

Lemina

Generated by Doxygen 1.10.0

1	Lamina: A Molecular Dynamics Package	1
	1.1 # Lamina	1
	1.2 Why "Lamina"?	1
	1.3 Key Features	1
	1.4 Interaction Potentials	1
	1.5 Thermostats and Temperature Control	2
	1.6 Time Integration	2
	1.7 Physical Observables	2
	1.8 Output and Utilities	2
	1.9 Project Structure	2
	1.9.1 Project Structure	2
2	File Index	5
	2.1 File List	5
3	File Documentation	7
	3.1 README.md File Reference	7
	3.2 source/AccumProps.c File Reference	7
	3.2.1 Function Documentation	8
	3.2.1.1 AccumProps()	8
	3.3 AccumProps.c	8
	3.4 source/AccumVacf.c File Reference	9
	3.4.1 Function Documentation	10
	3.4.1.1 AccumVacf()	10
	3.4.1.2 Integrate()	11
	3.4.1.3 PrintVacf()	11
	3.4.1.4 ZeroVacf()	12
	3.5 AccumVacf.c	12
	3.6 source/AllocArrays.c File Reference	13
	3.6.1 Function Documentation	14
	3.6.1.1 AllocArrays()	14
	3.7 AllocArrays.c	14
	3.8 source/ApplyBoundaryCond.c File Reference	15
	3.8.1 Function Documentation	16
	3.8.1.1 ApplyBoundaryCond()	16
	3.9 ApplyBoundaryCond.c	17
	3.10 source/ApplyDrivingForce.c File Reference	17
	3.10.1 Function Documentation	18
	3.10.1.1 ApplyDrivingForce()	18
	3.11 ApplyDrivingForce.c	19
	3.12 source/ApplyForce.c File Reference	20
	3.12.1 Function Documentation	20
	3.12.1.1 ApplyForce()	20

3.13 ApplyForce.c
3.14 source/ApplyLeesEdwardsBoundaryCond.c File Reference
3.14.1 Function Documentation
3.14.1.1 ApplyLeesEdwardsBoundaryCond()
3.15 ApplyLeesEdwardsBoundaryCond.c
3.16 source/ApplyShear.c File Reference
3.16.1 Function Documentation
3.16.1.1 ApplyShear()
3.17 ApplyShear.c
3.18 source/ApplyViscous.c File Reference
3.18.1 Function Documentation
3.18.1.1 ApplyViscous()
3.19 ApplyViscous.c
3.20 source/BrownianStep.c File Reference
3.20.1 Function Documentation
3.20.1.1 BrownianStep()
3.21 BrownianStep.c
3.22 source/Close.c File Reference
3.22.1 Function Documentation
3.22.1.1 Close()
3.23 Close.c
3.24 source/ComputeBondForce.c File Reference
3.24.1 Function Documentation
3.24.1.1 ComputeBondForce()
3.25 ComputeBondForce.c
3.26 source/ComputeBondForce.h File Reference
3.26.1 Function Documentation
3.26.1.1 ComputeBondForce()
3.27 ComputeBondForce.h
3.28 source/ComputeForcesCells.c File Reference
3.28.1 Function Documentation
3.28.1.1 ComputeForcesCells()
3.29 ComputeForcesCells.c
3.30 source/ComputePairForce.c File Reference
3.30.1 Function Documentation
3.30.1.1 ComputePairForce()
3.31 ComputePairForce.c
3.32 source/ComputePairForce.h File Reference
3.32.1 Function Documentation
3.32.1.1 ComputePairForce()
3.33 ComputePairForce.h
3.34 source/DisplaceAtoms.c File Reference

3.34.1 Function Documentation	. 52
3.34.1.1 DisplaceAtoms()	. 52
3.35 DisplaceAtoms.c	. 53
3.36 source/DumpBonds.c File Reference	. 53
3.36.1 Function Documentation	. 54
3.36.1.1 DumpBonds()	. 54
3.37 DumpBonds.c	. 54
3.38 source/DumpPairs.c File Reference	. 55
3.38.1 Function Documentation	. 55
3.38.1.1 DumpPairs()	. 55
3.39 DumpPairs.c	. 56
3.40 source/DumpRestart.c File Reference	. 57
3.40.1 Function Documentation	. 57
3.40.1.1 DumpRestart()	. 57
3.41 DumpRestart.c	. 58
3.42 source/DumpState.c File Reference	. 59
3.42.1 Function Documentation	. 60
3.42.1.1 DumpState()	. 60
3.43 DumpState.c	. 60
3.44 source/EvalCom.c File Reference	. 61
3.44.1 Function Documentation	. 62
3.44.1.1 EvalCom()	. 62
3.45 EvalCom.c	. 62
3.46 source/EvalProps.c File Reference	. 63
3.46.1 Function Documentation	. 64
3.46.1.1 EvalProps()	. 64
3.47 EvalProps.c	. 65
3.48 source/EvalRdf.c File Reference	. 65
3.48.1 Function Documentation	. 66
3.48.1.1 EvalRdf()	. 66
3.49 EvalRdf.c	. 67
3.50 source/EvalSpacetimeCorr.c File Reference	. 68
3.50.1 Function Documentation	. 68
3.50.1.1 EvalSpacetimeCorr()	. 68
3.51 EvalSpacetimeCorr.c	. 70
3.52 source/EvalUnwrap.c File Reference	. 71
3.52.1 Function Documentation	. 72
3.52.1.1 EvalUnwrap()	. 72
3.53 EvalUnwrap.c	. 73
3.54 source/EvalVacf.c File Reference	. 73
3.54.1 Function Documentation	. 74
3.54.1.1 AccumVacf()	. 74

3.54.1.2 EvalVacf()	75
3.55 EvalVacf.c	75
3.56 source/EvalVrms.c File Reference	76
3.56.1 Function Documentation	77
3.56.1.1 EvalVrms()	77
3.57 EvalVrms.c	77
3.58 source/global.h File Reference	78
3.58.1 Macro Definition Documentation	82
3.58.1.1 EXTERN	82
3.58.1.2 NDIM	82
3.58.1.3 SignR	83
3.58.1.4 Sqr	83
3.58.2 Typedef Documentation	83
3.58.2.1 real	83
3.58.3 Variable Documentation	83
3.58.3.1 atom1	83
3.58.3.2 atom2	83
3.58.3.3 atomID	83
3.58.3.4 atomIDInterface	84
3.58.3.5 atomMass	84
3.58.3.6 atomRadius	84
3.58.3.7 atomType	84
3.58.3.8 ax	84
3.58.3.9 ay	84
3.58.3.10 bond	85
3.58.3.11 BondEnergy	85
3.58.3.12 BondEnergyPerAtom	85
3.58.3.13 BondID	85
3.58.3.14 BondLength	85
3.58.3.15 BondType	85
3.58.3.16 cellList	85
3.58.3.17 cells	86
3.58.3.18 cfOrg	86
3.58.3.19 cfVal	86
3.58.3.20 com	86
3.58.3.21 ComX	86
3.58.3.22 ComX0	86
3.58.3.23 ComXRatio	86
3.58.3.24 ComY	87
3.58.3.25 ComY0	87
3.58.3.26 ComYRatio	87
3.58.3.27 countAcfAv	87

3.58.3.70 fy
3.58.3.71 FyBylx
3.58.3.72 gamman
3.58.3.73 HaltCondition
3.58.3.74 histRdf
3.58.3.75 ImageX
3.58.3.76 ImageY
3.58.3.77 indexAcf
3.58.3.78 indexCorr
3.58.3.79 initUcell
3.58.3.80 isBonded
3.58.3.81 kappa
3.58.3.82 kb
3.58.3.83 kinEnergy
3.58.3.84 Kn
3.58.3.85 limitAcfAv
3.58.3.86 limitCorrAv
3.58.3.87 limitRdf
3.58.3.88 master
3.58.3.89 molID
3.58.3.90 momentum
3.58.3.91 moreCycles
3.58.3.92 nAtom
3.58.3.93 nAtomBlock
3.58.3.94 nAtomInterface
3.58.3.95 nAtomType
3.58.3.96 nBond
3.58.3.97 nBondType
3.58.3.98 nBuffAcf
3.58.3.99 nBuffCorr
3.58.3.100 nDiscInterface
3.58.3.101 nFunCorr
3.58.3.102 nodeDragx
3.58.3.103 nodeDragy
3.58.3.104 nPairActive
3.58.3.105 nPairTotal
3.58.3.106 nValAcf
3.58.3.107 nValCorr
3.58.3.108 pair
3.58.3.109 Pairatom1
3.58.3.110 Pairatom2
3.58.3.111 PairlD

3.58.3.112 PairXij
3.58.3.113 PairYij
3.58.3.114 potEnergy
3.58.3.115 prefix
3.58.3.116 pressure
3.58.3.117 RadiusIJ
3.58.3.118 RadiusIJInv
3.58.3.119 rangeRdf
3.58.3.120 rank
3.58.3.121 rCut
3.58.3.122 rdf
3.58.3.123 region
3.58.3.124 regionH
3.58.3.125 result
3.58.3.126 rfAtom
3.58.3.127 ro
3.58.3.128 rx
3.58.3.129 rxUnwrap
3.58.3.130 ry
3.58.3.131 ryUnwrap
3.58.3.132 shearDisplacement
3.58.3.133 shearVelocity
3.58.3.134 size
3.58.3.135 sizeHistRdf
3.58.3.136 sKinEnergy
3.58.3.137 solver
3.58.3.138 spacetimeCorr
3.58.3.139 spacetimeCorrAv
3.58.3.140 speed
3.58.3.141 sPotEnergy
3.58.3.142 sPressure
3.58.3.143 SqrRadiusIJ
3.58.3.144 ssKinEnergy
3.58.3.145 ssPotEnergy
3.58.3.146 ssPressure
3.58.3.147 ssTotEnergy
3.58.3.148 stepAcf
3.58.3.149 stepAvg
3.58.3.150 stepCorr
3.58.3.151 stepCount
3.58.3.152 stepDump
3.58.3.153 stepEquil 106

