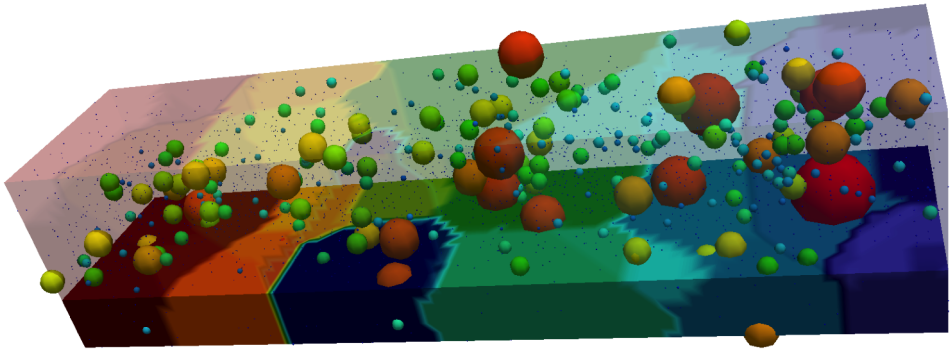


SRSCD

Spatially Resolved Stochastic Cluster Dynamics

Reference Manual

Aaron Dunn, Laurent Capolungo
Georgia Institute of Technology

Rémi Dingreville
Sandia National Laboratories

Enrique Martínez
Los Alamos National Laboratory

Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Georgia Institute of Technology

Sandia National Laboratories

Los Alamos NATIONAL LABORATORY EST. 1943

Contents

1 Introduction **1**

1.1 Introduction of SRSCD algorithm . . . 2

1.1.1 Rate theory background . . . 2

1.1.2 Reaction rates . . . 4

1.1.2.1 3D-migrating defects . . . 5

1.1.2.2 1D-migrating defects . . . 6

1.1.2.3 Mixed 3D-1D migration . . . 7

1.1.2.4 2D diffusion on a surface . . . 7

1.1.2.5 Spatially resolved rate equations . . . 9

1.1.3 Reaction rates for discrete elements . . . 10

1.1.4 Monte Carlo algorithm . . . 11

1.2 Cascade implantation in SRSCD . . . 11

1.2.1 Method 1: single volume element implantation . . . 13

1.2.2 Method 2: distributed cascade implantation . . . 14

1.2.3 Method 3: adaptive meshing . . . 15

1.2.3.1 Adaptive meshing: formulation . . . 18

1.2.3.2 Adaptive meshing: performance . . . 20

1.2.3.3 Computation time . . . 21

1.3 Synchronous parallel kinetic Monte Carlo algorithm . . . 25

1.3.1 Scaling of synchronous parallel SRSCD . . . 26

1.3.1.1 Damage accumulation in characteristic simulations of irradiated α-Fe: allowed defects and reactions . . . 27

1.3.1.2 Weak scaling during Frenkel pair implantation . . . 27

1.3.1.3 Scaling during displacement cascade implantation . . . 29

1.4 Validation: comparison to other methods and experimental results . . . 31

1.4.1 Comparison to cluster dynamics . . . 31

1.4.2 Comparison to OKMC . . . 34

1.4.3 Comparison to experiment: helium desorption from Fe thin films . . . 34

1.4.4 Comparison to experiment: neutron irradiation of bulk Fe . . . 38

2 Outline of code structure **41**

2.1 Brief overview . . . 43

2.1.1 Flow chart of main code and subroutines . . . 44
2.2 Initialization . . . 44
2.2.1 Mesh initialization . . . 44
2.2.2 Material and defect input . . . 45
2.2.3 Cascade input . . . 46
2.2.4 Damage profile input . . . 46
2.2.5 Simulation parameter input . . . 47
2.2.6 Random seed initialization . . . 47
2.2.7 Allocation of lists . . . 47
2.2.8 Initialization of lists . . . 47
2.3 KMC loop . . . 48
2.3.1 Compile total reaction rate for timestep choice . . . 48
2.3.1.1 Allocate list used to track updated defects . . . 48
2.3.2 Timestep choice . . . 49
2.3.3 Reaction choice . . . 49
2.3.4 Update defect list . . . 50
2.3.5 Update reaction list . . . 53
2.3.6 Delete cascade (if necessary) . . . 54
2.3.7 Check reaction rates (every n steps) . . . 55
2.3.8 Outputs (every m steps or time increment) . . . 55
2.3.9 Exit KMC loop . . . 56
2.4 Other actions . . . 56
2.4.1 Initialize annealing . . . 56
2.4.2 Deallocate memory: 1 . . . 57
2.4.3 Loop over multiple simulations . . . 57
2.4.4 Legacy code: Search for sink efficiency . . . 58
2.4.5 Deallocate memory: 2 . . . 58
2.4.6 Finalize MPI . . . 58

3 Input and output files **59**
3.1 List of toggles . . . 60
3.2 Standard input files . . . 60
3.2.1 Parameters.txt file . . . 61
3.2.2 Mesh input file and meshGenerator input file . . . 64
3.2.2.1 Case 1: Single material type . . . 64
3.2.2.2 Case 2: Multiple material types . . . 66
3.2.3 Defect input file . . . 68
3.2.4 Cascade input file (for cascade damage) . . . 76
3.2.5 Strain input file (for stress-assisted diffusion, implemented but not tested) . . . 76
3.2.6 Dose rate input file (for nonuniform implantation) . . . 77

3.2.7 Debug restart input file (for simulations restarting from nonzero defects) . . . 78
3.3 Standard output files . . . 79
3.3.1 Defect raw data . . . 79
3.3.2 Total defects . . . 80
3.3.3 Postprocessing . . . 81

4 Example simulations **83**
4.1 Cascade implantation in coarse-grained α-Fe . . . 84
4.2 Frenkel pair implantation in α-Fe thin films . . . 86
4.2.1 Frenkel pairs only (no helium) . . . 86
4.2.2 Frenkel pairs and helium in α-Fe thin films . . . 88
4.2.3 Frenkel pairs and helium in α-Fe thin films with non-uniform implantation profile . . . 88
4.3 Isochronal annealing . . . 89
4.4 Debug restart . . . 91
4.5 Grain boundary simulations . . . 93
4.6 Polycrystal simulations . . . 95

5 Modules Index **103**
5.1 Modules List . . . 103

6 Data Type Index **105**
6.1 Data Types List . . . 105

7 File Index **107**
7.1 File List . . . 107

8 Module Documentation **109**
8.1 derivedtype Module Reference . . . 109
8.1.1 Detailed Description . . . 110
8.2 meshreader Module Reference . . . 110
8.2.1 Detailed Description . . . 111
8.2.2 Function/Subroutine Documentation . . . 111
8.2.2.1 createconnectglobalfreesurfnonuniform . . . 111
8.2.2.2 createconnectglobalfreesurfuniform . . . 112
8.2.2.3 createconnectglobalperiodicnonuniform . . . 112
8.2.2.4 createconnectglobalperiodicuniform . . . 112
8.2.2.5 createconnectlocalfreesurfuniform . . . 113
8.2.2.6 createconnectlocalnonuniform . . . 113
8.2.2.7 createconnectlocalperiodicuniform . . . 114
8.2.2.8 createproccoordlist . . . 114
8.2.2.9 findglobalcell . . . 115
8.2.2.10 findlocalcell . . . 115

8.2.2.11 findneighborproc . . . 115
8.2.2.12 readmeshnonuniform . . . 115
8.2.2.13 readmeshuniform . . . 116
8.3 mod_srscd_constants Module Reference . . . 117
8.3.1 Detailed Description . . . 122
8.3.2 Variable Documentation . . . 122
8.3.2.1 activecascades . . . 122
8.3.2.2 alpha_i . . . 122
8.3.2.3 alpha_v . . . 122
8.3.2.4 annealidentify . . . 123
8.3.2.5 annealsteps . . . 123
8.3.2.6 annealtemp . . . 123
8.3.2.7 annealtempinc . . . 123
8.3.2.8 annealtime . . . 123
8.3.2.9 annealtype . . . 123
8.3.2.10 atomsize . . . 123
8.3.2.11 bindfunc . . . 123
8.3.2.12 bindsingle . . . 123
8.3.2.13 burgers . . . 123
8.3.2.14 cascadeconnectivity . . . 123
8.3.2.15 cascadeelementvol . . . 123
8.3.2.16 cascadelist . . . 124
8.3.2.17 cascadereactionlimit . . . 124
8.3.2.18 cascadevolume . . . 124
8.3.2.19 clusterreactions . . . 124
8.3.2.20 conc_i . . . 124
8.3.2.21 conc_v . . . 124
8.3.2.22 debugtoggle . . . 124
8.3.2.23 defectdensity . . . 124
8.3.2.24 defectlist . . . 124
8.3.2.25 difffunc . . . 124
8.3.2.26 diffreactions . . . 124
8.3.2.27 diffsingle . . . 124
8.3.2.28 dipolefilename . . . 125
8.3.2.29 dipolestore . . . 125
8.3.2.30 dislocationdensity . . . 125
8.3.2.31 dissocreactions . . . 125
8.3.2.32 dpa . . . 125
8.3.2.33 dparate . . . 125
8.3.2.34 elapsedtimereset . . . 125

8.3.2.35 finelength . . . 125
8.3.2.36 grainboundarytoggle . . . 125
8.3.2.37 hedparatio . . . 125
8.3.2.38 hesiatoggle . . . 125
8.3.2.39 ierr . . . 125
8.3.2.40 implantdist . . . 126
8.3.2.41 implantratedata . . . 126
8.3.2.42 implantreactions . . . 126
8.3.2.43 implantscheme . . . 126
8.3.2.44 implanttype . . . 126
8.3.2.45 impuritydensity . . . 126
8.3.2.46 impurityreactions . . . 126
8.3.2.47 kboltzmann . . . 126
8.3.2.48 master . . . 126
8.3.2.49 max3dint . . . 126
8.3.2.50 maxbuffersize . . . 126
8.3.2.51 maxrate . . . 126
8.3.2.52 meanfreepath . . . 127
8.3.2.53 meshingtype . . . 127
8.3.2.54 myboundary . . . 127
8.3.2.55 mymesh . . . 127
8.3.2.56 myproc . . . 127
8.3.2.57 numannihilate . . . 127
8.3.2.58 numcascades . . . 127
8.3.2.59 numcells . . . 127
8.3.2.60 numcellscascade . . . 127
8.3.2.61 numclusterreac . . . 127
8.3.2.62 numdiffreac . . . 127
8.3.2.63 numdipole . . . 127
8.3.2.64 numdisplacedatoms . . . 128
8.3.2.65 numdissocreac . . . 128
8.3.2.66 numemitsia . . . 128
8.3.2.67 numemitv . . . 128
8.3.2.68 numfuncbind . . . 128
8.3.2.69 numfuncdiff . . . 128
8.3.2.70 numgrains . . . 128
8.3.2.71 numheimplantevents . . . 128
8.3.2.72 numheimplanteventsreset . . . 128
8.3.2.73 numheimplanttotal . . . 128
8.3.2.74 numimplantdatapoints . . . 128

8.3.2.75 numimplantevents . . . 128
8.3.2.76 numimplanteventsreset . . . 129
8.3.2.77 numimplantreac . . . 129
8.3.2.78 numimpurityreac . . . 129
8.3.2.79 nummaterials . . . 129
8.3.2.80 numsims . . . 129
8.3.2.81 numsinglebind . . . 129
8.3.2.82 numsinglediff . . . 129
8.3.2.83 numsinkreac . . . 129
8.3.2.84 numspecies . . . 129
8.3.2.85 numtrapsia . . . 129
8.3.2.86 numtrapv . . . 129
8.3.2.87 numxcascade . . . 129
8.3.2.88 numycascade . . . 130
8.3.2.89 numzcascade . . . 130
8.3.2.90 omega . . . 130
8.3.2.91 omega1d . . . 130
8.3.2.92 omega2d . . . 130
8.3.2.93 omegacircle1d . . . 130
8.3.2.94 omegastar . . . 130
8.3.2.95 omegastar1d . . . 130
8.3.2.96 outputdebug . . . 130
8.3.2.97 pi . . . 130
8.3.2.98 polycrystal . . . 130
8.3.2.99 postprtoggle . . . 130
8.3.2.100 profiletoggle . . . 131
8.3.2.101 rawdattoggle . . . 131
8.3.2.102 reactionlist . . . 131
8.3.2.103 reactionradius . . . 131
8.3.2.104 recombinationcoeff . . . 131
8.3.2.105 restartfilename . . . 131
8.3.2.106 siapinmin . . . 131
8.3.2.107 siapintoggle . . . 131
8.3.2.108 singleelemkmc . . . 131
8.3.2.109 sinkeffsearch . . . 131
8.3.2.110 sinkreactions . . . 131
8.3.2.111 strainfield . . . 131
8.3.2.112 strainfilename . . . 132
8.3.2.113 systemvol . . . 132
8.3.2.114 temperature . . . 132

8.3.2.115 tempstore . . . 132
8.3.2.116 totaldpa . . . 132
8.3.2.117 totalimplantevents . . . 132
8.3.2.118 totalrate . . . 132
8.3.2.119 totalratevol . . . 132
8.3.2.120 totalvolume . . . 132
8.3.2.121 totdattoggle . . . 132
8.3.2.122 vtktoggle . . . 132
8.3.2.123 xyztoggle . . . 132
8.3.2.124 zint . . . 133
8.4 randdp Module Reference . . . 133
8.4.1 Detailed Description . . . 133
8.4.2 Function/Subroutine Documentation . . . 133
8.4.2.1 dprand . . . 133
8.4.2.2 sdprnd . . . 134
8.4.3 Variable Documentation . . . 135
8.4.3.1 dp . . . 135
8.4.3.2 index . . . 135
8.4.3.3 offset . . . 135
8.4.3.4 other . . . 135
8.4.3.5 poly . . . 135
8.5 reactionrates Module Reference . . . 135
8.5.1 Detailed Description . . . 137
8.5.2 Function/Subroutine Documentation . . . 137
8.5.2.1 adddiffusioncoarsetofine . . . 137
8.5.2.2 adddiffusionreactions . . . 138
8.5.2.3 adddiffusionreactionsfine . . . 139
8.5.2.4 addmultidefectreactions . . . 140
8.5.2.5 addmultidefectreactionsfine . . . 140
8.5.2.6 addsingledefectreactions . . . 141
8.5.2.7 addsingledefectreactionsfine . . . 142
8.5.2.8 checkreactionlegality . . . 143
8.5.2.9 defectcombinationrules . . . 143
8.5.2.10 finddparatelocal . . . 144
8.5.2.11 findheimplantratelocal . . . 144
8.5.2.12 findreactioninlist . . . 145
8.5.2.13 findreactioninlistdiff . . . 145
8.5.2.14 findreactioninlistmultiple . . . 145
8.5.2.15 findreactionrate . . . 146
8.5.2.16 findreactionratecoarsetofine . . . 146

8.5.2.17 findreactionratediff . . . 147
8.5.2.18 findreactionratedifffine . . . 147
8.5.2.19 findreactionratedissoc . . . 148
8.5.2.20 findreactionratedissocfine . . . 148
8.5.2.21 findreactionratefine . . . 149
8.5.2.22 findreactionrateimpurity . . . 149
8.5.2.23 findreactionrateimpurityfine . . . 150
8.5.2.24 findreactionratemultiple . . . 150
8.5.2.25 findreactionratemultiplefine . . . 151
8.5.2.26 findreactionratesink . . . 151
8.5.2.27 findreactionratesinkfine . . . 152

9 Data Type Documentation 153

9.1 derivedtype::bindingfunction Type Reference . . . 153
9.1.1 Detailed Description . . . 153
9.1.2 Member Data Documentation . . . 153
9.1.2.1 defecttype . . . 153
9.1.2.2 functiontype . . . 153
9.1.2.3 max . . . 154
9.1.2.4 min . . . 154
9.1.2.5 numparam . . . 154
9.1.2.6 parameters . . . 154
9.1.2.7 product . . . 154
9.2 derivedtype::bindingsingle Type Reference . . . 154
9.2.1 Detailed Description . . . 154
9.2.2 Member Data Documentation . . . 154
9.2.2.1 defecttype . . . 154
9.2.2.2 eb . . . 155
9.2.2.3 product . . . 155
9.3 derivedtype::boundarymesh Type Reference . . . 155
9.3.1 Detailed Description . . . 156
9.3.2 Member Data Documentation . . . 156
9.3.2.1 defectlist . . . 156
9.3.2.2 length . . . 156
9.3.2.3 localneighbor . . . 156
9.3.2.4 material . . . 156
9.3.2.5 proc . . . 156
9.3.2.6 strain . . . 156
9.3.2.7 volume . . . 156
9.4 derivedtype::cascade Type Reference . . . 156

9.4.1 Detailed Description . . . 157
9.4.2 Member Data Documentation . . . 157
9.4.2.1 cascadeid . . . 157
9.4.2.2 cellnumber . . . 157
9.4.2.3 localdefects . . . 158
9.4.2.4 next . . . 158
9.4.2.5 prev . . . 158
9.4.2.6 reactionlist . . . 158
9.4.2.7 totalrate . . . 158
9.5 derivedtype::cascadedefect Type Reference . . . 158
9.5.1 Detailed Description . . . 159
9.5.2 Member Data Documentation . . . 159
9.5.2.1 coordinates . . . 159
9.5.2.2 defecttype . . . 159
9.5.2.3 next . . . 159
9.6 derivedtype::cascadeevent Type Reference . . . 159
9.6.1 Detailed Description . . . 160
9.6.2 Member Data Documentation . . . 160
9.6.2.1 listofdefects . . . 160
9.6.2.2 nextcascade . . . 160
9.6.2.3 numdefectstotal . . . 160
9.6.2.4 numdisplacedatoms . . . 160
9.7 derivedtype::defect Type Reference . . . 160
9.7.1 Detailed Description . . . 161
9.7.2 Member Data Documentation . . . 161
9.7.2.1 cellnumber . . . 161
9.7.2.2 defecttype . . . 161
9.7.2.3 next . . . 161
9.7.2.4 num . . . 161
9.8 derivedtype::defectupdatetracker Type Reference . . . 161
9.8.1 Detailed Description . . . 162
9.8.2 Member Data Documentation . . . 162
9.8.2.1 cascadenumber . . . 162
9.8.2.2 cellnumber . . . 162
9.8.2.3 defecttype . . . 162
9.8.2.4 dir . . . 162
9.8.2.5 neighbor . . . 162
9.8.2.6 next . . . 163
9.8.2.7 num . . . 163
9.8.2.8 proc . . . 163

9.9 derivedtype::diffusionfunction Type Reference . . . 163
9.9.1 Detailed Description . . . 163
9.9.2 Member Data Documentation . . . 163
9.9.2.1 defecttype . . . 163
9.9.2.2 functiontype . . . 164
9.9.2.3 max . . . 164
9.9.2.4 min . . . 164
9.9.2.5 numparam . . . 164
9.9.2.6 parameters . . . 164
9.10 derivedtype::diffusionsingle Type Reference . . . 164
9.10.1 Detailed Description . . . 164
9.10.2 Member Data Documentation . . . 164
9.10.2.1 d . . . 164
9.10.2.2 defecttype . . . 165
9.10.2.3 em . . . 165
9.11 derivedtype::dipoletensor Type Reference . . . 165
9.11.1 Detailed Description . . . 165
9.11.2 Member Data Documentation . . . 165
9.11.2.1 equilib . . . 165
9.11.2.2 max . . . 165
9.11.2.3 min . . . 165
9.11.2.4 saddle . . . 166
9.12 derivedtype::mesh Type Reference . . . 166
9.12.1 Detailed Description . . . 166
9.12.2 Member Data Documentation . . . 166
9.12.2.1 coordinates . . . 166
9.12.2.2 length . . . 166
9.12.2.3 material . . . 167
9.12.2.4 neighborprocs . . . 167
9.12.2.5 neighbors . . . 167
9.12.2.6 numneighbors . . . 167
9.12.2.7 proc . . . 167
9.12.2.8 strain . . . 167
9.12.2.9 volume . . . 167
9.13 derivedtype::processordata Type Reference . . . 167
9.13.1 Detailed Description . . . 168
9.13.2 Member Data Documentation . . . 168
9.13.2.1 globalcoord . . . 168
9.13.2.2 localcoord . . . 168
9.13.2.3 numtasks . . . 168

9.13.2.4 procneighbor . . . 168
9.13.2.5 taskid . . . 168
9.14 derivedtype::reaction Type Reference . . . 168
9.14.1 Detailed Description . . . 169
9.14.2 Member Data Documentation . . . 169
9.14.2.1 cellnumber . . . 169
9.14.2.2 next . . . 169
9.14.2.3 numproducts . . . 169
9.14.2.4 numreactants . . . 169
9.14.2.5 products . . . 169
9.14.2.6 reactants . . . 169
9.14.2.7 reactionrate . . . 170
9.14.2.8 taskid . . . 170
9.15 derivedtype::reactionparameters Type Reference . . . 170
9.15.1 Detailed Description . . . 170
9.15.2 Member Data Documentation . . . 170
9.15.2.1 functiontype . . . 170
9.15.2.2 max . . . 170
9.15.2.3 min . . . 171
9.15.2.4 numproducts . . . 171
9.15.2.5 numreactants . . . 171
9.15.2.6 products . . . 171
9.15.2.7 reactants . . . 171

10 File Documentation **173**

10.1 Cascade_implantation.f90 File Reference . . . 173
10.1.1 Function/Subroutine Documentation . . . 173
10.1.1.1 addcascadeexplicit . . . 173
10.1.1.2 cascadecount . . . 174
10.1.1.3 cascademixingcheck . . . 174
10.1.1.4 cascadeupdatestep . . . 174
10.1.1.5 choosecascade . . . 175
10.1.1.6 createcascadeconnectivity . . . 176
10.2 CoarseMesh_subroutines.f90 File Reference . . . 176
10.2.1 Function/Subroutine Documentation . . . 176
10.2.1.1 clearreactionlistsinglecell . . . 176
10.2.1.2 countreactionscoarse . . . 177
10.2.1.3 finddefectinlist . . . 177
10.2.1.4 findnumdefect . . . 178
10.2.1.5 findnumdefectboundary . . . 178

10.2.1.6 resetreactionlistsinglecell . . . 178
10.2.1.7 updateimplantratesinglecell . . . 179
10.3 Deallocate_Lists.f90 File Reference . . . 180
10.3.1 Function/Subroutine Documentation . . . 180
10.3.1.1 deallocateboundarydefectlist . . . 180
10.3.1.2 deallocatecascadelist . . . 180
10.3.1.3 deallocatedefectlist . . . 181
10.3.1.4 deallocatematerialinput . . . 181
10.3.1.5 deallocatereactionlist . . . 182
10.4 Debug_subroutines.f90 File Reference . . . 182
10.4.1 Function/Subroutine Documentation . . . 182
10.4.1.1 debugcheckforunadmissible . . . 183
10.4.1.2 debugprintdefects . . . 183
10.4.1.3 debugprintdefectupdate . . . 183
10.4.1.4 debugprintreaction . . . 184
10.4.1.5 debugprintreactionlist . . . 184
10.5 Defect_attributes.f90 File Reference . . . 184
10.5.1 Function/Subroutine Documentation . . . 184
10.5.1.1 bindingcompute . . . 185
10.5.1.2 diffusivitycompute . . . 185
10.5.1.3 findbinding . . . 185
10.5.1.4 finddefectsize . . . 185
10.5.1.5 finddiffusivity . . . 185
10.5.1.6 findstrainenergy . . . 185
10.5.1.7 findstrainenergyboundary . . . 186
10.6 dprand.f90 File Reference . . . 186
10.7 FineMesh_subroutines.f90 File Reference . . . 186
10.7.1 Function/Subroutine Documentation . . . 187
10.7.1.1 chooserandomcell . . . 187
10.7.1.2 countreactionsfine . . . 187
10.7.1.3 findcellwithcoordinatesfinemesh . . . 187
10.7.1.4 findnumdefectfine . . . 188
10.7.1.5 findnumdefecttotalfine . . . 188
10.7.1.6 releasefinemeshdefects . . . 188
10.8 Initialization_subroutines.f90 File Reference . . . 189
10.8.1 Function/Subroutine Documentation . . . 190
10.8.1.1 annealinitialization . . . 190
10.8.1.2 initializeboundarydefectlist . . . 190
10.8.1.3 initializedebugrestart . . . 190
10.8.1.4 initializedefectlist . . . 191

10.8.1.5 initializefinemesh . . . 191
10.8.1.6 initializemesh . . . 192
10.8.1.7 initializerandomseeds . . . 193
10.8.1.8 initializereactionlist . . . 193
10.8.1.9 initializetotalrate . . . 194
10.9 kMC_subroutines.f90 File Reference . . . 194
10.9.1 Function/Subroutine Documentation . . . 195
10.9.1.1 choosereaction . . . 195
10.9.1.2 choosereactionsinglecell . . . 195
10.9.1.3 generatetimestep . . . 196
10.9.1.4 updatedefectlist . . . 196
10.9.1.5 updatedefectlistmultiple . . . 197
10.9.1.6 updatereactionlist . . . 198
10.10 MeshReader.f90 File Reference . . . 199
10.11 Misc_functions.f90 File Reference . . . 200
10.11.1 Function/Subroutine Documentation . . . 200
10.11.1.1 binomial . . . 200
10.11.1.2 factorial . . . 201
10.11.1.3 totalratecascade . . . 201
10.11.1.4 totalratecheck . . . 201
10.12 mod_derivedtypes.f90 File Reference . . . 201
10.13 mod_srscd_constants.f90 File Reference . . . 202
10.14 Postprocessing_srscd.f90 File Reference . . . 207
10.14.1 Function/Subroutine Documentation . . . 208
10.14.1.1 computeiconc . . . 208
10.14.1.2 computevconc . . . 208
10.14.1.3 outputdebugrestart . . . 208
10.14.1.4 outputdefects . . . 209
10.14.1.5 outputdefectsboundary . . . 209
10.14.1.6 outputdefectsprofile . . . 210
10.14.1.7 outputdefectstotal . . . 210
10.14.1.8 outputdefectsvtk . . . 211
10.14.1.9 outputdefectsxyz . . . 212
10.14.1.10 outputrates . . . 212
10.15 ReactionRates.f90 File Reference . . . 212
10.16 Read_inputs.f90 File Reference . . . 214
10.16.1 Function/Subroutine Documentation . . . 214
10.16.1.1 readcascadelist . . . 214
10.16.1.2 readimplantdata . . . 215
10.16.1.3 readmaterialinput . . . 215

10.16.1.4 readparameters . . . 215
10.16.1.5 readreactionlistsizes . . . 216
10.16.1.6 selectmaterialinputs . . . 216
10.17SRSCD_par.f90 File Reference . . . 217
10.17.1 Function/Subroutine Documentation . . . 217
10.17.1.1 srscd . . . 217
10.18StrainSubroutines.f90 File Reference . . . 218
10.18.1 Function/Subroutine Documentation . . . 219
10.18.1.1 calculatedeltaem . . . 219
10.18.1.2 diffusivitycomputestrain . . . 219
10.18.1.3 finddiffusivitystrain . . . 219
10.18.1.4 readdipoletensors . . . 219

11 Acknowledgements **221**

Index **223**

Bibliography **223**

Chapter 1

Introduction

Spatially Resolved Stochastic Cluster Dynamics (SRSCD) is a synchronous parallel kinetic Monte Carlo algorithm for simulating radiation defect implantation and accumulation in metals. The scientific background and mathematical formulation for the various aspects of this code can also be found in the PhD dissertation of Dunn (2016). SRSCD is intended to provide the user with a computationally efficient tool which can be used to study defect populations under a variety of irradiation conditions and in spatially resolved microstructures such as thin films, bulk materials, and nano-grained materials.

In the following sections, a mathematical basis for SRSCD is given. In Section 1.1, the governing equations of SRSCD are given. In Section 1.2, several models for simulating displacement cascade damage are presented. In Section 1.3, an implementation of SRSCD using a synchronous parallel kinetic Monte Carlo algorithm is discussed. Finally, in Section 1.4, SRSCD is compared to other modeling techniques as well as experimental results for several types of irradiation.

1.1 Introduction of SRSCD algorithm

The two most commonly used methods for simulating spatially resolved time-evolution of radiation damage in metals are the object kinetic Monte Carlo method (OKMC) (Stoller et al. (2008); Soneda and De La Rubia (1998); Domain et al. (2004); Caturla et al. (2000)) and spatially resolved rate theory (RT) or cluster dynamics (CD) methods (Ortiz and Caturla (2007); Ortiz et al. (2007); Li et al. (2012); Dunn et al. (2013a); Xu et al. (2007)), using finite element or finite difference algorithms for the spatial dependence. Each of these methods presents challenges due to computational demands. OKMC simulations limit the number of assumptions made about defect evolution by following individual defects as they diffuse stochastically throughout a material. However, due to the fact that the simulation follows each defect's diffusion pathway, simulations of reactor-relevant irradiation conditions can become computationally prohibitive using this method. Displacement damage in OKMC simulations is typically limited to less than 1 dpa and the minimum concentration of defects that can be simulated is limited to one defect per simulation cell volume (Stoller et al. (2008)). Spatially resolved rate theory assumes spatial homogeneity within each volume element and thus does not track individual defect movements. In contrast to OKMC, rate theory simulations can reach dpa levels of 100 or more and can model arbitrarily small defect concentrations (Stoller et al. (2008)). However, the number of rate equations to be solved increases exponentially with the number of species modeled (Marian and Bulatov (2011)). Increased numbers of mobile defect species, such as glissile interstitial loops, also dramatically increases the complexity of the rate equations to be solved. The effects of impurities, alloying elements, and multi-species gas implantation on microstructural evolution have been shown to be significant (Tapasa et al. (2007); Odette and Lucas (2001); Wang et al. (1986); Tanaka et al. (2004)), but these simulations are frequently not within the scope of feasible rate theory simulations.

An alternative approach to solving spatially homogeneous problems of this type, first proposed for any chemical species in Gillespie (1976) and developed further for radiation defects in metals in Marian and Bulatov (2011), avoids many of these problems. In this approach, called stochastic cluster dynamics, the rate equations of traditional rate theory are treated in a homogeneous volume element, but the populations of defects are limited to integer populations within the volume. The reactions between defects such as clustering and dissociation are treated stochastically, using a Monte Carlo algorithm. Thus, the migration of individual defects is ignored and the simulation is able to include large numbers of mobile species which can interact without exponentially increasing computational time.

In order to modify this approach for problems in which nano- and micro-scale spatial dependence is necessary, the present work amends the method of stochastic cluster dynamics by creating several volume elements. Inside each element, the population of defects is assumed to be homogeneously distributed, but migration can occur between elements based on diffusion rates, defect concentrations, and element sizes. In the work of Gillespie (1976), the inclusion of spatial dependence is proposed in this way. Thus, free surfaces, inhomogeneous defect implantation, and other effects can be studied. This approach will be referred to as spatially resolved stochastic cluster dynamics (SRSCD).

1.1.1 Rate theory background

In an infinite, isotropic medium, the evolution of the defect populations can be modeled by tracking individual defect locations and behaviors, or by using a mean field approximation (Stoller et al. (2008)). The use of a mean field approximation allows the simulation to ignore individual defect behavior, thus reducing the computation time, but

assumes spatial homogeneity of defects. In this section, the rate equations for mean field rate theory (MFRT) are described and adapted to the model of the present work.

Here we will present the rate equations for a simple MFRT model containing only two mobile species - vacancies and interstitials - and no helium. This model is taken from Stoller et al. (2008). The evolution of the atomic fraction of SIA (i) or vacancy (v) clusters size n is given by:

$$\begin{aligned}\frac{dC_{vn}}{dt} &= K_{vn}(t) + J_v(n-1,t) - J_v(n,t)\\ \frac{dC_{in}}{dt} &= K_{in}(t) + J_i(n-1,t) - J_i(n,t)\end{aligned} \tag{1.1}$$

where K_{vn} and K_{in} are the generation rate of vacancy and self-interstitial clusters of size n, $J_v(n-1,t)$ and $J_i(n-1,t)$ are the rate of vacancy and interstitial clusters of size $n-1$ converting to size n, and $J_v(n,t)$ and $J_i(n,t)$ are the rate of vacancy and interstitial clusters of size n converting to size $n+1$. Care should be taken here to note that $C_{vn}(t)$ and $C_{in}(t)$ are atomic fractions, which are unitless. Other formulations of MFRT use concentration formulated in defects per volume, and constants are adjusted accordingly (Ortiz and Caturla (2007); Marian and Bulatov (2011)). In a simple model with only Frenkel pair implantation and no migration of clusters, $K_n = 0$ for all n except $n = 1$. The reactions that are accounted for in the growth and annihilation of vacancy and interstitial clusters are as follows:

1. $V_n + V \rightarrow V_{n+1}$ and $I_n + I \rightarrow I_{n+1}$ cluster growth

2. $V_n \rightarrow V_{n-1} + V$ and $I_n \rightarrow I_{n-1} + I$ thermally activated cluster dissociation

3. $V_n + I \rightarrow V_{n-1}$ and $I_n + V \rightarrow I_{n-1}$ vacancy-interstitial annihilation

Taking these reactions into account, the cluster growth terms in equation (1.1) are expressed as

$$\begin{aligned}J_v(n,t) &= P_{vn}(t)C_{vn}(t) - Q_{v(n+1)}(t)C_{v(n+1)}(t)\\ J_i(n,t) &= P_{in}(t)C_{in}(t) - Q_{i(n+1)}(t)C_{i(n+1)}(t)\end{aligned} \tag{1.2}$$

where P_{vn} and P_{in} are the rates of $V_n + V$ and $I_n + I$ clustering respectively, and Q_{vn} and Q_{in} are the rates of recombination and dissociation of vacancy and interstitial clusters of size n. Assuming thermally activated, 3D diffusion, spherical vacancy clusters, and interstitial clusters in the form of circular dislocation loops, the clustering and dissociation rates are given by (Stoller et al. (2008); Katz and Wiedersich (1971)):

$$\begin{aligned}P_{vn}(t) &= \omega n^{\frac{1}{3}} D_v C_v(t)\\ P_{in}(t) &= Z_{\text{int}}\omega_{2\text{D}} n^{\frac{1}{2}} D_i C_i(t)\\ Q_{vn}(t) = Q^i_{vn} + Q^v_{vn} &= \omega n^{\frac{1}{3}}\left(D_i C_i(t) + D_v e^{-\frac{E^v_b(n)}{k_b T}}\right)\\ Q_{in}(t) = Q^v_{in} + Q^i_{in} &= \omega_{2\text{D}} n^{\frac{1}{2}}\left(D_v C_v(t) + D_i e^{-\frac{E^i_b(n)}{k_b T}}\right)\end{aligned} \tag{1.3}$$

Here, $C_{v,i}$ and $D_{v,i}$ are the concentration and diffusion rates of free vacancies and interstitials, respectively. Z_{int} is a constant reflecting the preference of interstitial clusters to absorb other interstitials, commonly taken as $Z_{\text{int}} = 1.15$. $E^{v,i}_b(n)$ is the binding energy of a vacancy or interstitial to a cluster size $n-1$, k_b is Boltzmann's constant, and T is the temperature. The constants ω and ω_i are geometric constants determined by the sink strength of spherical and circular absorbers (Stoller et al. (2008); Brailsford and Bullough (1981)),

$$\omega = \left(\frac{48\pi^2}{\Omega^2}\right)^{\frac{1}{3}}$$
$$\omega_{2\mathrm{D}} = \left(\frac{4\pi}{\Omega b}\right)^{\frac{1}{2}} \tag{1.4}$$

where Ω is an atomic volume and b is the Burgers vector of a dislocation loop.

The rate equations for the time evolution of mobile point defects are more complicated due to the large number of available interactions. The rate equations for single vacancies and single interstitials are as follows (Stoller et al. (2008)):

$$\begin{aligned}
\frac{dC_v}{dt} &= K_v - \mu_R(D_i + D_v)C_i(t)C_v(t) \\
&\quad - \sum_{n=2}^{\infty}\left[P_{vn}(t)C_{vn}(t) + Q^v_{in}(t)C_{in}(t) - Q^v_{vn}(t)C_{vn}(t)\right] \\
&\quad -2P_{v1}(t)C_v(t) + Q^v_{v2}(t)C_{v2}(t) \\
\frac{dC_i}{dt} &= K_i - \mu_R(D_i + D_v)C_i(t)C_v(t) \\
&\quad - \sum_{n=2}^{\infty}\left[P_{in}(t)C_{in}(t) + Q^i_{vn}(t)C_{vn}(t) - Q^i_{in}(t)C_{in}(t)\right] \\
&\quad -2P_{i1}(t)C_i(t) + Q^i_{i2}(t)C_{i2}(t)
\end{aligned} \tag{1.5}$$

where μ_R is a coefficient for the recombination of point defects, given by $\mu_R = \frac{4\pi(r_v + r_i)}{\Omega}$. The final terms in each expression represent the fact that the reactions $v + v \Leftrightarrow 2v$ and $i + i \Leftrightarrow 2i$ add or remove two point defects for each reaction.

Rate equations can vary widely based on the number of defect types present in the system, which defect types are allowed to migrate, and the allowed reactions of the model chosen. As noted by Marian and Bulatov (2011), the number of rate equations required increases exponentially with the number of defect species due to the need to simulate mixed-species clusters. The complexity of the equations also increases as the number of migrating defects of a given species increases. Although grouping schemes exist in rate theory for large clusters in order to speed computation (Golubov et al. (2001)), rate theory simulations are commonly limited to a small number of species and mobile defects.

1.1.2 Reaction rates

In the following sections, it will become useful to discuss the reaction rate of a specific reaction, for example $v+3v \rightarrow 4v$, instead of the entire rate equation governing the population of a single defect type. It can be seen from above that the rate of combination of a mobile point defect i and a sink is given by

$$\textit{reaction rate} = k^2 D_i C_i(t) \tag{1.6}$$

where the sink strength k^2 has units $\frac{1}{m^2}$ and represents the inverse of the square of the mean free path travelled by the point defect before it is absorbed by the stationary defect. It can be seen that the above equations are expressed in this form, with k^2 depending on many factors, including the size and shape of the sinks, whether the sink is also migrating, and the type of migration (1D vs 3D).

When many sink types are present in a system, sink strengths are not independent and have been shown to increase when other sinks are present in high concentrations (Doan and Martin (2003)). This effect is most likely to be significant in the sink strengths of grain boundaries and free surfaces due to the large number of defects within the grains or metal layers. Large planar sinks therefore should not be treated as homogeneously distributed sinks with constant sink strength k^2.

In SRSCD simulations, planar sinks act as 'primary' sinks. These sinks are simulated using boundary conditions on the system by setting the concentration of all defect types equal to 0 at a free surface or perfectly absorbing grain

boundary. All other defects (helium, vacancies, and self-interstitials and their clusters) are simulated as homogeneously distributed 'secondary' sinks with sink strength k^2. This approach has been used in other simulations of sink strength and defect evolution in metals (Doan and Martin (2003); Dunn et al. (2013a); Dudarev et al. (2003)).

1.1.2.1 3D-migrating defects

To calculate the sink strength of defects and defect clusters, it will first be useful to note that for a spherical cluster size n, the radius is given by

$$r = \left(\frac{3n\Omega}{4\pi}\right)^{\frac{1}{3}} \tag{1.7}$$

and for a circular interstitial dislocation loop size n, the radius is given by

$$r = \left(\frac{n\Omega}{\pi b}\right)^{\frac{1}{2}} \tag{1.8}$$

where Ω is the atomic volume and b is the Burgers vector.

For a point defect migrating in three dimensions interacting with a spherical (immobile) sink j, the sink strength is given by (Brailsford and Bullough (1981)):

$$k^2 = \frac{4\pi r_j C_j}{\Omega} \tag{1.9}$$

Thus, the reaction rate for a point defect to interact with a spherical sink is

$$\textit{reaction rate} = \omega n_j^{\frac{1}{3}} D_i C_i C_j \tag{1.10}$$

For a point defect interacting with a circular (immobile) sink j, the sink strength is given by (Stoller et al. (2008)):

$$k^2 = \frac{2\pi r_j C_j}{\Omega} \tag{1.11}$$

Thus, the reaction rate for a point defect to interact with a circular sink is

$$\textit{reaction rate} = \omega_{2\mathrm{D}} n_j^{\frac{1}{2}} D_i C_i C_j \tag{1.12}$$

The reaction rates presented above correspond to the clustering terms in the rate equations presented in the previous section.

In the previous equations, interstitial clusters are treated as circular dislocation loops, and their cross-section for interaction with migrating point defects is adjusted accordingly. However, these defects are assumed to be immobile in most rate theory simulations. By contrast, atomistic studies have shown that the small dislocation loops formed by self-interstitial clusters are in fact very mobile, undergoing one-dimensional glide motion with migration energy less than 0.1 eV (Soneda and De La Rubia (1998); Osetsky et al. (2002)). The following rate equations will account for interactions of multiple mobile defects of varying geometry and migration dimensionality.

The first modification of equations (1.10) and (1.12) occurs when the point defect and the sink are both mobile, spherical defects. In that case, the interaction radius becomes the sum of the two radii of the spherical objects, and the relative diffusion rate is the sum of the two diffusion rates of each defect. This is the same as the sum of the rate at which mobile defect i encounters stationary defect j and mobile defect j encounters stationary defect i. Thus the reaction rate for two spherical defects interacting is

$$\textit{reaction rate} = \omega \left(n_i^{\frac{1}{3}} + n_j^{\frac{1}{3}}\right)(D_i + D_j) C_i(t) C_j(t) \tag{1.13}$$

1.1.2.2 1D-migrating defects

Next, we will treat the case of migrating circular dislocation loops. Since these loops migrate in one dimension, their cross-section for interaction with other defects is different than the case of 3D diffusion. For the case of a point defect migrating in 1D interacting with stationary spherical sinks with absorption cross section σ, the inverse of the average distance travelled before being trapped at a sink is given by (Schilling et al. (1970)):

$$\frac{1}{\lambda} = c\sigma = \frac{C\sigma}{\Omega} \tag{1.14}$$

where the volume concentration of sinks c has been changed to the atomic concentration divided by atomic volume $\frac{C}{\Omega}$. Using $\sigma = \pi r^2$ for the absorption cross section of a spherical sink, the sink strength $k^2 = \frac{1}{\lambda^2}$ is given by (Trinkaus et al. (2000)):

$$k^2 = \left(\frac{\pi r^2 C}{\Omega}\right)^2 \tag{1.15}$$

We next consider a circular dislocation loop i migrating and interacting with an (immobile) point defect sink j. The interaction radius used here is the radius of the circular dislocation loop. Substituting equation (1.15) into equation (1.8), the reaction rate becomes:

$$\textit{reaction rate} = \left(\frac{n_i}{b}\right)^2 D_i C_i(t) C_j(t)^2 \tag{1.16}$$

Note that the reaction rate is quadratic in concentration of the sinks. The fact that we are using the radius of a circular object in the formula for sink strength of a spherical object means that our reaction rate is an upper estimate for the reaction radius of this reaction. However, the form of the reaction rate should remain the same and only vary by a constant due to the shape of the defects involved.

The reaction rate for a circular dislocation loop i interacting with a sessile spherical cluster j is given by the same formula, using the reaction radius as the sum of the radii of the circular loop and the spherical cluster. This reaction rate is given by

$$\textit{reaction rate} = \left[\left(\frac{n_i}{b}\right)^{\frac{1}{2}} + \left(\frac{9\pi n_j^2}{16\Omega}\right)^{\frac{1}{6}}\right]^4 D_i C_i(t) C_j(t)^2 \tag{1.17}$$

Note that the form of this reaction is the same as equation (1.16), with only a change in the radius term accounting for the size of the spherical cluster.

If both the circular dislocation loop i and the spherical defect j migrate, one in 1D and the other in 3D, the reaction rate becomes the sum of the reaction rates for the two types of migration. In this case, we take the sum of the rate for a 1D migrating circular loop to react with a sessile spherical cluster and the rate for a 3D migrating spherical cluster to react with a sessile loop. Thus the reaction rate becomes:

$$\begin{aligned}\textit{reaction rate} = &\left[\left(\frac{n_i}{b}\right)^{\frac{1}{2}} + \left(\frac{9\pi n_j^2}{16\Omega}\right)^{\frac{1}{6}}\right]^4 D_i C_i(t) C_j(t)^2 \\ &+ \left(\omega_{2\mathrm{D}} n_i^{\frac{1}{2}} + \omega n_j^{\frac{1}{3}}\right) D_j C_i(t) C_j(t)\end{aligned} \tag{1.18}$$

Here, the first term accounts for the migration of the dislocation loop and the second term accounts for the migration of the spherical defect. Both terms use the sum of the radii of the dislocation loop and the spherical defect as the reaction radius.

Finally, the reaction rate for two 1D-migrating dislocation loops to interact is again found by summing the rates for each individual loop interacting with the other while the other is stationary. Again, the radius used in the reaction

is the sum of the radii of the two loops. Thus, the reaction rate for two 1D-migrating dislocation loops i and j to combine is given by:

$$\begin{aligned} \textit{reaction rate} = & \frac{(n_i^{\frac{1}{2}} + n_j^{\frac{1}{2}})^4}{b^2}(D_i C_j(t) + D_j C_i(t)) \\ & \times C_i(t) C_j(t) \end{aligned} \tag{1.19}$$

1.1.2.3 Mixed 3D-1D migration

Small SIA clusters have been found to migrate in one dimension along close-packed directions but with occasional changes between equivalent close-packed directions (Soneda and De La Rubia (1998)). Direction changes occur according to an Arrhenius law such that the migration behavior transitions between one-dimensional at low temperatures and three-dimensional at higher temperatures. Assuming 1D and 3D sink strengths are known, the sink strength due to a given sink type i for a defect migrating with this mixed 1D-3D character has been derived in Trinkaus et al. (2002):

$$k_i^2 = \frac{k_{i(1\mathrm{D})}^2}{2}\left(1 + \left(1 + \frac{4}{\frac{l^2 k_{1\mathrm{D}}^2}{12} + \frac{k_{i(1\mathrm{D})}^4}{k_{i(3\mathrm{D})}^4}}\right)^{\frac{1}{2}}\right) \tag{1.20}$$

where $k_{1\mathrm{D}}^2$ is the total sink strength for all sinks using 1D migration, $k_{i(1\mathrm{D})}^2$ and $k_{i(3\mathrm{D})}^2$ are the 1D and 3D sink strengths of sink i, and l is the average distance travelled in one dimension before a direction change.

The sink strength calculated in this way reproduces the 3D sink strength in the limit of frequent direction changes and sparse sinks and reproduces the 1D sink strength in the limit of infrequent direction changes. The use of this sink strength (rather than exclusively 3D or 1D diffusion sink strengths) has been show to not significantly impact the results of helium desorption simulations in iron thin films, and is therefore not currently implemented in SRSCD.

1.1.2.4 2D diffusion on a surface

When simulating defect diffusion and interaction within grain boundaries, the methodology used by Brailsford and Bullough (1981) to calculate the sink strength given in equation (1.9) must be modified to compute reaction rates for defect interaction on two-dimensional surfaces. In this section, the same methodology is applied to solve for sink strength k^2 of a circular sink on a two-dimensional surface, in order to extend this methodology to multiple diffusing defects on a surface. Similarly to the work of Brailsford and Bullough (1981), we consider two concentration rate equations $c_e(t)$ and $c_h(t)$, the exact and homogeneous distributions of a given defect type, respectively. Assuming a constant rate of implantation K (in units of defects·m^{-2}s^{-1}), the rate equations for the population evolution of these two formulations for concentration are:

$$\frac{dc_e}{dt} = K + D\nabla^2 c_e \ , \ \frac{dc_h}{dt} = K - Dk_s^2 c_h \tag{1.21}$$

The rate equation for c_e uses boundary conditions to treat the impact of sinks, while c_h uses a homogenized sink strength term k_s^2 to treat the presence of sinks. Therefore, k^2 is found by solving the rate equation for c_e explicitly around a single sink and setting the concentrations c_e and c_h equal.

The solution to $\frac{dc_e}{dt}$ is found in a two-dimensional surface around a circular sink with radius r_s inside a circular area with outer radius r_o. The value of r_o is chosen such that the area A enclosed by r_o, given by $A = \pi r_o^2$, is the inverse of the concentration of sinks in the global medium, $c_s = \frac{1}{\pi r_o^2}$. Assuming cylindrical symmetry, the spatial derivatives simplify to:

$$\frac{dc_e}{dt} = K + D\left(\frac{1}{r}\frac{d}{dr}r\frac{dc_e}{dr}\right) \tag{1.22}$$

which has steady-state $\left(\frac{dc_e}{dt} = 0\right)$ solution given by:

$$c_e(r) = \alpha_1 \ln(r) + \alpha_2 - \frac{Kr^2}{4D} \tag{1.23}$$

where α_1 and α_2 are integration constants. By imposing the boundary conditions $c_e(r_s) = 0$ and $\frac{dc_e}{dr}(r_o) = 0$, this equation simplifies to:

$$c_e(r) = \frac{K}{4D}\left[2r_o^2 \ln\left(\frac{r}{r_s}\right) + (r_s^2 - r^2)\right] \tag{1.24}$$

The steady-state solution to $\frac{dc_h}{dt} = 0$ is given by $c_h = \frac{K}{Dk_s^2}$. To solve for k^2, the concentration $c_e(r_o)$ is set equal to c_h, giving:

$$k_s^2 = \frac{4}{2r_o^2 \ln\left(\frac{r_o}{r_s}\right) + (r_s^2 - r_o^2)} \tag{1.25}$$

Substituting $c_s = \frac{1}{\pi r_o^2}$, this simplifies to:

$$k_s^2 = \frac{4\pi c_s}{2\ln\left(\frac{r_o}{r_s}\right) + \left(\frac{r_s^2}{r_o^2} - 1\right)} \tag{1.26}$$

To simplify this equation, we require that $r_o >> r_s$, allowing the second term in the denominator to be neglected, giving:

$$k_s^2 = \frac{4\pi c_s}{2\ln\left(\frac{r_o}{r_s}\right) - 1} \tag{1.27}$$

Substituting the sink strength k_s^2 into $\frac{dc_h}{dt}$ in equation (1.21), we get the following rate equation for the evolution of the population of defects in a homogeneous medium with sinks:

$$\frac{dc_h}{dt} = K - \frac{4\pi D c_s c_h}{2\ln\left(\frac{r_o}{r_s}\right) - 1} \tag{1.28}$$

Noting again that $c_s = \frac{1}{\pi r_o^2}$, this equation can be expressed as a function of c_s, c_h, and r_s:

$$\frac{dc_h}{dt} = K - \frac{4\pi D c_s c_h}{2\ln\left(\sqrt{\frac{1}{\pi c_s r_s^2}}\right) - 1} \tag{1.29}$$

In typical mean field rate theory problems, instead of treating one defect type in a concentration field of sinks, we typically simulate two or more types of defects, each acting as a sink for the other defects. Therefore, in a model two-species system of vacancies and self-interstitials, c_v and c_i, with a reaction distance r and diffusivities D_v and D_i, the total equation for the evolution of the system is given by:

$$\begin{aligned}
\frac{dc_v}{dt} &= K_v - \frac{4\pi D_v c_v c_i}{2\ln\left(\sqrt{\frac{1}{\pi c_i r^2}}\right) - 1} - \frac{4\pi D_i c_v c_i}{2\ln\left(\sqrt{\frac{1}{\pi c_v r^2}}\right) - 1} \\
\frac{dc_i}{dt} &= K_i - \frac{4\pi D_v c_v c_i}{2\ln\left(\sqrt{\frac{1}{\pi c_i r^2}}\right) - 1} - \frac{4\pi D_i c_v c_i}{2\ln\left(\sqrt{\frac{1}{\pi c_v r^2}}\right) - 1}
\end{aligned} \tag{1.30}$$

Note that in equation (1.30), two terms are added to each rate equation - one representing the rate of vacancy diffusion and capture by stationary self-interstitials, and one representing mobile self-interstitial diffusion and capture by stationary vacancies. This method assumes that, when both defect types are mobile, the reaction rate for interaction between mobile defects can be deconvoluted into two reaction rates in which one defect type is mobile and one defect type is stationary.

The validity of the model presented in equation (1.30) is investigated here by comparison with a 2D OKMC simulation including single vacancies and single interstitials. The only allowed reactions in the OKMC simulation are implantation, diffusion, and annihilation when two defects come within a reaction radius of each other. All parameters are the same in the OKMC and MFRT simulations. Using a single implantation rate $K_v = K_i = K$, the population of vacancies and interstitials as a function of total dose is compared at three different temperatures (yielding different diffusivities D_v and D_i) between OKMC and MFRT simulations. Figure 1.1 shows these results. Both steady-state and transient populations show a good fit between OKMC and MFRT simulations.

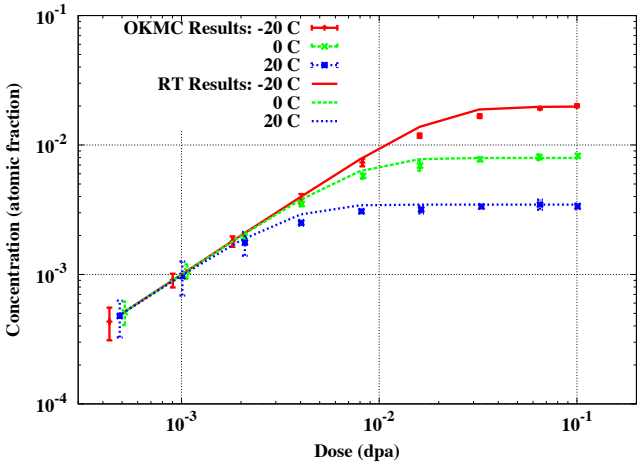


Figure 1.1: Comparison of mean field rate theory and two-dimensional OKMC simulation of point defect diffusion and annihilation using the 2D reaction rate given in equation (1.30). Defect diffusivities are taken from values for single vacancies and self-interstitials in α-Fe at three different temperatures.

As described in Dunn et al. (2013b), the rate equations (1.30) are converted to units of s^{-1} and the reaction rate for interaction between two defect types i and j used in SRSCD is given by:

$$\text{reaction rate } (\text{s}^{-1}) = \frac{4\pi N_i N_j}{A} \times \left[\frac{D_i}{2\ln\left(\sqrt{\frac{A}{\pi N_j}}\left(\frac{1}{r_i+r_j}\right)\right) - 1} + \frac{D_j}{2\ln\left(\sqrt{\frac{A}{\pi N_i}}\left(\frac{1}{r_i+r_j}\right)\right) - 1} \right] \tag{1.31}$$

where N_i, N_j, D_i, and D_j are the numbers of defects type i and j and their diffusivities, respectively, inside the grain boundary element in SRSCD. A is the area of the grain boundary element. The values r_i and r_j are the radii of each defect type, given by

$$r_i = \left(\frac{3n_i\Omega}{4\pi}\right)^{\frac{1}{3}} \tag{1.32}$$

such that $r_i + r_j$ is the reaction distance between the two defects. The value n_i is the number of vacancies or interstitials making up defect type i. This approximation assumes that, although the defects diffuse in a two-dimensional surface, they still take the form of spheres.

1.1.2.5 Spatially resolved rate equations

So far, reaction rates have been derived for a single finite volume element which is assumed to be spatially homogeneous throughout. This is an approximation of an infinite medium. However, a physically representative prediction of defect evolution in a heterogeneous microstructure, e.g. polycrystals, nano-structured materials, and nano-laminates, can only be accessed through a spatially resolved method. Indeed, the behavior of defects in the neighborhood of grain boundaries, dislocations, and hetero-interfaces is known to be different from the bulk (Zinkle and Singh (2000); Ryazanov et al. (1996); Heinisch et al. (2006, 2007); Demkowicz et al. (2007); Dunn et al. (2013a); Demkowicz et al. (2010); Hattar et al. (2008); Zhernenkov et al. (2011)). While traditional approaches use averaging arguments to treat the impact of grain boundaries and dislocations in homogeneous models, it is likely that in nano-structured materials this approximation can no longer be made. In addition, defect populations resulting from cascade implantation have been shown to depend on the spatial resolution of the initial cascade state (Soneda and De La Rubia (1998); Caturla et al. (2000); Ortiz and Caturla (2007)). It is therefore necessary to develop a method that can simulate both large timescales and the spatially resolved structures of nano-structured materials.

It was noted by the original author of stochastic rate theory (Gillespie (1976)) that the method could approximate spatial resolution by creating several volume elements instead of a single one. Within each volume element, the

system is assumed to be 'well-mixed' and therefore spatially homogeneous, but differences in numbers of defects occur between elements. The reaction rates of combination and dissociation are calculated within each element according to the rate equations presented above. To calculate the reaction rates of migration between elements, we begin with a standard gradient-driven diffusion equation (in units of atomic fraction):

$$\frac{dC}{dt} = \nabla \cdot (D \cdot \nabla C) + f(x,t) \tag{1.33}$$

where $f(x,t)$ accounts for all of the terms discussed in the previous sections.

To convert this equation to a reaction rate for a finite volume, we first integrate over the volume element and apply the divergence theorem. Neglecting $f(x,t)$, this gives:

$$\int_V \frac{dC}{dt} dV = - \oint_S (D \cdot \nabla C) \cdot \vec{n} dS \tag{1.34}$$

where the second integral is now a surface integral over the boundary of the element. Multiplying by $\frac{1}{\Omega}$, the left hand side of equation 1.34 is now $\frac{dN}{dt}$ where N is the absolute number of defects of this type in the volume element. We will refer to this element as i and its neighbors as j, with $j \in [1,6]$. Assuming the element is rectangular, diffusion is isotropic and constant, and approximating ∇C using the neighboring volume elements, we approximate the surface integral and get:

$$\frac{dN_i}{dt} = \sum_{\text{neighbors}} \left(\frac{D A_{ij} (N_j - N_i)}{V L_{ij}} \right) \tag{1.35}$$

where A_{ij} is the area of the facet connecting elements i and j and L_{ij} is the distance between the centers of the two elements. Thus the reaction rate for a defect migrating from from element i to element j is given by

$$\textit{reaction rate} = D A_{ij} \frac{N_i - N_j}{V L_{ij}} \tag{1.36}$$

This reaction is treated similar to all other reactions listed above, with a rate (in units of s^{-1}) for a single defect to migrate from volume element i to j. This approach mirrors that of a finite-element or finite-difference approximation for spatial resolution of mean field rate theory equations, which has been carried out for some systems (Ortiz and Caturla (2007); Ortiz et al. (2007); Dunn et al. (2013a); Xu et al. (2007)). The reaction rate associated with inter-element diffusion is included in Table 1.1.

Using spatially resolved finite volumes in this way, materials such as thin films with free surfaces can be approximated by holding the concentration of defects outside the material equal to zero at the boundaries on one axis and applying periodic boundary conditions on other axes. Cascade implantation can also be simulated by implanting all cascade products into one volume element. The optimal method for representing cascade damage in this scheme is still an open question, and three methods for doing so are presented in the Section 1.2.

1.1.3 Reaction rates for discrete elements

The central idea of stochastic rate theory is to solve the rate equations representing time evolution of the concentration of various defect types stochastically in a finite volume element. Given a volume element with volume V, if the concentration of defects in the volume is C (in atomic fraction), then the total number of defects in that volume is $N = \frac{CV}{\Omega}$. This number is an integer value, and reaction rates for defects not present in the volume are not calculated.

Given an initial set of defects within a volume element, assuming spatial homogeneity within that element, the reaction rates for any two defects to combine or any one defect to dissociate are given by the same reaction rates as above, which are converted to rates for a finite volume element by multiplying them by $\frac{V}{\Omega}$:

$$\left(\frac{dC}{dt} \right) \left(\frac{V}{\Omega} \right) = \frac{dN}{dt} \tag{1.37}$$

Converting concentration C into $\frac{N\Omega}{V}$ and multiplying all reaction rates by $\frac{V}{\Omega}$, we convert all reaction rates to finite-volume rates with units s^{-1}. Table 1.1 shows these rates. It should be noted that other formulations of stochastic

rate theory (Marian and Bulatov (2011)) have used concentration rate equations that are in units of defects·m^{-3}, so these equations are only multiplied by V.

1.1.4 Monte Carlo algorithm

In order to solve the rate equations presented here in a stochastic way, reactions are chosen and time is iterated in a stochastic manner, instead of using a standard finite-difference time iteration formulation as in the case of rate theory. It has been proven (Gillespie (1976)) that this approach correctly solves the master equation for the time evolution of the entire system.

Given an initial set of defects in the system, all possible reactions and their rates can be calculated using the rates in Table 1.1. Thus, unlike mean field rate theory, only reaction rates for defects present in the system are calculated and only integer numbers of defects are treated. Each reaction μ has a reaction rate a_μ in the system. Therefore, the total reaction rate for the system (in reactions per second) for all reactions is

$$a = \sum_{\mu} a_\mu \tag{1.38}$$

Thus, as in standard Monte Carlo techniques, the probability that the first reaction after time t in the system will occur between time $t + \tau$ and $t + \tau + \delta\tau$ is given by

$$P_1(\tau)\delta\tau = ae^{-a\tau}\delta\tau \tag{1.39}$$

and the probability that the next reaction will be reaction μ is

$$P_2(\mu) = \frac{a_\mu}{a} \tag{1.40}$$

Therefore, in the simulation, the amount of time that passes before the next reaction is carried out is chosen stochastically by choosing a random number $r_1 \in (0, 1)$ and iterating time by timestep

$$\tau = \frac{1}{a} \ln\left(\frac{1}{r_1}\right) \tag{1.41}$$

and reaction μ is carried out, with μ chosen by choosing a second random number $r_2 \in (0, 1)$ and finding μ such that

$$\sum_{\nu=1}^{\mu-1} a_\nu < r_2 a < \sum_{\nu=1}^{\mu} a_\nu \tag{1.42}$$

A derivation of this algorithm can be found in Gillespie (1976). After each timestep, the number of defects in the simulation is updated for each defect type involved in the reaction chosen. The reaction rates for all relevant reactions are subsequently updated. Thus, if a defect type disappears from the simulation during a reaction, all reaction rates associated with that defect type are removed from the list of possible reactions that the system can choose. This greatly reduces the amount of computation required compared to mean field rate theory, and allows arbitrarily sized defects to migrate without creating rate equations that are unmanageable.

Due to the spatial resolution of the system, rates for reactions are calculated within each volume element and between each adjacent volume element for migration reactions. However, the time iteration is carried out for the entire system at once. Thus, only one possible reaction is chosen from among all volume elements in the system per timestep.

1.2 Cascade implantation in SRSCD

The implementation of damage due to displacement cascades is an important consideration in designing models of radiation damage. Displacement cascades are caused by energetic collisions between incident particles (neutrons, heavy ions) and lattice atoms that are energetic enough that the displaced lattice atom can cause several

Reaction	Reaction rate (s^{-1})
Clustering reactions	
$V_n + V_m \rightarrow V_{n+m}$	$\omega(n^{\frac{1}{3}} + m^{\frac{1}{3}})(D_{Vn} + D_{Vm})N_{Vn}(t)N_{Vm}(t)\frac{\Omega}{V}$
$V_n + I_m \rightarrow V_{n-m}$ or I_{m-n} (3D SIA)	$\omega(n^{\frac{1}{3}} + m^{\frac{1}{3}})(D_{Vn} + D_{Im})N_{Vn}(t)N_{Im}(t)\frac{\Omega}{V}$
$V_n + I_m \rightarrow V_{n-m}$ or I_{m-n} (1D SIA)	$\left[\left(\frac{m}{b}\right)^{\frac{1}{2}} + \left(\frac{9\pi n^2}{16\Omega}\right)^{\frac{1}{6}}\right]^4 D_{Im}N_{Im}(t)N_{Vn}(t)^2\frac{\Omega^2}{V^2}$ $+ \left(\omega_{2\mathrm{D}}m^{\frac{1}{2}} + \omega n^{\frac{1}{3}}\right)D_{Vn}N_{Im}(t)N_{Vn}(t)\frac{\Omega}{V}$
$I_n + I_m \rightarrow I_{n+m}$ (3D + 3D SIA)	$Z_{\mathrm{int}}\omega(n^{\frac{1}{3}} + m^{\frac{1}{3}})(D_{In} + D_{Im})N_{In}(t)N_{Im}(t)\frac{\Omega}{V}$
$I_n + I_m \rightarrow I_{n+m}$ (3D + 1D SIA)	$Z_{\mathrm{int}}^4\left[\left(\frac{m}{b}\right)^{\frac{1}{2}} + \left(\frac{9\pi n^2}{16\Omega}\right)^{\frac{1}{6}}\right]^4 D_{Im}N_{Im}(t)N_{In}(t)^2\frac{\Omega^2}{V^2}$ $+ \left(\omega_{2\mathrm{D}}m^{\frac{1}{2}} + \omega n^{\frac{1}{3}}\right)D_{In}N_{Im}(t)N_{In}(t)\frac{\Omega}{V}$
$I_n + I_m \rightarrow I_{n+m}$ (1D + 1D SIA)	$Z_{\mathrm{int}}^4\frac{\left(n^{\frac{1}{2}}+m^{\frac{1}{2}}\right)^4}{b^2}\left(D_{In}N_{Im}(t) + D_{Im}N_{In}(t)\right)N_{In}(t)N_{Im}(t)\frac{\Omega^2}{V^2}$
2D Diffusion on a surface	
$i + j$ (clustering or annihilation)	$\frac{4\pi N_i N_j}{A}\left[\frac{D_i}{2\ln\left(\sqrt{\frac{A}{\pi N_j}}\left(\frac{1}{r_i+r_j}\right)\right)-1} + \frac{D_j}{2\ln\left(\sqrt{\frac{A}{\pi N_i}}\left(\frac{1}{r_i+r_j}\right)\right)-1}\right]$
Dissociation reactions	
$V_n \rightarrow V + V_{n-1}$	$\omega n^{\frac{1}{3}}D_V e^{-\frac{E_b(n)}{k_b T}}N_{Vn}(t)$
$I_n \rightarrow I + I_{n-1}$ (3D SIA)	$\omega n^{\frac{1}{3}}D_I e^{-\frac{E_b(n)}{k_b T}}N_{In}(t)$
$I_n \rightarrow I + I_{n-1}$ (1D SIA)	$\omega_{2\mathrm{D}}n^{\frac{1}{2}}D_I e^{-\frac{E_b(n)}{k_b T}}N_{In}(t)$
Migration reactions	
$X^i \rightarrow X^j$ (volume element i to j)	$D_X A_{ij}\frac{N_X^i(t) - N_X^j(t)}{V L_{ij}}$
Implantation reactions	
$0 \rightarrow V + I$	$(\mathrm{dpa\ rate})\left(\frac{V}{\Omega}\right)$
$0 \rightarrow$ (*cascade*)	$\left(\frac{\mathrm{dpa\ rate}}{N_{\mathrm{displaced}}}\right)\left(\frac{V}{\Omega}\right)$

Table 1.1: Reaction rates for vacancy and interstitial reactions in a finite volume element, size V. N_i indicates the absolute number of species i present in the volume. All rates are in units of s^{-1}. 3D SIA indicates that the SIA cluster is approximated as a sphere that migrates in three dimensions, 1D SIA indicates that the SIA cluster is approximated as a circular dislocation loop that migrates in one dimension. In the migration reaction, species X migrates from volume element i to j, with boundary surface area A_{ij} and separation L_{ij}.

other lattice atoms to also displace from their lattice site. Therefore, several defects are created in a small volume, and in many cases defect clusters are created directly inside cascades (Soneda and De La Rubia (1998)). The stable defects created in displacement cascades have been modeled in several materials using atomistic simulations (Soneda and De La Rubia (1998); Stoller et al. (1997); Stoller and Calder (2000)). Larger-scale models must incorporate these defects into their model appropriately in order to accurately predict the resulting damage accumulation. OKMC models are able to directly place the defects generated by cascades into a simulation volume, due to the spatial resolution afforded by such methods (Becquart et al. (2010); Jansson and Malerba (2013, 2014); Soneda et al. (2003)). By contrast, typical mean-field cluster dynamics models do not include spatial resolution, making incorporation of cascade damage more difficult. Several techniques for overcoming this obstacle have been developed, including using OKMC to generate irradiation source terms that are input into CD models (Jourdan and Crocombette (2012); Meslin et al. (2008); Ortiz and Caturla (2007)). Cascade damage in SRSCD has been implemented using three techniques, with the intent of preserving the spatial correlation between defects inside cascades while still simulating large doses and volumes.

