Originally develop at the IMDEA Materials Institute
Currently supported at github.com/imartinbragado/MMonCa

August 8, 2025

Contents

Chapter 1

Preliminaries

1.1 Citing

Please, if using MMonCa cite it as in Ref. **?**. If you use the LKMC module only, you can cite Ref. **?**.

1.2 Introduction

1.2.1 Installation from source code

1. Download and unpack the MMonCa source code from the repository at `https://github.com/imartinbragado/MMo`
2. Be sure you have a modern version of the g++ compiler installed in your system.
   - Type `make` to obtain some help on how the `Makefile` system works.
   - The make command expects one `machine` to be specified. The available machines are in the directory `sysmakes`.
   - Each `machine` is a specific configuration for a particular architecture. The `make` commands creates a binary directory called `Obj_machine` directory, with the `mmonca` binary inside.
3. The compilation system looks for several libraries. In an Ubuntu or Debian system these libraries can be installed using `apt-get`. The required libraries (for Ubuntu) are:

   compulsory `tcl-dev`.

   compulsory `libboost-dev`.

   compulsory `libboost-iostreams-dev`

4. Depending on the location of your workspace, add the following line to `.bashrc`: `export MCPATH=/home/`*`username`*`/workspace/MMonCa/config`

5. After compiling the code navigate to (cd command): /home/*username*/workspace/MMonCa/test, and enter runAll.sh in order to run the KMC tests and confirm a successful installation. The test suite expects a MMonCa binary in a directory called Release. You can copy or link a binary there for the test suite to work. By default, it uses the binary inside Obj_g++.

1.2.2 Submission Policy

When collaborating on a program, it is essential to preserve the functionality of existing code before adding your own. Therefore, after code is developed it is important to run all existing test to check that there are no failures. Also, new tests are required to be created for every new feature in the code.

The test-suite can be run with the command runAll.sh in the test directory as previously explained.

1.2.3 Installation from binary sources

- Untar and unzip the MMonCa binary with something like:

  tar -xvzf MMonCa-VERSION-bin.tar.gz

- Be sure you have the following libraries installed in your system:
  - libtcl
  - libblas
  - liblapack
  - libsuperlu
- Depending on the location of your workspace, add the following line to .bashrc: export MCPATH=/home/*username*/workspace/MMonCa/config.

1.3 Mechanical settings

There are two different modules to produce mechanical output (strain and stress) for MMonCa. These modules can be set up with the command:

```
1 param set type=string key=Mechanics/General/model  value=module
```

The different available modules are:

Uniform Simple model that allows to specify a fixed value in Mechanics/Uniform. The values are stress.xx, stress.yy and stress.zz.

None No model.

1.3.1 Testing MMonCa

MMonCa includes a test suite of several test cases. If you want to check the integrity of the distribution, or whether yuor compilation is correct, all tests should be run and should pass. Sometimes few tests might fail even when the distribution is correct if it was compiled in an architecture different than the original. The original architecture is `Ubuntu` LTS 14.04. In this cases, the failing cases are usually a little bit out of the random variation allowed in the original architecture.

All the tests are in the `test` subdirectory. The following scripts are available:

`runAll.sh` To run all the tests and display the results.

`runFailed.sh` To run only the failing tests and display all the results.

`collect.sh` To display the results only, no running of tests.

Chapter 2

Running MMonCa

2.1 MMonCa simulation geometry

MMonCa is a three-dimensional simulator. It simulates a cuboid area consisting of one or more materials, which can have defects or dopant particles in them. While the location of these particles within the simulation area is arbitrary, the materials are represented by a cuboid mesh, where each cell has a homogeneous material. The subdivision of this cuboid mesh can be uniform and non-uniform, but in either case, the mesh is defined by planes perpendicular to either axis (X, Y or Z) of the Cartesian coordinate system. The mesh can be described by either

- the extent of the whole cuboid area, so the minimum and maximum coordinates along each axis, together with the approximate subdivision steps along each axis. In this case, MMonCa automatically adjusts the given step values to yield equidistantly placed division planes.
- the list of positions of the planes along each axis. The extent of the whole cuboid area is implicitly given by the two extreme positions along each axis.

In the general case, the nonuniform mesh can look like Fig.2.1. Here, the subdivision along the Z axis is uniform, but nonuniform for the other two axes. Note, a mesh with a nonuniform subdivision along any axis counts as a nonuniform mesh. This nonuniform mesh can represent a curved interface between two materials more accurately with a lower total cell count than the original uniform version. A further advantage of nonuniform meshes is that they enable MMonCa to use material descriptions from other simulators.

Please refer to 7.4 for more details about the mesh definition.

2.2 Generalities

To run a MMonCa script just call the `mmonca` binary with the input file you want to execute as a parameter name.

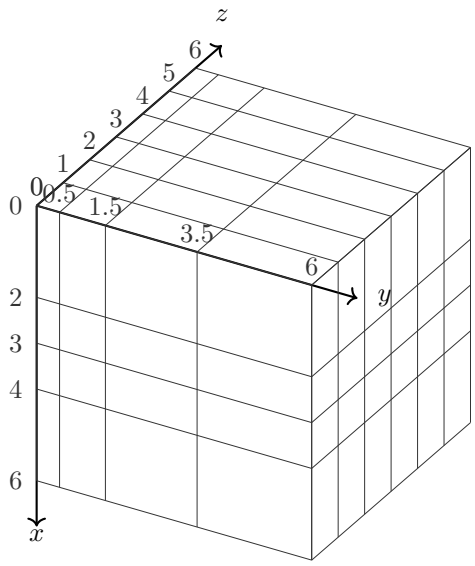

Figure 2.1: Nonuniform mesh example

```
1 mmonca input.mc
```

The input scripts are regular `TCL` scripts that include additional MMonCa commands. All regular TCL commands (to manipulate variables, loops, procedures, etc) should be available. For a list of the additional commands included in MMonCa allowing Monte Carlo simulations and post-processing of information, consult Chapter 7.

2.2.1 Starting from scratch

The following is a list of required and optional instructions that should be provided in a MMonCa script when starting from scracth:

required Telling MMonCa with materials to use by setting `MC/General/materials`

required Defining the distribution of materials in space with a procedure (`proc material` in this case)

optional Redefining some extra parameters with `param`.

required Initializing the simulation box with `init`.

optional Reading the initial damage to anneal with `cascade`.

required Letting the system evolve in time with the `anneal`.

optional Obtaining some useful information with `extract`.

optional Saving some information for visualization purposes with `save`.

optional Saving the simulation status for restarting with `restart`.

2.2.2 Restarting

An alternative is to start MMonCa from a file saved with the `restart` option. In this case, the following is the list of required and optional instructions:

required Using the `restart` option as the first command.

optional Redefining some extra parameters with `param`.

optional Reading the initial damage to anneal with `cascade`.

optional Letting the system evolve in time with the `anneal`.

optional Obtaining some useful information with `extract`.

optional Saving some information for visualization purposes with `save`.

optional Saving the simulation status for further restarting with `restart`.

2.3 Object Kinetic Monte Carlo simulation: an example

The following is an example of a MMonCa input script, taken from Ref. **?**, that performs the isochronal annealing of α-Fe. The goal is to simulate the evolution of defects and resistivity recovery during isochronal annealing of high-purity electron-irradiated iron. The script contains both the commands needed to set the simulation conditions, and the commands required to do some post-processing of information, mainly extracting the evolution of defects with time to an external file. There are also typical TCL commands that help in the extraction.

The initial damage in this example is read from an `electron.cascade` file containing the IV pairs produced by electron irradiation.

This example can be found in the examples directory.

2.3.1 The script

```
1  param set type=map<string,string>   key=MC/General/materials
       value="S_Iron␣Fe"
2  set size 143.5
3  set time 300
4
5  proc material { x y z } { return "S_Iron" }
6
7  #parameters
8  param set type=bool   key=MC/Mesh/periodic.x value=true
9
10 param set type=arrhenius key=S_Iron/Vacancy/V(migration)   value="5e-5␣
       0.67"
11 param set type=arrhenius key=S_Iron/Iron/I(migration)      value="3.2e-3␣
       0.34"
12
```

```
13 init minx=0 miny=0 minz=0 maxx=$size maxy=$size maxz=$size
       material=material
14
15 cascade file=electron.cascade format=B:C*.287:D*.287:E*.287 periodic
       flux=4.8562e9 do.not.react
16
17 set X 0
18 set a 77.2
19 set b 1.030927
20
21 set FILE [open "results.txt" w]
22 close $FILE
23
24 set oldTotal 0
25 set oldC 0
26
27 save ovito=evolution
28 while { $X < 65 } {
29       incr X
30       set c [expr $a*$b]
31       lowmsg "Running␣for␣$c␣K"
32       set FILE [open "results.txt" a]
33       anneal time=$time temp=[expr $c - 273.15]
34       save ovito=evolution append
35       set total [extract count.particles]
36       set deriv [expr ($total - $oldTotal)/($c - $oldC)]
37       set I     [extract count.particles particle=I
       defect=MobileParticle]
38       set V     [extract count.particles particle=V
       defect=MobileParticle]
39       set I2    [extract count.particles defect=ICluster ID=I2]
40       set V2    [extract count.particles defect=VCluster ID=V2]
41       set I3    [extract count.particles defect=ICluster ID=I3]
42       set V3    [extract count.particles defect=VCluster ID=V3]
43       set I111  [extract count.particles defect=<111>]
44       set oI    [expr [extract count.particles defect=ICluster] -$I2
       -$I3]
45       set oV    [expr [extract count.particles defect=VCluster] -$V2
       -$V3]
46       puts $FILE "$c␣$total␣$deriv␣$I␣$V␣$I2␣$V2␣$I3␣$V3␣$oI␣$oV␣$I111"
47       lowmsg     "$c␣$total␣$deriv␣$I␣$V␣$I2␣$V2␣$I3␣$V3␣$oI␣$oV␣$I111"
48       close $FILE
49       set a $c
50       set oldC $c
51       set oldTotal $total
52 }
```

2.3.2 The script, dissected

```
1 param set type=map<string,string>   key=MC/General/materials
      value="S_Iron␣Fe"
```

Line needed to define the materials to be used in the simulation.

```
2 set size 143.5
3 set time 300
```

TCL variables defined to easily change the simulation size and the annealing times between steps.

```
5 proc material { x y z } { return "S_Iron" }
```

One of the two possibilities to define the material in the simulation. In this case, we use a Tcl function defining a block containing only Iron. We also have the possibility to define the material and the mesh in a JSON file, see **??**.

```
7 #parameters
8 param set type=bool   key=MC/Mesh/periodic.x value=true
9 param set type=arrhenius key=S_Iron/Vacancy/V(migration)   value="5e-5␣
      0.67"
10 param set type=arrhenius key=S_Iron/Iron/I(migration)      value="3.2e-3␣
      0.34"
```

Some redefinition of parameters: setting of periodic boundary conditions in x (false by default, because there are usually free surfaces in this axis), and small changes in point defect migration parameters.

```
13 init minx=0 miny=0 minz=0 maxx=$size maxy=$size maxz=$size
       material=material
```

Required initialization of the simulator, specifying the simulation sizes and the script containing the material definition. In this case, we are using the TCL variables, previously defined, `size`.

```
15 cascade file=electron.cascade format=B:C*.287:D*.287:E*.287 periodic
       flux=4.8562e9 do.not.react
```

Initial conditions for the simulation: insertion of damage. This command is not strictly needed, but is very common, because damage has to be introduced to simulate its evolution, unless equilibrium concentrations want to be set.

```
17 set X 0
18 set a 77.2
19 set b 1.030927
```

Definition of some convenient TCL variables to later compute the temperature ramps.

```
21 set FILE [open "results.txt" w]
22 close $FILE
```

This confusing commands open a file to close it right away. The goal is to overwrite previous existing files with an empty file. Later, information will be appended to the file.

```
24 set oldTotal 0
25 set oldC 0
```

Definition of TCL variables to compute the first derivative of the total damage.

```
27 save lammps=evolution
```

Command to create a initial file with the atomistic information, using the `lammps` format that can be later used by `ovito` to visualize the information. More save commands will be issued later to append new time frames into the file.

```
28 while { $X < 65 } {
29       incr X
30       set c [expr $a*$b]
31       lowmsg "Running␣for␣$c␣K"
```

Initialization of the loop to increase the temperature, and operations to compute the next temperature for the annealing.

```
32       set FILE [open "results.txt" a]
```

TCL command to open an (existing) file and append information to it.

```
33       anneal time=$time temp=[expr $c - 273.15]
```

The fundamental `anneal` command, needed to start the annealing of defects in time. Both the time and temperature need to be specified, in this case, using TCL variables defined previously. Since the temperature in "c" is in Kelvin, but the anneal command requires the temperature to be written in Celsius, a translation is defined in line.

```
34       save lammps=evolution append
```

Command to append one more snapshot for visualization

```
35       set total [extract count.particles]
36       set deriv [expr ($total - $oldTotal)/($c - $oldC)]
37       set I     [extract count.particles particle=I
    defect=MobileParticle]
38       set V     [extract count.particles particle=V
    defect=MobileParticle]
39       set I2    [extract count.particles defect=ICluster ID=I2]
40       set V2    [extract count.particles defect=VCluster ID=V2]
41       set I3    [extract count.particles defect=ICluster ID=I3]
42       set V3    [extract count.particles defect=VCluster ID=V3]
43       set I111  [extract count.particles defect=<111>]
44       set oI    [expr [extract count.particles defect=ICluster] -$I2
    -$I3]
45       set oV    [expr [extract count.particles defect=VCluster] -$V2
    -$V3]
46       puts $FILE "$c␣$total␣$deriv␣$I␣$V␣$I2␣$V2␣$I3␣$V3␣$oI␣$oV␣$I111"
47       lowmsg     "$c␣$total␣$deriv␣$I␣$V␣$I2␣$V2␣$I3␣$V3␣$oI␣$oV␣$I111"
48       close $FILE
```

Post-processing of the information after each annealing. The number of particles for different defect types is asked, and stored in TCL variables. These variables are used to add one line to the previously opened file, with the different values, then allowing to have a file with different columns, and the number of particles of different defects in each column. The derivative of the number of defects is also computed.

```
49      set a $c
50      set oldC $c
51      set oldTotal $total
52  }
```

Variable operations to update the temperature ramp, and to make possible the computation of the first derivative. End of the script.