3.58.3.154 stepLimit
3.58.3.155 stepRdf
3.58.3.156 stepTraj
3.58.3.157 sTotEnergy
3.58.3.158 strain
3.58.3.159 strainRate
3.58.3.160 strech
3.58.3.161 stress
3.58.3.162 svirSum
3.58.3.163 thermo
3.58.3.164 timeNow
3.58.3.165 TotalBondEnergy
3.58.3.166 TotalMass
3.58.3.167 totEnergy
3.58.3.168 uSum
3.58.3.169 uSumPair
3.58.3.170 uSumPairPerAtom
3.58.3.171 virSum
3.58.3.172 virSumBond
3.58.3.173 virSumBondxx
3.58.3.174 virSumBondxy
3.58.3.175 virSumBondyy
3.58.3.176 virSumPair
3.58.3.177 virSumPairxx
3.58.3.178 virSumPairxy
3.58.3.179 virSumPairyy
3.58.3.180 virSumxx
3.58.3.181 virSumxy
3.58.3.182 virSumyy
3.58.3.183 visc
3.58.3.184 viscAcf
3.58.3.185 viscAcfAv
3.58.3.186 viscAcfInt
3.58.3.187 viscAcfOrg
3.58.3.188 VMeanSqr
3.58.3.189 vrms
3.58.3.190 VRootMeanSqr
3.58.3.191 VSqr
3.58.3.192 vSum
3.58.3.193 vSumX
3.58.3.194 vSumY
3.58.3.195 vvSum

3.58.3.196 vx
3.58.3.197 vy
3.58.3.198 xBoundary
3.58.3.199 xyz
3.58.3.200 yBoundary
3.59 global.h
3.60 source/Halt.c File Reference
3.60.1 Function Documentation
3.60.1.1 HaltConditionCheck()
3.61 Halt.c
3.62 source/Init.c File Reference
3.62.1 Function Documentation
3.62.1.1 Init()
3.63 Init.c
3.64 source/InitVacf.c File Reference
3.64.1 Function Documentation
3.64.1.1 InitVacf()
3.64.1.2 ZeroVacf()
3.65 InitVacf.c
3.66 source/Integrate.c File Reference
3.66.1 Function Documentation
3.66.1.1 Integrate()
3.67 Integrate.c
3.68 source/LeapfrogStep.c File Reference
3.68.1 Function Documentation
3.68.1.1 LeapfrogStep()
3.69 LeapfrogStep.c
3.70 source/main.c File Reference
3.70.1 Macro Definition Documentation
3.70.1.1 DEFINE_GLOBALS
3.70.2 Function Documentation
3.70.2.1 AccumProps()
3.70.2.2 ApplyBoundaryCond()
3.70.2.3 ApplyDrivingForce()
3.70.2.4 ApplyForce()
3.70.2.5 ApplyLeesEdwardsBoundaryCond()
3.70.2.6 ApplyShear()
3.70.2.7 ApplyViscous()
3.70.2.8 BrownianStep()
3.70.2.9 Close()
3.70.2.10 ComputeForcesCells()
3.70.2.11 DisplaceAtoms()

3.70.2.12 DumpBonds()	42
3.70.2.13 DumpPairs()	43
3.70.2.14 DumpRestart()	44
3.70.2.15 DumpState()	44
3.70.2.16 EvalCom()	45
3.70.2.17 EvalProps()	46
3.70.2.18 EvalRdf()	47
3.70.2.19 EvalSpacetimeCorr()	48
3.70.2.20 EvalUnwrap()	49
3.70.2.21 EvalVacf()	49
3.70.2.22 EvalVrms()	50
3.70.2.23 HaltConditionCheck()	51
3.70.2.24 Init()	51
3.70.2.25 LeapfrogStep()	54
3.70.2.26 main()	54
3.70.2.27 PrintCom()	57
3.70.2.28 PrintMomentum()	58
3.70.2.29 PrintStress()	58
3.70.2.30 PrintSummary()	59
3.70.2.31 PrintVrms()	59
3.70.2.32 SetupJob()	60
3.70.2.33 Trajectory()	61
3.70.2.34 VelocityVerletStep()	61
3.70.3 Variable Documentation	62
3.70.3.1 prefix	62
3.71 main.c	62
3.72 source/PrintCom.c File Reference	64
3.72.1 Function Documentation	65
3.72.1.1 PrintCom()	65
3.73 PrintCom.c	66
3.74 source/PrintMomentum.c File Reference	66
3.74.1 Function Documentation	67
3.74.1.1 PrintMomentum()	67
3.75 PrintMomentum.c	67
3.76 source/PrintStress.c File Reference	67
3.76.1 Function Documentation	68
3.76.1.1 PrintStress()	68
3.77 PrintStress.c	68
3.78 source/PrintSummary.c File Reference	69
3.78.1 Function Documentation	69
3.78.1.1 PrintSummary()	69
3.79 PrintSummary.c	70

185

3.80 source/PrintVacf.c File Reference
3.80.1 Function Documentation
3.80.1.1 PrintVacf()
3.81 PrintVacf.c
3.82 source/PrintVrms.c File Reference
3.82.1 Function Documentation
3.82.1.1 PrintVrms()
3.83 PrintVrms.c
3.84 source/SetupJob.c File Reference
3.84.1 Function Documentation
3.84.1.1 AccumProps()
3.84.1.2 AllocArrays()
3.84.1.3 InitVacf()
3.84.1.4 SetupJob()
3.85 SetupJob.c
3.86 source/Trajectory.c File Reference
3.86.1 Function Documentation
3.86.1.1 Trajectory()
3.87 Trajectory.c
3.88 source/VelocityVerletStep.c File Reference
3.88.1 Function Documentation
3.88.1.1 VelocityVerletStep()
3.89 VelocityVerletStep.c
3.90 source/ZeroVacf.c File Reference
3.90.1 Function Documentation
3.90.1.1 ZeroVacf()
3.91 ZeroVacf.c

Index

# Chapter 1

# Lamina: A Molecular Dynamics Package

Welcome to the Lamina documentation!

### 1.1 # Lamina

**Lamina** is a modular 2D molecular dynamics simulation package designed for simulating hybrid soft solids, including spring networks and finite-size discs. **Lamina** is a modular and extensible molecular dynamics (MD) simulation package written in C, designed to model a wide variety of soft and condensed matter systems. It supports time evolution using robust integrators and a range of thermostats, with accurate force evaluations for bonded and nonbonded interactions. Originally built for 2D simulations of bonded systems, **Lamina** has grown to support broader research goals including active matter, granular solids, and complex fluids.

### 1.2 Why "Lamina"?

The word **Lamina** comes from Latin, meaning "a thin layer", "a plate", or "a sheet". In nature and science, laminae often refer to structural elements that are flat and extended in two dimensions for example, leaves, thin metal sheets, or tissue membranes.

This name reflects both the **two-dimensional** (2D) nature of the simulations and the types of materials **Lamina** is built to study; **liquids**, **soft solids**, and **networked structures** confined to thin sheets or layers. Just as natural laminae exhibit rich structural and dynamic behaviors in a seemingly simple geometry, this code is designed to explore the complexity of emergent phenomena in 2D materials and soft matter systems.

# 1.3 Key Features

### 1.4 Interaction Potentials

Yukawa potential (screened Coulomb interactions), Lennard-Jones potential (standard 12-6), Harmonic bond potential (elastic network models), Hookean granular contact potential (for soft granular matter).

### 1.5 Thermostats and Temperature Control

Gaussian thermostat, Nose-Hoover thermostat, Langevin thermostat, Configurational temperature evaluation and control.

## 1.6 Time Integration

Velocity-Verlet integrator, Brownian (overdamped) dynamics,

### 1.7 Physical Observables

- -Radial Distribution Function (RDF) -Velocity Autocorrelation Function (VACF) -Root-Mean-Square Velocity (VRMS)
- -Stress tensor and momentum -Center-of-mass motion -Space-time correlation functions

## 1.8 Output and Utilities

The output files are saved at the ../output folder. So you have a make a directory ../ location from where you ae running ./main prefix -Structured output files (.xyz, .bond, .pair, .com, .result) -Restart and resume capability (.restart and .state files) -Clean separation of source code, unit tests, and output -Support for Lees-Edwards boundary conditions (sheared systems) -Configurable halting conditions (based on VRMS or custom metric) -Modular design for easy extension of potentials and features

### 1.9 Project Structure

## 1.9.1 Project Structure

|- README.md

```
Lamina/
 |- source/
                     # All C source files; avoid placing README.md here to prevent extra re
    |- main.c
                     # Main driver
   |- *.c, *.h
 |- unittest/
 |- output/
 |- prepros/
 |- postpros/
                    # Example postprocessing scripts (shell, python, etc.)
                    # Doxygen configuration and auxiliary files
 |- doxygen/
    |- Doxyfile
                    # Custom LaTeX header for documentation
    |- extra_stylesheet.css # Optional CSS to customize HTML output (e.g., hide Related Pages to
 |- figures/
              # All figures, logos, icons used in docs and code
    # Generated documentation output (HTML, LaTeX, PDFs)
 I- docs/
                     # Doxygen-generated HTML documentation
    |- html/
    |- latex/
                     # Doxygen-generated LaTeX source files
    |- refman.pdf
                     # Generated PDF documentation (from LaTeX)
 |- Makefile
                     # Build system
```

# Main project documentation (used as main page in Doxygen)

1.9 Project Structure 3

### ##Documentation

- Browse full HTML documentation
- Download Source code PDF manual
- User manual Physics PDF manual This documentation was generated using Doxygen 1.10.0 to ensure transparency and ease of review.

# **Chapter 2**

# **File Index**

# 2.1 File List

Here is a list of all files with brief descriptions:

source/AccumProps.c
source/AccumVacf.c
source/AllocArrays.c
source/ApplyBoundaryCond.c
source/ApplyDrivingForce.c
source/ApplyForce.c
source/ApplyLeesEdwardsBoundaryCond.c
source/ApplyShear.c
source/ApplyViscous.c
source/BrownianStep.c
source/Close.c
source/ComputeBondForce.c
source/ComputeBondForce.h
source/ComputeForcesCells.c
source/ComputePairForce.c
source/ComputePairForce.h
source/DisplaceAtoms.c
source/DumpBonds.c
source/DumpPairs.c
source/DumpRestart.c
source/DumpState.c
source/EvalCom.c
source/EvalProps.c
source/EvalRdf.c
source/EvalSpacetimeCorr.c
source/EvalUnwrap.c
source/EvalVacf.c
source/EvalVrms.c
source/global.h
source/Halt.c
source/Init.c
source/InitVacf.c
source/Integrate.c
source/LeapfrogStep.c
source/main.c

6 File Index

source/PrintCom.c																		164
source/PrintMomentum.c																		166
source/PrintStress.c																		167
source/PrintSummary.c																		169
source/PrintVacf.c																		170
source/PrintVrms.c																		172
source/SetupJob.c																		173
source/Trajectory.c																		178
source/VelocityVerletStep	.c																	180
course/Zero)/acf o																		100

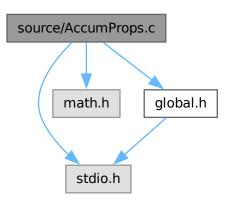
# **Chapter 3**

# **File Documentation**

# 3.1 README.md File Reference

# 3.2 source/AccumProps.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
Include dependency graph for AccumProps.c:
```



### **Functions**

• void AccumProps (int icode)

### 3.2.1 Function Documentation

#### 3.2.1.1 AccumProps()

```
void AccumProps (
          int icode )
```

Definition at line 25 of file AccumProps.c.

References kinEnergy, potEnergy, pressure, sKinEnergy, sPotEnergy, sPressure, Sqr, ssKinEnergy, ssPotEnergy, ssPressure, ssTotEnergy, stepAvg, sTotEnergy, svirSum, totEnergy, and virSum.

Referenced by SetupJob().

Here is the caller graph for this function:



# 3.3 AccumProps.c

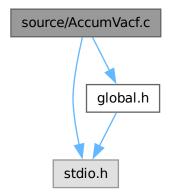
#### Go to the documentation of this file.

