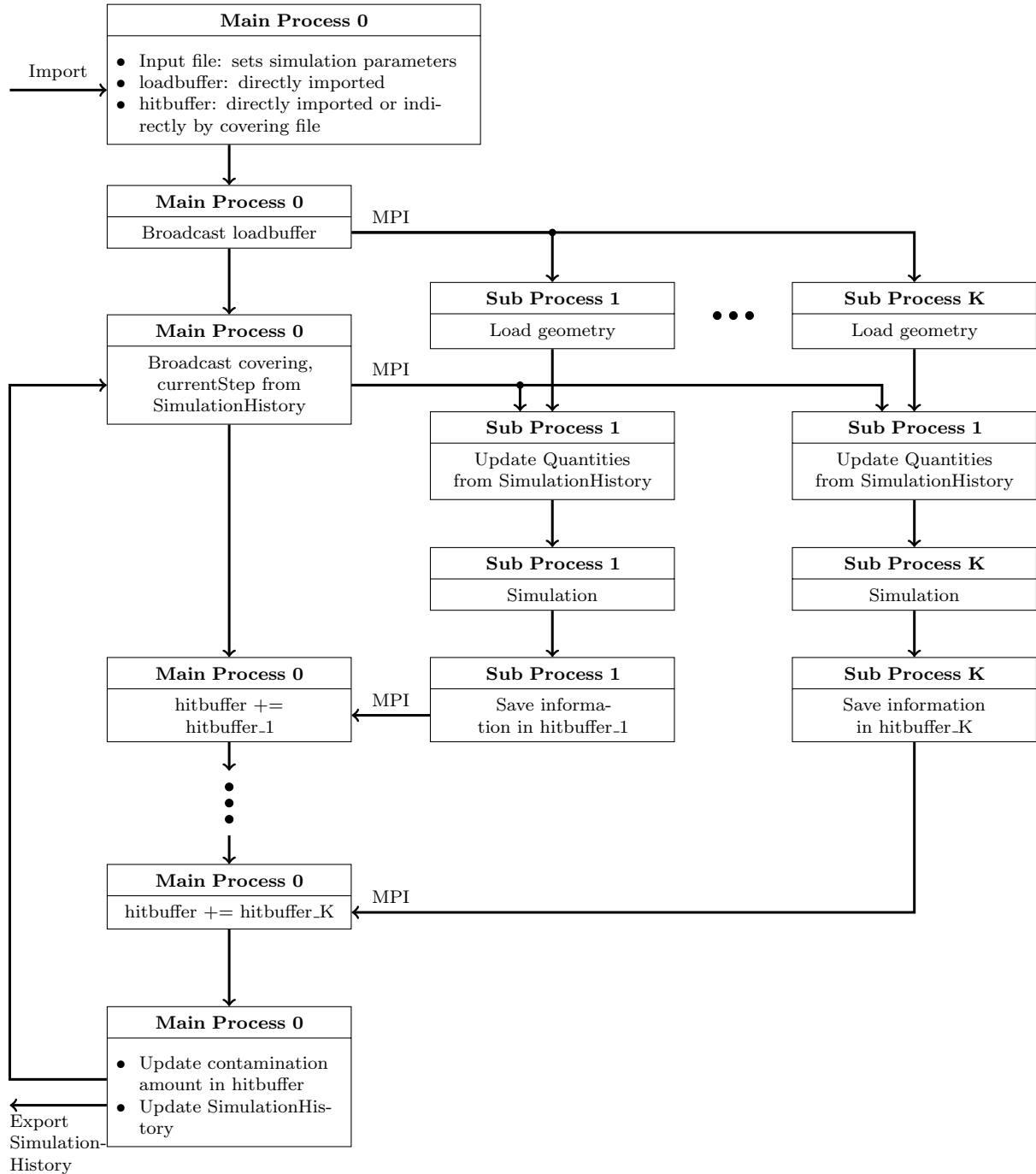


Figure 2.1.: Processing of data in main and sub processes

2.1. Call of Application from Command line

New class ProblemDef

- Defines parameters used for simulation
- Possible adaptation of default parameters 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

For convenience use the tcshell script:

- `source StartContaminationFlow N inputfile`: MPI is loaded and `MolflowLinux` is executed.

