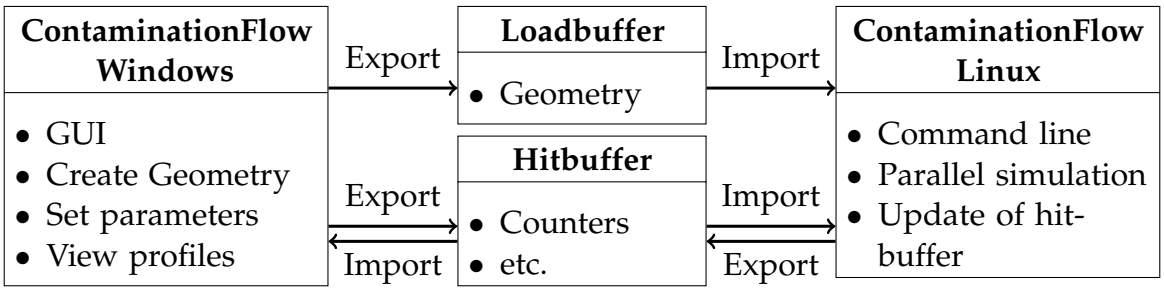
Documentation

ContaminationFlow on Linux and Windows

Hoai My Van

Contents

1. General Structure

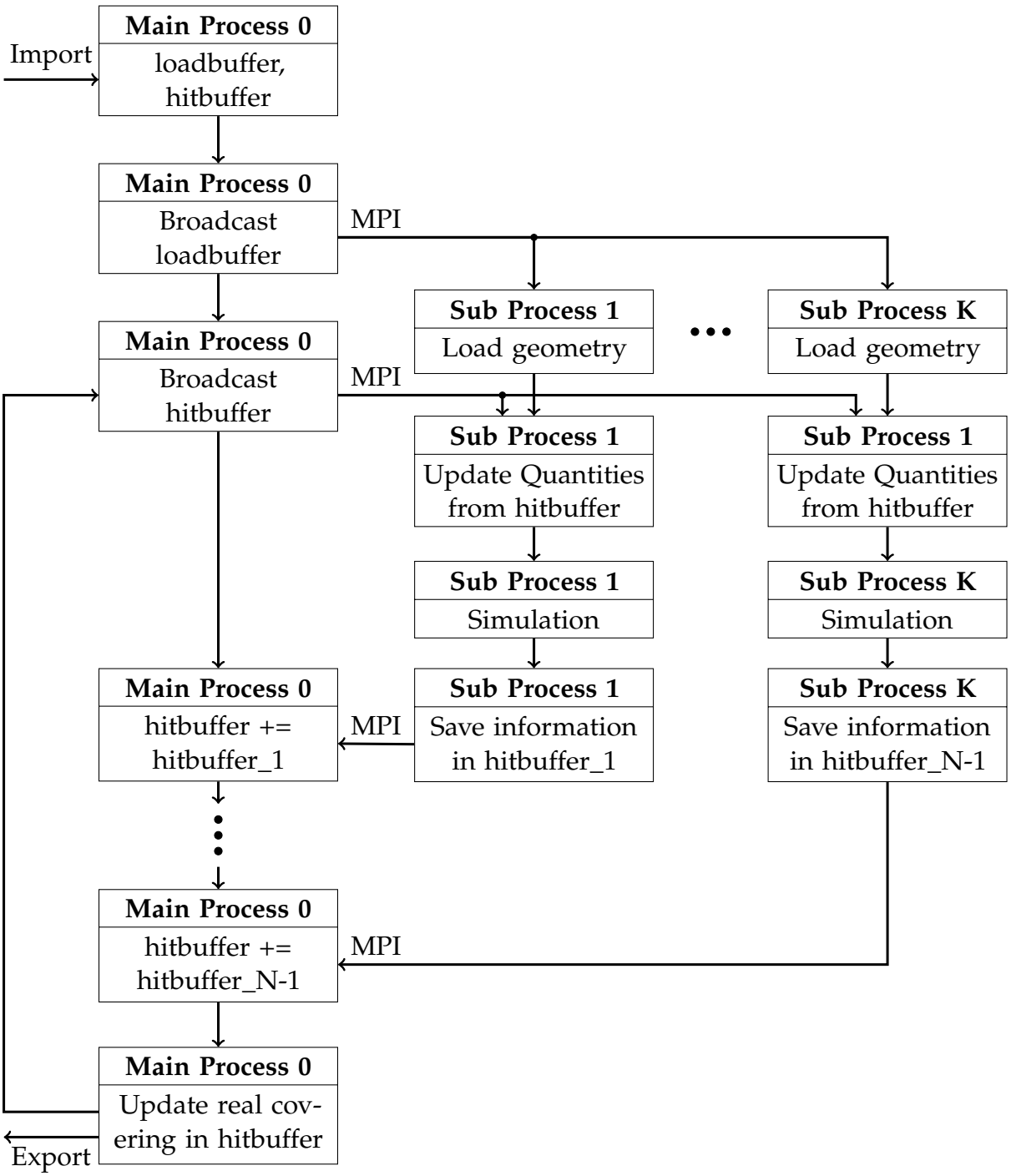


General structure for ContaminationFlow simulation

- 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
- 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, final hitbuffer and used parameters exported to results folder



2.1. Call of Application from Command line

2.1.1. Application with standard parameters

Call Molflow Linux application with standard parameters in the command line:

```
$ module load mpi
$ mpirun -n N MolflowLinux loadbuffer hitbuffer resultbuffer
simulationtime unit
```

with the following parameters:

- n: desired number of worker processes; simulation on K=N-1 worker processes
- MolflowLinux: path to application, e.g. `~/MolflowLinux/Debug/MolflowLinux`
- loadbuffer: path to loadbuffer file, that contains geometry, e.g. `~/loadbuffer`
- hitbuffer: path to hitbuffer file, that contains counters, etc., e.g. `~/hitbuffer`
- resultbuffer: path to resultbuffer file, where the final hitbuffer is exported to, e.g. `~/resultbuffer`
- simulationtime: simulation time, e.g. `2.5`
- unit (optional): simulation time unit, e.g. `min`; default: `s`

2.1.2. Application with custom parameters

Call Molflow Linux application with custom parameters in the command line:

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

with the input file defining the following parameters:

- n: desired number of worker processes; simulation on K=N-1 worker processes
- MolflowLinux: path to application, e.g. `~/MolflowLinux/Debug/MolflowLinux`