```
00001 /*
00002 * This file is part of Lamina.
00003 *
00004 * Lamina is free software: you can redistribute it and/or modify
00005 * it under the terms of the GNU General Public License as published by
00006 * the Free Software Foundation, either version 3 of the License, or
00007 * (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
00014 \,\, * You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<math.h>
00023 #include"global.h"
00025 void AccumProps(int icode){
00026 if(icode == 0){
00027 sPotEnergy = ssPotEnergy = 0.;
00028 sKinEnergy = ssKinEnergy = 0.;
00029 sPressure = ssPressure = 0.;
00037 sTotEnergy += totEnergy;
00038 ssTotEnergy += Sqr(totEnergy);
00039 sPressure += pressure;
00040 ssPressure += Sqr(pressure);
00041 svirSum += virSum;
00042 }else if(icode == 2) {
00043 sPotEnergy /= stepAvg;
00044 ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045 sTotEnergy /= stepAvg;
00045 sTotEnergy /= stepAvg;
00046 ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047 sKinEnergy /= stepAvg;
00048 ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049 sPressure /= stepAvg;
00050 ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051 svirSum /= stepAvg;
00052 } }
```

# 3.4 source/AccumVacf.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for AccumVacf.c:
```



### **Functions**

- double Integrate (double \*, int)
- void PrintVacf ()
- void ZeroVacf ()
- void AccumVacf ()

### 3.4.1 Function Documentation

### 3.4.1.1 AccumVacf()

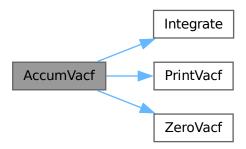
```
void AccumVacf ( )
```

Definition at line 27 of file AccumVacf.c.

References countAcfAv, deltaT, indexAcf, Integrate(), kinEnergy, limitAcfAv, nBuffAcf, nValAcf, PrintVacf(), region, stepAcf, viscAcf, viscAcfAv, viscAcfInt, and ZeroVacf().

Referenced by EvalVacf().

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.1.2 Integrate()

```
double Integrate ( \label{eq:condition} \mbox{double * $f$,} \\ \mbox{int $nf$ )}
```

Definition at line 25 of file Integrate.c.

```
00025

00026 double s;

00027 int i;

00028 s = 0.5*(f[1] + f[nf]);

00029 for(i = 2; i <= nf - 1; i ++)

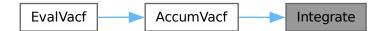
00030 s += f[i];

00031 return(s);

00032 }
```

Referenced by AccumVacf().

Here is the caller graph for this function:



### 3.4.1.3 PrintVacf()

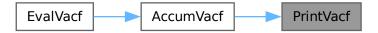
```
void PrintVacf ( )
```

### Definition at line 25 of file PrintVacf.c.

References deltaT, fpvisc, nValAcf, stepAcf, viscAcfAv, and viscAcfInt.

Referenced by AccumVacf().

Here is the caller graph for this function:



### 3.4.1.4 ZeroVacf()

```
void ZeroVacf ( )
```

Definition at line 25 of file ZeroVacf.c.

```
00025 {
00026 int j;
00027 countAcfAv= 0;
00028 for(j = 1; j <= nValAcf; j ++)
00029 viscAcfAv[j] = 0;
00030 }
```

References countAcfAv, nValAcf, and viscAcfAv.

Referenced by AccumVacf().

Here is the caller graph for this function:



### 3.5 AccumVacf.c

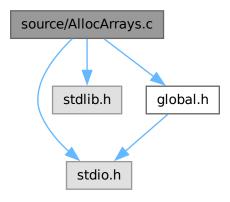
Go to the documentation of this file.

```
00001 /*
00002 * This file is part of Lamina.
00003 *
00004 * Lamina is free software: you can redistribute it and/or modify
00005 * it under the terms of the GNU General Public License as published by
00006 * the Free Software Foundation, either version 3 of the License, or
00007 * (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
00012 * GNU General Public License for more details.
00013
00014 \,\,\star\,\, You should have received a copy of the GNU General Public License
00015 \star along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00019
00020
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 double Integrate(double *, int);
00025 void PrintVacf();
00026 void ZeroVacf();
00027 void AccumVacf() {
00028 double fac;
00020 int j, nb;
00030 for(nb = 1; nb <= nBuffAcf; nb ++){
         if(indexAcf[nb] == nValAcf) {
  for(j = 1; j <= nValAcf; j ++) {
    viscAcfAv[j] += viscAcf[nb][j];</pre>
00032
00033
00034
00035
         indexAcf[nb] = 0;
         countAcfAv ++;
if (countAcfAv == limitAcfAv) {
00036
00038
          fac = 1./(kinEnergy*region[1]*region[2]*limitAcfAv);
00039
          viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
00040
          PrintVacf();
00041
          ZeroVacf();
00042 } } }
00043
```

# 3.6 source/AllocArrays.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "global.h"
Include dependency graph for AllocArrays.c:
```



#### **Functions**

· void AllocArrays ()

### 3.6.1 Function Documentation

#### 3.6.1.1 AllocArrays()

```
void AllocArrays ( )
```

Definition at line 25 of file AllocArrays.c.

```
00026
       // SPACETIME CORRELATIONS
00027
       cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00028
       for (n = 0; n <= nBuffCorr; n++)</pre>
00029
        cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00032 cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00033 indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00034
00035 spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00036 for (n = 0; n <= nBuffCorr; n++)
00037 spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00038
00039
       spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00040 // VISCOSITY
00041 indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00042 viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00043 for (n = 0 ; n \le nBuffAcf ; n ++)
00044
        viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));
00045
00046  viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00047  viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00048
00050
        histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
00051 }
```

References cfOrg, cfVal, histRdf, indexAcf, indexCorr, nBuffAcf, nBuffCorr, nFunCorr, nValAcf, nValCorr, sizeHistRdf, spacetimeCorr, spacetimeCorrAv, viscAcf, viscAcfAv, and viscAcfOrg.

Referenced by SetupJob().

Here is the caller graph for this function:



# 3.7 AllocArrays.c

#### Go to the documentation of this file.

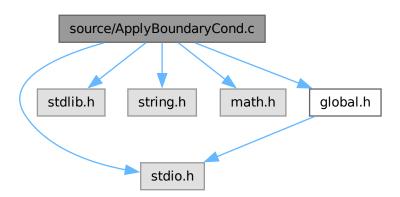
```
00002
      * This file is part of Lamina.
00003
00004
       * Lamina is free software: you can redistribute it and/or modify * it under the terms of the GNU General Public License as published by
00005
       * the Free Software Foundation, either version 3 of the License, or
00007
       * (at your option) any later version.
80000
00009 \star Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
       * GNU General Public License for more details.
00013
```

```
00014 \,\star\, You should have received a copy of the GNU General Public License
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void AllocArrays() {
00026 int n;
00027 // SPACETIME CORRELATIONS
00028 cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00029 for (n = 0; n <= nBuffCorr; n++)</pre>
        cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00030
00031
00032 cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00033 indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00034
00035 spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00036 for (n = 0; n <= nBuffCorr; n++)
00037 spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00038
00039 spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00040 // VISCOSITY
00041 indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00042 viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00043 for(n = 0; n <= nBuffAcf; n ++)
00044
        viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));
00045
00046 viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00047
       viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00048
00049
00050
        histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
00051 }
```

# 3.8 source/ApplyBoundaryCond.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for ApplyBoundaryCond.c:



#### **Functions**

· void ApplyBoundaryCond ()

### 3.8.1 Function Documentation

#### 3.8.1.1 ApplyBoundaryCond()

```
void ApplyBoundaryCond ( )
```

Definition at line 27 of file ApplyBoundaryCond.c.