1.2.1 Method 1: single volume element implantation

The first and most direct method of simulating cascade damage in SRSCD is to simply place all stable defects generated by atomistic simulations into a single volume element in SRSCD. However, when using this method, care must be taken to choose the appropriate volume element length. A 20 keV cascade in α-Fe has a diameter of approximately 10 nm. The spatial proximity of defects inside the cascade determines the likelihood of spatially correlated reactions between defects as they diffuse after initial implantation. However, SRSCD assumes that all defects inside a volume element are homogeneously distributed and their spatial correlation is not accounted for beyond the length scale of the volume element. Therefore, the volume element size chosen must roughly match the size of the cascade in order to produce the correct probability of spatially correlated reactions between cascade defects. Elements that are too large will assume a low initial density of defects and too few spatially correlated reactions, and elements that are too small will assume a high initial density of defects and too many spatially correlated reactions. Thus, mesh convergence cannot be obtained using this method for cascade implantation.

To compare the performance of this method to OKMC, the results of cascade implantation in Cu are compared to those in Caturla et al. (2000). In these simulations, 20 keV cascades from a database produced using molecular dynamics simulations are implanted into copper at 10^{-4} dpa·s^{-1}. In this simulation, vacancy clusters up to size 4 and SIA clusters up to size 60 are mobile, with large self-interstitial clusters migrating in one dimension.

The simulation parameters are listed in Table 1.2. All simulation parameters used in this work are the same as reported in Caturla et al. (2000). Migration energies of vacancy clusters are taken from Sabochick and Yip (1988), single-interstitials from Corbett et al. (1959), and small SIA clusters from Schober and Zeller (1978). Larger interstitial clusters are assumed to maintain the same migration energy but decrease their diffusion prefactor as the size of the cluster grows. The binding energy of vacancy clusters is found by fitting the values for small vacancy clusters from Sabochick and Yip (1988) to a fitting function, using the formation energy of a vacancy as the binding energy of an infinite sized cluster. The binding energy of small SIA clusters is taken from Schober and Zeller (1978), while the binding energy of larger SIA clusters is assumed to be the formation plus migration energy of a single SIA.

In order to reproduce this simulation using SRSCD, 20 keV cascades from Caturla et al. (2000) are converted into a list of initial defects for each cascade. These were then implanted into the volume elements in the simulation with a rate given by the dpa rate and the number of defects in the cascade, according to the equation in Table 1.1. Since the entire cascade is implanted at the same time into a single volume element, and SRSCD assumes that defects are homogeneously distributed within a volume element, the size of the volume elements used determines the local concentration of defects in a cascade. Therefore, the spatial resolution of this method is necessary because the volume element size must match approximately the size of the cascade to provide the correct initial concentration of defects.

The effect of changing the size of the volume element on the profile of vacancy cluster concentrations is shown in Figure 1.2. For large meshes, the size of a volume element is much larger than the size of a cascade and the simulation produces mesh-independent results that do not match the results of OKMC. As the mesh gets smaller, the initial defect concentration increases dramatically, changing the initial clustering and annihilation rates for the defects in a given cascade. The optimal mesh size for this simulation was found to be approximately 10 nm, which is similar to the size of a 20 keV cascade.

The results of SRSCD simulations of cascade damage accumulation using 10 nm volume elements and the comparison to the data of Caturla et al. (2000) are shown in Figure 1.3. The quantitative results of this simulation differ

Parameters used	
temperature	340 K
atomic volume	$1.17 \cdot 10^{-2}$ nm^3
Burgers vector	0.36 nm
dpa rate	$1 \cdot 10^{-4}$ $\frac{\text{dpa}}{s}$
grain size	1 μm
Diffusion rates	$D = D_0 e^{-\frac{E_m}{k_b T}}$
single vacancy	$E_m = 0.72$ eV, $D_0 = 2.5 \cdot 10^{13}$ $\frac{\text{nm}^2}{s}$
2-v cluster	$E_m = 0.55$ eV, $D_0 = 3.6 \cdot 10^{13}$ $\frac{\text{nm}^2}{s}$
3-v cluster	$E_m = 0.56$ eV, $D_0 = 1.2 \cdot 10^{13}$ $\frac{\text{nm}^2}{s}$
4-v cluster	$E_m = 0.38$ eV, $D_0 = 1.4 \cdot 10^{13}$ $\frac{\text{nm}^2}{s}$
larger vacancy cluster ($n > 4$)	(immobile)
single interstitial	$E_m = 0.13$ eV, $D_0 = 2 \cdot 10^{11}$ $\frac{\text{nm}^2}{s}$
2-i cluster	$E_m = 0.11$ eV, $D_0 = 1 \cdot 10^{11}$ $\frac{\text{nm}^2}{s}$
3-i cluster	$E_m = 0.2$ eV, $D_0 = 6.6 \cdot 10^{10}$ $\frac{\text{nm}^2}{s}$
4-i cluster	$E_m = 0.1$ eV, $D_0 = 5 \cdot 10^{10}$ $\frac{\text{nm}^2}{s}$
larger interstitial cluster ($n > 4$)	$E_m = 0.1$ eV, $D_0 = \frac{2 \cdot 10^{11}}{n}$ $\frac{\text{nm}^2}{s}$
Binding energies	
2-v cluster	$E_b^v(2) = 0.05$ eV
3-v cluster	$E_b^v(3) = 0.15$ eV
4-v cluster	$E_b^v(4) = 0.28$ eV
5-v cluster	$E_b^v(5) = 0.65$ eV
larger vacancy cluster ($n > 5$)	$E_b^v(n) = 1.2 - 2.121(n^{\frac{2}{3}} - (n-1)^{\frac{2}{3}})$ eV
small interstitial cluster	$E_b^i(2-4) = 1.16$ eV
larger interstitial cluster ($n > 4$)	$E_b^i(n) = 2.62$ eV

Table 1.2: Material and experimental constants used in the simulation of Caturla et al. (2000) for Cu.

from those of Caturla, but the qualitative trends match. The concentration of vacancy clusters increases linearly with dose, while the concentration of single vacancies reaches an early saturation and decreases slowly. The concentration of visible (to TEM, taken as clusters > 1 nm in diameter) clusters also increases linearly with dose. The vacancy cluster population profile also qualitatively matches the results of Caturla et al, although the concentration of small vacancy clusters of size < 10 differs from the results of Caturla. The differences between SRSCD and OKMC results shown here are likely due to different initial defects included in the cascade, as these simulations may not have used the exact same list of defects as used by Caturla et al.

1.2.2 Method 2: distributed cascade implantation

In order to more accurately simulate cascade implantation in metals, a second 'multi-element' method has been developed. Here, the coordinates of each defect in the cascade relative to the center of the cascade are recorded. The center of the cascade is randomly placed within the volume of the simulation. The coordinates of each defect in the cascade are then used to identify which volume element each defect is implanted into. Thus, as the mesh size decreases, the cascade is spread among more volume elements and the concentration of defects within each element remains approximately constant. This method allows both stability of the solution and the ability to distribute the defects in the cascade in a non-homogeneous manner throughout the simulation volume. Note that within each volume element, defect distribution is still assumed to be homogeneous and defects do not have individual positions. An example of the spatial distribution of vacancy clusters using this method for cascade implantation to 10^{-3} DPA in Fe is shown in Figure 1.4.

The dependence of cascade implantation results on mesh size is shown for the single-element and multi-element methods in Figure 1.5. At large mesh sizes, both methods underestimate the concentration of defects in the cascades after initial implantation. As mesh size decreases, the single-element method first overestimates then underestimates the vacancy cluster concentration, while the multi-element method provides a much more stable solution. The differences between the two methods are due to the fact that in the single-element method, defect density within the cascade increases as mesh size decreases while in the multi-element method, defect density remains relatively

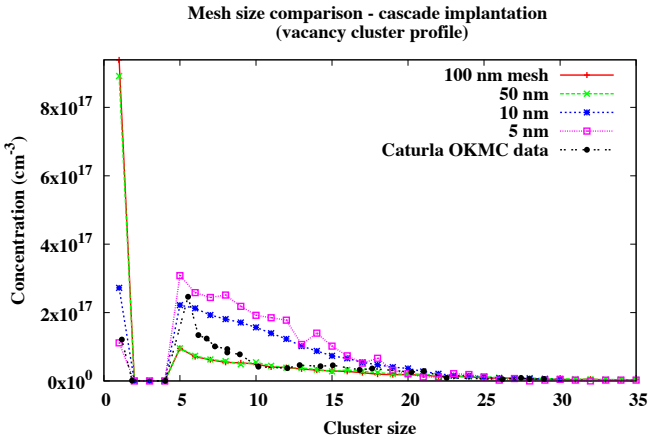


Figure 1.2: Comparison of vacancy cluster profiles with varying mesh sizes at 8×10^{-4} dpa. The mesh size is critical in determining the initial concentration of defects in a cascade, and thus the mesh must match the cascade size to produce qualitatively correct results.

constant once the mesh size is smaller than the cascade size.

The lack of true convergence of the multi-element method is due to the fact that the reaction rates presented above are derived for a continuous distribution of defects, and therefore some systematic errors occur when the number of defects in a given volume element approaches zero. However, this method does successfully simulate cascade implantation with a range of volume element sizes over a large range of DPA. This method is not currently implemented in SRSCD due to the difficulty associated with distributing defects over several volume elements in a parallel simulation using several processors.

1.2.3 Method 3: adaptive meshing

Several challenges remain in order for SRSCD to accurately reproduce experimental damage accumulation in bulk metals. Spatial resolution on the order of a few nanometers is necessary in SRSCD to adequately simulate cascade damage accumulation in metals (Dunn et al. (2013b, 2014)). However, simulations of large volumes can become computationally prohibitive due to the high spatial resolution necessary to simulate cascade damage.

Here, a novel scheme for simulating radiation damage accumulation in metals is developed that addresses how to introduce cascades in a numerically efficient fashion while maintaining the spatial resolution necessary for accurate damage evolution. An adaptive meshing scheme is introduced to provide both the high spatial resolution necessary for cascade implantation as well as computational efficiency for large volume and high DPA simulations. Due to the computational efficiency of SRSCD, simulation volumes are large enough in this work to compare both vacancy cluster and self-interstitial loop populations to experimental results.

The amount of computation per step in the SRSCD algorithm is strongly dependent on the number of possible reactions in the system. An estimate of the upper bound for the number of reactions in a volume element containing n total mobile defect types and m total stationary defect types can be obtained for each of the four broad classes of reactions treated in this simulation: clustering, dissociation, trapping at sinks and traps (with k sink/trap types treated), and element-to-element diffusion.

For binary clustering reactions, there are at most $\binom{n}{2}$ possible reactions between distinct mobile defect types, n possible reactions representing two defects of the same type combining, and mn possible reactions between mobile defect species and immobile defect species. Therefore, the number of clustering reactions dramatically increases when several mobile species are present because the number of clustering reactions varies quadratically with n and linearly with m. Dissociation reactions are typically limited to $m+n$ total reactions, as each defect type has at most one possible dissociation reaction associated with it. The number reactions representing sinks and traps is typically kn, representing one reaction for each mobile defect type interacting with each sink or trap type. Finally, diffusion of mobile defects between volume elements leads to at most $6n$ reactions, representing six diffusion directions for each mobile defect species. Table 1.3 shows the upper bound estimate of the number of reactions present in a volume element.

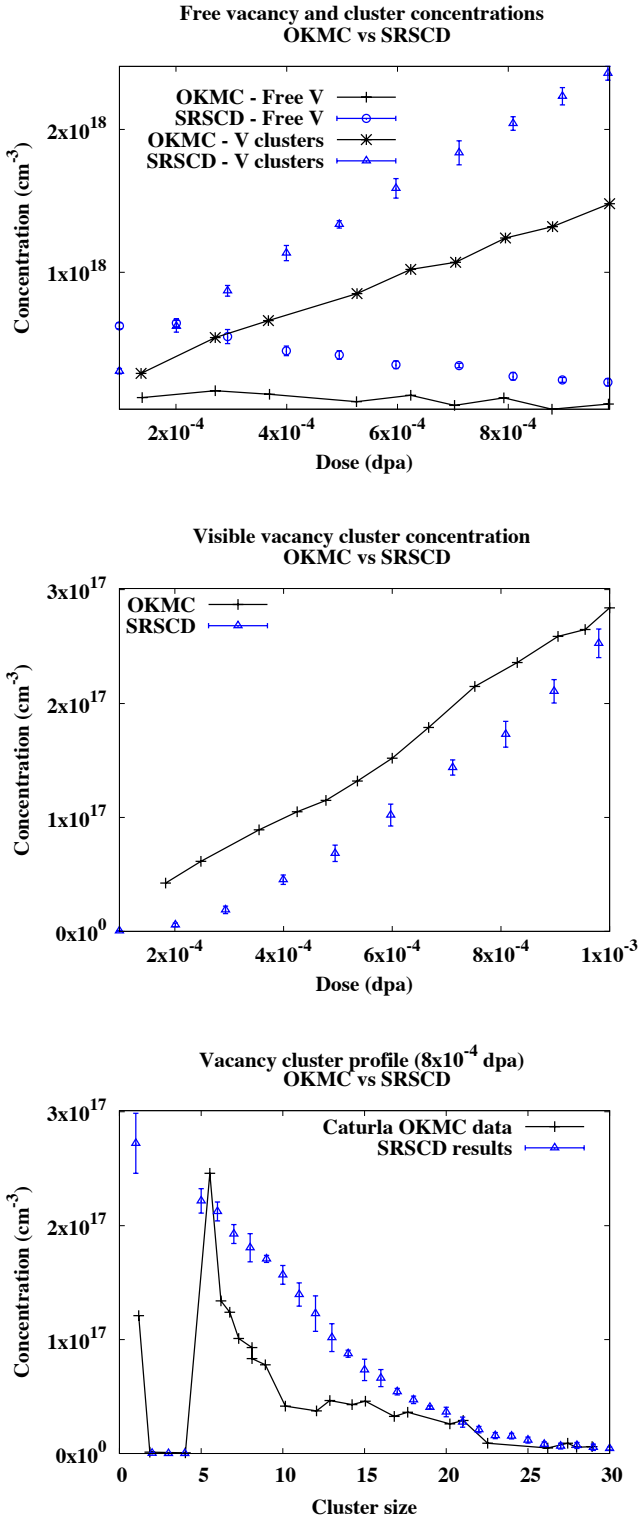


Figure 1.3: SRSCD results compared to the OKMC results of Caturla et al Caturla et al. (2000) for 20 keV cascades implanted in Cu at a dpa rate of 10^{-4} dpa/s. The qualitative results match the results of OKMC.

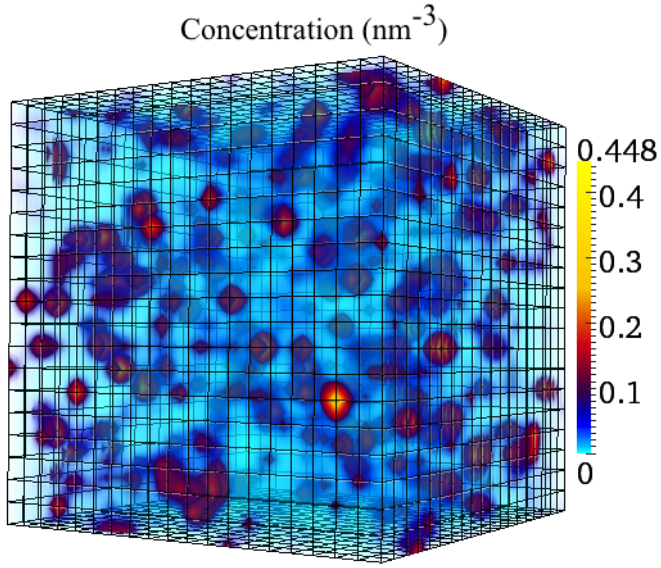


Figure 1.4: Vacancy cluster distribution inside bulk Fe under cascade implantation to 10^{-3} DPA at 273 K using SRSCD. Volume elements (shown with grid lines) are cubes with side length 5 nm.

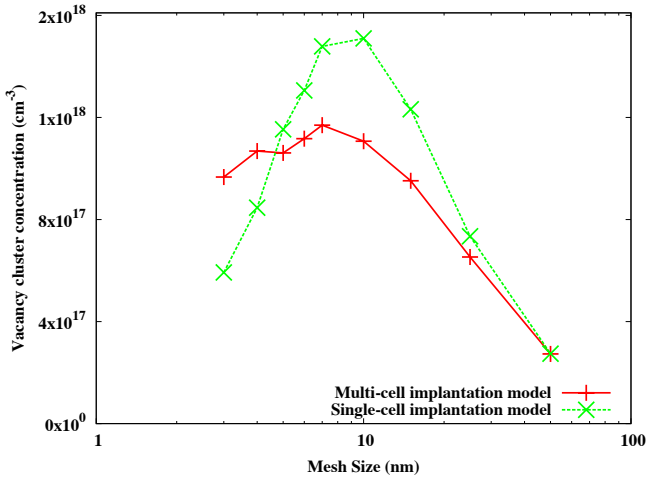


Figure 1.5: Mesh convergence using the 'multi-cell' cascade implantation method compared 'single-cell' method. Results shown are for 20 keV cascade implantation in Fe at 373 K to 10^{-3} DPA.

If a large number of mobile defect species are present within a volume element, most reactions are associated with defect clustering. Conversely, if a small number of defects are spread out over many elements, diffusion, dissociation, and trapping at sinks are the most common reactions. The number of mobile defect species in a given simulation depends strongly on both the parameter set used to characterize defect behavior and the simulated irradiation conditions.

Simulations with larger volume elements have fewer total reactions and take less computation per step than simulations with many smaller volume elements. However, as described previously, fine spatial resolution is necessary to adequately simulate displacement cascade implantation in metals due to spatially correlated reactions between defects in cascades. In this section, an adaptive meshing method is implemented to allow both computational efficiency for simulations of larger volumes and accurate treatment of cascade implantation.

1.2.3.1 Adaptive meshing: formulation

Figure 1.6 shows a two-dimensional schematic of the adaptive meshing scheme used here in a single volume element. The simulation is carried out using a coarse mesh with volume elements of length $L_c = 80 - 125$ nm. When a cascade implantation event is chosen, a separate fine mesh is created with element length $l_f = 5$ nm. In this work, the fine mesh created contains 6 elements per side. The size of the fine mesh was chosen to best reproduce the results given by the uniform meshing method while maintaining reasonable computation times and keeping the size of the fine mesh significantly smaller than the size of a coarse mesh element.

This fine mesh is first populated with existing defects from the coarse mesh randomly so that the concentration of defects of various types is the same in the fine mesh and the coarse mesh. The cascade defects are then introduced into the fine mesh, using the coordinates of the cascade defects relative to the cascade center to choose which volume element in the fine mesh each defect is implanted into. This way the spatial correlation between the defects in the cascade is preserved with resolution determined by the fine mesh element length.

After implantation, the entire system (both fine mesh and coarse mesh) is simultaneously evolved in time using the Monte Carlo algorithm, allowing defects to diffuse between the fine mesh and coarse mesh. Once the total reaction rate for all reactions within the fine mesh reaches a low threshold, all of the remaining defects in the fine mesh are deposited into the coarse mesh and the fine mesh is subsequently deleted. At this point, the spatial correlation between defects in the cascade provided by the fine mesh is lost and all defects are assumed to be homogeneously distributed within the coarse mesh. This model therefore assumes that all spatially correlated reactions have already occurred by the time that the fine mesh is deleted. This is the principal tradeoff for the increased numerical efficiency of the adaptive meshing scheme.

The threshold reaction rate for the removal of the fine mesh is treated as a parameter and chosen in order to provide results that most closely match those of the uniform mesh simulation. In this work, the threshold reaction rate within a fine mesh for removal and insertion back into the coarse mesh is 10 times the cascade implantation rate. Note that at any given time, several fine meshes can exist in this simulation, but multiple cascades cannot exist within the same fine mesh. At the damage rates studied in this work, the effect of interacting cascades is considered negligible.

A critical parameter in this simulation is the diffusion length used in the reaction rate for diffusion between the fine mesh and the coarse mesh. The fine mesh is assumed to be placed randomly within the coarse mesh, and an effective length $\overline{L}$ is used to find the reaction rate for diffusion between the two meshes. To derive this length, we consider a fine mesh placed randomly in the interior of a coarse mesh element, as shown in Figure 1.6. The coarse and fine mesh element lengths are labelled L_c and l_f, respectively. The total fine mesh has length nl_f, where n is the number of elements along the length of the (cubic) fine mesh. The distances from the edges of the fine mesh to the edges of the coarse mesh element are labelled d_i, with $i \in [1, 6]$ (only two dimensions shown in Figure 1.6).

Reaction type	Number of reactions
Clustering	$\binom{n}{2} + n + mn$
Dissociation	$n + m$
Sinks/traps	kn
(k sink/trap types)	
Diffusion	$6n$

Table 1.3: Maximum number of reactions in SRSCD inside a single volume element with n mobile defect types and m stationary defect types.

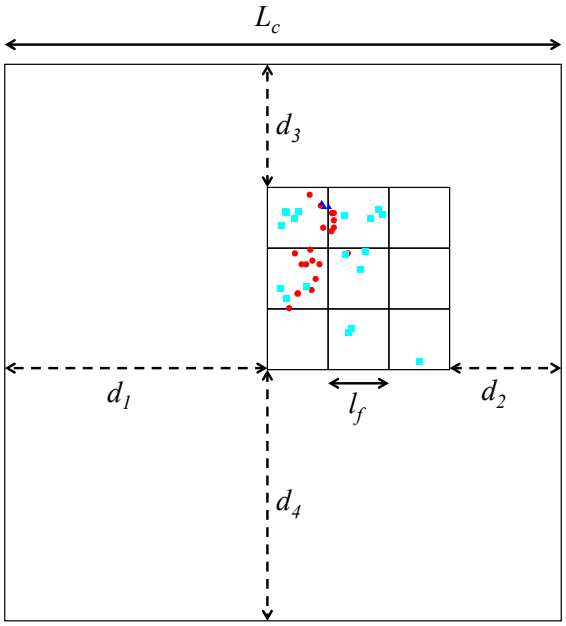


Figure 1.6: Two-dimensional schematic of a single coarse mesh element with a fine mesh inside. The coarse mesh element is divided into virtual elements for the sake of computing the diffusion between the fine and coarse meshes. The fine mesh is assumed to be randomly placed within the coarse mesh element.

The area of a single face of a fine mesh element is a_f, and the total area of the fine mesh is $A_f = 6n^2 a_f$.

In general, the diffusive flux of a single defect species with diffusivity D between rectangular volume elements i and j sharing a face area A_{ij} is given by:

$$\textit{flux} = \frac{DA_{ij}(c_i - c_j)}{L_{ij}} \tag{1.43}$$

where L_{ij} is the distance between the centroids of i and j perpendicular to the normal of the shared face. Therefore, the flux from a fine mesh element on the exterior of the fine mesh to the coarse mesh is given by:

$$\textit{flux} = \frac{Da_f(c_f - c_c)}{\overline{L}} \tag{1.44}$$

where c_f is the concentration of a given defect species in the fine mesh element, c_c is the concentration in the coarse mesh element, and $\overline{L}$ is an effective length to be determined.

To determine this effective length, we refer back to Figure 1.6. We will temporarily assume that the concentration of defects in the fine mesh, c_f, is constant over all fine mesh elements. In this case, using equation (1.43), the total flux of defects from all fine mesh elements on the outer surface of the fine mesh into the coarse mesh element is given by:

$$\textit{flux} = D(n^2 a_f)(c_f - c_c)\left(\sum_{i=1}^{6} \frac{2}{l_f + d_i}\right) \tag{1.45}$$

where d_i are determined by the (random) location of the fine mesh inside the coarse mesh such that $d_1 + d_2 + nl_f = L_c$ (and similarly for the other d_i). Therefore equation (1.45) can be rewritten as:

$$\textit{flux} = D(n^2 a_f)(c_f - c_c)\left(\sum_{i=1,3,5} \frac{2}{l_f + d_i} + \frac{2}{l_f + d_{i+1}}\right) \tag{1.46}$$

where, from above, $d_{i+1} = L_c - d_i - nl_f$ for $i \in [1, 3, 5]$. Integrating from $d_i = 0$ to $d_i = L_c - nl_f$, the maximum value for d_i, the average value for the diffusive flux over all possible placements of the fine mesh within the coarse mesh is given by:

$$\begin{aligned} \textit{flux} = & \frac{D(n^2 a_f)(c_f - c_c)}{(L_c - nl_f)^3} \times \\ & \sum_{i=1,3,5} \int_0^{L_c - nl_f} \iiint \left(\frac{2}{l_f + d_i} + \frac{2}{L_c - (n-1)l_f - d_i} \right) dd_1 dd_3 dd_5 \end{aligned} \tag{1.47}$$

This reduces to:

$$\begin{aligned} \textit{flux} = & \\ & \frac{D(6n^2 a_f)(c_f - c_c)}{L_c - nl_f} \log \left(\frac{(L_c - (n-1)l_f)^2}{l_f^2} \right) \end{aligned} \tag{1.48}$$

Dividing by $6n^2$ to find the flux out of a single fine mesh face, we can compare this equation to equation (1.44) to find $\overline{L}$:

$$\overline{L} = \frac{L_c - nl_f}{\log \left(\frac{(L_c - (n-1)l_f)^2}{l_f^2} \right)} \tag{1.49}$$

In all subsequent simulations, the reaction rate for the flux of defects of a given species out of the fine mesh and into the coarse mesh is given by equation (1.44) with $\overline{L}$ given by Equation (1.49). Note that the concentrations of defects c_f and c_c are given by the number of defects in the fine and coarse mesh elements divided by the fine and coarse element volumes, respectively, where the coarse element volume has been adjusted to subtract the fine mesh volume.

For the case of diffusion from the coarse mesh into the fine mesh, calculating the diffusion rate from the coarse mesh into each element on the outer surface of the fine mesh using $\overline{L}$ given by equation (1.49) is computationally expensive due to the large number of volume elements on the surface of the fine mesh. Therefore, when calculating flux from the coarse mesh to the fine mesh, the entire fine mesh is treated as a single element and equation (1.49) becomes:

$$\overline{L} = \frac{L_c - nl_f}{\log \left(\frac{L_c^2}{(nl_f)^2} \right)} \tag{1.50}$$

When a defect diffuses from the coarse mesh element into the fine mesh in this way, it is randomly placed within one of the fine mesh elements. This procedure allows computation of only one reaction rate for each defect type diffusing into the fine mesh.

The effective length $\overline{L}$ given in equation (1.49) is similar to the value given by simply placing the fine mesh in the center of the coarse mesh element, $\overline{L}_{\text{center}}$, for the simulation parameters chosen in this work. However, as the coarse mesh length L_c becomes large (assuming constant fine mesh length l_f), the ratio $\frac{\overline{L}}{L_c}$ approaches 0 while the ratio $\frac{\overline{L}_{\text{center}}}{L_c}$ approaches $\frac{1}{4}$. Therefore, using the value of $\overline{L}$ presented here instead of $\overline{L}_{\text{center}}$ would have a significant impact on results in cases where the coarse mesh is much larger than the fine mesh.

1.2.3.2 Adaptive meshing: performance

Three metrics are used to study the effectiveness of the adaptive meshing procedure. First, 20 keV cascade implantation in bulk Fe is carried out at various temperatures to 10^{-4} DPA (with all other parameters given in Tables 1.6 and 1.7). The percent of implanted vacancies retained and annihilated by recombination with self-interstitials are computed at each temperature using both the adaptive meshing procedure and a uniform 5 nm

mesh. Second, 20 keV cascade implantation is again carried out at 353 K up to 10^{-2} DPA using both the adaptive meshing procedure and a uniform 5 nm mesh. The concentration of vacancies and vacancy clusters is compared between the two methods. Third, the cluster profiles produced in SRSCD are compared between the uniform and adaptive mesh simulations at a dose of 10^{-4} DPA with an elevated density of traps for SIA loops compared to the density used in other simulations in this work (100 ppm instead of 30 ppm). The reason for this choice is that the increased concentration of SIA loops provided by the increased trap density allows better comparison between the two methods. The results of all three metrics are shown in Figure 1.7. The results show good agreement over the range of temperatures and doses studied. The difference between the vacancy concentrations given by the two methods at high doses seen in Figure 1.7 is likely due to the loss in spatial correlations between defects in the adaptive meshing algorithm when fine meshes are deleted from the system, as discussed previously. This is a limitation of the adaptive meshing method.

1.2.3.3 Computation time

The use of adaptive meshing in this study is a key factor in reaching total doses and simulated volumes that allow these simulations to be compared directly to experiments. To demonstrate the computational efficiency gained with this procedure, 20 keV cascade implantation is carried out to 10^{-4} DPA at 343 K inside a 300 nm cube using adaptive meshing and inside a 100 nm cube using a uniform 5 nm mesh. For the case of the uniform mesh, all counts of reactions and reaction rates are multiplied by 27, the volume difference between the two simulations. The total numbers of reactions of each type listed in Table 1.3 and the average reaction rates are shown in Figure 1.8. The reaction rates in the simulation with the uniform 5 nm mesh are greater than for the adaptive meshing method for all reaction types listed in Table 1.3, and in some cases more than an order of magnitude greater. Figure 1.8 also shows the evolution of the number of reactions of each type as a function of dose in both methods. Again, the uniform mesh has a higher total number of reactions. In addition, the fraction of clustering reactions in the uniform 5 nm mesh is significantly lower than in the adaptive meshing method because of the lower probability that multiple defects are in the same volume element. Both methods show a maximum in the number of diffusion and dissociation reactions as the number of diffusive species saturates and more defects become immobile as larger clusters grow.

When comparing computation times of the two methods, the computational gain provided by the adaptive meshing method varied from a factor of approximately 24 at 10^{-4} DPA to 7.5 at 10^{-3} DPA. Note however that the relative speed of each method depends strongly on the numbers and rates of reactions of the various types shown in Figure 1.8 so that the speedup may not be constant as factors such as simulation volume, temperature, and other variables change. This can be seen in Figure 1.8, as the number of reactions of different types evolves differently in the adaptive and uniform mesh simulations as dose increases.

Trends for computation time in SRSCD using the adaptive meshing scheme as a function of total dose, dose rate, and volume element length are shown in Figure 1.9, using a total dose of 10^{-3} DPA. These results are dependent on both the specific implementation of the SRSCD algorithm used in this study as well as the computational power available to the authors. Therefore, the results shown in Figure 1.9 should be thought of as trends only. Computation time increased in a consistent fashion as a function of both total dose and volume element length. The dependence of computation time on dose rate is much more complex. This is due to the fact that many competing effects change occur simultaneously as dose rate changes. From 7×10^{-13} to 7×10^{-7} DPA/s, two competing processes occur: lower dose rates mean mobile defects can take more diffusive steps in between cascade implantation events, which serves to increase the computation time. However, at lower dose rates the number of mobile defects is also lower on average, decreasing the number of clustering reactions and making computation faster. For the set of parameters chosen here, the result of this is a slightly increasing computation time with dose rate, although this trend may change as other parameters such as temperature change. From 7×10^{-6} to 7×10^{-4} DPA, a different trend dominates: at higher dose rates, fewer large vacancy clusters are formed. This indicates fewer clustering reactions and therefore faster computation times. Finally, at 7×10^{-3} DPA, the computation is dominated by the fact that new cascades are formed so quickly that the number of fine meshes present in the simulation is high. This causes additional computation as the program has to search both fine and coarse meshes for reactions.

Overall, computation time is more strongly effected by total dose than dose rate, as changing the dose rate by several orders of magnitude changes the computation time by less than an order of magnitude. In addition, the dependence of computation time on simulation parameters is very complex as several effects can increase or decrease computation time.

The average timestep is relatively constant for a given simulation in SRSCD because diffusivities of the various defects modeled here differ by many orders of magnitude and therefore the timestep is largely determined by the

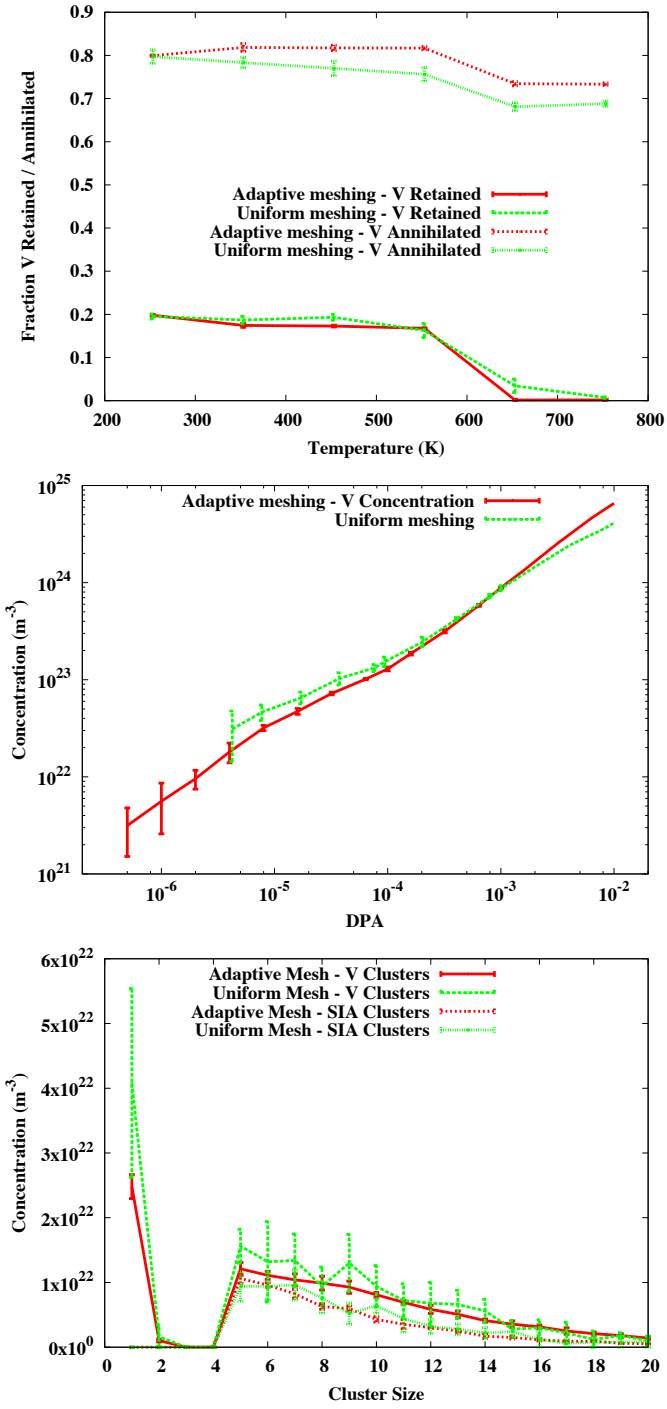


Figure 1.7: Comparison of adaptive meshing and uniform meshing procedures with 20 keV cascade implantation in Fe: (top) percent vacancies retained and annihilated with self-interstitials at various temperatures after 10^{-4} DPA implantation, (center) concentration evolution of vacancies and vacancy clusters at 353 K, and (bottom) density profiles for SIA and vacancy clusters at 10^{-4} DPA.

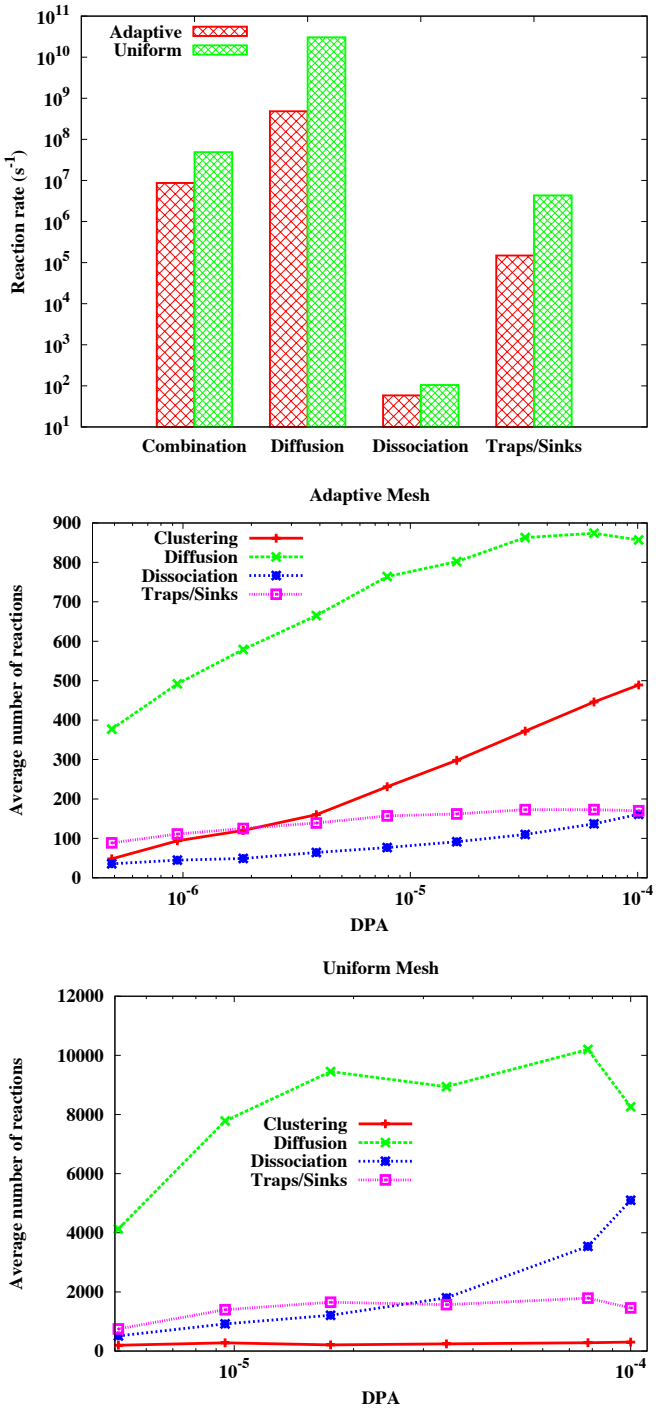


Figure 1.8: Comparison of SRSCD simulations of 20 keV cascade implantation in a 300 nm cube to 10^{-4} DPA using adaptive and uniform meshing methods. (top): comparison of total reaction rates from the four types of reactions modeled in this work. (center and bottom): comparison of numbers of reactions of each reaction type as a function of DPA using the adaptive and uniform meshing methods.

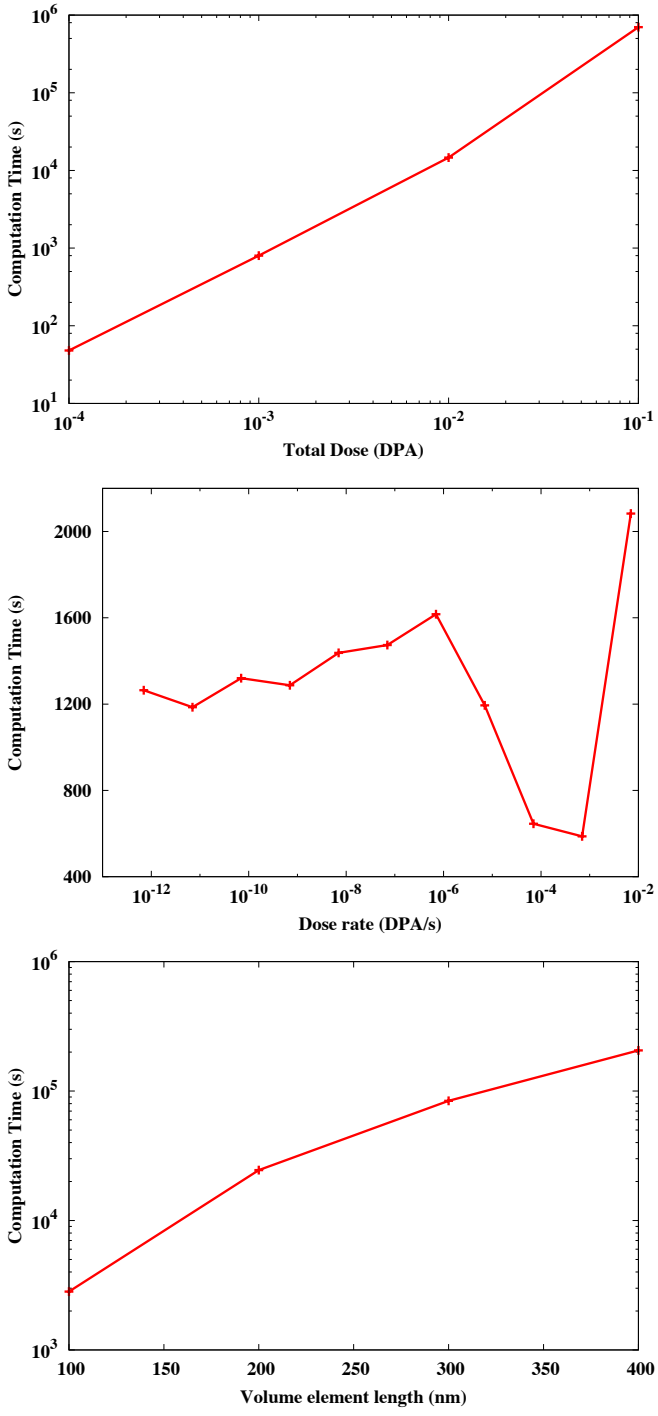


Figure 1.9: Computation time in SRSCD using adaptive meshing as a function of total dose (top), dose rate (center), and volume element length (bottom).

Final dose (DPA)	Simulation size (nm)	Computation time (s)	Number of steps
10^{-4}	$300 \times 300 \times 300$	6.7×10^2	4×10^6
10^{-3}	$200 \times 200 \times 200$	2.2×10^3	9×10^6
10^{-2}	$160 \times 160 \times 160$	1.5×10^4	4×10^7
10^{-1}	$160 \times 160 \times 160$	7.0×10^5	3×10^8

Table 1.4: Comparison of simulation box sizes, computation times, and number of steps of SRSCD simulations using the adaptive meshing algorithm.

fastest reaction in the system, in this case diffusion of small SIA loops. Therefore a large number of reactions may be required to simulate experimentally relevant doses. The number of reactions required to reach these doses is given in Table 1.4.

Using the adaptive meshing scheme described here, simulations of cascade implantation in bulk α-iron are computationally feasible up to a total dose of 10^{-1} DPA. The simulation volumes of the SRSCD simulations performed in the following sections are varied depending on the total dose in each simulation, in order to achieve greater volume at lower doses and greater computational efficiency at higher doses. The box sizes and computation times for the simulations used to generate a range of final doses from $10^{-4} - 10^{-1}$ dpa are given in Table 1.4. By comparison, OKMC simulations of the same experimental conditions have been carried out using simulation volumes of $100 \times 100 \times 100$ nm (Soneda et al. (2003)) and $100 \times 115 \times 130$ nm (Jansson and Malerba (2013, 2014)), but computation times are not reported.

1.3 Synchronous parallel kinetic Monte Carlo algorithm

To reach large simulation volumes one necessarily needs to implement tools such as SRSCD, OKMC, and cluster dynamics in parallel. In a deterministic model (cluster dynamics), such implementation is straightforward as each parallel domain will explicitly march in time with the same time step. However, in the case of kinetic Monte Carlo approaches the problem is far more complex because the time increment at each step is a weighted random choice based on the allowed reactions in the system. Therefore, domains running in parallel may become asynchronous, making reactions such as defect diffusion from one to the other difficult to properly carry out. In order to address this problem, a synchronous parallel kinetic Monte Carlo algorithm has been developed by Martínez et al. (2008) in which a single time increment is chosen for all processors and the possibility of choosing null events is added to the KMC algorithm. Although this algorithm has been implemented for OKMC and lattice kinetic Monte Carlo (LKMC) systems (Martínez et al. (2008, 2011); Martin-Bragado et al. (2015)), it has not been applied to the SRSCD methodology.

The synchronous parallel kinetic Monte Carlo method of Martínez et al. (2008) was developed to avoid asynchronous time evolution in multiple domains which can occur in parallel kinetic Monte Carlo methods. In this method, the total simulation volume is divided between D domains. Inside each domain, there are M possible reactions that can occur and N total defect types. Therefore, X_i^d and A_μ^d represent the population of species X_i in domain d and the reaction rate of reaction R_μ^d in domain d, respectively, with $\mu \in \{1, ..., M\}$, $i \in \{1, ..., N\}$, and $d \in \{1, ..., D\}$. Similarly to the kinetic Monte Carlo algorithm presented earlier, the total reaction rate A^d is computed in domain d:

$$A^d = \sum_{\mu=1}^{M} A_\mu^d \tag{1.51}$$

The maximum total reaction rate is then found among all domains d:

$$A_{\max} = \max_d \{A^d\} \tag{1.52}$$

In order implement the synchronous parallel kinetic Monte Carlo algorithm, a null event R_0^d is then added to the list of reactions in each domain. This reaction represents the possibility of no reaction being carried out in a given domain during a global time step. The reaction rate of the null event A_0^d is defined in each domain as the difference between the total reaction rate in domain d and the maximum reaction rate among all domains:

$$A_0^d = A_{\max} - A^d \tag{1.53}$$

Therefore the rate of R_0^d is nonzero in all domains with total reaction rate $A^d < A_{\max}$. By adding null events to each domain in this way, each domain has a new total reaction rate A^{*d} with value given by:

$$A^{*d} = \sum_{\mu=0}^{M} A_\mu^d = A_{\max} \tag{1.54}$$

where here $\mu = 0$ is included in the sum in order to include the null event in the list of reactions. Since the total reaction rate A^{*d} is the same in each domain, the kinetic Monte Carlo algorithm can choose a single time step for the global system while choosing an independent reaction in each system. This is done by choosing independent random numbers r_1 and $r_2^d \in (0, 1)$. These are used to choose a single time step τ for all domains and a separate reaction R_μ^d in each domain d:

$$\tau = \frac{1}{A_{\max}} \log\left(\frac{1}{r_1}\right) \ , \ \sum_{\nu=0}^{\mu-1} A_\nu^d < r_2^d A_{\max} \leq \sum_{\nu=0}^{\mu} A_\nu^d \tag{1.55}$$

The main difference between equations (1.41), (1.42) and (1.55) is the fact that τ is treated as a global time step and that in each domain the possibility of choosing a null event now exists. The addition of null events to the kinetic Monte Carlo algorithm in this way has been proven not to alter the overall time evolution of the system (Martínez et al. (2008)), while allowing the choice of a single time step and multiple reactions per step in the entire system.

The term 'domain' used above describes a sub-volume of the simulation volume within which one choice of reaction is made during each step. The smallest size this domain can take in SRSCD is a single volume element, such that a separate reaction is chosen inside each volume element during each step, and the largest size that this domain can take is all of the volume elements ascribed to a single processor. Choosing smaller domains increases the number of reactions at each step but makes null events more likely, and the optimal choice of domain size can vary depending on the simulation. All simulations in Section 1.3.1 are carried out with one domain per processor.

The parallel kinetic Monte Carlo algorithm presented here can cause boundary errors in OKMC methods when two reactions are chosen during the same step in neighboring domains such that two defects move within a predefined interaction distance of each other (Martínez et al. (2008)). In SRSCD, boundary errors are caused when defects in neighboring volume elements in different domains exchange positions during the same step. In doing so, the defects are unable to interact because they are never located in the same volume element. However, these errors are extremely unlikely except for unusual cases, for example in a system with two volume elements and two domains only. Therefore, these errors are considered negligible in the simulations performed in this work. Indeed, no difference between defect populations were found in serial and parallel implementations of SRSCD, even in cases with very small total volume per domain.

1.3.1 Scaling of synchronous parallel SRSCD

If a kinetic Monte Carlo simulation with total reaction rate A is divided among D domains using the synchronous parallel algorithm presented above, the best-case scenario for scaling (assuming zero probability of null events in any domain) gives $A_{\max} = \frac{A}{D}$. Therefore, the best-case increase in time step is $\tau_{\text{parallel}} = D \cdot \tau_{\text{serial}}$. Any deviation from this is due to uneven distribution of total reaction rates A^d among the domains, leading to non-zero probabilities of null events being chosen inside some domains at each step. When simulating very stiff problems, e.g. problems in which reaction rates A_μ vary by orders of magnitude for different reactions, total reaction rates A^d are likely to vary as well due to heterogeneous distributions of fast-reacting defects in the system. Because of the large number of defect types modeled in simulations of damage accumulation in metals such as α-Fe, as well as the fact that their diffusivities can vary by many orders of magnitude, this problem occurs frequently in such simulations. Therefore, when investigating the performance of SRSCD, it is key to include both ideal cases that demonstrate the capabilities of the architecture itself as well as cases that demonstrate parallel performance in typical simulations of damage accumulation.

The following scaling simulations were performed on Sandia's High Performance Computing Sky Bridge cluster from Cray, Inc. The cluster has 16 processors and 64GB of RAM per node with 2.6 GHz Intel Sandy Bridge:2S:8C processors. Processor communication is handled through a fully connected QDR InfiniBand interconnect.

1.3.1.1 Damage accumulation in characteristic simulations of irradiated α-Fe: allowed defects and reactions

In Sections 1.3.1.2 and 1.3.1.3, the parallel performance of SRSCD is investigated for two cases: an idealized case in which clustering is not allowed and the only reactions are Frenkel pair formation, diffusion, and recombination, and a case representing realistic defect behaviors. In the latter case, vacancy and self-interstitial clusters are allowed to form, and displacement cascades are introduced in Section 1.3.1.3. Parameters for defect migration and binding energies are taken from ab-initio and atomistic simulations in α-Fe (Fu et al. (2005); Soneda and Diaz de La Rubia (2001)) and can be found elsewhere (Dunn and Capolungo (2015)). Vacancy clusters are assumed to take a spherical shape while clusters of self-interstitials are assumed to form circular dislocation loops. The diffusivity of small vacancy and interstitial clusters is assumed to be three-dimensional, while larger dislocation loops are assumed to glide in one dimension in the direction parallel to their Burgers vector (Soneda and De La Rubia (1998); Soneda and Diaz de La Rubia (2001)). The reaction rates A_μ for the various allowed reactions between defects depend on the diffusivity of the various defects, their geometry, and the character of their diffusion. Reaction rates for all allowed reactions in this work are found in Dunn and Capolungo (2015).

In sections 1.3.1.2 and 1.3.1.3, scaling is investigated using the case of one KMC domain per processor only. Simulations with one KMC domain per volume element are less efficient when defect clustering is allowed due to the large number of null events chosen when the parameters presented above are applied. Further discussion of the scaling of simulations with one KMC domain per volume element can be found in Dunn et al. (2015).

1.3.1.2 Weak scaling during Frenkel pair implantation

In this work, weak scaling is used as the principal metric for parallel performance of SRSCD. In weak scaling, the change in computation time is measured as the simulation volume per processor is held constant while the total simulation volume is varied. This metric indicates how effectively the parallel algorithm enables simulations of larger volumes than are feasible using the serial algorithm. The metric used in this work for parallel performance associated with weak scaling, $\eta_w(p)$, is defined as:

$$\eta_w(p) = \frac{t_s}{t_p}\,, \tag{1.56}$$

where p is the number of processors, t_s is the computation time for a standard serial simulation, and t_p is the computation time for a parallel simulation with p processors and total volume p times the volume of the serial simulation. When $\eta_w = 1$, the system exhibits perfect weak scaling, indicating that the addition of simulation volume and processors does not increase computation time. If $\eta_w < \frac{1}{p}$, the parallel simulation is slower than the equivalent simulation in serial, indicating that the parallel algorithm does not allow efficient simulation of large volumes.

In this section, the weak scaling performance of synchronous parallel SRSCD is characterized for two categories of simulations: (1) simulations in which the only allowed reactions in the system are Frenkel pair implantation, vacancy and self-interstitial diffusion, and vacancy-interstitial recombination and (2) simulations in which defect clustering is also allowed, as described in Section 1.3.1.1. All simulations include volume elements with 20 nm length and periodic boundary conditions. Weak scaling simulations are performed with 1000 volume elements per processor. In order to compare scaling in simulations with different allowed reactions, simulations with and without clustering are started from the same initial defect state. The initial defect state chosen corresponds to the steady-state population of single vacancies and interstitials found when clustering is not allowed for a given dose rate (in dpa·s^{-1}). Steady-state populations of defects are reached before 10^{-4} dpa for all dose rates studied here, and therefore all simulations are started from an initial dose of 10^{-4} dpa. Simulations with no clustering remain at this steady state population while simulations in which clustering is allowed deviate from steady-state as clusters form. Although the latter case does not represent a true physical problem, it was chosen in order to better compare scaling between problems with different allowed reactions. The steady-state population of vacancies and interstitials is shown in Figure 1.10 for the various dose rates used in this study.

Weak scaling results are shown in Figure 1.10 for the cases of no clustering and clustering, respectively. The final dose in all weak scaling simulations is 10^{-3} dpa. In all cases, scaling remains above the $\frac{1}{p}$ limit, indicating that significant computational gains are achieved by using the synchronous parallel SRSCD algorithm. When no clustering is allowed, η_w is almost independent of dose rate indicating that very few null events were chosen in these simulations and the distribution of reaction rates is fairly homogeneous between KMC domains. Therefore all decreases in η_w with increasing numbers of processors are due to increased communication time between processors. By contrast, when clustering is allowed, η_w decreases with decreasing dose rate until 10^{-3} dpa·s^{-1}

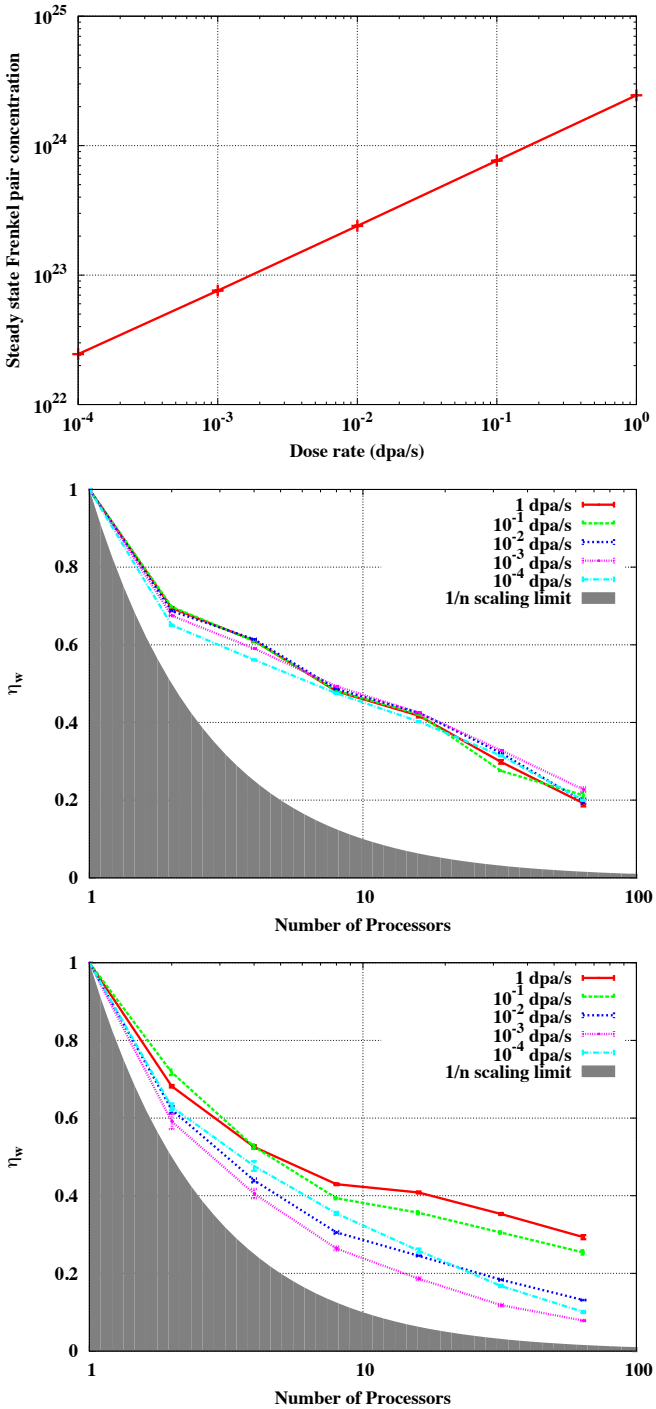


Figure 1.10: Parallel performance associated with weak scaling using several different dose rates corresponding to different initial concentrations of Frenkel pairs. The initial concentration of Frenkel pairs in each simulation is given by the steady-state concentration with no clustering allowed for various dose rates (top). Results are compared for the cases of no defect clustering (center) and clustering (bottom).

and then increases slightly between 10^{-3} and 10^{-4} dpa·s^{-1}. The initial decrease in η_w is due to the increased heterogeneity of reaction rates between processors due to the lower number of defects and the presence of small numbers of fast-diffusing defects like dislocation loops. Below 10^{-3} dpa·s^{-1}, fast-moving loops become scarce again and the population of immobile loops grows, leading to improved scaling as reaction rates become more homogeneous between processors once again.

In summary, weak scaling simulations show that for the simplified system studied here, fast computation of defect evolution is made possible by the synchronous parallel algorithm even for very large total simulation volumes. The scaling of any given SRSCD simulation is dependent on the heterogeneity of reaction rates among processors as well as factors such as the amount of computation time expended choosing reactions and the amount of time spent communicating information between processors.

1.3.1.3 Scaling during displacement cascade implantation

Simulations of cascade damage in SRSCD use a pseudo-adaptive meshing scheme (Dunn and Capolungo (2015)) in which a fine mesh is introduced when cascade implantation events are chosen in the kinetic Monte Carlo algorithm, allowing spatially correlated reactions between defects in the cascade to occur. When all such spatially correlated reactions have occurred, the fine mesh is discarded and the simulation continues with a coarser mesh. This scheme was developed in order to enable simulations of larger volumes and greater radiation doses than feasible in past SRSCD simulations.

This methodology causes simulations of cascade damage to exhibit poor parallel performance, as the total reaction rate when a cascade is present can be as much as seven orders of magnitude greater than after the spatially correlated reactions in the cascade have been carried out. This is shown in Figure 1.11, which depicts the total reaction rates inside each processor in a two-processor simulation of cascade implantation in α-Fe at room temperature. The total reaction rate inside each processor in the simulation oscillates between 10^9 s^{-1} while a cascade is present and 10^2 s^{-1} otherwise. Due to the Monte Carlo nature of cascade implantation, this oscillation leads to a situation where a cascade is typically present in only one processor at a time, and during that time the other processor chooses null events. Thus weak scaling of this method leads to values of η_w close to the $\frac{1}{p}$ limit, as only a single processor typically chooses a reaction at each step.

In order to mitigate this problem, an explicit cascade implantation scheme has been implemented. In this case, cascade implantation is not treated as a reaction that can be chosen with the Monte Carlo algorithm. Instead, cascades are regularly implanted into all processors simultaneously at regular time intervals. The frequency of cascade implantation is chosen such that the overall dose rate remains the same. Thus, spatially correlated reactions between defects in the cascades can be carried out simultaneously for several cascades in multiple processors, decreasing the probability that a null event will be chosen. This is also shown in Figure 1.11, in which the total reaction rates for each processor inside a two-processor system are once again shown. Since cascades are implanted simultaneously in both processors, the amount of overlap between the reaction rates in each processor increases, allowing for better weak scaling.

To demonstrate the computational gain provided by this explicit cascade implantation scheme, the speedup gained by using the explicit method has been calculated for both computation time and number of kinetic Monte Carlo steps:

$$\text{Speedup} = \frac{t_{\text{MC}} - t_{\text{exp}}}{t_{\text{MC}}} \, , \tag{1.57}$$

where t_{exp} and t_{MC} are the computation time in explicit or Monte Carlo cascade implantation mode, respectively. These simulations are performed with 8 coarse volume elements per processor due to computational constraints with length 80 nm and periodic boundary conditions, and displacement cascades are implanted up to 10^{-3} dpa at 7×10^{-7} dpa·s^{-1} and room temperature. Figure 1.12 shows the speedup calculated in this way as a function of the size of the simulated system (keeping volume per processor constant as in weak scaling simulations).

Although the explicit implantation scheme does provide a decrease in computation time and number of steps as the system becomes larger, the improvement is not as strong as seen in Section 1.3.1.2. This is due to the fact that fast-moving interstitial clusters are more likely to form due to the spatial correlation between defects in the cascades, and these defects cause the system to exhibit poor scaling due to their extremely high diffusivities even when the explicit cascade implantation scheme is used. A different choice of material parameters in which dislocation loops are assumed to be immobile, such as in the work of Fu et al. (2005), or a first-passage Monte Carlo algorithm (Opplestrup et al. (2006)) could be used to remove these fast-diffusing clusters and would significantly improve the scalability of the explicit implantation method.

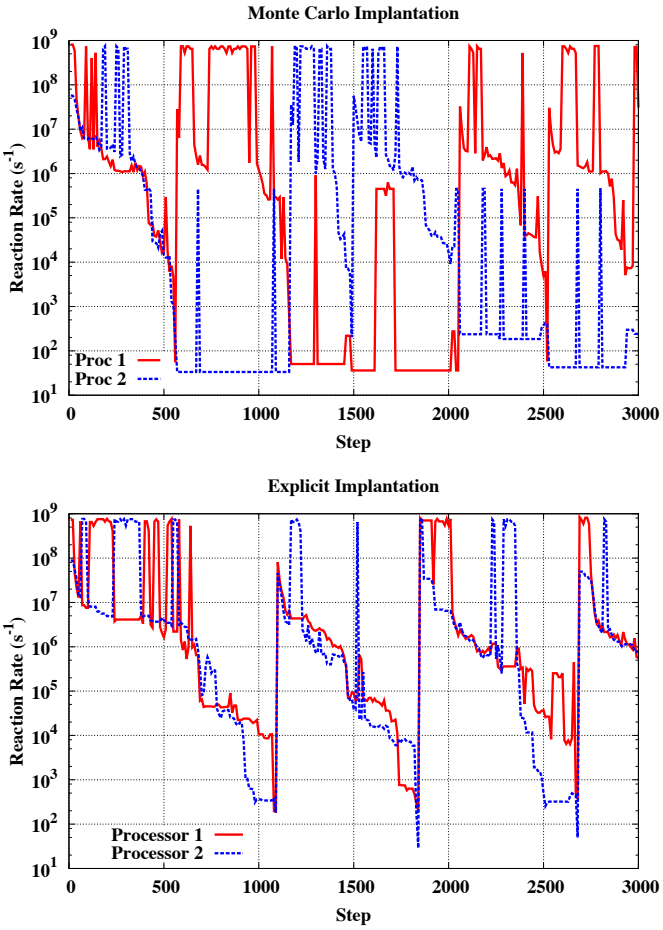


Figure 1.11: Total reaction rates inside each processor during the first 3000 Monte Carlo steps in a two-processor cascade implantation simulation, using the Monte Carlo algorithm for cascade implantation (top) or using an explicit implantation scheme at the same dose rate (bottom). Increasing overlap of reaction rates allows for better scaling of parallel SRSCD simulations.

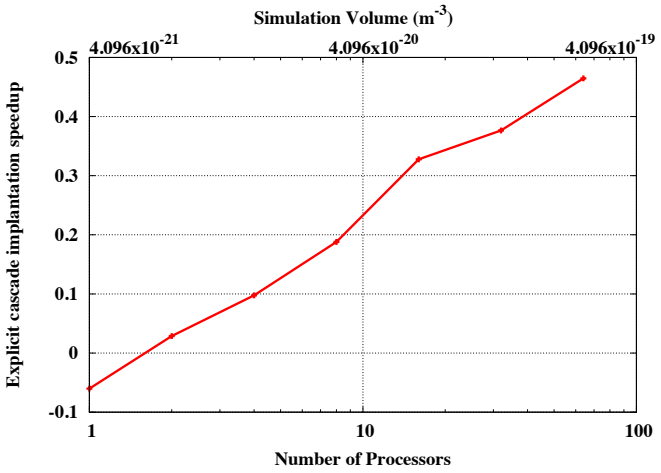


Figure 1.12: Speedup in computation time using the explicit cascade implantation method vs Monte Carlo cascade implantation for simulations of cascade damage in α-Fe.

Parameters used		
Temperature	373 K	
Atomic volume	$1.189 \cdot 10^{-2}$ nm^3	
Burgers vector	0.2876 nm	
dpa rate	$4 \cdot 10^{-7}$ $\frac{\text{dpa}}{s}$	
Diffusion and binding parameters		
v formation energy	E_f^v	1.6 eV
v diffusion prefactor	D_0^v	$6.02 \cdot 10^{10}$ $\frac{\text{nm}^2}{s}$
v migration energy	E_m^v	.65 eV
v_n binding energy	$E_b^v(n)$	$E_f^v + (.2 - E_f^v)\left(\frac{n^{\frac{2}{3}} - (n-1)^{\frac{2}{3}}}{2^{\frac{2}{3}} - 1}\right)$
i diffusion prefactor	D_0^i	$6.02 \cdot 10^{10}$ $\frac{\text{nm}^2}{s}$
i migration energy	E_m^i	.3 eV

Table 1.5: Material and experimental parameters used in the simulation of Stoller et al. (2008). Interstitials were assumed perfectly bound to interstitial clusters and could not dissociate.

1.4 Validation: comparison to other methods and experimental results

1.4.1 Comparison to cluster dynamics

Validation of the SRSCD model described above has been carried out through a series of simulations comparing the results of this model with those of others. The first model chosen for comparison with SRSCD is that of Stoller et al. (2008) which compares the results of MFRT with OKMC. In order to simplify the rate equations used, this model treats only single vacancies and single interstitials as mobile. Circular SIA clusters are immobile in this model, so all migration is in 3D and only the corresponding reaction rates with 3D migration are used (see Table 1.1). In this simulation, Frenkel pairs are implanted homogeneously in an infinite, initially defect-free medium at a constant rate. The vacancy and vacancy cluster populations are recorded as a function of dpa.