2.3.3 The output

The script creates a list named `results.txt` that looks like:

```
79.58756439999999 59955 753.3212060450993 29962 29993 0 0 0 0 0 0 0
82.04896900419878 59955 0.0 29962 29993 0 0 0 0 0 0 0
84.58649746859163 59953 -0.7881684986255068 29961 29992 0 0 0 0 0 0 0
87.20250407580276 59937 -6.116192503449846 29953 29984 0 0 0 0 0 0 0
89.8994159193551 59917 -7.415889417303392 29943 29974 0 0 0 0 0 0 0
92.67973515549299 59813 -37.40577651955731 29891 29922 0 0 0 0 0 0 0
95.54604132464692 59453 -125.59718981669849 29711 29742 0 0 0 0 0 0 0
98.50099374469427 58089 -461.5979569573388 29029 29060 0 0 0 0 0 0 0
101.54733397823642 54015 -1337.342413412217 26978 27023 14 0 0 0 0 0 0
104.68788837618133 43219 -3437.6096166538605 21560 21625 34 0 0 0 0 0 0
107.92557069999148 23693 -6030.857276022542 11749 11862 82 0 0 0 0 0 0
111.26338482503012 9445 -4268.661904543617 4601 4738 106 0 0 0 0 0 0
114.70442752751381 6359 -896.8211867212708 3060 3195 104 0 0 0 0 0 0
118.25189135765723 5303 -297.6774536859221 2516 2667 120 0 0 0 0 0 0
121.90906760167549 4757 -149.2955120478667 2229 2394 134 0 0 0 0 0 0
125.6793493353925 4399 -94.95311631448243 2015 2215 166 0 3 0 0 0 0
129.56623457228818 4087 -80.26992848628163 1811 2059 214 0 3 0 0 0 0
133.57332950890532 3801 -71.3734025581751 1493 1916 374 0 18 0 0 0 0
137.70435187062722 3439 -87.62963942153782 1100 1735 500 0 96 0 8 0 0
141.9631343609301 2967 -110.82979726593929 611 1499 604 0 213 0 40 0 0
146.3536282173106 2535 -98.39439801793499 229 1283 604 0 303 0 108 0 8
150.87990687718735 2327 -45.9538653339702 44 1179 578 0 327 0 184 0 15
```

Fig. 2.2 shows the results processed with the `gnuplot` tool.

2.4 Lattice Kinetic Monte Carlo simulation: an example

In this section we will show an input script that, using the Lattice Kinetic Monte Carlo model, simulates the Si(100), Si(011) and Si(111) Solid Phase epitaxial regrowth of a partially amorphized sample, and computes its recrystallization speed and roughness. These scripts used the models published in Refs. **????**, and are partially based in the scripts distributed under the folder `tests`, sub-folders `standard/lkmc`.

2.4.1 The script

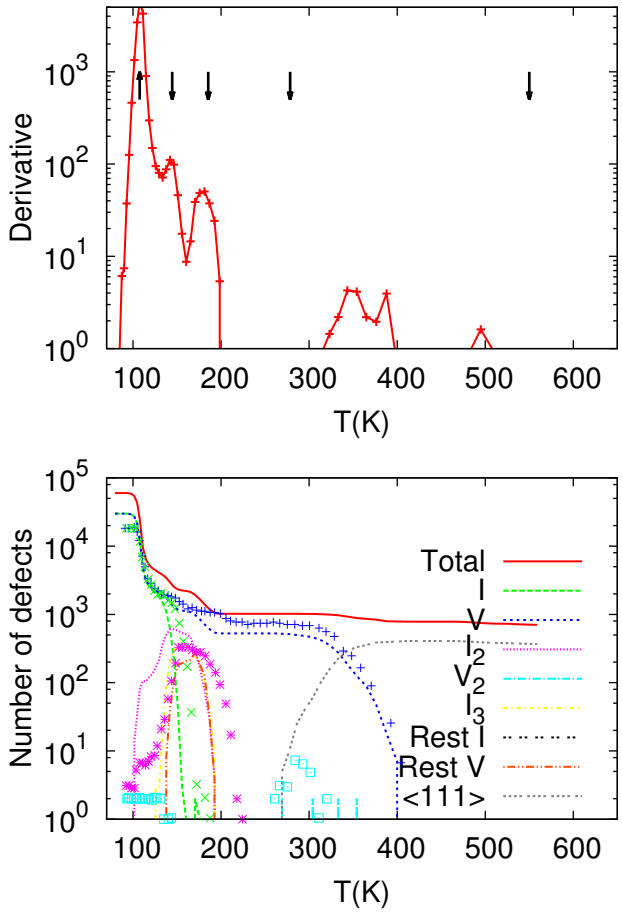

Figure 2.2: Isochronal annealing of Iron. **?**

```
param set type=map<string,string>   key=MC/General/materials
    value="Silicon␣Si␣AmorphousSilicon␣aSi␣Gas␣Gas"

set T 550
set name SPER
#0 9 or 14
set i 0
proc material { x y z } {
        set res "Unknown"
        if { $x < 0 } {
                set res "Gas"
        } elseif { $x < 52 } {
                set res "AmorphousSilicon"
        } else {
                set res "Silicon"
        }
        return $res
}

set angle(0)  00
set angle(9)  55
set angle(14) 90

set sizeZ    [expr sqrt(2.)*.5431*26]
set sizeY    [expr 180]

param set type=bool key=MC/Mesh/periodic.y value=false
param set type=bool key=MC/Mesh/periodic.z value=true

set radians "$angle($i).0*2.0*3.1415926535897931/360.0"
set S [expr sin($radians)]
set C [expr cos($radians)]
set R [expr sqrt(2.0)]
set waferorient "i␣$C␣␣j␣[expr␣$S/$R]␣k␣[expr␣$S/$R]"
set flatorient  "i␣-$S␣j␣[expr␣$C/$R]␣k␣[expr␣$C/$R]"

param set type=map<string,float> key=Silicon/Lattice/wafer.orientation
    value="$waferorient"
param set type=map<string,float> key=Silicon/Lattice/flat.orientation
    value="$flatorient"
param set type=map<string,float>
    key=AmorphousSilicon/Lattice/wafer.orientation value="$waferorient"
param set type=map<string,float>
    key=AmorphousSilicon/Lattice/flat.orientation  value="$flatorient"

init minx=-2 miny=0 minz=0 maxx=54 maxy=$sizeY maxz=$sizeZ
    material=material

anneal time=1 temp=$T depth=51
set orig_time [extract time]
set orig_depth [lindex [extract ac.mean min.y=60 max.y=120 min.z=0
    max.z=$sizeZ] 0]
lowmsg "Original␣depth␣is␣$orig_depth"
```

```
47
48 anneal time=1 temp=$T depth=31
49 set end_time [extract time]
50 set end_depth [lindex [extract ac.mean min.y=60 max.y=120 min.z=0
       max.z=$sizeZ] 0]
51 lowmsg "$angle($i) - Final depth is $end_depth"
52 set velocity [expr ($orig_depth - $end_depth)/($end_time-$orig_time)*60]
53 set roughnes [extract ac.stdev]
54 lowmsg "Velocity  for angle $angle($i) is $velocity in nm/min."
55 lowmsg "Roughness for angle $angle($i) is $roughnes in nm."
```

2.4.2 The script, dissected

This script adds the complication that it has been done to accept different substrate orientation with minimum changes in the script. This complication is managed using a TCL array called `angle`, and used to compute the substrate orientation automatically. Also, since having periodic boundary conditions is not possible for all possible substrate orientations, the script defines a quite large y domain (180 nm) without boundary conditions, but measures the recrystallization velocity only in the middle of the domain, where it is expected to be flat.

```
1 param set type=map<string,string>   key=MC/General/materials
      value="Silicon Si AmorphousSilicon aSi Gas Gas"
```

Definition of the materials used.

```
3 set T 550
4 set name SPER
```

Definition of temperature and script name, to be used later

```
5 #0 9 or 14
6 set i 0
```

The substrate angle is codified in this integer.

```
7  proc material { x y z } {
8          if { $x < 0 } {
9                  set res "Gas"
10         } elseif { $x < 52 } {
11                 set res "AmorphousSilicon"
12         } else {
13                 set res "Silicon"
14         }
15         return $res
16 }
```

Definition of the materials. Three sections are defined, "Gas", for $x < 0$, "AmorphousSilicon" for $0 < x < 52$ and "Silicon" (i.e., crystalline silicon) for $x \geq 52$.

```
19 set angle(0)  00
20 set angle(9)  55
21 set angle(14) 90
```

Definition of an array indexed by integers with the possible angles. 0 for Si(100), 9 for Si(111) and 14 for Si(011). This infrastructure allows the definition of intermediate angles for the other is.

```
23 set sizeZ    [expr sqrt(2.)*.5431*26]
24 set sizeY    [expr 180]
```

Definition of the size as a variable. y should be big enough to have a flat region without periodic boundary conditions, but especial dimensions are set for z to have periodicity.

```
26 param set type=bool key=MC/Mesh/periodic.y value=false
27 param set type=bool key=MC/Mesh/periodic.z value=true
```

Boundary conditions for z only. By default x has no PBC.

```
29 set radians "$angle($i).0*2.0*3.1415926535897931/360.0"
30 set S [expr sin($radians)]
31 set C [expr cos($radians)]
32 set R [expr sqrt(2.0)]
33 set waferorient "i␣$C␣␣j␣[expr␣$S/$R]␣k␣[expr␣$S/$R]"
34 set flatorient  "i␣-$S␣j␣[expr␣$C/$R]␣k␣[expr␣$C/$R]"
```

Example of use of the TCL language embedded in the MonCa scripting to compute the substrate orientation.

```
36 param set type=map<string,float> key=Silicon/Lattice/wafer.orientation
       value="$waferorient"
37 param set type=map<string,float> key=Silicon/Lattice/flat.orientation
       value="$flatorient"
38 param set type=map<string,float>
       key=AmorphousSilicon/Lattice/wafer.orientation value="$waferorient"
39 param set type=map<string,float>
       key=AmorphousSilicon/Lattice/flat.orientation  value="$flatorient"
```

Definition of the substrate orientation as a parameter, once computed previously.

```
41 init minx=-2 miny=0 minz=0 maxx=54 maxy=$sizeY maxz=$sizeZ
       material=material
```

Compulsory definition of the initial simulation box and initial materials.

```
43 anneal time=1 temp=$T depth=51
```

Since the amorphous/crystalline (A/C) interface is artificially flat plane (atomically flat) an small initial annealing (recrystallization) is issued to recrystallize a maximum of 1 nm, creating an starting condition where the A/C interface is more realistic.

```
44 set orig_time [extract time]
45 set orig_depth [lindex [extract ac.mean min.y=60 max.y=120 min.z=0
       max.z=$sizeZ] 0]
46 lowmsg "Original␣depth␣is␣$orig_depth"
```

Use of TCL to store the initial position of the interface.

```
48 anneal time=1 temp=$T depth=31
```

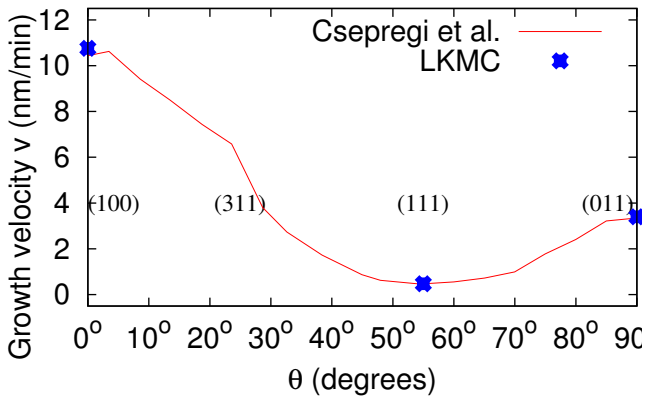

Figure 2.3: Solid Phase Epitaxial Regrowth of amorphized silicon. **?**

Main recrystallization: From depth 51 to depth 31, a maximum of 20 nm is requested.

```
49 set end_time [extract time]
50 set end_depth [lindex [extract ac.mean min.y=60 max.y=120 min.z=0
       max.z=$sizeZ] 0]
51 lowmsg "$angle($i) - Final depth is $end_depth"
52 set velocity [expr ($orig_depth - $end_depth)/($end_time-$orig_time)*60]
53 set roughnes [extract ac.stdev]
```

The annealing command stops when the A/C touches the requested depth, but here we want to compute the recrystallization speed using average values, not maximum ones. Thus, the average depth is computed for $60 < y < 120$ and stored. The velocity is then computed using the regular $v = \frac{\Delta x}{\Delta t}$ equation. The roughness is extracted using a specific MMonCa command.

```
54 lowmsg "Velocity  for angle $angle($i) is $velocity in nm/min."
55 lowmsg "Roughness for angle $angle($i) is $roughnes in nm."
```

Finally, the values are displayed.

