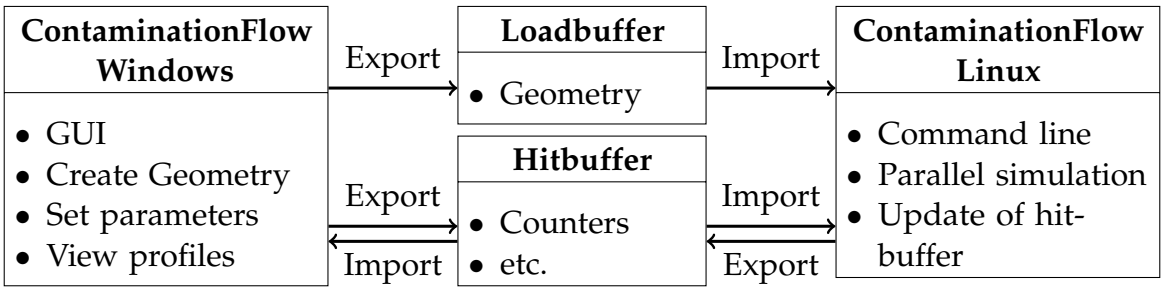
Documentation

ContaminationFlow on Linux and Windows

Hoai My Van

Contents

1. General Structure

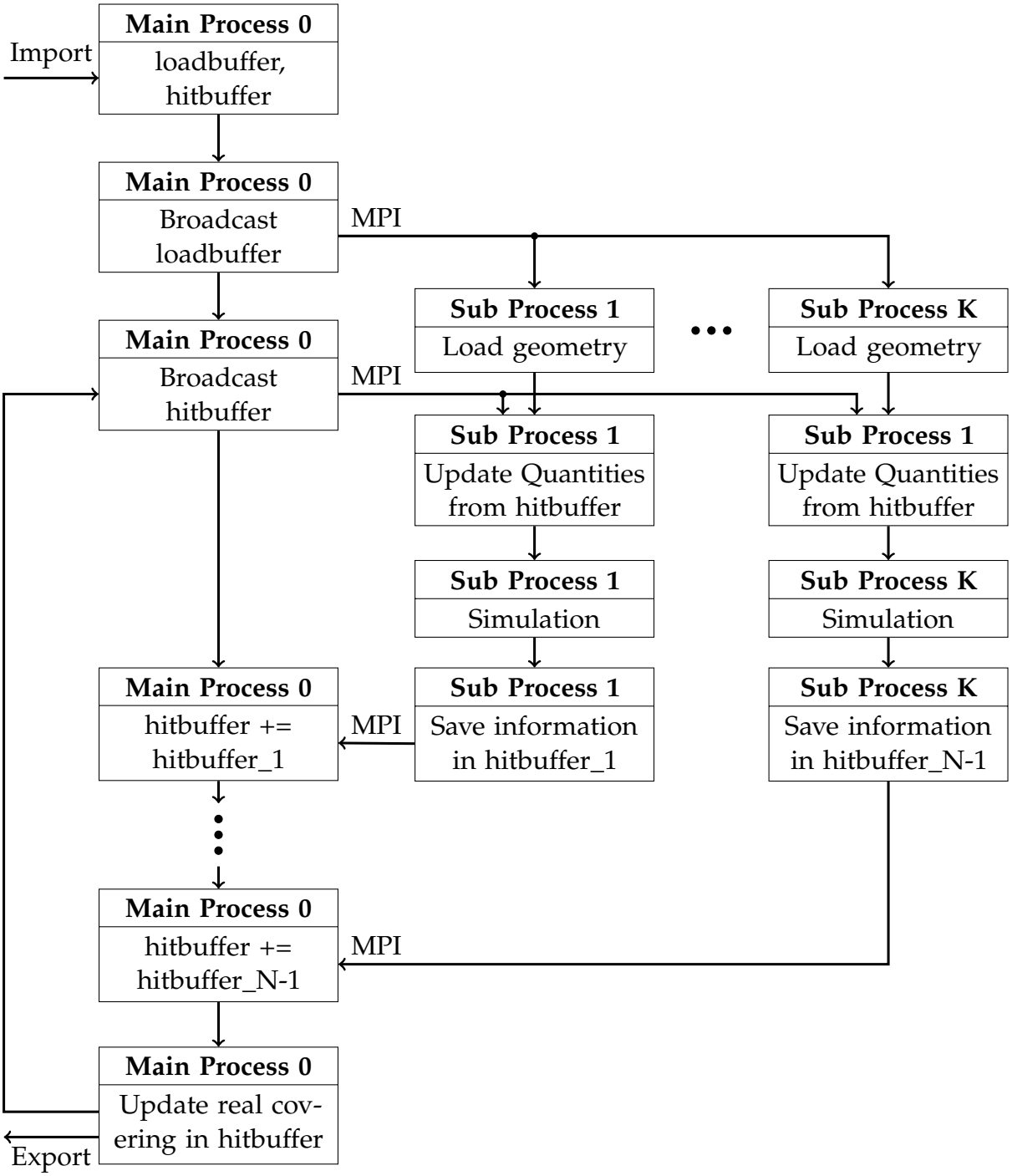


General structure for ContaminationFlow simulation

- 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 save simulationtime
unit
```

with the following 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`
- loadbuffer: path to loadbuffer file, contains geometry, e.g. `~/loadbuffer`
- hitbuffer: path to hitbuffer file, contains counters, etc., e.g. `~/hitbuffer`
- save: determines whether result directory is created (1: true, 0:false), default:1
- 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 save
```

with the following command line parameters:

- 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:

- 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, contains geometry, e.g. `~/loadbuffer`
- hitbufferPath: path to hitbuffer file, contains counters, etc., e.g. `~/hitbuffer`
- simulationTime: simulation time per iteration step; default: `10.0`
- unit: simulation time unit; default: `s`
- maxTime: maximum simulation time; default: `10.0`
- maxUnit: maximum simulation time unit; default: `y`
- iterationNumber: number of iterations; default: 43200
- d , power for base of coverage used for calculation of desorption; default: `1`
- E_{de} , maximum energy used for calculation of desorption; default: `1E-21`
- H_{vap} , minimum energy used for calculation of desorption; default: `1E-21`
- W_{tr} , window width used for calculation of desorption; default: `1E-21`
- $sticking$, constant sticking coefficient for all facets; default: `0.8E-19`

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

Result Directory

- Final resultbuffer
- Final covering, input file and console output as text files

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

- Ratio adsorbed particles to impinging particles
- Set to 0, can be adapted through input file

Binding energy

- Calculate energy using E_{de}, H_{vap} and W_{tr}
- Desorption computed from `Simulationcalc.cpp` file in `calcEnergy(·)`

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

- Calculated from energy, 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()`

$K_{real/virtual}$

- 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 desorption rate and sojourn frequency/energy in sub processes using `UpdateDesorptionRate(·)` and `UpdateSojourn(·)` 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;
  int currentStep;

  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

Calculate Step Size

- Uses `SimulationHistory::currentStep` to calculate logarithmic step size
- Duration between outgassing/desorption and adsorption

Management of Step Size

- Checks whether current step size would cause desorption to be larger than covering
- Adapts time step if needed
- Increments if step size not decreased
- Management in `UpdateMainProcess.cpp` file in `manageStepSize(·)`

Management of Simulation Time

- Increases simulation time (=computation time) per iteration if not enough particles desorbed
- Adapts simulation time if needed
- If enough desorption: compute maximum ratio *stepsize/computationtime*
- If not enough desorption: adapt computation time using ration *stepsize/computationtime*
- Management in `UpdateMainProcess.cpp` file in `manageSimulationTime(·)`

Error Calculation

- Calculates statistical error dependent on hits for every facet, weighted with opacity of that facet
- TODO weighting for total error

Sojourn

- Sojourn time of bounce calculated from energy and frequency

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

- "Particles out" renamed to Contamination level
  - Text field for covering
  - Text field for coverage
- 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

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

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

$$\begin{aligned} carbondiameter &= 2 \cdot 76\text{E} - 12 \\ K_b &= 1.38\text{E} - 23 \\ h &= 6.626\text{E} - 34 \end{aligned} \tag{A.1}$$

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

$$\Delta N_{surf} = \frac{\text{carbon equivalent gas mass}}{12.011} \tag{A.3}$$

Covering θ^*

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

Coverage θ

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

Energy E

$$E = \frac{E_{de} - H_{vap}}{2} \cdot \tanh\Big((1 - coverage) \cdot \frac{5.4}{W_{tr}}\Big) + \frac{E_{de} + H_{vap}}{2} \tag{A.6}$$

Sojourn

$$\begin{aligned} Frequency &= \frac{K_b T}{h} \\ Energy &= E \end{aligned} \tag{A.7}$$

Desorption rate des

$$\tau = \frac{h}{K_b T}$$

$$des = \begin{cases} \frac{1}{\tau}\ \theta^d\ \exp\left(-\frac{E}{K_b T}\right) \cdot \frac{N_{mono}}{\Delta N_{surf}} \cdot K_b T, & \text{if } \theta^* \geq 100 \\ 0, & \text{otherwise} \end{cases} \tag{A.8}$$

Outgassing rate out

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

$K_{real/virtual}$

$$K_{\text{real/virtual}} = \frac{\sum\limits_{\text{facets}} \left(out + \frac{des}{K_b T} \right)}{\text{number of total desorbed molecules}} \tag{A.10}$$

Step Size T_{step}

$$\begin{aligned} T_{min} &= 0.0001 \\ T_i &= T_{min} \cdot \exp\left(\text{i} \cdot \ln(\text{max. simulation time}/T_{min})/\text{max. \# of steps}) \right) \\ T_{step} &= T_{currentStep+1} - T_{currentStep} \end{aligned} \tag{A.11}$$

Error

$$error = \left(\frac{1}{\text{hits on facet}} \cdot \frac{1 - \text{hits on facet}}{\text{total hits}} \right)^{0.5} \tag{A.12}$$