To further validate the results of SRSCD, the simulation was carried out in this work using both SRSCD and MFRT to verify that the rate equations and constants were being applied correctly. The diffusion and binding parameters for this simulation are shown in Table 1.5. The concentration of vacancies and vacancy clusters was plotted as a function of dpa and compared to the results of Stoller et al. (2008). Note that spatial resolution was disregarded in the SRSCD model in this simulation due to the spatial homogeneity of the problem. The results of these simulations are shown in Figure 1.13.

It can be seen that all three sets of results are in good agreement. Due to the fact that the MFRT and SRSCD results carried out in this work agree, any differences between these results and the results of Stoller et al. are assumed to be minor. The details of matching these results to the published results of Stoller et al. are considered out of the scope of this work and are not pursued further. It should also be noted that the rate theory results carried out in this study were not extended beyond 10^{-3} dpa, because of computational limitations. The stochastic method, due to its increased computational efficiency, was able to easily reach the larger 10^{-2} dpa range.

Demonstration of the spatially resolved capabilities of the SRSCD model developed here was carried out by modifying the simulation from an infinite medium to a single-crystal layer of material with thickness 400 nm and free surfaces on either side. The free surfaces are treated as infinite sinks for all migrating defects, so that any defect that migrates out of the free surface is lost from the system. This spatially-resolved system was evolved to a much smaller dpa of 10^{-6} due to computational limitations of MFRT. The spatial resolution in the rate theory model was carried out through the use of a finite difference approximation, and in the SRSCD model through the use of the method described above. The spatially resolved profiles of self-interstitial and vacancy concentrations are shown in Figure 1.14.

It can be seen that the results of SRSCD and spatially resolved rate theory agree. Again, the ability of the rate theory results to reach large dpa and timescales was severely limited by the need for small enough timesteps for the solution to converge. However, since SRSCD chooses timesteps and does not compute rates for cluster populations that are not present in the material, this method could easily reach much larger dpa ranges.

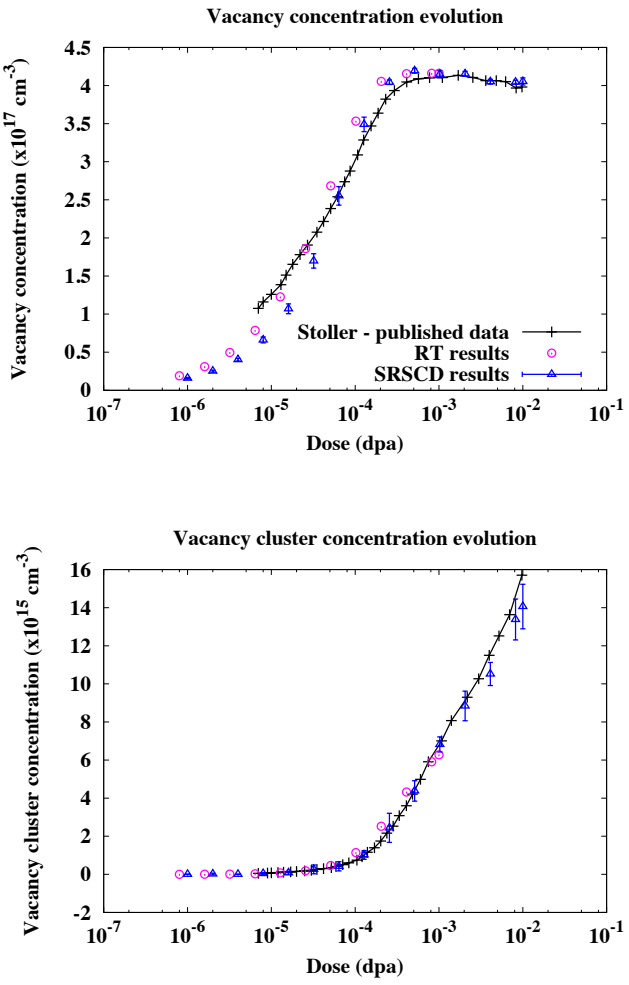


Figure 1.13: Vacancy and vacancy cluster concentrations as a function of dpa for the work of Stoller et al. (2008), rate theory carried out in this work, and SRSCD.

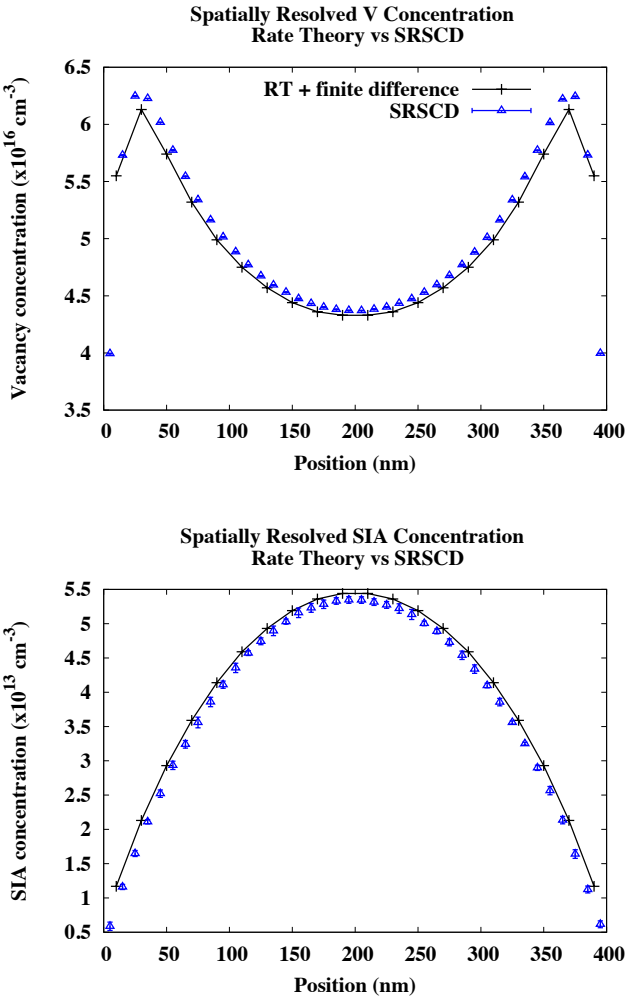


Figure 1.14: Spatially resolved vacancy and interstitial concentration profiles at 10^{-6} dpa.

Default implantation	
Thickness	200 nm
DPA rate	10^{-4} DPA/s
Helium to DPA ratio	2×10^{-2}
Total DPA	10^{-5} to 10^{-2} DPA
Implantation type	20 keV cascades
Quantities varied	
Temperature	273 K to 673 K
DPA rate	10^{-6} to 10^{-2} DPA/s
Helium to DPA ratio	2×10^{-1} and 2×10^{-2}
Thickness	50 nm, 200 nm, and 800 nm
Implantation type	Frenkel pairs and 20 keV cascades

Table 1.6: Table of implantation conditions used in simulations. The effect of changing each parameter on effective diffusivity of helium was studied.

1.4.2 Comparison to OKMC

To test the ability of SRSCD to reproduce the results of OKMC, two simulations are carried out at 373 K over a large range of damage: 20 keV cascade implantation (without helium) in bulk iron and helium implantation with 20 keV cascades in an iron thin film. The simulations were carried out using OKMC and SRSCD with all other implantation conditions set to the default conditions listed in Table 1.6. Bulk iron is simulated with periodic boundary conditions in all directions. In bulk iron, the ability of grain boundaries to absorb $\langle 111 \rangle$ mobile self-interstitial clusters was simulated by removing such clusters from the simulation after migration over 1 μm. Previous work has shown that only $\langle 111 \rangle$ clusters need be removed in this way due to their greater mobility compared to other defects in the material (Dunn et al. (2013b)). Allowed defects and associated binding and migration energies are given in Table 1.7. Defect concentrations are shown for both simulations in Figure 1.15. These results are in good agreement over four orders of magnitude of implantation.

1.4.3 Comparison to experiment: helium desorption from Fe thin films

In order to further demonstrate the capabilities of SRSCD, the annealing of iron foils implanted with helium was simulated. This has been carried out both by experiment (Vassen et al. (1991)) and spatially resolved rate theory developed by Ortiz et al. (2007). In this experiment, three iron foils varying in width from 2.5 μm to 20.6 μm are implanted with helium and annealed. The material parameters for the simulation are given in Table 1.8.

To simulate the experimental conditions, Ortiz et al. first implanted helium, vacancies, and self-interstitials homogeneously in the material to the concentrations listed in Table 1.8, with 200 Frenkel pairs per helium atom introduced. The system was then allowed to reach steady-state at 300 K and subsequently annealed at high temperature. The boundaries of the iron foil were treated as free surfaces (infinite sinks for all mobile point defects) and the amount of helium released from the system as a function of time was tracked.

In the rate theory simulation by Ortiz et al. (2007), the mobile species were limited to interstitial helium, single vacancies, single interstitials, and 2-interstitial clusters. It is important to note that the rate equations used by the authors differ from the rate equations presented here, and can be found in Ortiz and Caturla (2007). Using rate theory, Ortiz et al were able to achieve good agreement between simulation results and experiment, assuming some modified energies due to the effect of impurities in the material.

In the SRSCD simulation, the same material parameters are used, but all species which have been found to be mobile in iron are allowed to migrate. In addition, HeV and HeV_3 clusters have been studied in Nb (Dunn et al. (2013a)) but do not have migration data available in Fe. For this simulation the migration values from Nb are used for HeV and HeV_3, but since these defects have relatively high migration energy the impact of their migration on the simulation results is negligible. The migration and binding energies of all cluster types used in this simulation are given in Table 1.7.

In order to include helium in the rate equations presented above, HeV clusters are assumed to be spherical with radius given by the number of vacancies in the cluster. Helium clusters are also assumed to be spherical, but the population of interstitial helium clusters is never significant. Thus reaction rates for He+HeV, V+HeV, HeV+HeV,

Migration parameters	$D = D_0 e^{-\frac{E_m}{k_b T}}$	
Single vacancy	$E_m = 0.67$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
2-v cluster	$E_m = 0.62$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
3-v cluster	$E_m = 0.35$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
4-v cluster	$E_m = 0.48$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
Larger v clusters	(immobile)	
Single interstitial	$E_m = 0.34$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
2-i cluster	$E_m = 0.42$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
3-i cluster	$E_m = 0.43$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
4-i cluster	$E_m = 0.43$ eV, $D_0 = 8.2 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Fu et al. (2005)
n-i cluster (1D)	$E_m = .06 + .11(n^{-1.6})$, $D_0 = (3.5 \times 10^{10} + 1.7 \times 10^{11} n^{-1.7})$ $\frac{\text{nm}^2}{s}$	Soneda and Diaz de La Rubia (2001)
Single-He	$E_m = 0.077$ eV, $D_0 = 5 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
2-He cluster	$E_m = 0.055$ eV, $D_0 = 3 \times 10^{10}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
3-He cluster	$E_m = 0.062$ eV, $D_0 = 1.5 \times 10^{10}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
4-He cluster	$E_m = 0.062$ eV, $D_0 = 5 \times 10^{9}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
HeV (value for Nb)	$E_m = 2.57$ eV, $D_0 = 1.15 \times 10^{12}$ $\frac{\text{nm}^2}{s}$	Dunn et al. (2013a)
HeV_2	$E_m = 0.27$ eV, $D_0 = 4.1 \times 10^{10}$ $\frac{\text{nm}^2}{s}$	Fu and Willaime (2005)
HeV_3 (value for Nb)	$E_m = 1.42$ eV, $D_0 = 1.15 \times 10^{12}$ $\frac{\text{nm}^2}{s}$	Dunn et al. (2013a)
He_2V	$E_m = 0.33$ eV, $D_0 = 1.16 \times 10^{11}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
He_3V	$E_m = 0.31$ eV, $D_0 = 2 \times 10^{10}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
He_4V	$E_m = 0.28$ eV, $D_0 = 2.36 \times 10^{9}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
He_2V_3	$E_m = 0.55$ eV, $D_0 = 7.82 \times 10^{9}$ $\frac{\text{nm}^2}{s}$	Terentyev et al. (2009)
Binding energies		
$\text{V}_2 \rightarrow \text{V}+\text{V}$	$E_b = 0.3$ eV	Fu et al. (2005)
$\text{V}_3 \rightarrow \text{V}_2+\text{V}$	$E_b = 0.37$ eV	Fu et al. (2005)
$\text{V}_4 \rightarrow \text{V}_3+\text{V}$	$E_b = 0.62$ eV	Fu et al. (2005)
Vacancy clusters $(n > 4)$	$E_b^v(n) = 2.07 - 3.01(n^{\frac{2}{3}} - (n-1)^{\frac{2}{3}})$	Fu et al. (2005)
$\text{I}_2 \rightarrow \text{I}+\text{I}$	$E_b = 0.8$ eV	Fu et al. (2005)
$\text{I}_3 \rightarrow \text{I}_2+\text{I}$	$E_b = 0.92$ eV	Fu et al. (2005)
Interstitial clusters $(n > 3)$	$E_b^i(n) = 3.77 - 5.05(n^{\frac{2}{3}} - (n-1)^{\frac{2}{3}})$	Fu et al. (2005)
$\text{He}_2 \rightarrow \text{He}+\text{He}$	$E_b = 0.43$ eV	Ortiz et al. (2007)
$\text{He}_3 \rightarrow \text{He}_2+\text{He}$	$E_b = 0.95$ eV	Ortiz et al. (2007)
$\text{He}_4 \rightarrow \text{He}_3+\text{He}$	$E_b = 0.98$ eV	Ortiz et al. (2007)
Small He_mV_n clusters $(m, n \leq 4)$	E_b taken from Ortiz and Caturla	Ortiz et al. (2007)
He_mV_n (He binding, $\frac{m}{n} \leq 0.5$)	(does not dissociate)	
He_mV_n (He binding, $\frac{m}{n} > 0.5$)	$E_b^{\text{He}} = 2.2 - 1.55 \log\left(\frac{m}{n}\right) - .53 \log\left(\frac{m}{n}\right)^2$	Terentyev et al. (2009)
He_mV_n (V binding, $\frac{m}{n} \leq 0.5$)	$E_b^v(n) = 2.07 - 3.01(n^{\frac{2}{3}} - (n-1)^{\frac{2}{3}})$	Fu et al. (2005)
He_mV_n (V binding, $\frac{m}{n} > 0.5$)	$E_b^v = 1.55 + 3.19 \log\left(\frac{m}{n}\right) + 3.0 \log\left(\frac{m}{n}\right)^2$	Terentyev et al. (2009)

Table 1.7: Migration and binding parameters used in OKMC and SRSCD simulations. The diffusion of HeV and HeV_3 clusters are taken from the values found for Nb, which is assumed to be similar to the behavior of Fe. He_mV_n clusters with $m/n \leq 0.5$ are assumed to act as vacancy clusters only, and do not allow He dissociation. Vacancy and helium binding energies in He_mV_n for $m/n > 0.5$ are taken from Terentyev et al. (2009).

Anneal temp	He concentration	Sample thickness
559 K	1.39 ppm	2.5 μm
577 K	0.013 ppm	20.6 μm
667 K	0.109 ppm	2.6 μm

Table 1.8: Material parameters used in experiment (Vassen et al. (1991)) and rate theory (Ortiz et al. (2007)) studies of helium desorption from Fe foils

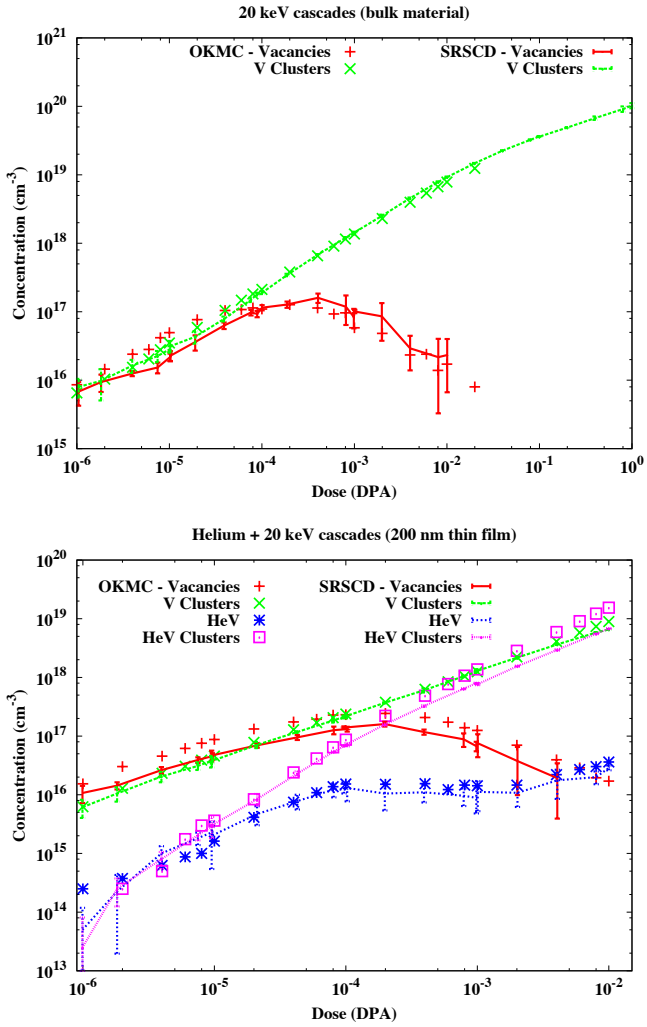


Figure 1.15: Species populations in an infinite Fe medium implanted with 20 keV cascades (top) and a Fe thin film implanted with helium and 20 keV cascades (bottom). Results are shown for OKMC and SRSCD. The two methods are in good agreement for the range of DPA shown.

and SIA+HeV (with the number of V greater or equal to the number of SIA) are found using the same methods as described above.

Binding of helium to single interstitials and self-interstitial clusters is a problem that has not been fully addressed in the literature. Marian and Bulatov (2011) and Becquart and Domain (2009) have studied defect evolution including He-SIA cluster formation, while the work of others such as Ortiz et al. (2007) neglects this effect. In general, the availability of parameters governing the behavior of He-SIA clusters is low. Therefore, this simulation was carried out using only He-V clustering. In order to test the significance of this choice, identical helium desorption simulations were performed using the He-SIA parameters reported by Marian et al, assuming that He-SIA clusters are stable and immobile. Although a large number of He-SIA clusters appeared at the beginning of the annealing stage of the simulation, the change in the overall desorption results was minor. In addition, the physical mechanism of helium desorption does not change in this case.

The system is evolved for 1 second at 300 K and subsequently annealed for 2×10^4 seconds at the higher temperature listed in Table 1.8, with the boundaries of the system free surfaces for the entire simulation. The boundary conditions of the system during the low-temperature annealing part of the simulation were shown to not strongly impact the results of the simulation. The amount of helium released from the system and the types of helium clusters that leave the system are recorded.

The results provided by SRSCD differ from the experimental results when using the parameters listed in Table 1.7. Figure 1.16 shows helium desorption for the 559 K sample using SRSCD compared to experiment. Although the qualitative results match, the model predicts significantly more helium release than was measured in experiment or simulated in the rate theory model of Ortiz et al. (2007).

Assuming that the material has some impurity content, interstitial clusters could be trapped and become immobile when interacting with impurity atoms (Hudson et al. (2004)). In the work of Ortiz et al. (2007), the migration energy of vacancies was used as a fitting parameter in order for the rate theory results to match experiment. This variation of vacancy migration energy is explained by Ortiz to be the result of the presence of impurities in the iron foil. In this simulation, impurities are accounted for by varying the mean free path for interstitial clusters to become immobile due to interaction with impurities. The simulation results were shown to depend only weakly on the actual value of the mean free path chosen and thus an optimal value of mean free path was not found. Nonetheless, using this method for immobilizing SIA clusters causes SRSCD results to match the experimentally measured desorption results of Vassen et al. (1991) more closely.

The results of the SRSCD simulation compared to experiment are shown in Figure 1.17. Good agreement is achieved between SRSCD results and experimentally measured helium desorption for a wide range of foil thicknesses, initial helium content, and annealing temperatures (see Table 1.8). It should be noted that SRSCD necessarily depends on a large number of binding and migration energies and prefactors, some of which are not agreed upon in the literature. The simulation results are most sensitive to binding and migration energies of small clusters such as V_2 and HeV_2. Changing any of these parameters changes the quantitative results of the simulation while (generally) keeping the qualitative results the same.

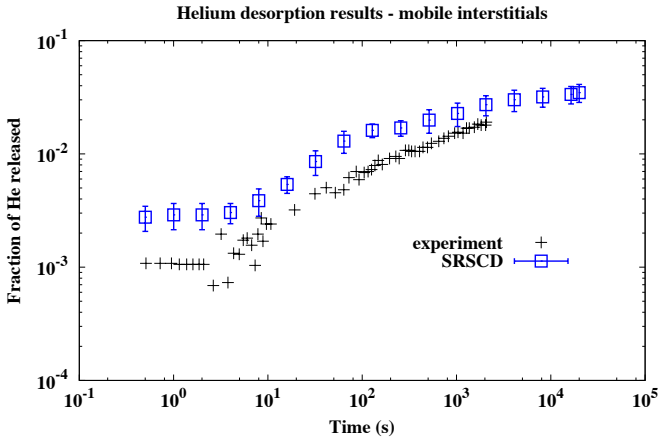


Figure 1.16: % He released, 559 K anneanling. Interstitial clusters are assumed to be mobile until interacting with a second interstitial cluster, at which point they form a junction and become immobile. These results do not match the experimental results of Vassen et al. (1991).

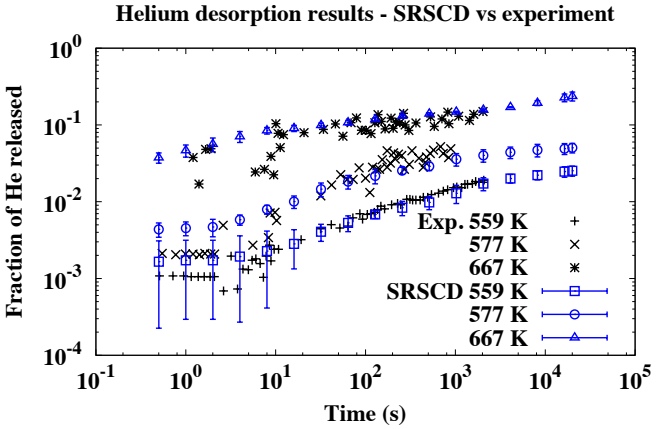


Figure 1.17: Fraction of He released for experimental parameters given in Table 1.8.

1.4.4 Comparison to experiment: neutron irradiation of bulk Fe

The final validation of SRSCD is comparison of accumulated defect populations with the measured vacancy and dislocation loop populations given by Eldrup et al. (2002) and Zinkle and Singh (2006) in neutron-irradiated coarse-grained α-Fe at room temperature. Due to the large dose and relatively large volume simulated in these simulations, the adaptive meshing scheme described in Section 1.2.3 is implemented in these SRSCD simulations for increased computational efficiency. The evolution of the concentration of vacancy and self-interstitial clusters as a function of total dose is compared to the results given by Eldrup et al. (2002) and Zinkle and Singh (2006) at doses ranging from 10^{-4} to 10^{-1} DPA. These simulations are carried out using 20 keV cascades and assuming a 30 ppm concentration of traps for SIA loops. All other simulation parameters are given in Table 1.7.

In these experiments, vacancy cluster concentration is measured by positron annihilation spectroscopy (PAS) and therefore includes even very small defects such as single vacancies. Conversely, self-interstitial loop concentration is measured by transmission electron microscopy (TEM) and therefore a lower threshold on the detection limit of this method must be included in the computation of SIA loop densities. In this study, results are shown assuming SIA loops are visible above 1 nm in diameter, with the range of results representing a detection limit of 0.9-1.1 nm shown as well. The large variation in SIA cluster concentration observed when changing the threshold diameter for detection indicates that these results are very sensitive to the actual threshold SIA loop size for detection using TEM.

Results for SIA and vacancy concentrations using SRSCD are shown in Figure 1.18. The results of SRSCD show a qualitative match to experimental results at low doses (below 10^{-2} DPA). At high doses, SRSCD predicts a saturation and eventual decrease in visible SIA loop density that is not seen in experimental results. The cause of the difference between SRSCD and experimental results is likely related to the treatment of cascade-defect interactions. In particular, the assumption of random combination between cascade defects and pre-existing defects during cascade events does not account for effects that may occur in irradiated metals such as the break-up of large defect clusters by cascades. Overall, SRSCD is able to qualitatively reproduce experimentally observed trends for defect accumulation, although more information about cascade damage evolution at high doses is needed to more closely reproduce experimental results.

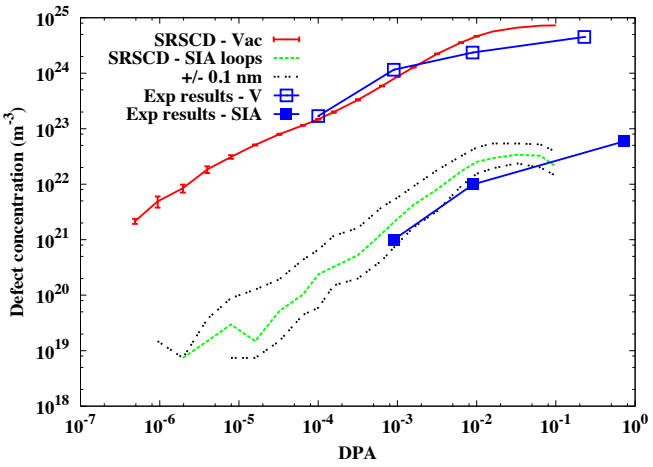


Figure 1.18: Match of SRSCD results with experimental results of Eldrup et al. (2002). The concentration of SIA clusters is given assuming visible (in TEM) clusters of diameter greater than 1 nm, with black dotted lines indicating detection limits of 0.9 and 1.1 nm.

Chapter 2

Outline of code structure

In this chapter, the various sections in the code will be outlined in detail and the data flow through the synchronous parallel kinetic Monte Carlo algorithm will be described. The outline of the various procedures carried out in the SRSCD code, as well as the sections in this document describing their structure, is as follows:

Overview of SRSCD

1. General description of SRSCD loop (Section 2.1)
2. Flow chart of main subroutines (Section 2.1.1)

Initialization

1. Mesh initialization (Section 2.2.1)
2. Read in material and defect properties (Section 2.2.2)
3. Read in cascade data (if necessary, Section 2.2.3)
4. Read in non-uniform damage profile (if necessary, Section 2.2.4)
5. Read in simulation parameters (Section 2.2.5)
6. Initialize random seed (Section 2.2.6)
7. Allocate derived type variable structures (defect lists, reaction lists) (Section 2.2.7)
8. Initialize defect list, reaction list, and total rates (Section 2.2.8)

Kinetic Monte Carlo loop

1. Find total reaction rate (Section 2.3.1)
2. Choose timestep (Section 2.3.2)
3. Choose reaction (one per volume element or one per processor, Section 2.3.3)
4. Update defect list as well as boundary defect list and create a list of defects that have changed (for reaction update step, Section 2.3.4)
   (a) If cascade chosen, create fine mesh and carry out cascade initialization
5. Update reaction rates for all defects that have changed and update total reaction rate in each volume element and in entire processor (Section 2.3.5)
6. If cascade is annealed, remove it, place defects in coarse mesh, and reset reaction rates (Section 2.3.6)
7. Check that total reaction rate is correct (every n steps, Section 2.3.7)
8. Intermediate output / post processing (every m steps or after a set amount of time passage, Section 2.3.8)
9. If elapsed time has passed the total time, exit KMC loop (Section 2.3.9)

Other actions

1. Re-initialize simulation for annealing (Section 2.4.1)
2. De-allocate variables that are reset in each simulation (Section 2.4.2)
3. Carry out multiple simulations (Section 2.4.3)
4. Deallocate all memory (Section 2.4.5)
5. Finalize MPI (Section 2.4.6)

Each of these steps will be detailed in the following sections.

2.1 Brief overview

In this section, a shorter overview of the structure of the SRSCD simulation will be given, without using the specific details of the code implemented here. Therefore, the overview described here should apply to any implementation of the SRSCD algorithm, and is based on the work of Gillespie (1976).

Outline of SRSCD algorithm

1. **Initialization:** Identify allowed defects and reactions, gather input data such as temperature and dose rate, and create data structures to store defects and reactions.
2. **Kinetic Monte Carlo (KMC) loop:**
   (a) Choose timestep τ using KMC timestep formula:
   $$\tau = \frac{1}{a} \log\left(\frac{1}{r_1}\right) \tag{2.1}$$
   where a is the total reaction rate inside the system (sum of reaction rates of all allowed reactions μ $a = \sum_\mu a_\mu$), r_1 is a random number $r_1 \in [0, 1]$.
   (b) Choose reaction μ using KMC reaction choice formula:
   $$\sum_{\nu=1}^{\mu-1} a_\nu < r_2 a < \sum_{\nu=1}^{\mu} a_\nu \tag{2.2}$$
   where reaction μ has reaction rate a_μ and r_2 is a second random number $r_2 \in [0, 1]$. This is a weighted random choice of reactions, using the reaction rates as weights.
   (c) Carry out reaction μ: For example, if reaction μ is Frenkel pair implantation in a volume element, then add one vacancy and one self-interstitial to the defect list of that volume element.
   (d) Re-calculate all relevant reaction rates a_μ: After the number of defects in the system has changed, the reaction rates (from cluster dynamics) for those defects to interact have changed. Therefore, the reaction rates in the list of allowed reactions are re-calculated for all relevant reactions at each step. This step depends on the allowed reactions, which are part of the input to the system.
   (e) Re-calculate total reaction rate a, for timestep choice in next iteration of KMC loop.
3. **Output:** This can take various forms, such as the numbers of each defect type present in each volume element, any post-processing that the user desires, or outputs in formats that can be plotted by programs such as ParaView
4. **Finalization:** Delete defect and reaction structures, finalize any parallel processing, etc.

2.1.1 Flow chart of main code and subroutines

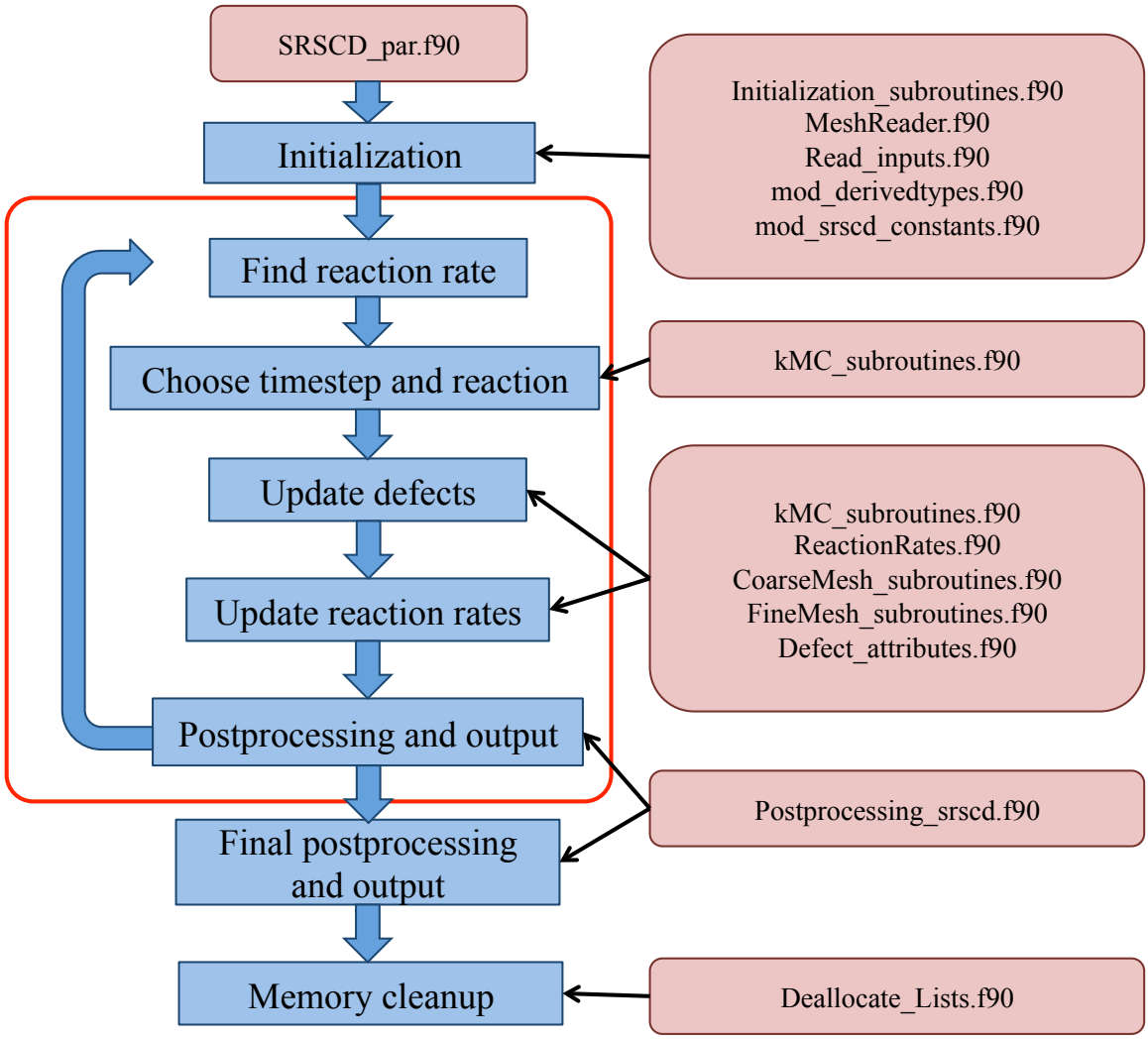


Figure 2.1: Flow chart of code structure and which files contain subroutines used in each section of the SRSCD algorithm

2.2 Initialization

2.2.1 Mesh initialization

This procedure is carried out in the readMeshUniform() subroutine (another subroutine, called readMeshNonUniform(), has also been created but not implemented with the full SRSCD code). As this is intended to read in a mesh from a file and assign volume elements to various processors in a parallel configuration, this subroutine contains several steps. The overall outline is below:

1. The filename (which was passed into the subroutine) is used to open a mesh file that has been correctly formatted (see Chapter 4 for examples).
2. Various toggles and parameters are read in from the mesh file, including whether the simulation has periodic boundary conditions or free surfaces in the $z-$direction, the length of volume elements, and the global coordinates of the vertexes of the simulation volume.
3. If the strainField toggle is set to 'yes', then the previous steps are repeated for a separate file that contains the strain field (all six components) at each node that is listed inside the mesh file. Therefore, the strain field

file must contain the same number of volume elements with the same coordinates as the mesh file. Note also that although strain has been implemented in SRSCD, it has not been applied to any problems yet and therefore this is unused code at the moment.

4. The processors being used in this simulation are divided over the simulation volume, using an arrangement that minimizes the shared surface area between processors. Therefore, if a cubic simulation volume is divided among 8 processors, it will be divided in a $2 \times 2 \times 2$ configuration, but if it is divided among 5 processors, it will be divided in a $5 \times 1 \times 1$ configuration. The simulation therefore identifies in each processor the vertexes of the simulation volume that it contains.

5. After dividing the simulation volume among the processors, connectivity between processors is created in each of the 6 cubic directions: each processor is assigned a processor to its left and right, front and back, and top and bottom. In the event of a processor division such as $5 \times 1 \times 1$ processors, periodicity implies that processors point at themselves in directions where they have no neighbors.

6. A mesh of volume elements inside each local processor is created. This consists of several steps:

   (a) The total (global, over all processors) mesh is read in from the data file

   (b) If the strainField toggle is set to 'yes', the strain field is read in from the strain file into a global strain field.

   (c) A global connectivity matrix is created

   (d) Any element being read into the global mesh whose center is within the local processor's domain is counted. The size of the local domain is therefore defined in the x, y, and z-directions.

   (e) If any processors don't have any volume elements inside them, an error is output and the SRSCD simulation is stopped

   (f) The local mesh myMesh is allocated using the number of volume elements inside the local processor

   (g) For each volume element inside myMesh, coordinates of the center of the volume element and the material ID number (if applicable, used for simulating grain boundaries or polycrystalline materials) are assigned. This information is taken from the global mesh. The strain field is also taken from the global mesh, if applicable.

   (h) Local connectivity, which is stored in myMesh, is created using the subroutines createconnectlocalperiodicuniform and createconnectlocalfreesurfuniform.

7. A boundary mesh is created, which represents the volume elements in neighboring processors that are adjacent to volume elements in the local processor. This also takes several steps:

   (a) The structure myBoundary is allocated (it is made larger than necessary; several of its elements are unused).

   (b) For every element in myMesh, if its neighboring processors (defined when connectivity was created) are not inside the local processor, then it has an adjacent volume element that is inside a different processor. If that is the case, an element within myBoundary is defined using the global mesh. This data structure contains the processor ID number, the length of the volume element, the volume of the volume element, the material type (used for grain boundary simulations), and the ID number of the LOCAL volume element that is a neighbor of this BOUNDARY element.

8. Lastly, all mesh input files are closed.

2.2.2 Material and defect input

In the main program (SRSCD_par.f90), after the previous step the subroutine selectMaterialInputs is called. In this subroutine, the following steps are carried out:

1. The global integer numMaterials is read from parameters.txt

2. The global variables representing the number of binding energies, diffusivities, and reactions of various types are allocated as arrays with size numMaterals, so that a separate value can be read in from each material type into each variable.

3. The filename of the various files containing information on diffusivity, binding, and allowed reactions are read in for each material

4. The number of diffusivity, binding, and reactions of each type contained in each material file are read in from the material files

5. The derived type variables containing diffusivity, binding, and reaction information are allocated based on the number of parameters / reactions of each type in each file.

6. The subroutine readMaterialInput is called for each material.

Inside readMaterialInput, the following data is input from the input files (with filenames given above):

1. The material ID number and number of species are read in.

2. The number of single and functional forms for diffusivities (D_0 and E_m) are read in, and then the diffusivity values for species listed in single and functional forms are read in.

3. The number of single and functional forms for binding energies (E_b) are read in, and then the binding energy values for species listed in the single and functional forms are read in.

4. The number of dissociation reactions is read in, and then the reaction parameters for all allowed dissociation reactions are read in

5. The previous step is repeated for diffusion reactions, sink removal reactions (such as homogeneously distributed dislocations), impurity trapping reactions, clustering reactions, and implantation reactions.

2.2.3 Cascade input

If the toggle implantType is set to 'Cascades', then the subroutine readCascadeList is carried out. This subroutine contains the following steps:

1. The toggles implantType and implantScheme (for Monte Carlo or explicit cascade implantation, if explicit is chosen but implantType is 'FrenkelPair', the program returns an error) and meshingType (adaptive or non-adaptive) are read in.

2. If cascade implantation is chosen, then the cascade file is opened (filename read in from parameters.txt)

3. The average number of displaced atoms numDisplacedAtoms (for calculation of DPA) is read in from the cascade file.

4. The number of different cascades is read in from the cascade file and the derived type variable cascadeList is allocated (it is a pointer list).

5. For each cascade, the defects in the cascade are read in along with their coordinates and the number of displaced atoms in that cascade. After reading in each cascade, the next member of cascadeList is allocated. A separate derived type is used to store cascade defects within cascade list (also a pointer list)

2.2.4 Damage profile input

In this section, the subroutine readImplantData is called. This is only activated if the toggle implantDist is 'nonUniform'. If so, the file name for the non-uniform DPA rate and He implant rate are read in from parameters.txt. Then the number of implantation rate data points is read in (does not need to match the mesh, if it does not match an averaging procedure is used to estimate the implantation rate at each mesh point). Finally the implantation rate data variable is allocated and the implantation rate data is read in for each data point.

NOTE that in this subroutine, only a one-dimensional varying implantation rate is allowed.

2.2.5 Simulation parameter input

This calls subroutine readParameters. This is necessary for all types of simulations. In this subroutine, the default parameters for all variables in parameters.txt are set. Next, the values of each of these variables are read in from parameters.txt. Any variables that are not present in parameters.txt are set to their default value. This includes parameters such as the temperature, atomic volume, number of simulations, dose and dose rate, as well as many types of toggles such as the activation of non-spatially resolved grain boundary sinks, SIA loop pinning by He atoms, and others.

At the end of this subroutine, the geometric constants used to calculate reaction rates for clustering and dissociation are computed using values from parameters.txt.

2.2.6 Random seed initialization

The subroutine initializeRandomSeeds is called. In this subroutine in the master processor the random seed is initialized using an integer taken from the intrinsic system_clock function. Then the master processor generates a random integer for each slave processor and that integer is sent via MPI to the slave processor, which uses it to initialize its random number generator. Thus the random number seeds for each processor are as uncorrelated as possible.

Random number seeding is called using the sdprnd function.

2.2.7 Allocation of lists

In the main program (SRSCD_par.f90), the global derived type pointer lists defectList, reactionList, and the double precision array totalRateVol are allocated with size numCells. DefectList and reactionList are arrays of pointer lists with size equal to the number of volume elements in the simulation.

2.2.8 Initialization of lists

Here, the defect lists and reaction lists are initialized, along with the boundary lists and (if debugRestart is toggled to 'yes') the defects initially present in the material.

1. In subroutine initializeDefectList, the first defect in defectList for each cell is initialized as a zero defect - it has no type and there are zero of these defects. All other defects that are created are added on to this list using the %next pointer, and this defect is never deleted (it always remains as the first defect in the list).
2. In subroutine initializeBoundaryDefectList, similar initialization is carried out to initializeDefectList, except that this is only done in certain elements in the array myBoundary, which was created as larger than the actual size of the boundary for simplicity. Therefore, some elements in this array are unused.
3. In subroutine initializeReactionList, for each volume element, the following steps are carried out:
   (a) If Frenkel pair implantation is chosen, the first reaction in the reaction list for each element is Frenkel pair implantation. The number of reactants for that reaction is zero and the number of products for that reaction is 2. The reaction rate for that reaction is found using findReactionRate. If Helium implantation is also included, then a second reaction is also created for helium implantation (no reactants, 1 product, reaction rate from findReactionRate as well). Each reaction is created in each element of the reactionList array using the reaction derived type.
   (b) If cascade implantation is chosen, the number of reactants is set to -10 (as a toggle) and the number of products is set to 0 (cascades are manually implanted). Otherwise the cascade reaction rate is also found using findReactionRate and the reaction is created similarly to Frenkel pair implantation. Helium implantation is also included in the same way as the previous step.
4. In subroutine initializeTotalRate, the total rate in the entire processor is initialized by passing through each reaction in each element in reactionList and adding it to totalRate. Similarly, the total rate in each volume element is initialized by adding up all of the reaction rates in each volume element. Note that if the toggle

singleElemKMC is set to 'yes', then the total rate in the processor is instead set to the MAX rate among all volume elements in the processor.

5. In subroutine initializeDebugRestart, defects are read in from an external file and then added to the defect list in the following fashion:

   (a) The name of the debug restart file is read in

   (b) The number of processors, number of implantation events that have occurred, number of helium implant events that have occurred, and elapsed time is read in. If the number of processors does not match the actual number of processors, an error is output.

   (c) The read-in part of the subroutine skips to the part of the input file relevant to the processor ID that is reading it in.

   (d) In each volume element, the coordinates of the volume element are read in and compared to the coordinates of myMesh(cellNumber) to ensure that the mesh in the debug file matches the mesh in the simulation.

   (e) In each volume element, the number of defect types in that volume element is read in.

   (f) In each volume element, the defect types present and the numbers of each defect type present are read in. A defect is created in defectList(cellNumber) for each defect type read in each volume element.

   Thus the simulation is able to continue as if it is starting from the point that the debug file was created. If the debug restart toggle is set to 'yes', the simulation then sets all counters such as the number of implantation events and number of helium implantation events (for tracking purposes) and resets all of the reaction lists in all volume elements due to the presence of defects (until this point reaction lists have simply been initialized with implantation reactions, defect clustering reactions have not been added). More on reaction list resetting in Section 2.3.5.

2.3 KMC loop

After initialization, the code passes into the kinetic Monte Carlo (KMC) loop. This loop continues while the elapsed time is less than the total time, which is defined as:

$$\mathrm{totalTime} = \frac{\mathrm{totalDPA}}{\mathrm{DPARate}} \tag{2.3}$$

where totalDPA and DPARate are variables read in during initialization from parameters.txt.

2.3.1 Compile total reaction rate for timestep choice

The first step in the synchronous parallel KMC algorithm is to find the reaction rate used to choose the timestep. If the size of the KMC domain is one reaction choice per processor per step, then the total rate inside each processor has already been computed and the maximum of those is chosen among processors using the MPI_ALLREDUCE command. If the size of the KMC domain is one reaction choice per element per step, then the maximum reaction rate among volume elements is chosen in each processor (using totalRateVol) and then the maximum of those is chosen using the same MPI_ALLREDUCE command.

2.3.1.1 Allocate list used to track updated defects

The pointer list defectUpdate, which is of derived type defectUpdateTracker, is first allocated before any timesteps or reactions are chosen. This will be used to store which defects have changed based on the reactions chosen, and will then be used to guide which reaction rates to update. The reason for initializing this list early is that the KMC loop proceeds to go into different options based on the singleElemKMC toggle.

2.3.2 Timestep choice

Here, we have several options based on the toggles that have been set for this simulation:

1. If the singleElemKMC toggle is set to 'yes', then a timestep is generated using the GenerateTimestep subroutine in the master processor only. The formula for the timestep generation is:

$$\tau = \frac{1}{a} \log \left(\frac{1}{r_1} \right) \tag{2.4}$$

   where a is the maximum reaction rate among all single volume elements (see Section 2.3.1) and r_1 is a double precision random number generated using subroutine dprand.

2. If the singleElemKMC toggle is set to 'no' and the implantScheme toggle is set to 'explicit', cascades are implanted at regular intervals in all processors simultaneously such that the dpa rate matches the DPARate variable. In this case, if the elapsed time has passed such that a cascade implantation step is next, the timestep τ is set to zero for the instantaneous cascade implantation step. Otherwise, a timestep is generated using equation (2.4), with a in this case representing the greatest *total* reaction rate for all elements in a single processor (see Section 2.3.1).

3. If the singleElemKMC toggle is set to 'no' and the implantScheme toggle is set to 'MonteCarlo', then cascade implantation is included in the reaction choice section of the algorithm (Section 2.3.3) and a timestep is chosen at each step using equation (2.4). Note that in this case, the reaction rate for cascade choice is included in the computation of a, unlike in the previous scheme.

The timestep τ, which is generated only in the master processor, is then added to elapsedTime and transmitted to all other processors (NOTE: this transmission typically happens after other actions have been carried out, as it has no impact on the other aspects of the KMC loop).

2.3.3 Reaction choice

This section of the KMC algorithm also proceeds differently depending on the toggles chosen:

1. If the singleElemKMC toggle is set to 'yes', then one reaction must be chosen for each volume element (including the possibility of null events). Therefore, a list of all the reactions chosen is kept in a reaction derived type pointer list called reactionChoiceList. A reaction is then chosen in each cell using the subroutine chooseReactionSingleCell. The reaction is chosen by pointing reactionCurrent (a reaction derived type pointer) at the chosen reaction. The outline of chooseReactionSingleCell is as follows:

   (a) The global variable totalRate has already been set to the maximum total reaction rate for a volume element among all volume elements in all processors (see Section 2.3.1).

   (b) A random number $r_2 \in [0, 1]$ is generated and multiplied by totalRate (the value is stored in variable r2timesa).

   (c) If the value of totalRateVol(cell), or the total reaction rate within that volume element, is greater than r2timesa, then we will choose a reaction within this volume element. Otherwise, we will choose a null event and the reaction derived type pointer reactionCurrent is not associated.

   (d) If we are choosing a reaction within this cell, we point reactionCurrent at reactionList(cell), or the first reaction in the pointer list of reactions stored in this volume element.

   (e) We add the reaction rate of reactionCurrent to a variable called atemp (starting at zero) and continue forward through the pointer list by moving to reactionCurrent%next until the value of atemp is greater than r2timesa. Once this happens, reactionCurrent is pointing at the reaction that we have chosen and we exit the loop.

   (f) The number of reactions of different types chosen, such as annihilation reactions and Frenkel pair implant reactions, is then tracked for postprocessing purposes.

   After every reaction is chosen in each volume element, if a null event is not chosen, then the reaction chosen is added to the list reactionChoiceList described above.

2. If the singleElemKMC toggle is set to 'no', then one reaction is chosen in all volume elements inside the local processor's mesh. This reaction is chosen using subroutine chooseReaction, which points the reactionCurrent pointer at the chosen reaction. The outline of chooseReaction is as follows:

   (a) The global variable totalRate has already been set to the maximum total reaction rate for all volume elements within a single processor (see Section 2.3.1).

   (b) A random number $r_2 \in [0, 1]$ is generated and multiplied by totalRate (the value is stored in r2timesa).

   (c) The variable atemp_cell is set to zero and then a loop passes through each volume element. In each volume element, the value of totalRateVol(cell), or the total reaction rate for reactions inside that volume element, is added to atemp_cell. If atemp_cell is greater than r2timesa, then we will choose a reaction within this element.

   (d) If we are choosing a reaction within this element, the value of atemp is set to the value of atemp_cell in the previous step. Then the reaction pointer reactionCurrent is pointed at the reaction list inside that volume element. We add the value of the reaction rate for reactionCurrent to atemp and continue forward through the pointer list by moving to reactionCurrent%next until the value of atemp is greater than r2timesa. Once this happens, reactionCurrent is pointing at the reaction that we have chosen and we exit the loop.

   (e) The possibility to choose a reaction within one of the fine meshes also exists in this formulation. If no reaction is chosen in any of the coarse volume elements, a cascade is then chosen using the same procedure used to choose a volume element, except that instead of taking reaction rates from the totalRateVol variable, total reaction rates for each cascade are taken from the CascadeCurrent%totalRate variable (CascadeCurrent is a cascade derived type pointer list of cascades, including defect and reaction lists).

   (f) Once a cascade volume element is chosen, a reaction is chosen within that volume element in the same way that a reaction was chosen in a coarse mesh volume element in the previous steps.

   (g) The number of reactions of different types chosen, such as annihilation reactions and Frenkel pair implantation reactions, is then tracked for postprocessing purposes.

   As only one reaction is chosen per processor per step in this method, a list of reactions chosen does not need to be created and the algorithm moves to the next step with reactionCurrent pointing at the chosen reaction.

2.3.4 Update defect list

This is the most complex step of the KMC algorithm in this code. If the singleElemKMC toggle is set to 'no', this step is carried out by the updateDefectList subroutine. Otherwise it is carried out by the updateDefectListMultiple subroutine. Due to the similarity in these subroutines, only the structure of updateDefectList will be described here. The main difference between the two is that updateDefectListMultiple goes through a list of chosen reactions and updates defects accordingly, while updateDefectList only updates defects due to one reaction chosen.

The structure of updateDefectList is as follows:

1. If the reaction chosen during the previous step was cascade implantation, then the following steps are carried out:

   (a) If the meshingType toggle is set to 'adaptive', then we must create a fine mesh when implanting a cascade. This is a complicated process consisting of the following steps:

      i. One cascade is chosen from the list of cascades using the ChooseCascade subroutine.

      ii. The pointer CascadeCurrent (cascade derived type pointer) is pointed to the global cascade list ActiveCascades and then moved to the end of the list.

      iii. A new cascade is allocated at the end of the list of cascades using CascadeCurrent. The coarse mesh volume element that this cascade resides within as well as the ID number of this cascade is stored within the derived type for this cascade.

      iv. The volume of the coarse mesh element is decreased by the amount of the volume of the fine mesh being created.

v. The fine mesh is initialized using the InitializeFineMesh subroutine, which initializes the defect and reaction lists in CascadeCurrent (these are pointer lists of type defect and reaction, similar to defectList and reactionList in the coarse mesh). InitializeFineMesh then removes defects from the coarse mesh and places them into the fine mesh with probability given by the ratio of the fine mesh volume to the coarse mesh volume. Defects placed inside the fine mesh in this way are done so randomly.

vi. A list of defects to be implanted in the fine mesh from the cascade is created and stored in the pointer list defectStoreList.

vii. The defects placed in the fine mesh during InitializeFineMesh are then randomly combined with defects in defectStoreList with a probability given by the ratio of the cascade volume to the volume of the fine mesh. The subroutine defectCombinationRules is used to determine the resultant defect type when two defects are combined (for example, if a He-SIA cluster can form). This subroutine is mostly hard-coded.

viii. The resulting defects in defectStoreList (after combination with defects in the fine mesh) are then implanted into the fine mesh using the coordinates of the defects in defectStoreList to determine which volume element they are implanted into.

ix. The defect update tracker derived type variable defectUpdate (pointer list), which is used in Section 2.3.5 to know which defects have changed in order to update the list of possible reactions, is created based on all of the defects placed into the fine mesh.

x. The defectStore pointer list is deallocated and memory is freed.

(b) If the meshingType toggle is set to 'nonAdaptive', then the entire cascade is placed within a single volume element (which should be the same volume as the cascade to preserve initial defect densities). No fine mesh is created. In this case, the following steps are carried out:

i. A list of defects to be implanted in the fine mesh from the cascade is created and stored in the pointer list defectStoreList.

ii. The defects already present inside the volume element that the cascade is being implanted into are combined randomly with the defects in defectStoreList - each defect already present in the volume element is randomly assigned a defect in the cascade with which to combine, and the resulting defect type is determined with defectCombinationRules.

iii. All (modified by the previous step) defects in defectStoreList are implanted into the volume element that the cascade is being implanted into.

iv. The defect update tracker derived type variable defectUpdate (pointer list), which is used in Section 2.3.5 to know which defects have changed in order to update the list of possible reactions, is created based on all of the defects placed into the volume element from the cascade.

v. The defectStore pointer list is deallocated and memory is freed.

2. If the reaction chosen is located within a fine mesh (inside a cascade that was implanted during a previous step), then the reactionCurrent variable will be pointing at the chosen reaction and CascadeCurrent will be pointing at the cascade that the chosen reaction is located inside. In this case, defects are updated according to the following steps:

(a) The reactants, listed in reactionCurrent%reactants, are removed from the fine mesh volume element defect list (CascadeCurrent%localDefects(cellNumber)). At the same time, a new item is added to defectUpdateList containing which reactants were removed from the system (used to track which reaction should be updated in Section 2.3.5).

(b) If the grainBoundaryToggle toggle is set to 'yes', then defects that diffuse between volume element can be removed with a probability equal to the ratio of the volume element size to grain size.

(c) The products, listed in reactionCurrent%products, are added to the fine mesh volume element defect list (CascadeCurrent%localDefects(cellNumber)). At the same time, a new item is added to defectUpdateList containing which products were added to the system (used to track which reaction should be updated in Section 2.3.5).

(d) If the reaction chosen is diffusion from the fine mesh to the coarse mesh, then the code checks whether the coarse volume element has neighbors that are in another processor using myMesh%neighborProcs. If so, then this defect is added to the localBuffer list, which is used to send information to neighboring processors about which defects have changed in elements on the boundaries of adjoining processors. Defects are then added to the coarse mesh element defect list (defectList(CascadeCurrent%cellNumber)).

3. If the reaction chosen is located within the coarse mesh, then the reactionCurrent variable will be pointed at the chosen reaction and CascadeCurrent will not be associated. In this case, defects are updated similarly to the case of choosing a reaction in the fine mesh, but more care is taken to create buffers of information that need to be sent between neighboring processors if defects have changed near the boundary of another processor.

   (a) The reactants, listed in reactionCurrent%reactants, are removed from defectList(reactionCurrent%cellNumber). At the same time, a new item is added to defectUpdateList containing which reactants have been removed.

   (b) If this volume element is on the boundary of another processor (check using myMesh(reactionCurrent %cellNumber) %neighborProcs), then this defect is added to the localBuffer array which will be sent to the neighboring processor later.

   (c) If the grainBoundaryToggle toggle is set to 'yes', then defects that diffuse between volume elements can be removed with a probability equal to the ratio of the volume element size to grain size.

   (d) In the next set of steps, the products of the reaction are added to the system. However, due to the parallel nature of this code, the products of a reaction chosen in the local processor in the coarse mesh could end up in a volume element in a different processor (due to diffusion). Therefore, adding products to the system consists of the following steps:

      i. If the cellNumber variable inside reactionCurrent associated with this product is -1, then the product is leaving the system via a free surface. Thus the code does nothing and the product is not added to any defect lists.

      ii. If the defect is diffusing from the coarse mesh into a fine mesh (cascade) located within this coarse mesh volume element, the defect is added to defectUpdate (for updating reaction lists later) and the defect is also added to the defect list of a randomly chosen volume element within the fine mesh. In this case, no information needs to be passed to neighboring processors as the fine mesh does not border other processors.

      iii. If the defect is diffusing from a volume element in the local processor to a different processor, then the defect is first added to a boundary volume element inside the local variable myBoundary (derived type boundaryMesh). This is a separate mesh that tracks all of the defects in nearby processors to the local processor, for the purpose of calculating accurate diffusion rates without communicating between processors more than to transmit the information in the boundary. The defect is simultaneously added to defectUpdate (for updating reaction rates later) and to a bndryBuffer array which contains information on defects that have diffused form this processor to a neighboring one.

      iv. Otherwise, all products remain in the coarse mesh of the local processor. In this case, the products are added to the local defectList(reactionCurrent%cellNumber), the defect is added to defectUpdate for updating reaction rates later, and if the volume element is on the boundary of another processor the defect is added to the localBuffer array which will be used to tell neighboring processors which defects have changed near their boundaries.

4. At this point, all local defect updating has been carried out. The next steps are associated with communicating between processors what defects have either diffused between processors or what defects have changed on the boundaries of processors. During the preceding steps, two buffer lists have been created: localBuffer, which contains all defects that have changed in the local processor but which may be on the boundary of another processor (and therefore are needed by the neighboring processor to correctly calculate diffusion rates), and bndryBuffer, which contains all defects that have diffused from the local processor to an neighboring processor. The steps associated with communication are as follows:

   (a) The local processor sends all of its neighbors the number of defects in the local buffer and boundary buffer.

   (b) If the numbers of defects in the local and boundary buffers are nonzero (defects have changed in the boundary or have diffused between processors), the defects in the local and boundary buffers are sent to the neighboring processors.

   (c) The local processor receives from all of its neighbors the number of defects in the *neighbor's* local and boundary buffers.

   (d) If the number of defects in the neighbor's local and boundary buffers are nonzero, then the defects in the neighbor's local and boundary buffers are received from the neighboring processors.

(e) All defects that are received from neighbors have either changed on the boundary of the local processor or have diffused from the neighboring processor into the local processor. In the first case, these defects are still not in the local processor but instead are added to the myBoundary mesh, which is used to calculate diffusion rates between volume elements in different processors. In the second case, defects that have diffused into the local processor from neighboring processors are added to defectList, the list of defects in the local processor's volume elements.

(f) A new defect is added to defectUpdateList for each defect received, whether it is in the local processor or on the boundary in a neighboring processor. This will be used to update reaction rates in Section 2.3.5.

(g) As a final check, if a defect has diffused from a neighboring processor into the local processor in a volume element that is *also* on the boundary of a third processor, then that defect is now in the boundary of the third processor as well as being added to the local processor. Therefore, a final buffer is created for all defects that have been added to the local processor through diffusion that are also on the boundary of a third processor. This buffer is then sent to all neighboring processors (first the size of the buffer, which may be zero, and then the buffer itself if nonzero).

(h) The local processor receives from all its neighbors the number of defects in the final buffer

(i) If nonzero, the local processor receives the final buffer from all its neighbors. These are all defects that are only in the boundary of the local processor, and these defects are therefore added to myBoundary and not to defectList. A new defect is added to defectUpdateList for each boundary defect received in this way.

At the end of the updateDefectList subroutine, all defects in the local mesh defectList have been updated based on the reactions chosen in the local mesh. In addition, any defects that have diffused from this processor to a neighboring processor or vice versa have been subtracted/added to the local mesh. Finally, defects on the boundary mesh myBoundary, representing defects in neighboring processors that are needed to calculate diffusion rates, have been updated and all boundary behavior has been communicated with neighboring processors. A list of defects that have been changed in defectList and myBoundary has been created for use in Section 2.3.5.

2.3.5 Update reaction list

After all defects have been updated, the list of allowed reactions is updated using subroutine updateReactionList. The information passed into this subroutine is defectUpdate, a defect derived type pointer list with all of the defects that have changed due to the reaction(s) chosen during this KMC step. UpdateReactionList carries out the following steps:

1. DefectUpdateCurrent (also a defect pointer) is pointed at defectUpdate. The following steps are taken for each element of defectUpdate by pointing defectUpdateCurrent at each element successively. The loop ends when defectUpdateCurrent comes to the end of defectUpdate.

2. If defectUpdateCurrent is located within the local processor, then a defect has changed in a local volume element. In this case, if the defect updated is within the coarse mesh, then the following subroutines are called which update reaction lists and reaction rates in the coarse mesh:

   (a) addSingleDefectReactions adds (or updates reaction rates for) single-defect reactions associated with defectUpdateCurrent in the coarse mesh: dissociation, trapping, and capture by sinks. Allowed reactions are input from a file (see Section 2.2.2).

   (b) addMultiDefectReactions adds (or updates reaction rates for) multi-defect reactions such as clustering associated with defectUpdateCurrent and every other defect present in the volume element.

   (c) addDiffusionReactions adds (or updates reaction rates for) diffusion reactions from this volume element to neighboring volume element (associated with defectUpdateCurrent). The numbers of defects of the same type as defectUpdateCurrent in neighboring volume elements are used to calculate diffusion rates. When diffusion rates are negative, reactions are not added.

   (d) If the polycrystal toggle is set to 'yes', then diffusion reactions between neighboring elements in different grains are treated instead like diffusion reactions to a free surface - the grain boundary acts as a perfect sink and there are no products (the reactants are deleted from the system).

In all cases, when reactions are either added, deleted, or have their rates updated, the totalRate in for the entire processor as well as totalRateVol for each volume element that the reaction is inside is updated accordingly. This avoids the need to re-calculate the total reaction rate during each step.

3. If the if defectUpdateCurrent is inside the local processor but inside a cascade as well, the same subroutines listed in the previous step are called except for the fine mesh. Thus, instead of addSingleDefectReactions, the simulation calls addSingleDefectReactionsFine. Otherwise, this step is the same as the previous step.

4. If defectUpdateCurrent is within the boundary mesh, then it is a defect that has changed in the neighboring processor and not this one. Therefore, only diffusion rates between the neighboring processor's volume element and any adjacent volume elements in the local processor need to be updated. Therefore only the addDiffusionReactions subroutine is called in this case.

At the end of the subroutine updateReactionList, defectUpdate is deallocated (all memory is erased) and all defectUpdate and defectUpdateCurrent pointers are nullified.

2.3.6 Delete cascade (if necessary)

If a reaction is chosen within a cascade and the resulting total reaction rate within that cascade (measured using subroutine totalRateCascade) drops below the cascadeReactionLimit initially read in, the cascade must be removed. This is done using the subroutine releaseFineMeshDefects:

1. The volume of the coarse mesh element that the cascade resides within is increased by the volume of the fine mesh

2. A defect type pointer defectCurrentFine is pointed at the list of defects within the cascade, and a defect type pointer defectCurrent coarse is pointed at the list of defects in the coarse mesh element. A cascade type pointer cascadeCurrent is pointed at the cascade chosen (it contains lists of defects in each volume element, which defectCurrentFine is pointing at)

3. DefectCurrentFine is moved through all defects in the fine mesh and defectCurrentCoarse is used to add those defects to the coarse mesh.

4. All of the defects in all of the volume elements in the cascade are deallocated (these are pointer lists).

5. All of the reaction lists in all of the volume elements in the cascade are deallocated (these are pointer lists).

6. The cascade itself is deallocated and the list of active cascades is appropriately modified to keep the pointer chain intact.

After this step, the subroutine resetReactionListSingleCell is used to reset all reactions in the coarse mesh volume element containing the cascade that was just deleted, using the new volume and all of the cascade defects which were deposited into the coarse mesh element.

If a cascade is deleted in a volume element that is adjacent to volume elements in neighboring processors, the diffusion rates between the local volume element and the volume elements in neighboring processors may have changed. Therefore, this information must be passed to neighboring processors at this step. The subroutine cascadeUpdateStep performs this task:

1. If no cascade was destroyed this step, the subroutine does nothing. It sends a signal to its neighbors that nothing has changed.

2. If a cascade was destroyed this step in an element bounding another processor, a buffer is created containing:

   (a) The cell number of the volume element in the neighboring processor

   (b) The cell number of the volume element in the local processor that the cascade was removed from

   (c) The number of defects that are now in the coarse mesh of the volume element that the cascade was removed from

(d) The number of different buffers of defects that will be sent to the neighboring defect (this was created because MPI_SEND cannot send more than a certain size of data packet at a time, so multiple SEND actions have to be called for large lists of defects)

(e) The new volume of the volume element in the local processor that the cascade was removed from

3. A defect buffer is then sent to the neighboring processor containing all of the defects in the local volume element that the cascade was deleted from.

4. This defect buffer is divided into several smaller MPI_SEND commands if it is too large

5. Information is received from neighboring processors about whether or not cascades have been deleted and whether or not defect buffers are going to be received.

6. If yes, defect data is received from neighboring processors and defects are added to the myBoundary mesh defect lists.

7. Diffusion rates from local volume elements into neighboring volume elements in which cascades have been deleted are updated using addDiffusionReactions.

2.3.7 Check reaction rates (every n steps)

The total reaction rate totalRate in the local processor is updated as the reaction list is updated, as described above. However, in the event that there is an error causing reaction rates to not update correctly OR numerical drift due to machine precision causes the total reaction rate to vary from the value it should be, the double precision function TotalRateCheck is called. This subroutine carries out the following steps:

1. reactionCurrent is pointed at every local volume element in reactionList and the reaction rates of all defects in all reaction lists are summed in the local mesh.

2. If the total reaction rate inside a given volume element is significantly different from the reaction rate for that volume element stored in totalRateVol, then it outputs an error and changes the reaction rate inside that volume element to the correct value in totalRateVol.

3. ReactionCurrent is then pointed at the reaction lists in each volume element of each cascade, and the process is repeated. If the total reaction rate stored within CascadeCurrent$\%$totalRate(cell) is different from the total rate computed for that fine mesh volume element, an error message is printed and the total reaction rate within that fine mesh volume element is reset to the value computed.

4. Finally, the reaction rate for all volume elements in the local processor, including fine meshes, is computed and compared to totalRate. If it is significantly different, an error message is output. The correct reaction rate is used as the output of this function and totalRate is reset using this value inside the main program.

The definition of 'significantly different' is set at 1.0 s^{-1} for the moment, and the amount of numerical drift of totalRate has not been thoroughly investigated. TotalRateCheck is called every 1000 timesteps during Frenkel pair implantation, or every time a new cascade is implanted during cascade implantation.