- loadbufferPath: path to loadbuffer file, that contains geometry, e.g. `~/loadbuffer`
- hitbufferPath: path to hitbuffer file, that contains counters, etc., e.g. `~/hitbuffer`
- resultbufferPath: path to resultbuffer file, where the final hitbuffer is exported to, e.g. `~/resultbuffer`
- simulationTime (optional): simulation time per iteration step; default: `10.0`
- unit (optional): simulation time unit; default: `s`
- maxTime (optional): maximum simulation time; default: `10.0`
- maxUnit (optional): maximum simulation time unit; default: `y`
- iterationNumber(optional): number of iteration with legth of simulationTime * unit; default: 43200
- s_1, coefficient used for calculation of sticking coefficient; default: `1`
- s_2, coefficient used for calculation of sticking coefficient; default: `0.2`
- E_{ad}, energy used for calculation of sticking coefficient; default: `1E-21`
- E_{de}, energy used for calculation of desorption; default: `1E-21`
- d, power for base of coverage used for calculation of desorption; default: `1`

2.2. Application

2.2.1. General Changes

Replacement/removal of Windows libraries/functions

- Databuff struct with import/export instead of using Dataports
- Replacements of libraries, e.g. `#include <time.h>` with `#include <sys/time.h>`

Removal of functions used in `AC_MODE`

- Only `MC_MODE` used
- Removal of `AC_MODE` cases and functions

New class ProblemDef

- Defines parameters used for simulation
- Possible adaptation of paramaters through input file or command line arguments
- Conducts some intern conversions
- Creates result folder for each simulation