```
00029
         for(n = 1 ; n <= nAtom ; n ++) {</pre>
00030
         if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){
                                                                                            // P.B.C along x and y axis
          rx[n] -= region[1]*rint(rx[n]/region[1]);
ry[n] -= region[2]*rint(ry[n]/region[2]);
} else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0) { //R.B.C. along x and y
00031
00032
00033
      axis
00034
             if((rx[n] + atomRadius[n]) >= regionH[1]){
00035
                rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n];
00036
             }if((rx[n]-atomRadius[n]) < -regionH[1]){</pre>
                rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n];
00037
00038
            if((ry[n] + atomRadius[n])>= regionH[2]){
00039
            ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n]; if((ry[n]-atomRadius[n]) < -regionH[2]){
00040
00041
00042
               ry[n] = -0.999999 * regionH[2] + atomRadius[n]; vy[n] = -vy[n];
00043
           else if (strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "r") == 0){      //P.B.C. along x and R.B.C
00044
      along v axis
00045
          rx[n] -= region[1]*rint(rx[n]/region[1]);
00046
            if((ry[n] + atomRadius[n]) >= regionH[2]){
            ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
}if((ry[n] - atomRadius[n]) < -regionH[2]){</pre>
00047
00048
                ry[n] = -0.999999 * regionH[2] + atomRadius[n]; vy[n] = -vy[n];
00049
00050
00051
           else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){     //R.B.C. along x and P.B.C
      along y axis
00052
           if((rx[n] + atomRadius[n]) >= regionH[1]){
            rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n]; 
}if((rx[n] - atomRadius[n]) < -regionH[1]){
00053
00054
              rx[n] = -0.999999 * regionH[1] + atomRadius[n]; vx[n] = -vx[n];
00055
00056
00057
             ry[n] -= region[2]*rint(ry[n]/region[2]);
00058
00059
           // Print error message and exit the program
           fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary); exit(EXIT_FAILURE); // Exit with failure status
00060
00061
00062
00063
00064 }
```

References atomRadius, fpresult, nAtom, region, regionH, rx, ry, vx, vy, xBoundary, and yBoundary.

Referenced by main().

Here is the caller graph for this function:



# 3.9 ApplyBoundaryCond.c

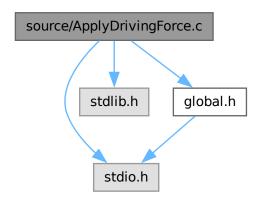
```
Go to the documentation of this file.
00001 /*
00002
       * This file is part of Lamina.
00003
00004
       * Lamina is free software: you can redistribute it and/or modify
      * it under the terms of the GNU General Public License as published by
00005
00006
       * the Free Software Foundation, either version 3 of the License, or
00007 \star (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful.
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
00014 \,\, * You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include<string.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void ApplyBoundaryCond() {
00028
        int n:
        for (n = 1 ; n \le nAtom ; n ++) {
         if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){
                                                                               // P.B.C along x and v axis
        rx[n] -= region[1]*rint(rx[n]/region[1]);
ry[n] -= region[2]*rint(ry[n]/region[2]);
} else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0) { //R.B.C. along x and y
00031
00032
00033
     axis
00034
           if((rx[n] + atomRadius[n]) >= regionH[1]){
00035
              rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n];
00036
           if((rx[n]-atomRadius[n]) < -regionH[1]){
00037
             rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n];
00038
00039
           if((ry[n] + atomRadius[n])>= regionH[2]){
              ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n];
00041
           }if((ry[n]-atomRadius[n]) < -regionH[2]){</pre>
00042
             ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n];
00043
          00044
     along y axis
00045
          rx[n] -= region[1]*rint(rx[n]/region[1]);
          if((ry[n] + atomRadius[n]) >= regionH[2]){
00046
           ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
}if((ry[n] - atomRadius[n]) < -regionH[2]){</pre>
00047
00048
               ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n];
00049
00050
          else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){ //R.B.C. along x and P.B.C
00051
     along y axis
          if((rx[n] + atomRadius[n]) >= regionH[1]){
00052
00053
             rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n];
00054
            \inf((rx[n] - atomRadius[n]) < -regionH[1]){
            rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n];
00055
00056
           ry[n] -= region[2]*rint(ry[n]/region[2]);
00058
00059
          // Print error message and exit the program
00060
          fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
00061
          exit(EXIT_FAILURE); // Exit with failure status
00062
00063
       }
00064 }
```

## 3.10 source/ApplyDrivingForce.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "global.h"
```

Include dependency graph for ApplyDrivingForce.c:



### **Functions**

• void ApplyDrivingForce ()

### 3.10.1 Function Documentation

### 3.10.1.1 ApplyDrivingForce()

```
void ApplyDrivingForce ( )
```

### Definition at line 25 of file ApplyDrivingForce.c.

```
00026 int n;
double Vxblock, Vyblock;

00027 double Vxblock, Vyblock;

00028 double Vxsubstrate, Vysubstrate;

00029 Vxblock = 0.0; Vyblock = 0.0;

00030 Vxsubstrate = 0.0; Vysubstrate = 0.0;
00031 double gammav;
00032 gammav = 0.0;
00033
00033 double count_substrate = 0;
00035 double count_block = 0;
00036
00037
            for (n = 1 ; n \le nAtom; n ++) {
             if(atomType[n] == 1 || atomType[n] == 2){
Vxsubstrate += vx[n]; Vysubstrate += vy[n];
count_substrate++;
00038
00039
00040
00041
              if (atomType[n] == 3 || atomType[n] == 4) {
Vxblock += vx[n]; Vyblock += vy[n];
00042
00043
0\,0\,0\,4\,4
              count_block++;
00045
00046
00047
              if(count_substrate > 0) {
                 Vxsubstrate /= count_substrate;
Vysubstrate /= count_substrate;
00048
00049
00050
00051
00052
              if(count_block > 0) {
  Vxblock /= count_block;
  Vyblock /= count_block;
00053
00054
00055
```

```
00056
00057
for(n = 1; n <= nAtom; n ++) {
00058
if(atomType[n] == 1 || atomType[n] == 2) {
00059
ax[n] += -gammav * (vx[n] - Vxsubstrate);
00061
ay[n] += -gammav * (vy[n] - Vysubstrate);
00061
}
00063
ax[n] += -gammav * (vx[n] - Vxblock);
00064
ay[n] += -gammav * (vy[n] - Vyblock);
00065
} } }
</pre>
```

References atomType, ax, ay, nAtom, vx, and vy.

# 3.11 ApplyDrivingForce.c

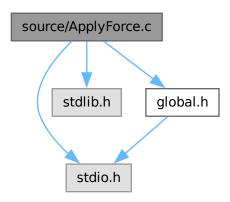
#### Go to the documentation of this file.

```
00001 /*
00002 * This file is part of Lamina.
00003 *
       * Lamina is free software: you can redistribute it and/or modify
00005
        \star it under the terms of the GNU General Public License as published by
00006 \star the Free Software Foundation, either version 3 of the License, or
00007 \star (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \star GNU General Public License for more details.
* 00013 * 00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyDrivingForce(){
00026 int n;
00027 double Vxblock, Vyblock;
00028 double Vxsubstrate, Vysubstrate;
00029 Vxblock = 0.0; Vyblock = 0.0;
00030 Vxsubstrate = 0.0; Vysubstrate = 0.0;
00031 double gammav;
00032 gammav = 0.0;
00033
00034 double count_substrate = 0;
00035 double count_block = 0;
00036
00037
        for(n = 1 ; n <= nAtom; n ++) {</pre>
        if (atomType[n] == 1 || atomType[n] == 2) {
Vxsubstrate += vx[n]; Vysubstrate += vy[n];
00038
00039
00040
         count substrate++;
00041
00042
          if(atomType[n] == 3 || atomType[n] == 4){
00043
         Vxblock += vx[n]; Vyblock += vy[n];
         count_block++;
00044
00045
         } }
00046
00047
         if(count_substrate > 0) {
          Vxsubstrate /= count_substrate;
Vysubstrate /= count_substrate;
00048
00049
00050
00051
00052
         if(count_block > 0) {
00053
          Vxblock /= count_block;
         Vyblock /= count_block;
00054
00055
00056
00057
        for (n = 1 ; n \le nAtom; n ++) {
         if(n = 1; n <= natom; n ++){
  if(atomType[n] == 1 || atomType[n] == 2){
  ax[n] += -gammav * (vx[n] - Vxsubstrate);</pre>
00058
00059
         ay[n] += -gammav * (vy[n] - Vysubstrate);
00060
00061
00062
         if(atomType[n] == 3 || atomType[n] == 4){
        00063
00064
00065
        } } }
00066
00067
```

# 3.12 source/ApplyForce.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "global.h"
```

Include dependency graph for ApplyForce.c:



### **Functions**

• void ApplyForce ()

### 3.12.1 Function Documentation

### 3.12.1.1 ApplyForce()

```
void ApplyForce ( )
```

Definition at line 25 of file ApplyForce.c.

```
00025 {
00026 int n;
00027 double lx;
00028 lx = regionH[1];
00029 fy = (FyBylx * lx)/nAtomBlock;
00030 fx = fxByfy * fy;
00031 for(n = 1; n <= nAtom; n ++) {
00032 if(molID[n] == 2) {
00033 ax[n] += fx;
00034 ay[n] -= fy;
00035 } }
```

References ax, ay, fx, fxByfy, fy, FyBylx, molID, nAtom, nAtomBlock, and regionH.

Referenced by main().

3.13 ApplyForce.c 21

Here is the caller graph for this function:



# 3.13 ApplyForce.c

#### Go to the documentation of this file.

```
00002
          * This file is part of Lamina.
00003
00004
          * Lamina is free software: you can redistribute it and/or modify
         * it under the terms of the GNU General Public License as published by * the Free Software Foundation, either version 3 of the License, or
00005
00006
00007 * (at your option) any later version.
00009 * Lamina is distributed in the hope that it will be useful,
00010 \,\,\star\, but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 \,\,\star\, MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \, * GNU General Public License for more details. 00013 \, *
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyForce(){
00026 int n;
00027 double lx;
00027 double 1x;

00028 lx = regionH[1];

00029 fy = (FyBylx * lx)/nAtomBlock;

00030 fx = fxByfy * fy;

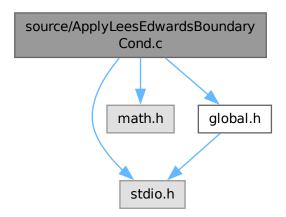
00031 for(n = 1; n <= nAtom; n ++) {

00032 if(molID[n] == 2) {
00033 ax[n] += fx;
00034 ay[n] -= fy;
00035 } }
```

# 3.14 source/ApplyLeesEdwardsBoundaryCond.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for ApplyLeesEdwardsBoundaryCond.c:



### **Functions**

void ApplyLeesEdwardsBoundaryCond ()

### 3.14.1 Function Documentation

### 3.14.1.1 ApplyLeesEdwardsBoundaryCond()

```
void ApplyLeesEdwardsBoundaryCond ( )
```

Definition at line 25 of file ApplyLeesEdwardsBoundaryCond.c.

```
00026 int n;
00027 for (n = 1; n <= nAtom; n++) {
00028 //PBC along x-direction
00029 if(rx[n] >= regionH[1])
00030 rx[n] -= region[1];
00031 else if(rx[n] < -regionH[1])
00032
               rx[n] += region[1];
00033
00034 //LEBC along y-direction
00035 if(ry[n] >= regionH[2]){
00036    rx[n] -= shearDisplacement;
             if(rx[n] < -regionH[1]) rx[n] += region[1];
//vx[n] -= shearVelocity;
ry[n] -= region[2];
}else if(ry[n] < -regionH[2]) {
rx[n] += shearDisplacement;</pre>
00038
00039
00040
00041
               if(rx[n] >= regionH[1]) rx[n] -= region[1];
//vx[n] += shearVelocity;
ry[n] += region[2];
00042
00043
00044
00045
00046
00047 }
```

References nAtom, region, regionH, rx, ry, and shearDisplacement.

## 3.15 ApplyLeesEdwardsBoundaryCond.c

## Go to the documentation of this file.