2.3.8 Outputs (every m steps or time increment)

If the simulated time or number of steps has reached a pre-set increment, the simulation pauses and outputs data to data files. This process is currently set to occur when the elapsed time passes a value defined by:

$$\mathrm{outputTime} = \frac{\mathrm{totalTime}}{200 \times 2^n} \tag{2.5}$$

where n is the number of these output steps that have already occurred. This gives outputs on a logarithmic timescale. This is an arbitrary choice, and could be changed to outputs at regular time intervals or more frequent logarithmic time intervals.

The output steps consist of the following:

1. Compute the total number of cascades/Frenkel pairs and helium implantation events using MPI_ALLREDUCE (over all processors)
2. Compute the total DPA in the entire system (over all processors) using the total number of implantation events and the total volume (stored in myProc)
3. In the master processor only: output simulated time, dose, number of KMC steps, number of cascades/Frenkel pairs, number of helium implantation events, and the computation time.
4. At this point, several output options are available and can be called by calling several different output subroutines. The main choices available at this point are:
   (a) outputDefects: This outputs the raw data of defect types and numbers inside each volume element into a file. This is typically not activated unless the user needs them for a specific reason, as these data files can take a large amount of memory.
   (b) outputDefectsBoundary: This outputs the TOTAL (not raw data) defects present inside the volume elements that have been labelled as grain boundary elements. It essentially ignores defects in other volume elements. This subroutine is used for identifying defects that are accumulating on grain boundaries during such simulations. This subroutine will only work if the variable numMaterials$= 2$.
   (c) outputDefectsTotal: This outputs the TOTAL (not raw data) defects present inside all volume elements in the simulation. It presents results as a list of defect types with the number of defects present of each type. Some post-processing information is also given in a separate file.
   (d) outputDefectsVTK: This outputs a VTK file, which can be used to make images and movies of the defect accumulation such as the one seen on the cover of this manual using ParaView.
   (e) outputDefectsXYZ: This outputs an XYZ file, which is used by other plotting programs besides ParaView. It has not been extensively used.
   (f) outputDefectsProfile: This outputs defect populations as a function of depth in the material. It was designed for use when simulating non-uniform radiation damage in a material and also applies to simulations with free surfaces in one direction.
   (g) outputDebugRestart: This outputs a file which is in the format needed to create a restart input file (see above). The purpose of this subroutine is to allow the user to start several simulations from the same starting defect population and radiation dose, in order to either debug an error that occurs after a long amount of computation time or to provide better statistics. The contents of the file created during this subroutine can be converted easily into an input file that can be used to start a simulation from a non-pristine material.

2.3.9 Exit KMC loop

If the variable elapsedTime is greater than totalTime, we have completed the simulation and the final output is called. This is identical to the outputs available as described in Section 2.3.8 above, although the user has the option of choosing different outputs during the final output step than were chosen at each of the intermediate output steps.

All subsequent steps that are taken have to do with memory management, deallocation, and other issues that are not part of the kinetic Monte Carlo algorithm.

2.4 Other actions

In this section, other important tasks that are carried out in the main section of the code are described. These tasks are not necessary for the kinetic Monte Carlo algorithm to operate but are linked to the desire to provide the user with more simulation options, as well as memory management.

2.4.1 Initialize annealing

After the main kinetic Monte Carlo loop described in Section 2.3, the simulation either exits the KMC loop entirely and proceeds to the final output, or it can step into a secondary KMC loop that is used to simulate annealing.

The simulation steps into this secondary loop only in the case that the variable annealTime is greater than 0. The procedure for initializing the annealing KMC loop is the following:

1. The subroutine annealInitialization() is called, which carries out the following tasks:
   (a) The global temperature is set to the annealing temperature (input from parameters.txt).
   (b) All implantation rates are set to zero (since we are annealing, there is no longer additional damage being added to the system)
2. The reaction rates of all allowed reactions inside all volume elements are reset using the new temperature by calling the subroutine resetReactionLIstSingleCell(cell) for all volume elements.
3. Double precision function TotalRateCheck() is once again used to ensure that all total reaction rates (for volume elements, cascades, and the global system) have been calculated correctly and the total rate inside the local processor is reset.
4. The simulation steps into a secondary KMC loop, which operates using the same principles as the previously described KMC loop but with all implantation rates set to zero and the temperature changed. It stays in this loop until the elapsed time is greater than the sum of both the implantation time and the annealing time. Outputs are also carried out similarly to those described in Section 2.3.8, using a similarly defined logarithmic scale on output times (this is arbitrary and can be changed by the user)

It should be noted that an additional option is available to the user that allows annealing through multiple steps, each at a different temperature. This is triggered by the annealSteps input variable, which is read in from parameters.txt. If this variable is greater than 1, then annealing will be divided into steps of equal length (in time). During each step, the temperature will be incrementally increased either additively or multiplicatively, using toggle annealType to define which type of annealing is carried out. This toggle is also found in parameters.txt. See examples for additional details.

2.4.2 Deallocate memory: 1

After the final output, all pointer lists are cleared by iteratively passing through them and deallocating all derived type variables in each element of each pointer list, then deallocating the elements of the point list themselves. This is done using several subroutines:

- releaseFineMeshDefects()
- resetReactionListSingleCell() (for both volume elements as well as the coarse mesh reaction list)
- deallocateDefectList()
- deallocateReactionList()
- deallocateBoundaryDefectList()

The variable totalRateVol, containing the total reaction rates inside each volume element in the coarse mesh, is also deallocated at this point.

Note that in this section, only memory that is used for a specific simulation is deallocated. This allows the simulation to repeat several times, and other memory such as input data read from files is deallocated later

2.4.3 Loop over multiple simulations

If multiple simulations are being carried out for the same irradiation conditions, then at this point all output files are finalized and the code loops up to initialization of random seeds, defect lists, reaction lists, and output files (beginning with Section 2.2.6). The number of simulations to read in is stored in variable numSims, read in from parameters.txt.

2.4.4 Legacy code: Search for sink efficiency

This is code that was developed for using a 1-D cluster dynamics model to fit the sink efficiency of a grain boundary. It is legacy code and is no longer used. Any further simulations of grain boundaries that report sink efficiency η do so by calculating it using the following formula:

$$\eta_{\mathrm{V,I}} = 1 - \frac{\mathrm{num\ released}}{\mathrm{num\ trapped}} \tag{2.6}$$

where $\mathrm{num\ released}$ and $\mathrm{num\ trapped}$ are the numbers of defects (vacancies, SIAs) released from and trapped on the grain boundary, respectively.

The sink efficiency search code can be removed but has been left in the code for the time being.

2.4.5 Deallocate memory: 2

After all simulations have been finished, the final memory deallocation is carried out. This consists of:

- Deallocating cascadeConnectivity
- deallocateCascadeList()
- deallocateMaterialInput()

These subroutines deallocate the data that was input at the beginning of the program that was used in all simulations, such as the list of available cascades or the various defects, their binding and migration energies, and their allowed reactions.

2.4.6 Finalize MPI

The subroutine MPI_FINALIZE(ierr) is called to finalize the OpenMPI simulation. This is the final step of the SRSCD code.

Chapter 3

Input and output files

3.1 List of toggles

Below are a list of the various toggles that can be chosen in SRSCD. All toggles must be set in the parameters.txt file. All toggles that have 'yes' or 'no' options are set to 'no' as a default.

The following toggles MUST have a value assigned to them and must be located in the parameters.txt file in the correct order (see example parameters.txt file):

1. **meshType:** 'uniform' or 'nonUniform', non-uniform mesh not currently implemented
2. **strainField:** 'yes' or 'no', toggles whether there is a strain field that impacts diffusion. This has been implemented at a basic level only and its use has not been investigated.
3. **debugRestart:** 'yes' or 'no', toggles restarting with defect state given by input file
4. **implantType:** 'cascade' or 'FrenkelPair', toggles cascades or FP damage
5. **implantScheme:** 'MonteCarlo' or 'explicit', toggles explicit cascade implantation scheme (works for cascades only)
6. **meshingType:** 'adaptive' or 'nonAdaptive', toggles adaptive cascade implantation (works for cascades only)
7. **implantDist:** 'uniform' or 'nonUniform', toggles non-uniform defect distribution due to ion beam profiles, does not work with adaptive meshing

The following toggles are read in the subroutine readParameters() and can be omitted from the parameters.txt file. If so, they are set to their default value.

1. **annealType:** 'add' or 'mult', toggles consecutive annealing steps with additive temperature or multiplied temperature (used for isochronal annealing)
2. **grainBoundaryToggle:** 'yes' or 'no', toggles absorption of fast-moving defects by grain boundaries that are not explicitly represented
3. **heSIAToggle:** 'yes' or 'no', toggles formation of He-SIA clusters (if 'yes', need to add values in Defects.txt file associated with those defects)
4. **Polycrystal:** 'yes' or 'no', toggles polycrystal simulations (each grain has different material ID but uses the same Defects.txt input file)
5. **vtkToggle:** 'yes' or 'no': toggles .vtk file output
6. **xyzToggle:** 'yes' or 'no': toggles .xyz file output
7. **outputDebug:** 'yes' or 'no': toggles output of a restart file that can be used to restart simulation
8. **singleElemKMC:** 'yes' or 'no': toggles one KMC domain per volume element or one KMC domain per processor
9. **sinkEffSearch:** 'yes' or 'no': toggles loop used to search for sink efficiency
10. **postprToggle:** 'yes' or 'no': toggles output of postpr.txt file
11. **totdatToggle:** 'yes' or 'no': toggles output of totdat.txt file
12. **rawdatToggle:** 'yes' or 'no': toggles output of raw data files
13. **profileToggle:** 'yes' or 'no': toggles output of damage profile as a function of depth

3.2 Standard input files

In this section, example versions of standard input files are given.

3.2.1 Parameters.txt file

This file controls the simulation conditions and the majority of the toggles that are available in SRSCD. A detailed description of each toggle will be given below.

parameters.txt

```
!****************************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!****************************************************************************************************

!Tell computer location of mesh input file
meshFile
Mesh_Bicrystal.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
yes

!Tell computer the name of the data file containing the strain field
strainFile
StrainFile.txt

!Tell computer the name of the data file containing dipole tensors
dipoleFile
DipoleTensors.txt

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
no

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
debugRestart.in

!Tell computer if the number of materials we are reading in
numMaterials
2

!Tell computer the location of the material input file(s)
materialFile
Defects_Fe.txt

materialFile
Defects_GB.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
FrenkelPair

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
Fe_Cascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
nonAdaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!****************************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!****************************************************************************************************

start                       !begin parameters

temperature               !Temperature, in K
293d0

dpaRate                       ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
1d-7

HeDPA                         ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                                !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                !(Fe)                        1.17d-2                !(Cu)               ! atomsize (nm^3)

burgers
.287d0                        !(Fe)                        .36d0                        !(Cu)               ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
1d-2                       ! total DPA in simulation 1

annealTemp
293d0                       !Annealing temperature, in K

annealSteps
1                              ! number of steps in the annealing process
```

```
annealTime
0d0                                ! total anneal time (s)

annealType
add                                ! (add) for adding constant temperature increment at each anneal step, (mult) for multiplying the temperature by a constant at each anneal st

annealTempInc
0d0                                ! Temperature increment for each annealing step (either added or multiplied by)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
no

grainSize
33000d0                         ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
1d-5                        ! dislocation density (nm^-2)

impurityConc
30d-6                        !30d-4                        ! carbon impurity concentration (atomic fraction)

max3DInt
4                               !maximum size for SIA defect to diffuse in 3D

cascadeVolume
1000                        !volume of cascade (nm^3) - used for cascade-defect interactions

HeSIAToggle
no                               !Toggle whether or not we are allowing He-SIA clusters to form ('yes' or 'no')

SIAPinToggle
no                               !Toggle whether or not we are allowing HeV clusters to pin SIA clusters ('yes' or 'no')

SIAPinMin
1                               !Smallest size of SIA that can pin at HeV clusters

numSims
1                               !number of times to repeat simulation

end                               !Tag for the end of the parameters file

!*********************************************************************************************
!Adaptive meshing parameters
!*********************************************************************************************

fineStart                ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                               ! Length of one cascade implantation element (nm)

numxFine
6                               ! number of cascade elements in x-direction (fine mesh)

numyFine
6                               ! number of cascade elements in y-direction

numzFine
6                               ! number of cascade elements in z-direction

end                               ! Tag for the end of meshing parameters
```

The first half of the parameters.txt file includes toggles that must be present in order for SRSCD to operate without errors. These toggles must be in order. A description of each is below:

1. **meshFile:** This gives the location of the file with the mesh in it. See below for a description of the mesh file. This file also toggles whether the simulation has periodic boundary conditions or free surfaces in the z-direction.
2. **meshType:** 'uniform or 'nonUniform': this is a toggle that can only be set to 'uniform' currently.
3. **strainField:** 'yes' or 'no': this is a toggle that identifies whether a strain field should be input from a file. This is used to calculate stress-assisted diffusion. It has been implemented but not tested.
4. **strainFile:** If strainField is 'yes', this contains the location of the file with the strains in it.
5. **dipoleFile:** If strainField is 'yes', this contains the location of the file with dipole tensors for the various defects whose diffusivity is impacted by strain.
6. **debugRestart:** 'yes' or 'no': this is a toggle that identifies whether the simulation is started using an input file. If 'yes', the simulation is started from a non-zero dpa, non-zero elapsed time, and defects already present.
7. **debugRestartFile:** If debugRestart is 'yes', this contains the location of the file with the defects present at the beginning of the simulation, as well as the dose and elapsed time.
8. **numMaterials:** This is the number of distinct materials in the system. Each material has its own set of defects that can exist, their migration and binding energies, and the reactions that are allowed between defects (as well as reaction rates). For simulations of grain boundary defect accumulation, the elements that represent the grain boundary are given a separate material ID.

9. **materialFile:** This contains the location of the file with the various allowed defects, energies, and reactions for each material type.
10. **implantType:** 'FrenkelPair' or 'Cascade': this is a toggle that chooses cascade or Frenkel pair implantation.
11. **implantScheme:** 'MonteCarlo' or 'explicit': this is a toggle that chooses between two schemes for implanting cascades. Explicit cascade implantation saves some computation time in a parallel configuration. This toggle should be set at 'MonteCarlo' if 'FrenkelPair' has been chosen above.
12. **cascadeFile:** If 'Cascade' has been chosen, this contains the location of the file with the cascades in it. See below for example of this file.
13. **meshingType:** 'adaptive' or 'nonAdaptive': this is a toggle that turns on adaptive meshing when cascade damage has been chosen. This saves some computation time but requires the use of large volume elements.
14. **implantDist:** 'uniform' or 'nonUniform': this is a toggle that switches between uniform implantation and implantation given by a depth profile in the z-direction
15. **implantFile:** this contains the location of the file with the dose rate depth distribution (see below for example of file).

The second half of the parameters are read in during the subroutine readParameters() and do not need to be in a set order. In addition, if any of these parameters are missing from the parameters.txt file, they are set to their default value and the code runs without error.

A list of all allowed parameters in this section is given below:

1. **temperature:** implantation temperature (in K)
2. **dpaRate:** implantation dose rate (in dpa·s^{-1}) (use average if non uniform implantation distribution)
3. **HeDPA:** helium to dpa ratio (use average if non uniform implantation distribution)
4. **atomSize:** volume of a lattice site (in nm^3)
5. **burgers:** burgers vector of the material (for determining volume of SIA loops), in nm
6. **totalDPA:** total radiation dose simulated, in dpa
7. **annealTemp:** temperature of annealing (if several steps, temperature of the first step)
8. **annealSteps:** number of distinct annealing steps
9. **annealTime:** total time of annealing, divided between annealing steps
10. **annealType:** 'add' or 'mult', identifies if annealing temperature increments are additive or multiplicative
11. **annealTempInc:** amount to add or multiply temperature by at each annealing step
12. **grainBoundaries:** 'yes' or 'no', toggles whether defects are removed after diffusing a set distance, determined by grainSize
13. **grainSize:** if grainBoundaries is 'yes', the size of the grains being simulated. Used to determine the probability of removing diffusing defects from the system.
14. **dislocDensity:** dislocation density (in nm^{-2}), used to calculate the probability of defect removal at dislocation sinks (if that reaction is include in the defects input file)
15. **impurityConc:** concentration of (carbon) impurity atoms, used to pin SIA loops (change from mobile to immobile)
16. **max3DInt:** maximum size of SIA loops that are treated as 3D-migrating defects. Above this size, they act as 1D-diffusing loops. This parameter is mostly legacy as this transition is now taken care of in the defects input file.
17. **cascadeVolume:** Volume (in nm^3) of a displacement cascade. Used to calculate the probability of interaction between cascades and defects already present in the fine mesh.

18. **cascadeRxnLimit:** Reaction rate (in s^{-1}) inside a fine mesh at which point it will be considered 'annealed' and the fine mesh is destroyed. This is an input set by the user.
19. **HeSIAToggle:** 'yes' or 'no': toggles the formation of helium-SIA clusters. Must be accompanied by reaction rates in defect input file. Most simulations of helium in irradiated materials do not account for the possibility of helium-SIA clusters forming.
20. **SIAPinToggle:** 'yes' or 'no': toggles whether the presence of HeV clusters can pin SIAs (without forming He-SIA clusters). This acts in a similar manner to the impurityConc mechanism.
21. **SIAPinMin:** Minimum size of SIA that can be pinned by HeV cluster (rather than annihilate with the V).
22. **numSims:** Number of times to repeat the simulation
23. **polycrystal:** 'yes' or 'no': toggles whether the simulation is a polycrystalline domain. This allows only a single material input file although the material ID number for each grain is different.
24. **numGrains:** Number of grains inside a polycrystalline domain (if polycrystal is 'yes')
25. **vtkToggle:** 'yes' or 'no': Toggles VTK output (for paraView)
26. **restartToggle:** 'yes' or 'no': Toggles output of a debug restart file at each output step
27. **singleElemKMC:** 'yes' or 'no': Toggles a different mode of running the parallel KMC algorithm: choose a reaction for every volume element, rather than one reaction in each processor. This was mainly designed to evaluate the parallel performance of the SRSCD code, and is not typically beneficial.
28. **sinkEffSearch:** 'yes' or 'no': Toggles a section of code at the end of the main program used to search for sink efficiency in a bicrystal. This toggle is mostly legacy code and should be set to 'no'
29. **alpha_v, alpha_i, conc_v, conc_i:** parameters for sinkEffSearch, and should be ignored (legacy)

The final section in the parameters.txt file contains a standard mesh used for the fine mesh. If adaptive meshing is not selected, this section is ignored. It contains the following parameters:

1. **fineLength**: Length of a volume element in the fine mesh
2. **numxFine, numyFine, numzFine:** Number of elements in the x, y, and z-directions in the fine mesh

3.2.2 Mesh input file and meshGenerator input file

3.2.2.1 Case 1: Single material type

In this case, MeshGenInput.txt is used as an input into the stand-alone program MeshGen.f90 (located in the tools folder) which generates a mesh file that is of the correct format to be read in by the main program. An example version of MeshGenInput.txt is given below:

MeshGenInput.txt

```
!****************************************************************************************************
!
! This is an input file for the mesh generator program that will output mesh text files that are
! usable in SRSCD_par. The goal of this program is to have an easy and quick way to create uniform
! meshes for SRSCD without changing the (more demanding) inputs in SRSCD.
!
! Inputs: Volume element length (uniform volume elements only), meshType (periodic or free surfaces),
!                 number of elements in x, y, and z-directions.
!
! Output: Text file that can be read into SRSCD_par with mesh. Each element's center coordinates
!               as well as the global max/min coordinates in the x, y, and z directions will be given.
!               All materials will be of type 1.
!
!****************************************************************************************************

filename                !Determines the file name of the SRSCD_par input file
Mesh_ThinFilm.txt

meshType                !Tells meshGen whether to create a material with periodic BC's or free surfaces in the z-dir ('periodic or freeSurfaces')
freeSurfaces

length (nm)               !Specify the length of volume elements in nm (uniform cubic mesh only)
20d0

numx
```

```
10
numy
10
numz
5

end
```

This file contains the location of the output file written by MeshGen, the mesh type (free surfaces or periodic boundary conditions), the length of the volume elements, and the number of volume elements in the x, y, and z-directions. These inputs are used to generate a mesh file.

The output of MeshGen is an input file called Mesh_ThinFilm.txt (as seen in MeshGenInput.txt above). Part of this file is shown below:

Mesh_ThinFilm.txt

```
!********************************************************************************
!This file contains a UNIFORM cubic mesh input for the parallel SRSCD
!code. The mesh will be read in via coordinates (x,y,z) of each element center
!as well as the material number of each element (eg. material 1=Fe, 2=Cu, etc)
!
!The first values that will be read in are the number of elements in each direction
!(x,y,z).
!
!********************************************************************************

!Specify mesh type (currently two types=periodic and freeSurfaces)
meshType
freeSurfaces

!Specify element lengths (nm)
length
  20.000000000000000

!Specify max and min of each coordinate (nm)
!NOTE: the actual max, min will be max, min +/- length/2 because these are assumed
!to be the center of the elements and not the edges
xminmax
  10.000000000000000        190.00000000000000
yminmax
  10.000000000000000        190.00000000000000
zminmax
  10.000000000000000        90.000000000000000

!Specify number of elements in each direction
numx
         10
numy
         10
numz
          5

!Coordinates of element centers (x, y, z, elementType) (nm)
!NOTE: these must be ordered in the same way as the connectivity matrix (loop x, then y, then z)
elements                        !tells computer that the following values are elements
  10.000000000000000        10.000000000000000        10.000000000000000                1
  30.000000000000000        10.000000000000000        10.000000000000000                1
  50.000000000000000        10.000000000000000        10.000000000000000                1
  70.000000000000000        10.000000000000000        10.000000000000000                1
  90.000000000000000        10.000000000000000        10.000000000000000                1
  110.00000000000000        10.000000000000000        10.000000000000000                1
  130.00000000000000        10.000000000000000        10.000000000000000                1
  150.00000000000000        10.000000000000000        10.000000000000000                1
  170.00000000000000        10.000000000000000        10.000000000000000                1
  190.00000000000000        10.000000000000000        10.000000000000000                1

  10.000000000000000        30.000000000000000        10.000000000000000                1
  30.000000000000000        30.000000000000000        10.000000000000000                1
  50.000000000000000        30.000000000000000        10.000000000000000                1
  70.000000000000000        30.000000000000000        10.000000000000000                1
  90.000000000000000        30.000000000000000        10.000000000000000                1
  110.00000000000000        30.000000000000000        10.000000000000000                1
  130.00000000000000        30.000000000000000        10.000000000000000                1
  150.00000000000000        30.000000000000000        10.000000000000000                1
  170.00000000000000        30.000000000000000        10.000000000000000                1
  190.00000000000000        30.000000000000000        10.000000000000000                1

  10.000000000000000        50.000000000000000        10.000000000000000                1
  30.000000000000000        50.000000000000000        10.000000000000000                1
  50.000000000000000        50.000000000000000        10.000000000000000                1
  70.000000000000000        50.000000000000000        10.000000000000000                1
  90.000000000000000        50.000000000000000        10.000000000000000                1
  110.00000000000000        50.000000000000000        10.000000000000000                1
  130.00000000000000        50.000000000000000        10.000000000000000                1
  150.00000000000000        50.000000000000000        10.000000000000000                1
  170.00000000000000        50.000000000000000        10.000000000000000                1
  190.00000000000000        50.000000000000000        10.000000000000000                1
etc ...
```

The parts of the generated mesh file are as follows:

1. **meshType**: set to periodic or freeSurfaces

2. **length**: gives the length of the volume elements in the mesh
3. **xminmax, yminmax, zminmax**: gives the minimum and maximum values of the coordinates of the centers of volume elements in the x, y, and z-directions. The true boundaries of the volume enclosed by the mesh are given by adding and subtracting half of the volume element length to these values.
4. **numx, numy, numz**: the number of volume elements in the x, y, and z-directions in this mesh
5. **elements**: gives the coordinates (x,y,z) of the centers of the volume elements and the material ID number (or grain ID number) of the volume elements

3.2.2.2 Case 2: Multiple material types

In this case, the mesh generator must use an additional input file that provides the material ID numbers of the various volume elements. The meshGenInput file for a periodic mesh with a single grain boundary in the center and 5 nm volume elements is as follows:

MeshGenInput.txt

```
!****************************************************************************************************
!
! This is an input file for the mesh generator program that will output mesh text files that are
! usable in SRSCD_par. The goal of this program is to have an easy and quick way to create uniform
! meshes for SRSCD without changing the (more demanding) inputs in SRSCD.
!
! Inputs: Volume element length (uniform volume elements only), meshType (periodic or free surfaces),
!                 number of elements in x, y, and z-directions.
!
! Output: Text file that can be read into SRSCD_par with mesh. Each element's center coordinates
!               as well as the global max/min coordinates in the x, y, and z directions will be given.
!               All materials will be of type 1.
!
!****************************************************************************************************

filename                !Determines the file name of the SRSCD_par input file
Mesh_Bicrystal.txt

meshType                !Tells meshGen whether to create a material with periodic BC's or free surfaces in the z-dir ('periodic or freeSurfaces')
periodic

length (nm)                !Specify the length of volume elements in nm (uniform cubic mesh only)
5d0

numGrains                !Specify the number of grains (for testing purposes only, will evenly distribute grains in z-direction)
2

iMPALEtoggle                !Toggle whether we are reading from an iMPALE polyxtal file
yes

iMPALEfilename                !name of iMPALE file to read in
Bicrystal.xyz

numx                        !If reading in from iMPALE file, these should match the data in the iMPALE file
10
numy
10
numz
21

end
```

Here, several additional options have been added to the meshGenInput file compared to the version shown in Section 3.2.2.1. These options are typically not necessary for standard meshes but are necessary for polycrystal or grain boundary simulations. They are:

1. **numGrains**: The number of grains or material types in the mesh
2. **iMPALEtoggle**: 'yes' or 'no', toggles whether the grain ID numbers will be read in from an external file (generated by iMPALE)
3. **iMPALEfilename**: the location of the file containing the grain ID numbers

The contents of Bicrystal.xyz are a list of the material ID numbers for each volume element generated by MeshGen. It must have the same number of elements as listed in MeshGenInput.txt:

Bicrystal.xyz

```
2100
#       I       J       K       MATID
```

```
1        1        1        1
2        1        1        1
3        1        1        1
4        1        1        1
5        1        1        1
6        1        1        1
7        1        1        1
8        1        1        1
9        1        1        1
10        1        1        1
1        2        1        1
2        2        1        1
3        2        1        1
4        2        1        1
5        2        1        1
6        2        1        1
7        2        1        1
8        2        1        1
9        2        1        1
10        2        1        1
1        3        1        1
2        3        1        1
3        3        1        1
4        3        1        1
5        3        1        1
6        3        1        1
7        3        1        1
8        3        1        1
9        3        1        1
10        3        1        1

etc ...
```

The first line of this file must include the number of volume elements in the file. The second line is skipped and is for comments. The following lines contain the location of the volume elements in the mesh (i, j, k positions) and the grain ID number or material ID number of each volume element.

After running MeshGen, the following mesh file is created:

Mesh_Bicrystal.txt

```
!********************************************************************************
!This file contains a UNIFORM cubic mesh input for the parallel SRSCD
!code. The mesh will be read in via coordinates (x,y,z) of each element center
!as well as the material number of each element (eg. material 1=Fe, 2=Cu, etc)
!
!The first values that will be read in are the number of elements in each direction
!(x,y,z).
!
!********************************************************************************

!Specify mesh type (currently two types=periodic and freeSurfaces)
meshType
periodic

!Specify element lengths (nm)
length
  5.0000000000000000

!Specify max and min of each coordinate (nm)
!NOTE: the actual max, min will be max, min +/- length/2 because these are assumed
!to be the center of the elements and not the edges
xminmax
  2.5000000000000000        47.500000000000000
yminmax
  2.5000000000000000        47.500000000000000
zminmax
  2.5000000000000000        102.50000000000000

!Specify number of elements in each direction
numx
         10
numy
         10
numz
         21

!Coordinates of element centers (x, y, z, elementType) (nm)
!NOTE: these must be ordered in the same way as the connectivity matrix (loop x, then y, then z)
elements                        !tells computer that the following values are elements
  2.5000000000000000        2.5000000000000000        2.5000000000000000                1
  7.5000000000000000        2.5000000000000000        2.5000000000000000                1
  12.500000000000000        2.5000000000000000        2.5000000000000000                1
  17.500000000000000        2.5000000000000000        2.5000000000000000                1
  22.500000000000000        2.5000000000000000        2.5000000000000000                1
  27.500000000000000        2.5000000000000000        2.5000000000000000                1
  32.500000000000000        2.5000000000000000        2.5000000000000000                1
  37.500000000000000        2.5000000000000000        2.5000000000000000                1
  42.500000000000000        2.5000000000000000        2.5000000000000000                1
  47.500000000000000        2.5000000000000000        2.5000000000000000                1
  2.5000000000000000        7.5000000000000000        2.5000000000000000                1
  7.5000000000000000        7.5000000000000000        2.5000000000000000                1
  12.500000000000000        7.5000000000000000        2.5000000000000000                1
  17.500000000000000        7.5000000000000000        2.5000000000000000                1
  22.500000000000000        7.5000000000000000        2.5000000000000000                1
  27.500000000000000        7.5000000000000000        2.5000000000000000                1
  32.500000000000000        7.5000000000000000        2.5000000000000000                1
  37.500000000000000        7.5000000000000000        2.5000000000000000                1
  42.500000000000000        7.5000000000000000        2.5000000000000000                1
  47.500000000000000        7.5000000000000000        2.5000000000000000                1
  2.5000000000000000        12.500000000000000        2.5000000000000000                1
  7.5000000000000000        12.500000000000000        2.5000000000000000                1
  12.500000000000000        12.500000000000000        2.5000000000000000                1
  17.500000000000000        12.500000000000000        2.5000000000000000                1
```

```
   22.500000000000000        12.500000000000000        2.5000000000000000              1
   27.500000000000000        12.500000000000000        2.5000000000000000              1
   32.500000000000000        12.500000000000000        2.5000000000000000              1
   37.500000000000000        12.500000000000000        2.5000000000000000              1
   42.500000000000000        12.500000000000000        2.5000000000000000              1
   47.500000000000000        12.500000000000000        2.5000000000000000              1

etc...
```

This mesh file is similar to the mesh file generated in Section 3.2.2.1, but in the **elements** section, the material ID number for each volume element is not necessarily equal to 1.

3.2.3 Defect input file

The standard defect input file (for defects in α-Fe) is given below. It contains the migration and binding energies of allowed defects as well as the allowed reactions in the system. Due to the complicated nature of this file, the entire standard defect input file for α-Fe is given below as a reference to the reader.

Defects_Fe.txt

```
!****************************************************************************************************
!This is the input file for iron (bulk).
!It includes:
!1) Number of allowed species and identity of each species (He, V, SIA_mobile, SIA_sessile)
!2) Allowed defects of each species and combinations (He_n, V_m, He_nV_m)
!3) Migration energies of each species and diffusion prefactors
!4) Binding energies of each species to single defects (eg. V_n->V+V_n-1). NOTE: defects with
!        multiple species (He_mV_n), need binding energies of each species allowed to dissociate
!        (single He or single V). Include possibility of larger (V_2) dissociation events as well.
!5) List of allowed single-defect reactions for each species and functional form of reaction rate,
!        including:
!5a)        Dissociation
!5b)        Diffusion between elements
!5c)        Removal (eg. by sinks)
!5d)        Conversion of species type (eg. SIA_mobile->SIA_sessile)
!
!6)        List of allowed multiple-defect reactions for each pair of species and functional form of
!        reaction rate (clustering reactions) - including final defect type
!        (eg. SIA_mobile+SIA_mobile->SIA_sessile)
!7)        List of allowed 0-defect reactions (cascade implantation), including:
!7a)        Cascade energies and avg. number of Frenkel pairs in each cascade energy
!7b)        DPA rate at each cascade energy
!7c)        Flag showing whether DPA rate is uniform or a distribution; if it is a distribution then
!                need to include functional form of DPA distrubtion as well as parameters input into
!                functional form.
!
!NOTE: functional forms 1 and 2 are reserved for two cases: zero and constant
!****************************************************************************************************

!Identify material number: Fe
material
1

!Define number of allowed species
!Here: 4 species: He, V, SIA_mobile, SIA_sessile
species
4

!List of allowed species, diffusion prefactors (nm^2/s), and migration energies (eV) for each

numSingle
21                              !Indicates there are 6 sizes of He that are treated individually
numFunction
5                               !Indicates that there is 1 functional form for all other He sizes

single                          !Indicates that we are about to read in single defect parameters
1        0        0        0                        !He_1
D0        5d11        Em        0.077d0
2        0        0        0                        !He_2
D0        3d10        Em        0.055d0
3        0        0        0                        !He_3
D0        1.5d10        Em        0.062d0
4        0        0        0                        !He_4
D0        5d9                Em        0.062d0
5        0        0        0                        !He_5
D0        1.6d9        Em        0.2d0
6        0        0        0                        !He_6
D0        3.9d9        Em        0.28d0
0        1        0        0                        !V_1
D0        8.2d11        Em        0.67d0
0        2        0        0                        !V_2
D0        8.2d11        Em        0.62d0
0        3        0        0                        !V_3
D0        8.2d11        Em        0.35d0
0        4        0        0                        !V_4
D0        8.2d11        Em        0.48d0
0        0        1        0                        !SIA_m 1
D0        8.2d11        Em        0.34d0
0        0        2        0                        !SIA_m 2
D0        8.2d11        Em        0.42d0
0        0        3        0                        !SIA_m 3
D0        8.2d11        Em        0.43d0
0        0        4        0                        !SIA_m 4
D0        8.2d11        Em        0.43d0
1        1        0        0                        !He_1V_1
D0        1.15d12        Em        2.57d0
1        2        0        0                        !He_1V_2
D0        4.1d10        Em        0.27d0
1        3        0        0                        !He_1V_3
```

```
D0        1.15d12        Em        1.42d0
2        1        0        0                        !He_2V_1
D0        1.16d11        Em        0.33d0
3        1        0        0                        !He_3V_1
D0        2d10        Em        0.31d0
4        1        0        0                        !He_4V_1
D0        3.68d9        Em        0.28d0
2        3        0        0                        !He_2V_3
D_0        7.82d9        Em        0.55d0


function                !Indicates that we are about to read in functional form parameters
1        0        0        0                        !He_n
nmin        7        nmax        -1
nmin        0        nmax        0
nmin        0        nmax        0
nmin        0        nmax        0
fType        1        param        0                                !functional form type 1 (used for immobile defects)

0        1        0        0                        !V_n
nmin        0        nmax        0
nmin        5        nmax -1
nmin        0        nmax        0
nmin        0        nmax        0
fType        1        param        0                                !functional form type 1 (used for immobile defects)

0        0        1        0                        !(SIA_m) n
nmin        0        nmax        0
nmin        0        nmax        0
nmin        5        nmax        -1
nmin        0        nmax        0
fType        3        param        6                                !functional form type 3 (hard coded in)
3.5d10        1.7d11        1.7d0        0.06d0        0.11d0        1.6d0        !input parameters into functional form type 2

0        0        0        1
nmin        0        nmax        0
nmin        0        nmax        0
nmin        0        nmax        0
nmin        1        nmax -1
fType        1        param        0                                !functional form type 1 (used for immobile defects)

1        1        0        0
nmin        1        nmax -1                !He number that can be considered with this function
nmin        1        nmax -1                !V number that can be considered with this function
nmin        0        nmax        0
nmin        0        nmax        0
fType        1        param        0        !Functional form type 1 (used for immobile defects)
                                                !NOTE: for any defect size that falls into functional form's range that is ALSO
                                                !listed in individual defects, individual defect -lormation is used.

bindingEnergies                        !indicates we are moving into binding energies

numSingle
39
numFunction
10

single
2        0        0        0                1        0        0        0        !indicates dissociation of 2He cluster
Eb        0.43d0
3        0        0        0                1        0        0        0        !dissociation of 3He cluster
Eb        0.95d0
4        0        0        0                1        0        0        0        !dissociation of 4He cluster
Eb        0.98d0

0        2        0        0                0        1        0        0        !dissociation of 2V cluster
Eb        0.3d0
0        3        0        0                0        1        0        0        !dissociation of 3V cluster
Eb        0.37d0
0        4        0        0                0        1        0        0        !dissociation of 4V cluster
Eb        0.62d0

0        0        2        0                0        0        1        0        !dissociation of 2SIA cluster
Eb        0.8d0
0        0        3        0                0        0        1        0        !dissociation of 3SIA cluster
Eb        0.92d0

1        1        0        0                1        0        0        0        !HeV-> He + V
Eb        2.3d0
2        1        0        0                1        0        0        0        !2HeV-> He+HeV
Eb        1.84d0
3        1        0        0                1        0        0        0        !3HeV-> He+2HeV
Eb        1.83d0
4        1        0        0                1        0        0        0        !4HeV-> He+3HeV
Eb        1.91d0
1        2        0        0                1        0        0        0        !He2V->He+2V
Eb        2.85d0
2        2        0        0                1        0        0        0        !2He2V->He+He2V
Eb        2.76d0
3        2        0        0                1        0        0        0        !3He2V->He+2He2V
Eb        2.07d0
4        2        0        0                1        0        0        0        !4He2V->He+3He2V
Eb        2.36d0
1        3        0        0                1        0        0        0        !He3V->He+3V
Eb        3.3d0
2        3        0        0                1        0        0        0        !2He3V->He+He3V
Eb        2.96d0
3        3        0        0                1        0        0        0        !3He3V->He+2He3V
Eb        2.91d0
4        3        0        0                1        0        0        0        !4He3V->He+3He3V
Eb        2.57d0
1        4        0        0                1        0        0        0        !He4V->He+4V
Eb        3.84d0
2        4        0        0                1        0        0        0        !2He4V->He+He4V
Eb        3.12d0
3        4        0        0                1        0        0        0        !3He4V->He+2He4V
Eb        3.16d0
4        4        0        0                1        0        0        0        !4He4V->He+2He4V
Eb        3.05d0

1        2        0        0                0        1        0        0        !He2V->V+HeV
Eb        0.78d0
1        3        0        0                0        1        0        0        !He3V->V+He2V
Eb        0.83d0
1        4        0        0                0        1        0        0        !He4V->V+He3V
Eb        1.16d0
```

```
2        1        0        0                0        1        0        0        !2HeV->V+2He
Eb        3.71d0
2        2        0        0                0        1        0        0        !2He2V->V+2HeV
Eb        1.61d0
2        3        0        0                0        1        0        0        !2He3V->V+2He2V
Eb        1.04d0
2        4        0        0                0        1        0        0        !2He4V->V+2He3V
Eb        1.32d0
3        1        0        0                0        1        0        0        !3HeV->V+3He
Eb        4.59d0
3        2        0        0                0        1        0        0        !3He2V->V+3HeV
Eb        1.85d0
3        3        0        0                0        1        0        0        !3He3V->V+3He2V
Eb        1.8d0
3        4        0        0                0        1        0        0        !3He4V->V+3He3V
Eb        1.57d0
4        1        0        0                0        1        0        0        !4HeV->V+4He
Eb        5.52d0
4        2        0        0                0        1        0        0        !4He2V->V+4HeV
Eb        2.3d0
4        3        0        0                0        1        0        0        !4He3V->V+4He2V
Eb        2.03d0
4        4        0        0                0        1        0        0        !4He4V->V+4He3V
Eb        1.97d0

function
1        0        0        0                1        0        0        0        !nHe->He+(n-1)He
nmin         5        nmax        -1
nmin         0        nmax        0
nmin         0        nmax        0
nmin         0        nmax        0
ftype         2        param        1                      !functional form type 2 (used for constant functions)
0.98d0

0        1        0        0                0        1        0        0        !nV->V+(n-1)V
nmin         0        nmax        0
nmin         5        nmax        -1
nmin         0        nmax        0
nmin         0        nmax        0
ftype         4        param        2                      !functional form type 4 (hard-coded in)
2.07d0        0.3d0                                       !parameters for functional form type 4 (only 2)

0        0        1        0                0        0        1        0        !nSIA->SIA+(n-1)SIA
nmin         0        nmax        0
nmin         0        nmax        0
nmin         4        nmax        -1
nmin         0        nmax        0
ftype         4        param        2                      !functional form type 5        (hard-coded in)
3.77d0        0.8d0                                       !parameters for functional form type 5 (only 2)

0        0        0        1                0        0        1        0        !nSIA->SIA+(n-1)SIA NOTE: mobile SIA ejected from Sessile SIA cluster
nmin         0        nmax        0
nmin         0        nmax        0
nmin         0        nmax        0
nmin         4        nmax        -1
ftype         4        param        2                      !functional form type 5        (hard-coded in)
3.77d0        0.8d0                                       !parameters for functional form type 5 (only 2)

1        1        0        0                1        0        0        0        !nHemV->He+(n-1)HemV
nmin         5        nmax -1
mmin         5        mmax -1
nmin         0        nmax        0
nmin         0        nmax        0
ftype         6        param        3                      !functional form type 6 (hard-coded in)
2.22d0        1.6d0        0.64d0

1        1        0        0                1        0        0        0        !nHemV->He+(n-1)HemV
nmin         1        nmax 4
mmin         5        mmax -1
nmin         0        nmax        0
nmin         0        nmax        0
ftype         6        param        3                      !functional form type 6 (hard-coded in)
2.22d0        1.6d0        0.64d0

1        1        0        0                1        0        0        0        !nHemV->He+(n-1)HemV
nmin         5        nmax -1
mmin         1        mmax 4
nmin         0        nmax        0
nmin         0        nmax        0
ftype         6        param        3                      !functional form type 6 (hard-coded in)
2.22d0        1.6d0        0.64d0

1        1        0        0                0        1        0        0        !nHemV->V+nHe(m-1)V
nmin         5        nmax -1
nmin         5        nmax -1
nmin         0        nmax        0
nmin         0        nmax        0
ftype         7        param        5                      !functional form type 7 (hard-coded in)
2.07d0        0.3d0        1.55d0        3.19d0        3.0d0

1        1        0        0                0        1        0        0        !nHemV->V+nHe(m-1)V
nmin         1        nmax 4
nmin         5        nmax -1
nmin         0        nmax        0
nmin         0        nmax        0
ftype         7        param        5                      !functional form type 7 (hard-coded in)
2.07d0        0.3d0        1.55d0        3.19d0        3.0d0

1        1        0        0                0        1        0        0        !nHemV->V+nHe(m-1)V
nmin         5        nmax -1
nmin         1        nmax 4
nmin         0        nmax        0
nmin         0        nmax        0
ftype         7        param        5                      !functional form type 7 (hard-coded in)
2.07d0        0.3d0        1.55d0        3.19d0        3.0d0

!****************************************************************************************************
!List of reactions by reaction type (dissociation, diffusion, clustering, implantation, trapping at sinks)
!****************************************************************************************************
singleDefect                        !the following will list single-defect reactions and their rates

dissociation
6                                              !Number of defect dissociation types

1        0        0        0                1        0        0        0        !He cluster dissociation
nmin         2        nmax        -1
```

```
nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
1                                                    !reactionRate form 1 (used for all dissociation rates in Fe, hard coded)

0        1        0        0                0        1        0        0        !nV->V+(n-1)V
nmin         0         nmax         0
nmin         2         nmax         -1
nmin         0         nmax         0
nmin         0         nmax         0
1                                                    !reactionRate form 1

0        0        1        0                0        0        1        0        !nSIA->SIA+(n-1)SIA
nmin         0         nmax         0
nmin         0         nmax         0
nmin         2         nmax         -1
nmin         0         nmax         0
1                                                    !reactionRate form 1

0        0        0        1                0        0        1        0        !nSIA->SIA+(n-1)SIA NOTE: mobile SIA ejected from Sessile SIA cluster
nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
nmin         2         nmax         -1
1                                                    !reactionRate form 1

1        1        0        0                1        0        0        0        !nHemV->He+(n-1)HemV
nmin         1         nmax -1
mmin         1         mmax -1
nmin         0         nmax         0
nmin         0         nmax         0
1                                                    !reactionRate form 1

1        1        0        0                0        1        0        0        !nHemV->V+nHe(m-1)V
nmin         1         nmax -1
nmin         1         nmax -1
nmin         0         nmax         0
nmin         0         nmax         0
1                                                    !reactionRate form 1

diffusion
4                                                    !number of defect types that can diffuse

1        0        0        0                1        0        0        0                                !He diffusion
nmin         1         nmax         6
nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
2                                                    !reactionRate form 2 (used for all diffusion events, hard coded)

0        1        0        0                0        1        0        0                                !V diffusion
nmin         0         nmax         0
nmin         1         nmax         4
nmin         0         nmax         0
nmin         0         nmax         0
2                                                    !reactionRate form 2

0        0        1        0                0        0        1        0                                !SIA_mobile diffusion
nmin         0         nmax         0
nmin         0         nmax         0
nmin         1          nmax         -1
nmin         0         nmax         0
2                                                    !reactionRate form 2

1        1        0        0                1        1        0        0                                !HeV diffusion
nmin         1         nmax         4
mmin         1         mmax         3
nmin         0         nmax         0
nmin         0         nmax         0
2                                                    !reactionRate form 2

sinkRemoval                                        !These reactions involve removing defects from the mesh by dislocations
4                                                    !number of defect types that can be removed this way

1        0        0        0                           !He trapping
nmin         1         nmax         6
nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
3                                                    !reactionRate form 3 (used for all dislocation removal events, hard coded)

0        1        0        0                           !V trapping
nmin         0         nmax         0
nmin         1         nmax         4
nmin         0         nmax         0
nmin         0         nmax         0
3                                                    !reactionRate form 3

0        0        1        0                           !SIA_mobile trapping
nmin         0         nmax         0
nmin         0         nmax         0
nmin         1          nmax         -1
nmin         0         nmax         0
3                                                    !reactionRate form 3

1        1        0        0                           !HeV trapping
nmin         1         nmax         4
mmin         1         mmax         3
nmin         0         nmax         0
nmin         0         nmax         0
3                                                    !reactionRate form 3

impurityTrapping                         !These reactions involve changing mobile SIA loops to sessile SIA loops via trapping by impurities
1                                                    !Only one defect type can be affected by this

0        0        1        0                0        0        0        1
nmin         0         nmax         0
nmin         0         nmax         0
nmin         5         nmax         -1
nmin         0         nmax         0
4                                                    !reactionRate form 4 (essentially clustering between SIA loop and point defect)

multipleDefect                               !the following is a list of multiple-defect reactions and their rates (clustering)

clustering
17                                                    !number of different reactions allowed
```

```
1       0       0       0               1       0       0       0               !He-He
nmin        1       nmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
nmin        0       nmax        0

mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
nmin        0       nmax        0
5                                                   !reactionRate form 5 (He-He clustering reaction rate includes limit on He cluster size)

1       0       0       0               0       1       0       0               !He-V
nmin        1       nmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
6                                                   !reactionRate form 6 - for spherical clusters interacting

0       1       0       0               0       1       0       0               !V-V
nmin        0       nmax        0
nmin        1       nmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
6                                                   !reactionRate form 6 - for spherical clusters

1       1       0       0               1       0       0       0               !HeV-He
nmin        1       nmax        -1
mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

lmin        1       lmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
nmin        0       nmax        0
7                                                   !reactionRate form 7 - for HeV clusters - use V number for volume of defect

1       1       0       0               0       1       0       0               !HeV-V
nmin        1       nmax        -1
mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
lmin        1       lmax        -1
nmin        0       nmax        0
nmin        0       nmax        0
7                                                   !reactionRate form 7 - for HeV Clusters - use V number for volume of defect

1       1       0       0               1       1       0       0               !HeV-HeV
nmin        1       nmax        -1
mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

lmin        1       lmax        -1
omin        1       omax        -1
nmin        0       nmax        0
nmin        0       nmax        0
7                                                   !reactionRate form 7 - for HeV Clusters - use V number for volume of defect

0       1       0       0               0       0       1       0               !V-SIA_mobile (annihilation)
nmin        0       nmax        0
nmin        1       nmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
nmin        0       nmax        0
mmin        1       mmax        4
nmin        0       nmax        0
6                                                   !reactionRate form 6 - for spherical clusters

0       1       0       0               0       0       1       0               !V-SIA_mobile (1D loop) (annihilation)
nmin        0       nmax        0
nmin        1       nmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
nmin        0       nmax        0
mmin        5       mmax        -1
nmin        0       nmax        0
8                                                   !reactionRate form 8 - for 1D circular - 3D spherical interactions

0       1       0       0               0       0       0       1               !V-SIA_sessile (annihilation)
nmin        0       nmax        0
nmin        1       nmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
nmin        0       nmax        0
nmin        0       nmax        0
mmin        5       mmax        -1
8                                                   !reactionRate form 8 - for 1D circular - 3D spherical interactions

1       1       0       0               0       0       1       0               !HeV-SIA_mobile (disallow reactions that leave He-SIA clusters)
nmin        1       nmax        -1
mmin        1       mmax        -1
nmin        0       nmax        0
nmin        0       nmax        0

nmin        0       nmax        0
```

```
nmin         0         nmax         0
lmin         1         lmax         4
nmin         0         nmax         0
7                                                    !reactionRate form 7 - for HeV clusters - use V number for volume of defects

1        1         0         0                0         0         1         0                !HeV-SIA_mobile (disallow reactions that leave He-SIA clusters)
nmin         1         nmax        -1
mmin         1         mmax        -1
nmin         0         nmax         0
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
lmin         5         lmax        -1
nmin         0         nmax         0
8                                                    !reactionRate form 8 - for 1D circular - 3D spherical interactions

1        1         0         0                0         0         0         1                !HeV-SIA_sessile (dissalow reactions that leave He-SIA clusters)
nmin         1         nmax        -1
mmin         1         mmax        -1
nmin         0         nmax         0
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
lmin         5         lmax        -1
8                                                    !reactionRate form 8 - for 1D circular - 3D spherical interactions

0        0         1         0                0         0         1         0                !small SIA_mobile clusters
nmin         0         nmax         0
nmin         0         nmax         0
nmin         1         nmax         4
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
mmin         1         mmax         4
nmin         0         nmax         0
6                                                    !reactionRate form 6 - for spherical clusters

0        0         1         0                0         0         1         0                !1D-3D SIA_mobile interactions
nmin         0         nmax         0
nmin         0         nmax         0
nmin         1         nmax         4
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
mmin         5         mmax        -1
nmin         0         nmax         0
8                                                    !reactionRate form 8 - for 1D circular - 3D spherical interactions

0        0         1         0                0         0         1         0                !1D-1D SIA_mobile interactions (leavese SIA_sessile)
nmin         0         nmax         0
nmin         0         nmax         0
nmin         5         nmax        -1
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
mmin         5         mmax        -1
nmin         0         nmax         0
9                                                    !reactionRate form 9 - for 1D-1D SIA loop interactions

0        0         1         0                0         0         0         1                !small SIA_mobile+SIA_sessile (leaves SIA_sessile)
nmin         0         nmax         0
nmin         0         nmax         0
nmin         1         nmax         4
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
mmin         5         mmax        -1
8                                                    !reactionRate form 8 - for 1D circular - 3D spherical interactions

0        0         1         0                0         0         0         1                !SIA_mobile loop + SIA_sessile (leaves SIA_sessile)
nmin         0         nmax         0
nmin         0         nmax         0
nmin         5         nmax        -1
nmin         0         nmax         0

nmin         0         nmax         0
nmin         0         nmax         0
nmin         0         nmax         0
mmin         5         mmax        -1
9                                                    !reactionRate form 9 - for 1D-1D SIA loop interactions

noDefect                                         !Reactions with no reactants (defect generation reactions)
3                                                    !Number of reactions with no reactants

FrenkelPair                                        !Frenkel pair implantation

0        1         0         0                0         0         1         0                !V+SIA creation
10                                                    !reactionRate form 10 - for Frenkel pair implantation

HeImplant                                        !Helium implantation

1        0         0         0                          !He creation
12                                                    !reactionRate form 12 - for He implantation

Cascade                                             !Cascade implantation

11                                                    !reactionRate form 11 - for cascade implantation
```

A detailed description of the inputs to this file is given below:

1. **material** This gives the material ID number that this input file refers to
2. **species** This gives the number of defect species allowed in this material type. This number is typically set to 4 (helium, vacancies, mobile SIAs, and immobile SIAs) but the user could change this value if desired. Note that the entire development of the code has centered around the use of 4 defect species, so addition or subtraction from this list is likely to cause bugs and should be done with extreme caution.
3. **List of diffusion parameters**
   (a) **numSingle, numFunction**: the number of defect types for which we have individual values of diffusion prefactors and migration energies, and the number of functional forms for diffusion and migration energies. These numbers are used to define the size of the variables that this data is read into. This strategy allows specific values to be set for the most critical defect types (small defects and clusters) while using standardized functional forms for the defects that are less important (very large clusters of defects).
   (b) **single defect data**: this section gives the diffusion prefactors (in nm^2s^{-1}) and migration energies (in eV) of several listed defects. First, the defect type is given using four integers, corresponding to the number of He, V, SIA, and immobile SIA defects in that cluster. For example, 1 0 0 0 is a single (interstitial) helium, 2 3 0 0 is a He_2V_3 cluster, and 0 0 5 0 is a 5-SIA mobile dislocation loop. The next line contains D_0, then the diffusion prefactor, then E_m, then the migration energy. This is repeated for each individual defect for which migration energies are known.
   (c) **functional forms**: this section allows the user to use a functional form for the diffusivity of various groups of defects. The format of an entry in this section is the following:
      i. The defect type is given using the four numeral system described above, except that only 1 and 0 are used to describe the defect. For example, 1 0 0 0 indicates interstitial helium clusters of all sizes, rather than just a single interstitial helium.
      ii. The minimum and maximum number of defects of each species in that defect type are listed in the next four lines, with each line referring to a different defect type. -1 is used to indicate a maximum of infinity for that defect type. Therefore, nmin$= 7$ and nmax$= -1$ for the first functional form's first defect type, indicating that this functional form applies to interstitial clusters size $[7, \infty)$. The other defect types have nmin$= 0$ and nmax$= 0$, indicating that this defect type is for helium only.
      iii. The next information is the function ID number, 'ftype', and the number of input parameters to read in. The ftype value tells the SRSCD program which functional form to use in the subroutine DiffusivityCompute(), and the number of input parameters should match the number of input parameters used in that subroutine for that function.
      iv. The next line contains the various input parameters for that functional form, which are used in the subroutine DiffusivityCompute() to calculate the diffusivity of that defect type.

   An example of the functional form formatting and how it is applied is given below:

0	0	1	0		
nmin	0	nmax	0		
nmin	0	nmax	0		
nmin	5	nmax	-1		
nmin	0	nmax	0		
ftype	3	param	6		
3.5×10^{10}	1.7×10^{11}	1.7	0.06	0.11	1.6

   - The first line indicates that this functional form is applies to mobile SIA clusters (0 0 1 0)
   - The next four lines indicate that the functional form applies to mobile SIA clusters with sizes $\in [5, \infty)$
   - The next line indicates that the functional form used for the mobility of these defects has ID number 3, and that 6 inputs are needed for this functional form. Inside DiffusivityCompute(), functional form 3 can be seen to give the diffusion prefactor D_0 and migration energy E_m as the following:

$$D_0 = p_1 + \frac{p_2}{n^{p_3}} \ , \ E_m = p_4 + \frac{p_5}{n^{p_6}}$$

   where p_i is input parameter i and n is the size of the defect cluster. This is taken from the work of Soneda and Diaz de La Rubia (2001).

- The next line gives the six input parameters needed in this functional form to compute defect diffusivities.

4. **List of binding parameters**

   (a) **numSingle, numFunction**: Similar to the list of diffusion parameters, the number of single defects and functional forms for binding energies in the input file.

   (b) **single defect data**: This consists of two defect types necessary for dissociation reactions: first, the defect cluster type that exists before the dissociation reaction; second, the point defect that is dissociating away from that cluster. The SRSCD program automatically calculates what the remaining defect type is. For example, 4 2 0 0 1 0 0 0 indicates that a He_4V_2 cluster is emitting a He_1 interstitial helium atom, leaving behind a He_3V_2 cluster. The next line contains the binding energy required for this dissociation reaction

   (c) **functional forms**: This is identical to functional forms used for diffusion parameters, except that the defect type of the cluster and the emitted defect are given in the first line (so 1 0 0 0 1 0 0 0 in this case indicates that a single helium interstitial is emitting from a larger helium cluster) and the subroutine in which the functional forms are located is BindingCompute().

5. **Lists of allowed reactions**: several different reaction types are given in the following section, along with the ID number of the functional form for their reaction rates. The allowed reaction types implemented in SRSCD are dissocation, diffusion, removal by sinks, trapping by impurities, clustering (including annihilation), and implantation (Frenkel pair or cascade). An example of an entry in this section for clustering is given below:

```
1 1 0 0          0 0 1 0
nmin     1       nmax    -1
nmin     1       nmax    -1
nmin     0       nmax    0
nmin     0       nmax    0

nmin     0       nmax    0
nmin     0       nmax    0
nmin     5       nmax    -1
nmin     0       nmax    0
8
```

   (a) The first line indicates that this reaction represents a HeV cluster interacting with a mobile SIA cluster (1 1 0 0 represents HeV clusters, and 0 0 1 0 represents mobile SIA clusters).

   (b) The next four lines give the first defect in the reaction: an HeV cluster with minimum size He_1V_1 and maximum size $He_\infty V_\infty$.

   (c) The next four lines give the second defect in the reaction: a mobile SIA cluster with size $\in [5, \infty)$.

   (d) The next line contains the ID number of the functional form used to compute the reaction rate between these two defect types (functional form 8 in this case). These reaction rates are contained in the following subroutines (depending on the type of reaction):

      - findReactionRate
      - findReactionRateDiff
      - findReactionRateDissoc
      - findReactionRateImpurity
      - findRecationRateMultiple
      - findReactionRateSink

      as well as the associated subroutines for reaction rates inside the fine mesh.

   Note that the input for Frenkel pair or cascade implantation is much simpler than described above, as no inputs are needed for this reaction rate.

3.2.4 Cascade input file (for cascade damage)

The standard cascade input file (for 20 keV cascades in α-Fe) is partially given below. It contains a list of the defects in several cascades generated by atomistics as well as their location relative to the center of the cascade.

Fe_Cascades.txt

```
61.6667d0        !Average number of displaced atoms per cascade (used for dpa calculation)

9                !number of cascades

119               !number of defects
65               !number of displaced atoms
0         0      3         0         0
0.6553421798       2.9838466387        -0.2058733894        I3
0         0      2         0         0
-2.1295878202       -1.977720028        -0.2786733894        I2
0         0      2         0         0
-2.0059628202       2.345679972         2.9040266106        I2
0         0      2         0         0
-0.8665028202       2.634129972        -0.5640233894        I2
0         0      2         0         0
1.5937751798       2.463879972        -1.4486733894        I2
0         0      2         0         0
4.3434271798       -3.649420028        -3.9206733894        I2
0         0      1         0         0
-4.8871478202       4.001479972         4.9809266106        I1
0         0      1         0         0
-4.5517178202       -1.898420028         4.3200266106        I1
0         0      1         0         0
-3.3858378202       0.282779972         0.9505266106        I1
0         0      1         0         0
-3.3849778202       -0.030220028         1.2116266106        I1
0         0      1         0         0
-3.2453278202       3.607679972         2.5450266106        I1
0         0      1         0         0
-3.3861178202       4.418779972         4.5558266106        I1
0         0      1         0         0
-3.7001378202       4.154879972         4.5407266106        I1
0         0      1         0         0
-3.3687278202       6.457479972         6.5613266106        I1
0         0      1         0         0
-3.3701278202       6.759479972         6.2804266106        I1
0         0      1         0         0
-3.1436878202       -5.507520028        -5.5763733894        I1
0         0      1         0         0
-2.0277978202       -0.457320028        -3.5247733894        I1
0         0      1         0         0
-2.6899578202       -0.746620028         5.5872266106        I1
0         0      1         0         0
-2.3771178202       2.548179972        -2.1375733894        I1
0         0      1         0         0
-1.9590178202       3.432079972         0.6609266106        I1
0         0      1         0         0
-2.9499878202       4.086179972         1.8080266106        I1
0         0      1         0         0
-2.2243078202       6.449979972         6.6128266106        I1
etc...
```

The first line in this input file gives the average number of displaced atoms per cascade. The second line gives the number of cascades stored in this file. Next, each cascade is listed, in the following format:

1. The first number for each cascade represents the number of defects that must be read in from the file
2. The second number represents the number of displaced atoms caused by this cascade (used for calculating dpa)
3. The defects are given next, with 2 lines per defect. The first line (0 0 3 0 0) represents the defect type - note that here, 5 numbers are used to indicate defect type. This is legacy from previous iterations of SRSCD, and the fifth number can be ignored. The second line indicates the coordinates of that defect relative to the center of the cascade. The 'I3' text at the end of the second line indicates that this is a 3-SIA cluster, but this information is not read in by the SRSCD code and is for the user only.

3.2.5 Strain input file (for stress-assisted diffusion, implemented but not tested)

An example strain input file is given below. It contains the location of the center of each volume element as well as the six strain components inside that volume element.

StrainFile.txt

```
!*****************************************************************************
!This file contains the strain field inside each volume element (defined at the center of the volume element)
!
```

```
!It is defined as e11, e22, e33, e12, e23, e13
!*****************************************************************************

!Data
!Order: volume element center position x, y, z, then e11, e22, e33, e12, e23, e13

start

2.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
7.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
12.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
17.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
22.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
27.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
32.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
37.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
42.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
47.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
52.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
57.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
62.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
67.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
72.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
77.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
82.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
87.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
92.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
97.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
102.50000        2.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
2.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
7.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
12.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
17.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
22.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
27.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
32.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
37.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
42.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
47.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
52.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
57.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
62.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
67.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
72.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
77.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
82.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
87.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
92.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
97.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000
102.50000        7.50000        2.50000        0.01000        -0.01000        0.00000        0.00000        0.00000        0.00000

etc...
```

This data file begins with the word 'start' and a skipped line. After that, each line contains the coordinates of the center of the volume element that the line represents, followed by the six components of strain present in that volume element (ϵ_{11}, etc...). There must be an identical number of lines containing strain information as there are volume elements.

3.2.6 Dose rate input file (for nonuniform implantation)

An example of a non-uniform dose rate profile for ion implantation is given below. It contains a list of z-coordinates and the corresponding dose rate and helium implantation rate at each depth.

ImplantProfile.txt

```
!This        file        contains        a        nonuniform        He        and        DPA        implantation        profile.
!It        does        not        need        to        match        with
!the        mesh        elements        of        mesh        file,        but        it        should
!have        coordinates        (z-coordinates        only)        that        are
!greater        than        or        equal        to        in        span        those        of
!the        mesh        file.

!Tell        the        computer        how        many        values        of        DPA
!rate        and        He        implant        rate        to        read        in
numImplantDataPoints
100

!Tell        the        computer        to        start        reading in values
!z_coord(nm)        dpa_rate        he(at.%/s)
!Values generated from SRIM for He ion implantation in Fe thin films
start
1.00E+00        2.83E-02        3.88E-05
2.00E+00        4.52E-02        9.47E-05
3.00E+00        5.45E-02        1.24E-04
4.00E+00        5.57E-02        1.44E-04
5.00E+00        5.56E-02        1.73E-04
6.00E+00        5.39E-02        1.93E-04
7.00E+00        5.28E-02        2.18E-04
8.00E+00        5.15E-02        2.37E-04
9.00E+00        4.95E-02        2.40E-04
1.00E+01        4.73E-02        2.58E-04
1.10E+01        4.57E-02        2.66E-04
1.20E+01        4.44E-02        2.73E-04
1.30E+01        4.29E-02        2.81E-04
1.40E+01        4.13E-02        2.90E-04
1.50E+01        3.98E-02        3.11E-04
```

```
1.60E+01         3.83E-02         3.15E-04
1.70E+01         3.67E-02         3.32E-04
1.80E+01         3.58E-02         3.18E-04
1.90E+01         3.48E-02         3.22E-04
2.00E+01         3.37E-02         3.43E-04
etc...
```

This data file begins with the word 'start'. The next line contains three numbers: the depth (or z-coordinate, in nm), the dpa rate (in dpa·s^{-1}), and the helium implantation rate (in atomic fraction per second). Unlike previous input files, the dose rate input file can exhibit a mismatch with the mesh. In this case, a linear interpolation or an average over several points is used to determine the dose rate inside each volume element (which must be uniform in each volume element).

3.2.7 Debug restart input file (for simulations restarting from nonzero defects)

An example of an input file used to restart a simulation from a previously incomplete simulation (or from a situation with nonzero damage and nonzero defects initially present) is given below:

debugRestart.in

```
numProcs
4

numImplantEvents
75979

numHeImplantEvents
0

elapsedTime
45715.9027420948

start

 processor
 0

 coordinates
 30.0000000000000        30.0000000000000       30.0000000000000

 numDefectTypes
 174
          0           0           0           5 cell           1 num
          5
          0           0           0           6 cell           1 num
          6
          0           0           0           7 cell           1 num
          2
          0           0           0           8 cell           1 num
          1
          0           0           0           9 cell           1 num
          3
          0           0           0          10 cell           1 num
          2
          0           0           0          11 cell           1 num
          1
          0           0           0          12 cell           1 num
          2
          0           0           0          13 cell           1 num
          1
 end

 coordinates
 90.0000000000000        30.0000000000000     30.0000000000000

 numDefectTypes
 183
          0           0           0           5 cell           2 num
          6
          0           0           0           6 cell           2 num
          5
          0           0           0           7 cell           2 num
          4
          0           0           0           8 cell           2 num
          4
          0           0           0           9 cell           2 num
          4
          0           0           0          10 cell           2 num
          1
          0           0           0          11 cell           2 num
          1
          0           0           0          13 cell           2 num
          3
          0           0           0          15 cell           2 num
          5
          0           0           0          16 cell           2 num
 end

 processor
 1

 coordinates
 30.0000000000000        90.0000000000000       30.0000000000000

 numDefectTypes
 188
          0           0           0           6 cell           1 num
```

```
           6
           0           0           0           7 cell           1 num
           6
           0           0           0           8 cell           1 num
           8
           0           0           0           9 cell           1 num
           6
           0           0           0          10 cell           1 num
           2
           0           0           0          11 cell           1 num
           4
           0           0           0          12 cell           1 num
           1
           0           0           0          14 cell           1 num
           1
           0           0           0          15 cell           1 num
etc...
```

This file contains several components, which must be formatted correctly:

1. **numProcs**: the number of processors in the simulation. It must match the actual SRSCD simulation being run.
2. **numImplantEvents, numHeImplantEvents**: the number of implantation events (Frenkel pair or cascade) and helium implantation events that have already been carried out prior to this debug file being created.
3. **elapsedTime**: the total elapsed time in seconds prior to the creation of this debug file.
4. **start**: indicates that defect input is about to start
5. **processor**: indicates that defect input is about to start for processor $i \in [0, n-1]$ where n is the number of processors in the simulation.
6. **coordinates**: indicates the coordinates of the center of the volume element for which defects are about to be read in. Must match the coordinates of the mesh for this SRSCD simulation.
7. **numDefectTypes**: indicates the number of defect types present in this volume element.
8. **defect information**: gives the defect type (0 0 0 5 indicates an immobile 5-SIA cluster, for example), the volume element number, and the number of defects of this type present inside this volume element.
9. **end**: indicates that this volume element is finished.