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 the Windows version of ContaminationFlow, 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`: Maximum simulated time; default: 10.0
- `maxUnit`: Maximum simulated time unit; default: y
- `iterationNumber`: Number of iterations; default: 100
- `usePCMMethod`: Use predictor-corrector method "1", "2" or simulate without predictor-corrector method "0"; default: 0
- `Ede`: Binding energy of a particle on pure substrate; default: 1.6E-19
- `Hvap`: Vaporization enthalpy of a particle if multilayer contamination; default: 0.8E-19
- `errorMode`: Type of error monitored, "covering" or "event"; default: covering

- `targetParticles` : Minimum number of desorbed test-particles per iteration; default: 1000
- `targetError` : Avg. statistical uncertainty (error) to be achieved for each iteration, calculated as the (by the facets area weighted) average 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 fraction of the step size) at which pressure and density is sampled; default: 0.1
- `rollingWindowSize` : Number of iterations for average statistics; default: 10
- `convergenceTarget` : Target for average statistics to indicate convergence; default: 1
- `stopConverged` : Stop simulation if average statistics indicate convergence; default: 1
- `convergenceTime` : Minimum 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
- `saveResults` : Determines if results are saved; default: 1
- `saveConsole` : Determines if console output is additionally saved; default: 0

Optional covering file to replace covering values from hitbuffer. Hitbuffer's covering values 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, separated via blanks
- set coverage: `coverage` followed by coverage values per facet, separated 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(.)`
- Subprocesses 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 particles on facet
- Increases with adsorption, decreases with desorption
- Extracted from new FacetHitBuffer counter in `getCovering()` (`SimulationCalc.cpp`)

Coverage

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

Binding energy

- Either E_{de} or H_{vap} , depending on the coverage
- 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 desorbing particles
- Calculated from binding energy, coverage, facet area, temperature and step size
- Desorption computed in `calcDesorption()` (`SimulationCalc.cpp`)

Outgassing

- Number of particles outgassing from the bulk of the material, whose surface is represented by a facet
- Calculated from facet's (time-dependent) outgassing
- Outgassing computed in `CalcTotalOutgassingWorker()` (`Worker.cpp`)

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

- Number of real, physical particles represented by Monte-Carlo test-particles
- Calculated from desorption & outgassing as well as the number of desorbed & outgassed test-particles
- $K_{\text{real/virtual}}$ computed in `GetMoleculesPerTP()` (`SimulationCalc.cpp`)

Statistical uncertainty ("error")

- Statistical uncertainty based on all particle-wall interaction events ('event'): calculated from the number of 'hits', desorbed and outgassed test-particles (of every facet and the entire surface)
- Statistical uncertainty based on all particle-wall interaction events, which change the 'covering' value ('covering'): calculated from adsorbed and desorbed test-particles (of every facet and the entire surface)
- Used to determine significance of simulation results of iteration
- Error calculated in `UpdateErrorList()` for facet error & in `UpdateErrorAll()` for the average error weighted by the facets' areas (`UpdateSubProcess.cpp`)

Step size

- Step size computed in `getStepSize()` (`UpdateMainProcess.cpp`)

Particle density

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

Pressure

- Calculated from sum of orthogonal velocity portions, 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
- Start time computed in `calcStartTime()` (`SimulationCalc.cpp`)

2.5. Iterative Algorithm

2.5.1. Initialization of simulation

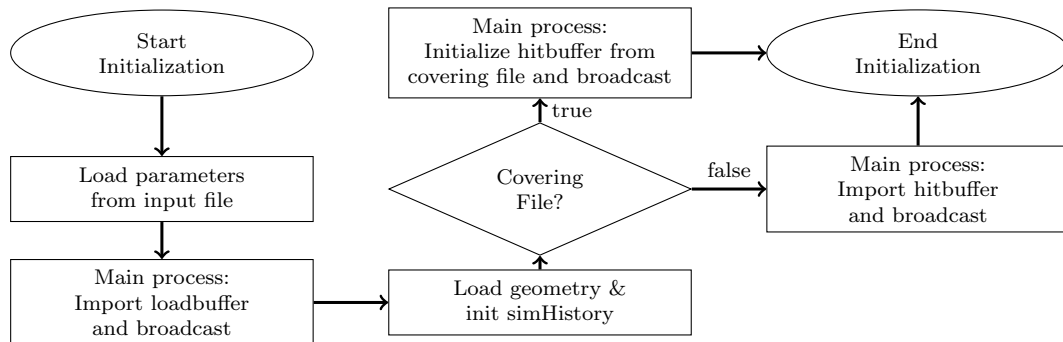


Figure 2.2.: Overview: Initialize simulation

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, desorbed and outgassed test-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 counter so that smallest covering \geq `ProblemDef::coveringThreshMin`
- Multiply covering threshold with same factor
- Calculation in `checkSmallCovering()` (`SimulationLinux.cpp`)