2.4.3 Simulation results

Fig. 2.3 shows the results for Si(100), Si(011) and Si(111) simulations compared to experimental results taken from Ref. **?**

2.5 Parallelism

2.5.1 Standard

"Poor man parallelization" is implemented. It can be used with the parameter: `/MC/General/domains` set to the number of threads. Default is 1.

This type of parallelization is done by simple splitting of the z axis, as can be seen in Fig. 2.4 and running in a totally independent way such splits in different

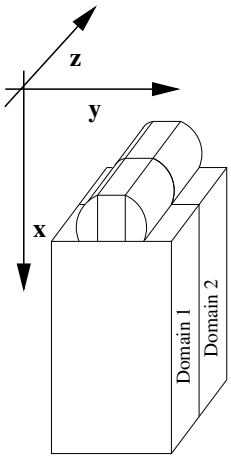

Figure 2.4: Simple parallelization splits the z domain in independent sub-domains and runs them isolated from each other. All input and output of information is transparent to the user.

threads, with no communication at all between them during the annealing. Insertion and extraction of information is, nevertheless, transparent to the user, meaning that the user input (cascade, profile, anneal) and output (extract) commands will work as if there is only 1 domain.

This parallelization is optimal when a better statistics is needed and there is 1D or 2D symmetry in the system. Special care has to be taken to be be sure that each domain is big enough to represent the physics of the system simulated. For example, if a system requires at least 20 nm in the z axis, a simulation with at least $20 \times n$, being n the number of threads, needs to be run, because each thread will run a simulation with $20 \times n/n = 20$ nm in the z axis.

Parallel execution is also possible using the new nonuniform mesh. The simulation area is divided along the Z axis, but only at the division planes. This division happens possibly in a uniform way. Should the linesz resolution be too coarse somewhere, a domain may end up with 0 Z-size. This happens for example when one sets 16 domains but has only 6 division planes along the Z axis.. In this case, MMonCa exits with an error message.

Note, however, nonuniform meshes don't support more than 1 subdomains.

Alternative ways of parallelization are being currently studied. If you want to have more information ask `ignacio.martin@imdea.org`.

2.5.2 Experimental

Experimental parallelization based on the work of **?** has been implemented as explained in **?**. As explained in the cited paper, this parallelization works but should be used with extreme caution to avoid very slow simulations.

Chapter 3

Syntax

3.1 Materials

MonCa assumes that the space is divided in `materials`. A material can be a unary material (i.e. Silicon, Iron), or a binary material (i.e., SiC, GaAs). All materials are defined as directories in the `config` MonCa folder. For MonCa to access one of such folders, the material has to be named in `MC/General/materials`. Materials are defined with two strings, the first one for the real (long) name, and the second one for a small (short) name.

```
1 map<string,string> materials \
2 Silicon Si AmorphousSilicon aSi SiO2 SiO2 Nitride Ni \
3 Iron Fe \
4 Gas Gas
```

Inside each material folder, the `Models` file defines the most important material properties. In particular, the `material.composition`. This string links the material with the elements. For a unary material, it is defined as `string material.composition Fe`, while for a binary material as `string material.composition Si,C`. The migration distance for particles in every material together and the default capture radius is defined as the `lambda` parameter inside the `Models` file.

3.2 Particles

Mobile particles and Cluster defects are made of particles. A particle is defined as an `element` and a `position`. For instance, the silicon interstitial is a Si in an interstitial position, where the element is silicon and the position is interstitial. Different authors use different notations for the same defects. In our example, the silicon interstitial could be just I, or Si_i, SiI, etc...

The notation chosen in MonCa tries to, in the one hand, be accurate, but on the other hand, be simple enough. To obtain such "equilibrium", two different notations are used, one for unary materials, or materials like Si, Fe, etc. where there is just one type of lattice site, and another one for binary materials, like SiC,

where it is important to distinguish between an impurity in a C site, or in a Si site, and between a Si interstitial and a C interstitial.

3.2.1 Syntax for particles in unary materials

Interstitials and vacancies are just called "I" and "V", while impurities (for instance, C in Fe) are called C, CI, or CV according to their positions. See the following sections for more information.

3.2.2 Syntax for particles in binary materials

"I" or "V" are ambiguous in a binary material like SiC. Consequently, the particular type has to be specified. Valid notations would be SiI and CI for the Si and C self interstitial respectively. Similarly, a vacancy can be in a C position (VC) or in a Si position (VSi). The same notation applies to substitutional impurities. Being A a generic impurity, it can be in a C (AC) or Si (ASi) position, as well as being interstitial (AI) or paired with a vacancy (AV).

3.2.3 Elements

The elements are defined in `MC/Particles/elements`. They are defined with their element name plus a string. The string contains a comma-separated array of properties: the full name, the atomic number and the atomic weight.

```
1  // name, element, mass (a.m.u.)
2  map<string,string> elements {
3      As Arsenic,33,74.9216
4      B  Boron,5,10.81
5      C  Carbon,6,12.011
6      Cr Chromium,24,55.9961
7      Fe Iron,26,55.845
8      Ge Germanium,32,74.9216
9      He Helium,2,4.002602
10     Si Silicon,14,28.085
11 }
```

Added to this element list, there is an "especial element" definition: vacancy. A vacancy is the lack of an element.

3.2.4 Positions

MonCa defines the following positions:

First material An element can be in the substitutional position of the first material.

Second material An element can be in the substitutional position of the second material. This position is not defined for unary materials, only for binary materials.

Interstitial A particle can be in an interstitial position.

Table 3.1: Combinations for a unary material: Iron

NO	0 (Fe)	1	I	V
V	V			
Fe			I	
C	C		CI	CI
He	He		HeI	HeV

Table 3.2: Combinations for a binary material: Si,C

NO	0 (Si)	1 (C)	I	V
V	VSi	VC		
Si		SiC	SiI	
C	CSi		CI	
He	HeSi	HeC	HeI	HeV

Vacancy A particle can be paired with a vacancy. This is not strictly speaking a position, but it is a very useful definition for paired defects and for backward compatibility with MonCa and other KMC codes.

No position For particles in clusters or at interfaces it is useful to define the element without a particular position.

3.2.5 Valid combinations

Not all the combinations of positions and elements are valid for each material. For instance, table 3.1 shows valid possibilities for the unary material Iron, and table 3.2 for the binary material SiC.

Interestingly, some combinations might be possible in some materials while not in others. For instance, Si in the position 0 is valid for Iron, but not for SiC.

3.2.6 Particle syntax

Unary materials

Self interstitials and vacancies are specified simply as I and V. Interstitial impurities are specified with a I suffix. Vacancy-paired impurities have a V suffix. Impurities in a substitutional position can be specified with just the impurity name, although such notation is also valid for interstitial impurities, as long as all the notation in the material is consistent. An example can be seen in table 3.3

Binary materials

Particles are specified with the element and the position. For instance, FeI, VFe, SiI, SiC, etc... This notation allows to specify self-interstitials and vacancies (VFe, VSi, SiI, CI), antisites (CSi, SiC), impurities in substitutional position (BSi, HeFe),

Table 3.3: Alternative notations for unary materials

Type	Notation 1 (μelectronics)	Notation 2 (Energy)
Self interstitial	I	I
Self vacancy	V	V
Impurity in or near lattice position	A	AV
Interstitial impurity	AI	A
2 interstitial impurities	A2I2	A2
1 lattice, 1 interstitial	A2I	A2V

impurities in interstitial position (HeI) and impurities paired with vacancies (HeV). Impurities paired with vacancies assume that both positions are taken, i.e., for HeV in SiC, a Si and a C position are taken.

3.3 Clusters

3.3.1 Unary materials

The notation of cluster for unary materials follows the standard criteria of specifying the impurities together with "I" or "V". This way, a cluster with 3 self interstitials is just I3. A cluster that contains one impurity atom A and two interstitial atoms (regardless of whether they are self interstitials or one of them is the impurity) is AI2.

Amorphous pockets can be specified as InVm, where n and m are the number of self "I" and "V". It is noteworthy that internally the clusters are represented using the full notation, i.e., the simulator represents them internally as n material atoms in an interstitial position and m vacancies of the material.

"I" atoms do not actually exist in the simulation. For a material M they are actually MI. When specifying "I", MonCa translates it automatically to MI. This can produced some peculiar outputs that need to be understood. Let's imagine we have a simulation with 2 "I"s and 1 BI3. The command

```
1 extract count.particles name=I
```

will return 2, because there are just two MI. ¿Where are the other "I"s in the cluster BI3?. They are not stored like MI, but rather as a cluster of the type M2B, with three interstitial positions (a M2B^I3 in the binary notation). To obtain the total 5 interstitials, a

```
1 extract count.positions position=I
```

has to be issued.

3.3.2 Binary materials

Since clusters are agglomeration of particles, they are defined by a set of elements and positions. In the case of clusters, the symbol "^" is used to separate the

elements from the positions. For instance, a cluster of self interstitials in Iron is `Fe3^I3`. The self interstitials might be from different materials, like `Si2C^I3`.

Such a notation can be abbreviated, and the code accepts such abbreviations and uses them in its output. For instance in SiC, a notation like `Si2^I3` implies that the full notation would be `Si2C^I3`.

For clusters with possible recombinations (amorphous pockets or similar), some abbreviations are also possible. Something like `V^I` is possible instead of the full notation of `SiV^ISi`. The user is free to use one notation or the other, but she has to be consistent with the one used and used it everywhere in the parameters and in the input. Some differences might be noted, though. Although conceptually similar, `V^I` will actually be a cluster with only 1 particle, while `SiV^ISi` will contain 2. Nevertheless, both of them will have only 1 recombination event to disappear.

Chapter 4

Object KMC: defects and particles

4.1 Defects

Figure 4.1 shows the generic defect evolution for OKMC simulations in MMonCa. It starts from point defects, either intrinsic, like interstitials or vacancies, or extrinsic as impurities or dopants. This defects can react between them producing other defective objects. This objects are:

MobileParticle A single particle (point) defect. Migration, break-up and FT emission are possible.

Cluster Anything that is not a mobile particle. It can be extended defects (I_8), clusters (He_4V_5), combinations (I_8CP_3), amorphous pocktes (I_8V_4) etc. They can migrate, transform from to other Clusters, trap and re-emit mobile particles, recombine IV pairs and trap other multiclusters. They can have particular shapes.

Interfaces Interfaces are not formed by particles, although they can contain particles. They are 2D surfaces that separate different materials. They can trap, emit and annihilate particles, and can annihilate clusters. Particle diffusion on the interfaces is also possible.

4.2 Definition of defects

The defects being used in the simulation are defined in the `Models` file for each material. Description of the important definitions there follow.