```
00002
        * This file is part of Lamina.
00003
00004
        * Lamina is free software: you can redistribute it and/or modify
00005
       * it under the terms of the GNU General Public License as published by
00006
       * the Free Software Foundation, either version 3 of the License, or
00007 \star (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
* You should have received a copy of the GNU General Public License
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include <stdio.h>
00022 #include <math.h>
00023 #include "global.h"
00025 void ApplyLeesEdwardsBoundaryCond() {
00026 int n;

00027 for (n = 1; n <= nAtom; n++) {

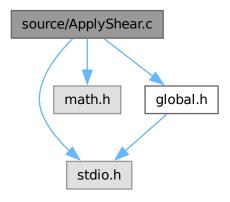
00028 //PBC along x-direction

00029 if(rx[n] >= regionH[1])
          rx[n] -= region[1];
00031 else if(rx[n] < -regionH[1])
00032
          rx[n] += region[1];
00033
00034 //LEBC along y-direction
00035
         if(ry[n] >= regionH[2]) {
  rx[n] -= shearDisplacement;
00036
00037
           if(rx[n] < -regionH[1]) rx[n] += region[1];</pre>
          //vx[n] -= shearVelocity;
ry[n] -= region[2];
00038
00039
         }else if(ry[n] < -regionH[2]){
rx[n] += shearDisplacement;</pre>
00040
00042
          if(rx[n] >= regionH[1]) rx[n] -= region[1];
00043
          //vx[n] += shearVelocity;
          ry[n] += region[2];
00044
00045
00046
00047 }
```

# 3.16 source/ApplyShear.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for ApplyShear.c:



### **Functions**

· void ApplyShear ()

### 3.16.1 Function Documentation

## 3.16.1.1 ApplyShear()

```
void ApplyShear ( )
```

### Definition at line 25 of file ApplyShear.c.

```
00025 {
00026 int n;
00027 for(n = 1; n <= nAtom; n ++) {
00028 rx[n] += strain * ry[n];
00029 //vx[n] += stranRate * ry[n];
00030 } }
```

References nAtom, rx, ry, and strain.

# 3.17 ApplyShear.c

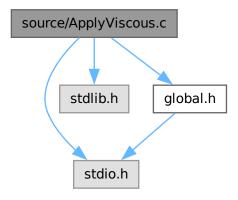
```
00001 /*
00002 * This file is part of Lamina.
00003 *
00004 * Lamina is free software: you can redistribute it and/or modify
00005 * it under the terms of the GNU General Public License as published by
0006 * the Free Software Foundation, either version 3 of the License, or
00007 * (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
00014 * You should have received a copy of the GNU General Public License
```

```
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00020
00019 */
00020
00021 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00022 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00023 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00024 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00024 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00024 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00025 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00026 #include<a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00027 #include<a href="https://www.gnu.org/licenses/">https://w
```

## 3.18 source/ApplyViscous.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "global.h"
```

Include dependency graph for ApplyViscous.c:



## **Functions**

• void ApplyViscous ()

### 3.18.1 Function Documentation

### 3.18.1.1 ApplyViscous()

```
void ApplyViscous ( )

Definition at line 25 of file ApplyViscous.c.
```

```
00025 {
00026 int n;
00027 double gammav;
00028 gammav = 1.0;
00029 for(n = 1 ; n <= nAtom; n ++) {
00030 ax[n] += -gammav * vx[n];
00031 ay[n] += -gammav * vy[n];
00032 } }
```

References ax, ay, nAtom, vx, and vy.

## 3.19 ApplyViscous.c

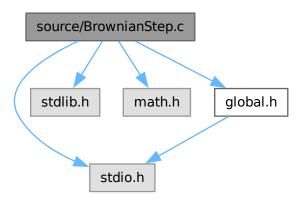
## Go to the documentation of this file.

```
00001 /
00002
        * This file is part of Lamina.
00003
00004
        * Lamina is free software: you can redistribute it and/or modify
00005
        * it under the terms of the GNU General Public License as published by
00006
        * the Free Software Foundation, either version 3 of the License, or
00007
        \star (at your option) any later version.
80000
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
       * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \,\star\, GNU General Public License for more details.
00013
* You should have received a copy of the GNU General Public License
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyViscous(){
00026 int n;
00027 double gammav;
00028 gammav = 1.0;
        for (n = 1; n <= nAtom; n ++) {</pre>
        ax[n] += -gammav * vx[n];
ay[n] += -gammav * vy[n];
00030
00031
00032
00033
00034
```

# 3.20 source/BrownianStep.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for BrownianStep.c:



3.21 BrownianStep.c 27

### **Functions**

· void BrownianStep ()

### 3.20.1 Function Documentation

### 3.20.1.1 BrownianStep()

```
void BrownianStep ( )
```

### Definition at line 26 of file BrownianStep.c.

```
if (stepCount <= stepEquil) {</pre>
00027
           double A, S1, S2, T;
00028
           int n;
S1 = 0.; S2 = 0;
00029
00030
00031
           double halfdt = 0.5*deltaT;
            for (n = 1; n <= nAtom; n++) {
00032
00033
             T = vx[n] + halfdt * ax[n];
00034
             S1 += T * ax[n];
             S2 += Sqr(T);
00035
00036
             T = vy[n] + halfdt * ay[n];
             S1 += T * ay[n];
S2 += Sqr(T);
00038
00039
00040
           A = -S1 / S2;
00041
           double C = 1 + A*deltaT;
double D = deltaT * (1 + 0.5 * A * deltaT);
00042
00043
00044
           for (n = 1; n <= nAtom; n++) {</pre>
00045
              vx[n] = C * vx[n] + D * ax[n];
              rx[n] += deltaT * vx[n];
vy[n] = C * vy[n] + D * ay[n];
00046
00047
             ry[n] += deltaT * vy[n];
00048
00050
         }else{
00051
              int n;
00052
              //SETTING TEMP = 0.0
            if (stepCount == stepEquil+1) {
for(n = 1; n <= nAtom; n ++) {</pre>
00053
00054
            vx[n] = 0.0;
00055
00056
            vy[n] = 0.0;
00057
00058
            double zeta = 1.0;
00059
            double dx, dy;
            for (n = 1; n <= nAtom; n ++) {
  dx = rx[n];</pre>
00060
00061
00062
              rx[n] += zeta * ax[n] * deltaT;
00063
             dx = rx[n] - dx;
00064
              vx[n] = dx/deltaT;
             dy = ry[n];
ry[n] += zeta * ay[n] * deltaT;
00065
00066
              dy = ry[n] - dy;
vy[n] = dy/deltaT;
00067
00069
        }
00070
00071 }
```

References ax, ay, deltaT, nAtom, rx, ry, Sqr, stepCount, stepEquil, vx, and vy.

# 3.21 BrownianStep.c

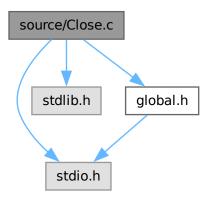
```
00001 /*
00002 * This file is part of Lamina.
00003 *
00004 * Lamina is free software: you can redistribute it and/or modify
00005 * it under the terms of the GNU General Public License as published by
00006 * the Free Software Foundation, either version 3 of the License, or
00007 * (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
```

```
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details.
00013 *
00014 \,\, * You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00029
             int n;
00030
             S1 = 0.; S2 = 0;
00031
            double halfdt = 0.5*deltaT;
            for (n = 1; n <= nAtom; n++) {
    T = vx[n] + halfdt * ax[n];
    S1 += T * ax[n];
00032
00033
00034
00035
              S2 += Sqr(T);
00036
00037
              T = vy[n] + halfdt * ay[n];
              S1 += T * ay[n];
00038
              S2 += Sqr(T);
00039
00040
00041
            A = -S1 / S2;
            double C = 1 + A*deltaT;
double D = deltaT * (1 + 0.5 * A * deltaT);
00042
00043
            for (n = 1; n <= nAtom; n++) {
    vx[n] = C * vx[n] + D * ax[n];
    rx[n] += deltaT * vx[n];
    vy[n] = C * vy[n] + D * ay[n];</pre>
00044
00045
00046
00048
               ry[n] += deltaT * vy[n];
00049
00050
         }else{
             int n;
00051
               //SETTING TEMP = 0.0
00052
              if (stepCount == stepEquil+1) {
00053
              for(n = 1; n <= nAtom; n ++) {
vx[n] = 0.0;
00054
00055
00056
              vy[n] = 0.0;
00057
              }}
00058
              double zeta = 1.0;
              double dx, dy;
for(n = 1; n <= nAtom; n ++) {</pre>
00059
00060
00061
              dx = rx[n];
00062
               rx[n] += zeta * ax[n] * deltaT;
               dx = rx[n] - dx;
vx[n] = dx/deltaT;
00063
00064
00065
               dy = ry[n];
00066
               ry[n] += zeta * ay[n] * deltaT;
00067
               dy = ry[n] - dy;
00068
               vy[n] = dy/deltaT;
00069
         }
00070
00071 }
00072
```

## 3.22 source/Close.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "global.h"
```

Include dependency graph for Close.c:



### **Functions**

• void Close ()

### 3.22.1 Function Documentation

## 3.22.1.1 Close()

void Close ( )

```
Definition at line 24 of file Close.c.
```

```
00024
00025
00026
          free(rx);
00027
          free(ry);
00028
          free (vx);
00029
          free(vy);
00030
          free(ax);
00031
          free(ay);
00032
          free(fax);
00033
          free(fay);
00034
          free(cellList);
00035
00036
          free(atomID); free(atomType); free(atomRadius); free(atomMass);
00037
          free (speed);
          free(atom1); free(atom2); free(BondID);
free(BondType); free(kb); free(ro);
free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
free(atomIDInterface);
free(PairID); free(Pairatom1); free(Pairatom2);
00038
00039
00040
00041
00042
00043
          free(PairXij); free(PairYij);
00044
00045
          free (DeltaXijOld);
00046
          free (DeltaYijOld);
00047
00048
          free (molID);
00049
00050
          for (n = 0; n <= nAtom; n++) {</pre>
00051
           free(isBonded[n]);
00052
00053
           free(isBonded);
00054
00055
00056
```

```
for (n = 0; n \le nAtom; n++) {
00058
         free (DeltaXijOldPair[n]);
00059
         free (DeltaYijOldPair[n]);
00060
00061
          free(DeltaXijOldPair);
00062
          free (DeltaYijOldPair);
00063
00064
        for (n = 0; n <= nBuffCorr; n++) {</pre>
00065
          free(cfOrg[n]);
00066
          free(spacetimeCorr[n]);
00067
00068
        free (cfOrg);
00069
        free(spacetimeCorr);
00070
        free(cfVal);
00071
        free(indexCorr);
00072
        free(spacetimeCorrAv);
00073
00074
        free(indexAcf);
00075
        free(viscAcfOrg);
00076
        free(viscAcfAv);
00077
        for (n = 0 ; n \le nBuffAcf ; n ++)
00078
          free(viscAcf[n]);
00079
        free(viscAcf);
00080
00081 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, BondID, BondType, cellList, cfOrg, cfVal, DeltaXijOld, DeltaXijOldPair, DeltaYijOld, DeltaYijOldPair, fax, fay, ImageX, ImageY, indexAcf, indexCorr, isBonded, kb, molID, nAtom, nBuffAcf, nBuffCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, ro, rx, rxUnwrap, ry, ryUnwrap, spacetimeCorr, spacetimeCorrAv, speed, viscAcf, viscAcfAv, viscAcfOrg, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.23 Close.c