Calculate desorption and outgassing

- Desorption 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 temporal distribution of desorption or outgassing

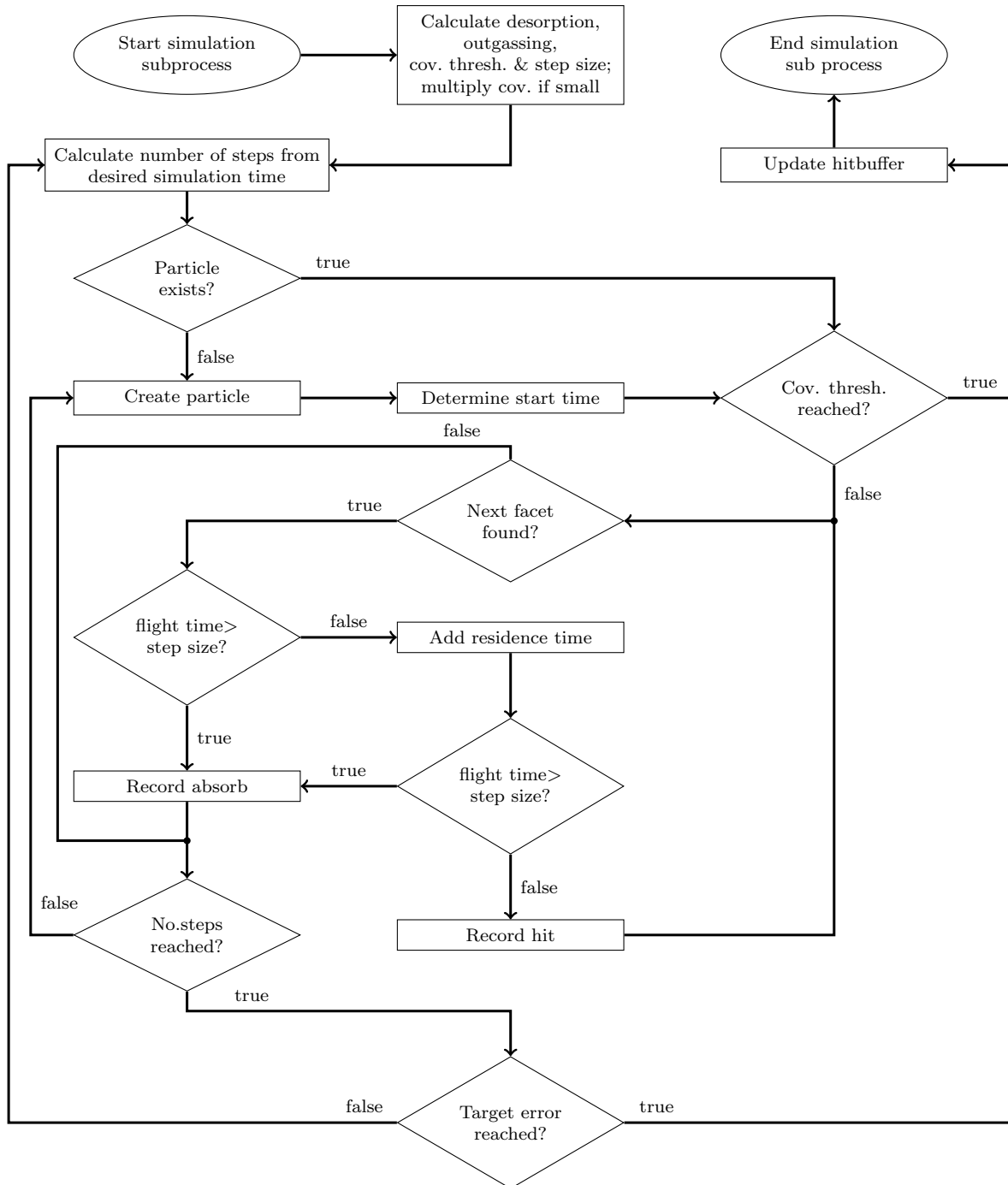


Figure 2.3.: Overview: simulation on subprocesses

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

Calculate residence time

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

Increase facet counters in case of desorb, absorb or hit

- Increase hit, desorb, outgassed 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 uncertainty 'error' in `UpdateError()` from `UpdateSubProcess.cpp` file
- Average error calculated from summing facet error weighted with facet area
- Error to check can be either covering or event error