4.2.1 particles

Defines the `particles` to be used in the material. For instance:

```
1 map<string,bool> particles {
2    HeI true
3    He  true
```

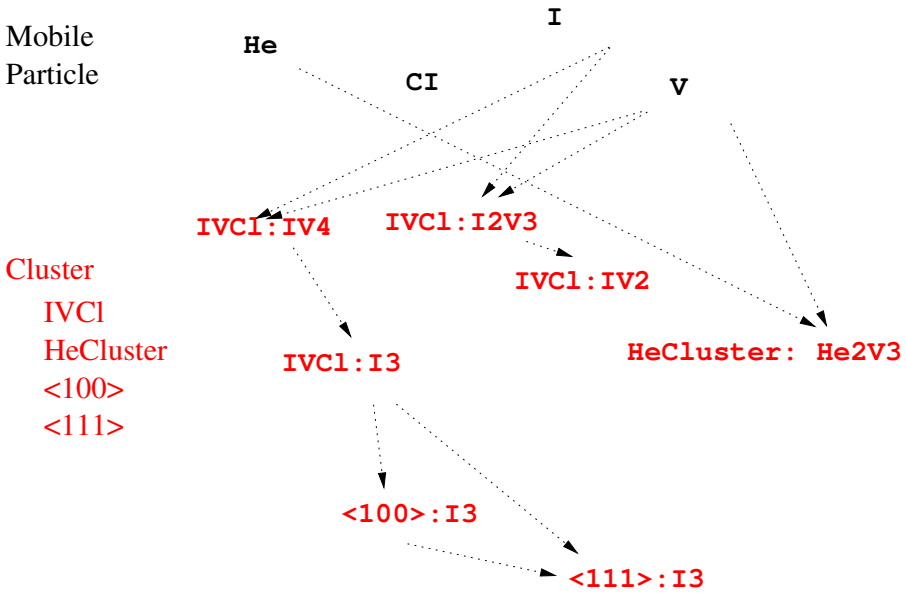

Figure 4.1: Different generic defects and their evolution

```
4     HeV true
5     I   true
6     V   true
7     C   true
8     CI  true
9     CV  true
10 }
```

4.2.2 defined

Defines wich generic defects will be used (true/false). There are no reserved words. Any name will be understood as an implemented cluster. For instance, `ICluster`, `Void`, `BICs`, `IVCluster`, etc.

An example is

```
1  map<string,bool> defined {
2     IVCluster true
3     ICluster  true
4     VCluster  true
5     BICs      true
6  }
```

4.2.3 interactions

The parameter `interactions` is used to:

- Specify whether the interaction between particular defects is allowed or disabled.

- Specify, optionally, the capture radius for a particular interaction. The interaction will happen when the distance between two particles belonging to each defect is smaller or equal than the specified capture radius. If not specified, the default value of `lambda` is used.

This parameter requires a list of all possible interactions. Wildcards are allowed for interactions involving `Clusters`. The result of interactions allow an optional capture radius for the particular interaction listed. All interactions have the following format:

```
1 Reactant1+Reactant2  0/true/false/result[,probability][,capture radius]
```

The logic for the reaction syntax is the following:

- Reactions between mobile particles giving a particle are defined con true/false.
- Reactions between clusters, are defined with true/false.
- Reactions between particles, producing a cluster, are defined with the name of the cluster and the probability to follow that path.
- In the particular case of reactions between mobile particles producing a recombination, 0 can be specified indicating instant recombination.
- Reaction between a defect and an interface are specified with true/false.
- For reactions between particles, particles and clusters, or clusters (i.e., all except interfaces) an optional capture radius can be specified.

Reactants are the inputs for the interactions.

The procedure to define interactions, assuming that M is the name of a unary material, is defined in Table 4.1.

All the possible interactions, and their capture radius, that can be defined are summed up in Table 4.2. If a capture radius is not specified it is taken as the default value of `lambda` for the material.

The interactions are:

- Mobile particles interact with mobile particles.
- If a Cluster is formed, the multicluster name and the probability is written (i.e., <111>,1).
- If there is a recombination, either 0 or Cluster is written.
- For MobileParticles, true is written.
- For interfaces, the interface is written as second reactant: e.g. MI+Gas, <111>+Gas. They interact with everything.

4.3 Description of defects and parameters

4.3.1 `MobileParticle`

MobileParticles are single point defects, i.e., they are represented with three coordinates only. They can be either self point defects (interstitials or vacancies) or

Table 4.1: Reactions defined in Models/interactions. M represents the material for binary materials, being empty for unary ones. A represents an "impurity", an element not constituting the material.

Generic	Reactant 1	Reactant 2	result	example	comments
MP+MP	MI	MV	0 or MC,prob	MI+VM IVCluster,1	0 means annihilation
MP+MP	MI	MI	MC,probability	MI+MI ICluster,0.4	Total probability should be 1.
MP+MP	MV	MV	MC, probability	MV+VM VCluster,1	Similar to I+I
MP+MP	MI or MV	AM	true	MI+CM true	Produces another MP
MP+MP	MI	AV	true	MI+HeV true	Produces another MP
MP+MP	MV	AI	true	MV+HeI true,0.3	A capture radius is specified
MP+MP	MI or MV	AI or AV	MC,probability	MV+CV CCluster,1	Produces a MC
MP+MP	AM	AI or AV	MC,probability	CM+CV CCluster,.5	Produces a MC
MP+MP	AV or AI	AV or AI	MC,probability	HeV+HeV HeCluster,1	Produces a MC
MC+MP	MC:ID	MP	true/false	HeCluster:He2V2+mI true	Wildacards allowed, as in HeCluster:*+mI true.
MC+MC	MC	MC	true/false	HeCluster+HeCluster true,0.5	All or nothing, with a non-standard capture radius.
MP+Int	MP	Material	true/false	mI+Gas true	Reactions with an interface with the specified material.
MC+Int	MC	Material	true/false	HeCluster+Gas true	Cluster reaction with specified material.

Table 4.2: Results of the defined interactions

	MP	**MC**
MP	MP MC 0	MC
MC	MC	MC

impurities (carbon, helium, etc). For each impurity or element A, a mobile particle with each of the 5 different positions explained in section 3.2.4 can also be found.

MobileParticle parameters, for a given species A are the following ones:

A(migration) An arrhenius with the diffusivity.

A(formation) An arrhenius with the formation energy in the bulk.

The events that a MobileParticle can perform are:

Migration Standard.

Break-up For instance, $AI \rightarrow AM + MI$, being M the material. The formation parameters are used to define the break-up frequencies, but $E_{\text{binding}}(AI) = E_f(AM) + E_f(MI) - E_f(AI)$.

Frank-Turnbull For instance, $AM \rightarrow AI + VM$ or $AM \rightarrow AV + MI$. Formation values are used to compute such break-up frequencies.

If the charge model is activated, additional state charges can be defined for each particle. For instance

AM(state.charge) AM 0

AI(state.charge) `AI_0` 0 `AI_-` -1

The levels in the band gap for the transitions between charges have also to be defined.

```
1 float AI(e(-1,0))   .5
```

And, finally, a frequency to update the charge state between different states is required.

```
1 arrhenius AI(update) { 1 0.7 }
```

4.3.2 `Cluster`

Clusters are the agglomeration of any number of impurities with Is or Vs. The first step to include them in a simulation is to allow them in the `Model` file by defining its name. For instance:

```
1 map<string,bool> defined {
2    IVCluster true
3    ICluster  true
4    HeCluster true
5 }
```

Defines 3 `Clusters`: `ICluster`, `IVCluster` and `HeCluster`.

Once the defects are defined, its implementation is to be written in a file named as the just-defined cluster. Such implementation relies on the following parameters:

All the prefactor units are "diffusivity" units (cm^2s^{-1}). Consequently, they will be transformed into frequency units using the conversion factor $6/\lambda^2$. The lamdba used is not the `lambda` defined for each cluster, but the general one in `Models/lambda`.

shape of the defect, can be `disk`, `plane311`, `irregular` or `sphere`.

to which defect it evolves. Any valid cluster.

from which defect it evolved. Any valid cluster.

density.cm2 for `disk` and `plane311` only. Surface density (atoms/cm^2).

density.cm3 for `irregular` and `sphere` only. Volumetric density (atoms/cm^3).

migration.type for the defect. Could be `3d`, `parallel` or `perpendicular` to the axis 0 and 1 of the defect.

lambda used for the defect diffusion. Bigger or smaller lamdbas than the one in `Models/lambda` can be used. In particular, use of a bigger lambda will speed up the simulation of cluster diffusion while maintaning the correct diffusivity, but some caution should be taken, because if lambda is too big the cluster might diffuse over particles without reacting with them. Consequently, lambdas bigger than twice the minimum capture distance are not suggested.

axis.0, axis.1 Three axis for the defect. 0 and 1 are the plane for planar defects. 2 is the perpendicular axis to the plane. A `dist`, for instance, will be grown using the axis 0 and 1.

axes.ratio For `disk` and $\{311\}$ is the geometrical ratio of `axis.0` versus `axis.1`. A value of 2 would mean the defect is twice large in axis 0 than 1.

not.in.plane Defects are created in the specified plane and all its families. For instance, 1 0 0 will crate also 0 1 0, 0 0 1, defects. If one of these particular planes has to be avoided, it can be specified here. The main use is to avoid defects perpendicular to the surface that will diffuse in 1D parallel to the surface and then will never recombine, slowing down the simulation.

IV.model True or false. If false, instantaneous recombination of Is with Vs will happen. Otherwise, IV pairs will be maintained and recombined with a specified barrier. Setting this value to true forces to define also an IV.barrier parameter.

formation The formation energy (E_f) of the clusters. The origin of energies is an empty, perfect system. Consequently, formation energy would be the total addition of the formation energies of the isolated, constituent particles, minus the binding energies of each of this particles to the cluster.

In some cases, the potential energy (E_p) might be available. For potential energy if the cluster is something like `An^Im`, the origin of energies ($E = 0$) is assumed as a system where all the particles of the cluster exist, but are

infinitely separated (i.e., they do not interact). In this case, a system with n AM and m MI. Then, we would have that ($E_f(A_nI_m) = nE_f(AM) + mE_f(MI) + E_p(A_nI_m)$.

The procedure should returns a list with all the cluster IDs and its formation energies. This list is used as the existing clusters later. For instance, if CCluster is defined as a cluster and we want to have 3 clusters (`C2^I`, `C^I2` and `C2`), formation should return something like `C2^I -2 C^I2 -1 C2 .3`.

This parameter is of extreme importance because it also defines the available clusters. If a cluster is not defined here, but in other parameters, MonCa will ignore it. If a cluster is defined here but not in other parameters, MonCa will assume default values for the non-defined cluster in the other parameters. Formation, then, is the "canonical" definition of available clusters, for both types and syntaxis.

prefactor The emission prefactor for each emitted particle. The cluster is written first, and the emitted particle is attached with a comma. For instance, emission of MI from `C^I2` would be `C^I2,MI`. Creation of FPs is allowed, and also controlled with the prefactor (and the opposite reaction). For instance, He5 $\rightarrow$ He5V + MI. Since this is an I emission, the prefactor would be `He5V,MI`. A list with all the clusters and all the possible emissions is expected. If some prefactor for a cluster defined in the formation is missing, the code assumes its prefactor is 0.

percolation Boolean parameter to allow the reaction of a cluster with another one while growing, and without the need of diffusion (i.e., without needing one cluster moving into the other).

transform.to Procedure that returns the rates for transforming the defect to `to` . A list with all the clusters with the prefactors and energies is expected. If a cluster is not included, its Tranformation rate is "0 5", i.e., no transformation.

transform.from Procedure that returns the rates for transforming the defect to `from`. A list with all the clusters with the prefactors and energies is expected. If a cluster is not included, its transformation rate is "0 5", i.e., no transformation.

migration Migration energy of the cluster, as an arrhenius value (prefactor and activation energy). A list with all the clusters with the prefactors and energies is expected. If a cluster is not included, its transformation rate is "0 5", i.e., no migration.

IV.barrier Procedure that returns the rates for recombination of IV pairs. If a cluster is not included, the rate is "0 5", i.e., no recombination.