B. Overview of new Classes and Functions

B.1. New Classes

SimulationHistory	
coveringList	of class HistoryList, stores covering history
numFacet	number of Facets
nbDesorbed_old	number of total desorbed molecules of previous iteration $\Rightarrow$ To calculate difference between consecutive iterations
flightTime	number of Facets
nParticles	Simulated flight time for iteration
numFacet	Simulated particles for iteration
lastTime	Total simulated time
currentStep	step of logarithmic time step calculation in `getStepSize()`
updateHistory()	Reset and update from hitbuffer
appendList()	Updates coveringList from hitbuffer
print()	Print to terminal
write()	Write to file

HistoryList	
pointintime_list	list containing history of time steps and respective facet values
currentList	list containing facet values at current step
reset()	Resets lists
initCurrent()	Initializes size of lists
print()	Print list as table to terminal, optional message
write()	Write to file
read()	Read from file
empty()	Checks if pointintime_list is empty
setCurrentList()	Set value of desired facet in currentList
getCurrent()	Get value of desired facet from currentList, or Get all facet values from last entry in pointintime_list

ProblemDef	
resultpath	Path of result folder
loadbufferPath	Path of loadbuffer file
hitbufferPath	Path of hitbuffer file
simulationTime, unit ⇒simulationTimeMS	Computation time of each iteration in milliseconds
maxTime, maxUnit ⇒maxTimeS	Maximal total simulated time in seconds
iterationNumber	Number of iterations
E_de, d	Parameters to calculate desorption rate, see equation A.8
readArg()	Initialization from command line arguments
readInputfile()	Initialization from input file
printInputfile()	Print to terminal
writeInputfile()	Write to file

B.2. New Functions

B.2.1. molflowlinux_main.cpp

Preprocessing	
parametercheck()	Checks input parameters from command line or input file
importBuff()	Import load- and hitbuffer to main process
MPI_Bcast()	Send loadbuffer to sub processes
LoadSimulation()	Load geometry from loadbuffer
initCoveringThresh()	Initialize covering threshold
simHistory=SimulationHistory()	Initialize simulation history

Simulation Loop	
initbufftozero()	Reset hitbuffer except covering
MPI_Bcast()	Send hitbuffer to sub processes
setCoveringThreshold()	Sets covering threshold for each facet
simulateSub()	Simulation on sub processes
MPI_Send(), MPI_Recv	Send sub hitbuffer, simHistory→flighttime and simHistory→nbParticles to main process
UpdateMCMainHits()	Add simulation results from sub hitbuffer to main hitbuffer
UpdateCovering()	Calculate and save new covering to simHistory
UpdateCoveringphys()	Saves current covering to hitbuffer
End simulation if maximum iteration number or simulation time is reached	

Postprocessing	
exportBuff()	Export final hitbuffer
simHistory→write()	Export simulation history

B.2.2. SimulationLinux.cpp

simulateSub()	
Input	hitbuffer, rank, simutime
Output	eos: indicates whether the simulation was ended early facetNum: facets that have reached their covering threshold
Pipeline	Update sticking and desorption rate from hitbuffer Simulate until simutime or covering threshold reached ⇒ Avoid covering getting negative Update hitbuffer from simulation

B.2.3. Calculations in SimulationCalc.cpp etc.

SimulationCalc.cpp	
getCovering()	Get covering from hitbuffer
getHits()	Get number of hits from hitbuffer
getnbDesorbed()	Get number of total desorbed molecules from hitbuffer
calcNmono()	see equation A.2
calcdNsurf()	see equation A.3
calcCoverage()	see equation A.7
calcEnergy()	see equation A.6
calcStickingnew()	sets sticking coefficient to p→sticking
calcDesorption(), calcDesorptionRate()	see equation A.8
GetMoleculesPerTP()	see equation A.10
calctotalDesorption	calculates desorption for `startFromSource()`
calcPressure(), calcParticleDensity()	has to be verified

worker.cpp	
CalcTotalOutgassingWorker()	see equation A.9, calculates desorption for `startFromSource()`

SimulationLinux.cpp	
covertunit()	Converts simutime*unit to milliseconds

B.2.4. UpdateSubProcess.cpp

Update sHandle paramters from hitbuffer	
UpdateSticking()	Updates sticking
UpdateDesorptionRate()	Updates desorption rate
UpdateSojourn()	Updates sojourn freqency and energy

Update hitbuffer	
initbufftozero()	Sets hitbuffer except covering to zero
UpdateMCSubHits()	Saves simulation results from sHandle into hitbuffer

B.2.5. UpdateMainProcess.cpp

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

Update real covering in hitbuffer	
getStepSize()	Calculates step size for current step, see equation A.11
manageStepSize()	Adapts step size if desorptionrate · step size $>$ than covering
UpdateCovering()	Uses time_step and Krealvirt to calculate new covering Saved to simHistory→coveringList
UpdateCoveringphys()	Saves current real covering to hitbuffer

B.2.6. Iteration.cpp

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