 - 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 did not reach the `ProblemDef::hitRatioLimit` are excluded of the error calculation
 - Facets with no events or no change of covering (no desorb and no absorb)
 - If vip facet: own target error automatically reached
- Check in `checkErrorSub(.)` (`UpdateSubProcess.cpp`)

2.5.3. Update main process

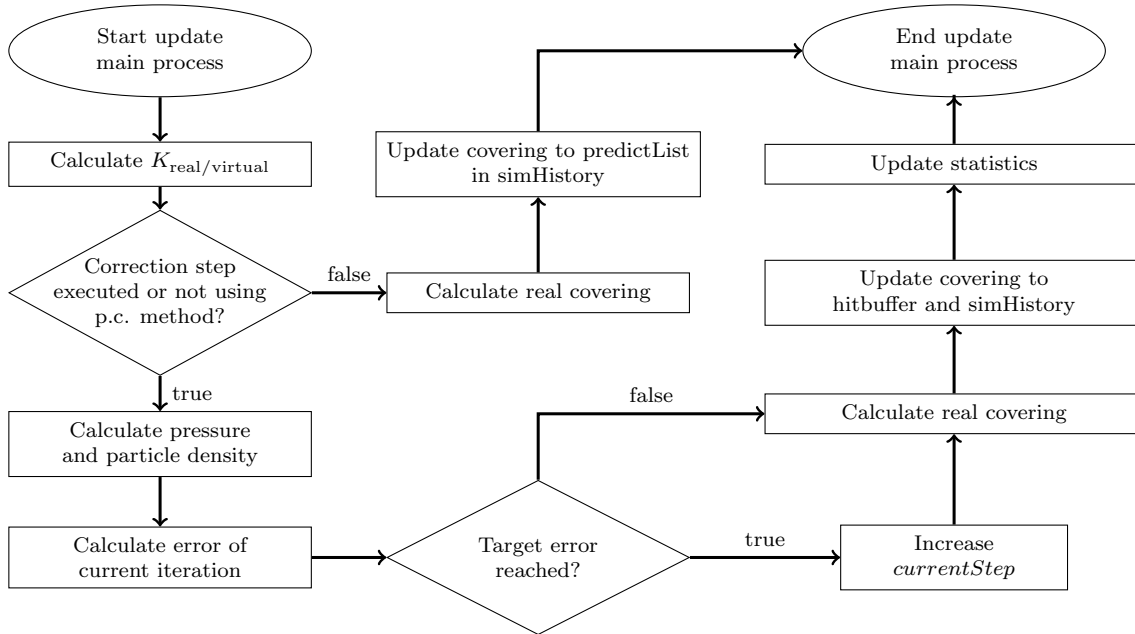


Figure 2.4.: Overview: update of covering in hitbuffer and simHistory

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`), the covering values from `currentList` if `sHandle->currentParticle.flightTime <= 1/2*simHistory->stepSize` will be read, otherwise covering values from `predictList` will be read.
- 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`
 \Rightarrow 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