3.3 Standard output files

In this section, several examples of the various output files provided by subroutines in SRSCD are given. This is not an exhaustive list, and does not include output files that are read in by programs such as ParaView, but the most commonly used output files are shown here.

3.3.1 Defect raw data

The standard raw data output file is given below. It gives the number and type of each defect present in each volume element. This data file can be very large, depending on the size of the domain.

rawdat.out

```
 dpa  5.196663000000000E-005
 processor           0
 coordinates   10.0000000000000        10.0000000000000
   10.0000000000000
           0           1           0           0 cell           1 num
           6
           0           5           0           0 cell           1 num
           1
           0           6           0           0 cell           1 num
           1
 coordinates   30.0000000000000        10.0000000000000
   10.0000000000000
           0           1           0           0 cell           2 num
          10
           0           5           0           0 cell           2 num
           1
 coordinates   50.0000000000000        10.0000000000000
```

```
   10.0000000000000
           0           1           0           0 cell           3 num
           5
           1           7           0           0 cell           3 num
           1
 coordinates   70.0000000000000        10.0000000000000
   10.0000000000000
           0           1           0           0 cell           4 num
           9
           0           2           0           0 cell           4 num
           1
 coordinates   90.0000000000000        10.0000000000000
   10.0000000000000
           0           1           0           0 cell           5 num
          12
 coordinates   10.0000000000000        30.0000000000000
   10.0000000000000
           0           1           0           0 cell           6 num
           6
           0           6           0           0 cell           6 num
           1
           1           3           0           0 cell           6 num
           1
etc...

 Released all fine mesh defects
```

This file contains the output generated at every intermediate output step as well as the final output in SRSCD. The first piece of data in the output is the dpa at which this output was written. The processor number, volume element coordinates, and defects inside the volume element are then included similarly to the format of Section 3.2.7. The main difference is that the raw data output file contains the defect state at several time snapshots, rather than just a single time snapshot.

3.3.2 Total defects

The standard total data output file is given below. It gives the total defect numbers in the entire simulation during each output step.

totdat.out

```
 dpa  5.196663000000000E-005
 Defects num
           0           0           0           0           0
           0           0           1           0           3
           0           1           0           0        1500
           0           2           0           0          54
           0           4           0           0           8
           0           5           0           0          57
           0           6           0           0          12
           0           7           0           0           3
           0           8           0           0           2
           0           9           0           0           2
           1           0           0           7           1
           1           0           0          10           1
           1           1           0           0          47
           1           3           0           0          15
           1           4           0           0           6
           1           5           0           0           1
           1           6           0           0           2
           1           7           0           0           1
           1           9           0           0           1
           1          10           0           0           1
           1          11           0           0           1
           1          12           0           0           1
           2           0           0           2           1
           2           2           0           0           1
           2           3           0           0           2
           2           4           0           0           1
           2           5           0           0           1
           2           6           0           0           1
 HeNum          85 Conc  2.125000000000000E+022
 VNum         1720 Conc  4.300000000000000E+023
 Voids           0 Conc  0.000000000000000E+000
 SIANum          6 Conc  1.500000000000000E+021
 Loops           0 Conc  0.000000000000000E+000

 dpa  1.032772500000000E-004
 Defects num
           0           0           0           0           0
           0           0           1           0           5
           0           1           0           0        1652
           0           2           0           0          67
           0           4           0           0           4
           0           5           0           0          71
           0           6           0           0          18
           0           7           0           0          10
           0           8           0           0           8
           0           9           0           0           4
           0          10           0           0           1
           0          11           0           0           2
           0          12           0           0           1
           0          13           0           0           1
           0          14           0           0           2
           0          23           0           0           1
           1           0           0           9           1
           1           0           0          19           1
           1           1           0           0          42
```

```
           1           2           0           0           1
           1           3           0           0          22
           1           4           0           0           8
           1           5           0           0           6
           1           6           0           0          10
           1           7           0           0           6
           1           8           0           0           4
           1           9           0           0           3
           1          10           0           0           1
           1          11           0           0           2
           1          13           0           0           1
           2           1           0           0           1
           2           2           0           0           8
           2           3           0           0           3
           2           4           0           0           2
           2           5           0           0           1
           2           6           0           0           2
           2           7           0           0           1
           2           8           0           0           1
           2          11           0           0           1
           3           2           0           0           2
 HeNum         130 Conc  3.250000000000000E+022
 VNum         1970 Conc  4.925000000000000E+023
 Voids           0 Conc  0.000000000000000E+000
 SIANum          7 Conc  1.750000000000000E+021
 Loops           0 Conc  0.000000000000000E+000

etc...
```

This file is also written to during each intermediate output step as well as the final output step. For each output step, the file contains the total dose (in dpa), a list of all of the defects inside the simulation (summed over all volume elements), and then a summary of the total number of defects of each type - helium, vacancy, and SIA. The number of voids (vacancy clusters with > 40 defects) and visible loops (SIA clusters with > 20 defects) is also output.

The format of the defect output is as follows: The first four numbers indicate the defect type (He, V, SIA, immobile SIA) and the last number indicates the number of defects of that type present in the simulation. Thus 1 7 0 0 6 indicates that there are six He_1V_7 clusters present in the simulation during that time snapshot.

3.3.3 Postprocessing

The standard postprocessing output file is given below. It provides information at each output step including the defect concentration of vacancies, SIAs, and defects with helium as well as other information that can be easily changed by the user.

postpr.out

```
time  3.086673510564005E-003 dpa  5.196663000000000E-005 steps       35448
Cascades/Frenkel pairs       17586 He implant events         201
computation time   6.117070
Fraction null steps  0.439573459715640

HeNum          85 Conc  2.125000000000000E+022
VNum        1720 Conc  4.300000000000000E+023
Voids           0 Conc  0.000000000000000E+000
SIANum          6 Conc  1.500000000000000E+021
Loops           0 Conc  0.000000000000000E+000
AverageHeSize   1.08235294117647
AverageVSize   1.31220930232558
AverageSIASize   3.16666666666667
AverageVoidSize NaN
AverageLoopSize NaN
PercentHeRetained  0.457711442786070
PercentVRetained  0.128340725577164
PercentVAnnihilated  0.832821562606619
NumReactionsCoarse        4134
NumReactionsFine           0
AverageTimestep  8.707609768009493E-008

time  6.172905224812123E-003 dpa  1.032772500000000E-004 steps       67983
Cascades/Frenkel pairs       34950 He implant events         384
computation time   13.36897
Fraction null steps  0.442595943103423

HeNum         130 Conc  3.250000000000000E+022
VNum        1970 Conc  4.925000000000000E+023
Voids           0 Conc  0.000000000000000E+000
SIANum          7 Conc  1.750000000000000E+021
Loops           0 Conc  0.000000000000000E+000
AverageHeSize   1.18461538461538
AverageVSize   1.52385786802030
AverageSIASize   4.00000000000000
AverageVoidSize NaN
AverageLoopSize NaN
PercentHeRetained  0.401041666666667
PercentVRetained  8.589413447782547E-002
PercentVAnnihilated  0.860858369098712
NumReactionsCoarse        4542
NumReactionsFine           0
AverageTimestep  9.080071819149086E-008

time  1.234574037747289E-002 dpa  2.078990250000000E-004 steps      131773
Cascades/Frenkel pairs       70355 He implant events         809
computation time   29.07258
```

```
 Fraction null steps  0.451154637141144

 HeNum         251 Conc  6.275000000000000E+022
 VNum         2279 Conc  5.697499999999999E+023
 Voids           0 Conc  0.000000000000000E+000
 SIANum          2 Conc  4.999999999999999E+020
 Loops           1 Conc  2.500000000000000E+020
 AverageHeSize   1.49800796812749
 AverageVSize   1.81088196577446
 AverageSIASize   16.5000000000000
 AverageVoidSize NaN
 AverageLoopSize   23.0000000000000
 PercentHeRetained  0.464771322620519
 PercentVRetained  5.865965460876981E-002
 PercentVAnnihilated  0.882964963399900
 NumReactionsCoarse        5303
 NumReactionsFine           0
 AverageTimestep  9.368945366253246E-008

etc...

 computation time   2542.232
 total steps     6113073
 Released all fine mesh defects
```

Similarly to the previous two filetypes, output is written to the postpr.out file at each output step. The specific outputs that the user chooses can be changed by making changes inside the subroutine outputDefectsTotal(). The outputs shown in the example above include the following:

- simulation time
- dose (in dpa)
- the number of KMC steps
- the number of cascades/Frenkel pairs and helium implant events
- computation time
- the fraction of null steps (for evaluating the efficacy of the parallel algorithm)
- The number of helium, vacancy, and SIA defects as well as voids and visible SIA loops (> 40 and > 20 defects, respectively)
- The average size of helium, vacancy, and SIA clusters (in units of number of defects)
- The fraction of helium and vacancies originally implanted in the system that is still present
- The fraction of vacancies originally implanted in the system that have been annihilated
- The number of possible reactions in the coarse mesh
- The number of possible reactions in any fine meshes that may be present
- The average timestep of the simulation

Chapter 4

Example simulations

4.1 Cascade implantation in coarse-grained α-Fe

In this simulation, cascade damage is implanted in coarse-grained iron at $T = 343$ K at a dose rate of 7×10^{-7} dpa·s^{-1}, corresponding to the damage conditions measured in Eldrup et al. (2002) for neutron-irradiated iron. Cascade implantation is used with adaptive meshing, and therefore the mesh file contains a small number of large volume elements rather than a large number of small volume elements. For the current example, the mesh file consists of a $2 \times 2 \times 2$ grid of 80 nm volume elements. The mesh file Mesh_Bulk.txt is given below:

Mesh_Bulk.txt

```
!********************************************************************************
!This file contains a UNIFORM cubic mesh input for the parallel SRSCD
!code. The mesh will be read in via coordinates (x,y,z) of each element center
!as well as the material number of each element (eg. material 1=Fe, 2=Cu, etc)
!
!The first values that will be read in are the number of elements in each direction
!(x,y,z).
!
!********************************************************************************

!Specify mesh type (currently two types=periodic and freeSurfaces)
meshType
periodic

!Specify element lengths (nm)
length
  80.000000000000000

!Specify max and min of each coordinate (nm)
!NOTE: the actual max, min will be max, min +/- length/2 because these are assumed
!to be the center of the elements and not the edges
xminmax
  40.000000000000000        120.00000000000000
yminmax
  40.000000000000000        120.00000000000000
zminmax
  40.000000000000000        120.00000000000000

!Specify number of elements in each direction
numx
          2
numy
          2
numz
          2

!Coordinates of element centers (x, y, z, elementType) (nm)
!NOTE: these must be ordered in the same way as the connectivity matrix (loop x, then y, then z)
elements                    !tells computer that the following values are elements
  40.000000000000000        40.000000000000000        40.000000000000000                1
  120.00000000000000        40.000000000000000        40.000000000000000                1

  40.000000000000000        120.00000000000000        40.000000000000000                1
  120.00000000000000        120.00000000000000        40.000000000000000                1

  40.000000000000000        40.000000000000000        120.00000000000000                1
  120.00000000000000        40.000000000000000        120.00000000000000                1

  40.000000000000000        120.00000000000000        120.00000000000000                1
  120.00000000000000        120.00000000000000        120.00000000000000                1
```

The parameters.txt file for this simulation is given below:

parameters.txt

```
!************************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!************************************************************************************************

!Tell computer location of mesh input file
meshFile
Mesh_Bulk.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
no

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
no

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
reset.in

!Tell computer if the number of materials we are reading in
numMaterials
1

!Tell computer the location of the material input file(s)
materialFile
```

```
Fe_Defects.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
Cascade

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
Fe_Cascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
adaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!*****************************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!*****************************************************************************************************

start                        !begin parameters

temperature                !Temperature, in K
343d0

annealTemp
343d0                        !Annealing temperature, in K

dpaRate                        ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
7d-7

HeDPA                        ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                                !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                !(Fe)                        1.17d-2                !(Cu)                ! atomsize (nm^3)

burgers
.287d0                        !(Fe)                        .36d0                        !(Cu)                ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
1d-3                        ! total DPA in simulation 1

annealTime
0d0                                ! total anneal time (s)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
yes

grainSize
33000d0                        ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
1d-5                        ! dislocation density (nm^-2)

impurityConc
30d-6                        ! carbon impurity concentration (atomic fraction)

max3DInt
4                                !maximum size for SIA defect to diffuse in 3D

cascadeVolume
1000d0                        !volume of cascade (nm^3) - used for cascade-defect interactions

numSims
1                                !number of times to repeat simulation

end                                !Tag for the end of the parameters file

!*****************************************************************************************************
!Adaptive meshing parameters
!*****************************************************************************************************

fineStart                ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                                ! Length of one cascade implantation element (nm)

numxFine
6                                ! number of cascade elements in x-direction (fine mesh)

numyFine
6                                ! number of cascade elements in y-direction

numzFine
6                                ! number of cascade elements in z-direction

end                                ! Tag for the end of meshing parameters
```

The only other necessary files for this simulation are the cascade input file Fe_Cascades.txt and Fe_Defects.txt. As these are standard versions of these files, they are not shown here. They can be found in the tests folder of the SRSCD code. Using the output of postpr_1.out, we can investigate the concentration of voids and dislocation loops as a function of dose. These results are shown in Figure 4.1.

4.2 Frenkel pair implantation in α-Fe thin films

In this section, three very similar simulations are carried out and their results are compared in order to both demonstrate the inputs that need to change between the simulation types as well as to demonstrate how the outputs can change when these inputs change. The three simulations carried out are: uniform Frenkel pair implantation in 100 nm α-Fe thin films, uniform Frenkel pair and helium implantation, and nonuniform Frenkel pair implantation and helium implantation. In all cases, the (average) dose rate and helium-to-dpa ratio are identical, as well as the total dose and temperature (except for the first case, in which no helium is implanted).

4.2.1 Frenkel pairs only (no helium)

In this simulation, a 100-nm thin film is simulated using 20 nm length volume elements. The total mesh is consists of a $10 \times 10 \times 5$ grid of 20 nm volume elements, with free surfaces in the z-direction and periodic boundary conditions in the other directions. The mesh file for this simulation is given below:

Mesh_ThinFilm.txt

```
!********************************************************************************
!This file contains a UNIFORM cubic mesh input for the parallel SRSCD
!code. The mesh will be read in via coordinates (x,y,z) of each element center
!as well as the material number of each element (eg. material 1=Fe, 2=Cu, etc)
!
!The first values that will be read in are the number of elements in each direction
!(x,y,z).
!
!********************************************************************************

!Specify mesh type (currently two types=periodic and freeSurfaces)
meshType
freeSurfaces

!Specify element lengths (nm)
length
  20.000000000000000

!Specify max and min of each coordinate (nm)
!NOTE: the actual max, min will be max, min +/- length/2 because these are assumed
!to be the center of the elements and not the edges
xminmax
  10.000000000000000        190.00000000000000
yminmax
  10.000000000000000        190.00000000000000
zminmax
  10.000000000000000        90.000000000000000

!Specify number of elements in each direction
numx
         10
numy
         10
numz
          5

!Coordinates of element centers (x, y, z, elementType) (nm)
!NOTE: these must be ordered in the same way as the connectivity matrix (loop x, then y, then z)
elements                        !tells computer that the following values are elements
  10.000000000000000        10.000000000000000        10.000000000000000                1
  30.000000000000000        10.000000000000000        10.000000000000000                1
  50.000000000000000        10.000000000000000        10.000000000000000                1
  70.000000000000000        10.000000000000000        10.000000000000000                1
  90.000000000000000        10.000000000000000        10.000000000000000                1
  110.00000000000000        10.000000000000000        10.000000000000000                1
  130.00000000000000        10.000000000000000        10.000000000000000                1
  150.00000000000000        10.000000000000000        10.000000000000000                1
  170.00000000000000        10.000000000000000        10.000000000000000                1
  190.00000000000000        10.000000000000000        10.000000000000000                1

  10.000000000000000        30.000000000000000        10.000000000000000                1
  30.000000000000000        30.000000000000000        10.000000000000000                1
  50.000000000000000        30.000000000000000        10.000000000000000                1
  70.000000000000000        30.000000000000000        10.000000000000000                1
  90.000000000000000        30.000000000000000        10.000000000000000                1
  110.00000000000000        30.000000000000000        10.000000000000000                1
  130.00000000000000        30.000000000000000        10.000000000000000                1
  150.00000000000000        30.000000000000000        10.000000000000000                1
  170.00000000000000        30.000000000000000        10.000000000000000                1
  190.00000000000000        30.000000000000000        10.000000000000000                1

etc ...
```

The parameters.txt file for this simulation is given below:

parameters.txt

```
!************************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!************************************************************************************************
```

```
!Tell computer location of mesh input file
meshFile
Mesh_ThinFilm.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
no

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
no

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
debugRestart.in

!Tell computer if the number of materials we are reading in
numMaterials
1

!Tell computer the location of the material input file(s)
materialFile
Fe_Defects.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
FrenkelPair

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
AnnealedCascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
nonAdaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!*****************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!*****************************************************************************************

start                        !begin parameters

temperature                !Temperature, in K
273d0

dpaRate                       ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
1.620d-2

HeDPA                         ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                                !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                 !(Fe)                          1.17d-2                !(Cu)                ! atomsize (nm^3)

burgers
.287d0                          !(Fe)                          .36d0                          !(Cu)                ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
1d-2                          ! total DPA in simulation 1

annealTemp
273d0                         !Annealing temperature, in K

annealSteps
1                                ! number of steps in the annealing process

annealTime
0d0                                ! total anneal time (s)

annealType
add                                ! (add) for adding constant temperature increment at each anneal step, (mult) for multiplying the temperature by a constant at each anneal st

annealTempInc
0d0                                ! Temperature increment for each annealing step (either added or multiplied by)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
no

grainSize
33000d0                       ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
1d-5                          ! dislocation density (nm^-2)

impurityConc
30d-6                         !30d-4                        ! carbon impurity concentration (atomic fraction)

max3DInt
4                                !maximum size for SIA defect to diffuse in 3D

cascadeVolume
1000                         !volume of cascade (nm^3) - used for cascade-defect interactions

numSims
```

```
1                               !number of times to repeat simulation

restartToggle
no                       !toggles output of debug restart file

end                               !Tag for the end of the parameters file

!****************************************************************************************************
!Adaptive meshing parameters
!****************************************************************************************************

fineStart                ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                               ! Length of one cascade implantation element (nm)

numxFine
6                               ! number of cascade elements in x-direction (fine mesh)

numyFine
6                               ! number of cascade elements in y-direction

numzFine
6                               ! number of cascade elements in z-direction

end                               ! Tag for the end of meshing parameters
```

Note that although the files AnnealedCascades.txt and debugRestart.in are listed in parameters.txt, they are not needed as the cascade and debug restart toggles are turned off. The only other file needed to run this simulation is Fe_Defects.txt, which is the standard version of this file and is not included here.

From the simulation parameters listed in parameters.txt, we can see that this simulation includes uniform Frenkel pair irradiation at 273 K (0 C) at 1.62 dpa·s^{-1} to a total dose of 10^{-2} dpa. No annealing is carried out. We can also see that the grain boundary sink mechanism for defect removal from the system has been turned off. Using the output from postpr_1.txt, we can generate a similar figure to the one in the previous section, but for these material and implantation conditions. These results are shown in Figure 4.2.

The results of this simulation strongly contrast with the results of cascade implantation at 100 C shown in Figure 4.1 due to the higher dose rate, lower dose rate, and different damage type leading to much smaller defects on average inside the material.

4.2.2 Frenkel pairs and helium in α-Fe thin films

This simulation is identical to the previous simulation, except that uniform helium implantation is also conducted with a helium to dpa ratio of 1.16×10^{-2}. Therefore, the mesh file is the same as the previous example and the only change in parameters.txt is the addition of a non-zero helium to dpa ratio. The impact of adding helium on results can be seen in Figure 4.3.

4.2.3 Frenkel pairs and helium in α-Fe thin films with non-uniform implantation profile

This simulation is identical to the previous simulation, except that instead of a uniform implantation profile for damage and helium, a non-uniform input file ImplantProfile.txt is used. The values in ImplantProfile.txt are taken from SRIM simulations of helium ion implantation in iron thin films. The only change to parameters.txt required to run this simulation rather than the previous simulation is changing implantDist to 'NonUniform' and naming the implantFile correctly. The implantation profile used for this simulation is the same one shown in Section 3.2.6.

The impact of a non-uniform damage profile on vacancy, SIA, and helium cluster accumulation can be seen in Figure 4.4. For the damage conditions chosen here, no significant differences are seen between a uniform damage profile and the non-uniform damage profile. Any differences that would occur at higher temperatures or lower dose rates are caused by the fact that most damage is done near the incident surface, while the average depth for helium implantation is much further. Therefore, damage evolution occurs differently due to the varying helium to dpa ratio at different depths in the material.

The method of using a non-uniform damage profile is therefore not necessary in the particular simulation used here as an example. As an exercise, change the temperature in both this simulation and the simulation in the previous section until significant differences are observed in output between the two simulations. This type of exercise can be used to identify when various approximations can be applied to simulations without significant loss of information.

4.3 Isochronal annealing

SRSCD can be used to simulate isochronal annealing in a manner similar to the work of Fu et al. (2005) and Dalla Torre et al. (2006). In these simulations, Frenkel pairs are implanted into a α-Fe bulk material to a low dose (simulating electron irradiation, which is typically low dose) at liquid helium temperature. At this temperature, reaction rates are so low that the only reaction that is carried out is Frenkel pair implantation.

After implantation, the temperature is increased to 127.6 K and left for 5 minutes. At the end of the 5 minutes, the temperature is increased by a (multiplicative) factor of 1.03 and the material is annealed for another 5 minutes. This process is repeated for 44 annealing steps, ending at approximately 470 K.

Isochronal annealing is a standard experimental technique for measuring the activation energy associated with various different mechanisms of defect evolution in irradiated metals. In these experiments, the resistivity of the irradiated metal is measured as a function of temperature during the annealing stages. As different defects become mobile and therefore cause defects to cluster or annihilate, the resistivity changes, typically in steps. The first derivative of the resistivity as a function of temperature gives a chart whose peaks correspond to the activation of various defect mobilities. This can be used to determine the migration energies of these defects using Arrhenius laws and assuming standard attempt frequencies. An example of such an experiment can be found in Takaki et al. (1983), and kinetic Monte Carlo and cluster dynamics reproductions of these experiments can be found in Fu et al. (2005) and Dalla Torre et al. (2006).

To simulate isochronal annealing with SRSCD, a large $40 \times 40 \times 40$ periodic mesh of 20 nm volume elements is created. The beginning of the mesh file is given below:

Mesh_IsoAnneal.txt

```
!*******************************************************************************
!This file contains a UNIFORM cubic mesh input for the parallel SRSCD
!code. The mesh will be read in via coordinates (x,y,z) of each element center
!as well as the material number of each element (eg. material 1=Fe, 2=Cu, etc)
!
!The first values that will be read in are the number of elements in each direction
!(x,y,z).
!
!*******************************************************************************

!Specify mesh type (currently two types=periodic and freeSurfaces)
meshType
periodic

!Specify element lengths (nm)
length
  20.000000000000000

!Specify max and min of each coordinate (nm)
!NOTE: the actual max, min will be max, min +/- length/2 because these are assumed
!to be the center of the elements and not the edges
xminmax
  10.000000000000000        790.00000000000000
yminmax
  10.000000000000000        790.00000000000000
zminmax
  10.000000000000000        790.00000000000000

!Specify number of elements in each direction
numx
         40
numy
         40
numz
         40

!Coordinates of element centers (x, y, z, elementType) (nm)
!NOTE: these must be ordered in the same way as the connectivity matrix (loop x, then y, then z)
elements                        !tells computer that the following values are elements
  10.000000000000000        10.000000000000000        10.000000000000000                1
  30.000000000000000        10.000000000000000        10.000000000000000                1
  50.000000000000000        10.000000000000000        10.000000000000000                1
  70.000000000000000        10.000000000000000        10.000000000000000                1
  90.000000000000000        10.000000000000000        10.000000000000000                1
  110.00000000000000        10.000000000000000        10.000000000000000                1
  130.00000000000000        10.000000000000000        10.000000000000000                1
  150.00000000000000        10.000000000000000        10.000000000000000                1
  170.00000000000000        10.000000000000000        10.000000000000000                1
  190.00000000000000        10.000000000000000        10.000000000000000                1
  210.00000000000000        10.000000000000000        10.000000000000000                1
  230.00000000000000        10.000000000000000        10.000000000000000                1
  250.00000000000000        10.000000000000000        10.000000000000000                1
  270.00000000000000        10.000000000000000        10.000000000000000                1
  290.00000000000000        10.000000000000000        10.000000000000000                1
  310.00000000000000        10.000000000000000        10.000000000000000                1
  330.00000000000000        10.000000000000000        10.000000000000000                1
  350.00000000000000        10.000000000000000        10.000000000000000                1
  370.00000000000000        10.000000000000000        10.000000000000000                1
  390.00000000000000        10.000000000000000        10.000000000000000                1
  410.00000000000000        10.000000000000000        10.000000000000000                1
  430.00000000000000        10.000000000000000        10.000000000000000                1
  450.00000000000000        10.000000000000000        10.000000000000000                1
  470.00000000000000        10.000000000000000        10.000000000000000                1
  490.00000000000000        10.000000000000000        10.000000000000000                1
  510.00000000000000        10.000000000000000        10.000000000000000                1
  530.00000000000000        10.000000000000000        10.000000000000000                1
  550.00000000000000        10.000000000000000        10.000000000000000                1
  570.00000000000000        10.000000000000000        10.000000000000000                1
```

```
   590.00000000000000        10.000000000000000        10.000000000000000               1
   610.00000000000000        10.000000000000000        10.000000000000000               1
   630.00000000000000        10.000000000000000        10.000000000000000               1
   650.00000000000000        10.000000000000000        10.000000000000000               1
   670.00000000000000        10.000000000000000        10.000000000000000               1
   690.00000000000000        10.000000000000000        10.000000000000000               1
   710.00000000000000        10.000000000000000        10.000000000000000               1
   730.00000000000000        10.000000000000000        10.000000000000000               1
   750.00000000000000        10.000000000000000        10.000000000000000               1
   770.00000000000000        10.000000000000000        10.000000000000000               1
   790.00000000000000        10.000000000000000        10.000000000000000               1
etc...
```

No other input files are required other than the standard Fe_Defects.txt file required by all simulations. The parameters.txt file is given by the following:

parameters.txt

```
!************************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!************************************************************************************************

!Tell computer location of mesh input file
meshFile
Mesh_IsoAnneal.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
no

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
no

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
debugRestart.in

!Tell computer if the number of materials we are reading in
numMaterials
1

!Tell computer the location of the material input file(s)
materialFile
Fe_Defects.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
FrenkelPair

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
Fe_Cascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
nonAdaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!************************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!************************************************************************************************

start                        !begin parameters

temperature                !Temperature, in K
4.5d0

dpaRate                          ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
3.36d-4

HeDPA                        ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                                !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                 !(Fe)                          1.17d-2                !(Cu)                ! atomsize (nm^3)

burgers
0.287d0                !(Fe)                         0.36d0                         !(Cu)                ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
4.95d-8                        !2d-6                        ! total DPA in simulation 1

annealTemp
127.5998372114        !Starting annealing temperature, in K
```

```
annealSteps
44                                ! number of steps in the annealing process

annealTime
13200                          ! total anneal time (s)

annealType
mult                           ! (add) for adding constant temperature increment at each anneal step, (mult) for multiplying the temperature by a constant at each anneal step

annealTempInc
1.03                           ! Temperature increment for each annealing step (either added or multiplied by)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
yes

grainSize
33000d0                        ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
0d0                               !1d-5                        ! dislocation density (nm^-2)

impurityConc
0d0                               !30d-6                        !30d-4                        ! carbon impurity concentration (atomic fraction)

max3DInt
4                                 !maximum size for SIA defect to diffuse in 3D

cascadeVolume
1000                           !volume of cascade (nm^3) - used for cascade-defect interactions

numSims
1                                 !number of times to repeat simulation

end                                 !Tag for the end of the parameters file

!****************************************************************************************************
!Adaptive meshing parameters
!****************************************************************************************************

fineStart               ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                                 ! Length of one cascade implantation element (nm)

numxFine
6                                 ! number of cascade elements in x-direction (fine mesh)

numyFine
6                                 ! number of cascade elements in y-direction

numzFine
6                                 ! number of cascade elements in z-direction

end                                 ! Tag for the end of meshing parameters
```

In this file, we can see the use of annealTemp, annealSteps, annealTime, and annealTempInc.

To plot the output, in Figure 4.5 the concentration of vacancies as a function of temperature during annealing is plotted along with the derivative of the vacancy concentration. The peaks in the graphs of the derivative of vacancy concentration should correspond to peaks in experimental studies of resistivity recovery, which can then be used to ensure that the correct defect migration energies are being used. In Figure 4.5, the first two peaks correspond to activation of SIA mobility (correlated followed by uncorrelated recombination with vacancies). The small intermediate peaks correspond to small SIA cluster mobility, and the final large peak beginning at 300 K corresponds to the onset of vacancy mobility.

4.4 Debug restart

This simulation demonstrates how the debug restart function can be used. This function allows simulations to restart from a partially completed simulation. This can be beneficial in searching for bugs, creating nonstandard irradiation sequences that cannot currently be accessed, and measuring the parallel performance of the code.

Restarting the code from a partially complete simulation is done over two steps:

1. First, a simulation is run using a normal set of parameters. In the parameters.txt file's second section, **restartToggle** is set to 'yes'. This causes the program to output files named Restart_x.in during each output step, where x is the number of the output step. If, for example, the code crashes after output step 5, then the file Restart_5.in represents the most recent output of the defects in the simulation.
2. Second, a simulation is run with the **debugRestart** toggle set to 'yes' and the path of the **debugRestartFile** set to the name of the restart file (Restart_5.in in this example, or it can be renamed to a standard name such as debugRestart.in). The second SRSCD simulation must be run with the same number of processors and all other input files the same as the first simulation.

The form of the debugRestart.in file is the same as in Section 3.2.7 for this example, and the parameters.txt file for the second simulation is given below for a simulation of cascade implantation in bulk iron (similar to Section 4.1).

parameters.txt

```
!****************************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!****************************************************************************************************

!Tell computer location of mesh input file
meshFile
Mesh_Bulk.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
no

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
yes

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
debugRestart.in

!Tell computer if the number of materials we are reading in
numMaterials
1

!Tell computer the location of the material input file(s)
materialFile
Fe_Defects.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
Cascade

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
Fe_Cascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
adaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!****************************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!****************************************************************************************************

start                        !begin parameters

temperature                !Temperature, in K
343d0

annealTemp
343d0                        !Annealing temperature, in K

dpaRate                        ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
7d-7

HeDPA                        ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                                !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                !(Fe)                        1.17d-2                !(Cu)                ! atomsize (nm^3)

burgers
.287d0                        !(Fe)                        .36d0                        !(Cu)                ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
1d-3                        ! total DPA in simulation 1

annealTime
0d0                                ! total anneal time (s)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
yes

grainSize
33000d0                        ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
1d-5                        ! dislocation density (nm^-2)

impurityConc
30d-6                        ! carbon impurity concentration (atomic fraction)

max3DInt
4                                !maximum size for SIA defect to diffuse in 3D
```

```
cascadeVolume
1000d0                        !volume of cascade (nm^3) - used for cascade-defect interactions

numSims
1                               !number of times to repeat simulation

end                               !Tag for the end of the parameters file

!****************************************************************************************
!Adaptive meshing parameters
!****************************************************************************************

fineStart               ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                               ! Length of one cascade implantation element (nm)

numxFine
6                               ! number of cascade elements in x-direction (fine mesh)

numyFine
6                               ! number of cascade elements in y-direction

numzFine
6                               ! number of cascade elements in z-direction

end                               ! Tag for the end of meshing parameters
```

Note that toggle **debugRestart** is set to 'yes'. This requires that the file debugRestart.in is present in the folder that SRSCD is run from.

4.5 Grain boundary simulations

The ability to simulate defect accumulation within grain boundaries has been implemented with SRSCD. In these simulations, the mesh is divided into two regions: the interior of the grain and the grain boundary. In each region, a separate input file gives the allowed defects, reactions, and reaction rates. For the study of grain boundary sink efficiency, defects diffuse to and are trapped by the grain boundary, but they can be emitted by the grain boundary with a binding energy that is varied. By varying this binding energy as well as the properties of defects trapped in the grain boundary, estimates can be made of the various expected behaviors caused by the presence of grain boundaries in materials. This can lead to, among other things, void denuded zones in the grain of width that varies depending on the grain boundary character.

The parameters.txt file for this simulation is given below. Notice the use of two material types and two material input files:

parameters.txt

```
!****************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!****************************************************************************************

!Tell computer location of mesh input file
meshFile
Mesh_Bicrystal.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
no

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
no

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
debugRestart.in

!Tell computer if the number of materials we are reading in
numMaterials
2

!Tell computer the location of the material input file(s)
materialFile
Defects_Fe.txt

materialFile
Defects_GB.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
FrenkelPair

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
```

```
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
Fe_Cascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
nonAdaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!*****************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!*****************************************************************************************

start                       !begin parameters

temperature               !Temperature, in K
293d0

dpaRate                       ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
1d-7

HeDPA                       ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                               !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                !(Fe)                        1.17d-2                !(Cu)               ! atomsize (nm^3)

burgers
.287d0                        !(Fe)                        .36d0                        !(Cu)               ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
1d-3                        ! total DPA in simulation 1

annealTemp
293d0                        !Annealing temperature, in K

annealSteps
1                               ! number of steps in the annealing process

annealTime
0d0                               ! total anneal time (s)

annealType
add                               ! (add) for adding constant temperature increment at each anneal step, (mult) for multiplying the temperature by a constant at each anneal st

annealTempInc
0d0                               ! Temperature increment for each annealing step (either added or multiplied by)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
no

grainSize
33000d0                       ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
1d-5                        ! dislocation density (nm^-2)

impurityConc
30d-6                        !30d-4                        ! carbon impurity concentration (atomic fraction)

max3DInt
4                               !maximum size for SIA defect to diffuse in 3D

cascadeVolume
1000                        !volume of cascade (nm^3) - used for cascade-defect interactions

HeSIAToggle
no                               !Toggle whether or not we are allowing He-SIA clusters to form ('yes' or 'no')

SIAPinToggle
no                               !Toggle whether or not we are allowing HeV clusters to pin SIA clusters ('yes' or 'no')

SIAPinMin
1                               !Smallest size of SIA that can pin at HeV clusters

numSims
5                               !number of times to repeat simulation

end                               !Tag for the end of the parameters file

!*****************************************************************************************
!Adaptive meshing parameters
!*****************************************************************************************

fineStart                ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                               ! Length of one cascade implantation element (nm)

numxFine
6                               ! number of cascade elements in x-direction (fine mesh)

numyFine
6                               ! number of cascade elements in y-direction

numzFine
6                               ! number of cascade elements in z-direction

end                               ! Tag for the end of meshing parameters
```

The contents of Defects_GB.txt will not be displayed here for brevity, but the main difference is the addition of a reaction for defects to emit from the grain boundary back into the bulk and all clustering and diffusion reaction rates are changed to 2D reaction rates, as discussed in the introduction. Also, the migration and binding energies of various defects inside the grain boundary have been varied as part of a parameter study. The mesh and mesh generation file used in this simulation are the same as in Section 3.2.2.2.

An example of the use of simulations of this type is shown below in Figure 4.6. The vacancy concentration, average size, and sink efficiencies of the grain boundary for vacancies and SIAs are shown as a function of the binding energy of vacancies and SIAs to the grain boundary. In addition, the depth profile of vacancies is shown for several grains with binding energies chosen from a list given by Tschopp et al. (2012). This type of simulation can be used to identify grain boundary types with different expected behavior.

4.6 Polycrystal simulations

SRSCD has also been implemented with the capability of simulating polycrystalline domains. In this case, instead of having multiple defect files for different domains as in the case of grain boundaries described in Section 4.5, there are multiple domains representing different grains but they all share the same defect data file (Fe_Defects.txt). The boundaries between the grains are treated as perfect sinks here, although this choice could be modified in the future. The MeshGenInput.txt file, which creates the polycrystalline mesh, is the following:

MeshGenInput.txt

```
!*********************************************************************************************
!
! This is an input file for the mesh generator program that will output mesh text files that are
! usable in SRSCD_par. The goal of this program is to have an easy and quick way to create uniform
! meshes for SRSCD without changing the (more demanding) inputs in SRSCD.
!
! Inputs: Volume element length (uniform volume elements only), meshType (periodic or free surfaces),
!                 number of elements in x, y, and z-directions.
!
! Output: Text file that can be read into SRSCD_par with mesh. Each element's center coordinates
!                as well as the global max/min coordinates in the x, y, and z directions will be given.
!                All materials will be of type 1.
!
!*********************************************************************************************

filename                !Determines the file name of the SRSCD_par input file
Mesh_Polycrystal.txt

meshType                !Tells meshGen whether to create a material with periodic BC's or free surfaces in the z-dir ('periodic or freeSurfaces')
periodic

length (nm)                !Specify the length of volume elements in nm (uniform cubic mesh only)
5d0

numGrains                !Specify the number of grains (for testing purposes only, will evenly distribute grains in z-direction)
30

iMPALEtoggle                !Toggle whether we are reading from an iMPALE polyxtal file
yes

iMPALEfilename                !name of iMPALE file to read in
PolyXtal30Grains.xyz

numx                        !If reading in from iMPALE file, these should match the data in the iMPALE file
80
numy
20
numz
20

end
```

Note the use of an xyz input file, PolyXtal30Grains.xyz. This file is output by iMPALE, which was used to generate the polycrystal in this case. The parameters.txt file for this simulation is the following:

parameters.txt

```
!*********************************************************************************************
!2014/05/21: This is the first version of the overall parameters file that will be used for the
!synchronous parallel Monte Carlo code. It also contains filenames of various other input files
!used in this code.
!*********************************************************************************************

!Tell computer location of mesh input file
meshFile
```

```
Mesh_Polycrystal.txt

!Tell computer if this is a uniform or non-uniform mesh file
meshType
uniform

!Toggle for calculating defect diffusion interaction with strain field
strainField
no

!DEBUG TOOL: toggle restart from data file (instead of restarting at 0 dpa) ('yes' or 'no')
debugRestart
no

!DEBUG TOOL: name of file to restart from (only used if debugRestart=='yes')
debugRestartFile
debugRestart.in

!Tell computer if the number of materials we are reading in
numMaterials
1

!Tell computer the location of the material input file(s)
materialFile
Fe_Defects.txt

!Type of implantation ('Cascade' for cascades, 'FrenkelPair' for Frenkel pairs)
implantType
FrenkelPair

!Toggle Monte Carlo cascade introduction vs explicit cascade introduction (for better weak scaling)
!('MonteCarlo' for MC cascade implantation, 'explicit' for explicit cascade implantation)
implantScheme
MonteCarlo

!Tell computer the location of the cascade input file
cascadeFile
AnnealedCascades.txt

!Tell computer whether we are using adaptive meshing protocol or not (adaptive for yes, nonAdaptive for no)
meshingType
nonAdaptive

!Tell computer whether we are implanting defects uniformly or if we have implantation listed separately
!at each material point (for non-uniform implantation) - (Uniform or NonUniform)
implantDist
Uniform

!Tell computer the name of the data file containing non-uniform implantation profile
implantFile
ImplantProfile.txt

!****************************************************************************************************
!Simulation Parameters
!
!The order of these parameters can be adjusted but each parameter must come directly after the tag
!for it. For example, the temperature must come directly after 'temperature'
!****************************************************************************************************

start                       !begin parameters

temperature               !Temperature, in K
293d0

dpaRate                       ! NOTE: if the implant profile is non-uniform, this should be the AVERAGE DPA rate.
7d-7

HeDPA                         ! Helium/DPA ratio! He ImplantRate (atomic fraction / sec) - only used if implantDist='Uniform'
0d0                                !(NOTE: if implantDist='NonUniform', must have HeDPA .NE. 0d0 to add nonuniform He implant rates)

atomSize
1.182d-2                !(Fe)                        1.17d-2                !(Cu)               ! atomsize (nm^3)

burgers
.287d0                          !(Fe)                          .36d0                          !(Cu)               ! dislocation loop burgers vector (lattice constant) (nm)

totalDPA
1d-1                        ! total DPA in simulation 1

annealTemp
293d0                       !Annealing temperature, in K

annealSteps
1                              ! number of steps in the annealing process

annealTime
0d0                            ! total anneal time (s)

annealType
add                            ! (add) for adding constant temperature increment at each anneal step, (mult) for multiplying the temperature by a constant at each anneal st

annealTempInc
0d0                            ! Temperature increment for each annealing step (either added or multiplied by)

grainBoundaries        !Toggle whether we are going to include the effect of grain boundaries (Removing defects that travel too far)
no

grainSize
33000d0                       ! Mean free path for interstitial clusters to travel before removal (nm)

dislocDensity
1d-5                        ! dislocation density (nm^-2)

impurityConc
30d-6                       !30d-4                      ! carbon impurity concentration (atomic fraction)

max3DInt
4                              !maximum size for SIA defect to diffuse in 3D

cascadeVolume
1000                        !volume of cascade (nm^3) - used for cascade-defect interactions

numSims
1                              !number of times to repeat simulation

polycrystal
```

```
yes                                !identify if we have a single crystal or polycrystal simulation

numGrains
30                                !idenify the number of grains in our polycrystal simulation (default=1)

vtkToggle
yes                                !toggle output in VTK format

end                                !Tag for the end of the parameters file

!*********************************************************************************************
!Adaptive meshing parameters
!*********************************************************************************************

fineStart                ! Begin fine mesh parameters (for adaptive meshing only, not used if adaptive meshing turned off)

fineLength
5                                ! Length of one cascade implantation element (nm)

numxFine
6                                ! number of cascade elements in x-direction (fine mesh)

numyFine
6                                ! number of cascade elements in y-direction

numzFine
6                                ! number of cascade elements in z-direction

end                                ! Tag for the end of meshing parameters
```

Note that numMaterials is 1 here, though there are several grains. The toggle polycrystal is set to 'yes' and numGrains is set to 30, the number of grains in the xyz file and in MeshGenInput. Also not that VTKToggle is set to 'yes', because a paraView output of this simulation using a VTK file is a convenient way to visualize this polycrystalline domain. The image generated by ParaView using such a file for a 400 nm $\times$ 100 nm $\times$ 100 nm domain with 15 grains is shown in Figure 4.7.

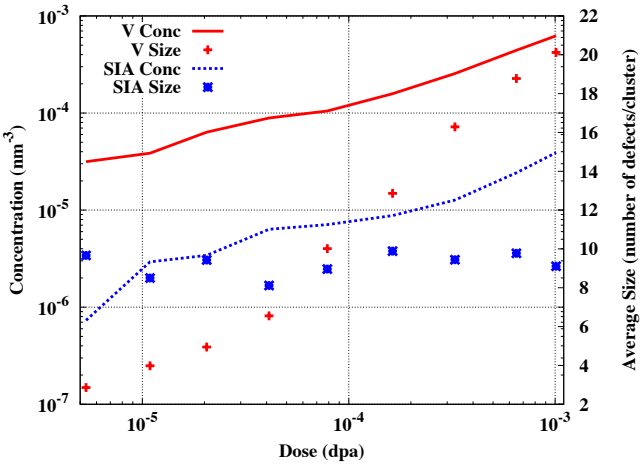


Figure 4.1: Vacancy and SIA cluster density and average size as a function of radiation dose for 20 keV cascade implantation in bulk α-Fe at 100 C (373 K) at a dose rate of 7×10^{-7} dpa·s^{-1}.

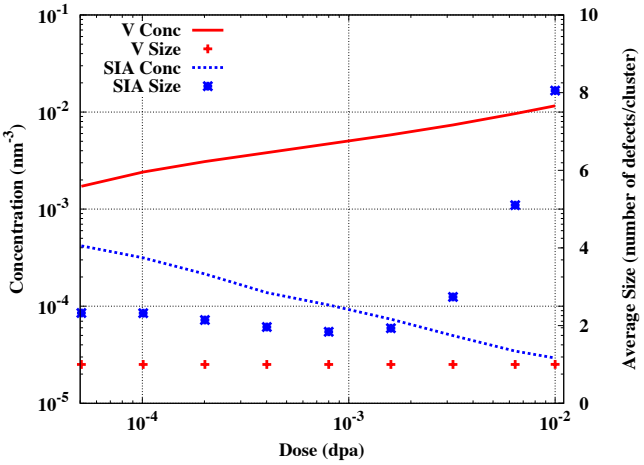


Figure 4.2: Vacancy and SIA cluster density and average size as a function of radiation dose for Frenkel pair implantation in α-Fe thin films (100 nm) at 0 C and a dose rate of 1.62×10^{-2} dpa·s^{-1}.

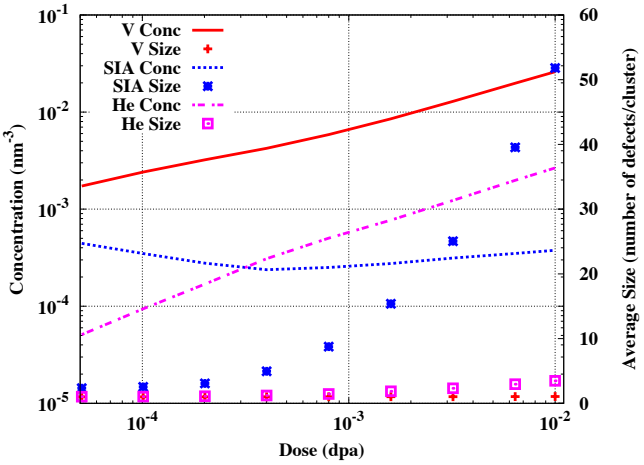


Figure 4.3: Vacancy, SIA, and He cluster density and average size as a function of radiation dose for Frenkel pair and helium implantation in α-Fe thin films (100 nm) at 0 C, a dose rate of 1.6×10^{-2} dpa·s^{-1}, and a helium to dpa ratio of 1.16×10^{-2}.

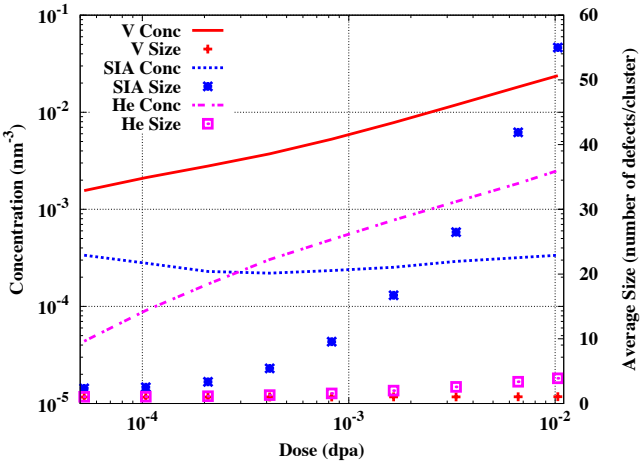


Figure 4.4: Vacancy, SIA, and He cluster density and average size as a function of radiation dose for nonuniform Frenkel pair and helium implantation in α-Fe thin films (100 nm) at 0 C, an average dose rate of 1.6×10^{-2} dpa·s^{-1}, and an average helium to dpa ratio of 1.16×10^{-2}.

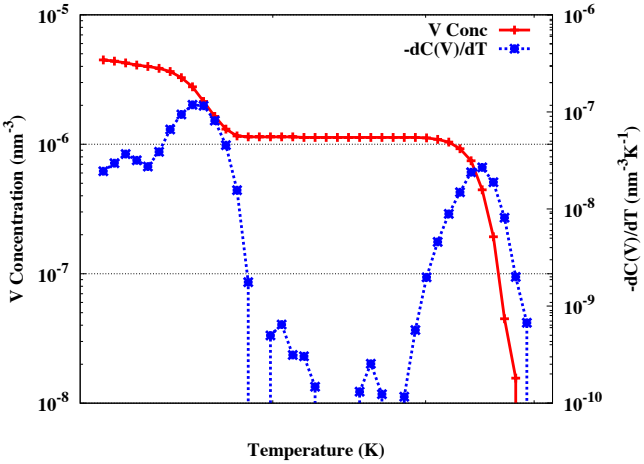


Figure 4.5: Vacancy concentration and derivative of vacancy concentration with temperature during annealing as described above in electron-irradiated bulk α-Fe. Peaks correspond to the activation of SIA and vacancy mobility.

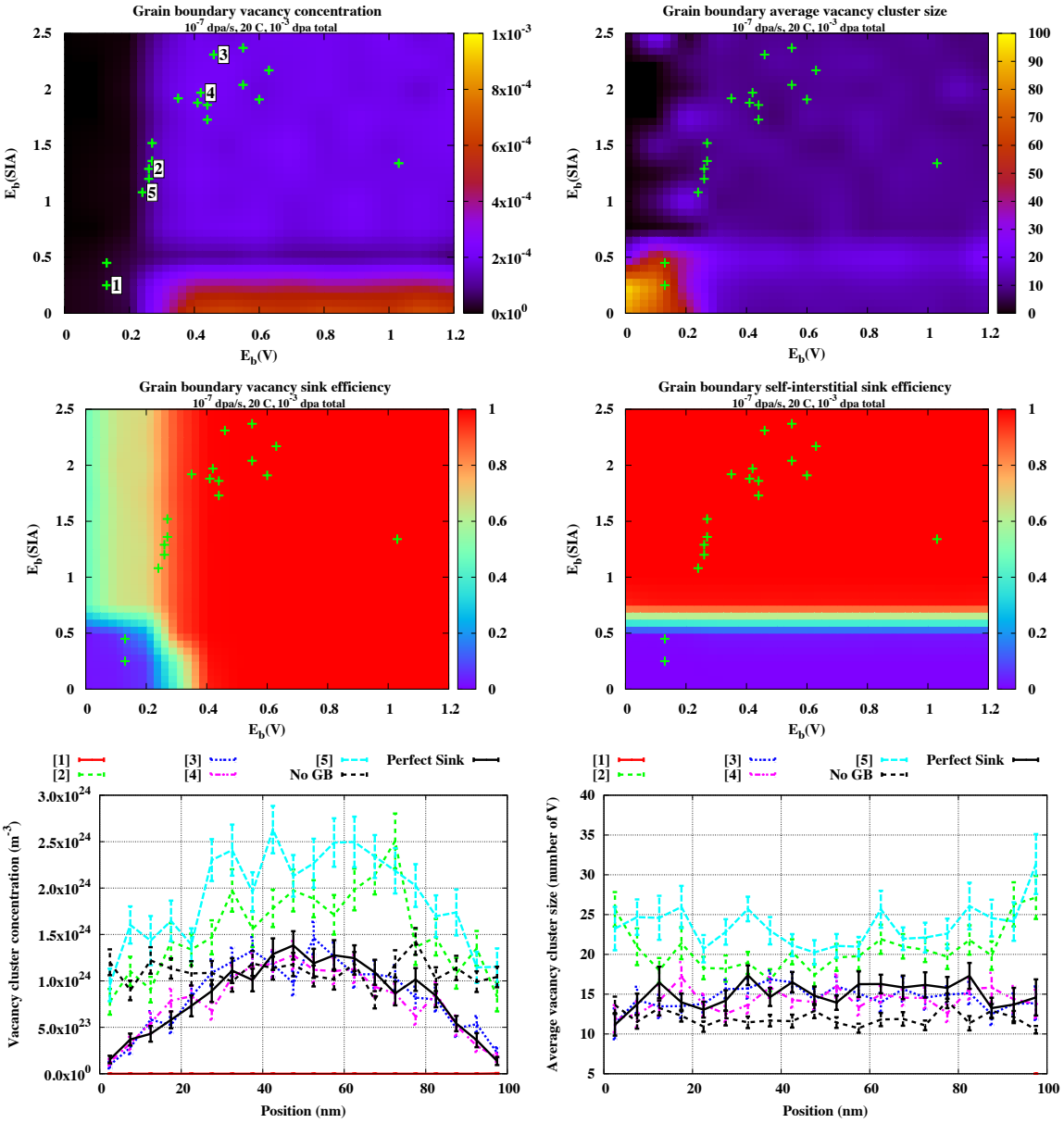


Figure 4.6: Maps of void concentration, average size, and sink efficiencies η_{V} and η_{I} on the grain boundary as a function of binding energy of vacancies and self-interstitials to the grain boundary, at a total dose of 10^{-3} dpa. Marked points indicate average binding energies of defects to grain boundaries found by Tschopp et al. (2012). Profiles showing vacancy cluster concentration and average size inside the bulk for several chosen values of E_b(V) and E_b(SIA), corresponding to values for specific grain boundaries found by Tschopp et al. (2012).

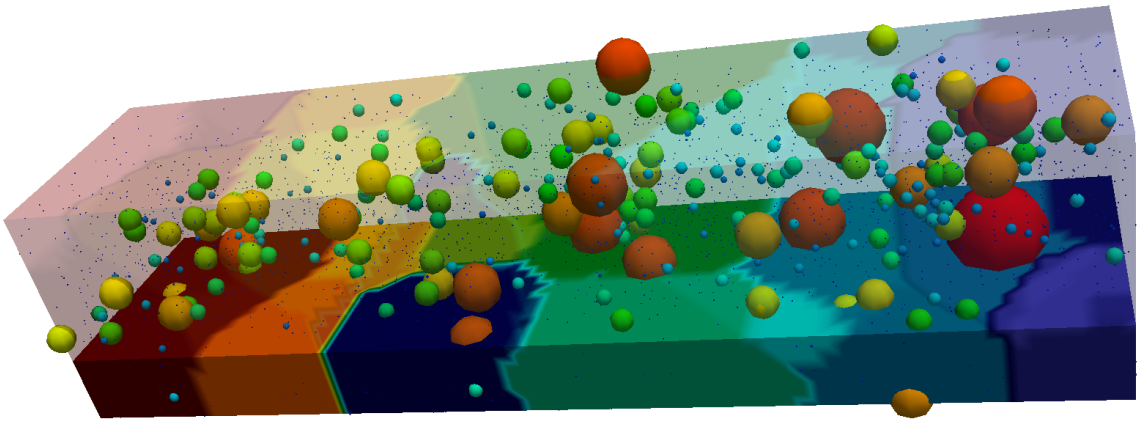


Figure 4.7: Example image of vacancy accumulation (spheres) in a polycrystalline domain with 15 grains (shaded colors) using ParaView to display data output by SRSCD.

Chapter 5

Modules Index

5.1 Modules List

Here is a list of all modules with brief descriptions:

derivedtype
: Mod_DerivedTypes contains the variable derived types created for SRSCD 109

meshreader
: Module Mesh Reader: creates a processor mesh and volume element mesh using input from file 110

mod_srscd_constants
: Module mod_SRSCD_constants (list of globally shared variables and pointers) 117

randdp
: Module randdp: double precision random number generation 133

reactionrates
: Module: ReactionRates - adds reactions and calculates rates for the various reaction types in the system 135

Chapter 6

Data Type Index

6.1 Data Types List

Here are the data types with brief descriptions:

derivedtype::bindingfunction
: Type: binding function (list of functional forms for defect binding energies read in from input file) 153

derivedtype::bindingsingle
: Type: binding single (list of binding energies) . . . 154

derivedtype::boundarymesh
: Type: boundary mesh (one array of this type per processor) . . . 155

derivedtype::cascade
: Type: cascade (pointer list containing defects and reactions for cascades that are currently present in the system) . . . 156

derivedtype::cascadedefect
: Type: cascade defect (pointer list of defect types in a cascade) . . . 158

derivedtype::cascadeevent
: Type: cascade event (pointer list of read-in cascade data from input file) . . . 159

derivedtype::defect
: Type: defect (pointer list) . . . 160

derivedtype::defectupdatetracker
: Type: defect update tracker (pointer list) . . . 161

derivedtype::diffusionfunction
: Type: diffusion function (list of functional forms for defect diffusivities read in from input file) . . . 163

derivedtype::diffusionsingle
: Type: diffusion single (list of single migration energies and diffusion prefactors) . . . 164

derivedtype::dipoletensor
: Type: dipole tensor (stored dipole tensor data for various defect types, input from file) . . . 165

derivedtype::mesh
: Type: mesh (local, one array of this type per processor) . . . 166

derivedtype::processordata
: Type: processorData (one variable per processor) . . . 167

derivedtype::reaction
: Type: reaction (pointer list) . . . 168

derivedtype::reactionparameters
: Type: reaction parameters (list of allowed reactions, input from file) . . . 170

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Cascade_implantation.f90 . . . 173
CoarseMesh_subroutines.f90 . . . 176
Deallocate_Lists.f90 . . . 180
Debug_subroutines.f90 . . . 182
Defect_attributes.f90 . . . 184
dprand.f90 . . . 186
FineMesh_subroutines.f90 . . . 186
Initialization_subroutines.f90 . . . 189
kMC_subroutines.f90 . . . 194
MeshReader.f90 . . . 199
Misc_functions.f90 . . . 200
mod_derivedtypes.f90 . . . 201
mod_srscd_constants.f90 . . . 202
Postprocessing_srscd.f90 . . . 207
ReactionRates.f90 . . . 212
Read_inputs.f90 . . . 214
SRSCD_par.f90 . . . 217
StrainSubroutines.f90 . . . 218

Chapter 8

Module Documentation

8.1 derivedtype Module Reference

mod_DerivedTypes contains the variable derived types created for SRSCD.

Data Types

- type bindingfunction

  Type: binding function (list of functional forms for defect binding energies read in from input file)
- type bindingsingle

  Type: binding single (list of binding energies)
- type boundarymesh

  Type: boundary mesh (one array of this type per processor)
- type cascade

  Type: cascade (pointer list containing defects and reactions for cascades that are currently present in the system)
- type cascadedefect

  Type: cascade defect (pointer list of defect types in a cascade)
- type cascadeevent

  Type: cascade event (pointer list of read-in cascade data from input file)
- type defect

  Type: defect (pointer list).
- type defectupdatetracker

  Type: defect update tracker (pointer list)
- type diffusionfunction

  Type: diffusion function (list of functional forms for defect diffusivities read in from input file)
- type diffusionsingle

  Type: diffusion single (list of single migration energies and diffusion prefactors)
- type dipoletensor

  Type: dipole tensor (stored dipole tensor data for various defect types, input from file)
- type mesh

  Type: mesh (local, one array of this type per processor)
- type processordata

  Type: processorData (one variable per processor)
- type reaction

  Type: reaction (pointer list)
- type reactionparameters

  Type: reaction parameters (list of allowed reactions, input from file)

8.1.1 Detailed Description

mod_DerivedTypes contains the variable derived types created for SRSCD.

This module contains derived variable types used in SRSCD. They are roughly divided into defect and reaction lists (pointer lists), processor/mesh variables (used for the structure of the code), cascade information such as lists of defects, and material properties/allowed reactions (including defect diffusion and binding energies, allowed clustering reactions, etc). Some comments on the variable types introduced here are below:

General comments:

Defects and reactions are all treated as pointers, and a list of defects and reactions is created for each volume element. CellNumber referes to the cell WITHIN A PROCESSOR that the defect is located in.

processorData gives the processor ID#, the total number of processors, the boundaries of the global system as well as the local processor's physical boundaries, and the processor numbers of the neighboring processors (in the 'processor mesh').

mesh contains the information on the local mesh within a processor. Each element of mesh has coord- inates, a processor number, a material number, a length (cubic elements only), the number of neighbors in each direction (for nonuniform meshes), and a list of neighboring elements and their processors.

mesh and processor are NOT pointers, these are arrays of a variable type created during the mesh initialization step.

DiffusionFunction, diffusionSingle, bindingFunction, bindingSingle, and reactionParameters are material classes that are read in from a file and stored in a particular way in order to quickly find reaction rates of all allowed reactions.

8.2 meshreader Module Reference

Module Mesh Reader: creates a processor mesh and volume element mesh using input from file.

Functions/Subroutines

- subroutine readmeshuniform (filename)

  Subroutine read Mesh Uniform - creates processor and mesh files from uniform mesh input.
- subroutine createconnectglobalperiodicuniform (connectivity, numx, numy, numz)

  Subroutine create global connectivity (uniform mesh, periodic boundary conditions)
- subroutine createconnectglobalfreesurfuniform (connectivity, numx, numy, numz)

  Subroutine create global connectivity (uniform mesh, free surfaces in z-directions and periodic boundary conditions in other directions)
- subroutine createconnectlocalperiodicuniform (numx, numy, numz, globalMeshCoord, globalMeshConnect)

  Subroutine create local connectivity (uniform mesh, periodic boundary conditions)
- subroutine createconnectlocalfreesurfuniform (numx, numy, numz, globalMeshCoord, globalMeshConnect)

  Subroutine create local connectivity (uniform mesh, free surfaces in z-direction and periodic boundary conditions in other directions)
- integer function findglobalcell (coord, gCoord)

  Subroutine find Global Cell.
- subroutine readmeshnonuniform (filename)

  Subroutine read Mesh Non Uniform - creates processor and mesh files from non-uniform mesh input.
- subroutine createconnectglobalperiodicnonuniform (Connect, Coord, NumNeighbors, Length, numTotal, bdryCoord)

  Subroutine create global connectivity (non-uniform mesh, periodic boundary conditions)
- subroutine createconnectglobalfreesurfnonuniform (Connect, Coord, NumNeighbors, Length, numTotal, bdryCoord)

Subroutine create global connectivity (non-uniform mesh, free surfaces in the z-direction and periodic boundary conditions in other directions)

- subroutine createconnectlocalnonuniform (globalMeshConnect, globalMeshCoord, localElem, procCoord↩ List)

  Subroutine create local connectivity (non-uniform mesh, either free surfaces or periodic bc's)

- integer function findlocalcell (coord)

  Function find local cell.

- subroutine createproccoordlist (procCoordList, procDivision)

  subroutine create processor coordinates list

- integer function findneighborproc (globalMeshCoord, procCoordList, elem)

  Function find neighboring processor.

8.2.1 Detailed Description

Module Mesh Reader: creates a processor mesh and volume element mesh using input from file.

This module is responsible for reading the mesh from a file (in a specific format) and creating myProc and my↩ Mesh(:), the information in each processor and each mesh, including connectivity and the division of volume elements over processors.

There are two formats of mesh accepted at this point: a uniform cubic mesh, and a non-uniform cubic mesh. The first set of subroutines below is associated with a uniform cubic mesh, and the second set is associated with a non-uniform cubic mesh. The code can use either set of subroutines to create a valid myMesh(:) array, but the input file must match the rules for either uniform or nonuniform mesh input (see rules in input files).

The processors are divided among the global volume in such a way as to minimize the surface area between processors, minimizing the amount of communication necessary between them.

5.28.2014: NOTE: the SRSCD_par program is currently unable to accept meshes in which the processor of a neighboring element in a given direction is not equal to the (global) neighboring processor in that same direction (can be caused by uneven meshing and irregular division of elements)

8.2.2 Function/Subroutine Documentation

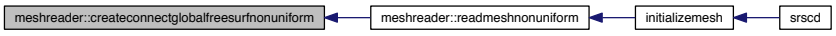
8.2.2.1 subroutine meshreader::createconnectglobalfreesurfnonuniform (integer, dimension(:,:,:), allocatable *Connect,* double precision, dimension(:,:), allocatable *Coord,* integer, dimension(:,:), allocatable *NumNeighbors,* double precision, dimension(:), allocatable *Length,* integer *numTotal,* double precision, dimension(6) *bdryCoord*)

Subroutine create global connectivity (non-uniform mesh, free surfaces in the z-direction and periodic boundary conditions in other directions)

This subroutine creates a global mesh (not dividing between processors) as well as the connectivity between elements for the case of a non-uniform cubic mesh with free surfaces.

Inputs: coordinates list, length list, total number of elements, coordinates of boundary Output: connectivity, number of neighbors for each element

Here is the caller graph for this function:



8.2.2.2 subroutine meshreader::createconnectglobalfreesurfuniform (integer, dimension(numx∗numy∗numz, 6) *connectivity,* integer *numx,* integer *numy,* integer *numz*)

Subroutine create global connectivity (uniform mesh, free surfaces in z-directions and periodic boundary conditions in other directions)

This subroutine creates a global mesh (not dividing between processors) as well as the connectivity between elements for the case of a uniform cubic mesh with free surfaces.

Inputs: numx, numy, numz Output: connectivity

Here is the caller graph for this function:

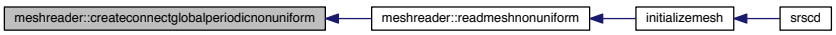
8.2.2.3 subroutine meshreader::createconnectglobalperiodicnonuniform (integer, dimension(:,:,:), allocatable *Connect,* double precision, dimension(:,:), allocatable *Coord,* integer, dimension(:,:), allocatable *NumNeighbors,* double precision, dimension(:), allocatable *Length,* integer *numTotal,* double precision, dimension(6) *bdryCoord*)

Subroutine create global connectivity (non-uniform mesh, periodic boundary conditions)

This subroutine creates a global mesh (not dividing between processors) as well as the connectivity between elements for the case of a non-uniform periodic cubic mesh.

Inputs: coordinates list, length list, total number of elements, coordinates of boundary Output: connectivity, number of neighbors for each element

Here is the caller graph for this function:



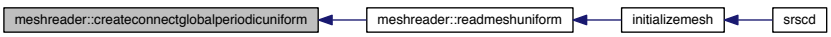
8.2.2.4 subroutine meshreader::createconnectglobalperiodicuniform (integer, dimension(numx∗numy∗numz, 6) *connectivity,* integer *numx,* integer *numy,* integer *numz*)

Subroutine create global connectivity (uniform mesh, periodic boundary conditions)

This subroutine creates a global mesh (not dividing between processors) as well as the connectivity between elements for the case of a uniform cubic mesh.

Inputs: numx, numy, numz Output: connectivity

Here is the caller graph for this function:



8.2.2.5 subroutine meshreader::createconnectlocalfreesurfuniform (integer *numx,* integer *numy,* integer *numz,* double precision, dimension(:,:), allocatable *globalMeshCoord,* integer, dimension(:,:), allocatable *globalMeshConnect*)

Subroutine create local connectivity (uniform mesh, free surfaces in z-direction and periodic boundary conditions in other directions)

This subroutine creates a local mesh (for the local processor only) as well as the connectivity between elements for the case of a uniform cubic mesh with free surfaces. This subroutine has to choose which elements of the global mesh are in the local mesh, and puts those in myMesh along with the element coordinates and their neighbor element ID numbers and processors.