All these parameters are defined with a key and a value. The keys used in `formation` are used in the other procedures. Different notations might be available for the same cluster, as explained in sec. 3.3, but we strongly suggest being consistent with the one chosen. The parameters with procedures are described in Table 4.3

For instance, for a hypothetical carbon cluster defined in silicon:

Table 4.3: Keys and values needed to defined clusters

Parameter name	key	value	key example	value example	default
formation	cluster	float	CV2	-2.4	NO
prefactor	cluster,particle	float	CV2,V	1e-3	0
migration	cluster	arrhenius	CV2	0.01 1.2	0 5
transform.to	cluster	arrhenius	CV2	1e-3 .7	0 5
transform.from	cluster	arrhenius	CV2	1e-3 .5	0 5
IV.barrier	cluster	arrhenius	CV2^I	1e-2 .4	0 5

```
string shape            irregular
float density.cm3       5e22
string to            CCluster
string from          CCluster
string migration.type   3d
// coordinates axis.2   1 0 0
coordinates axis.1      0 1 0
coordinates axis.0      0 0 1
coordinates not.in.plane 0 0 0
float       axes.ratio  1
float       lambda 0.387

bool IV.model false
proc transform.to   { return "" }
proc transform.from { return "" }
proc migration      { return "" }

proc formation {
    set EfC 0
    set EfI 3.6
    set list ""

    append list "C2I␣␣␣[expr␣2*$EfC␣+␣1*$EfI␣-2.3]␣"
    append list "C2␣␣␣␣[expr␣2*$EfC␣+␣0*$EfI␣-1.2]␣"
    append list "C2I2␣␣[expr␣2*$EfC␣+␣2*$EfI␣-7]␣"
    append list "C3I␣␣␣[expr␣3*$EfC␣+␣1*$EfI␣-1.7]␣"
    append list "C3I2␣␣[expr␣3*$EfC␣+␣2*$EfI␣-9.7]␣"
    append list "C3I3␣␣[expr␣3*$EfC␣+␣3*$EfI␣-11.5]␣"
    append list "C4I3␣␣[expr␣4*$EfC␣+␣3*$EfI␣-13.7]␣"
    append list "C4I4␣␣[expr␣4*$EfC␣+␣4*$EfI␣-16.5]␣"
    append list "C4I2␣␣[expr␣4*$EfC␣+␣3*$EfI␣-12.5]␣"
    append list "C5I3␣␣[expr␣5*$EfC␣+␣4*$EfI␣-13]␣"
    append list "C5I4␣␣[expr␣5*$EfC␣+␣4*$EfI␣-20.5]␣"
    append list "C5I5␣␣[expr␣5*$EfC␣+␣5*$EfI␣-25.0]␣"
    append list "C6I5␣␣[expr␣6*$EfC␣+␣5*$EfI␣-26.9]␣"
    append list "C6I4␣␣[expr␣6*$EfC␣+␣4*$EfI␣-25.4]␣"
    append list "C6I6␣␣[expr␣6*$EfC␣+␣6*$EfI␣-30.9]␣"
    return $list
}

proc prefactor {
     set list ""
```

```
43      append list "C2I,SiI␣␣␣2.03e-2␣"
44      append list "C2I,CI␣␣␣␣2.03e-2␣"
45      append list "C2I,VSi␣␣␣2.03e-2␣"
46      append list "C2,SiI␣␣␣␣2.03e-2␣"
47      append list "C2,CI␣␣␣␣␣2.03e-2␣"
48      append list "C2I2,VSi␣␣2.03e-2␣"
49      append list "C2I2,CI␣␣␣2.03e-2␣"
50      append list "C3I,SiI␣␣␣2.03e-2␣"
51      append list "C3I,CI␣␣␣␣2.03e-2␣"
52      append list "C3I2,SiI␣␣2.03e-2␣"
53      append list "C3I2,CI␣␣␣2.03e-2␣"
54      append list "C3I2,VSi␣␣2.03e-2␣"
55      append list "C3I3,VSi␣␣2.03e-2␣"
56      append list "C3I3,CI␣␣␣2.03e-2␣"
57      append list "C4I3,SiI␣␣2.03e-2␣"
58      append list "C4I3,SiI␣␣2.03e-2␣"
59      append list "C4I3,CI␣␣␣2.03e-2␣"
60      append list "C4I2,CI␣␣␣2.03e-2␣"
61      append list "C4I2,SiI␣␣2.03e-2␣"
62      append list "C4I4,CI␣␣␣2.03e-2␣"
63      append list "C4I4,VSi␣␣2.03e-2␣"
64      append list "C5I3,SiI␣␣2.03e-2␣"
65      append list "C5I3,CI␣␣␣2.03e-2␣"
66      append list "C5I4,SiI␣␣2.03e-2␣"
67      append list "C5I4,VSi␣␣2.03e-2␣"
68      append list "C5I4,CI␣␣␣2.03e-2␣"
69      append list "C5I5,CI␣␣␣2.03e-2␣"
70      append list "C5I5,VSi␣␣2.03e-2␣"
71      append list "C6I5,SiI␣␣2.03e-2␣"
72      append list "C6I5,VSi␣␣2.03e-2␣"
73      append list "C6I4,CI␣␣␣2.03e-2␣"
74
75      return $list
76  }
```

Finally, for a cluster with a `IV.model` set to true, an extra producedure would be required.

```
1   proc IV.barrier {
2    set prefactor   5.0e-4
3    set energy(1)   0.43
4    set energy(2)   0.7
5    set energy(199) 1.6
6    set energy(255) 2.7
7    set alpha       1.0
8
9    set list ""
10   for { set size 0 } { $size < 50 } { incr size } {
11       for { set iv 1 } { $iv < 250 } { incr iv } {
12      set pref [expr ($prefactor*pow($iv,$alpha))]
13      set ener $energy(255)
14
15      if { $iv <= 1 } { set ener $energy(1) }
16      if { $iv == 2 } { set ener $energy(2) }
17      if { $iv <= 199 && $iv > 2 } {
```

```
18          set b [expr ($energy(199) - $energy(2))/(199. -2.)]
19          set a [expr $energy(199) - $b*199.]
20          set ener [expr $a + $b*$iv]
21      }
22      if { $iv <= 255 && $iv > 199 } {
23          set b [expr ($energy(255) - $energy(199))/(255. -199.)]
24          set a [expr $energy(255) - $b*255.]
25          set ener [expr $a + $b*$iv]
26      }
27      lappend list V$iv^I[expr $size+$iv]
28      lappend list $pref
29      lappend list $ener
30      lappend list V[expr $size + $iv]^I$iv
31      lappend list $pref
32      lappend list $ener
33       }
34     }
35     return $list
```

Cluster interactions

The result of the interactions between clusters needs to be defined in an array of strings names `interaction.result` placed in the `Models` file for each material. Every line in the array must contain 5 arguments in the form `reactant1 operator reactant2 = result`.

reactant1 Type of the first interacting defect. (For instance, ICluster)

reactant2 Type of the second interacting defect. (For instance, $<111>$)

operator An operator to take the defect sizes (s_1 and s_2, counted as the total number of particles) into account. One of the following:

- **+** To specify the result independently on the size of both reactants.
- **==** When one reactant has the same size as the other. ($s_1 = s_2$)
- **∼=,n** When one reactant is approximately the same size as the other, measured as
$$|s_1 - s_2|/\max(s_1, s_2) < n$$
.
- $<$ When $s_1 < s_2$.
- $>$ When $s_1 > s_2$.

Only one reaction for each case has to be specified. For instance `<111> > <100> = <111>` also implies that `<100> < <111> = <111>`, but this last one does not need to be specified. If a reaction is specified several times, a warning will be issued and the last reaction will be taken only.

result Name of the resulting defect. (For instance, $<111>$)

Example

```
array<string> interaction.result {

        ICluster + ICluster = ICluster
        ICluster + <100> = <100>
        ICluster + <111> = <111>
        VCluster + VCluster = VCluster
        <100> + <100> = <100>

        <100> ~=,.05  <111> = <100>
        <100> >   <111> = <100>
        <100> <   <111> = <111>

        <111> ~=,.05 <111> = <100>
        <111> >  <111> = <111>
        <111> <  <111> = <111>

        VCluster == <100> = <100>
        VCluster < <100> = <100>
        VCluster > <100> = VCluster

        VCluster == <100> = <100>
        VCluster == <111> = <111>
        VCluster < <111> = <111>
        VCluster > <111> = VCluster

        VCluster == ICluster = VCluster
        VCluster >  ICluster = VCluster
        VCluster <  ICluster = ICluster
}
```

Summary

Clusters in MonCa are very flexible defects. They can be used as multi clusters for activation deactivation of defects (i.e., BICs and similar in semiconductors), to simulate bubbles (HeClusters in metals) extended defects ({311}, <111>, etc...) and amorphous pockets (`V2I` etc...).

Clusters can migrate, transform, emit constituent particles and recombine existing IV pairs. Clusters can adopt different shapes, and interact between them.

4.3.3 Interfaces

An interfaces is a defect between materials with different properties, or between a material and the lack of it (a free interface).

Parameters

Interface defects are not user-specified. They are build by default between materials with different names. For a free interface, the special material `Gas` can be used.

Because an interface involves two materials, the interface parameters are defined in folders that are created by appending the name of both materials separated with

an underline "_", in alphabetical order. This way, an interface between Iron and Gas would be parametrized under the `Gas_Iron` name.

Each material folder contains the parameters for each element that can interact with the interface. The parameters allowed for these elements are listed below. The parameter `left` is important, because it defines what is called "left" in the parameters using it. This way, if `left` is set to "Iron", any param finishing in "left" will refer to "Iron", and "right" will refer about the other material in the interface, independently on the name of the folder.

For instance, in `Iron_Gas` if `left` is Iron, then any reference to "right" will refer to Gas. In `Copper_Iron`, if `left` is also Iron, "right" will mean Copper. As you can see, this is independent on whether the folder name (ordering the names alphabetically) starts or finishes with Iron.

`barrier.left` The (extra) energetic barrier to be overcome to react with the interface.

`barrier.right` See barrier.left.

`desorption.high` See desorption.threshold.

`desorption.low` See desorption.threshold

`desorption.threshold` For surface concentrations lower than this value, the parameter `desorption.low` will be used. Otherwise `desorption.high` will be applied. Desorption is the probability for a particle to be annihilated at the surface.

`formation` The formation energy of the impurity when trapped at the interface. The particle will be emitted to either side depending on the difference of energies between this formation energy and the formation energy in the materials at either side.

`left` Definition of which material is considered left, independently on the name of the folder for the interface.

`migration` Migration energy and prefactor for the inpurity on the interface.

`recombination.lenght.left` For self-interstitials and self-vacancies, defines the sink efficiency at the interface.

`recombination.lenght.right` See recombination.lenght.left.

A graphical description of the meaning of these parameters can be seen in Fig. 4.2

Interactions

The interactions with interfaces are defined in the `Models` file for each side of such interface. The syntax for them is

```
1 Defect+MaterialFullName true/false
```

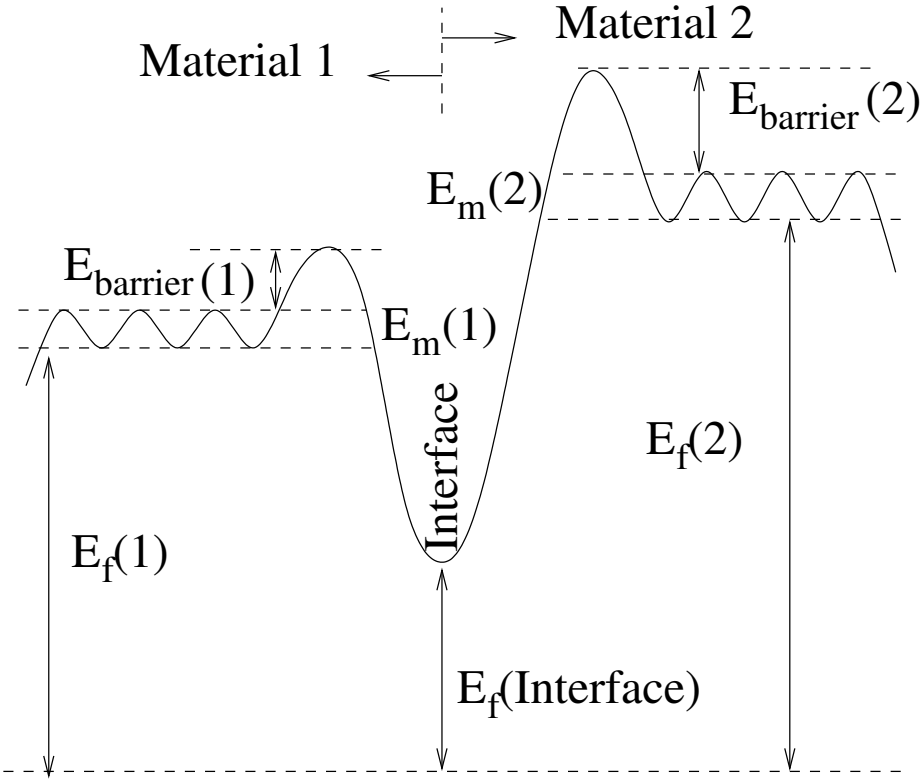

Figure 4.2: Energies involved at the interface reactions.

the full name for the material is to be used. This way the syntax checker can distinguish between the interaction with an interface (for instance, `I+Iron`, or with an impurity of the same material `I+Fe`.

As an example, in the case of a `Copper_Iron` interface, at the `Iron` side, the interaction with the material is defined at `Iron/Models/interactions` as

```
1 Defect+Copper true
```

being `Defect` the defect being considered (I, V, ICluster, etc...). Similarly, at the Copper side we will see

```
1 MP+Iron true
```

Conclusions

Interfaces are 2D planar objects isolating one material from a different one. Mobile particles and clusters are allowed to interact with them. For mobile particles, they can be annihilated, or trapped and re-emitted to either side. For clusters they can only be annihilated (desorption).

4.4 Amorphization

When a region of the material overcomes a certain threshold of damage concentration (i.e. self-interstitials and vacancies), that region is amorphized, so all the atoms from the material are displaced from their crystalline position, and no more damage can be introduced in those areas. Self-interstitial and vacancy definitions no longer make sense in those areas. Such threshold value is defined in the `Models` file for each material.

amorphization.threshold=<value> Units are atoms/cm^3 units. To deactivate the amorphization model, it is enough to not assign a value to it.

If the amorphization model is wanted to be active, the definition of the amorphous material corresponding to the crystalline one needs to be defined. For example, to active the amorphous model in the material `Silicon` , `AmorphousSilicon` must be defined.