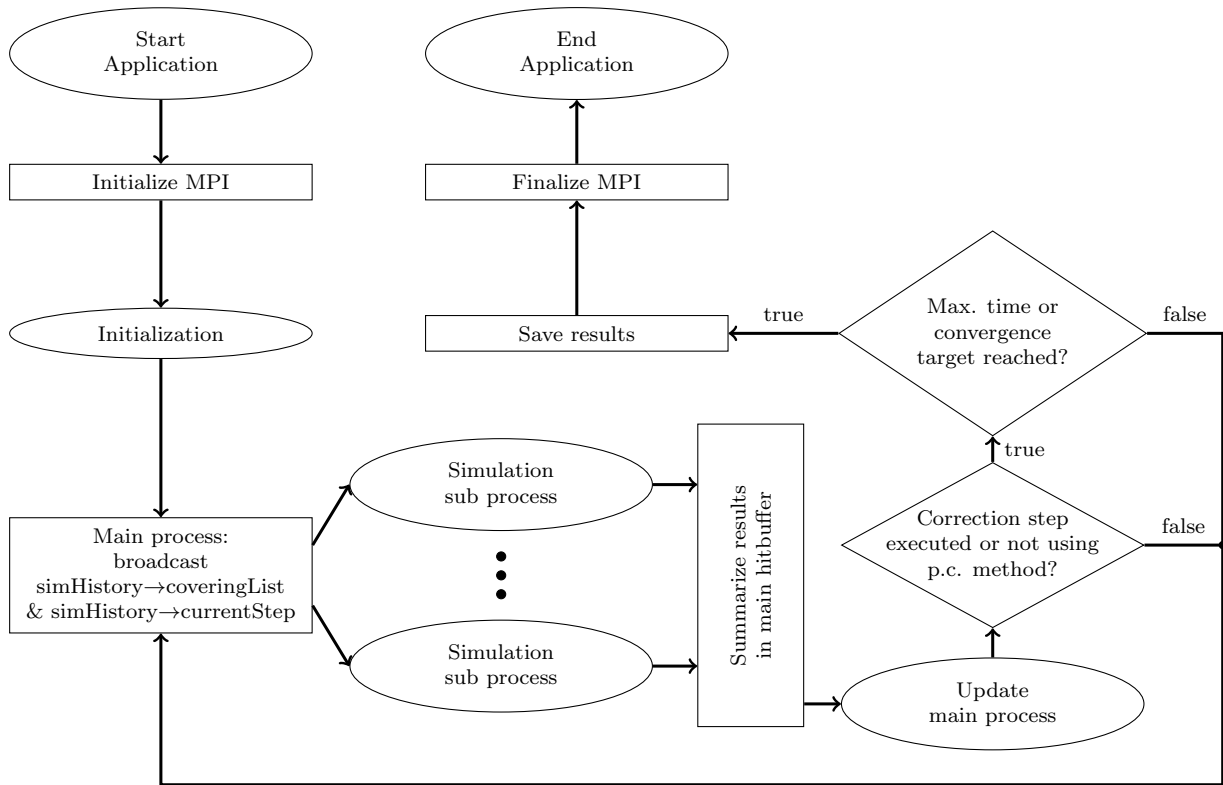
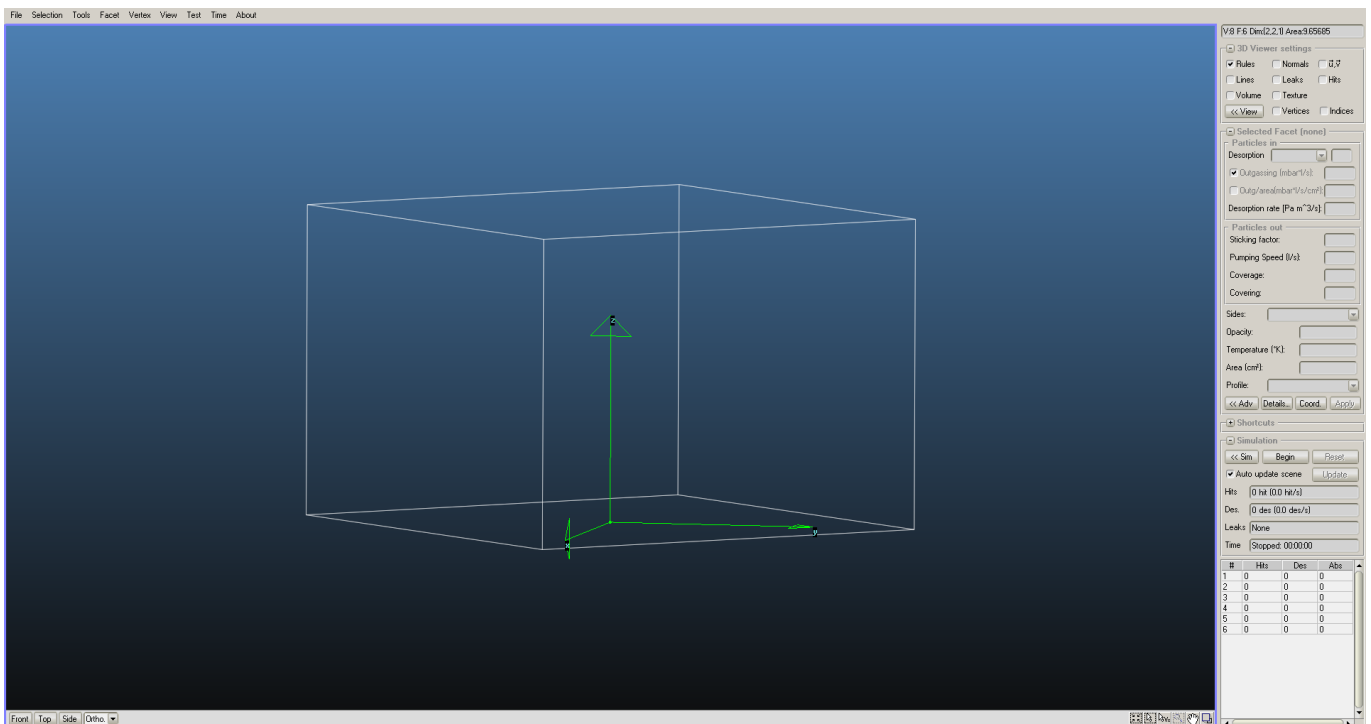