Inputs: numx, numy, numz, global mesh coordinates, global mesh connectivity Output: myMesh

Here is the call graph for this function:

Here is the caller graph for this function:

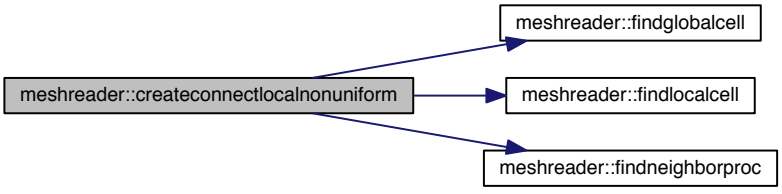
8.2.2.6 subroutine meshreader::createconnectlocalnonuniform (integer, dimension(:,:,:), allocatable *globalMeshConnect,* double precision, dimension(:,:), allocatable *globalMeshCoord,* integer *localElem,* double precision, dimension(:,:), allocatable *procCoordList*)

Subroutine create local connectivity (non-uniform mesh, either free surfaces or periodic bc's)

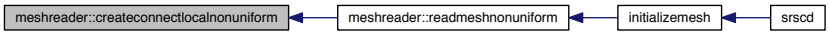
This subroutine creates a local mesh (for the local processor only) as well as the connectivity between elements for the case of a non-uniform cubic mesh. This subroutine has to choose which elements of the global mesh are in the local mesh, and puts those in myMesh along with the element coordinates and their neighbor element ID numbers and processors.

Inputs: global mesh coordinates, global mesh connectivity, list of local elements, list of coordinate boundaries of each processor Output: myMesh

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.2.7 subroutine meshreader::createconnectlocalperiodicuniform (integer *numx,* integer *numy,* integer *numz,* double precision, dimension(:,:), allocatable *globalMeshCoord,* integer, dimension(:,:), allocatable *globalMeshConnect*)

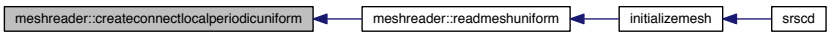
Subroutine create local connectivity (uniform mesh, periodic boundary conditions)

This subroutine creates a local mesh (for the local processor only) as well as the connectivity between elements for the case of a uniform cubic mesh. This subroutine has to choose which elements of the global mesh are in the local mesh, and puts those in myMesh along with the element coordinates and their neighbor element ID numbers and processors.

Inputs: numx, numy, numz Inputs: numx, numy, numz, global mesh coordinates, global mesh connectivity Output: myMesh

Here is the call graph for this function:

Here is the caller graph for this function:



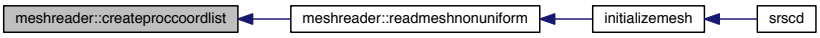
8.2.2.8 subroutine meshreader::createproccoordlist (double precision, dimension(:,:), allocatable *procCoordList,* integer, dimension(3) *procDivision*)

subroutine create processor coordinates list

This subroutine creates a list of the bounds (coordinates) of each processor to be used to identify which processor contains a given volume element (necessary for non-uniform mesh creation)

Inputs: division of processors (see main subroutine) Output: list of boundaries of each processor

Here is the caller graph for this function:



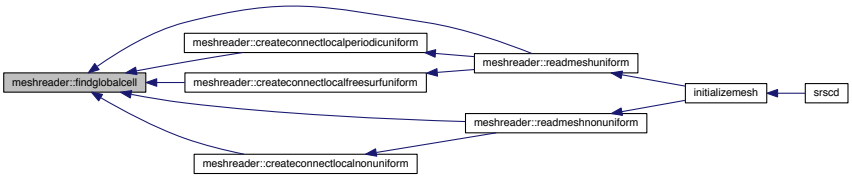
8.2.2.9 integer function meshreader::findglobalcell (double precision, dimension(3) *coord,* double precision, dimension(:,:), allocatable *gCoord*)

Subroutine find Global Cell.

This subroutine searches through a list of global cell coordinates and identifies which global cell ID number the coordinates provided corresponds to

Inputs: coordinates of cell we are searching for, list of global cell coordinates Output: cell ID

Here is the caller graph for this function:



8.2.2.10 integer function meshreader::findlocalcell (double precision, dimension(3) *coord*)

Function find local cell.

Finds the cell ID number in the local mesh using the coordinates of the centroid of the cell

Input: coordinates Output: cell ID number

Here is the caller graph for this function:

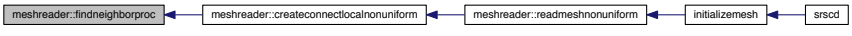
8.2.2.11 integer function meshreader::findneighborproc (double precision, dimension(:,:), allocatable *globalMeshCoord,* double precision, dimension(:,:), allocatable *procCoordList,* integer *elem*)

Function find neighboring processor.

This function finds the processor ID for a volume element that has ID number==elem in the GLOBAL connectivity.

Inputs: list of global mesh coordinates, list of bounds of each processor's domain, element number Output: processor ID for a given volume element

Here is the caller graph for this function:



8.2.2.12 subroutine meshreader::readmeshnonuniform (character∗50 *filename*)

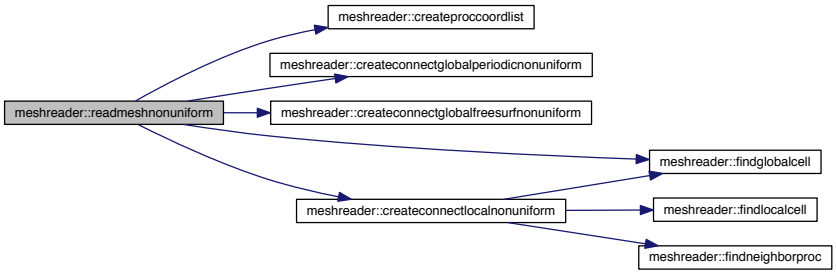
Subroutine read Mesh Non Uniform - creates processor and mesh files from non-uniform mesh input.

This is the main subroutine that controls reading the non-uniform mesh. It reads in the mesh from a file and divides the volume elements between the processors in the parallel simulation. If a division such that each processor has at least one element is not possible, then the subroutine returns an error. It also creates a 'processor mesh', indicating the bounds of each processor. NOTE: this subroutine is for non-uniform meshes only (and these have not been implemented in the rest of the program)

Input: filename of mesh file

Outputs: myProc (processors with meshes), myMesh, and myBoundary

Here is the call graph for this function:



Here is the caller graph for this function:

8.2.2.13 subroutine meshreader::readmeshuniform (character∗50 *filename*)

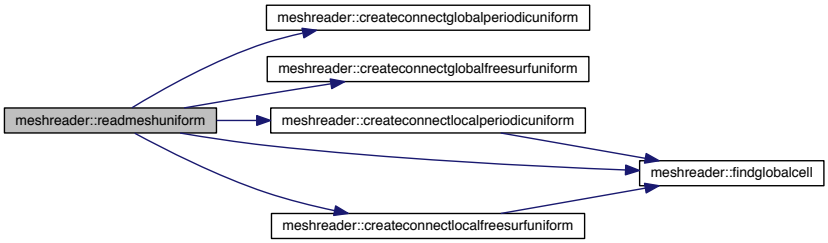
Subroutine read Mesh Uniform - creates processor and mesh files from uniform mesh input.

This is the main subroutine that controls reading the uniform mesh. It reads in the mesh from a file and divides the volume elements between the processors in the parallel simulation. If a division such that each processor has at least one element is not possible, then the subroutine returns an error. It also creates a 'processor mesh', indicating the bounds of each processor. NOTE: this subroutine is for uniform meshes only, a different subroutine has been created to read in non-uniform meshes (and these have not been implemented in the rest of the program)

Input: filename of mesh file

Outputs: myProc (processors with meshes), myMesh, and myBoundary

Here is the call graph for this function:



Here is the caller graph for this function:

8.3 mod_srscd_constants Module Reference

Module mod_SRSCD_constants (list of globally shared variables and pointers)

Variables

- type(processordata) myproc

  Contains processor information (id, neighbors)
- type(mesh), dimension(:), allocatable mymesh

  Contains (local) mesh information.
- type(boundarymesh), dimension(:,:), allocatable myboundary

  Boundary elements (direction, element #)
- integer numcells

  Number of cells in local mesh.
- type(reaction), dimension(:), pointer reactionlist

  List of reactions in local (coarse) mesh.
- type(defect), dimension(:), pointer defectlist

  List of defects in local (coarse) mesh.
- double precision totalrate

  Total reaction rate in this processor.
- double precision, dimension(:), allocatable totalratevol

  Total reaction rate in each volume element.
- double precision maxrate

  Max reaction rate in all processors.
- type(cascadeevent), pointer cascadelist

  List of cascades (read from file) that can be implanted.

- type(cascade), pointer activecascades

  List of fine meshes that are active due to recent cascade implantation (Contains defect lists and reaction lists)
- integer numcascades

  number of cascades in the cascade input file (used to choose cascades to input in simulation)
- integer numcellscascade

  number of volume elements within a cascade (fine) mesh
- integer numxcascade

  number of elements in cascade x-direction
- integer numycascade

  number of elements in cascade y-direction
- integer numzcascade

  number of elements in cascade z-direction
- integer, dimension(:,:), allocatable cascadeconnectivity

  connectivity matrix for cascade meshes (same for all fine meshes)
- double precision finelength

  length of a cascade volume element (nm)
- double precision cascadeelementvol

  volume of a cascade element (nm^ 3)
- integer nummaterials

  Number of material types (eg. copper, niobium or bulk, grain boundary)
- integer numspecies

  Number of chemical species (typically set to 4: He, V, SIA_glissile, SIA_sessile)
- type(diffusionfunction), dimension(:,:), allocatable difffunc

  Parameters for functional forms of diffusion rates for defects.
- type(diffusionsingle), dimension(:,:), allocatable diffsingle

  Parameters for diffusion of single defects.
- type(bindingsingle), dimension(:,:), allocatable bindsingle

  Parameters for binding of single defects.
- type(bindingfunction), dimension(:,:), allocatable bindfunc

  Parameters for functional forms of binding energies for defects.
- type(reactionparameters), dimension(:,:), allocatable dissocreactions

  List of allowed dissociation reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable diffreactions

  List of allowed diffusion reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable sinkreactions

  List of allowed sink reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable impurityreactions

  List of allowed impurity reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable clusterreactions

  List of allowed clustering reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable implantreactions

  List of allowed implantation reactions (and ref. to functional form of reaction rate)
- integer, dimension(:), allocatable numsinglediff

  Number of single defect diffusion rates in input file.
- integer, dimension(:), allocatable numfuncdiff

  Number of functional forms for diffusion rates in input files.
- integer, dimension(:), allocatable numsinglebind

  Number of single defect binding energies in input file.
- integer, dimension(:), allocatable numfuncbind

  Number of functional forms for binding energies in input files.
- integer, dimension(:), allocatable numdissocreac

Number of dissociation reactions in input file.

- integer, dimension(:), allocatable numdiffreac

  Number of diffusion reactions in input file.
- integer, dimension(:), allocatable numsinkreac

  Number of sink reactions in input file.
- integer, dimension(:), allocatable numimpurityreac

  Number of impurity reactions in input file.
- integer, dimension(:), allocatable numclusterreac

  Number of clustering reactions in input file.
- integer, dimension(:), allocatable numimplantreac

  Number of implantation reactions in input file (cascade, Frenkel pair, He currently implemented)
- double precision, parameter kboltzmann =8.6173324d-5

  Boltzmann's constant (eV/K)
- double precision, parameter pi =3.141592653589793

  Pi.
- double precision, parameter zint = 1.2

  Constant representing preference for clustering of interstitials by interstitial clusters (increases clustering cross-section)
- double precision, parameter reactionradius =.5065d0

  Material parameter used for reaction distances (impacts reaction rates)
- double precision temperature

  Temperature (K)
- double precision tempstore

  Temperature read in (K) - used when temp. changes several times during a simulation.
- double precision hedparatio

  Helium to dpa ratio (atoms per atom)
- double precision dparate

  DPA rate in dpa/s.
- double precision atomsize

  atomic volume (nm^3)
- double precision dpa

  DPA tracker (not a parameter)
- double precision defectdensity

  total density of defects (?), not sure if this is needed
- double precision dislocationdensity

  density of dislocations (sinks for point defects)
- double precision impuritydensity

  denstiy of impurity atoms (traps for SIA loops)
- double precision totaldpa

  total DPA in simulation
- double precision burgers

  magnitude of burgers vector, equal to lattice constant
- double precision numdisplacedatoms

  number of atoms displaced per cascade, read from cascade file
- double precision meanfreepath

  mean free path before a defect is absorbed by a grain boundary (AKA avg. grain size)
- double precision cascadereactionlimit

  Total reaction rate in a cascade cell to consider it annealed and release cascade back to coarse mesh (s^-1)
- double precision cascadevolume

  Volume of cascade (used for cascade mixing probability)
- double precision totalvolume

Volume of single processor's mesh (nm^ 3)

- double precision systemvol

  Volume of all processors (global), not including grain boundary elements (nm^ 3)
- double precision alpha_v

  Grain boundary sink efficiency for vacancies.
- double precision alpha_i

  Grain boundary sink efficiency for interstitials.
- double precision conc_v

  Concentration of vacancies found by GB model (used to fit alpha_v)
- double precision conc_i

  Concentration of interstitials found by GB model (used to fit alpha_i)
- double precision annealtime

  Temperature of anneal stage (K)
- double precision annealtemp

  Amount of time for anneal (s)
- double precision annealsteps

  Number of annealing steps.
- double precision annealtempinc

  Temperature increment at each annealing step (additive or multipliciative)
- character ∗20 annealtype

  ('mult' or 'add') toggles additive or multiplicative anneal steps
- logical annealidentify

  (.TRUE. if in annealing phase, .FALSE. otherwise) used to determine how to reset reaction rates (should we include implantation or not)
- integer numsims

  Number of times to repeat simulation.
- integer max3dint

  largest SIA size that can diffuse in 3D as spherical cluster
- integer siapinmin

  Smallest size of SIA that can pin at HeV clusters.
- integer numgrains

  Number of grains inside polycrystal (default 1)
- character ∗20 implanttype

  (Frenkel pairs or cascades), used to determine the type of damage in the simulation
- character ∗20 grainboundarytoggle

  Used to determine whether or not we are using grain boundaries to remove defects from simulation.
- character ∗20 hesiatoggle

  Toggles whether or not we allow HeSIA clusters to form ('yes' or 'no')
- character ∗20 siapintoggle

  Toggles whether or not we allow HeV clusters to pin SIA clusters (without annihilating V+SIA)
- character ∗20 meshingtype

  (adaptive or nonAdaptive), used to determine whether we are simulating cascade implantation with adaptive meshing
- character ∗20 implantscheme

  (MonteCarlo or explicit), used to determine if cascades are implanted through Monte Carlo algorithm or explicitly
- character ∗20 implantdist

  (Uniform or NonUniform), used to determine if defects are implanted uniformly or if DPA rate / He implant rate are given for each volume element
- character ∗20 polycrystal

  (yes or no), used to identify whether or not we have multiple grains in our crystal
- character ∗20 vtktoggle

  (yes or no), used to toggle whether we want vtk output at each time increment (log scale)

- character ∗20 outputdebug

  (yes or no), used to toggle whether we want to output a debug restart file at each time increment
- character ∗20 singleelemkmc

  (yes or no), used to toggle whether we are making one kMC choice per volume element or one kMC choice for the whole processors
- character ∗20 sinkeffsearch

  (yes or no), used to toggle search for effective sink efficiency
- character ∗20 strainfield

  (yes or no), used to toggle whether we are simulating diffusion in a strain field
- character ∗20 postprtoggle

  (yes or no), used to toggle whether we output the postpr.out data file
- character ∗20 totdattoggle

  (yes or no), used to toggle whether we output the totdat.out data file
- character ∗20 rawdattoggle

  (yes or no), used to toggle whether we output the rawdat.out data file
- character ∗20 xyztoggle

  (yes or no), used to toggle whether we output an .xyz data file (for visualization)
- character ∗20 profiletoggle

  (yes or no), used to toggle whether we output a DefectProfile.out data file
- double precision omega

  Geometric constant for 3D spherical clustering (see Dunn et al. JNM 2013)
- double precision omega2d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision omega1d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision omegastar

  Geometric constant for 3D spherical clustering (see Dunn et al. JNM 2013)
- double precision omegastar1d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision omegacircle1d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision recombinationcoeff

  Geometric constant for Frenkel pair recombination (see Dunn et al. JNM 2013)
- integer ierr

  used for initializing and finalizing MPI
- integer, parameter master =0

  Define the master node as ID=0.
- integer, parameter maxbuffersize =50

  Used to define the max size of a send/recieve buffer.
- integer numimplantevents

  Postprocessing: number of Frenkel pairs / cascades (local)
- integer numheimplantevents

  Postprocessing: number of He implantation events (local)
- integer totalimplantevents

  Postprocessing: number of implant events across all processors.
- integer numheimplanttotal

  Postprocessing: number of He implant events across all processors.
- integer numannihilate

  Postprocessing: number of annihilation reactions carried out.
- integer numtrapv

  Postprocessing: number of vacancies trapped on grain boundary.

- integer numtrapsia

  Postprocessing: number of SIAs trapped on grain boundary.
- integer numemitv

  Postprocessing: number of vacancies emitted from grain boundary.
- integer numemitsia

  Postprocessing: number of SIAs emitted from grain boundary.
- integer numimplanteventsreset

  For creating restart file (debugging tool, see example): number of cascades/Frenkel pairs.
- integer numheimplanteventsreset

  For creating restart file (debugging tool, see example): number of helium implantaion events.
- double precision elapsedtimereset

  For creating restart file (debugging tool, see example): elapsed time.
- character ∗20 debugtoggle

  ('yes' or 'no') input parameter indicating whether we are restarting from a file
- character ∗50 restartfilename

  Name of restart file.
- integer numimplantdatapoints

  For non-uniform implantation, number of input data points through-thickness.
- double precision, dimension(:,:), allocatable implantratedata

  Data containing implantation rates as a function of depth (for non-uniform implantation)
- integer numdipole

  Number of dipole tensors that are read in from a file.
- type(dipoletensor), dimension(:), allocatable dipolestore

  Array of dipole tensors for associated defects, size numDipole.
- character ∗50 strainfilename

  File name of strain field intput data.
- character ∗50 dipolefilename

  File name of dipole tensor input data.

8.3.1 Detailed Description

Module mod_SRSCD_constants (list of globally shared variables and pointers)

This module contains the list of all globally shared variables. This includes: 1) Processor and mesh information (code backbone) 2) Reaction lists, defect lists, and cascade lists, both in coarse and fine meshes 3) Global reaction rates 4) Material input information, in the form of derived types (diffusion and binding energies, etc) 5) Universal constants 6) Simulation parameters read in from parameters.txt 7) Other miscellaneous variables used for MPI, debugging, or postprocessing

8.3.2 Variable Documentation

8.3.2.1 type(cascade), pointer mod_srscd_constants::activecascades

List of fine meshes that are active due to recent cascade implantation (Contains defect lists and reaction lists)

8.3.2.2 double precision mod_srscd_constants::alpha_i

Grain boundary sink efficiency for interstitials.

8.3.2.3 double precision mod_srscd_constants::alpha_v

Grain boundary sink efficiency for vacancies.

8.3.2.4 logical mod_srscd_constants::annealidentify

(.TRUE. if in annealing phase, .FALSE. otherwise) used to determine how to reset reaction rates (should we include implantation or not)

8.3.2.5 double precision mod_srscd_constants::annealsteps

Number of annealing steps.

8.3.2.6 double precision mod_srscd_constants::annealtemp

Amount of time for anneal (s)

8.3.2.7 double precision mod_srscd_constants::annealtempinc

Temperature increment at each annealing step (additive or multipliciative)

8.3.2.8 double precision mod_srscd_constants::annealtime

Temperature of anneal stage (K)

8.3.2.9 character∗20 mod_srscd_constants::annealtype

('mult' or 'add') toggles additive or multiplicative anneal steps

8.3.2.10 double precision mod_srscd_constants::atomsize

atomic volume (nm^3)

8.3.2.11 type(bindingfunction), dimension(:,:), allocatable mod_srscd_constants::bindfunc

Parameters for functional forms of binding energies for defects.

8.3.2.12 type(bindingsingle), dimension(:,:), allocatable mod_srscd_constants::bindsingle

Parameters for binding of single defects.

8.3.2.13 double precision mod_srscd_constants::burgers

magnitude of burgers vector, equal to lattice constant

8.3.2.14 integer, dimension(:,:), allocatable mod_srscd_constants::cascadeconnectivity

connectivity matrix for cascade meshes (same for all fine meshes)

8.3.2.15 double precision mod_srscd_constants::cascadeelementvol

volume of a cascade element (nm^3)

8.3.2.16 type(cascadeevent), pointer mod_srscd_constants::cascadelist

List of cascades (read from file) that can be implanted.

8.3.2.17 double precision mod_srscd_constants::cascadereactionlimit

Total reaction rate in a cascade cell to consider it annealed and release cascade back to coarse mesh (s^-1)

8.3.2.18 double precision mod_srscd_constants::cascadevolume

Volume of cascade (used for cascade mixing probability)

8.3.2.19 type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::clusterreactions

List of allowed clustering reactions (and ref. to functional form of reaction rate)

8.3.2.20 double precision mod_srscd_constants::conc_i

Concentration of interstitials found by GB model (used to fit alpha_i)

8.3.2.21 double precision mod_srscd_constants::conc_v

Concentration of vacancies found by GB model (used to fit alpha_v)

8.3.2.22 character*20 mod_srscd_constants::debugtoggle

('yes' or 'no') input parameter indicating whether we are restarting from a file

8.3.2.23 double precision mod_srscd_constants::defectdensity

total density of defects (?), not sure if this is needed

8.3.2.24 type(defect), dimension(:), pointer mod_srscd_constants::defectlist

List of defects in local (coarse) mesh.

8.3.2.25 type(diffusionfunction), dimension(:,:), allocatable mod_srscd_constants::difffunc

Parameters for functional forms of diffusion rates for defects.

8.3.2.26 type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::diffreactions

List of allowed diffusion reactions (and ref. to functional form of reaction rate)

8.3.2.27 type(diffusionsingle), dimension(:,:), allocatable mod_srscd_constants::diffsingle

Parameters for diffusion of single defects.

8.3.2.28 character*50 mod_srscd_constants::dipolefilename

File name of dipole tensor input data.

8.3.2.29 type(dipoletensor), dimension(:), allocatable mod_srscd_constants::dipolestore

Array of dipole tensors for associated defects, size numDipole.

8.3.2.30 double precision mod_srscd_constants::dislocationdensity

density of dislocations (sinks for point defects)

8.3.2.31 type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::dissocreactions

List of allowed dissociation reactions (and ref. to functional form of reaction rate)

8.3.2.32 double precision mod_srscd_constants::dpa

DPA tracker (not a parameter)

8.3.2.33 double precision mod_srscd_constants::dparate

DPA rate in dpa/s.

8.3.2.34 double precision mod_srscd_constants::elapsedtimereset

For creating restart file (debugging tool, see example): elapsed time.

8.3.2.35 double precision mod_srscd_constants::finelength

length of a cascade volume element (nm)

8.3.2.36 character*20 mod_srscd_constants::grainboundarytoggle

Used to determine whether or not we are using grain boundaries to remove defects from simulation.

8.3.2.37 double precision mod_srscd_constants::hedparatio

Helium to dpa ratio (atoms per atom)

8.3.2.38 character*20 mod_srscd_constants::hesiatoggle

Toggles whether or not we allow HeSIA clusters to form ('yes' or 'no')

8.3.2.39 integer mod_srscd_constants::ierr

used for initializing and finalizing MPI

8.3.2.40 character∗20 mod_srscd_constants::implantdist

(Uniform or NonUniform), used to determine if defects are implanted uniformly or if DPA rate / He implant rate are given for each volume element

8.3.2.41 double precision, dimension(:,:), allocatable mod_srscd_constants::implantratedata

Data containing implantation rates as a function of depth (for non-uniform implantation)

8.3.2.42 type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::implantreactions

List of allowed implantation reactions (and ref. to functional form of reaction rate)

8.3.2.43 character∗20 mod_srscd_constants::implantscheme

(MonteCarlo or explicit), used to determine if cascades are implanted through Monte Carlo algorithm or explicitly

8.3.2.44 character∗20 mod_srscd_constants::implanttype

(Frenkel pairs or cascades), used to determine the type of damage in the simulation

8.3.2.45 double precision mod_srscd_constants::impuritydensity

denstiy of impurity atoms (traps for SIA loops)

8.3.2.46 type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::impurityreactions

List of allowed impurity reactions (and ref. to functional form of reaction rate)

8.3.2.47 double precision, parameter mod_srscd_constants::kboltzmann =8.6173324d-5

Boltzmann's constant (eV/K)

8.3.2.48 integer, parameter mod_srscd_constants::master =0

Define the master node as ID=0.

8.3.2.49 integer mod_srscd_constants::max3dint

largest SIA size that can diffuse in 3D as spherical cluster

8.3.2.50 integer, parameter mod_srscd_constants::maxbuffersize =50

Used to define the max size of a send/recieve buffer.

8.3.2.51 double precision mod_srscd_constants::maxrate

Max reaction rate in all processors.

8.3.2.52 double precision mod_srscd_constants::meanfreepath

mean free path before a defect is absorbed by a grain boundary (AKA avg. grain size)

8.3.2.53 character∗20 mod_srscd_constants::meshingtype

(adaptive or nonAdaptive), used to determine whether we are simulating cascade implantation with adaptive meshing

8.3.2.54 type(boundarymesh), dimension(:,:), allocatable mod_srscd_constants::myboundary

Boundary elements (direction, element #)

8.3.2.55 type(mesh), dimension(:), allocatable mod_srscd_constants::mymesh

Contains (local) mesh information.

8.3.2.56 type(processordata) mod_srscd_constants::myproc

Contains processor information (id, neighbors)

8.3.2.57 integer mod_srscd_constants::numannihilate

Postprocessing: number of annihilation reactions carried out.

8.3.2.58 integer mod_srscd_constants::numcascades

number of cascades in the cascade input file (used to choose cascades to input in simulation)

8.3.2.59 integer mod_srscd_constants::numcells

Number of cells in local mesh.

8.3.2.60 integer mod_srscd_constants::numcellscascade

number of volume elements within a cascade (fine) mesh

8.3.2.61 integer, dimension(:), allocatable mod_srscd_constants::numclusterreac

Number of clustering reactions in input file.

8.3.2.62 integer, dimension(:), allocatable mod_srscd_constants::numdiffreac

Number of diffusion reactions in input file.

8.3.2.63 integer mod_srscd_constants::numdipole

Number of dipole tensors that are read in from a file.

8.3.2.64 double precision mod_srscd_constants::numdisplacedatoms

number of atoms displaced per cascade, read from cascade file

8.3.2.65 integer, dimension(:), allocatable mod_srscd_constants::numdissocreac

Number of dissociation reactions in input file.

8.3.2.66 integer mod_srscd_constants::numemitsia

Postprocessing: number of SIAs emitted from grain boundary.

8.3.2.67 integer mod_srscd_constants::numemitv

Postprocessing: number of vacancies emitted from grain boundary.

8.3.2.68 integer, dimension(:), allocatable mod_srscd_constants::numfuncbind

Number of functional forms for binding energies in input files.

8.3.2.69 integer, dimension(:), allocatable mod_srscd_constants::numfuncdiff

Number of functional forms for diffusion rates in input files.

8.3.2.70 integer mod_srscd_constants::numgrains

Number of grains inside polycrystal (default 1)

8.3.2.71 integer mod_srscd_constants::numheimplantevents

Postprocessing: number of He implantation events (local)

8.3.2.72 integer mod_srscd_constants::numheimplanteventsreset

For creating restart file (debugging tool, see example): number of helium implantaion events.

8.3.2.73 integer mod_srscd_constants::numheimplanttotal

Postprocessing: number of He implant events across all processors.

8.3.2.74 integer mod_srscd_constants::numimplantdatapoints

For non-uniform implantation, number of input data points through-thickness.

8.3.2.75 integer mod_srscd_constants::numimplantevents

Postprocessing: number of Frenkel pairs / cascades (local)

8.3.2.76 integer mod_srscd_constants::numimplanteventsreset

For creating restart file (debugging tool, see example): number of cascades/Frenkel pairs.

8.3.2.77 integer, dimension(:), allocatable mod_srscd_constants::numimplantreac

Number of implantation reactions in input file (cascade, Frenkel pair, He currently implemented)

8.3.2.78 integer, dimension(:), allocatable mod_srscd_constants::numimpurityreac

Number of impurity reactions in input file.

8.3.2.79 integer mod_srscd_constants::nummaterials

Number of material types (eg. copper, niobium or bulk, grain boundary)

8.3.2.80 integer mod_srscd_constants::numsims

Number of times to repeat simulation.

8.3.2.81 integer, dimension(:), allocatable mod_srscd_constants::numsinglebind

Number of single defect binding energies in input file.

8.3.2.82 integer, dimension(:), allocatable mod_srscd_constants::numsinglediff

Number of single defect diffusion rates in input file.

8.3.2.83 integer, dimension(:), allocatable mod_srscd_constants::numsinkreac

Number of sink reactions in input file.

8.3.2.84 integer mod_srscd_constants::numspecies

Number of chemical species (typically set to 4: He, V, SIA_glissile, SIA_sessile)

8.3.2.85 integer mod_srscd_constants::numtrapsia

Postprocessing: number of SIAs trapped on grain boundary.

8.3.2.86 integer mod_srscd_constants::numtrapv

Postprocessing: number of vacancies trapped on grain boundary.

8.3.2.87 integer mod_srscd_constants::numxcascade

number of elements in cascade x-direction

8.3.2.88 integer mod_srscd_constants::numycascade

number of elements in cascade y-direction

8.3.2.89 integer mod_srscd_constants::numzcascade

number of elements in cascade z-direction

8.3.2.90 double precision mod_srscd_constants::omega

Geometric constant for 3D spherical clustering (see Dunn et al. JNM 2013)

8.3.2.91 double precision mod_srscd_constants::omega1d

Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)

8.3.2.92 double precision mod_srscd_constants::omega2d

Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)

8.3.2.93 double precision mod_srscd_constants::omegacircle1d

Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)

8.3.2.94 double precision mod_srscd_constants::omegastar

Geometric constant for 3D spherical clustering (see Dunn et al. JNM 2013)

8.3.2.95 double precision mod_srscd_constants::omegastar1d

Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)

8.3.2.96 character∗20 mod_srscd_constants::outputdebug

(yes or no), used to toggle whether we want to output a debug restart file at each time increment

8.3.2.97 double precision, parameter mod_srscd_constants::pi =3.141592653589793

Pi.

8.3.2.98 character∗20 mod_srscd_constants::polycrystal

(yes or no), used to identify whether or not we have multiple grains in our crystal

8.3.2.99 character∗20 mod_srscd_constants::postprtoggle

(yes or no), used to toggle whether we output the postpr.out data file

8.3.2.100 character∗20 mod_srscd_constants::profiletoggle

(yes or no), used to toggle whether we output a DefectProfile.out data file

8.3.2.101 character∗20 mod_srscd_constants::rawdattoggle

(yes or no), used to toggle whether we output the rawdat.out data file

8.3.2.102 type(reaction), dimension(:), pointer mod_srscd_constants::reactionlist

List of reactions in local (coarse) mesh.

8.3.2.103 double precision, parameter mod_srscd_constants::reactionradius =.5065d0

Material parameter used for reaction distances (impacts reaction rates)

8.3.2.104 double precision mod_srscd_constants::recombinationcoeff

Geometric constant for Frenkel pair recombination (see Dunn et al. JNM 2013)

8.3.2.105 character∗50 mod_srscd_constants::restartfilename

Name of restart file.

8.3.2.106 integer mod_srscd_constants::siapinmin

Smallest size of SIA that can pin at HeV clusters.

8.3.2.107 character∗20 mod_srscd_constants::siapintoggle

Toggles whether or not we allow HeV clusters to pin SIA clusters (without annihilating V+SIA)

8.3.2.108 character∗20 mod_srscd_constants::singleelemkmc

(yes or no), used to toggle whether we are making one kMC choice per volume element or one kMC choice for the whole processors

8.3.2.109 character∗20 mod_srscd_constants::sinkeffsearch

(yes or no), used to toggle search for effective sink efficiency

8.3.2.110 type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::sinkreactions

List of allowed sink reactions (and ref. to functional form of reaction rate)

8.3.2.111 character∗20 mod_srscd_constants::strainfield

(yes or no), used to toggle whether we are simulating diffusion in a strain field

8.3.2.112 character∗50 mod_srscd_constants::strainfilename

File name of strain field intput data.

8.3.2.113 double precision mod_srscd_constants::systemvol

Volume of all processors (global), not including grain boundary elements (nm^3)

8.3.2.114 double precision mod_srscd_constants::temperature

Temperature (K)

8.3.2.115 double precision mod_srscd_constants::tempstore

Temperature read in (K) - used when temp. changes several times during a simulation.

8.3.2.116 double precision mod_srscd_constants::totaldpa

total DPA in simulation

8.3.2.117 integer mod_srscd_constants::totalimplantevents

Postprocessing: number of implant events across all processors.

8.3.2.118 double precision mod_srscd_constants::totalrate

Total reaction rate in this processor.

8.3.2.119 double precision, dimension(:), allocatable mod_srscd_constants::totalratevol

Total reaction rate in each volume element.

8.3.2.120 double precision mod_srscd_constants::totalvolume

Volume of single processor's mesh (nm^3)

8.3.2.121 character∗20 mod_srscd_constants::totdattoggle

(yes or no), used to toggle whether we output the totdat.out data file

8.3.2.122 character∗20 mod_srscd_constants::vtktoggle

(yes or no), used to toggle whether we want vtk output at each time increment (log scale)

8.3.2.123 character∗20 mod_srscd_constants::xyztoggle

(yes or no), used to toggle whether we output an .xyz data file (for visualization)

8.3.2.124 double precision, parameter mod_srscd_constants::zint = 1.2

Constant representing preference for clustering of interstitials by interstitial clusters (increases clustering cross-section)

8.4 randdp Module Reference

Module randdp: double precision random number generation.

Functions/Subroutines

- subroutine sdprnd (iseed)

  Subroutine sdprnd - seeds random number generator in this processor using integer input.

- real(dp) function dprand ()

  Function dprand - generates a double precision random number between 0 and 1.

Variables

- integer, parameter, public dp = SELECTED_REAL_KIND(15, 60)
- real(dp), dimension(101), save, public poly
- real(dp), save, public other
- real(dp), save, public offset
- integer, save, public index

8.4.1 Detailed Description

Module randdp: double precision random number generation.

Nick Maclaren's double precision random number generator. This version, which is compatible with Lahey's ELF90 compiler, is by Alan Miller (alan @ vic.cmis.csiro.au, www.vic.cmis.csiro.au/∼alan)

Latest revision - 18 December 1997

Copyright (C) 1992 N.M. Maclaren Copyright (C) 1992 The University of Cambridge

This software may be reproduced and used freely, provided that all users of it agree that the copyright holders are not liable for any damage or injury caused by use of this software and that this condition is passed onto all subsequent recipients of the software, whether modified or not.

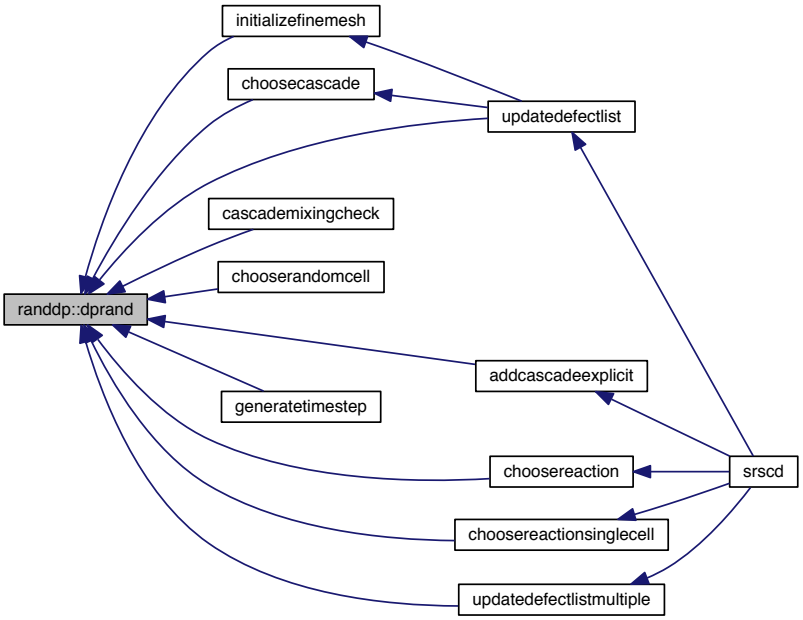
8.4.2 Function/Subroutine Documentation

8.4.2.1 real (dp) function randdp::dprand ( )

Function dprand - generates a double precision random number between 0 and 1.

Here is the call graph for this function:

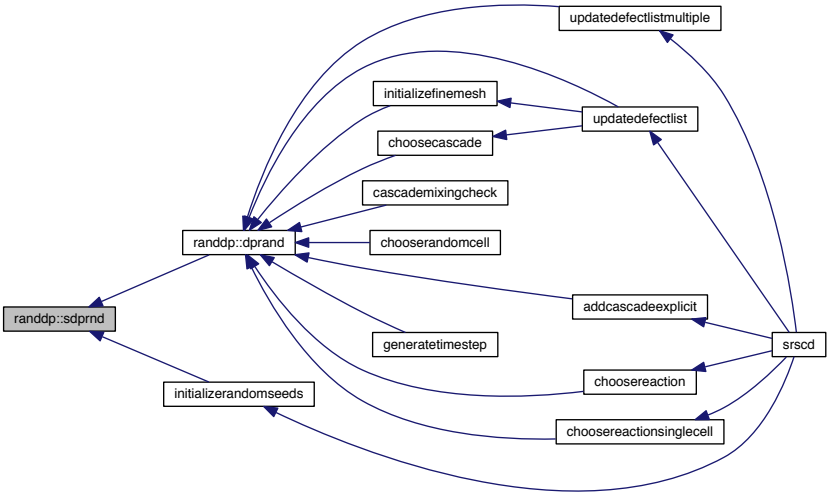
Here is the caller graph for this function:



8.4.2.2 subroutine randdp::sdprnd (integer, intent(in) *iseed*)

Subroutine sdprnd - seeds random number generator in this processor using integer input.

Here is the caller graph for this function:



8.4.3 Variable Documentation

8.4.3.1 integer, parameter, public randdp::dp = SELECTED_REAL_KIND(15, 60)

8.4.3.2 integer, save, public randdp::index

8.4.3.3 real (dp), save, public randdp::offset

8.4.3.4 real (dp), save, public randdp::other

8.4.3.5 real (dp), dimension(101), save, public randdp::poly

8.5 reactionrates Module Reference

Module: ReactionRates - adds reactions and calculates rates for the various reaction types in the system.

Functions/Subroutines

- subroutine addsingledefectreactions (cell, defectType)

  Subroutine add single defect reactions - adds reactions to a reaction list that require only a single reactant to be carried out. Examples: dissociation, trapping, sinks. Diffusion reactions are not included in this subroutine.
- subroutine addsingledefectreactionsfine (cascadeID, cell, defectType)

  Subroutine add single defect reactions fine - adds reactions to a reaction list inside a cascade (fine mesh) that require only a single reactant to be carried out. Examples: dissociation, trapping, sinks. Diffusion reactions are not included in this subroutine.
- subroutine addmultidefectreactions (cell, defectType1, defectType2)

  Subroutine add multi defect reactions - adds reactions to a reaction list that require multiple (only using 2 currently) defects to be carried out. This refers mainly to clustering reactions or pinning reactions.
- subroutine addmultidefectreactionsfine (cascadeID, cell, defectType1, defectType2)

  Subroutine add multi defect reactions (fine mesh) - adds reactions to a reaction list inside a cascade that require multiple (only using 2 currently) defects to be carried out. This refers mainly to clustering reactions or pinning reactions.
- subroutine adddiffusionreactions (cell1, cell2, proc1, proc2, dir, defectType)

Subroutine add diffusion reactions - adds reactions to a reaction list representing diffusion between volume elements.

- subroutine adddiffusioncoarsetofine (cell, proc, CascadeCurrent, defectType)

  Subroutine add diffusion reactions from the coarse mesh to a fine mesh element (cascade) - adds reactions to a reaction list representing diffusion between volume elements.
- subroutine adddiffusionreactionsfine (cascadeID, cell1, cell2, proc1, proc2, dir, defectType)

  Subroutine add diffusion reactions (fine mesh) - adds reactions to a reaction list representing diffusion between volume elements inside a cascade mesh.
- double precision function findreactionrate (cell, reactionParameter)

  Function find reaction rate - finds reaction rate for implantation reaction (He, Frenkel pairs, cascades).
- double precision function findreactionratefine (cell, reactionParameter)

  Function find reaction rate fine - finds reaction rate for implantation reaction (Helium or Frenkel pairs only) in a fine mesh. Cascades cannot be implanted inside other cascades.
- double precision function findreactionrateimpurity (defectType, cell, reactionParameter)

  Function find reaction rate impurity - finds reaction rate for trapping of SIA loops by impurities (Carbon).
- double precision function findreactionrateimpurityfine (CascadeCurrent, defectType, cell, reactionParameter)

  Function find reaction rate impurity fine - finds reaction rate for trapping of SIA loops by impurities (Carbon) inside a fine mesh (Cascade).
- double precision function findreactionratedissoc (defectType, products, cell, reactionParameter)

  Function find reaction rate dissociation - finds reaction rate for point defects to dissociate from clusters.
- double precision function findreactionratedissocfine (CascadeCurrent, defectType, products, cell, reaction↩ Parameter)

  Function find reaction rate dissociation - finds reaction rate for point defects to dissociate from clusters in the fine mesh (cascade)
- double precision function findreactionratesink (defectType, cell, reactionParameter)

  Function find reaction rate sink - finds reaction rate for defects to get absorbed at sinks (typically matrix dislocations)
- double precision function findreactionratesinkfine (CascadeCurrent, defectType, cell, reactionParameter)

  Function find reaction rate sink fine - finds reaction rate for defects to get absorbed at sinks (typically matrix dislocations) in the fine mesh (cascade)
- double precision function findreactionratemultiple (defectType1, defectType2, cell, reactionParameter)

  Function find reaction rate multiple - finds reaction rate for defect clustering.
- double precision function findreactionratemultiplefine (CascadeCurrent, defectType1, defectType2, cell, reactionParameter)

  Function find reaction rate multiple fine - finds reaction rate for defect clustering in the fine mesh (Cascade)
- double precision function findreactionratediff (defectType, cell1, proc1, cell2, proc2, dir, reactionParameter)

  Function find reaction rate diffusion - finds reaction rate for defect diffusion between elements.
- double precision function findreactionratecoarsetofine (defectType, cell, proc, numDefectsFine, reaction↩ Parameter)

  Function find reaction rate diffusion coarse to fine - finds reaction rate for defect diffusion between a coarse mesh element and a fine (cascade) mesh.
- double precision function findreactionratedifffine (CascadeCurrent, defectType, cell1, proc1, cell2, proc2, dir, reactionParameter)

  Function find reaction rate diffusion fine - finds reaction rate for defect diffusion between elements in the fine mesh (inside a cascade)
- subroutine findreactioninlist (reactionCurrent, reactionPrev, cell, reactants, products, numReactants, num↩ Products)

  Subroutine find Reaction In List.
- subroutine findreactioninlistdiff (reactionCurrent, reactionPrev, reactants, cell1, cell2, proc1, proc2)

  Subroutine find Reaction In List (diffusion)
- subroutine findreactioninlistmultiple (reactionCurrent, reactionPrev, cell, reactants, products, numReactants, numProducts)

  Subroutine find Reaction In List Multiple (clustering)
- subroutine defectcombinationrules (products, defectTemp)

  defectCombinationRules (subroutine)

- subroutine checkreactionlegality (numProducts, products, isLegal)

  Subroutine check Reaction Legality.

- double precision function finddparatelocal (zCoord)

  Function find DPA Rate Local (zCoord)

- double precision function findheimplantratelocal (zCoord)

  Function find Helium Implantation Rate Local (zCoord)

8.5.1 Detailed Description

Module: ReactionRates - adds reactions and calculates rates for the various reaction types in the system.

This module contains hard-coded information. It contains the allowed reactions (as read in from the input file) and thier rates.

This module also contains addSingleDefectReactions, addMultiDefectReactions, and addDiffusionReactions. These subroutines will, given (a) defect type(s) and a cell number, calculate the reaction rate of all single/multi-defect/diffusion reactions possible in that element and add those rates to the reaction list in that cell. They shold also delete the reactions that have new reaction rates of 0 in the cell (reactions that are no longer possible after a reaction has occurred).

8.5.2 Function/Subroutine Documentation

8.5.2.1 subroutine reactionrates::adddiffusioncoarsetofine (integer *cell,* integer *proc,* type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType*)

Subroutine add diffusion reactions from the coarse mesh to a fine mesh element (cascade) - adds reactions to a reaction list representing diffusion between volume elements.

Inputs: coarse mesh cell number, processor, cascade ID number, and defect type (we are adding all diffusion reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

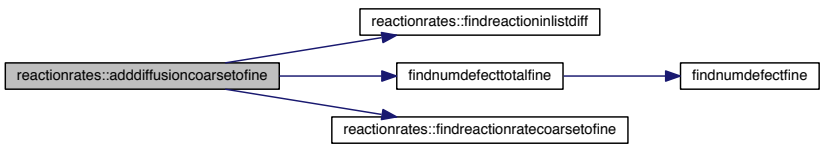
1) Identify if the defect type corresponds to an allowed diffusion reaction reaction using reaction lists

2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

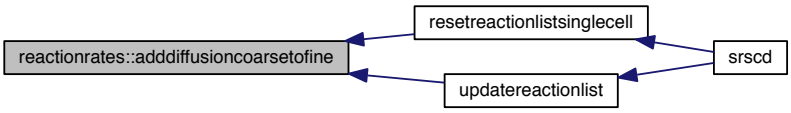
3) Calculate the reaction rate based on the defect type and numbers of defects of this type in this element and in the entire fine mesh. A modified reaction distance is used for this reaction rate.

4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.2.2 subroutine reactionrates::adddiffusionreactions (integer *cell1,* integer *cell2,* integer *proc1,* integer *proc2,* integer *dir,* integer, dimension(numspecies) *defectType*)

Subroutine add diffusion reactions - adds reactions to a reaction list representing diffusion between volume elements.

Inputs: initial and final cell numbers, initial and final processors (if diffusion occurs between domains of multiple processors), direction (1-6), and defect type (we are adding all diffusion reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

1) Identify if the defect type corresponds to an allowed diffusion reaction reaction using reaction lists

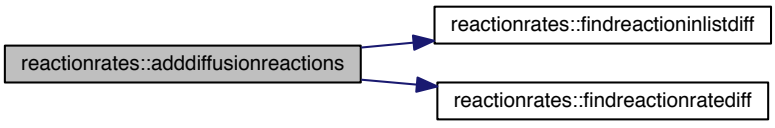
2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

3) Calculate the reaction rate based on the defect type and numbers of defects of this type in this element and in the neighboring element

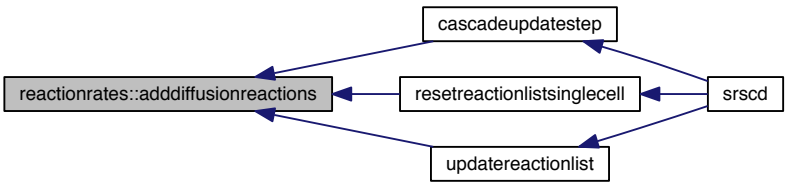
4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

2015.04.06: Potential bug: if a volume element shares multiple faces with another grain, we are only creating one reaction for diffusion from the grain to the grain boundary. Need to include diffusion direction in findReactionInListDiff

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.2.3 subroutine reactionrates::adddiffusionreactionsfine (integer *cascadeID,* integer *cell1,* integer *cell2,* integer *proc1,* integer *proc2,* integer *dir,* integer, dimension(numspecies) *defectType*)

Subroutine add diffusion reactions (fine mesh) - adds reactions to a reaction list representing diffusion between volume elements inside a cascade mesh.

Inputs: cascade ID number, initial and final cell numbers, initial and final processors (if diffusion occurs between domains of multiple processors), direction (1-6), and defect type (we are adding all diffusion reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

1) Identify if the defect type corresponds to an allowed diffusion reaction reaction using reaction lists

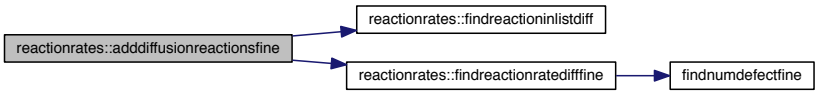
2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

3) Calculate the reaction rate based on the defect type and numbers of defects of this type in this element and in the neighboring element (using the fine mesh element volume)

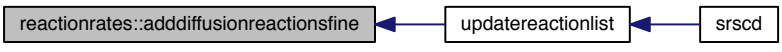
4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

NOTE: includes possibility of diffusion from fine mesh to coarse mesh.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.2.4 subroutine reactionrates::addmultidefectreactions (integer *cell,* integer, dimension(numspecies) *defectType1,* integer, dimension(numspecies) *defectType2*)

Subroutine add multi defect reactions - adds reactions to a reaction list that require multiple (only using 2 currently) defects to be carried out. This refers mainly to clustering reactions or pinning reactions.

Inputs: cell number, defect types (we are adding all clustering reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

1) Identify if the defect types both correspond to an allowed clustering reaction reaction using reaction lists

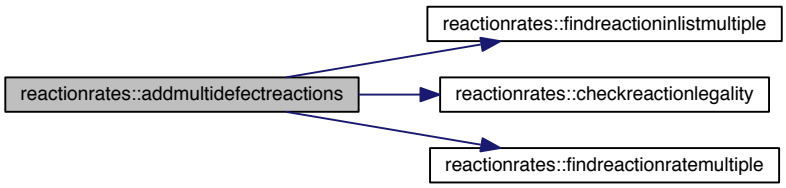
1a) Calculate the resulting product (if the reaction is a clustering reaction), using combination rules. Examples include annihilation of vacancy/interstitial pairs and not allowing He-SIA clusters to form.

2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

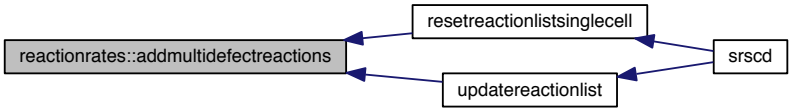
3) Calculate the reaction rate based on the defect types and number of defects

4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.2.5 subroutine reactionrates::addmultidefectreactionsfine (integer *cascadeID,* integer *cell,* integer, dimension(numspecies) *defectType1,* integer, dimension(numspecies) *defectType2*)

Subroutine add multi defect reactions (fine mesh) - adds reactions to a reaction list inside a cascade that require multiple (only using 2 currently) defects to be carried out. This refers mainly to clustering reactions or pinning reactions.

Inputs: cascade ID number, cell number, defect types (we are adding all clustering reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

1) Identify if the defect types both correspond to an allowed clustering reaction reaction using reaction lists

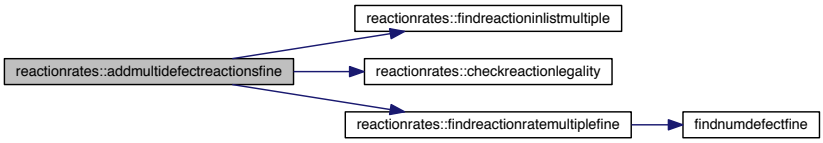
1a) Calculate the resulting product (if the reaction is a clustering reaction), using combination rules. Examples include annihilation of vacancy/interstitial pairs and not allowing He-SIA clusters to form.

2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

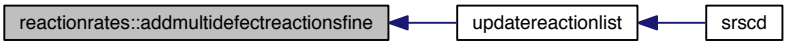
3) Calculate the reaction rate based on the defect types and number of defects (and using the cascade volume)

4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.2.6 subroutine reactionrates::addsingledefectreactions (integer *cell,* integer, dimension(numspecies) *defectType*)

Subroutine add single defect reactions - adds reactions to a reaction list that require only a single reactant to be carried out. Examples: dissociation, trapping, sinks. Diffusion reactions are not included in this subroutine.

Inputs: cell number, defect type (we are adding all single defect reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

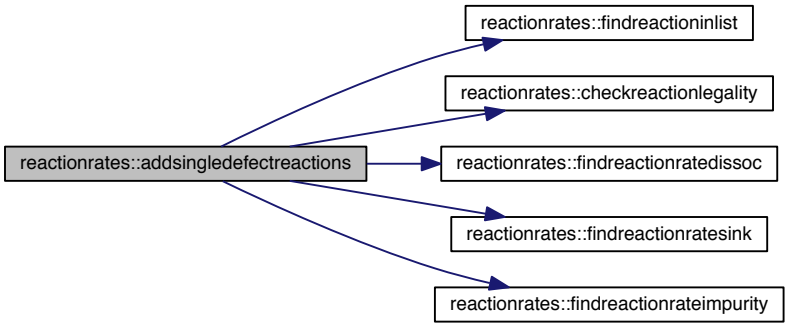
1) Identify if the defect type corresponds to an allowed reaction using reaction lists

2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

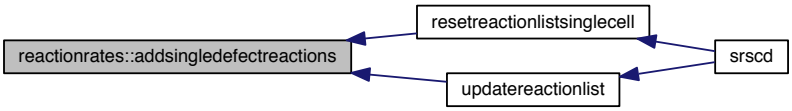
3) Calculate the reaction rate based on the defect type and number of defects

4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.2.7 subroutine reactionrates::addsingledefectreactionsfine (integer *cascadeID,* integer *cell,* integer, dimension(numspecies) *defectType*)

Subroutine add single defect reactions fine - adds reactions to a reaction list inside a cascade (fine mesh) that require only a single reactant to be carried out. Examples: dissociation, trapping, sinks. Diffusion reactions are not included in this subroutine.

Inputs: cascade ID number, cell number, defect type (we are adding all single defect reactions associated with a single defect type, which was a defect that changed in the previous Monte Carlo step)

Structure of subroutine:

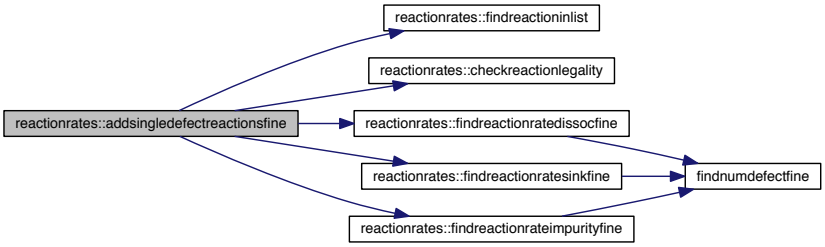
1) Identify if the defect type corresponds to an allowed reaction using reaction lists

2) Find the reaction in the reaction list for this volume element (if it exists) and go to the end of the list if not (NOTE: list is unsorted as of now, 03/31/2015)

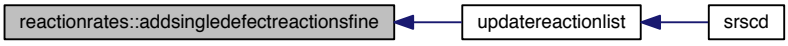
3) Calculate the reaction rate based on the defect type and number of defects (using fine mesh volume element size)

4) Update the reaction rate / add the reaction / remove the reaction, depending on if the reaction is already present and if the reaction rate is nonzero.

Here is the call graph for this function:



Here is the caller graph for this function:



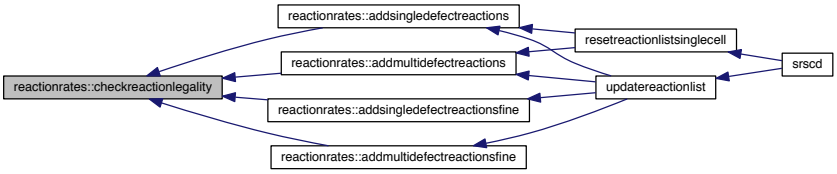
8.5.2.8 subroutine reactionrates::checkreactionlegality (integer *numProducts,* integer, dimension(numproducts, numspecies) *products,* logical *isLegal*)

Subroutine check Reaction Legality.

This subroutine looks at the products of a combination reaction and checks to see if the reaction is allowed (using hard-coded information). If not, the subroutine returns a value of .FALSE. to isLegal.

Inputs: numProducts, products(numProducts, numSpecies) Output: isLegal (boolean variable)

Here is the caller graph for this function:



8.5.2.9 subroutine reactionrates::defectcombinationrules (integer, dimension(numspecies) *products,* type(defect), pointer *defectTemp*)

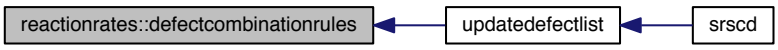
defectCombinationRules (subroutine)

Hard-coded information on what happens to defects when they cluster (for example, annihilation, formation of sessile SIA clusters, etc).

Inputs: products(numSpecies), defectTemp Outputs: products(numSpeices), updated using the combination rules from defectTemp

NOTE: defect combination rules are primarily input directly into subroutines addMultiDefectReactions and add↩ MultiDefectReactionsFine, and this subroutine is only called in order to get correct defect combination in the case of cascade-defect interactions. In the future, may want to move all defect combination rules to this subroutine so that they only need to be changed once.

Here is the caller graph for this function:



8.5.2.10 double precision function reactionrates::finddparatelocal (double precision *zCoord*)

Function find DPA Rate Local (zCoord)

This function finds the DPA rate in a non-uniform implantation profile given a z-coordinate of the center of the mesh element. An error is returned if the non-uniform implantation profile does not completely surround the mesh. This function uses information input from a file.

Input: zCoord, the z-coordinate of the center of the mesh element we are looking for Output: the DPA rate at that point in the non-uniform implantation profile, based on the input file

Here is the caller graph for this function:

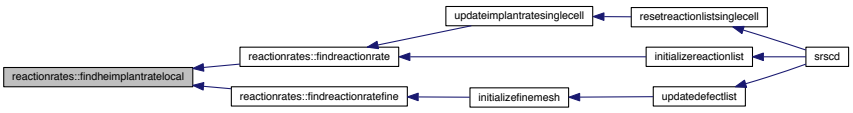
8.5.2.11 double precision function reactionrates::findheimplantratelocal (double precision *zCoord*)

Function find Helium Implantation Rate Local (zCoord)

This function finds the He implant rate rate in a non-uniform implantation profile given a z-coordinate of the center of the mesh element. An error is returned if the non-uniform implantation profile does not completely surround the mesh.

Input: zCoord, the z-coordinate of the center of the mesh element we are looking for Output: the DPA rate at that point in the non-uniform implantation profile, based on the input file

Here is the caller graph for this function:



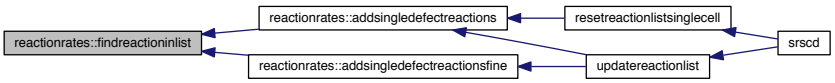
8.5.2.12 subroutine reactionrates::findreactioninlist (type(reaction), pointer *reactionCurrent,* type(reaction), pointer *reactionPrev,* integer *cell,* integer, dimension(:,:), allocatable *reactants,* integer, dimension(:,:), allocatable *products,* integer *numReactants,* integer *numProducts*)

Subroutine find Reaction In List.

Points reactionCurrent at the reaction in coarse or fine mesh with matching reactants and products in cell If reaction is not present, reactionCurrent is not associated and reactionPrev points to the end of the list.

Inputs: cell, reactants(:,:), products(:,:), numReactants, numProducts Outputs: reactionCurrent, reactionPrev (pointers)

Here is the caller graph for this function:



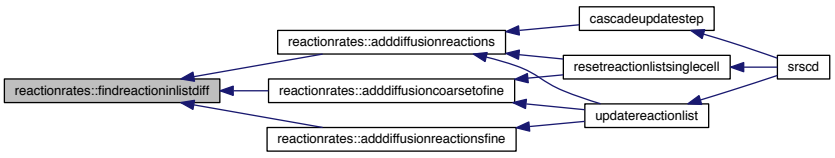
8.5.2.13 subroutine reactionrates::findreactioninlistdiff (type(reaction), pointer *reactionCurrent,* type(reaction), pointer *reactionPrev,* integer, dimension(:,:), allocatable *reactants,* integer *cell1,* integer *cell2,* integer *proc1,* integer *proc2*)

Subroutine find Reaction In List (diffusion)

Points reactionCurrent at the correct diffusion reaction in coarse or fine mesh with matching reactants and products in cells If reaction is not present, reactionCurrent is not associated and reactionPrev points to the end of the list.

Inputs: cells, processors, reactants(:,:) Outputs: reactionCurrent, reactionPrev (pointers)

Here is the caller graph for this function:



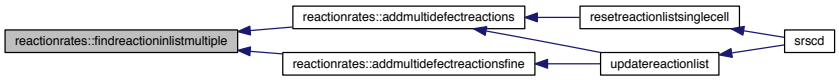
8.5.2.14 subroutine reactionrates::findreactioninlistmultiple (type(reaction), pointer *reactionCurrent,* type(reaction), pointer *reactionPrev,* integer *cell,* integer, dimension(:,:), allocatable *reactants,* integer, dimension(:,:), allocatable *products,* integer *numReactants,* integer *numProducts*)

Subroutine find Reaction In List Multiple (clustering)

Points reactionCurrent at the clustering reaction in the coarse or fine mesh with matching reactants and products in cell If reaction is not present, reactionCurrent is not associated and reactionPrev points to the end of the list.

Inputs: cell, reactants(:,:), products(:,:), numReactants, numProducts Outputs: reactionCurrent, reactionPrev (pointers)

Here is the caller graph for this function:



8.5.2.15 double precision function reactionrates::findreactionrate (integer *cell,* type(reactionparameters) *reactionParameter*)

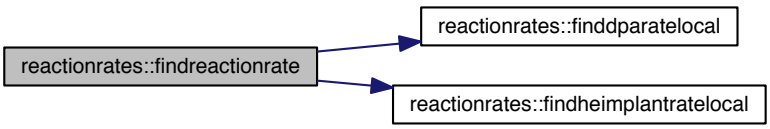
Function find reaction rate - finds reaction rate for implantation reaction (He, Frenkel pairs, cascades).

Inputs: cell ID, reaction parameters (input from file)

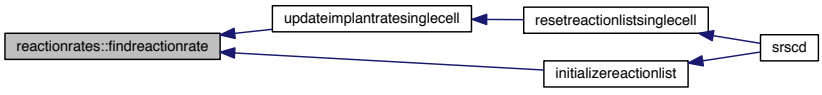
Calculates reaction rates according to hard-coded formulas. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

NOTE: for the case of non-uniform implantation, reaction rates can be dependent on the z-coordinate of a volume element, using a local DPA rate and helium implantation rate read in from an input file.

Here is the call graph for this function:



Here is the caller graph for this function:



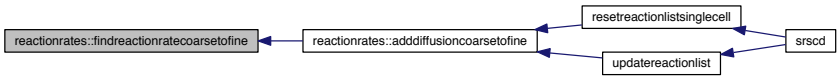
8.5.2.16 double precision function reactionrates::findreactionratecoarsetofine (integer, dimension(numspecies) *defectType,* integer *cell,* integer *proc,* integer *numDefectsFine,* type(reactionparameters) *reactionParameter*)

Function find reaction rate diffusion coarse to fine - finds reaction rate for defect diffusion between a coarse mesh element and a fine (cascade) mesh.

Inputs: cell ID, processor ID, reaction parameters (input from file), defect type (array size numSpecies), number of defects of this type in the fine mesh already

Calculates reaction rates for diffusion between volume elements. Using the finite-volume version of Fick's law to find rates (using first derivative), with a modified diffusion distance according to the formula in Dunn et al. (Computational Materials Science 2015)

Here is the caller graph for this function:



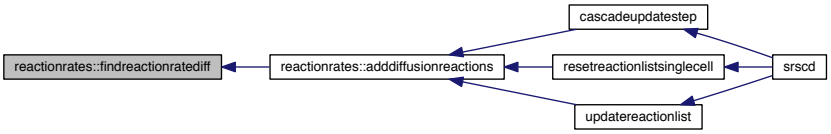
8.5.2.17 double precision function reactionrates::findreactionratediff (integer, dimension(numspecies) *defectType,* integer *cell1,* integer *proc1,* integer *cell2,* integer *proc2,* integer *dir,* type(reactionparameters) *reactionParameter*)

Function find reaction rate diffusion - finds reaction rate for defect diffusion between elements.

Inputs: cell IDs, processor IDs, diffusion direction, reaction parameters (input from file), defect type (array size numSpecies)

Calculates reaction rates for diffusion between volume elements. Using the finite-volume version of Fick's law to find rates (using first derivative).

Here is the caller graph for this function:



8.5.2.18 double precision function reactionrates::findreactionratedifffine (type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType,* integer *cell1,* integer *proc1,* integer *cell2,* integer *proc2,* integer *dir,* type(reactionparameters) *reactionParameter*)

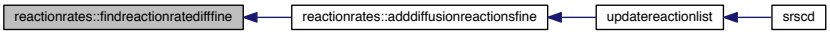
Function find reaction rate diffusion fine - finds reaction rate for defect diffusion between elements in the fine mesh (inside a cascade)

Inputs: cell IDs, processor IDs, diffusion direction, reaction parameters (input from file), defect type (array size numSpecies), cascade derived type

Calculates reaction rates for diffusion between volume elements. Using the finite-volume version of Fick's law to find rates (using first derivative). Includes rate for diffusion from fine mesh to coarse mesh.

Here is the call graph for this function:

Here is the caller graph for this function:



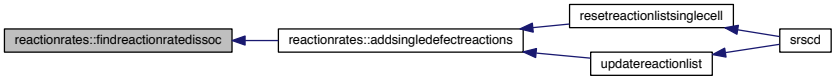
8.5.2.19 double precision function reactionrates::findreactionratedissoc (integer, dimension(numspecies) *defectType,* integer, dimension(2,numspecies) *products,* integer *cell,* type(reactionparameters) *reactionParameter*)

Function find reaction rate dissociation - finds reaction rate for point defects to dissociate from clusters.

Inputs: cell ID, reaction parameters (input from file), defect type (array size numSpecies), dissociating defect type

Calculates reaction rates for dissociation of a point defect from a cluster. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Here is the caller graph for this function:



8.5.2.20 double precision function reactionrates::findreactionratedissocfine (type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType,* integer, dimension(2,numspecies) *products,* integer *cell,* type(reactionparameters) *reactionParameter*)

Function find reaction rate dissociation - finds reaction rate for point defects to dissociate from clusters in the fine mesh (cascade)

Inputs: cell ID, reaction parameters (input from file), defect type (array size numSpecies), dissociating defect type, cascade derived type

Calculates reaction rates for dissociation of a point defect from a cluster. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Cascade derived type is used to find the number of defects in the volume element (it is required in a later subroutine)

Here is the call graph for this function:

Here is the caller graph for this function:

8.5.2.21 double precision function reactionrates::findreactionratefine (integer *cell,* type(reactionparameters) *reactionParameter*)

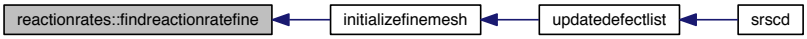
Function find reaction rate fine - finds reaction rate for implantation reaction (Helium or Frenkel pairs only) in a fine mesh. Cascades cannot be implanted inside other cascades.