When amorphizing an area of the simulation, the `LKMC` module is called, and lattice atoms are placed on the interfaces between amorphous and crystalline zones. This is done dynamically when using `cascade` or `profile` commands. When new areas are amorphized, lattice atoms are automatically removed in areas where the amorphous/crystalline interfaces no longer exist.

For output information of this model, see section 7.3.

Chapter 5

Lattice KMC: Lattice atoms

5.1 Introduction

MonCa contains a module to perform Lattice KMC simulations. Such module can be used independently or linked to the Object KMC module.

The basis of a Lattice KMC simulation is the "Lattice Atom". A lattice atom is a representation of a real atom in the lattice of a crystal, or a position of such atom if it had to be in the lattice. Depending on the local configuration and other material properties (temperature, concentrations, deformations), different events are associated with the lattice atom. The main event to be associated is the "filling" of such position, if its represents an empty position.

Using such a simple approach, two different epitaxies can be simulated:

Solid Phase Epitaxial Regrowth or SPER, where a crystalline phase advances against an amorphous phase. In this case the "filling" is interpreted as an atom in the amorphous phase attaching to a crystalline position, and thus being incorporated to the crystalline phase. More information on these type of models can be found in Refs. **???**.

Solid Gas Epitaxy The epitaxial growth of a material by reactions with existing gases, for instance, Selective Epitaxial Growth. For this models, the "filling" of a Lattice Site is directly related to the deposition of a new atom on the interface. An example can be found in Ref. **?**. These type o models can also include different mechanisms: diffusion, etching, deposition, etc. An example of such a model is found in Ref. **?**.

Chapter 6

Output

6.1 Updates

MMonCa uses an update mechanism to call update events (to be described later) based on the following parameters:

time.decade The update event wil be called this number of times per decade of simulated time. For instance, a value of 10 here will update at times 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, ...

time.delta The update event will be called at the specified interval. For instance, for `time.delta`=5, at 5, 10, 15, 20, ...

time.min Minimum time to start the updates.

events The update event will be called each `events`.

All the above options work at the same time. For instance, it is possible to have the updates 3 times per decade, and also every 10 seconds.

These update mechanisms are used for the following modules:

snapshots MMonCa outputs the status the simulation with a frequency specified in the `MC/General/snapshot` update. This update produces two different actions. First, it outputs the current status, and it also calls the `snapshot` procedure. The user can defined a snapshot procedure to extract information, save the time evolution of the simulation, etc. The option `events` does not work for snapshots.

charge model update The frequency at which the Poisson solver is called is specified in the `MC/Electrostatic/update` parameter.

mechanical update The frequency at which the mechanical solver is called and the mechanical information fed into KMC is specified in the `Mechanics/General/update` parameter.

6.2 Obtaining output

The two commands mostly used to obtain information from a MoonCa simulation are:

extract It extracts profiles, concentrations, defect numbers, etc... in plain ASCII text. See 7.3.

save It saves the atomistic information in `xyz` or `ovito` format for visualization. See 7.11.

Extract can collapse the information in 1, 2 and 3 dimensions with the use of the `dimension` option.

Chapter 7

Commands

All the commands can use the generic option `no.print` when you do not want MonCa to print out the command line.

7.1 anneal

Anneals the simulation at the requested temperature in Celsius degrees. Different criteria to exit the anneal are `depth`, `events` or `time`.

depth=<depth> Specifies recrystallized depth, in nm, as the exit criteria.

epitaxy="list" Specifies, in "list", a map of gas names and partial pressures, used for epitaxy.

events=<events> Specifies a number of simulated events as the exit criteria.

log=<filename> Specifies the name of a file to log a list of simulated events and time every 100 events. (This list can be quite big).

temp=<temperature> Requested temperature in Celsius degrees.

time=<time> Specifies simulated time, in seconds, as the exit criteria.

This command prints an information line at a rate of 10 snapshots per decade, i.e., at 1,2,3... 10,20,30... 100,200... After priting the line, the procedure `snaphot` is called to perform user-defined actions. Such procedure can be used, for instance, to save the simulation.

```
1 proc snapshot { } {
2    save ovito=evolution append
3 }
```

7.1.1 Examples

- `anneal temp=600 time=20`
- `anneal temp=600 depth=20`

- `anneal temp=300 time=0.5 events=1000000`
- `anneal temp=700  time=1 epitaxy="Si 1 Ge 0.5"`

7.2 cascade

Command used to implant or insert cascades of point defects into the simulation. Mostly used for damage simulation from an external source.

The global options for all `cascade` commands are

correct.for.surface Option to change the depth coordinate and adapt it to the place where the surface is. It assumes the cascades are computed for a surface at $x = 0$.

defects Procedure used to obtain the defect type of clusters. i.e., Is V4 a ICluster, a 111, etc.

fluence Fluence for cascades in the simulation. The number of cascades will be $A\mathrm{fluence}$ being A the YZ simulation area. Compulsory. Units: cm^{-2}.

do.not.react If specified, the new incoming particles will not react with the existing ones.

do.not.shift For file cascades only: do not change the y and z coordinates of the read cascade: use it as it is.

flux Optional option to specify the rate at which cascades will be annealed. If not specified, cascades will be assumed to be "instantaneous"

periodic If present, this flag allows for coordinates outside the simulation cell to be reinserted by means of periodic boundary conditions.

temp Optional. Temperature in Celsius degrees to anneal the cascades (to simulate dynamic annealing). If not specified, it is the current simulation temperature.

voluminic Optional. If present, this flag allows to introduce the cascades homogeneously in the volume of the sample instead of just in the surface.

The command `cascade` will introduce the cascades one by one in the simulator and perform a "dynamic annealing" of $-\Delta t \log(r)$ after each cascade. r is a random number between 0 and 1, and Δt is the average time between cascades, equals to `fluence/flux`.

The currently supported models for cascades are:

file The only supported source of cascades now: Read them from a file.

The following options are valid for `file` only.

file=<filename> Name of the input file. Compulsory.

format=<formattext> Optional format of the input file. It uses A:B:C:D if not specified.

Cascade accepts two different formats, called "New cascade" and "Number of particles". The general format of the input file is the same, the only difference is the label used to indicate when a new cascade starts. MMonCa detects the format automagically.

- A file can contain several cascades. Thus, it is necessary to insert some special label in the cascade file every time a new cascade starts.
- Each cascade starts with one of this two formats:

  # New cascade The verbatim label `# New cascade`.

  Number A number indicating how many defects (one per line) are contained in each cascade.

- After the special label, the list of point defects follows.
- The format for each line in the list of defects must be the same, and it has to contain, at least, the name of the point defect, and three columns with the x, y and z coordinates. The y and z of such coordinates refer to an impact point of $(0, 0)$.

The `cascade` command will read as many cascades as needed, in the order they have in the file, and shift them with random y and z numbers to fill the whole area. If more cascades than the ones specified in the file are needed the process will start again from the first cascade in the file.

The `format` command needed for the `file` option specifies the format for the point defects in the file. It is a string containing a list of field. Such list of fields is separated by colons ":". 4 fields are needed, representing the particle type (mobile particle types only) and the x, y and z coordinates in nanometers. The positions of such fields in this order will be indicated by using the letters A to Z. Arithmetic operations are possible. The letters A to Z represent the column number in the text file specified in the `filename`. Some examples follow.

The `defects` option allows the specification of a procedure to resolve the defect type of a given cluster:

```
1 proc defects { def } {
2    if { [string index $def 0] == "V" }
3       return "VCluster"
4    return "<111>"
5 }
```

7.2.1 Cascade examples

Two equivalent examples are given above in the two accepted formats. The format is automatically detected by MMonCa.

Format "# New cascade"

The following example contains 2 cascades:

```
1  # New cascade
2  I 23.4 .5  .9
3  V 21.2 .6  .85
4  I 17.2 .55 .92
5  V 18.0 .53 .87
6  I 20.0 .58 .91
7  # New cascade
8  I 19.9 -.1 .1
9  V 19.7 -.2 .2
10 I 17.2 .1  -.1
11 V 19.3  .2 .2
12 I 18.3 -.14   .17
13 V 20.1 -0.03 0.1
14 I 21.9  0.12 0.16
```

Format "number of defects"

The following example contains 2 cascades:

```
1  5
2  I 23.4 .5  .9
3  V 21.2 .6  .85
4  I 17.2 .55 .92
5  V 18.0 .53 .87
6  I 20.0 .58 .91
7  7
8  I 19.9 -.1 .1
9  V 19.7 -.2 .2
10 I 17.2 .1  -.1
11 V 19.3  .2 .2
12 I 18.3 -.14   .17
13 V 20.1 -0.03 0.1
14 I 21.9  0.12 0.16
```

7.2.2 Examples

Some examples for the `format` field follows:

- `A:B:C:D` A file in nanometers where the first column are the particles types and the following ones x, y and z.
- `A:C:D:E` A file, in nanometers, where the first column are the particle types and the x, y and z are codified in 3^{rd}, 4^{th} and 5^{th} columns. The second column in the file is not used.
- `D:A*1e7:B*1e7:C*1e7` A file, where x,y and z are in the first, second and third column and are specified in cm (being converted into nm by using the factor 10^7. The fourth column contains the particle types.
- `A:B+C*10:B+C*10:B+C*10` In this file, the x, y and z coordinates are going to be the same: the second column plus then times the third one.

Some examples for the whole command follows:

- `cascade file=cascade periodic fluence=5e14 flux=1e12 temp=-150`
- `cascade file=cascade periodic fluence=5e14 temp=-150 voluminic`
- `cascade file=cascade format=B:C*.287:D*.287:E*.287 periodic fluence=4.8e9`

7.3 extract

It extracts simulation information. It does not write information on screen, but returns it as a variable. If you want to see something on screen, use the `lowmsg` command to print it.

ac.coverage Returns the surface hydrogen coverage in epitaxial models, as a $[0-1]$ value (ML or monolayer).

ac.max Returns a list with the maximum values for the Lattice KMC interface position in x, y and z. Accepts the optional arguments `min.x=`<minx>, `max.x=`<max.x>, `min.y=`<miny>, `max.y=`<max.y>, `min.z=`<minz> and `max.z=`<max.z> to limit the extraction domain size.

ac.mean Returns a list of the mean value of the Lattice KMC interface position in x, y and z respectively. Accepts the optional arguments `min.x=`<minx>, `max.x=`<max.x>, `min.y=`<miny>, `max.y=`<max.y>, `min.z=`<minz> and `max.z=`<max.z> to limit the extraction domain size.

ac.min Returns a list with the minimum values for the Lattice KMC interface position in x, y and z. Accepts the optional arguments `min.x=`<minx>, `max.x=`<max.x>, `min.y=`<miny>, `max.y=`<max.y>, `min.z=`<minz> and `max.z=`<max.z> to limit the extraction domain size.

ac.stdev Returns a list of the standard deviation of the Lattice KMC interface in x, y and z respectively. Accepts the optional arguments `min.x=`<minx>, `max.x=`<max.x>, `min.y=`<miny>, `max.y=`<max.y>, `min.z=`<minz> and `max.z=`<max.z> to limit the extraction domain size.

amorphous.fraction Returns the fraction of space that is amorphous in an `OKMC` enviroment for the material specified in the required option `material=`<mt>. For more information, read Section 4.4.

configuration Comparing MonCa effective configurations is not always straightforward, especially when parameters were overruled in the simulation script. This command enables one to dump all the actual configuration to the log or a file. All dump invocations must be preceded by the init command. Any configuration parameter overruled after this command does not affect the already happened dump content. If the optional argument `filename` is given, the configuration is dumped into the given file; otherwise into a Tcl variable. The format is `key:data type:value`.

coord=<x y z> Uses the coordinate x, y and z as the starting center to look for neighbors in the `coordination` command.

coordination Returns a list of neighbors, distance to it, the neighbor type and total account of neighbor types. It requires the use of the option `coord` to specify the value of the coordinate to use as a center and `radius` for the look-up radius. It buils a crystal as a sphere with the specified radius plus 0.5 nanometers, so the specified coordinate should be close to 0 0 0.

count.defects Returns how many defects are in the simulation. It accepts the following options to "filter" the results: `material` for a particular material, `name` to specify a particle name, `defect` for a defect name, `min.size` for a minimum size and `ID` for a particular defect type.

count.particles Returns how many particles are in the simulation. It accepts the following options to "filter" the results: `material` for a particular material, `particle` to specify a particle name, `defect` for a defect name, `min.size` for sizes equal or bigger and `ID` for a particular defect type. It is different from `count.defects`. For instance, a B2I3 cluster will return 3 interstitial particles, but only 1 defect.

count.positions Returns how many particles are in a particular position. It requires the argument `position` and accepts the argument `material` to filter the particles to the specified material.

defect Parameter used by `histogram` to specify the defect type. Also used by `profile` to restrict the output to a particular cluster.

defect.radius It computes the average radius (in nm) of the specified defects. It requires the parameter `defect` and accepts the optional parameters `ID` to specify a particular type and `min.radius` to set up a threshold that defines when a defect is "visible" and will be included in the calculation.