2.2.2. Communication

Import and export of buffer files

- New Databuff struct

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

- New functions `importBuff(·)` and `exportBuff(·)` for import of buffer files and export of Databuff struct
- New functions `checkReadable(·)` and `checkWriteable(·)` to check if file is readable or writeable

Communication between worker processes via MPI

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

2.2.3. New Quantities

New counter `covering`

- Number of carbon equivalent particles on facet
- Added covering counter to hitbuffer
- Extracted from hitbuffer from `Simulationcalc.cpp` file in `getCovering(·)`
- Covering increases with adsorption, decreases with desorption

Coverage

- Number of carbon equivalent particles per monolayer on facet
- Calculated from covering, gas mass and facet area
- Coverage computed from `Simulationcalc.cpp` file in `calcCoverage(·)`

Sticking factor

- Calculated from coverage (and temperature)
- Sticking factor computed from `Simulationcalc.cpp` file in `calcStickingnew(·)`
- Updated and fixed before each interation

Desorption $[1/\mathbf{s}]$

- Calculated from covering and temperature
- Desorption computed from `Simulationcalc.cpp` file in `calcDesorption(·)`

Desorption Rate $[\mathbf{Pa\ m^3/s}]$

- Calculated from desorption, gas mass and facet area
- Desorption rate computed from `Simulationcalc.cpp` file in `calcDesorptionRate(·)`
- Used to determine starting point for new particle
- Updated and fixed before each interation

Outgassing Rate

- Calculated from sHandle values: facet outgassing and temerature
- Outgassing rate computed from `Worker.cpp` file in `CalcTotalOutgassingWorker()`

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

- Number of real particles represented by test particles
- Calculated from desorption rate, outgassing rate and number of desorbed molecules
- $K_{real/virtual}$ computed from `Simulationcalc.cpp` file in `GetMoleculesPerTP(·)`

2.2.4. Iterative algorithm

General Pipeline

- Initialize MPI to have desired number of processes (minimum 2)
- Send loadbuffer from main process 0 to sub processes using MPI and create sHandle and load geometry using `InitSimulation()` and `LoadSimulation()` in all processes
- Start iteration: iterationNumber steps of length simulationTime * unit:
  - Send hitbuffer from main process to subprocess using MPI
  - Update sticking factor and desorption rate in sub processes using `UpdateSticking(·)` and `UpdateDesorptionRate(·)` from `UpdateSubProcess.cpp`
  - Simulate for SimulationTime * unit using `SimulationRun()`
  - Update hitbuffer of sub processes from sHandle using `UpdateSubHits(·)` from `UpdateSubProcess.cpp`
  - Update Main process:
    - Send hitbuffer from sub process to main process using MPI
    - Update hitbuffer of main process from sub process in `UpdateMainHits(·)` from `UpdateMainProcess.cpp`
  - Calculate real covering in main process using $K_{real/virtual}$ and simulated time step in `UpdateCovering(·)` from `UpdateMainProcess.cpp`, save in SimulationHistory
  - Update real covering in hitbuffer of main process in `UpdateCoveringphys(·)` from `UpdateMainProcess.cpp`
- Export final results (hitbuffer and simulationHistory) to results folder

New class to store Simulation History

- SimulationHistory class

```
class SimulationHistory {
public:
  SimulationHistory();
  SimulationHistory(Databuff *hitbuffer);
  HistoryList<llong> coveringList;
  double flightTime;
  int nParticles;

  void appendList(Databuff *hitbuffer, double time);
  void print();
  void write(std::string path);
};
```

```
template <typename T> class HistoryList {
public:
  HistoryList();
  std::vector<std::pair<double,std::vector<T> > >
pointintime_list;
  std::vector<T> currentList;
};
```