Inputs: cell ID, reaction parameters (input from file)

Calculates reaction rates according to hard-coded formulas. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Here is the call graph for this function:

Here is the caller graph for this function:



8.5.2.22 double precision function reactionrates::findreactionrateimpurity (integer, dimension(numspecies) *defectType,* integer *cell,* type(reactionparameters) *reactionParameter*)

Function find reaction rate impurity - finds reaction rate for trapping of SIA loops by impurities (Carbon).

Inputs: cell ID, reaction parameters (input from file), defect type (array size numSpecies)

Calculates reaction rates for SIA loop trapping according to hard-coded formulas. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

In this case, glissile loops become sessile when they interact with impurities. Have the option for 3-D or 1-D diffusion of loops.

We have also added the reaction rate for sessile clusters to become mobile again using a binding energy in this subroutine. The binding energy is read in from an input file.

Here is the caller graph for this function:

8.5.2.23 double precision function reactionrates::findreactionrateimpurityfine (type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType,* integer *cell,* type(reactionparameters) *reactionParameter*)

Function find reaction rate impurity fine - finds reaction rate for trapping of SIA loops by impurities (Carbon) inside a fine mesh (Cascade).

Inputs: cascade derived type, cell ID, reaction parameters (input from file), defect type (array size numSpecies)

Calculates reaction rates for SIA loop trapping according to hard-coded formulas inside a fine mesh (cascade). Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

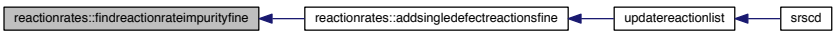
The cascade derived type must be passed into this subroutine in order to find the number of defects of this type.

In this case, glissile loops become sessile when they interact with impurities. Have the option for 3-D or 1-D diffusion of loops.

We have also added the reaction rate for sessile clusters to become mobile again using a binding energy in this subroutine. The binding energy is read in from an input file.

Here is the call graph for this function:

Here is the caller graph for this function:



8.5.2.24 double precision function reactionrates::findreactionratemultiple (integer, dimension(numspecies) *defectType1,* integer, dimension(numspecies) *defectType2,* integer *cell,* type(reactionparameters) *reactionParameter*)

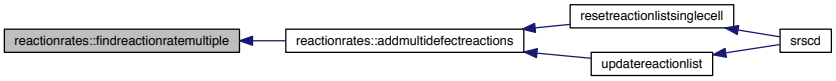
Function find reaction rate multiple - finds reaction rate for defect clustering.

Inputs: cell ID, reaction parameters (input from file), 2 defect types (array size numSpecies)

Calculates reaction rates for two defects to cluster with each other. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Several different hard-coded reaction rates are used here, according to the various reaction rates for clustering between defects depending on their geometry and diffusivity

Here is the caller graph for this function:



8.5.2.25 double precision function reactionrates::findreactionratemultiplefine (type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType1,* integer, dimension(numspecies) *defectType2,* integer *cell,* type(reactionparameters) *reactionParameter*)

Function find reaction rate multiple fine - finds reaction rate for defect clustering in the fine mesh (Cascade)

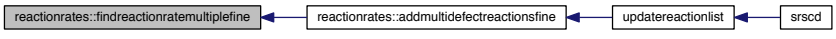
Inputs: cell ID, reaction parameters (input from file), 2 defect types (array size numSpecies), cascade derived type

Calculates reaction rates for two defects to cluster with each other inside the fine mesh (in a cascade). Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Several different hard-coded reaction rates are used here, according to the various reaction rates for clustering between defects depending on their geometry and diffusivity

Here is the call graph for this function:

Here is the caller graph for this function:



8.5.2.26 double precision function reactionrates::findreactionratesink (integer, dimension(numspecies) *defectType,* integer *cell,* type(reactionparameters) *reactionParameter*)

Function find reaction rate sink - finds reaction rate for defects to get absorbed at sinks (typically matrix dislocations)

Inputs: cell ID, reaction parameters (input from file), defect type (array size numSpecies)

Calculates reaction rates for capture of a defect by a sink. Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Here is the caller graph for this function:

8.5.2.27 double precision function reactionrates::findreactionratesinkfine (type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType,* integer *cell,* type(reactionparameters) *reactionParameter*)

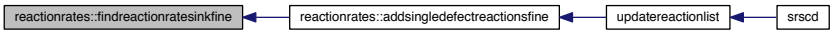
Function find reaction rate sink fine - finds reaction rate for defects to get absorbed at sinks (typically matrix dislocations) in the fine mesh (cascade)

Inputs: cell ID, reaction parameters (input from file), defect type (array size numSpecies), cascade derived type

Calculates reaction rates for capture of a defect by a sink in the fine mesh (in a cascade). Have the possibility to create an arbitrary number of unique formulas for computing reaction rates. Each formula has a function type associated with it, that function type is assigned in the input file.

Here is the call graph for this function:

Here is the caller graph for this function:



Chapter 9

Data Type Documentation

9.1 derivedtype::bindingfunction Type Reference

Type: binding function (list of functional forms for defect binding energies read in from input file)

Public Attributes

- integer, dimension(:), allocatable defecttype

  Type of cluster that is dissociating (1's and 0's only)
- integer, dimension(:), allocatable product

  Type of defect that dissociates from cluster.
- integer, dimension(:), allocatable min

  Minimum cluster size allowed to use this functional form for binding energy (size numSpecies)
- integer, dimension(:), allocatable max

  Maximum cluster size allowed to use this functional form for binding energy (size numSpecies)
- double precision, dimension(:), allocatable parameters

  Parameters to input into this functional form (read in from input file)
- integer functiontype

  ID number of functional form, function is hard coded into Defect_Attributes.f90.
- integer numparam

  Number of parameters needed to input into this functional form.

9.1.1 Detailed Description

Type: binding function (list of functional forms for defect binding energies read in from input file)

This derived type contains a list of the functional forms used to calculate the binding energies of various defect types to clusters. It is used in SRSCD as an array with size given by an input, numBindingFunc.

9.1.2 Member Data Documentation

9.1.2.1 integer, dimension(:), allocatable derivedtype::bindingfunction::defecttype

Type of cluster that is dissociating (1's and 0's only)

9.1.2.2 integer derivedtype::bindingfunction::functiontype

ID number of functional form, function is hard coded into Defect_Attributes.f90.

9.1.2.3 integer, dimension(:), allocatable derivedtype::bindingfunction::max

Maximum cluster size allowed to use this functional form for binding energy (size numSpecies)

9.1.2.4 integer, dimension(:), allocatable derivedtype::bindingfunction::min

Minimum cluster size allowed to use this functional form for binding energy (size numSpecies)

9.1.2.5 integer derivedtype::bindingfunction::numparam

Number of parameters needed to input into this functional form.

9.1.2.6 double precision, dimension(:), allocatable derivedtype::bindingfunction::parameters

Parameters to input into this functional form (read in from input file)

9.1.2.7 integer, dimension(:), allocatable derivedtype::bindingfunction::product

Type of defect that dissociates from cluster.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.2 derivedtype::bindingsingle Type Reference

Type: binding single (list of binding energies)

Public Attributes

- integer, dimension(:), allocatable defecttype
  Type of defect that can dissociate.
- integer, dimension(:), allocatable product
  Type of defect dissociating away.
- double precision eb
  Binding energy (eV)

9.2.1 Detailed Description

Type: binding single (list of binding energies)

This derived type contains a list of binding energies for defects read in from an input file. It contains both the initial defect type as well as the type of defect dissociating from it. It is used in SRSCD as an array with size given by a separate input, numBindingSingle

9.2.2 Member Data Documentation

9.2.2.1 integer, dimension(:), allocatable derivedtype::bindingsingle::defecttype

Type of defect that can dissociate.

9.2.2.2 double precision derivedtype::bindingsingle::eb

Binding energy (eV)

9.2.2.3 integer, dimension(:), allocatable derivedtype::bindingsingle::product

Type of defect dissociating away.

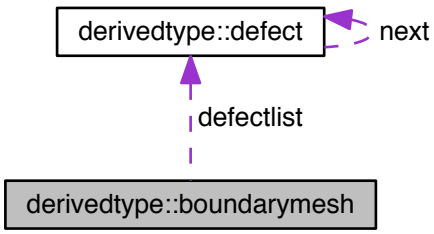
The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.3 derivedtype::boundarymesh Type Reference

Type: boundary mesh (one array of this type per processor)

Collaboration diagram for derivedtype::boundarymesh:



Public Attributes

- integer proc

  Processor ID number of this volume element (typically different from the local processor){.
- integer material

  Material ID number of this volume element (currently only set up for one material)
- integer localneighbor

  ID number of the volume element in the local mesh that borders this volume element.
- double precision length

  Side length of this volume element (cubic assumption)
- double precision volume

  Volume of this element.
- type(defect), pointer defectlist

  List of defects present in this volume element (See defect derived type)
- double precision, dimension(6) strain

  Strain tensor at the center of this volume element.

9.3.1 Detailed Description

Type: boundary mesh (one array of this type per processor)

Contains information on volume elements located in other processors that share a boundary with this processor's domain. Includes cell numbers and volumes as well as lists of defects present in neighboring processors' boundary elements. Each variable of this type represents a single volume element and its defects in a neighboring processor, and an array of these is created to represent the entire boundary.

9.3.2 Member Data Documentation

9.3.2.1 type(defect), pointer derivedtype::boundarymesh::defectlist

List of defects present in this volume element (See defect derived type)

9.3.2.2 double precision derivedtype::boundarymesh::length

Side length of this volume element (cubic assumption)

9.3.2.3 integer derivedtype::boundarymesh::localneighbor

ID number of the volume element in the local mesh that borders this volume element.

9.3.2.4 integer derivedtype::boundarymesh::material

Material ID number of this volume element (currently only set up for one material)

9.3.2.5 integer derivedtype::boundarymesh::proc

Processor ID number of this volume element (typically different from the local processor){.

9.3.2.6 double precision, dimension(6) derivedtype::boundarymesh::strain

Strain tensor at the center of this volume element.

9.3.2.7 double precision derivedtype::boundarymesh::volume

Volume of this element.

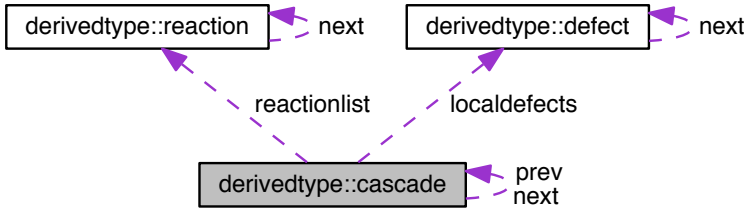
The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.4 derivedtype::cascade Type Reference

Type: cascade (pointer list containing defects and reactions for cascades that are currently present in the system)

Collaboration diagram for derivedtype::cascade:



Public Attributes

- integer cellnumber

  Coarse mesh cell number that this cascade is located inside.
- integer cascadeid

  ID number of this cascade.
- type(defect), dimension(:), pointer localdefects

  Array of defect lists (one for each element in the cascade)
- type(cascade), pointer next

  Pointer pointing to the next cascade currently present in the system.
- type(cascade), pointer prev

  Pointer pointing to the previous cascade currently present in the system.
- type(reaction), dimension(:), pointer reactionlist

  Array of reaction lists (one for each element in the cascade)
- double precision, dimension(:), allocatable totalrate

  Sum of the rates of all reactions in each cascade element.

9.4.1 Detailed Description

Type: cascade (pointer list containing defects and reactions for cascades that are currently present in the system)

This derived type contains a pointer list of all of the cascades that are currently present in the system, including a fine mesh of all of the defect lists inside the cascade and all of the reaction lists that can occur inside the cascade. This derived type is essentially creating a new mesh with a new set of defect lists and reaction lists inside a coarse mesh element.

9.4.2 Member Data Documentation

9.4.2.1 integer derivedtype::cascade::cascadeid

ID number of this cascade.

9.4.2.2 integer derivedtype::cascade::cellnumber

Coarse mesh cell number that this cascade is located inside.

9.4.2.3 type(defect), dimension(:), pointer derivedtype::cascade::localdefects

Array of defect lists (one for each element in the cascade)

9.4.2.4 type(cascade), pointer derivedtype::cascade::next

Pointer pointing to the next cascade currently present in the system.

9.4.2.5 type(cascade), pointer derivedtype::cascade::prev

Pointer pointing to the previous cascade currently present in the system.

9.4.2.6 type(reaction), dimension(:), pointer derivedtype::cascade::reactionlist

Array of reaction lists (one for each element in the cascade)

9.4.2.7 double precision, dimension(:), allocatable derivedtype::cascade::totalrate

Sum of the rates of all reactions in each cascade element.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.5 derivedtype::cascadedefect Type Reference

Type: cascade defect (pointer list of defect types in a cascade)

Collaboration diagram for derivedtype::cascadedefect:

Public Attributes

- integer, dimension(:), allocatable defecttype

  Type of defect (numSpecies)
- double precision, dimension(3) coordinates

  Location of defect relative to center of cascade (in nm)
- type(cascadedefect), pointer next

  Pointer pointing to the next defect in the list.

9.5.1 Detailed Description

Type: cascade defect (pointer list of defect types in a cascade)

This derived type contains a list of defects as well as their locations within a cascade. This information is read in from a file and stored for use whenever a cascade event is chosen in the Monte Carlo algorithm.

9.5.2 Member Data Documentation

9.5.2.1 double precision, dimension(3) derivedtype::cascadedefect::coordinates

Location of defect relative to center of cascade (in nm)

9.5.2.2 integer, dimension(:), allocatable derivedtype::cascadedefect::defecttype

Type of defect (numSpecies)

9.5.2.3 type(cascadedefect), pointer derivedtype::cascadedefect::next

Pointer pointing to the next defect in the list.

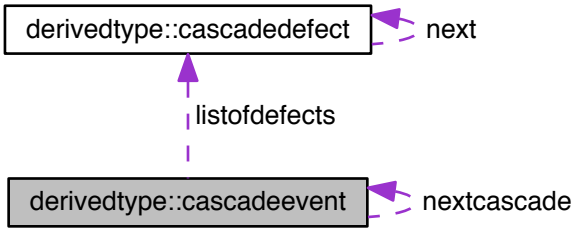
The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.6 derivedtype::cascadeevent Type Reference

Type: cascade event (pointer list of read-in cascade data from input file)

Collaboration diagram for derivedtype::cascadeevent:



Public Attributes

- integer numdefectstotal

  Number of total defects in the cascade list.
- integer numdisplacedatoms

  How much this cascade contributes to the DPA (how many lattice atoms are displaced)
- type(cascadedefect), pointer listofdefects

Pointer pointing to the list of defects in this cascade.

- type(cascadeevent), pointer nextcascade

  Pointer pointing to the next cascade in the list.

9.6.1 Detailed Description

Type: cascade event (pointer list of read-in cascade data from input file)

This derived type contains a list of the cascades that are read in from a file including what defects they contain. Cascades are randomly chosen for implantation when a cascade event is chosen in the Monte Carlo algorithm.

9.6.2 Member Data Documentation

9.6.2.1 type(cascadedefect), pointer derivedtype::cascadeevent::listofdefects

Pointer pointing to the list of defects in this cascade.

9.6.2.2 type(cascadeevent), pointer derivedtype::cascadeevent::nextcascade

Pointer pointing to the next cascade in the list.

9.6.2.3 integer derivedtype::cascadeevent::numdefectstotal

Number of total defects in the cascade list.

9.6.2.4 integer derivedtype::cascadeevent::numdisplacedatoms

How much this cascade contributes to the DPA (how many lattice atoms are displaced)

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.7 derivedtype::defect Type Reference

Type: defect (pointer list).

Collaboration diagram for derivedtype::defect:

Public Attributes

- integer, dimension(:), allocatable defecttype

Array containing the number of particles of each defect species in this defect type. (Example: here, 3 2 0 0 indicates He_3V_2 and 0 0 20 0 indicates SIA_20 (glissile)

- integer num

  Number of defects of this type inside this volume element.
- integer cellnumber

  Volume element that these defects are located in.
- type(defect), pointer next

  Pointer to the next defect in the same volume element (sorted list)

9.7.1 Detailed Description

Type: defect (pointer list).

Contains the defect type, the number of said defects in a volume element, the cell number, and a pointer to the next defect in the list.

9.7.2 Member Data Documentation

9.7.2.1 integer derivedtype::defect::cellnumber

Volume element that these defects are located in.

9.7.2.2 integer, dimension(:), allocatable derivedtype::defect::defecttype

Array containing the number of particles of each defect species in this defect type. (Example: here, 3 2 0 0 indicates He_3V_2 and 0 0 20 0 indicates SIA_20 (glissile)

9.7.2.3 type(defect), pointer derivedtype::defect::next

Pointer to the next defect in the same volume element (sorted list)

9.7.2.4 integer derivedtype::defect::num

Number of defects of this type inside this volume element.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.8 derivedtype::defectupdatetracker Type Reference

Type: defect update tracker (pointer list)

Collaboration diagram for derivedtype::defectupdatetracker:

Public Attributes

- integer, dimension(:), allocatable defecttype

  Type of defect that needs to be updated (see description of defect type for format)
- integer num

  Number of defects of this type in the volume element.
- integer cellnumber

  Volume element number that this defect is located in.
- integer proc

  Processor ID # that this volume element is located inside.
- integer dir

  If defect diffused from a different volume element, indicates direction from which defect came.
- integer neighbor

  If defect diffused from a different volume element, indicates volume element number from which defect came.
- integer cascadenumber

  If defect is inside a cascade, indicates cascade ID # that defect is located inside.
- type(defectupdatetracker), pointer next

  Pointer to the next defect that needs to have its reaction rate updated.

9.8.1 Detailed Description

Type: defect update tracker (pointer list)

Contains a list of defects that need to be updated (along with reaction rates) due to the reaction that got carried out during the previous step. Has more information than just the defect type in order to make the updateReactionList subroutine easier.

9.8.2 Member Data Documentation

9.8.2.1 integer derivedtype::defectupdatetracker::cascadenumber

If defect is inside a cascade, indicates cascade ID # that defect is located inside.

9.8.2.2 integer derivedtype::defectupdatetracker::cellnumber

Volume element number that this defect is located in.

9.8.2.3 integer, dimension(:), allocatable derivedtype::defectupdatetracker::defecttype

Type of defect that needs to be updated (see description of defect type for format)

9.8.2.4 integer derivedtype::defectupdatetracker::dir

If defect diffused from a different volume element, indicates direction from which defect came.

9.8.2.5 integer derivedtype::defectupdatetracker::neighbor

If defect diffused from a different volume element, indicates volume element number from which defect came.

9.8.2.6 type(defectupdatetracker), pointer derivedtype::defectupdatetracker::next

Pointer to the next defect that needs to have its reaction rate updated.

9.8.2.7 integer derivedtype::defectupdatetracker::num

Number of defects of this type in the volume element.

9.8.2.8 integer derivedtype::defectupdatetracker::proc

Processor ID # that this volume element is located inside.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.9 derivedtype::diffusionfunction Type Reference

Type: diffusion function (list of functional forms for defect diffusivities read in from input file)

Public Attributes

- integer, dimension(:), allocatable defecttype

  Type of defect that is allowed to diffuse using this functional form (1's and 0's only).
- integer, dimension(:), allocatable min

  Minimum defect size allowed to diffuse using this functional form.
- integer, dimension(:), allocatable max

  Maximum defect size allowed to diffuse using this functional form (-1 indicates infinity)
- double precision, dimension(:), allocatable parameters

  Parameters to input into this functional form (the function is hard-coded into Defect_Attributes.f90)
- integer functiontype

  ID number of the function type to use to calculate diffusivity.
- integer numparam

  Number of parameters needed to input into this functional form.

9.9.1 Detailed Description

Type: diffusion function (list of functional forms for defect diffusivities read in from input file)

This derived type contains information read in from an input file concerning which defects have diffusivity given by a functional form, which functional form to use, and parameters to input into the functional form. One list (array) of this type is created in each processor.

9.9.2 Member Data Documentation

9.9.2.1 integer, dimension(:), allocatable derivedtype::diffusionfunction::defecttype

Type of defect that is allowed to diffuse using this functional form (1's and 0's only).

9.9.2.2 integer derivedtype::diffusionfunction::functiontype

ID number of the function type to use to calculate diffusivity.

9.9.2.3 integer, dimension(:), allocatable derivedtype::diffusionfunction::max

Maximum defect size allowed to diffuse using this functional form (-1 indicates infinity)

9.9.2.4 integer, dimension(:), allocatable derivedtype::diffusionfunction::min

Minimum defect size allowed to diffuse using this functional form.

9.9.2.5 integer derivedtype::diffusionfunction::numparam

Number of parameters needed to input into this functional form.

9.9.2.6 double precision, dimension(:), allocatable derivedtype::diffusionfunction::parameters

Parameters to input into this functional form (the function is hard-coded into Defect_Attributes.f90)

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.10 derivedtype::diffusionsingle Type Reference

Type: diffusion single (list of single migration energies and diffusion prefactors)

Public Attributes

- integer, dimension(:), allocatable defecttype
  Type of defect that is allowed to diffuse.
- double precision d
  Diffusion prefactor (nm^2/s)
- double precision em
  Migration energy (eV)

9.10.1 Detailed Description

Type: diffusion single (list of single migration energies and diffusion prefactors)

This derived type contains a list of diffusion energies and prefactors for defects read in from an input file. It specifies the defect type and the diffusion parameters. It is used in SRSCD as an array with size given by a separate input, numDiffSingle

9.10.2 Member Data Documentation

9.10.2.1 double precision derivedtype::diffusionsingle::d

Diffusion prefactor (nm^2/s)

9.10.2.2 integer, dimension(:), allocatable derivedtype::diffusionsingle::defecttype

Type of defect that is allowed to diffuse.

9.10.2.3 double precision derivedtype::diffusionsingle::em

Migration energy (eV)

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.11 derivedtype::dipoletensor Type Reference

Type: dipole tensor (stored dipole tensor data for various defect types, input from file)

Public Attributes

- integer, dimension(:), allocatable min

  Smallest defect size for this defect type.
- integer, dimension(:), allocatable max

  Largest defect size for this defect type (-1 means infinity)
- double precision, dimension(6) equilib

  Equilibrium state dipole tensor(d11, d22, d33, d12, d23, d13)
- double precision, dimension(6) saddle

  Saddle point (for migration) dipole tensor.

9.11.1 Detailed Description

Type: dipole tensor (stored dipole tensor data for various defect types, input from file)

This derived type is used to store information on the dipole tensors that are used in the simulation to calculate modified diffusivity in the presence of a strain field. It contains the defect type for which the dipole tensor refers as well as the dipole tensor itself (in units of eV).

9.11.2 Member Data Documentation

9.11.2.1 double precision, dimension(6) derivedtype::dipoletensor::equilib

Equilibrium state dipole tensor(d11, d22, d33, d12, d23, d13)

9.11.2.2 integer, dimension(:), allocatable derivedtype::dipoletensor::max

Largest defect size for this defect type (-1 means infinity)

9.11.2.3 integer, dimension(:), allocatable derivedtype::dipoletensor::min

Smallest defect size for this defect type.

9.11.2.4 double precision, dimension(6) derivedtype::dipoletensor::saddle

Saddle point (for migration) dipole tensor.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.12 derivedtype::mesh Type Reference

Type: mesh (local, one array of this type per processor)

Public Attributes

- double precision, dimension(3) coordinates

  Coordinates of center of volume element.
- double precision length

  Side length of this volume element (assuming cubic elements)
- double precision volume

  Volume of this volume element.
- double precision, dimension(6) strain

  Strain tensor at the center of this volume element (e11, e22, e33, e12, e23, e13)
- integer proc

  Processor ID number that this element is located inside.
- integer material

  Material ID number that this element is composed of (currently only set up for one material type)
- integer, dimension(6) numneighbors

  Number of neighbors in each direction (left, right, etc). Could be not equal to 1 in the case of free surfaces or non-uniform mesh.
- integer, dimension(:,:), allocatable neighbors

  ID number of neighboring volume elements, regardless of if they are in this processor or not. Array size (num↩ Neighbors, 6)
- integer, dimension(:,:), allocatable neighborprocs

  Processor ID numbers of neighboring volume element. Array size (numNeighbors, 6)

9.12.1 Detailed Description

Type: mesh (local, one array of this type per processor)

Contains the mesh and connectivity of the volume elements in the local processor as well as neighboring processors and material ID numbers. Each variable of this type represents a single volume element, and a single array of this type is constructed to represent the entire mesh. This has been partially implemented to allow non-uniform meshes.

9.12.2 Member Data Documentation

9.12.2.1 double precision, dimension(3) derivedtype::mesh::coordinates

Coordinates of center of volume element.

9.12.2.2 double precision derivedtype::mesh::length

Side length of this volume element (assuming cubic elements)

9.12.2.3 integer derivedtype::mesh::material

Material ID number that this element is composed of (currently only set up for one material type)

9.12.2.4 integer, dimension(:,:), allocatable derivedtype::mesh::neighborprocs

Processor ID numbers of neighboring volume element. Array size (numNeighbors, 6)

9.12.2.5 integer, dimension(:,:), allocatable derivedtype::mesh::neighbors

ID number of neighboring volume elements, regardless of if they are in this processor or not. Array size (num↩ Neighbors, 6)

9.12.2.6 integer, dimension(6) derivedtype::mesh::numneighbors

Number of neighbors in each direction (left, right, etc). Could be not equal to 1 in the case of free surfaces or non-uniform mesh.

9.12.2.7 integer derivedtype::mesh::proc

Processor ID number that this element is located inside.

9.12.2.8 double precision, dimension(6) derivedtype::mesh::strain

Strain tensor at the center of this volume element (e11, e22, e33, e12, e23, e13)

9.12.2.9 double precision derivedtype::mesh::volume

Volume of this volume element.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.13 derivedtype::processordata Type Reference

Type: processorData (one variable per processor)

Public Attributes

- integer taskid

  ID number of this processor (0=master)
- integer numtasks

  Number of processors in this simulation.
- double precision, dimension(6) globalcoord

  Global boundaries of system (xmin, xmax, ymin, ymax, zmin, zmax)
- double precision, dimension(6) localcoord

  Local boundaries of this processor (xmin, xmax, ymin, ymax, zmin, zmax)
- integer, dimension(6) procneighbor

  ID numbers of procesors neighboring this one (left, right, front, back, up, down). Accounts for periodic boundary conditions, if applicable.

9.13.1 Detailed Description

Type: processorData (one variable per processor)

Contains information about this processor (ID number, coordinates of its boundaries) as well as its neighbors (ID numbers) and the global system (number of processors, coordinates of entire volume's boundaries)

9.13.2 Member Data Documentation

9.13.2.1 double precision, dimension(6) derivedtype::processordata::globalcoord

Global boundaries of system (xmin, xmax, ymin, ymax, zmin, zmax)

9.13.2.2 double precision, dimension(6) derivedtype::processordata::localcoord

Local boundaries of this processor (xmin, xmax, ymin, ymax, zmin, zmax)

9.13.2.3 integer derivedtype::processordata::numtasks

Number of processors in this simulation.

9.13.2.4 integer, dimension(6) derivedtype::processordata::procneighbor

ID numbers of procesors neighboring this one (left, right, front, back, up, down). Accounts for periodic boundary conditions, if applicable.

9.13.2.5 integer derivedtype::processordata::taskid

ID number of this processor (0=master)

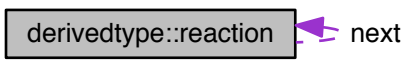
The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.14 derivedtype::reaction Type Reference

Type: reaction (pointer list)

Collaboration diagram for derivedtype::reaction:



Public Attributes

- integer numreactants

Number of reactants in this reaction.

- integer numproducts

  Number of products in this reaction.

- integer, dimension(:,:), allocatable reactants

  Reactants in this reaction - array size:(numReactants, numSpecies)

- integer, dimension(:,:), allocatable products

  Products in this reaction - array size:(numReactants, numSpecies)

- integer, dimension(:), allocatable cellnumber

  Cell numbers of defects involved in this reaction. Array size: (numReactants+numProducts). Important for diffusion reactions.

- integer, dimension(:), allocatable taskid

  Processor number of defects involved in this reaction. Array size: (numReactants+numProducts). May not all be the same processor number in the case of diffusion across processor boundaries.

- double precision reactionrate

  Rate (s^-1) for this reaction to be carried out, based on the defects present in the volume.

- type(reaction), pointer next

  Pointer to the next reaction in the list in the same volume element (unsorted list as of now)

9.14.1 Detailed Description

Type: reaction (pointer list)

Contains a list of reactions that can be carried out inside a given volume element, along with the reactants, products, number of reactants, number of products, and reaction rate.

9.14.2 Member Data Documentation

9.14.2.1 integer, dimension(:), allocatable derivedtype::reaction::cellnumber

Cell numbers of defects involved in this reaction. Array size: (numReactants+numProducts). Important for diffusion reactions.

9.14.2.2 type(reaction), pointer derivedtype::reaction::next

Pointer to the next reaction in the list in the same volume element (unsorted list as of now)

9.14.2.3 integer derivedtype::reaction::numproducts

Number of products in this reaction.

9.14.2.4 integer derivedtype::reaction::numreactants

Number of reactants in this reaction.

9.14.2.5 integer, dimension(:,:), allocatable derivedtype::reaction::products

Products in this reaction - array size:(numReactants, numSpecies)

9.14.2.6 integer, dimension(:,:), allocatable derivedtype::reaction::reactants

Reactants in this reaction - array size:(numReactants, numSpecies)

9.14.2.7 double precision derivedtype::reaction::reactionrate

Rate (s^-1) for this reaction to be carried out, based on the defects present in the volume.

9.14.2.8 integer, dimension(:), allocatable derivedtype::reaction::taskid

Processor number of defects involved in this reaction. Array size: (numReactants+numProducts). May not all be the same processor number in the case of diffusion across processor boundaries.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

9.15 derivedtype::reactionparameters Type Reference

Type: reaction parameters (list of allowed reactions, input from file)

Public Attributes

- integer, dimension(:,:), allocatable reactants
  Defect types for reactants in this reaction.
- integer, dimension(:,:), allocatable products
  Defect types for products in this reaction.
- integer, dimension(:), allocatable min
  Smallest defect size for reactants (size numSpecies)
- integer, dimension(:), allocatable max
  Largest defect size for reactants (size numSpecies, -1 indicates infinity)
- integer numreactants
  Number of reactants (0, 1, or 2 typically)
- integer numproducts
  Number of products.
- integer functiontype
  ID number of function type used to calculate reaction rate, functions are hard coded in ReactionRates.f90.

9.15.1 Detailed Description

Type: reaction parameters (list of allowed reactions, input from file)

This derived type is used in SRSCD as an array with size given by the number of allowed reactions that can be carried out in the material model chosen. It contains the parameters as well as an ID number for the function type that determine the reaction rate for a given allowed reaction. Functional forms for reaction rates are hard coded in ReactionRates.f90

9.15.2 Member Data Documentation

9.15.2.1 integer derivedtype::reactionparameters::functiontype

ID number of function type used to calculate reaction rate, functions are hard coded in ReactionRates.f90.

9.15.2.2 integer, dimension(:), allocatable derivedtype::reactionparameters::max

Largest defect size for reactants (size numSpecies, -1 indicates infinity)

9.15.2.3 integer, dimension(:), allocatable derivedtype::reactionparameters::min

Smallest defect size for reactants (size numSpecies)

9.15.2.4 integer derivedtype::reactionparameters::numproducts

Number of products.

9.15.2.5 integer derivedtype::reactionparameters::numreactants

Number of reactants (0, 1, or 2 typically)

9.15.2.6 integer, dimension(:,:), allocatable derivedtype::reactionparameters::products

Defect types for products in this reaction.

9.15.2.7 integer, dimension(:,:), allocatable derivedtype::reactionparameters::reactants

Defect types for reactants in this reaction.

The documentation for this type was generated from the following file:

- mod_derivedtypes.f90

Chapter 10

File Documentation

10.1 Cascade_implantation.f90 File Reference

Functions/Subroutines

- subroutine choosecascade (CascadeTemp)

  Subroutine: Choose Cascade Takes list of cascades (read from input file) and chooses one randomly.
- integer function cascadecount ()

  Integer function cascadeCount()
- subroutine addcascadeexplicit (reactionCurrent)

  subroutine addCascadeExplicit
- logical function cascademixingcheck ()

  logical function cascadeMixingCheck()
- subroutine cascadeupdatestep (cascadeCell)

  Subroutine CascadeUpdateStep(cascadeCell)
- subroutine createcascadeconnectivity ()

  subroutine createCascadeConnectivity()

10.1.1 Function/Subroutine Documentation

10.1.1.1 subroutine addcascadeexplicit (type(reaction), pointer *reactionCurrent*)

subroutine addCascadeExplicit

This subroutine takes the place of chooseReaction in the case of explicit cascade implantation. It forces the program to 'choose' a cascade reaction, instead of using the Monte Carlo algorithm to do so.

Inputs: none Outputs: reactionCurrent and CascadeCurrent, pointing at cascade reaction.

Here is the call graph for this function:

Here is the caller graph for this function:

10.1.1.2 integer function cascadecount ( )

Integer function cascadeCount()

This subroutine counts how many cascades are active in the LOCAL mesh (not on all processors) and returns that value

Inputs: none Outputs: number of cascades present

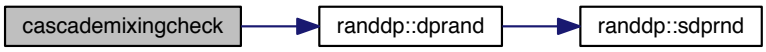
10.1.1.3 logical function cascademixingcheck ( )

logical function cascadeMixingCheck()

cascadeMixingCheck checks whether a given defect in the fine mesh interacts with the cascade. It returns a logical value of true or false.

Input: none Output: logical value for whether or not a fine mesh defect combines with a cascade.

Here is the call graph for this function:



10.1.1.4 subroutine cascadeupdatestep (integer *cascadeCell*)

Subroutine CascadeUpdateStep(cascadeCell)

Carries out the communication necessary when a cascade is created or destroyed in an element that bounds another processor.

Step 1: Check to see if cascade was created/destroyed in element that is in the boundary of another processor.

Step 2: Send message to neighboring processors about whether a cascade was created/destroyed in their boundaries, and if so the number of defects to recieve in the boundary element

Step 3: Send/recieve boundary element defects (just completely re-write boundary element defects in this step)

Step 4: Update reaction rates for all diffusion reactions from elements neighboring the boundary elements which have been updated

Inputs: cascadeCell (integer, 0 if no cascade, otherwise gives number of volume element that cascade event has occurred in)

Outputs: none

Actions: see above, sends/recieves information on boundary updates and updates reaction lists.

Here is the call graph for this function:

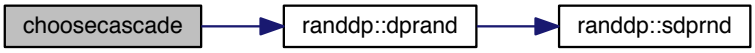
Here is the caller graph for this function:

10.1.1.5 subroutine choosecascade (type(cascadeevent), pointer *CascadeTemp*)

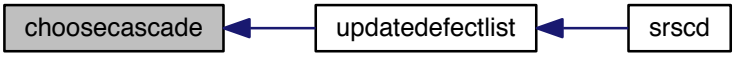
Subroutine: Choose Cascade Takes list of cascades (read from input file) and chooses one randomly.

Inputs: CascadeList (global variable) Output: CascadeTemp (pointing at the cascade we want)

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.1.6 subroutine createcascadeconnectivity ( )

subroutine createCascadeConnectivity()

This subroutine assigns values to the connectivity matrix (global variable) used for all cascades

Input: numxcascade, numycascade, nunmzcascade (global variables) : from parameters.txt Output: cascade↩ Connectivity (global variable)

Here is the caller graph for this function:

10.2 CoarseMesh_subroutines.f90 File Reference

Functions/Subroutines

- subroutine finddefectinlist (defectCurrent, defectPrev, products)

  Subroutine Find defect in list - points defectCurrent at the appropriate defect in a linked list.
- integer function findnumdefect (defectType, cellNumber)

  Subroutine find number of defects - finds the number of defects of a given type inside a certain volume element in the local mesh.
- integer function findnumdefectboundary (defectType, cellNumber, dir)

  Subroutine find number of defects boundary - finds the number of defects of a given type inside a certain volume element in the boundary mesh.
- subroutine countreactionscoarse (reactionsCoarse)

  subroutine countReactionsCoarse(reactionsCoarse) - counts number of reactions in the coarse mesh
- subroutine updateimplantratesinglecell (cell)

  Subroutine updateImplantRateSingleCell(cell) - updates defect implantation rates due to change in volume.
- subroutine resetreactionlistsinglecell (cell)

  subroutine resetReactionListSingleCell(cell) - resets an entire reaction list in a single volume element
- subroutine clearreactionlistsinglecell (cellNumber)

  Subroutine clearReactionListSingleCell(cellNumber) - clears reactions from a reaction list in a single volume element.

10.2.1 Function/Subroutine Documentation

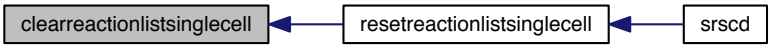
10.2.1.1 subroutine clearreactionlistsinglecell (integer *cellNumber*)

Subroutine clearReactionListSingleCell(cellNumber) - clears reactions from a reaction list in a single volume element.

Clears all reactions from a reaction list in a cell, so that these reactions can be reset (used for cascade implantation/removal when all reactions in a coarse mesh element need to be reset)

Input: cellNumber Output: none Actions: clears reactions in reactionList(cellNumber)

Here is the caller graph for this function:



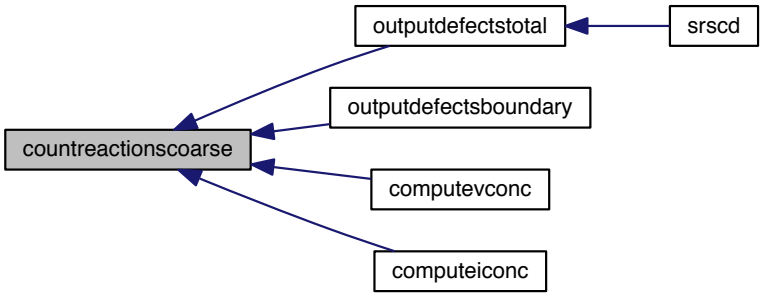
10.2.1.2 subroutine countreactionscoarse (integer *reactionsCoarse*)

subroutine countReactionsCoarse(reactionsCoarse) - counts number of reactions in the coarse mesh

Counts the total number of reactions in the coarse mesh (all processors) and returns the sum.

Inputs: none Output: reactionsCoarse, total number of reactions in all coarse mesh elements including all processors

Here is the caller graph for this function:



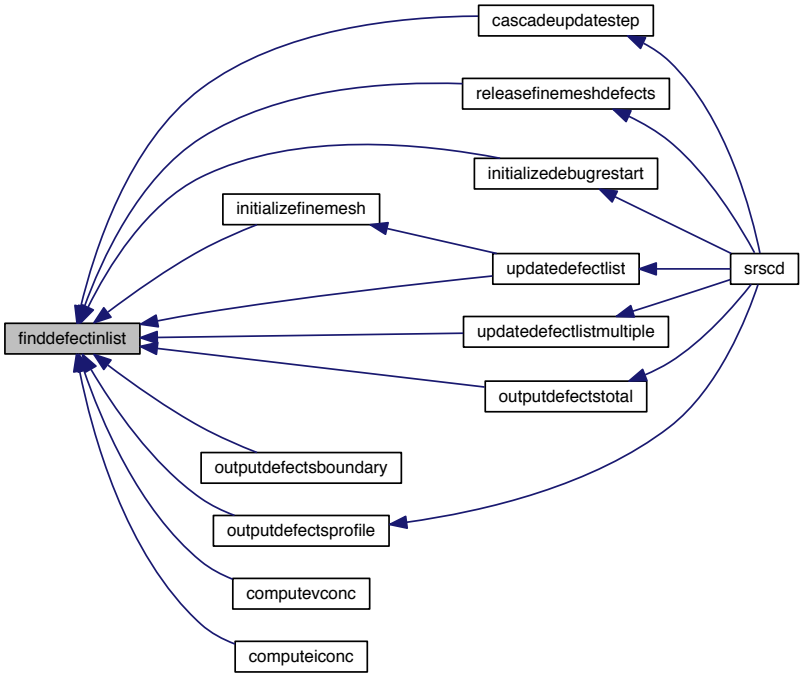
10.2.1.3 subroutine finddefectinlist (type(defect), pointer *defectCurrent,* type(defect), pointer *defectPrev,* integer, dimension(numspecies) *products*)

Subroutine Find defect in list - points defectCurrent at the appropriate defect in a linked list.

This subroutine places defectCurrent on the defect being added to the system if it is already in the system If not, defectCurrent points to the place after the inserted defect and defectPrev points to the place before the inserted defect Ordering: first by He content, then by V content, then by interstitial content

Inputs: products Outputs: defectCurrent and defectPrev pointed at location in list

Here is the caller graph for this function:



10.2.1.4 integer function findnumdefect (integer, dimension(numspecies) *defectType,* integer *cellNumber*)

Subroutine find number of defects - finds the number of defects of a given type inside a certain volume element in the local mesh.

This subroutine searches through a linked list for the a defect of a given type and returns the number of defects of that type present in volume element (cellNumber)

Inputs: defectType, cellNumber Outputs: returns number of defects of type in cell

10.2.1.5 integer function findnumdefectboundary (integer, dimension(numspecies) *defectType,* integer *cellNumber,* integer *dir*)

Subroutine find number of defects boundary - finds the number of defects of a given type inside a certain volume element in the boundary mesh.

This subroutine searches through a linked list for the a defect of a given type and returns the number of defects of that type present in volume element (cellNumber) - in the boundary mesh

Inputs: defectType, cellNumber, direction (used to identify correct element of myBoundary) Outputs: returns number of defects of type in cell

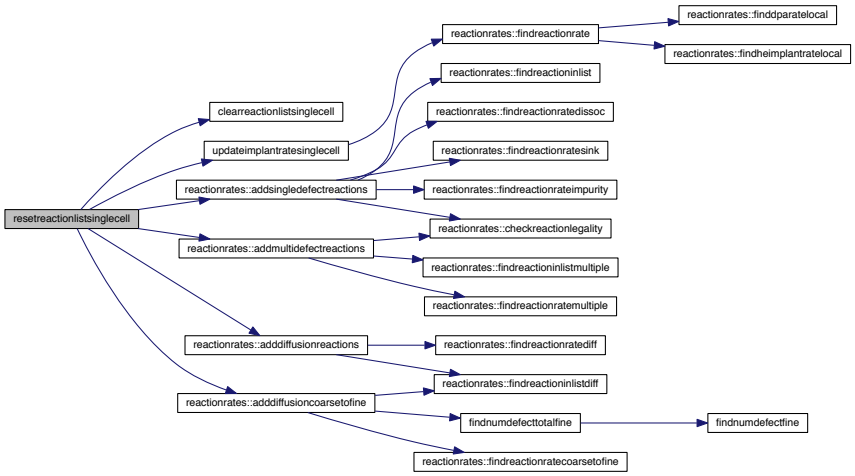
10.2.1.6 subroutine resetreactionlistsinglecell (integer *cell*)

subroutine resetReactionListSingleCell(cell) - resets an entire reaction list in a single volume element

Resets the reaction list for all reactions within a single cell (used when a cascade is created or deleted within that cell, all reaction rates change because volume changes)

Input: cell (integer): cell number Outputs: reaction rates for all reactions in a cell

Here is the call graph for this function:



Here is the caller graph for this function:

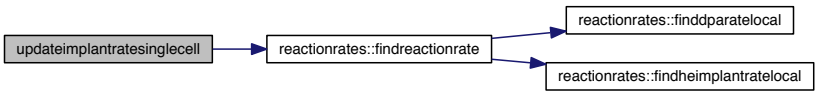
10.2.1.7 subroutine updateimplantratesinglecell (integer *cell*)

Subroutine updateImplantRateSingleCell(cell) - updates defect implantation rates due to change in volume.

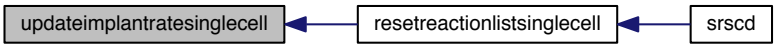
This subroutine updates the implantation rate in a coarse mesh element due to the change in volume from cascade implantation / deletion

Input: cell (integer, coarse mesh element id number) Output: updates cascade implantation rate

Here is the call graph for this function:



Here is the caller graph for this function:



10.3 Deallocate_Lists.f90 File Reference

Functions/Subroutines

- subroutine deallocateboundarydefectlist ()

  Subroutine deallocateBoundaryDefectList() - deallocates boundary defect lists.
- subroutine deallocatecascadelist ()

  Subroutine deallocateCascadeList() - deallocates all stored cascade data (read in from file) in cascade list.
- subroutine deallocatematerialinput ()

  Subroutine deallocateMaterialInput() - deallocates material input data (binding and migration energies, etc)
- subroutine deallocatedefectlist ()

  Subroutine deallocateDefectList() - deallocates defects in the coarse mesh.
- subroutine deallocatereactionlist ()

  Subroutine deallocateReactionList() - deallocates reactions in the coarse mesh.

10.3.1 Function/Subroutine Documentation

10.3.1.1 subroutine deallocateboundarydefectlist ( )

Subroutine deallocateBoundaryDefectList() - deallocates boundary defect lists.

This subroutine deallocates all the defects in the boundary mesh at the end of the simulation

Inputs: none Outputs: none Action: goes through boundary mesh and deallocates all defects

Here is the caller graph for this function:

10.3.1.2 subroutine deallocatecascadelist ( )

Subroutine deallocateCascadeList() - deallocates all stored cascade data (read in from file) in cascade list.

This subroutine deallocates all the cascades in the cascade list at the end of the simulation.

Inputs: none Outputs: none Action: goes through each cascade list and deallocates all cascades (note: these are the lists used as inputs, not the fine mesh defects)

Here is the caller graph for this function:

10.3.1.3 subroutine deallocatedefectlist ( )

Subroutine deallocateDefectList() - deallocates defects in the coarse mesh.

This subroutine deallocates all the defects in the coarse mesh at the end of the simulation.

Inputs: none Outputs: none Action: goes through each defect list for each volume element and deallocates all defects

Here is the caller graph for this function:

10.3.1.4 subroutine deallocatematerialinput ( )

Subroutine deallocateMaterialInput() - deallocates material input data (binding and migration energies, etc)

This subroutine deallocates global variables used for calculating allowed reactions:

type(diffusionFunction), allocatable :: DiffFunc(:)

type(diffusionSingle), allocatable :: DiffSingle(:)

type(bindingSingle), allocatable :: BindSingle(:)

type(bindingFunction), allocatable :: BindFunc(:)

type(reactionParameters), allocatable :: DissocReactions(:), DiffReactions(:), SinkReactions(:)

type(reactionParameters), allocatable :: ImpurityReactions(:), ClusterReactions(:), ImplantReactions(:)

Here is the caller graph for this function:

10.3.1.5 subroutine deallocatereactionlist ( )

Subroutine deallocateReactionList() - deallocates reactions in the coarse mesh.

This subroutine deallocates all the reactions in the coarse mesh at the end of the simulation.

Inputs: none Outputs: none Action: goes through each reaction list for each volume element and deallocates all reactions

Here is the caller graph for this function:

10.4 Debug_subroutines.f90 File Reference

Functions/Subroutines

- subroutine debugcheckforunadmissible (reactionCurrent, step)

  subroutine DEBUGcheckForUnadmissibleDefects(reactionCurrent) - checks to see if any defects are not allowed
- subroutine debugprintdefectupdate (defectUpdate)

  Subroutine DEBUG Print Defect Update - prints defectUpdate, which is used to identify which defects have been updated (used to update reaction list)
- subroutine debugprintreactionlist (step)

  Subroutine DEBUG Print reaction list - prints all reaction lists at a given Monte Carlo step.
- subroutine debugprintdefects (step)

  Subroutine Debug print defects - prints all defects in all volume elements.
- subroutine debugprintreaction (reactionCurrent, step)

  Subroutine debug print reaction - outputs the reaction chosen at a given step in the master processor. Used for debugging.

10.4.1 Function/Subroutine Documentation

10.4.1.1 subroutine debugcheckforunadmissible (type(reaction), pointer *reactionCurrent,* integer *step*)

subroutine DEBUGcheckForUnadmissibleDefects(reactionCurrent) - checks to see if any defects are not allowed

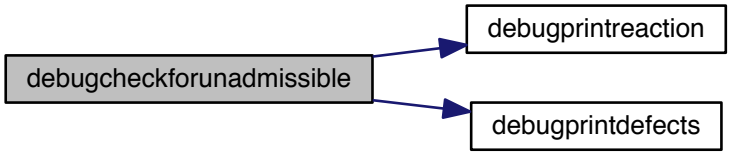
This subroutine peruses the defect list and cascade defect lists and checks to see if any defects are unadmissible (hard coded information on what constitutes an unadmissible defect; in this case we are concerned about defect↩ Type= 5 10 150 3 for example.

If an unadmissible defect is found, the subroutine outputs the defect list and the reaction that was chosen at that step.

INPUT: reactionCurrent, used to display the reaction that was chosen at this step if needed step, used to display what step we are on

OUTPUT: unadmissible defects, if any (on the screen)

Here is the call graph for this function:



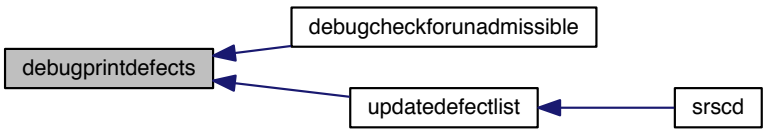
10.4.1.2 subroutine debugprintdefects (integer *step*)

Subroutine Debug print defects - prints all defects in all volume elements.

This subroutine prints all defects in the coarse mesh and / or the cascades in the master processor only for the purposes of debugging the code.

Have the option to skip volume elements that are empty.

Here is the caller graph for this function:



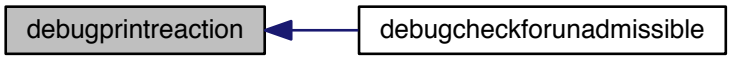
10.4.1.3 subroutine debugprintdefectupdate (type(defectupdatetracker), pointer *defectUpdate*)

Subroutine DEBUG Print Defect Update - prints defectUpdate, which is used to identify which defects have been updated (used to update reaction list)

10.4.1.4 subroutine debugprintreaction (type(reaction), pointer *reactionCurrent,* integer *step*)

Subroutine debug print reaction - outputs the reaction chosen at a given step in the master processor. Used for debugging.

Here is the caller graph for this function:



10.4.1.5 subroutine debugprintreactionlist (integer *step*)

Subroutine DEBUG Print reaction list - prints all reaction lists at a given Monte Carlo step.

Prints all reactions in each volume element (using linked reaction list). Have the option to ignore all reactions with rate=0, or to ignore all diffusion reactions, or to ignore implantation reactions (depending on what you are trying to debug)

Prints reaction lists in coarse mesh as well as in cascades, if any are present

10.5 Defect_attributes.f90 File Reference

Functions/Subroutines

- double precision function finddiffusivity (matNum, DefectType)

  double precision function find diffusivity - returns the diffusivity of a given defect type
- double precision function diffusivitycompute (DefectType, functionType, numParameters, parameters)

  double precision function diffusivityCompute - computes diffusivity using a functional form for defects that don't have their diffusivity given by a value in a list.
- double precision function findbinding (matNum, DefectType, productType)

  double precision function find binding - returns the binding energy of a given defect type
- double precision function bindingcompute (DefectType, product, functionType, numParameters, parameters)

  double precision function bindingCompute - computes binding energy using a functional form for defects that don't have their binding energy given by a value in a list.
- integer function finddefectsize (defectType)

  integer function findDefectSize - returns the size of a cluster
- double precision function findstrainenergy (defectType, cell)

  double precision function findStrainEnergy - Returns the interaction energy of the defect with the local strain field
- double precision function findstrainenergyboundary (defectType, dir, cell)

  double precision function findStrainEnergyBoundary - Returns the interaction energy of the defect with the local strain field in a boundary element

10.5.1 Function/Subroutine Documentation

10.5.1.1 double precision function bindingcompute (integer, dimension(numspecies) *DefectType,* integer, dimension(numspecies) *product,* integer *functionType,* integer *numParameters,* double precision, dimension(numparameters) *parameters*)

double precision function bindingCompute - computes binding energy using a functional form for defects that don't have their binding energy given by a value in a list.

This function has several hard-coded functional forms for binding energy, including vacancy clusters, SIA clusters, He/V clusters, and the activation energy for a sessile-glissile SIA loop transformation

10.5.1.2 double precision function diffusivitycompute (integer, dimension(numspecies) *DefectType,* integer *functionType,* integer *numParameters,* double precision, dimension(numparameters) *parameters*)

double precision function diffusivityCompute - computes diffusivity using a functional form for defects that don't have their diffusivity given by a value in a list.

This function has several hard-coded functional forms for diffusivity, including immobile defects, constant functions, and mobile SIA loops. Additional functional forms can be added as needed.

10.5.1.3 double precision function findbinding (integer *matNum,* integer, dimension(numspecies) *DefectType,* integer, dimension(numspecies) *productType*)

double precision function find binding - returns the binding energy of a given defect type

This function looks up the binding energy of a given defect type using input data from material parameter input file. It either looks up binding energy from values in a list or computes binding energy using bindingCompute in the case of a functional form.

Input: defect type, product type (what type of defect dissociates from the cluster) Output: binding energy (eV)

10.5.1.4 integer function finddefectsize (integer, dimension(numspecies) *defectType*)

integer function findDefectSize - returns the size of a cluster

This function will find the effective size of the defect (hard-coded information), used for determining the radius of the defect (for dissociation and clustering reactions). It returns n, the number of lattice spaces taken up by this defect.

NOTE: for He_nV_m clusters, this function returns the larger of m or n

10.5.1.5 double precision function finddiffusivity (integer *matNum,* integer, dimension(numspecies) *DefectType*)

double precision function find diffusivity - returns the diffusivity of a given defect type

This function looks up the diffusivity of a given defect type using input date from material parameter input file. It either looks up diffusivity from values in a list or computes diffusivity using diffusivityCompute in the case of a functional form.

Input: defect type Output: diffusivity (nm^2/s)

10.5.1.6 double precision function findstrainenergy (integer, dimension(numspecies) *defectType,* integer *cell*)

double precision function findStrainEnergy - Returns the interaction energy of the defect with the local strain field

This function takes the double dot product of the defect's dipole tensor with the local strain field and returns that amount (energy, in eV).

NOTE: if the dipole tensor is asymmetric, an averaged strain energy is taken which accounts for all possible orientations of the defect. It is not known if this is the correct averaging procedure that should be used here.

10.5.1.7 double precision function findstrainenergyboundary (integer, dimension(numspecies) *defectType,* integer *dir,* integer *cell*)

double precision function findStrainEnergyBoundary - Returns the interaction energy of the defect with the local strain field in a boundary element

This function takes the double dot product of the defect's dipole tensor with the local strain field and returns that amount (energy, in eV).

NOTE: if the dipole tensor is asymmetric, an averaged strain energy is taken which accounts for all possible orientations of the defect. It is not known if this is the correct averaging procedure that should be used here.

10.6 dprand.f90 File Reference

Modules

- module randdp

  Module randdp: double precision random number generation.

Functions/Subroutines

- subroutine randdp::sdprnd (iseed)

  Subroutine sdprnd - seeds random number generator in this processor using integer input.
- real(dp) function randdp::dprand ()

  Function dprand - generates a double precision random number between 0 and 1.

Variables

- integer, parameter, public randdp::dp = SELECTED_REAL_KIND(15, 60)
- real(dp), dimension(101), save, public randdp::poly
- real(dp), save, public randdp::other
- real(dp), save, public randdp::offset
- integer, save, public randdp::index

10.7 FineMesh_subroutines.f90 File Reference

Functions/Subroutines

- subroutine releasefinemeshdefects (CascadeCurrent)

  Function releaseFineMeshDefects(CascadeCurrent) - releases fine mesh back to coarse mesh when cascade is annealed.
- integer function findnumdefectfine (CascadeCurrent, defectType, cellNumber)

  function findNumDefectFine - finds the number of defects of a certain type in a volume element in the fine mesh
- integer function findnumdefecttotalfine (defectType, CascadeCurrent)

  function findNumDefectTotalFine - finds the total number of defects of a type in all fine mesh elements
- integer function findcellwithcoordinatesfinemesh (coordinates)

  integer function findCellWithCoordinatesFineMesh(coordinates) - finds a cell with local coordinates in the fine mesh
- integer function chooserandomcell ()

  Function chooseRandomCell() - chooses a random fine mesh volume element ID.
- subroutine countreactionsfine (reactionsFine)

  subroutine countReactionsFine(reactionsFine) - counts reactions in the fine mesh

10.7.1 Function/Subroutine Documentation

10.7.1.1 integer function chooserandomcell ( )

Function chooseRandomCell() - chooses a random fine mesh volume element ID.

Chooses a cell number at random from the fine mesh. (Uniform distribution)

Inputs: none (just mod_srscd_constants values such as the number of cells) Outputs: cell number

Here is the call graph for this function:

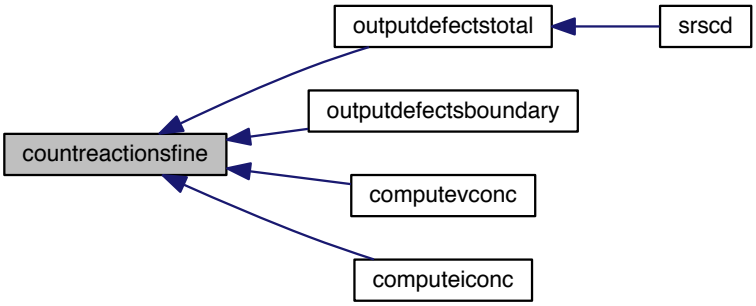
10.7.1.2 subroutine countreactionsfine (integer *reactionsFine*)

subroutine countReactionsFine(reactionsFine) - counts reactions in the fine mesh

Counts the total number of reactions in the fine mesh (all processors) and returns the sum (used for postprocessing)

Inputs: none Output: reactionsfine, total number of reactions in all fine mesh elements including all processors (and all cascades)

Here is the caller graph for this function:



10.7.1.3 integer function findcellwithcoordinatesfinemesh (double precision, dimension(3) *coordinates*)

integer function findCellWithCoordinatesFineMesh(coordinates) - finds a cell with local coordinates in the fine mesh

FindCellWithCoordinates returns the cell id number that contains the coordinates given in the double precision variable coordinates (for defect implantation in fine mesh at beginning of cascade)

Inputs: coordinates(3) (double precision) Output: fine mesh cell id number (according to standard cubic meshing connectivity scheme)

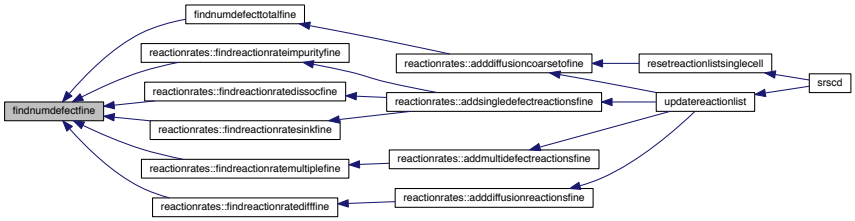
10.7.1.4 integer function findnumdefectfine (type(cascade), pointer *CascadeCurrent,* integer, dimension(numspecies) *defectType,* integer *cellNumber*)

function findNumDefectFine - finds the number of defects of a certain type in a volume element in the fine mesh

Finds the number of defects of type defectType in CascadeCurrentlocalDefects. If none, returns 0

Inputs: CascadeCurrent (pointer), defectType(numSpecies), cellNumber Output: numDefects

Here is the caller graph for this function:



10.7.1.5 integer function findnumdefecttotalfine (integer, dimension(numspecies) *defectType,* type(cascade), pointer *CascadeCurrent*)

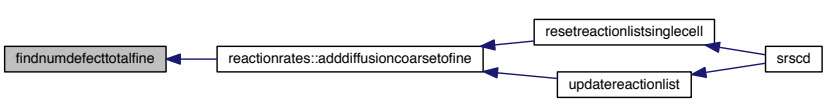
function findNumDefectTotalFine - finds the total number of defects of a type in all fine mesh elements

Finds the number of defects of type defectType in CascadeCurrentlocalDefects. If none, returns 0

Inputs: CascadeCurrent (pointer), defectType(numSpecies), cellNumber Output: numDefects

Here is the call graph for this function:

Here is the caller graph for this function:



10.7.1.6 subroutine releasefinemeshdefects (type(cascade), pointer *CascadeCurrent*)

Function releaseFineMeshDefects(CascadeCurrent) - releases fine mesh back to coarse mesh when cascade is annealed.

This function deletes the fine mesh and deposits all defects into the coarse mesh cell containing the fine mesh.

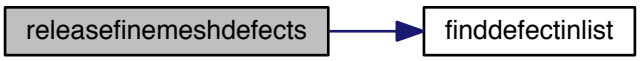
This function also restores the coarse mesh volume element's volume by the cascade volume

Inputs: CascadeCurrent (contains defects and reaction lists, as well as identifying info. such as coarse mesh volume element, etc.

Outputs: none

Actions: defects are deposited in coarse mesh, memory is deallocated and the previous cascade is pointed towards the next cascade.

Here is the call graph for this function:



Here is the caller graph for this function:

10.8 Initialization_subroutines.f90 File Reference

Functions/Subroutines

- subroutine initializedefectlist ()

  Subroutine initialize defect list - initializes defect list for each volume element.
- subroutine initializerandomseeds ()

  Subroutine initialize random seeds - seeds random number generator differently for each processor.
- subroutine initializetotalrate ()

  Subroutine initialize total rate - finds the total reaction rate in the local processor once reaction lists have been initialized (including implantation reactions)
- subroutine initializereactionlist ()

  Subroutine initialize reaction list - creates a new reaction list for each volume element and initializes implantation reactions (with rates)
- subroutine initializedebugrestart ()

  Subroutine initializeDebugRestart() - initializes defect and reaction lists if we are restarting from debug file.
- subroutine initializeboundarydefectlist ()

  Subroutine initialize boundary defect list - initializes defect lists in the boundary mesh.
- subroutine initializefinemesh (CascadeCurrent)

  Subroutine: Initialize Fine Mesh - initializes defect and reaction lists in a newly created fine mesh.

- subroutine initializemesh ()

  Subroutine initializeMesh() - begins the mesh initialization process at the beginning of the program.
- subroutine annealinitialization ()

  Subroutine annealInitialization() - changes implantation rates and temperature for annealing.

10.8.1 Function/Subroutine Documentation

10.8.1.1 subroutine annealinitialization ( )

Subroutine annealInitialization() - changes implantation rates and temperature for annealing.

This subroutine carries out the following tasks in order to switch from damage introduction to annealing:

Temperature changed to annealTemp; Damage rate changed to 0; He implant rate changed to 0

Inputs: none (uses global variable annealTemp) Outputs: none Actions: prepares system for annealing

Here is the caller graph for this function:

10.8.1.2 subroutine initializeboundarydefectlist ( )

Subroutine initialize boundary defect list - initializes defect lists in the boundary mesh.

This subroutine initializes the defect lists within the boundary mesh.

NOTE: it is currently not set up to handle situaions where myMesh(cell)neighborProcs(dir,k) .NE. myProcproc↩ Neighbor(dir) (AKA when the element to the left is a different proc than the proc to the left, due to uneven meshing) This may cause errors when the non-uniform mesh is used.

Here is the caller graph for this function:

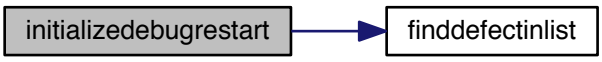
10.8.1.3 subroutine initializedebugrestart ( )

Subroutine initializeDebugRestart() - initializes defect and reaction lists if we are restarting from debug file.

This subroutine is used to populate the mesh with defects and initialize the DPA at a non-zero value for debugging (restart from a saved point).

This subroutine reads in from an input file. The mesh must match the mesh in the actual simulation. The number of processors must also match from the reset file and the current simulation.

Here is the call graph for this function:



Here is the caller graph for this function:

10.8.1.4 subroutine initializedefectlist ( )

Subroutine initialize defect list - initializes defect list for each volume element.

Begins with a defect with type 0 0 0 0 and num 0. Note that numCells is needed in this subroutine

Here is the caller graph for this function:

10.8.1.5 subroutine initializefinemesh (type(cascade), pointer *CascadeCurrent*)

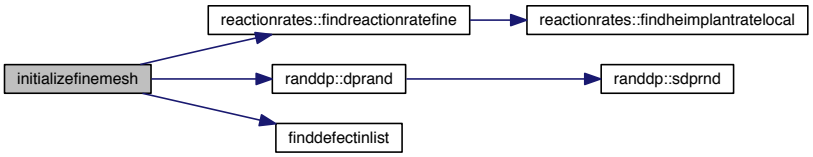
Subroutine: Initialize Fine Mesh - initializes defect and reaction lists in a newly created fine mesh.

Allocates the size of the fine mesh (fine defect list and reaction list included as part of the cascade derived type), populates with defects from the coarse mesh and removes those defects from the coarse mesh.

This subroutine includes cascade defect interaction with pre-existing defects.

Inputs: CascadeCurrent (cascade type derived type variable) Output: CascadeCurrent (initialized with defects from coarse mesh)

Here is the call graph for this function:



Here is the caller graph for this function:

10.8.1.6 subroutine initializemesh ( )

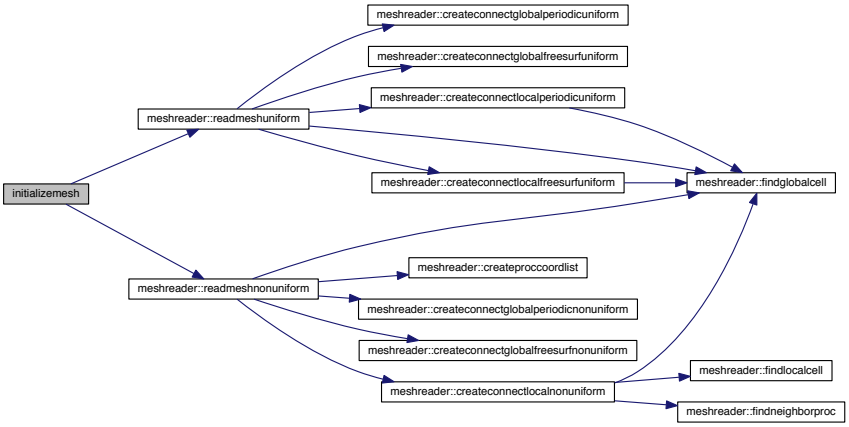
Subroutine initializeMesh() - begins the mesh initialization process at the beginning of the program.

This subroutine reads the name of the mesh file from the central input file (parameters.txt) and sends it to read↩ MeshUniform() or readMeshNonUniform().

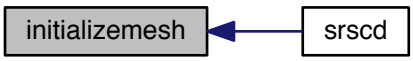
NOTE: although the mesh file and creation of mesh files and connectivity differ for uniform/nonuniform meshes, the format of the final global variable created (class mesh, myMesh, in mod_srscd_constants) is the same for both readMeshUniform and readMeshNonUniform. Thus the rest of the program can use it either way.

Debug tool: if debugRestart=='yes', then we populate the mesh with the defects in the debug file. This allows us to start the simulation further along than the beginning of the simulation.