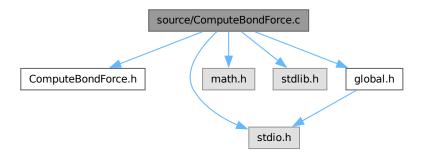
```
00001 /*
00002 * This file is part of Lamina.
00004
        \star Lamina is free software: you can redistribute it and/or modify
00005
        \star it under the terms of the GNU General Public License as published by
00006
        \star the Free Software Foundation, either version 3 of the License, or
00007
        * (at your option) any later version.
80000
00009
        * Lamina is distributed in the hope that it will be useful,
00010
          but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
        \star MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
       * GNU General Public License for more details.
00013 *
      * You should have received a copy of the GNU General Public License * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00014
00016
00017
        Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
```

```
00023 #include"global.h"
00024 void Close(){
00025
        int n;
00026
        free(rx);
00027
         free(ry);
00028
         free(vx);
         free(vy);
00030
00031
         free(ay);
00032
         free (fax);
00033
         free(fay);
00034
         free(cellList);
00035
00036
         free(atomID); free(atomType); free(atomRadius); free(atomMass);
00037
         free(speed);
         free(atom1); free(atom2); free(BondID);
free(BondType); free(kb); free(ro);
free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00038
00039
00040
00041
         free(atomIDInterface);
00042
         free(PairID); free(Pairatom1); free(Pairatom2);
00043
         free(PairXij); free(PairYij);
00044
00045
         free (DeltaXijOld);
00046
         free (DeltaYijOld);
00047
00048
         free (molID);
00049
00050
         for (n = 0; n <= nAtom; n++) {</pre>
00051
          free(isBonded[n]);
00052
00053
          free (isBonded):
00054
00055
00056
         for(n = 0; n <= nAtom; n++) {
  free(DeltaXijOldPair[n]);</pre>
00057
00058
00059
          free (DeltaYijOldPair[n]);
00060
00061
           free(DeltaXijOldPair);
00062
           free(DeltaYijOldPair);
00063
         for (n = 0; n <= nBuffCorr; n++) {</pre>
00064
           free(cfOrg[n]);
00065
           free(spacetimeCorr[n]);
00066
00067
00068
         free(cfOrg);
00069
        free(spacetimeCorr);
00070
         free(cfVal);
00071
         free (indexCorr):
         free(spacetimeCorrAv);
00072
00074
         free(indexAcf);
00075
         free(viscAcfOrg);
         free(viscAcfAv);
for(n = 0 ; n <= nBuffAcf ; n ++)
  free(viscAcf[n]);</pre>
00076
00077
00078
         free(viscAcf);
08000
00081 }
```

# 3.24 source/ComputeBondForce.c File Reference

```
#include "ComputeBondForce.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "global.h"
```

Include dependency graph for ComputeBondForce.c:



### **Functions**

00066

void ComputeBondForce ()

### 3.24.1 Function Documentation

## 3.24.1.1 ComputeBondForce()

```
void ComputeBondForce ( )
```

```
Definition at line 28 of file ComputeBondForce.c.
00029
00030
         double dr[NDIM+1], r, rr, ri, roi;
00031
         double uVal, fcVal;
00032
00033
         uVal = 0.0; TotalBondEnergy = 0.0;
         virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;
00034
00035
00036
         double vr[NDIM+1], fdVal, rri;
00037
00038
         for (n = 1 ; n <= nAtom ; n ++) {</pre>
          nodeDragx[n] = 0.0;
nodeDragy[n] = 0.0;
00039
00040
00041
         } //Important change made on 03Apr2025. Mention it in GitHub
00042
00043
         int atom1ID, atom2ID;
00044
00045
         for (n=1; n<=nBond; n++) {</pre>
         rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
atomlID = atoml[n];
atom2ID = atom2[n];
00046
00047
00048
00049
00050
           dr[1] = rx[atom1ID] - rx[atom2ID];
          if(dr[1] >= regionH[1])
dr[1] -= region[1];
else if(dr[1] < -regionH[1])</pre>
00051
00052
00053
00054
            dr[1] += region[1];
00055
           dr[2] = ry[atom1ID] - ry[atom2ID];
00056
           if(dr[2] >= regionH[2]) {
  dr[1] -= shearDisplacement;
00057
00058
            f(f[] < -regionH[1]) dr[1] += region[1];
dr[2] -= region[2];</pre>
00059
00060
00061
          }else if(dr[2] < -regionH[2]){</pre>
          dr[1] += shearDisplacement;
if(dr[1] >= regionH[1]) dr[1] -= region[1];
dr[2] += region[2];
00062
00063
00064
00065
```

```
rr = Sqr(dr[1]) + Sqr(dr[2]);
           r = sqrt(rr);
00068
00069
           rri = 1.0/rr;
00070
           ri = 1.0/r;
           roi = 1.0/ro[n];
00071
           strech = (r * roi - 1.0);
uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00072
00074
           fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
           vr[1] = vx[atom1ID] - vx[atom2ID];
vr[2] = vy[atom1ID] - vy[atom2ID];
00076
00077
00078
           fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
08000
           //DampFlag = 1. LAMMPS version
00081
            if(DampFlag == 1){
00082
           nodeDragx[atom1ID] = fdVal * dr[1]; //node-node drag //Important change made on 03Apr2025.
       Mention it in GitHub
00083
           {\color{red} nodeDragy[atom1ID] = fdVal * dr[2]; //node-node drag //Adding the drag forces is wrong. Only adding the drag forces is wrong.} \\
           nodeDragx[atom2ID] = -fdVal * dr[1]; //node-node drag //total force
nodeDragy[atom2ID] = -fdVal * dr[2]; //node-node drag
00084
00085
00086
           ax[atom1ID] += (fcVal + fdVal) * dr[1];
ay[atom1ID] += (fcVal + fdVal) * dr[2];
ax[atom2ID] += -(fcVal + fdVal) * dr[1];
00087
00088
00089
           ay[atom2ID] += -(fcVal + fdVal) * dr[2];
00090
00091
00092
00093
           //DampFlag = 2. Suzanne notes version
           else if(DampFlag == 2){
00094
           nodeDragx[atom1ID] = -gamman * vr[1]; //node-node drag
nodeDragy[atom1ID] = -gamman * vr[2]; //node-node drag
nodeDragx[atom2ID] = -(-gamman * vr[1]); //node-node drag
nodeDragy[atom2ID] = -(-gamman * vr[2]); //node-node drag
00095
00096
00097
00098
00099
           ax[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00100
           av[atomlID] += (fcVal * dr[2] - gamman * vr[2]);
ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00101
00102
00103
           ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00104
00105
           //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00106
           else if(DampFlag == 3) {
  DeltaXijNew = dr[1];
00107
00108
            DeltaYijNew = dr[2];
00109
00110
00111
             if(stepCount == 0) { // First timestep
             DeltaXijOld[n] = DeltaXijNew;
DeltaYijOld[n] = DeltaYijNew;
00112
00113
00114
00115
00116
             DeltaXij = DeltaXijNew - DeltaXijOld[n];
            DeltaYij = DeltaYijNew - DeltaYijOld[n];
DeltaVXij = DeltaXij / deltaT;
00117
00118
             DeltaVYij = DeltaYij / deltaT;
00119
00120
00121
             // Now update for the next timestep
             DeltaXijOld[n] = DeltaXijNew;
00122
00123
             DeltaYijOld[n] = DeltaYijNew;
00124
            00125
00126
00127
00128
00129
            ax[atom1ID] += (fcVal * dr[1] - gamman * DeltaVXij);
ay[atom1ID] += (fcVal * dr[2] - gamman * DeltaVYij);
ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
00130
00131
00132
             ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
00133
00134
00135
00136
           BondLength[n] = r;
BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00137
00138
00139
                                  += BondEnergy[n];
           TotalBondEnergy
00140
00141
                               0.5 * (fcVal + fdVal) * rr;
           virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00142
00143
           virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00144
00145 } }
```

References atom1, atom2, ax, ay, BondLengry, BondLength, DampFlag, deltaT, DeltaVXij, DeltaVYij, DeltaXij, DeltaXij, DeltaXijNew, DeltaXijNew, DeltaYijOld, gamman, kb, nAtom, nBond, NDIM, nodeDragx, nodeDragy, region, regionH, ro, rx, ry, shearDisplacement, Sqr, stepCount, strech, TotalBondEnergy, virSumBond, virSumBondxx, virSumBondxy, virSumBondyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