- In `SimulationLinux.h` and `SimulationLinux.cpp` file
- SimulationHistory updated after each iterarion in `UpdateCovering(·)` from `UpdateMainProcess.cpp` file
- Currently recorded quantities: covering for all facets
- Time given in simulated time (accumulated time steps) rather than computed time

Set covering threshold

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

Estimation of time step T_{min}

- Determines minimum timestep for simulation, average time between outgassing/desorption and adsorption
- T_{min} computed in `Iteration.cpp` file in `estimateAverageFlightTime()` using simulationHistory: flightTime/nParticles

Pretesting of time step T_{min}

- Checks whether current time step would cause covering to get negative
- Adapts time step if needed
- Pretesting in `UpdateMainProcess.cpp` file in `preTestTimeSteo(·)`

3. ContaminationFlow Windows

- Create Geometry and set parameters such as pumping speed or sticking
- Evaluate profiles such as pressure profile
- Simulation also possible for testing, but mostly done on Linux

3.1. Graphical User Interface

Add screenshot of GUI

New GUI elements

- Text field for covering
- Text field for coverage
- Text field for new sticking coefficient
- Window for CoveringHistory (reworked to SimulationHistory in ContaminationFlow Linux)
- PressureEvolution window expanded
  - Added list that contains information of graph
  - Option to show only selected facets or all
  - List exportable

3.2. Application

3.2.1. 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 struct
- New options in file menu: `Export buffer` and `Import buffer`

3.2.2. New Quantities

New counter `covering`

- Covering computed in `SimulationMC.cpp` file in `updatecovering(·)`
- Added covering counter to hitbuffer
- Added covering to GUI, can be defined through textfield

Compute sticking factor based on covering

- Sticking factor computed in `Molflow.cpp` file in `calcStickingnew()`
- Updated automatically whenever covering is changed

Removal of Flow-Sticking dependency

- `calcSticking()` and `calcFlow()` in `Molflow.cpp` file not used anymore
- Flow not needed for iterative Algorithm

3.2.3. Iterative algorithm

New class to store covering for all facets at any time

- In `HistoryWin.cpp` and `HistoryWin.h` file
- `std::vector< std::pair< double,std::vector<double> > > pointintime_list` to store points in time and respective covering for all facets
- New GUI option to add and remove entries for `pointintime_list`
- New GUI option to export or import a complete list

A. Formulas for new Quantities

Constants

$$\begin{aligned} carbondiameter &= 2 \cdot 76\text{E} - 12 \\ K_b &= 1.38\text{E} - 23 \\ \tau &= 1.0\text{E} - 13 \end{aligned} \tag{A.1}$$

Number of carbon equivalent particles of one monolayer

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

Carbon equivalent relative mass factor

$$\begin{aligned} \Delta N_{surf} &= \frac{\text{carbon equivalent gas mass}}{12.011} \\ N_{surf} &= \sum_{\substack{\text{adsorbed} \\ \text{particles}}} \Delta N_{surf} \end{aligned} \tag{A.3}$$

Coverage θ

$$\theta = \frac{N_{surf}}{N_{mono}} \tag{A.4}$$

Sticking factor sc

$$sc(\theta) = \begin{cases} (s_1(1-\theta) + s_2\theta) \cdot (1 - \exp(-\frac{E_{ad}}{K_b T})), & \text{if } \theta < 1 \\ s_2 \cdot (1 - \exp(-\frac{E_{ad}}{K_b T})), & \text{otherwise.} \end{cases} \tag{A.5}$$

Desorption rate des

$$des = \frac{1}{\tau}\ \theta^d\ \exp(-\frac{E_{de}}{K_b T}) \cdot \frac{N_{mono}}{\Delta N_{surf}} \cdot K_b T \tag{A.6}$$

Outgassing rate out

$$des = \frac{\text{Facet outgassing}}{K_b T} \tag{A.7}$$

$\mathrm{K}_{\mathrm{real/virtual}}$

$$\mathrm{K}_{\mathrm{real/virtual}} = \frac{\sum\limits_{\text{facets}} (out + des)}{\text{number of total desorbed molecules}} \tag{A.8}$$