defects=<list> Returns all the defects (OKMC particles) and positions in the simulation. If all the defects are requested, an empty list (defects={ }) is to be provided. Otherwise, an enumeration of the defects requested, separated by spaces, is to be provided. For MobileParticles, `MobileParticle` has to be specified. For other defects, the defect name (eg. `ICluster`, `VCluster`) is to be written. If alloy atoms are requested, type `Alloy`.

diffusivity Returns the diffusivity of the specified particle `name` in the specified `material`. It does not work for I or V particles. If `macroscopic` is specified, then it tracks the diffusivity of the whole family of particles associated. Use `extract reset` to start measuring.

dimension Sets the dimension to collapse the output. Accepts 0, 1 2 and 3. If 0 is specified, no geometrical information is obtained, just a value with the overall mean value.

dose Extracts the damage recieved by the material by the `cascade` command. The units are in dpa. This command assumes that the cascades contain Frenkel pairs, that is, same number of interstitials and vacancies (accepts clusters but the total ammount of interstitials and vacancies must remain equal), if not this command will give an underestimated or overestimated dpa value.

histogram Requires the parameters `defect` and `material`. Returns a histogram for the given defect/s name. Several defects can be used separated by commas, for instance `ICluster,VCluster`. A histogram is considered a list of cluster names and the number of them present when issuing the command.

file=<filename> This option can be added to *any* of the previous parameters to write the results in filename.

fuzz Returns a "y z depth" array listing all the depths at which a free interface was detected.

jumps Returns the number of jumps for the mobile particle specified in the required option `name=<pt>`

material.location Returns the long material name at the location given by `x=<value>`, `y=<value>` and `z=<value>`. It can be used to unit test material definition Tcl functions or JSON files.

material.map Extracts the internal mesh's material attributes. For the sake of simple implementation, it represents each mesh cell with a slightly smaller cell, with each corner having the material attribute of the cell. This file is best viewed using linear interpolation, for example in Paraview. The linear interpolation of the thin layers between the cells are to be ignored. The material attributes in the VTK file come from the internal MMonCa representation, which is also visible in the log. Requires the `filename=<VTK filename>`

min.radius Optional parameter for `defect.radius`. It considers only "visible" defects with radius equal or bigger than the specified one.

min.size Optional parameter for `count.defects` or `count.particles`. It limits the count to defects with size equals or bigger than the specified. The size of a defect is defined as its total number of particles.

profile Returns the concentration for the particle specified in the required option `name=<pt>`. When simulating binary alloys it is possible to extract the atom counters of the A and B species in the AB binary alloy with the `name=A.atoms` and `name=B.atoms` respectively. Also it is possible to extract the profile of the A atoms of the AB binary alloy by no introducing the `name` keyword and introducing the desired material profile with the `material` keyword. LKMC information can also be extracted through `name=lkmc.defects` and `name=lkmc.ac` options and corresponding to defective configurations and amorphous/crystalline interface respectively. Additonal OKMC options are in the examples.

profile.damage Returns the damage concentration (i.e. self-Interstitials and Vacancies) for the material specified in the required option `material=<mt>`. Such concentration saturates at the `amorphization.threshold` value. For more information, read Section 4.4.

profile.mobile Returns the mobile concentration for the mobile particle specified in the required option `name=<pt>`. It assumes $[X] = \mathrm{jumps}/(\Delta t \Delta V \nu(pt))$.

reset Resets the information for mobile particles. It allows incremental displays of properties for mobile particles. If `reset` is not used, then all the properties for mobile particles are accounted from the very beggining of the simulation, otherwise, the magnitudes are an average between the last reset and the current time.

strain.xy Returns the shear strain (xy) loaded into OKMC in a 2D X/Y projection.

time Returns the current simulated time.

7.3.1 Examples

- `extract count.particles defect=ICluster` Extracts how many particles are in "ICluster"
- `extract count.particles defect=VCluster ID=V3` Extracts how many particles are contained in V3 in "VCluster".
- `extract diffusivity macroscopic name=He material=Copper`
- `extract time`
- `extract ac.mean` Extracts a list containing the x, y and z mean values of amorphous/crystalline interface positions. Use `[lindex [extract ac.mean] 0]` to extract the x mean value.
- `extract histogram defect=<111> material=Iron`
- `extract profile name=Cr dimension=1 material=Iron` Extracts the profile [cm^{-3}] of the Cr atoms collapsed to the X-dimension
- `extract profile material=Iron` Extracts the profile [cm^{-3}] of the Iron lattice atoms (computed from the cell counters)
- `extract profile name=X*` Extracts the profile for all the active (not clustered) X atoms.
- `extract profile defect=* name=X` Extracts the profile for all clustered B atoms such that a B_3 cluster is counted three times.
- `extract profile defect=BICs material=Silicon name=B2I` Extracts the profile for the B_2I cluster within the BICs family.
- `extract profile defect=BICs material=Silicon name=B*` Extracts the profile for all clustered B atoms within the BICs cluster family, such that a B_3 cluster is counted three times.
- `extract profile defect=BICs material=Silicon` Extracts the profile for all BICs clusters.
- `extract material.map filename="materials.vtk"` Extracts the VTK file rendered in Fig.7.1 from the JSON mesh description below.

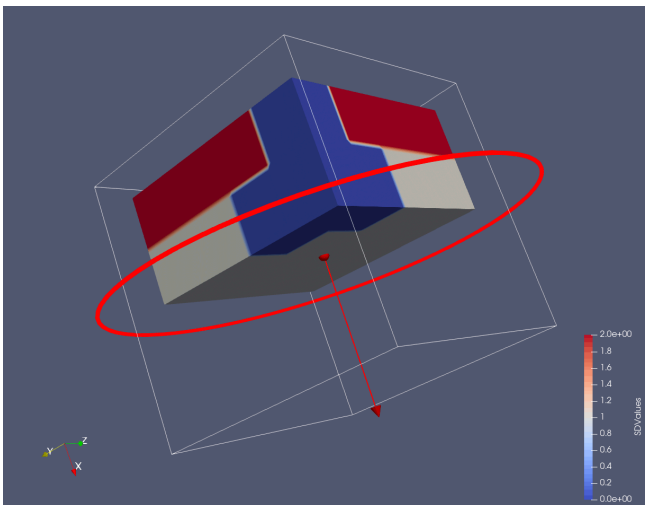

Figure 7.1: Clipped rendering of the VTK file exported from the JSON mesh below

```
{
  "linesX": [
    0.0, 1.5, 3.0, 4.5, 6.0
  ],
  "linesY": [
    0.0, 1.4, 2.8, 4.4, 6.0
  ],
  "linesZ": [
    0.0, 1.3, 2.6, 3.9, 6.0
  ],
  "materialIDs": [
    0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
  ],
  "materialMapping": {
    "SiO2": 0,
    "Silicon": 1,
    "SiliconGermanium": 2
  }
}
```

7.4 init

Defines the simulation cell sizes and the material specification to be used.

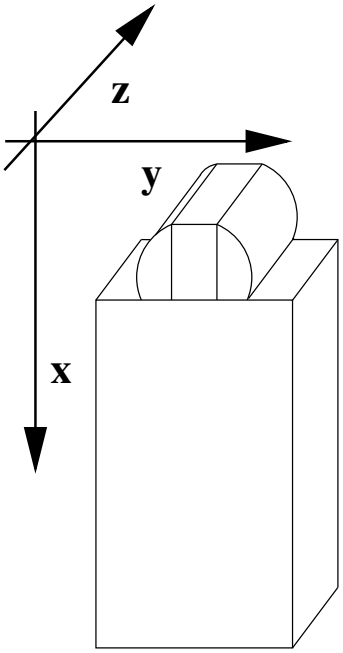

Figure 7.2: Definition of axes in MMonCa

7.4.1 Uniform mesh specified using minimum and maximum

material=<procedure> Specifies the name of a procedure where MMonCa will obtain the material information. The procedure has three input arguments, x, y and z, and returns the name of a valid material.

maxx=<X> Maximum X size, in nanometers. (X is depth, see Fig. **??**)

maxy=<Y> Maximum Y size, in nm. Y is width.

maxz=<Z> Maximum Z size, in nm.

minx=<x> Minimum x size.

miny=<y> Minimum y size.

minz=<z> Minimum z size.

totalseconds=<seconds> Batch execution systems like SLURM require an upper runtime to be specified for the job. Exceeding this limit immediately terminates the job. MMonCa runtimes are hard to estimate in advance. By setting the total MMonCa simulation time to a somewhat lower value than the job time limit, results of a long-lasting simulation can be printed after the interrupted annealing. Only annealing has this implementation; other long-lasting calculations need to come.

Here, the mesh subdivision is ruled by configuration parameters `MC/Mesh/spacing.*` where * can be x, y, or z. Example:

```
proc material { x y z } {
        if { $x < 0 } { return "Gas" }
        return "Iron"
}
set sizeX  8
set sizeYZ 80
init minx=-2 miny=0 minz=0 maxx=$sizeX maxy=$sizeYZ maxz=$sizeYZ
    material=material
```

7.4.2 Nonuniform mesh specified using division plane locations

material=<**procedure**> Specifies the name of a procedure where MonCa will obtain the material information. The procedure has three input arguments, x, y and z, and returns the name of a valid material.

linesx=<{**x1 x2 ...**}> A Tcl array of real numbers in strictly increasing order. These define the location of division planes along the X axis, each one plane perpendicular to that axis.

linesy=<{**y1 y2 ...**}> Similar description for the Y axis...

linesz=<{**z1 z2 ...**}> and the Z axis.

totalseconds=<**seconds**> as above.

Example: the mesh in Fig.2.1 can be described as

```
proc material { x y z } {
        if { $x < 2 } { return "Gas" }
        return "Iron"
}
init linesx={0 2 3 4 6} linesy={0 0.5 1.5 3.5 6} linesz={0 1 2 3 4 5 6}
    material=material
```

Here, we define 5 planes perpendicular to the X axis, 5 planes for the Y axis and 7 planes for the Z axis. As these planes cut the simulation area into cells, there will be a total of (5-1)(5-1)(7-1)=112 cells. As for the original equidistant division, each cell is assigned a single material, here defined by the Tcl function in the material parameter.

7.4.3 Nonuniform mesh and material specified using a JSON file

It is also possible to load a JSON file containing all the geometrical and material information. This file has explicitelly to include definitions and sizes of all the volumes involved in the simulation; i.e. underlying substrates, specific structures as well as the gas volumes used in CVD chambers, etc.

mesh=<**filename**> Specifies the name of the JSON file.

totalseconds=<seconds> as above.

The JSON file contains the locations of the division planes along each axis, together with a list of material IDs for each cell defined by the planes. A mapping of material name to ID used in the cell list must also be provided. Here, the material names must correspond to the long material names in MMonCa. In the example below, Gas is on the top (x is the depth coordinate), below it resides Si:

```
init mesh="mesh.json"
```

with the JSON file

```
{
  "linesX": [ 0.0, 1.5, 3.0 ],
  "linesY": [ 0.0, 1.4, 3.0 ],
  "linesZ": [ 0.0, 1.4, 2.8, 4.4, 6.0 ],
  "materialIDs": [ 33, 33, 33, 33, 33, 33, 33, 33,  22, 22, 22, 22, 22,
    22, 22, 22 ],
  "materialMapping": {
    "Silicon": 22,
    "Gas": 33
  }
}
```

The material ID values must be less than 245, and have nothing to do with MMonCa's internal material IDs (which appear in the log). The materialIDs array must have a length of $cells_X * cells_Y * cells_Z$, where $C_i = lines_i$ length - 1. This must hold because the lines* arrays specify the division planes for the mesh, and the cells are the areas divided by these planes.

The materialIDs array run first along the Z axis, then on Y and then on X. (x is the depth coordinate) In other words, if we take the index i (starting from 0) of a number in the materialIDs array, the cell indices along the X, Y and Z axes can be calculated by

```
ix = i / cellsY / cellsZ
iy = (i % (cellsY * cellsZ)) / cellsZ
iz = (i % (cellsY * cellsZ)) % cellsZ
```

7.5 insert

It inserts OKMC objects in the simulator. Accepted defects are:

Clusters Requires a `defect` name and an `ID`.

Mobile particles Requires a `particle` name.

coord=<x y z> Coordinates for the defect or the center of mass of the defect.

defect=<name> Inserts a defect with the specified name. It requires the proper `ID` parameter.

do.not.react Flag to avoid the reaction of just inserted particles. (i.e., if an inserted particles is on the capture radius of an existing one, it does not react).

ID=<ID> Defect's ID.

interface Flag to insert particles at interfaces. Use together with the `particle` parameter.

particle=<type> Inserts a particle with the specified type at the requested coordinates.