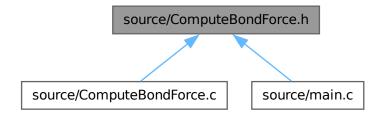
## 3.25 ComputeBondForce.c

```
00002
       * This file is part of Lamina.
00003
00004
       \star Lamina is free software: you can redistribute it and/or modify
00005
       * it under the terms of the GNU General Public License as published by
00006
       * the Free Software Foundation, either version 3 of the License, or
       * (at your option) any later version.
80000
00009
       \star Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
      * GNU General Public License for more details.
00014 \,\star\, You should have received a copy of the GNU General Public License
00015
       * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
       Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include "ComputeBondForce.h"
00022
00023 #include<stdio.h>
00024 #include<math.h>
00025 #include<stdlib.h>
00026 #include"global.h"
00027
00028 void ComputeBondForce(){
00029
        int n;
        double dr[NDIM+1], r, rr, ri, roi; double uVal, fcVal;
00030
00031
00032
00033
        uVal = 0.0; TotalBondEnergy = 0.0;
00034
        virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;
00035
00036
        double vr[NDIM+1], fdVal, rri;
00037
00038
        for(n = 1 ; n \le nAtom ; n ++) {
         nodeDragx[n] = 0.0;
nodeDragy[n] = 0.0;
00039
00040
00041
        00042
00043
        int atom1ID, atom2ID;
00044
00045
         for (n=1; n<=nBond; n++) {</pre>
00046
          rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
         atom1ID = atom1[n];
atom2ID = atom2[n];
00047
00048
00049
00050
          dr[1] = rx[atom1ID] - rx[atom2ID];
00051
          if(dr[1] >= regionH[1])
00052
           dr[1] -= region[1];
00053
          else if(dr[1] < -regionH[1])</pre>
00054
          dr[1] += region[1];
00055
00056
          dr[2] = ry[atom1ID] - ry[atom2ID];
          if(dr[2] >= regionH[2]){
00057
```

```
dr[1] -= shearDisplacement;
00059
                         if(dr[1] < -regionH[1]) dr[1] += region[1];</pre>
00060
                        dr[2] -= region[2];
                    }else if(dr[2] < -regionH[2]){</pre>
00061
                     dr[1] += shearDisplacement;
00062
00063
                         if (dr[1] >= regionH[1]) dr[1] -= region[1];
                      dr[2] += region[2];
00065
00066
00067
                     rr = Sqr(dr[1]) + Sqr(dr[2]);
00068
                     r = sqrt(rr);
00069
                     rri = 1.0/rr;
00070
                      ri = 1.0/r;
00071
                      roi = 1.0/ro[n];
                      strech = (r * roi - 1.0);
uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00072
00073
00074
                      fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
                      vr[1] = vx[atom1ID] - vx[atom2ID];
00077
                      vr[2] = vy[atom1ID] - vy[atom2ID];
00078
                      fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
08000
                      //DampFlag = 1. LAMMPS version
00081
                      if(DampFlag == 1){
00082
                      nodeDragx[atom1ID] = fdVal * dr[1]; //node-node drag //Important change made on 03Apr2025.
              Mention it in GitHub
                     \frac{1}{2} = \frac{1}{2} - \frac{1}
00083
                     nodeDragx[atom2ID] = -fdVal * dr[1]; //node-node drag //total force
nodeDragy[atom2ID] = -fdVal * dr[2]; //node-node drag
00084
00085
00086
                      ax[atom1ID] += (fcVal + fdVal) * dr[1];
ay[atom1ID] += (fcVal + fdVal) * dr[2];
00087
00088
                      ax[atom2ID] += -(fcVal + fdVal) * dr[1];
00089
00090
                      ay[atom2ID] += -(fcVal + fdVal) * dr[2];
00091
00092
                      //DampFlag = 2. Suzanne notes version
                      else if(DampFlag == 2){
00094
                     nodeDragy[atomIID] = -gamman * vr[1]; //node-node drag
nodeDragy[atomIID] = -gamman * vr[2]; //node-node drag
nodeDragy[atom2ID] = -(-gamman * vr[1]); //node-node drag
nodeDragy[atom2ID] = -(-gamman * vr[2]); //node-node drag
00095
00096
00097
00098
00099
                      ax[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
ay[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00100
00101
00102
00103
00104
00105
00106
                      //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00107
                      else if(DampFlag == 3){
00108
                        DeltaXijNew = dr[1];
                        DeltaYijNew = dr[2];
00109
00110
00111
                         if(stepCount == 0) { // First timestep
                           DeltaXijOld[n] = DeltaXijNew;
00112
00113
                           DeltaYijOld[n] = DeltaYijNew;
00114
00115
00116
                        DeltaXij = DeltaXijNew - DeltaXijOld[n];
                        DeltaYij = DeltaYijNew - DeltaYijOld[n];
00117
                        DeltaVXij = DeltaXij / deltaT;
DeltaVYij = DeltaYij / deltaT;
00118
00119
00120
00121
                         // Now update for the next timestep
00122
                        DeltaXijOld[n] = DeltaXijNew;
DeltaYijOld[n] = DeltaYijNew;
00123
00124
                        nodeDragx[atom1ID] = -gamman * DeltaVXij; //node-node drag
nodeDragy[atom1ID] = -gamman * DeltaVYij; //node-node drag
nodeDragx[atom2ID] = -(-gamman * DeltaVXij); //node-node drag
00125
00126
00127
                        nodeDragy[atom2ID] = -(-gamman * DeltaVYij); //node-node drag
00128
00129
                        ax[atom1ID] += (fcVal * dr[1] - gamman * DeltaVXij);
ay[atom1ID] += (fcVal * dr[2] - gamman * DeltaVYij);
ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
00130
00131
00132
00133
                         ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
00134
00135
00136
                      BondLength[n] = r;
BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
TotalBondEnergy += BondEnergy[n];
00137
00138
00139
00140
                      virSumBond += 0.5 * (fcVal + fdVal) * rr;
virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00141
00142
```

# 3.26 source/ComputeBondForce.h File Reference

This graph shows which files directly or indirectly include this file:



### **Functions**

• void ComputeBondForce ()

## 3.26.1 Function Documentation

### 3.26.1.1 ComputeBondForce()

```
void ComputeBondForce ( )
```

## Definition at line 28 of file ComputeBondForce.c.

```
00028
00029
         int n;
00030
         double dr[NDIM+1], r, rr, ri, roi;
00031
         double uVal, fcVal;
00032
00033
         uVal = 0.0; TotalBondEnergy = 0.0;
virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;
00034
00035
00036
         double vr[NDIM+1], fdVal, rri;
00037
00038
         for (n = 1 ; n <= nAtom ; n ++) {</pre>
         nodeDragx[n] = 0.0;
nodeDragy[n] = 0.0;
00039
00040
00041
         } //Important change made on 03Apr2025. Mention it in GitHub
00042
00043
         int atom1ID, atom2ID;
00044
00045
         for (n=1; n<=nBond; n++) {</pre>
         rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0; atomlID = atoml[n];
00046
00047
00048
          atom2ID = atom2[n];
00049
00050
          dr[1] = rx[atom1ID] - rx[atom2ID];
          if(dr[1] >= regionH[1])
dr[1] -= region[1];
else if(dr[1] < -regionH[1])</pre>
00051
00052
00053
00054
           dr[1] += region[1];
00055
```

```
dr[2] = ry[atom1ID] - ry[atom2ID];
00057
            if(dr[2] >= regionH[2]){
00058
             dr[1] -= shearDisplacement;
             if(dr[1] < -regionH[1]) dr[1] += region[1];</pre>
00059
00060
             dr[2] -= region[2];
00061
          }else if(dr[2] < -regionH[2]){</pre>
           dr[1] += shearDisplacement;
00063
              if(dr[1] >= regionH[1]) dr[1] -= region[1];
00064
            dr[2] += region[2];
00065
00066
           rr = Sqr(dr[1]) + Sqr(dr[2]);
00067
00068
           r = sqrt(rr);
00069
           rri = 1.0/rr;
00070
            ri = 1.0/r;
00071
            roi = 1.0/ro[n];
            strech = (r * roi - 1.0);
uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00072
00073
            fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
            vr[1] = vx[atom1ID] - vx[atom2ID];
vr[2] = vy[atom1ID] - vy[atom2ID];
00076
00077
00078
            fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
08000
            //DampFlag = 1. LAMMPS version
            if(DampFlag == 1) {
00081
00082
            nodeDragx[atom1ID] = fdVal * dr[1]; //node-node drag //Important change made on 03Apr2025.
       Mention it in GitHub
00083
           nodeDragy[atom1ID] = fdVal * dr[2]; //node-node drag //Adding the drag forces is wrong. Only add
       the
           nodeDragx[atom2ID] = -fdVal * dr[1]; //node-node drag //total force
nodeDragy[atom2ID] = -fdVal * dr[2]; //node-node drag
00084
00085
00086
            ax[atomlID] += (fcVal + fdVal) * dr[1];
ay[atomlID] += (fcVal + fdVal) * dr[2];
ax[atom2ID] += -(fcVal + fdVal) * dr[1];
00087
00088
00089
00090
            ay[atom2ID] += -(fcVal + fdVal) * dr[2];
00092
00093
            //DampFlag = 2. Suzanne notes version
           nodeDragx[atom1ID] = -gamman * vr[1]; //node-node drag
nodeDragx[atom2ID] = -gamman * vr[2]; //node-node drag
nodeDragx[atom2ID] = -(-gamman * vr[1]); //node-node drag
nodeDragy[atom2ID] = -(-gamman * vr[2]); //node-node drag
00094
00095
00096
00097
00098
00099
           ax[atomlID] += (fcVal * dr[1] - gamman * vr[1]);
ay[atomlID] += (fcVal * dr[2] - gamman * vr[2]);
ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00100
00101
00102
00103
00104
00105
00106
            //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
            else if (DampFlag == 3) {
  DeltaXijNew = dr[1];
00107
00108
00109
             DeltaYijNew = dr[2];
00110
00111
             if(stepCount == 0) { // First timestep
              DeltaXijOld[n] = DeltaXijNew;
DeltaYijOld[n] = DeltaYijNew;
00112
00113
00114
00115
00116
             DeltaXij = DeltaXijNew - DeltaXijOld[n];
00117
             DeltaYij = DeltaYijNew - DeltaYijOld[n];
             DeltaVXij = DeltaXij / deltaT;
00118
             DeltaVYij = DeltaYij / deltaT;
00119
00120
00121
             // Now update for the next timestep
00122
             DeltaXijOld[n] = DeltaXijNew;
             DeltaYijOld[n] = DeltaYijNew;
00123
00124
             nodeDragx[atom1ID] = -gamman * DeltaVXij; //node-node drag
nodeDragy[atom1ID] = -gamman * DeltaVYij; //node-node drag
nodeDragx[atom2ID] = -(-gamman * DeltaVXij); //node-node drag
nodeDragy[atom2ID] = -(-gamman * DeltaVXij); //node-node drag
00125
00126
00127
00128
00129
             ax[atom1ID] += (fcVal * dr[1] - gamman * DeltaVXij);
ay[atom1ID] += (fcVal * dr[2] - gamman * DeltaVYij);
ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
00130
00131
00132
             ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
00133
00134
00135
00136
00137
            BondLength[n] = r;
00138
            {\tt BondEnergy[n] = uVal; //No~0.5~factor~since~it~is~the~energy~of~the~bond}
00139
            TotalBondEnergy += BondEnergy[n];
00140
```

```
00141    virSumBond +=    0.5 * (fcVal + fdVal) * rr;
00142    virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00143    virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00144    virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00145    }
}
```

References atom1, atom2, ax, ay, BondLengry, BondLength, DampFlag, deltaT, DeltaVXij, DeltaVYij, DeltaXij, DeltaXij, DeltaXijNew, DeltaXijNew, DeltaYijOld, gamman, kb, nAtom, nBond, NDIM, nodeDragx, nodeDragy, region, regionH, ro, rx, ry, shearDisplacement, Sqr, stepCount, strech, TotalBondEnergy, virSumBond, virSumBondxx, virSumBondxy, virSumBondyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.27 ComputeBondForce.h