Figure 2.5.: Overview: ContaminationFlow application

3. ContaminationFlow Windows

- Extension of original Molflow for contamination problems
- Create Geometry and set parameters such as initial coverage and temperature
- Export of buffer files
- Export of facet groups

3.1. Graphical User Interface



New GUI elements

- "Particles out"
 - Text field for covering
 - Text field for coverage
- New menu options
 - File: Export 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 formatting 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 formatting 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

A. Formulas for new Quantities

Constants

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

Variables

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

Number of particles of one monolayer

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

Covering θ^*

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

Coverage θ

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

Binding energy E

$$E = \begin{cases} E_{de}, & \text{if particle binds with substrate} \\ H_{vap}, & \text{if particle binds with adsorbate} \end{cases}\tag{A.6}$$

Residence time τ

$$\begin{aligned}A &= \exp(-E/(k_b T)), \quad \tau_0 = \frac{h}{k_b T} \\ \tau &= \frac{-\ln(rnd) \cdot \tau_0}{A}\end{aligned}\tag{A.7}$$

Desorption des

$$\tau_0 = \frac{h}{k_b T}, \quad \tau_{subst} = \tau_0 \cdot \exp\left(\frac{E_{de}}{k_b T}\right), \quad \tau_{ads} = \tau_0 \cdot \exp\left(\frac{H_{vap}}{k_b T}\right)\tag{A.8}$$

Outgassing out

$$out = \frac{\text{Facet outgassing}}{k_b T}\tag{A.9}$$

Particle density

$$density = \frac{\text{sum over reciprocal of orthogonal velocity}}{\text{Area of Facet [m}^2] \cdot t_{step}} \cdot K_{\text{real/virtual}} \quad (\text{A.10})$$

Pressure [mbar]

$$density = \frac{\text{sum over orthogonal velocity}}{\text{Area of Facet [m}^2] \cdot t_{step}} \cdot \frac{\text{gas mass}}{1000/N_A} \cdot 0.01 \cdot K_{\text{real/virtual}} \quad (\text{A.11})$$

Small covering factor

$mincov$ = Smallest covering on a single facet that desorbs

$$\text{small covering factor} = \begin{cases} 1, & \text{if } mincov \geq \text{ProblemDef::coveringMinThresh} \\ 1 + 1.1 \cdot (\text{ProblemDef::coveringMinThresh}/mincov), & \text{otherwise} \end{cases} \quad (\text{A.12})$$

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

$$K_{\text{real/virtual}} = \frac{\sum_{\text{facets}} (out + des)}{\text{number of total desorbed molecules/small covering factor}} \quad (\text{A.13})$$