7.5.1 Examples

- `insert defect=Cluster ID=HeV2 coord={5 5 5}`
- `insert particle=HeV coord={$x1 $y1 $z1}` Insert an HeV at coordinates (x1, y1, z1) previously defined.
- `insert defect=ICluster ID=^I4 coord={3 4 5 }`

7.6 lowmsg

Prints its arguments, to be seen with the lowest verbosity, on the screen and in the default logfile. Use it instead of .

7.6.1 Examples

- `lowmsg "Here we are"` Puts "Here we are"
- `lowmsg "There are [extract count.particles] defects"` Puts the total number of defects.

7.7 param

Gets and sets parameters.

add Adds without overwriting the value for the specified parameter. It works for only two types:

array<string,string> Has to be used together with `type`, `key`, `index` and `value`.

array<string> Requires `type`, `key` and `value`.

index=<key> Operates in a particular a value of a map or array.

key=<key> Path to the parameter.

type=<type> The argument type: Only `array<string,string>` is supported.

value=<value> New value for the parameter.

get Obtains the value for the specified parameter. Has to be used together with `type` and `key`, and optionally with `index` and `size`.

index=<idx> Index for map or vectors. For vectors is an unsigned.

key=<key> Path to the parameter.

size=<size> Size for `proc.prefactor` or `proc.potential` or `proc.migration`.

type=<type> See the list of types.

get.reaction Returns `true` or `false` for a specified reaction looking at the internal look-up tables. It needs to compulsory parameters, `material` and `index`.

index=<key> Reaction to look at.

material=<mat> Material.

set Overwrites the value for the specified parameter. Has to be used together with `type`, `key` and `value`. The option `new` is optional and allows to create a new parameter when there is no value assigned yet. The option `index` is also optional, and allows to pick up a particular definition inside a map or array.

index=<key> Operates in a particular a value of a map instead of the whole map.

key=<key> Path to the parameter.

new Does not force the previous existence of the parameter.

type=<type> The argument type: One of `bool`, `int`, `float`, `string`, `array<string,string>`, `map<string,string>`, `map<string,float>`, `coordinates`, `arrhenius` or `proc`.

value=<value> New value for the parameter.

unset Deletes the value for the specified parameter. Has to be used together with `type`, `key` and `value`. The option `index` is optional, and allows to pick up a particular definition inside a map or array.

index=<key> Operates in a particular a value of a map instead of on the whole data.

key=<key> Path to the parameter.

type=<type> The argument type: One of `bool`, `int`, `float`, `string`, `array<string,string>`, `map<string,string>`, `map<string,float>`, `coordinates`, `arrhenius` or `proc`.

value=<value> New value for the parameter.

The types allowed in the option `type` are:

- `bool` set, get, unset
- `int` set, get, unset
- `float` set, get, unset
- `string` set, get, unset
- `array<string>` add, set, get, unset
- `array<string,string>` add, set, get, unset
- `map<string,bool>` set, get unset

- map<string,string> set, get, unset
- map<string,float> set, get, unset
- map<string,arrhenius> set, get, unset
- coordinates set, get, unset
- arrhenius set, get, unset
- proc set, get, unset

7.7.1 Examples

- param set type=bool key=MC/Mesh/periodic.x value=true
- param set type=float key=MC/Mesh/lambda value=0.287
- param set type=arrhenius key=Iron/Carbon/CI(binding) value={ 5e-2 0.87 }
- param set type=map<string,string> key=Iron/Iron/ interactions value={ FeI+Gas true }
- param set type=map<string,string> key=Iron/Iron/ interactions value=true index=FeI+Gas
- param set type=proc key=Iron/HeCluster/formation value={ { if {$size != "HeV2" } { return 5 }; return 1.5} }
- set l [param get type=float key=MC/Mesh/lambda]
- set PmV [lindex [param get type=arrhenius key=Iron/Vacancy/VFe(migration)] 0]
- set EmV [lindex [param get type=arrhenius key=Iron/Vacancy/VFe(migration)] 1]
- set PbHeV2 [param get type=proc key=Iron/HeCluster/prefactor index=HeV2,VFe]
- set EbHeV2 [param get type=proc key=Iron/HeCluster/formation index=HeV2]
- param get.reaction material=Iron index=He2V2+VFe

7.8 profile

This command "reads" a profile from a TCL procedure and atomizes it into the requested defect. It needs at least two arguments, the proc with the profile information, and the name of the defect.

The proc has to be specified by the user in the input script.

name can be a point defect, or a cluster. MonCa tries to figure out which type. If a cluster (for instance, "I56" is detected, and extra parameter type is requested.

do.not.react Avoids the reaction of an incoming particle with an existing one.

name=<ID> Specifies the particle or defect ID (i.e. Ci, C2I3, etc...)

defect=<defect>¿ For clusters, specifies between the different clusters (ICluster, Void, etc...)

proc=<procedure> Specifies the TCL procedure that, taking the arguments x, y and z in nanometers, will return the concentration.

7.8.1 Examples

For instance, if `proc=myName`

```
1 proc myName { x y z } {
2   if { $x > 19 && $x < 21 } { return 1e20 }
3   return 0
4 }
5
6 profile name=FeI proc=myName
7 profile name=V4 proc=myName defect=VCluster
```

7.9 report

This command it used to display reports. One or more of the following options can be used:

`all` All of the above but `domains` and `reactions.interface`.

`defects` Lists all the defects currently in the simulation.

`domains` Information on the different domains created for parallelization.

`events` Lists all the events performed by the KMC algorithm.

`insertions` Lists the particles and defects created (with `insert` or `profile`).

`mesh` If in LKMC mode, displays the current mesh status.

`reactions` Lists all the reactions that have taken place.

`reactions.interface` Details the ractions at interfaces.

7.9.1 Examples

- `report defects`
- `report events`
- `report mesh defects events reactions`

7.10 restart

The `restart` command allows to save and load the simulation state and the defined parameters, not to visualize it, but to "re-start" from it. The `restart load` option can be used as a substitute of the `init` command. A `.mmonca` file is generated or required.

The options specified in the `init` command are saved, making unnecessary to re-define a procedure with the `material`. All the parameters read, and also the ones changed with the `param` command are re-loaded. Consequently, there is no need to redefine parameters before the `restart load` option.

Information from both `OKMC` and `LKMC` modules can be saved and loaded.

load=<filename> Initializes a simulation using a previously saved state. `filename` contains the name of the file with the saved state. No extension is needed.

save=<filename> Dumps the state of the current simulation in `filename`. If a name with that file already exists, it is overwritten.

7.10.1 Format of the `.mmonca` file

The `.mmonca` file is a *gzipped* text file with the internal state of the simulator.

7.10.2 Examples

- `restart save=temporal` Saves the current state of the simulation in a file called "temporal.mmonca".
- `restart load=temporal` Inits the current simulation using the "temporal.mmonca" file as the current state.

7.11 save

Used to write a file with the simulation data such as LKMC and OKMC particles, materials or fields.

The type of file has to be chosen with one of the following options:

atomeye=<filename> Uses `filename` to generate and output file compatible with AtomEye.

xyz=<filename> Uses `filename` as the output xyz file.

lammps=<filename> Creates the file with a lammps format understood for `ovito` to read time evolution.

csv=<filename> Creates a file with the information separated by commas.

vtk=<filename> Generates a VTK file containing various datasets (eletrostatic potential, strain, stress and materials) using the XML VTK file format. This file can be opened with various softwares such as `ParaView` or `VisIt`.

The following optional parameters can be added to change the behavior of created files:

append Appends to the file instead of recreating it. (Use it for time evolution in the ovito format)

defects Allows the specification of a list of defects, separated by spaces, to be saved. Defects not specified will not be saved. For MobileParticles, `MobileParticle` has to be specified. For other defects, the defect name (eg. `ICluster`, `VCluster`) is to be written. If alloy atoms are requested, type `Alloy`.

lattice Writes all the lattice generated instead of the default A/C interface. By default, the full mesh cells near the A/C interface will be filled with atoms.

lkmc.defect Writes the defective lkmc atoms as well.

scale=<number> Applies number to scale all the positions.

margin=<number> When saving the lattice, only atoms within the given distance (in nm) from any solid/gas cell interface will be written. This helps visualize the actual simulation cell surface.

7.11.1 Examples

- `save xyz=filename` Saves the simulation data without lattice in XYZ format, overwriting the previous file if any.
- `save lammps=filename append scale=10` Appends the simulation data without lattice in LAMMPS format to the existing file.
- `save vtk=filename lattice` Saves the lattice near the A/C interface by filling full mesh cells. Fig.7.3
- `save vtk=filename lattice margin=0.4` Saves the lattice near the A/C interface by writing only the atoms within the $0.4\,\mathrm{nm}$ from any gas-solid interface. Fig.7.4

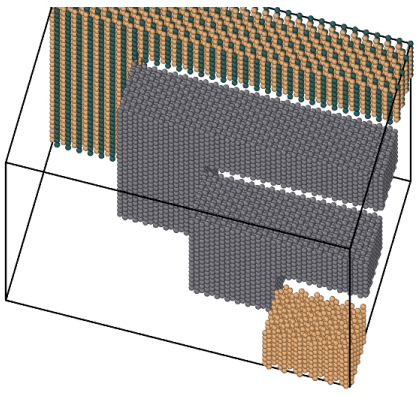

Figure 7.3: Simulation data saved with full lattice

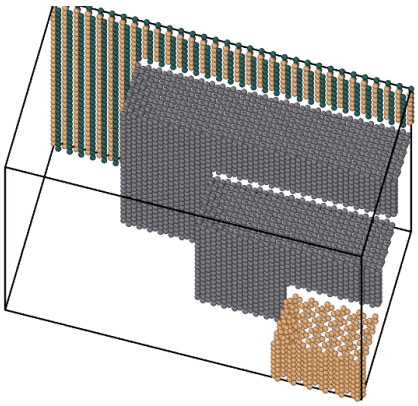

Figure 7.4: Simulation data saved with lattice atoms near the solid-gas cell interface

7.12 test

Used mainly to test the code. It checks that some conditions are true, otherwise it aborts with an error.

array=<x1 y1 x2 y2 ... xn yn> It requires the options `value=<value>`, `error=<error>`, `init=<init>` and `end=<end>` it checks that for all the xs between `init` and `end` the ys have a relative error smaller than `error` with respect to `value`. It prints the maximum relative error in the array.

array.2D Similar to `array` but with input in 2D.

array.3D Similar to `array` but with input in 3D.

arrays.2D Compares that, given two 2D arrays in `one` and `two`, their x and y are identical and the values are withing a relative error smaller than `error`.

float=<number> It requires the options `value=<value>` and `error=<error>`. It checks that `number` has a relative error smaller than `error` with respect to `value`. It prints the relative error in the array.

equal It tests that `one` is equal to `two`.

interval Tests that `value` is between `begin` and `end`.

not Inverts the meaning of the test, i.e., returns OK it is NOT passed.

tag=<tag> Optional argument that allows setting a "tag" associated with this test. Useful to distinguish between test commands when many are called.

7.12.1 Examples

- `test float=[extract count.particles] value=9 error=0`
- `test float=[extract count.particles particle=I] value=4 error=0`
- `test float=[extract count.particles particle=V] value=5 error=0`
- `test one="Hola" not equal two="hola"`

Chapter 8

Limitations

8.1 extract diffusivities

Only tracks diffusivity of impurities, i.e., not of I or V.

Chapter 9

Appendix

9.1 Binding energies

There is no `Particle(binding) { pref_b ener_b }`, instead there are two parameters: `Dopant(formation) { pref_d ener_d }` and `Particle(formation) { pref_p ener_p }`. Expressions for the variable change for energies and prefactors are shown below: $pref_{Dop} = 1 \quad ener_{Dop} = 0$

$$ener_{Part} = ener_{Dop} + ener_{IorV} - ener_{Bind} \tag{9.1}$$

$$pref_{Part} = \frac{pref_{Dop}}{pref_{Bind}} \cdot pref_{IorV} \cdot pref_{migIorV} \cdot v_{capt} \tag{9.2}$$

Where $ener_{IorV}$ is the formation energy of interstitials or vacancies, $pref_{IorV}$ is the initial concentration of interstitials or vacancies, $pref_{migIorV}$ is the migration prefactor of interstitials or vacancies (`IorV(migration)`), and v_{capt} is the capture volume defined as:

$$v_{capt} = 3.65 \cdot \lambda^3 \tag{9.3}$$

Where λ is the migration jump.

9.1.1 Example

- $E_f(C_i) = E_f(I) + E_f(C) - E_b(C_i)$
- $\left.\frac{C_{C_i}}{C_{C_{ref}}}\right|_0 = \left.\frac{C_C}{C_{C_{ref}}}\right|_0 \cdot C_{I_0} \cdot \nu_{mI_0} \cdot v_{capt} \cdot \frac{1}{\nu bin(C_i)}$

9.2 Copyrights

9.2.1 MMonCa

For the MMonCa code licensed under Apache 2:

9.2.2 Random Number Generator

The random number generator in MTonCA is not distributed under Apache2, but under the following, different conditions:

Bibliography

List of Figures

List of Tables