Here is the call graph for this function:



Here is the caller graph for this function:

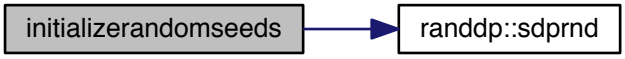


10.8.1.7 subroutine initializerandomseeds ( )

Subroutine initialize random seeds - seeds random number generator differently for each processor.

Uses the computer clock to initialize the random seed of the master processor. Then generates several integers in the master processor and uses them to initialize the random number seed of the other processors.

Here is the call graph for this function:



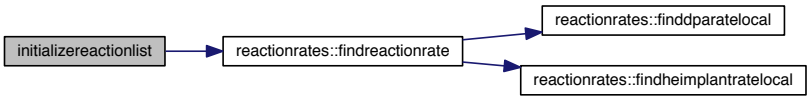
Here is the caller graph for this function:

10.8.1.8 subroutine initializereactionlist ( )

Subroutine initialize reaction list - creates a new reaction list for each volume element and initializes implantation reactions (with rates)

First reaction in list is either Frenkel pair implantation or cascade implantation, and second reaction in the list is helium implantation.

Here is the call graph for this function:



Here is the caller graph for this function:

10.8.1.9 subroutine initializetotalrate ( )

Subroutine initialize total rate - finds the total reaction rate in the local processor once reaction lists have been initialized (including implantation reactions)

This will also initialize the total rate in the case of a debug restart, in which many reactions are possible at the first step (besides just implantation reactions)

Here is the caller graph for this function:

10.9 kMC_subroutines.f90 File Reference

Functions/Subroutines

- double precision function generatetimestep ()

  double precision generate timestep - chooses a timestep using random number and Monte Carlo algorithm (this is a global timestep)
- subroutine choosereaction (reactionCurrent, CascadeCurrent)

  subroutine choose reaction - chooses a reaction in each processor according to the Monte Carlo algorithm (this is a local reaction)
- subroutine choosereactionsinglecell (reactionCurrent, CascadeCurrent, cell)

subroutine choose reaction - chooses a reaction in each processor according to the Monte Carlo algorithm (this is a local reaction)

- subroutine updatedefectlist (reactionCurrent, defectUpdateCurrent, CascadeCurrent)

  Subroutine update defect list - updates defects according to reaction chosen.

- subroutine updatedefectlistmultiple (reactionCurrent, defectUpdateCurrent, CascadeCurrent)

  Subroutine update defect list multiple - updates defects according to reactions chosen.

- subroutine updatereactionlist (defectUpdate)

  Subroutine update reaction list - updates reaction rates and creates/deletes reactions according to defect numbers that have changed:

10.9.1 Function/Subroutine Documentation

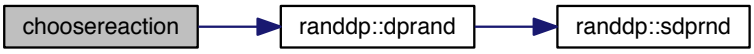
10.9.1.1 subroutine choosereaction (type(reaction), pointer *reactionCurrent,* type(cascade), pointer *CascadeCurrent*)

subroutine choose reaction - chooses a reaction in each processor according to the Monte Carlo algorithm (this is a local reaction)

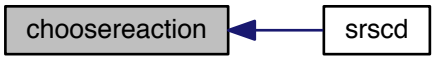
NOTE: to increase computation speed, need these lists to be ordered.

NOTE: to increase computation speed, choose an element first and then choose a reaction within that element.

Here is the call graph for this function:



Here is the caller graph for this function:



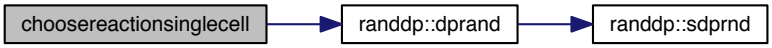
10.9.1.2 subroutine choosereactionsinglecell (type(reaction), pointer *reactionCurrent,* type(cascade), pointer *CascadeCurrent,* integer *cell*)

subroutine choose reaction - chooses a reaction in each processor according to the Monte Carlo algorithm (this is a local reaction)

NOTE: to increase computation speed, need these lists to be ordered.

NOTE: to increase computation speed, choose an element first and then choose a reaction within that element.

Here is the call graph for this function:

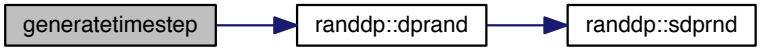


Here is the caller graph for this function:

10.9.1.3 double precision function generatetimestep ( )

double precision generate timestep - chooses a timestep using random number and Monte Carlo algorithm (this is a global timestep)

Here is the call graph for this function:

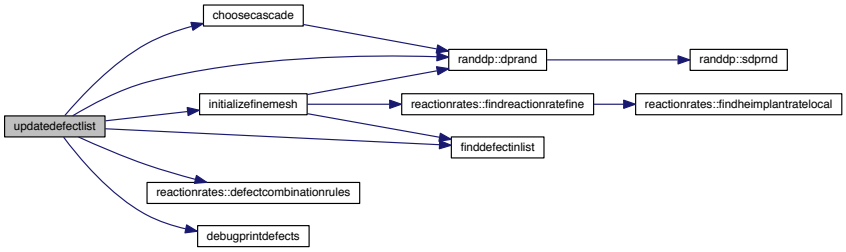


10.9.1.4 subroutine updatedefectlist (type(reaction), pointer *reactionCurrent,* type(defectupdatetracker), pointer *defectUpdateCurrent,* type(cascade), pointer *CascadeCurrent*)

Subroutine update defect list - updates defects according to reaction chosen.

This is the most involved and complex subroutine of the entire SRSCD algorithm. This subroutine updates defects in the local mesh according to the reaction chosen, and communicates with neighboring processors about defects that may have passed into a different processor as well as about defects that may have changed on the boundary. It also creates a list of defects that have been updated, to inform the next subroutine which reactions to update.

Here is the call graph for this function:



Here is the caller graph for this function:

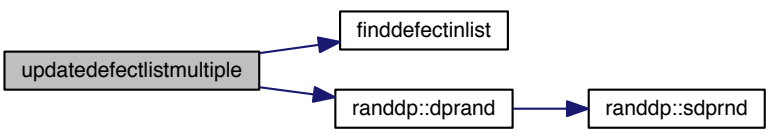
10.9.1.5 subroutine updatedefectlistmultiple (type(reaction), pointer *reactionCurrent,* type(defectupdatetracker), pointer *defectUpdateCurrent,* type(cascade), pointer *CascadeCurrent*)

Subroutine update defect list multiple - updates defects according to reactions chosen.

This is the most involved and complex subroutine of the entire SRSCD algorithm. This subroutine updates defects in the local mesh according to the reaction chosen, and communicates with neighboring processors about defects that may have passed into a different processor as well as about defects that may have changed on the boundary. It also creates a list of defects that have been updated, to inform the next subroutine which reactions to update.

This version of updateDefectList has been created for the case of one KMC domain per volume element, in which many reactions will be chosen during each step. Thus, we have to go through the list pointed to by reactionCurrent rather than just that one reaction.

Here is the call graph for this function:



Here is the caller graph for this function:

10.9.1.6 subroutine updatereactionlist (type(defectupdatetracker), pointer *defectUpdate*)

Subroutine update reaction list - updates reaction rates and creates/deletes reactions according to defect numbers that have changed:

This subroutine does the following:

1) Find list of affected volume elements (and their procs) - separate into list of neighbors and list of elements involved in reaction

1a) Make list of all defect types in each volume element that need to be checked

2) Delete all diffusion reactions associated with reactants and products in neighboring volume elements

3) Delete all reactions associated with reactants and products in volume elements associated with reaction

4) Add all diffusion reactions associated with reactants and products in neighboring volume elements

5) Add all reactions associated with reactants and prodcuts in volume elements associated with reaction

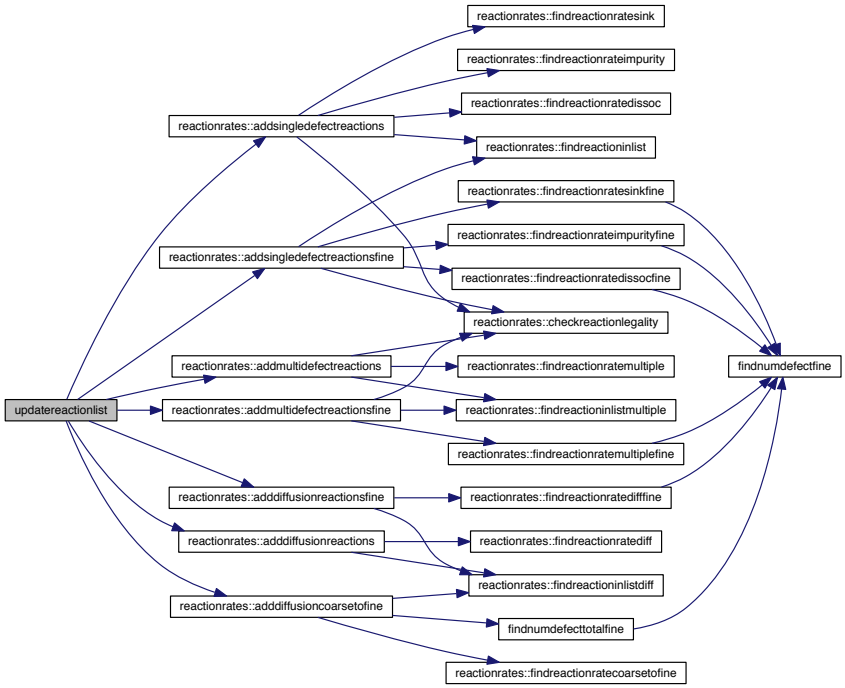
In steps 2-5, subtract/add to totalRate each time we add or subtract a reaction from the list

EDIT: it may be faster computationally to not delete and re-add reactions, but to just update reaction rates for all appropriate reactions, delete if 0, and add if non-existent.

In this way, we are ONLY updating the relevant reaction rates, not all possible reaction rates in the system.

As reaction rates are updated, the total reaction rate for the processor is updated as well.

Here is the call graph for this function:



Here is the caller graph for this function:

10.10 MeshReader.f90 File Reference

Modules

- module meshreader

  Module Mesh Reader: creates a processor mesh and volume element mesh using input from file.

Functions/Subroutines

- subroutine meshreader::readmeshuniform (filename)

  Subroutine read Mesh Uniform - creates processor and mesh files from uniform mesh input.
- subroutine meshreader::createconnectglobalperiodicuniform (connectivity, numx, numy, numz)

  Subroutine create global connectivity (uniform mesh, periodic boundary conditions)
- subroutine meshreader::createconnectglobalfreesurfuniform (connectivity, numx, numy, numz)

Subroutine create global connectivity (uniform mesh, free surfaces in z-directions and periodic boundary conditions in other directions)

- subroutine meshreader::createconnectlocalperiodicuniform (numx, numy, numz, globalMeshCoord, global↩ MeshConnect)

  Subroutine create local connectivity (uniform mesh, periodic boundary conditions)
- subroutine meshreader::createconnectlocalfreesurfuniform (numx, numy, numz, globalMeshCoord, global↩ MeshConnect)

  Subroutine create local connectivity (uniform mesh, free surfaces in z-direction and periodic boundary conditions in other directions)
- integer function meshreader::findglobalcell (coord, gCoord)

  Subroutine find Global Cell.
- subroutine meshreader::readmeshnonuniform (filename)

  Subroutine read Mesh Non Uniform - creates processor and mesh files from non-uniform mesh input.
- subroutine meshreader::createconnectglobalperiodicnonuniform (Connect, Coord, NumNeighbors, Length, numTotal, bdryCoord)

  Subroutine create global connectivity (non-uniform mesh, periodic boundary conditions)
- subroutine meshreader::createconnectglobalfreesurfnonuniform (Connect, Coord, NumNeighbors, Length, numTotal, bdryCoord)

  Subroutine create global connectivity (non-uniform mesh, free surfaces in the z-direction and periodic boundary conditions in other directions)
- subroutine meshreader::createconnectlocalnonuniform (globalMeshConnect, globalMeshCoord, localElem, procCoordList)

  Subroutine create local connectivity (non-uniform mesh, either free surfaces or periodic bc's)
- integer function meshreader::findlocalcell (coord)

  Function find local cell.
- subroutine meshreader::createproccoordlist (procCoordList, procDivision)

  subroutine create processor coordinates list
- integer function meshreader::findneighborproc (globalMeshCoord, procCoordList, elem)

  Function find neighboring processor.

10.11 Misc_functions.f90 File Reference

Functions/Subroutines

- integer function factorial (n)

  Subroutine factorial Factorial function returns n!
- integer function binomial (n, k)

  Subroutine binomial Binomial function returns n choose k.
- double precision function totalratecheck ()

  function TotalRateCheck() - checks if our total rate still matches the actual total rate
- double precision function totalratecascade (CascadeCurrent)

  Function total rate cascade - finds total reaction rate within cascade.

10.11.1 Function/Subroutine Documentation

10.11.1.1 integer function binomial (integer *n*, integer *k*)

Subroutine binomial Binomial function returns n choose k.

Inputs: n, k (integers) Outputs: Binomial (integer)

10.11.1.2 integer function factorial (integer *n*)

Subroutine factorial Factorial function returns n!

Inputs: n (integer) Outputs: factorial (integer)

This is limited to numbers n<17 for computer feasibility.

10.11.1.3 double precision function totalratecascade (type(cascade), pointer *CascadeCurrent*)

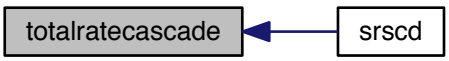
Function total rate cascade - finds total reaction rate within cascade.

Input: CascadeCurrenttotalRate(:), list of reaction rates within each volume element in the cascade

Output: sum of CascadeCurrenttotalRate(:), total reaction rate for entire cascade

This subroutine was added on 2015.04.03 when I switched to tracking the reaction rate in each volume element instead of the entire cascade, for the sake of computational efficiency when choosing reactions.

Here is the caller graph for this function:



10.11.1.4 double precision function totalratecheck ( )

function TotalRateCheck() - checks if our total rate still matches the actual total rate

This function is used as a diagnostic tool, it calcuates the total rate of all reactions in the system and compares it to totalRate, in order to test whether totalRate is being properly updated.

Inputs: none Output: total reaction rate (sum of all reaction rates)

10.12 mod_derivedtypes.f90 File Reference

Data Types

- type derivedtype::defect

  Type: defect (pointer list).
- type derivedtype::defectupdatetracker

  Type: defect update tracker (pointer list)
- type derivedtype::reaction

  Type: reaction (pointer list)
- type derivedtype::processordata

  Type: processorData (one variable per processor)
- type derivedtype::mesh

  Type: mesh (local, one array of this type per processor)
- type derivedtype::boundarymesh

  Type: boundary mesh (one array of this type per processor)

- type derivedtype::diffusionfunction

  Type: diffusion function (list of functional forms for defect diffusivities read in from input file)
- type derivedtype::diffusionsingle

  Type: diffusion single (list of single migration energies and diffusion prefactors)
- type derivedtype::bindingsingle

  Type: binding single (list of binding energies)
- type derivedtype::bindingfunction

  Type: binding function (list of functional forms for defect binding energies read in from input file)
- type derivedtype::reactionparameters

  Type: reaction parameters (list of allowed reactions, input from file)
- type derivedtype::dipoletensor

  Type: dipole tensor (stored dipole tensor data for various defect types, input from file)
- type derivedtype::cascadeevent

  Type: cascade event (pointer list of read-in cascade data from input file)
- type derivedtype::cascadedefect

  Type: cascade defect (pointer list of defect types in a cascade)
- type derivedtype::cascade

  Type: cascade (pointer list containing defects and reactions for cascades that are currently present in the system)

Modules

- module derivedtype

  mod_DerivedTypes contains the variable derived types created for SRSCD.

10.13 mod_srscd_constants.f90 File Reference

Modules

- module mod_srscd_constants

  Module mod_SRSCD_constants (list of globally shared variables and pointers)

Variables

- type(processordata) mod_srscd_constants::myproc

  Contains processor information (id, neighbors)
- type(mesh), dimension(:), allocatable mod_srscd_constants::mymesh

  Contains (local) mesh information.
- type(boundarymesh), dimension(:,:), allocatable mod_srscd_constants::myboundary

  Boundary elements (direction, element #)
- integer mod_srscd_constants::numcells

  Number of cells in local mesh.
- type(reaction), dimension(:), pointer mod_srscd_constants::reactionlist

  List of reactions in local (coarse) mesh.
- type(defect), dimension(:), pointer mod_srscd_constants::defectlist

  List of defects in local (coarse) mesh.
- double precision mod_srscd_constants::totalrate

  Total reaction rate in this processor.
- double precision, dimension(:), allocatable mod_srscd_constants::totalratevol

  Total reaction rate in each volume element.

- double precision mod_srscd_constants::maxrate

  Max reaction rate in all processors.
- type(cascadeevent), pointer mod_srscd_constants::cascadelist

  List of cascades (read from file) that can be implanted.
- type(cascade), pointer mod_srscd_constants::activecascades

  List of fine meshes that are active due to recent cascade implantation (Contains defect lists and reaction lists)
- integer mod_srscd_constants::numcascades

  number of cascades in the cascade input file (used to choose cascades to input in simulation)
- integer mod_srscd_constants::numcellscascade

  number of volume elements within a cascade (fine) mesh
- integer mod_srscd_constants::numxcascade

  number of elements in cascade x-direction
- integer mod_srscd_constants::numycascade

  number of elements in cascade y-direction
- integer mod_srscd_constants::numzcascade

  number of elements in cascade z-direction
- integer, dimension(:,:), allocatable mod_srscd_constants::cascadeconnectivity

  connectivity matrix for cascade meshes (same for all fine meshes)
- double precision mod_srscd_constants::finelength

  length of a cascade volume element (nm)
- double precision mod_srscd_constants::cascadeelementvol

  volume of a cascade element (nm^ 3)
- integer mod_srscd_constants::nummaterials

  Number of material types (eg. copper, niobium or bulk, grain boundary)
- integer mod_srscd_constants::numspecies

  Number of chemical species (typically set to 4: He, V, SIA_glissile, SIA_sessile)
- type(diffusionfunction), dimension(:,:), allocatable mod_srscd_constants::difffunc

  Parameters for functional forms of diffusion rates for defects.
- type(diffusionsingle), dimension(:,:), allocatable mod_srscd_constants::diffsingle

  Parameters for diffusion of single defects.
- type(bindingsingle), dimension(:,:), allocatable mod_srscd_constants::bindsingle

  Parameters for binding of single defects.
- type(bindingfunction), dimension(:,:), allocatable mod_srscd_constants::bindfunc

  Parameters for functional forms of binding energies for defects.
- type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::dissocreactions

  List of allowed dissociation reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::diffreactions

  List of allowed diffusion reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::sinkreactions

  List of allowed sink reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::impurityreactions

  List of allowed impurity reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::clusterreactions

  List of allowed clustering reactions (and ref. to functional form of reaction rate)
- type(reactionparameters), dimension(:,:), allocatable mod_srscd_constants::implantreactions

  List of allowed implantation reactions (and ref. to functional form of reaction rate)
- integer, dimension(:), allocatable mod_srscd_constants::numsinglediff

  Number of single defect diffusion rates in input file.
- integer, dimension(:), allocatable mod_srscd_constants::numfuncdiff

  Number of functional forms for diffusion rates in input files.
- integer, dimension(:), allocatable mod_srscd_constants::numsinglebind

Number of single defect binding energies in input file.

- integer, dimension(:), allocatable mod_srscd_constants::numfuncbind

  Number of functional forms for binding energies in input files.
- integer, dimension(:), allocatable mod_srscd_constants::numdissocreac

  Number of dissociation reactions in input file.
- integer, dimension(:), allocatable mod_srscd_constants::numdiffreac

  Number of diffusion reactions in input file.
- integer, dimension(:), allocatable mod_srscd_constants::numsinkreac

  Number of sink reactions in input file.
- integer, dimension(:), allocatable mod_srscd_constants::numimpurityreac

  Number of impurity reactions in input file.
- integer, dimension(:), allocatable mod_srscd_constants::numclusterreac

  Number of clustering reactions in input file.
- integer, dimension(:), allocatable mod_srscd_constants::numimplantreac

  Number of implantation reactions in input file (cascade, Frenkel pair, He currently implemented)
- double precision, parameter mod_srscd_constants::kboltzmann =8.6173324d-5

  Boltzmann's constant (eV/K)
- double precision, parameter mod_srscd_constants::pi =3.141592653589793

  Pi.
- double precision, parameter mod_srscd_constants::zint = 1.2

  Constant representing preference for clustering of interstitials by interstitial clusters (increases clustering cross-section)
- double precision, parameter mod_srscd_constants::reactionradius =.5065d0

  Material parameter used for reaction distances (impacts reaction rates)
- double precision mod_srscd_constants::temperature

  Temperature (K)
- double precision mod_srscd_constants::tempstore

  Temperature read in (K) - used when temp. changes several times during a simulation.
- double precision mod_srscd_constants::hedparatio

  Helium to dpa ratio (atoms per atom)
- double precision mod_srscd_constants::dparate

  DPA rate in dpa/s.
- double precision mod_srscd_constants::atomsize

  atomic volume (nm^3)
- double precision mod_srscd_constants::dpa

  DPA tracker (not a parameter)
- double precision mod_srscd_constants::defectdensity

  total density of defects (?), not sure if this is needed
- double precision mod_srscd_constants::dislocationdensity

  density of dislocations (sinks for point defects)
- double precision mod_srscd_constants::impuritydensity

  denstiy of impurity atoms (traps for SIA loops)
- double precision mod_srscd_constants::totaldpa

  total DPA in simulation
- double precision mod_srscd_constants::burgers

  magnitude of burgers vector, equal to lattice constant
- double precision mod_srscd_constants::numdisplacedatoms

  number of atoms displaced per cascade, read from cascade file
- double precision mod_srscd_constants::meanfreepath

  mean free path before a defect is absorbed by a grain boundary (AKA avg. grain size)
- double precision mod_srscd_constants::cascadereactionlimit

Total reaction rate in a cascade cell to consider it annealed and release cascade back to coarse mesh (s^-1)

- double precision mod_srscd_constants::cascadevolume

  Volume of cascade (used for cascade mixing probability)
- double precision mod_srscd_constants::totalvolume

  Volume of single processor's mesh (nm^3)
- double precision mod_srscd_constants::systemvol

  Volume of all processors (global), not including grain boundary elements (nm^3)
- double precision mod_srscd_constants::alpha_v

  Grain boundary sink efficiency for vacancies.
- double precision mod_srscd_constants::alpha_i

  Grain boundary sink efficiency for interstitials.
- double precision mod_srscd_constants::conc_v

  Concentration of vacancies found by GB model (used to fit alpha_v)
- double precision mod_srscd_constants::conc_i

  Concentration of interstitials found by GB model (used to fit alpha_i)
- double precision mod_srscd_constants::annealtime

  Temperature of anneal stage (K)
- double precision mod_srscd_constants::annealtemp

  Amount of time for anneal (s)
- double precision mod_srscd_constants::annealsteps

  Number of annealing steps.
- double precision mod_srscd_constants::annealtempinc

  Temperature increment at each annealing step (additive or multipliciative)
- character ∗20 mod_srscd_constants::annealtype

  ('mult' or 'add') toggles additive or multiplicative anneal steps
- logical mod_srscd_constants::annealidentify

  (.TRUE. if in annealing phase, .FALSE. otherwise) used to determine how to reset reaction rates (should we include implantation or not)
- integer mod_srscd_constants::numsims

  Number of times to repeat simulation.
- integer mod_srscd_constants::max3dint

  largest SIA size that can diffuse in 3D as spherical cluster
- integer mod_srscd_constants::siapinmin

  Smallest size of SIA that can pin at HeV clusters.
- integer mod_srscd_constants::numgrains

  Number of grains inside polycrystal (default 1)
- character ∗20 mod_srscd_constants::implanttype

  (Frenkel pairs or cascades), used to determine the type of damage in the simulation
- character ∗20 mod_srscd_constants::grainboundarytoggle

  Used to determine whether or not we are using grain boundaries to remove defects from simulation.
- character ∗20 mod_srscd_constants::hesiatoggle

  Toggles whether or not we allow HeSIA clusters to form ('yes' or 'no')
- character ∗20 mod_srscd_constants::siapintoggle

  Toggles whether or not we allow HeV clusters to pin SIA clusters (without annihilating V+SIA)
- character ∗20 mod_srscd_constants::meshingtype

  (adaptive or nonAdaptive), used to determine whether we are simulating cascade implantation with adaptive meshing
- character ∗20 mod_srscd_constants::implantscheme

  (MonteCarlo or explicit), used to determine if cascades are implanted through Monte Carlo algorithm or explicitly
- character ∗20 mod_srscd_constants::implantdist

  (Uniform or NonUniform), used to determine if defects are implanted uniformly or if DPA rate / He implant rate are given for each volume element

- character ∗20 mod_srscd_constants::polycrystal

  (yes or no), used to identify whether or not we have multiple grains in our crystal
- character ∗20 mod_srscd_constants::vtktoggle

  (yes or no), used to toggle whether we want vtk output at each time increment (log scale)
- character ∗20 mod_srscd_constants::outputdebug

  (yes or no), used to toggle whether we want to output a debug restart file at each time increment
- character ∗20 mod_srscd_constants::singleelemkmc

  (yes or no), used to toggle whether we are making one kMC choice per volume element or one kMC choice for the whole processors
- character ∗20 mod_srscd_constants::sinkeffsearch

  (yes or no), used to toggle search for effective sink efficiency
- character ∗20 mod_srscd_constants::strainfield

  (yes or no), used to toggle whether we are simulating diffusion in a strain field
- character ∗20 mod_srscd_constants::postprtoggle

  (yes or no), used to toggle whether we output the postpr.out data file
- character ∗20 mod_srscd_constants::totdattoggle

  (yes or no), used to toggle whether we output the totdat.out data file
- character ∗20 mod_srscd_constants::rawdattoggle

  (yes or no), used to toggle whether we output the rawdat.out data file
- character ∗20 mod_srscd_constants::xyztoggle

  (yes or no), used to toggle whether we output an .xyz data file (for visualization)
- character ∗20 mod_srscd_constants::profiletoggle

  (yes or no), used to toggle whether we output a DefectProfile.out data file
- double precision mod_srscd_constants::omega

  Geometric constant for 3D spherical clustering (see Dunn et al. JNM 2013)
- double precision mod_srscd_constants::omega2d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision mod_srscd_constants::omega1d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision mod_srscd_constants::omegastar

  Geometric constant for 3D spherical clustering (see Dunn et al. JNM 2013)
- double precision mod_srscd_constants::omegastar1d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision mod_srscd_constants::omegacircle1d

  Geometric constant for clustering with dislocation loops (see Dunn et al. JNM 2013)
- double precision mod_srscd_constants::recombinationcoeff

  Geometric constant for Frenkel pair recombination (see Dunn et al. JNM 2013)
- integer mod_srscd_constants::ierr

  used for initializing and finalizing MPI
- integer, parameter mod_srscd_constants::master =0

  Define the master node as ID=0.
- integer, parameter mod_srscd_constants::maxbuffersize =50

  Used to define the max size of a send/recieve buffer.
- integer mod_srscd_constants::numimplantevents

  Postprocessing: number of Frenkel pairs / cascades (local)
- integer mod_srscd_constants::numheimplantevents

  Postprocessing: number of He implantation events (local)
- integer mod_srscd_constants::totalimplantevents

  Postprocessing: number of implant events across all processors.
- integer mod_srscd_constants::numheimplanttotal

  Postprocessing: number of He implant events across all processors.

- integer mod_srscd_constants::numannihilate

  Postprocessing: number of annihilation reactions carried out.
- integer mod_srscd_constants::numtrapv

  Postprocessing: number of vacancies trapped on grain boundary.
- integer mod_srscd_constants::numtrapsia

  Postprocessing: number of SIAs trapped on grain boundary.
- integer mod_srscd_constants::numemitv

  Postprocessing: number of vacancies emitted from grain boundary.
- integer mod_srscd_constants::numemitsia

  Postprocessing: number of SIAs emitted from grain boundary.
- integer mod_srscd_constants::numimplanteventsreset

  For creating restart file (debugging tool, see example): number of cascades/Frenkel pairs.
- integer mod_srscd_constants::numheimplanteventsreset

  For creating restart file (debugging tool, see example): number of helium implantaion events.
- double precision mod_srscd_constants::elapsedtimereset

  For creating restart file (debugging tool, see example): elapsed time.
- character ∗20 mod_srscd_constants::debugtoggle

  ('yes' or 'no') input parameter indicating whether we are restarting from a file
- character ∗50 mod_srscd_constants::restartfilename

  Name of restart file.
- integer mod_srscd_constants::numimplantdatapoints

  For non-uniform implantation, number of input data points through-thickness.
- double precision, dimension(:,:), allocatable mod_srscd_constants::implantratedata

  Data containing implantation rates as a function of depth (for non-uniform implantation)
- integer mod_srscd_constants::numdipole

  Number of dipole tensors that are read in from a file.
- type(dipoletensor), dimension(:), allocatable mod_srscd_constants::dipolestore

  Array of dipole tensors for associated defects, size numDipole.
- character ∗50 mod_srscd_constants::strainfilename

  File name of strain field intput data.
- character ∗50 mod_srscd_constants::dipolefilename

  File name of dipole tensor input data.

10.14 Postprocessing_srscd.f90 File Reference

Functions/Subroutines

- subroutine outputdefects ()

  subroutine outputDefects - outputs raw data for defect populations in various volume elements
- subroutine outputdefectstotal (elapsedTime, step)

  Subroutine outputDefectsTotal() - outputs the total defects and post processing for the entire volume.
- subroutine outputdefectsboundary (elapsedTime, step)

  Subroutine outputDefectsBoundary() - outputs the total defects and post processing on the grain boundary.
- subroutine outputdefectsprofile (sim)

  Subroutine OutputDefectsProfile(sim) - outputs depth profile of defect concentrations.
- subroutine outputdefectsxyz ()

  Subroutine outputDefectsXYZ - outputs xyz file of defect concentrations.
- subroutine outputdefectsvtk (fileNumber)

  Subroutine outputDefectsVTK - outputs defect populations in vtk file format.
- subroutine outputrates (elapsedTime, step)

Subroutine outputRates(step) - outputs reaction rates to a file.

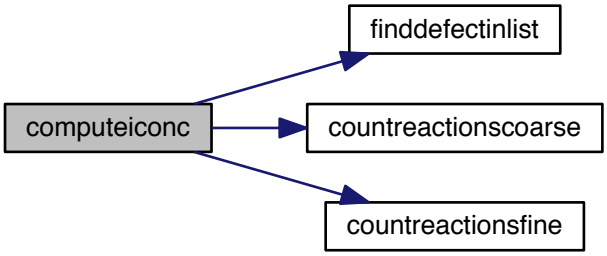
- subroutine outputdebugrestart (fileNumber, elapsedTime)

  Subroutine outputDebugRestart(outputCounter) - outputs debug restart.in file.
- double precision function computevconc ()
- double precision function computeiconc ()

10.14.1 Function/Subroutine Documentation

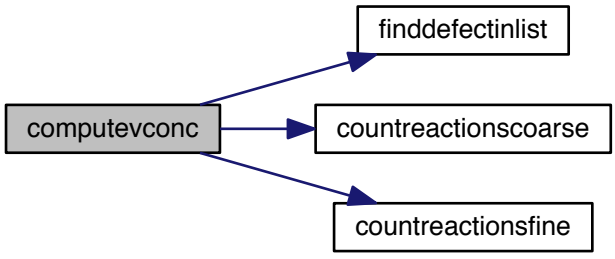
10.14.1.1 double precision function computeiconc ( )

Here is the call graph for this function:



10.14.1.2 double precision function computevconc ( )

Here is the call graph for this function:



10.14.1.3 subroutine outputdebugrestart (integer *fileNumber,* double precision *elapsedTime*)

Subroutine outputDebugRestart(outputCounter) - outputs debug restart.in file.

Outputs a file restart.in that allows the user to restart the simulation from the point that it ended. This can be used to find errors that occur after significant computation time as well as to run simulations with defects already present.

Here is the caller graph for this function:

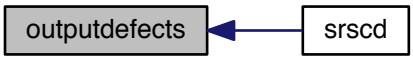
10.14.1.4 subroutine outputdefects ( )

subroutine outputDefects - outputs raw data for defect populations in various volume elements

Outputs into file: rawdat.out. These contain the complete defect populations per volume element.

Compiles data from local as well as global processors.

Here is the caller graph for this function:



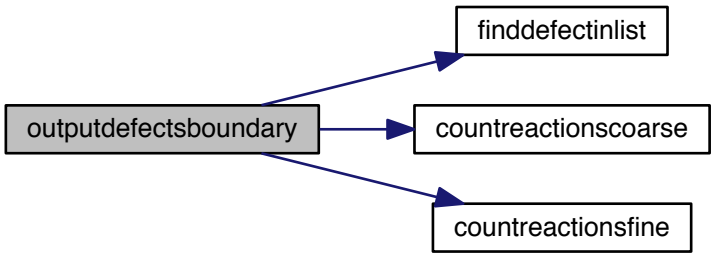
10.14.1.5 subroutine outputdefectsboundary (double precision *elapsedTime,* integer *step*)

Subroutine outputDefectsBoundary() - outputs the total defects and post processing on the grain boundary.

Outputs a list of all defects in the grain boundary (defect type, num), regardless of volume element. Compiles such a list using message passing between processors. Used to find total defect populations in simulation volume, not spatial distribution of defects.

This subroutine will also automatically output the concentration of vacancies, SIAs, and SIAs of size larger than 16, 20, and 24 defects. (Corresponding to diameters 0.9 nm, 1.0 nm, 1.1 nm)

Here is the call graph for this function:



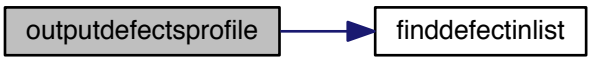
10.14.1.6 subroutine outputdefectsprofile (integer *sim*)

Subroutine OutputDefectsProfile(sim) - outputs depth profile of defect concentrations.

This subroutine outputs a list of the z-coordinates in the given system as well as the total concentration of vacancies, interstitials, and helium atoms at each depth. This is meant to be used by the Gnuplot script DepthProfile.p.

Inputs: Defect and mesh information, simulation number (for filename) Outputs: Text document with defect profile

Here is the call graph for this function:



Here is the caller graph for this function:

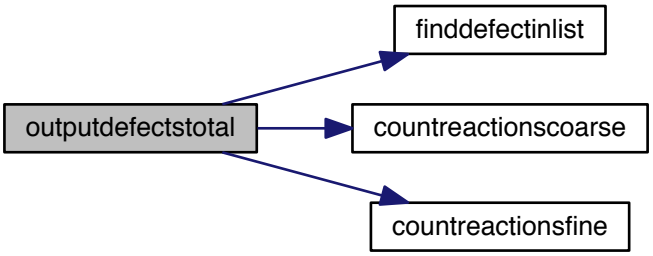
10.14.1.7 subroutine outputdefectstotal (double precision *elapsedTime,* integer *step*)

Subroutine outputDefectsTotal() - outputs the total defects and post processing for the entire volume.

Outputs a list of all defects in system (defect type, num), regardless of volume element. Compiles such a list using message passing between processors. Used to find total defect populations in simulation volume, not spatial distribution of defects.

This subroutine will also automatically output the concentration of vacancies, SIAs, and SIAs of size larger than 16, 20, and 24 defects. (Corresponding to diameters 0.9 nm, 1.0 nm, 1.1 nm)

Here is the call graph for this function:



Here is the caller graph for this function:

10.14.1.8 subroutine outputdefectsvtk (integer *fileNumber*)

Subroutine outputDefectsVTK - outputs defect populations in vtk file format.

Outputs defects in a VTK formatted file (3D unstructured grid of linear cubes)

Here is the caller graph for this function:

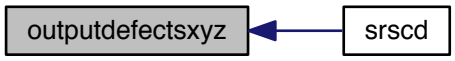
10.14.1.9 subroutine outputdefectsxyz ( )

Subroutine outputDefectsXYZ - outputs xyz file of defect concentrations.

Outputs number of vacancies / SIAs per volume element in .XYZ file (see wikipedia page for documentation on file format)

Only outputs defects in the main mesh (ignores fine meshes present)

Here is the caller graph for this function:



10.14.1.10 subroutine outputrates (double precision *elapsedTime,* integer *step*)

Subroutine outputRates(step) - outputs reaction rates to a file.

Outputs the total reaction rate in each processor to a file. Used for debugging and for parallel analysis

Inputs: integer step (the reaction step number) Outputs: reaction rate of each processor written in file

10.15 ReactionRates.f90 File Reference

Modules

- module reactionrates

  Module: ReactionRates - adds reactions and calculates rates for the various reaction types in the system.

Functions/Subroutines

- subroutine reactionrates::addsingledefectreactions (cell, defectType)

  Subroutine add single defect reactions - adds reactions to a reaction list that require only a single reactant to be carried out. Examples: dissociation, trapping, sinks. Diffusion reactions are not included in this subroutine.
- subroutine reactionrates::addsingledefectreactionsfine (cascadeID, cell, defectType)

  Subroutine add single defect reactions fine - adds reactions to a reaction list inside a cascade (fine mesh) that require only a single reactant to be carried out. Examples: dissociation, trapping, sinks. Diffusion reactions are not included in this subroutine.
- subroutine reactionrates::addmultidefectreactions (cell, defectType1, defectType2)

  Subroutine add multi defect reactions - adds reactions to a reaction list that require multiple (only using 2 currently) defects to be carried out. This refers mainly to clustering reactions or pinning reactions.
- subroutine reactionrates::addmultidefectreactionsfine (cascadeID, cell, defectType1, defectType2)

  Subroutine add multi defect reactions (fine mesh) - adds reactions to a reaction list inside a cascade that require multiple (only using 2 currently) defects to be carried out. This refers mainly to clustering reactions or pinning reactions.
- subroutine reactionrates::adddiffusionreactions (cell1, cell2, proc1, proc2, dir, defectType)

  Subroutine add diffusion reactions - adds reactions to a reaction list representing diffusion between volume elements.
- subroutine reactionrates::adddiffusioncoarsetofine (cell, proc, CascadeCurrent, defectType)

Subroutine add diffusion reactions from the coarse mesh to a fine mesh element (cascade) - adds reactions to a reaction list representing diffusion between volume elements.

- subroutine reactionrates::adddiffusionreactionsfine (cascadeID, cell1, cell2, proc1, proc2, dir, defectType)

  Subroutine add diffusion reactions (fine mesh) - adds reactions to a reaction list representing diffusion between volume elements inside a cascade mesh.
- double precision function reactionrates::findreactionrate (cell, reactionParameter)

  Function find reaction rate - finds reaction rate for implantation reaction (He, Frenkel pairs, cascades).
- double precision function reactionrates::findreactionratefine (cell, reactionParameter)

  Function find reaction rate fine - finds reaction rate for implantation reaction (Helium or Frenkel pairs only) in a fine mesh. Cascades cannot be implanted inside other cascades.
- double precision function reactionrates::findreactionrateimpurity (defectType, cell, reactionParameter)

  Function find reaction rate impurity - finds reaction rate for trapping of SIA loops by impurities (Carbon).
- double precision function reactionrates::findreactionrateimpurityfine (CascadeCurrent, defectType, cell, reactionParameter)

  Function find reaction rate impurity fine - finds reaction rate for trapping of SIA loops by impurities (Carbon) inside a fine mesh (Cascade).
- double precision function reactionrates::findreactionratedissoc (defectType, products, cell, reaction↩ Parameter)

  Function find reaction rate dissociation - finds reaction rate for point defects to dissociate from clusters.
- double precision function reactionrates::findreactionratedissocfine (CascadeCurrent, defectType, products, cell, reactionParameter)

  Function find reaction rate dissociation - finds reaction rate for point defects to dissociate from clusters in the fine mesh (cascade)
- double precision function reactionrates::findreactionratesink (defectType, cell, reactionParameter)

  Function find reaction rate sink - finds reaction rate for defects to get absorbed at sinks (typically matrix dislocations)
- double precision function reactionrates::findreactionratesinkfine (CascadeCurrent, defectType, cell, reactionParameter)

  Function find reaction rate sink fine - finds reaction rate for defects to get absorbed at sinks (typically matrix dislocations) in the fine mesh (cascade)
- double precision function reactionrates::findreactionratemultiple (defectType1, defectType2, cell, reaction↩ Parameter)

  Function find reaction rate multiple - finds reaction rate for defect clustering.
- double precision function reactionrates::findreactionratemultiplefine (CascadeCurrent, defectType1, defect↩ Type2, cell, reactionParameter)

  Function find reaction rate multiple fine - finds reaction rate for defect clustering in the fine mesh (Cascade)
- double precision function reactionrates::findreactionratediff (defectType, cell1, proc1, cell2, proc2, dir, reactionParameter)

  Function find reaction rate diffusion - finds reaction rate for defect diffusion between elements.
- double precision function reactionrates::findreactionratecoarsetofine (defectType, cell, proc, numDefectsFine, reactionParameter)

  Function find reaction rate diffusion coarse to fine - finds reaction rate for defect diffusion between a coarse mesh element and a fine (cascade) mesh.
- double precision function reactionrates::findreactionratedifffine (CascadeCurrent, defectType, cell1, proc1, cell2, proc2, dir, reactionParameter)

  Function find reaction rate diffusion fine - finds reaction rate for defect diffusion between elements in the fine mesh (inside a cascade)
- subroutine reactionrates::findreactioninlist (reactionCurrent, reactionPrev, cell, reactants, products, num↩ Reactants, numProducts)

  Subroutine find Reaction In List.
- subroutine reactionrates::findreactioninlistdiff (reactionCurrent, reactionPrev, reactants, cell1, cell2, proc1, proc2)

  Subroutine find Reaction In List (diffusion)
- subroutine reactionrates::findreactioninlistmultiple (reactionCurrent, reactionPrev, cell, reactants, products, numReactants, numProducts)

Subroutine find Reaction In List Multiple (clustering)

- subroutine reactionrates::defectcombinationrules (products, defectTemp)

  defectCombinationRules (subroutine)

- subroutine reactionrates::checkreactionlegality (numProducts, products, isLegal)

  Subroutine check Reaction Legality.

- double precision function reactionrates::finddparatelocal (zCoord)

  Function find DPA Rate Local (zCoord)

- double precision function reactionrates::findheimplantratelocal (zCoord)

  Function find Helium Implantation Rate Local (zCoord)

10.16 Read_inputs.f90 File Reference

Functions/Subroutines

- subroutine readmaterialinput (filename)

  Subroutine read material input - reads material input information from file.

- subroutine readcascadelist ()

  Subroutine readCascadeList() - reads defects in cascades and locations from a file.

- subroutine readimplantdata ()

  Subroutine readImplantData() - reads non-uniform implantation profiles (damage and helium)

- subroutine selectmaterialinputs ()

  Subroutine selectMaterialInputs() - controlling subroutines that calls other subroutines to read inputs.

- subroutine readparameters ()

  Subroutine readParameters() - reads in simulation parameters from parameters.txt.

- subroutine readreactionlistsizes (filename)

  Subroutine reaction list sizes - finds max sizes of reaction parameters for allocation.

10.16.1 Function/Subroutine Documentation

10.16.1.1 subroutine readcascadelist ( )

Subroutine readCascadeList() - reads defects in cascades and locations from a file.

This subroutine reads a list of cascades from a file (name given in readParameters). This information is stored in derived type cascadeEvent and used whenever a cascade is chosen throughout the duration of the program.

Here is the caller graph for this function:

10.16.1.2 subroutine readimplantdata ( )

Subroutine readImplantData() - reads non-uniform implantation profiles (damage and helium)

This subroutine recognizes whether we are in a uniform implantation scheme or a non-uniform implantation scheme. If the implantation is non-uniform, this subroutine reads from a file the implantation profile (in dpa/s for each material point).

Note: as constructed, this is hard-coded to read in one-dimensional DPA and He implantation rates (for DPA profiles through the thickness of a material, for example), instead of full 3D DPA distributions. This is typically for thin films with implantation profiles that vary through their depth.

This is written such that the input file does not have to have the same number of points with the same z-coordinates as the elements in the mesh. The code will interpolate from the input file what the DPA rate and He implant rate should be in each element. However, if the size of the input file is smaller than the size of the mesh in the z-direction, it will return an error.

Inputs: file with DPA and He implant rates in z-coordinates Outputs: information is stored in a global array.

Here is the caller graph for this function:

10.16.1.3 subroutine readmaterialinput (character∗50 *filename*)

Subroutine read material input - reads material input information from file.

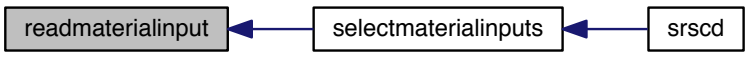
Information read in includes:

1) Allowed defects and their diffusion rates and binding energies

2) Allowed reactions, including single, multi, diffusion, and implantation reactions

NOTE: we are double-reading in the values of numSingleDiff(matNum), etc, because it has already been done in the subroutine readReactionListSizes().

Here is the caller graph for this function:



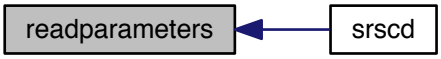
10.16.1.4 subroutine readparameters ( )

Subroutine readParameters() - reads in simulation parameters from parameters.txt.

This subroutine reads in all simulation and material parameters located in parameters.txt as well a the file names for all other input files (material input, mesh, cascades, implantation profile, etc). It also reads in toggles for various simulation options. Default values for several options are stored here.

Geometric constants used in computing clustering rates are computed at the end of this subroutine.

Here is the caller graph for this function:



10.16.1.5 subroutine readreactionlistsizes (character∗50 *filename*)

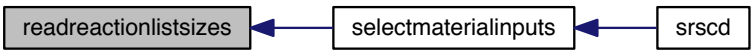
Subroutine reaction list sizes - finds max sizes of reaction parameters for allocation.

Information read in includes:

1) Number of allowed defects

2) Number of allowed reactions of each size

Here is the caller graph for this function:

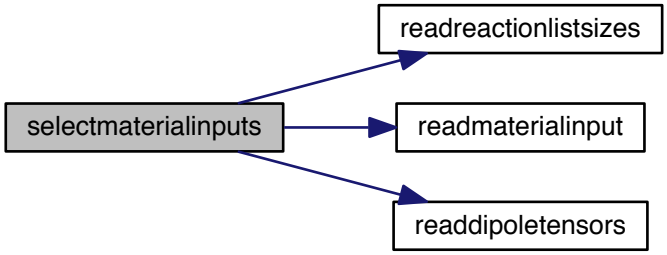


10.16.1.6 subroutine selectmaterialinputs ( )

Subroutine selectMaterialInputs() - controlling subroutines that calls other subroutines to read inputs.

This subroutine reads the name of the material input file(s) and then reads in relevant material constants (binding and migration energies, number of species, allowed reactions, reaction functional forms, etc)

Here is the call graph for this function:



Here is the caller graph for this function:

10.17 SRSCD_par.f90 File Reference

Functions/Subroutines

- program srscd

  Main program.

10.17.1 Function/Subroutine Documentation

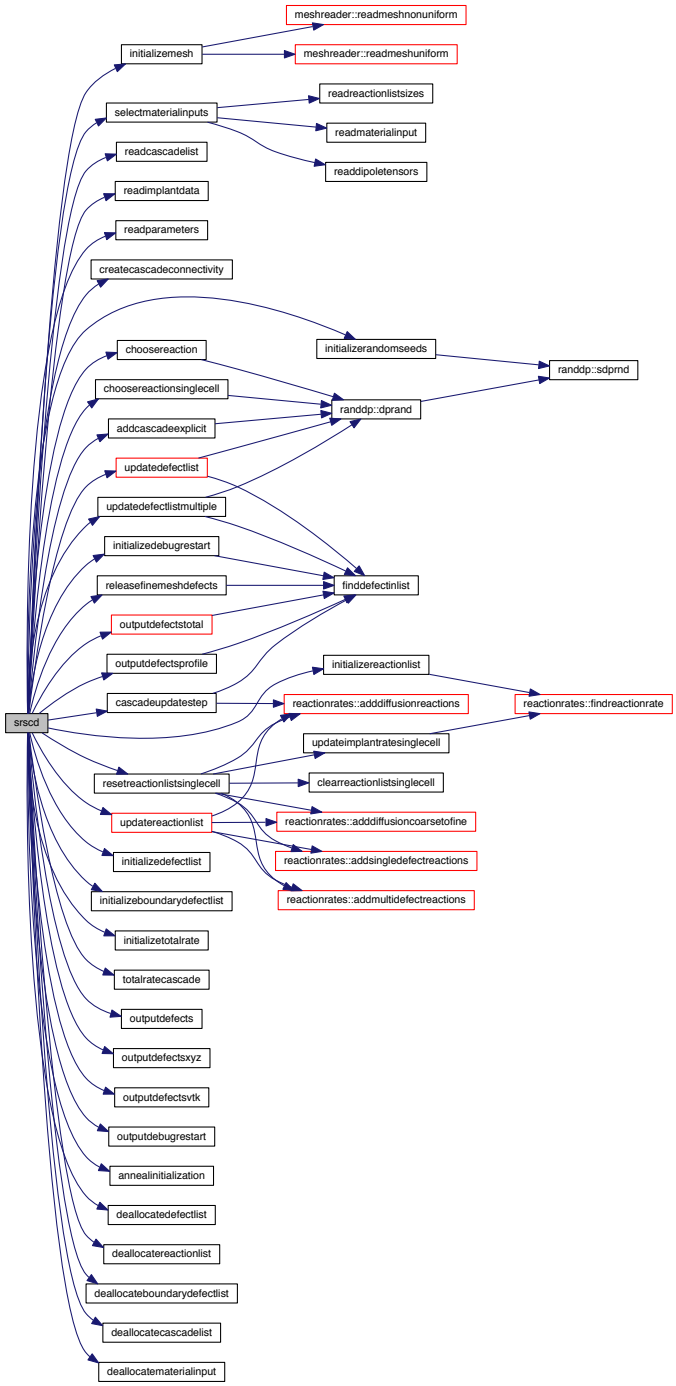
10.17.1.1 program srscd ( )

Main program.

For outline, see main page of manual. This program uses the kinetic Monte Carlo algorithm in a synchronous parallel implementation to simulate radiation damage accumulation and subsequent annealing in metals.

Several debugging options can be triggered by un-commenting them in this program, in the case of difficulty.

Here is the call graph for this function:



10.18 StrainSubroutines.f90 File Reference

Functions/Subroutines

- subroutine readdipoletensors (filename)

  Subroutine readDipoleTensors() - reads defect dipole tensors in from a file and stores them in global variable.

- double precision function calculatedeltaem (cellNumber, defectType)

  Subroutine calculateDeltaEm() - uses dipole tensors at equilibrium position and saddle point to calculate delta Em caused by strain field.
- double precision function finddiffusivitystrain (matNum, DefectType, cellNumber)

  double precision function find diffusivity strain - returns the diffusivity of a given defect type inside a given volume element including the impact of the strain field
- double precision function diffusivitycomputestrain (DefectType, functionType, numParameters, parameters, cellNumber)

  double precision function diffusivityComputeStrain - computes diffusivity using a functional form for defects that don't have their diffusivity given by a value in a list. Includes the effect of strain interacting with defect dipole tensor

10.18.1 Function/Subroutine Documentation

10.18.1.1 double precision function calculatedeltaem (integer *cellNumber,* integer, dimension(numspecies) *defectType*)

Subroutine calculateDeltaEm() - uses dipole tensors at equilibrium position and saddle point to calculate delta Em caused by strain field.

This subroutine calculates the double dot product of the strain field in a given volume element with the dipole tensor of a given defect type at the equilibrium position and the saddle point position to calculate the change in migration energy of that defect.

Initially, this subroutine will be set up without averaging the results over several orientations of the dipole tensor. Once we decide how to best average these results, this will be added to the subroutine.

10.18.1.2 double precision function diffusivitycomputestrain (integer, dimension(numspecies) *DefectType,* integer *functionType,* integer *numParameters,* double precision, dimension(numparameters) *parameters,* integer *cellNumber*)

double precision function diffusivityComputeStrain - computes diffusivity using a functional form for defects that don't have their diffusivity given by a value in a list. Includes the effect of strain interacting with defect dipole tensor

This function has several hard-coded functional forms for diffusivity, including immobile defects, constant functions, and mobile SIA loops. Additional functional forms can be added as needed.

Strain is included by modifying the migration energy Em by an amount equal to the defect dipole tensor double dot product with the strain field at the equilibrium and saddle points

10.18.1.3 double precision function finddiffusivitystrain (integer *matNum,* integer, dimension(numspecies) *DefectType,* integer *cellNumber*)

double precision function find diffusivity strain - returns the diffusivity of a given defect type inside a given volume element including the impact of the strain field

This function looks up the diffusivity of a given defect type using input date from material parameter input file. It either looks up diffusivity from values in a list or computes diffusivity using diffusivityCompute in the case of a functional form.

It then computes the interaction of the strain field inside a given volume element with the defect dipole and modifies the migration energy accordingly.

If the resulting migration energy is less than zero, it returns a migration energy of zero.

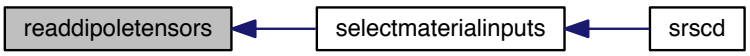
Input: defect type, cell ID number Output: diffusivity (nm^2/s)

10.18.1.4 subroutine readdipoletensors (character∗50 *filename*)

Subroutine readDipoleTensors() - reads defect dipole tensors in from a file and stores them in global variable.

This subroutine reads in the dipole tensors of various defects from a file. These dipole tensors are then stored in the global variable dipoleTensor for later reference when calculating the interaction energy between a strain field and various defects (vacancies, interstitials, etc.)

Here is the caller graph for this function:



Chapter 11

Acknowledgements

Supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

This work is also supported by the Sandia National Laboratories/Georgia Tech Excellence in Engineering Research Program

Bibliography

A. Dunn, Radiation damage accumulation and associated mechanical hardening in thin films and bulk materials, Ph.D. thesis, Georgia Institute of Technology, 2016. 2

R. E. Stoller, S. I. Golubov, C. Domain, C. Becquart, Mean field rate theory and object kinetic Monte Carlo: a comparison of kinetic models, Journal of Nuclear Materials 382 (2) (2008) 77–90. 2, 3, 4, 5, 31, 32

N. Soneda, T. D. De La Rubia, Defect production, annealing kinetics and damage evolution in α-Fe: an atomic-scale computer simulation, Philosophical Magazine A 78 (5) (1998) 995–1019. 2, 5, 7, 9, 13, 27

C. Domain, C. Becquart, L. Malerba, Simulation of radiation damage in Fe alloys: an object kinetic Monte Carlo approach, Journal of Nuclear Materials 335 (1) (2004) 121–145. 2

M. Caturla, N. Soneda, E. Alonso, B. Wirth, T. D. de la Rubia, J. Perlado, Comparative study of radiation damage accumulation in Cu and Fe, Journal of Nuclear Materials 276 (1) (2000) 13–21. 2, 9, 13, 14, 16

C. Ortiz, M. Caturla, Simulation of defect evolution in irradiated materials: role of intracascade clustering and correlated recombination, Physical Review B 75 (18) (2007) 184101. 2, 3, 9, 10, 13, 34

C. Ortiz, M. Caturla, C. Fu, F. Willaime, He diffusion in irradiated α- Fe: An ab-initio-based rate theory model, Physical Review B 75 (10) (2007) 100102. 2, 10, 34, 35, 37

Y. Li, W. Zhou, L. Huang, Z. Zeng, X. Ju, Cluster dynamics modeling of accumulation and diffusion of helium in neutron irradiated tungsten, Journal of Nuclear Materials 431 (1) (2012) 26–32. 2

A. Dunn, M. McPhie, L. Capolungo, E. Martinez, M. Cherkaoui, A rate theory study of helium bubble formation and retention in Cu–Nb nanocomposites, Journal of Nuclear Materials 435 (1) (2013a) 141–152. 2, 5, 9, 10, 34, 35

Q. Xu, N. Yoshida, T. Yoshiie, Accumulation of helium in tungsten irradiated by helium and neutrons, Journal of Nuclear Materials 367 (2007) 806–811. 2, 10

J. Marian, V. V. Bulatov, Stochastic cluster dynamics method for simulations of multispecies irradiation damage accumulation, Journal of Nuclear Materials 415 (1) (2011) 84–95. 2, 3, 4, 11, 37

K. Tapasa, A. Barashev, D. Bacon, Y. N. Osetsky, Computer simulation of carbon diffusion and vacancy–carbon interaction in α-iron, Acta materialia 55 (1) (2007) 1–11. 2

G. Odette, G. Lucas, Embrittlement of nuclear reactor pressure vessels, Jom 53 (7) (2001) 18–22. 2

L. Wang, R. Dodd, G. Kulcinski, Gas effects on void formation in 14 MeV nickel ion irradiated pure nickel, Journal of Nuclear Materials 141 (1986) 713–717. 2

T. Tanaka, K. Oka, S. Ohnuki, S. Yamashita, T. Suda, S. Watanabe, E. Wakai, Synergistic effect of helium and hydrogen for defect evolution under multi-ion irradiation of Fe–Cr ferritic alloys, Journal of nuclear materials 329 (2004) 294–298. 2

D. T. Gillespie, A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, Journal of computational physics 22 (4) (1976) 403–434. 2, 9, 11, 43

J. L. Katz, H. Wiedersich, Nucleation of voids in materials supersaturated with vacancies and interstitials, The Journal of Chemical Physics 55 (3) (1971) 1414–1425. 3

A. Brailsford, R. Bullough, The theory of sink strengths, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 302 (1465) (1981) 87–137. 3, 5, 7

S. Golubov, A. Ovcharenko, A. Barashev, B. Singh, Grouping method for the approximate solution of a kinetic equation describing the evolution of point-defect clusters, Philosophical Magazine A 81 (3) (2001) 643–658. 4

N. Doan, G. Martin, Elimination of irradiation point defects in crystalline solids: Sink strengths, Physical Review B 67 (13) (2003) 134107. 4, 5

S. Dudarev, A. Semenov, C. Woo, Heterogeneous void swelling near grain boundaries in irradiated materials, Physical Review B 67 (9) (2003) 094103. 5

Y. N. Osetsky, D. Bacon, B. N. Singh, B. Wirth, Atomistic study of the generation, interaction, accumulation and annihilation of cascade-induced defect clusters, Journal of nuclear materials 307 (2002) 852–861. 5

W. Schilling, K. Schroeder, H. Wollenberger, Three-Dimensional versus Crowdion Migration of Interstitials in Annealing Stage I of Irradiated Metals, physica status solidi (b) 38 (1) (1970) 245–257. 6

H. Trinkaus, B. Singh, S. Golubov, Progress in modelling the microstructural evolution in metals under cascade damage conditions, Journal of nuclear materials 283 (2000) 89–98. 6

H. Trinkaus, H. Heinisch, A. Barashev, S. Golubov, B. Singh, 1D to 3D diffusion-reaction kinetics of defects in crystals, Physical Review B 66 (6) (2002) 060105. 7

A. Y. Dunn, L. Capolungo, E. Martinez, M. Cherkaoui, Spatially resolved stochastic cluster dynamics for radiation damage evolution in nanostructured metals, Journal of nuclear materials 443 (1) (2013b) 128–139. 9, 15, 34

S. Zinkle, B. Singh, Microstructure of Cu–Ni alloys neutron irradiated at 210° C and 420° C to 14 dpa, Journal of nuclear materials 283 (2000) 306–312. 9

A. Ryazanov, D. Braski, H. Schroeder, H. Trinkaus, H. Ullmaier, Modeling the effect of creep on the growth of helium bubbles in metals during annealing, Journal of nuclear materials 233 (1996) 1076–1079. 9

H. L. Heinisch, F. Gao, R. J. Kurtz, E. Le, Interaction of helium atoms with edge dislocations in α-Fe, Journal of nuclear materials 351 (1) (2006) 141–148. 9

H. Heinisch, F. Gao, R. Kurtz, Atomistic modeling of helium interacting with screw dislocations in α-Fe, Journal of Nuclear Materials 367 (2007) 311–315. 9

M. Demkowicz, Y. Wang, R. Hoagland, O. Anderoglu, Mechanisms of He escape during implantation in CuNb multilayer composites, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 261 (1) (2007) 524–528. 9

M. Demkowicz, D. Bhattacharyya, I. Usov, Y. Wang, M. Nastasi, A. Misra, The effect of excess atomic volume on He bubble formation at fcc-bcc interfaces, Applied Physics Letters 97 (2010) 161903. 9

K. Hattar, M. Demkowicz, A. Misra, I. Robertson, R. Hoagland, Arrest of He bubble growth in Cu–Nb multilayer nanocomposites, Scripta Materialia 58 (7) (2008) 541–544. 9

M. Zhernenkov, M. S. Jablin, A. Misra, M. Nastasi, Y. Wang, M. J. Demkowicz, J. K. Baldwin, J. Majewski, Trapping of implanted He at Cu/Nb interfaces measured by neutron reflectometry, Applied Physics Letters 98 (24) (2011) 1913. 9

R. Stoller, G. Odette, B. Wirth, Primary damage formation in bcc iron, Journal of Nuclear Materials 251 (1997) 49–60. 13

R. Stoller, A. Calder, Statistical analysis of a library of molecular dynamics cascade simulations in iron at 100 K, Journal of nuclear materials 283 (2000) 746–752. 13

C. S. Becquart, A. Barbu, J. Bocquet, M. Caturla, C. Domain, C.-C. Fu, S. Golubov, M. Hou, L. Malerba, C. Ortiz, et al., Modeling the long-term evolution of the primary damage in ferritic alloys using coarse-grained methods, Journal of nuclear materials 406 (1) (2010) 39–54. 13

V. Jansson, L. Malerba, Simulation of the nanostructure evolution under irradiation in Fe–C alloys, Journal of Nuclear Materials 443 (1) (2013) 274–285. 13, 25

V. Jansson, L. Malerba, OKMC simulations of Fe–C systems under irradiation: Sensitivity studies, Journal of Nuclear Materials 452 (1) (2014) 118–124. 13, 25

N. Soneda, S. Ishino, A. Takahashi, K. Dohi, Modeling the microstructural evolution in bcc-Fe during irradiation using kinetic Monte Carlo computer simulation, Journal of nuclear materials 323 (2) (2003) 169–180. 13, 25

T. Jourdan, J.-P. Crocombette, Rate theory cluster dynamics simulations including spatial correlations within displacement cascades, Physical Review B 86 (5) (2012) 054113. 13

E. Meslin, A. Barbu, L. Boulanger, B. Radiguet, P. Pareige, K. Arakawa, C. Fu, Cluster-dynamics modelling of defects in α-iron under cascade damage conditions, Journal of Nuclear Materials 382 (2) (2008) 190–196. 13

M. Sabochick, S. Yip, Migration energy calculations for small vacancy clusters in copper, Journal of Physics F: Metal Physics 18 (8) (1988) 1689. 13

J. Corbett, R. Smith, R. Walker, Recovery of electron-irradiated copper. II. Interstitial migration, Physical Review 114 (6) (1959) 1460. 13

H. Schober, R. Zeller, Structure and dynamics of multiple interstitials in FCC metals, Journal of Nuclear Materials 69 (1978) 341–349. 13

A. Dunn, L. Agudo-Merida, I. Martin-Bragado, M. McPhie, M. Cherkaoui, L. Capolungo, A novel method for computing effective diffusivity: Application to helium implanted α-Fe thin films, Journal of Nuclear Materials 448 (1) (2014) 195–205. 15

E. Martínez, J. Marian, M. H. Kalos, J. M. Perlado, Synchronous parallel kinetic Monte Carlo for continuum diffusion-reaction systems, Journal of Computational Physics 227 (8) (2008) 3804–3823. 25, 26

E. Martínez, P. R. Monasterio, J. Marian, Billion-atom synchronous parallel kinetic Monte Carlo simulations of critical 3D Ising systems, Journal of Computational Physics 230 (4) (2011) 1359–1369. 25

I. Martin-Bragado, J. Abujas, P. Galindo, J. Pizarro, Synchronous parallel Kinetic Monte Carlo: Implementation and results for object and lattice approaches, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 352 (2015) 27–30. 25

C.-C. Fu, J. Dalla Torre, F. Willaime, J.-L. Bocquet, A. Barbu, Multiscale modelling of defect kinetics in irradiated iron, Nature materials 4 (1) (2005) 68–74. 27, 29, 35, 89

N. Soneda, T. Diaz de La Rubia, Migration kinetics of the self-interstitial atom and its clusters in bcc Fe, Philosophical Magazine A 81 (2) (2001) 331–343. 27, 35, 74

A. Dunn, L. Capolungo, Simulating radiation damage accumulation in α-Fe: A spatially resolved stochastic cluster dynamics approach, Computational Materials Science 102 (2015) 314–326. 27, 29

A. Dunn, R. Dingreville, E. Martinez, L. Capolungo, Synchronous parallel spatially resolved stochastic cluster dynamics, Journal of Computational Physics (2015) (submitted). 27

T. Opplestrup, V. V. Bulatov, G. H. Gilmer, M. H. Kalos, B. Sadigh, First-passage Monte Carlo algorithm: diffusion without all the hops, Physical review letters 97 (23) (2006) 230602. 29

D. Terentyev, N. Juslin, K. Nordlund, N. Sandberg, Fast three dimensional migration of He clusters in bcc Fe and Fe–Cr alloys, Journal of Applied Physics 105 (10) (2009) 103509. 35

C.-C. Fu, F. Willaime, Ab initio study of helium in α- Fe: Dissolution, migration, and clustering with vacancies, Physical Review B 72 (6) (2005) 064117. 35

R. Vassen, H. Trinkaus, P. Jung, Helium desorption from Fe and V by atomic diffusion and bubble migration, Physical Review B 44 (9) (1991) 4206. 34, 35, 37

C. Becquart, C. Domain, An object Kinetic Monte Carlo Simulation of the dynamics of helium and point defects in tungsten, Journal of Nuclear Materials 385 (2) (2009) 223–227. 37

T. S. Hudson, S. L. Dudarev, A. P. Sutton, Confinement of interstitial cluster diffusion by oversized solute atoms, in: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 460, The Royal Society, 2457–2475, 2004. 37

M. Eldrup, B. Singh, S. Zinkle, T. Byun, K. Farrell, Dose dependence of defect accumulation in neutron irradiated copper and iron, Journal of nuclear materials 307 (2002) 912–917. 38, 39, 84

S. J. Zinkle, B. N. Singh, Microstructure of neutron-irradiated iron before and after tensile deformation, Journal of nuclear materials 351 (1) (2006) 269–284. 38

J. Dalla Torre, C.-C. Fu, F. Willaime, A. Barbu, J.-L. Bocquet, Resistivity recovery simulations of electron-irradiated iron: Kinetic Monte Carlo versus cluster dynamics, Journal of nuclear materials 352 (1) (2006) 42–49. 89

S. Takaki, J. Fuss, H. Kuglers, U. Dedek, H. Schultz, The resistivity recovery of high purity and carbon doped iron following low temperature electron irradiation, Radiation effects 79 (1-4) (1983) 87–122. 89

M. A. Tschopp, K. Solanki, F. Gao, X. Sun, M. A. Khaleel, M. Horstemeyer, Probing grain boundary sink strength at the nanoscale: Energetics and length scales of vacancy and interstitial absorption by grain boundaries in α-Fe, Physical Review B 85 (6) (2012) 064108. 95, 100