Statistical error

$$\text{error}(counter) = \begin{cases} inf, & \text{if } (counter) \text{ on facet} = 0 \\ \sqrt{\frac{1}{(counter) \text{ on facet}} \cdot \left(1 - \frac{(counter) \text{ on facet}}{\text{total } (counter)}\right)}, & \text{else} \end{cases} \quad (\text{A.14})$$

$error_covering$ = error(adsorbed particles + desorbed particles)

$error_event$ = 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 | hitbuffer & tmpcounter |
| FacetHitBuffer::covering | llong | |
| 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, optional 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 <code>getStepSize()</code> |
| 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 \Rightarrow doCoveringFile | Path of covering file, hitbuffer not imported if given |
| saveResults | 1: save all results, 0: do not save results |
| simulationTime, unit \Rightarrow simulationTimeMS | Computation time of each iteration in milliseconds |
| maxTime, maxUnit \Rightarrow maxTimeS | Maximal total simulated time in seconds |
| iterationNumber | Number of iterations of simulation |
| usePCMethod | 0: do not use predictor-corrector-method, 1 or 2: use predictor-corrector-r |
| particleDia | Diameter of particles |
| E_de, H_vap | Parameters to calculate binding energy, see eq. A.6 |
| 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 | Alternating: 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 <code>p</code> |
| 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 |
| <code>simHistory</code> | Initialize SimulationHistory object |

| Simulation Loop | |
|--|---|
| initbufftozero() | Reset all hitbuffer counters except covering |
| MPI_Bcast() | Send <code>simHistory→coveringList</code> and <code>simHistory→currentStep</code> 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 |
| <code>simHistory→erase()</code> | Adapt historyList size of to <code>p→histSize</code> |
| updateStatistics(), getAverageStatistics() | Statistics over <code>p→rollingWindowSize</code> iterations |
| End simulation if <code>p→maxTimeS</code> or <code>p→convergenceTarget</code> is reached | |

| Postprocessing | |
|---------------------------------|---------------------------|
| <code>simHistory→write()</code> | Export simulation history |

C.2.2. SimulationLinux.cpp

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

| Small covering | |
|-----------------------------------|---|
| <code>checkSmallCovering()</code> | Find <code>smallCoveringFactor</code> to reach <code>p->coveringMinThresh</code> |
| Undo multiplication | In <code>UpdateCovering()</code> |

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

C.2.3. Iteration.cpp

| Set Covering Threshold to avoid negative covering | |
|---|--|
| <code>initCoveringThresh()</code> | Initialize size of covering threshold vector |
| <code>setCoveringThreshold()</code> | Set covering threshold for each facet |

| Error calculations | |
|----------------------------------|--|
| <code>getErrorList()</code> | get pointer to list corresponding to <code>simHistory->errorMode</code> |
| <code>getErrorVariables()</code> | get number hits, adsorbed, desorbed particles |
| <code>UpdateErrorList()</code> | Calculate error per facet, see eq. A.14. Save to <code>simHistory</code> |
| <code>CalcErrorAll()</code> | Sum up facet errors & weight by area for all error modes |
| <code>CheckError()</code> | 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 opened 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 <code>simHistory</code> |
| 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 |
| calcDesorption() | see eq. A.8 |
| GetMoleculesPerTP() | see eq. A.13 |
| calcTotalDesorption | Calculate desorption for <code>startFromSource()</code> |
| calcOutgassingFactor() | Calculate factor to determine outgassing particles |
| calcPressure() | see eq. A.11 |
| calcParticleDensity() | see eq. A.10 |
| calcStartTime() | Calculate start time of particle depending on desorption/outgassing distribution |

| worker.cpp | |
|-----------------------------|---|
| CalcTotalOutgassingWorker() | see eq. A.9, calculate outgassing distribution for <code>startFromSource()</code> |

| 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. ?? |
| UpdateCovering() | Use Krealvirt to calculate new covering Save to <code>simHistory→coveringList</code> |
| UpdateCoveringphys() | Save current real covering to hitbuffer |
| UpdateErrorMain() | UpdateErrorList(), adapt time entries |
| UpdateParticleDensityAndPressure() | Calculate pressure and particle density, see eq. A.10, A.11 |
| CalcPerIteration() | Calculate total error (covering and event) and covering over all facets per iteration |