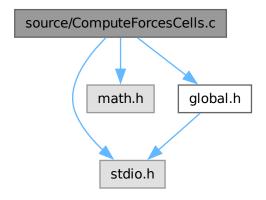
## Go to the documentation of this file.

```
00001 #ifndef COMPUTE_BOND_FORCE_H
00002 #define COMPUTE_BOND_FORCE_H
00003
00004 void ComputeBondForce();
00005
00006 #endif
```

# 3.28 source/ComputeForcesCells.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for ComputeForcesCells.c:



### **Functions**

void ComputeForcesCells ()

### 3.28.1 Function Documentation

## 3.28.1.1 ComputeForcesCells()

```
void ComputeForcesCells ( )
```

## Definition at line 25 of file ComputeForcesCells.c.

```
00025
00026
                             double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
                           int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset; int iofX[] = {0, 0, 1, 1, 0, -1, -1, 0, 1},
00027
00028
00029
                                           iofY[] = {0, 0, 0, 1, 1, 1, 0, -1, -1, -1};
00030
                           invWid[1] = cells[1]/region[1];
invWid[2] = cells[2]/region[2];
00031
00032
00033
00034
                           for(n = nAtom+1; n <= nAtom+cells[1]*cells[2]; n++)</pre>
                                   cellList[n] = 0;
00035
00036
00037
                           for (n = 1 ; n <= nAtom ; n ++) {</pre>
00038
                                  c = ((int)((rx[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n] + regionH[1])*invWid[1]) + (int)((rx[n] + regionH[1])*in
                    nAtom+ 1;
00039
                                  cellList[n] = cellList[c];
cellList[c] = n;
00040
00041
00042
00043
                             for (n = 1 ; n <= nAtom ; n ++) {</pre>
00044
                              ax[n] = 0.;
00045
                                 ay[n] = 0.;
00046
00047
00048
                           uSum = 0.0;
                           virSum = 0.0;
rfAtom = 0.0;
00049
00050
00051
                            RadiusIJ = 0.0;
00052
                           gamman = 1.0;
double vr[NDIM+1], fd, fdVal, rrinv;
00053
00054
00055
                           rrinv = 0.0;
00056
                           fd = 0.0;
```

```
00057
        fdVal = 0.0;
00058
00059
        int start = 1 + rank*(cells[2]/size);
        int end = (rank+1)*(cells[2]/size);
00060
00061
00062
        for (m1Y = start ; m1Y <= end ; m1Y ++) {</pre>
          for (m1X = 1 ; m1X <= cells[1] ; m1X ++) {</pre>
00064
            m1 = (m1Y-1) * cells[1] + m1X + nAtom;
             for(offset = 1; offset <= 9; offset ++){</pre>
00065
00066
          m2X = m1X + iofX[offset]; shift[1] = 0.;
          if(m2X > cells[1]){
00067
            m2X = 1; shift[1] = region[1];
00068
          else if(m2X == 0)
00069
00070
           m2X = cells[1]; shift[1] = -region[1];
00071
00072
           m2Y = m1Y + iofY[offset]; shift[2] = 0.;
00073
           if(m2Y > cells[2]) {
00074
            m2Y = 1; shift[2] = region[2];
           }else if(m2Y == 0){
00076
            m2Y = cells[2]; shift[2] = -region[2];
00077
00078
          m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079
          I = cellList[m1];
00080
          while (I > 0) {
00081
            J = cellList[m2];
             while (J > 0) {
00082
00083
               dr[1] = rx[I] - rx[J] - shift[1];
dr[2] = ry[I] - ry[J] - shift[2];
00084
00085
00086
                 rr = Sqr(dr[1]) + Sqr(dr[2]);
00087
                 RadiusIJ = atomRadius[I] + atomRadius[J];
00088
                 SqrRadiusIJ = Sqr(RadiusIJ);
00089
                 if(rr < SqrRadiusIJ) {</pre>
00090
               r = sqrt(rr);
00091
               ri = 1.0/r;
                       rrinv = 1.0/rr;
00092
                        vr[1] = vx[I] - vx[J];
vr[2] = vy[I] - vy[J];
00093
00095
               RadiusIJInv = 1.0/RadiusIJ;
               uVal = Sqr(1.0 - r * RadiusIJInv);
fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
00096
00097
                        fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00098
00099
00100
               f = fcVal * dr[1];
                       fd = fdVal * dr[1];
00101
00102
               ax[I] += (f + fd);
00103
                       discDragx[I] += fd; //disc-disc drag
00104
00105
               f = fcVal * dr[2]:
00106
                       fd = fdVal * dr[2];
               ay[I] += (f + fd);
00107
00108
                       discDragy[I] += fd; //disc-disc drag
00109
               uSum += 0.5 * uVal;
virSum += 0.5 * fcVal * rr;
00110
00111
               rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00112
00114
               }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
                 dr[1] = rx[I] - rx[J] - shift[1];
dr[2] = ry[I] - ry[J] - shift[2];
00115
00116
00117
                 rr = Sqr(dr[1]) + Sqr(dr[2]);
                 RadiusIJ = atomRadius[I] + atomRadius[J];
00118
00119
                 SqrRadiusIJ = Sqr(RadiusIJ);
00120
                 if(rr < SqrRadiusIJ) {</pre>
00121
               r = sqrt(rr);
00122
               ri = 1.0/r;
00123
                       rrinv = 1.0/r;
                        vr[1] = vx[I] - vx[J];
vr[2] = vy[I] - vy[J];
00124
00125
               RadiusIJInv = 1.0/RadiusIJ;
00126
               uVal = Sqr(1.0 - r * RadiusIJInv);
fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
00127
00128
                       fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00129
00130
00131
               f = fcVal * dr[1];
00132
                       fd = fdVal * dr[1];
00133
               ax[I] += (f + fd);
00134
                       discDragx[I] += fd; //disc-disc drag
00135
00136
               f = fcVal * dr[2]:
                       fd = fdVal * dr[2];
00137
               ay[I] += (f + fd);
00138
00139
                       discDragy[I] += fd; //disc-disc drag
00140
00141
               uSum += 0.5 * uVal;
               virSum += 0.5 * fcVal * rr;
rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00142
00143
```

```
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
}

    J = cellList[J];

    Graduation of the color of the
```

References atomRadius, ax, ay, cellList, cells, discDragx, discDragy, gamman, nAtom, NDIM, RadiusIJ, RadiusIJInv, rank, region, regionH, rfAtom, rx, ry, size, Sqr, SqrRadiusIJ, uSum, virSum, vx, and vy.

## 3.29 ComputeForcesCells.c

```
00002
             * This file is part of Lamina.
00003
00004 \,\star\, Lamina is free software: you can redistribute it and/or modify
00005
              * it under the terms of the GNU General Public License as published by
              * the Free Software Foundation, either version 3 of the License, or
00007 \star (at your option) any later version.
80000
00009 \,\,\star\,\, Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details.
00014 \,\,\star\,\, You should have received a copy of the GNU General Public License
00015
            * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include<stdio.h>
00022 #include<math.h>
00023 #include"global.h"
00024
00025 void ComputeForcesCells(){
00026 double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
00027
                 int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
                int iofX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
    iofY[] = {0, 0, 0, 1, 1, 1, 0, -1, -1, -1};
00028
00029
00030
00031
                invWid[1] = cells[1]/region[1];
00032
                 invWid[2] = cells[2]/region[2];
00033
00034
                 for(n = nAtom+1; n <= nAtom+cells[1]*cells[2]; n++)</pre>
00035
                  cellList[n] = 0;
00036
00037
                for(n = 1 ; n <= nAtom ; n ++) {</pre>
                    c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n] + regionH[1])*invWid[1]) + (int)((ry[n] + regionW[1]) + (int)((r
            nAtom+ 1;
                 cellList[n] = cellList[c];
cellList[c] = n;
00039
00040
00041
00042
00043
                 for (n = 1 ; n <= nAtom ; n ++) {</pre>
00044
                 ax[n] = 0.;
00045
                     ay[n] = 0.;
00046
00047
00048
                uSum = 0.0;
00049
                virSum = 0.0;
00050
                 rfAtom = 0.0;
00051
                RadiusIJ = 0.0;
00052
00053
                 gamman = 1.0:
                double vr[NDIM+1], fd, fdVal, rrinv;
00054
00055
                 rrinv = 0.0;
00056
                fd = 0.0;
00057
                fdVal = 0.0;
00058
00059
                 int start = 1 + rank*(cells[2]/size);
00060
                int end = (rank+1) * (cells[2]/size);
00061
                for (m1Y = start ; m1Y <= end ; m1Y ++) {</pre>
```

```
for (m1X = 1 ; m1X <= cells[1] ; m1X ++) {</pre>
           m1 = (m1Y-1) * cells[1] + m1X + nAtom;
for(offset = 1; offset <= 9; offset ++){
00064
00065
           m2X = m1X + iofX[offset]; shift[1] = 0.;
00066
00067
           if(m2X > cells[1]){
            m2X = 1; shift[1] = region[1];
00068
           else if(m2X == 0){
00069
00070
            m2X = cells[1]; shift[1] = -region[1];
00071
          m2Y = m1Y + iofY[offset]; shift[2] = 0.;
00072
00073
          if(m2Y > cells[2]){
00074
            m2Y = 1; shift[2] = region[2];
           else if (m2Y == 0) {
00075
00076
           m2Y = cells[2]; shift[2] = -region[2];
00077
00078
           m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079
          I = cellList[m1];
08000
          while(I > 0) {
            J = cellList[m2];
00082
             while (J > 0) {
00083
               if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
                 00084
00085
00086
00087
                 SqrRadiusIJ = Sqr(RadiusIJ);
00088
00089
                 if(rr < SqrRadiusIJ){</pre>
               r = sqrt(rr);
00090
               ri = \overline{1.0/r};
00091
00092
                        rrinv = 1.0/rr;
                        vr[1] = vx[I] - vx[J];
vr[2] = vy[I] - vy[J];
00093
00094
00095
               RadiusIJInv = 1.0/RadiusIJ;
               uVal = Sqr(1.0 - r * RadiusIJInv);
00096
               00097
00098
00099
00100
               f = fcVal * dr[1];
00101
                       fd = fdVal * dr[1];
00102
               ax[I] += (f + fd);
00103
                       discDragx[I] += fd; //disc-disc drag
00104
               f = fcVal * dr[2];
00105
00106
                       fd = fdVal * dr[2];
               ay[I] += (f + fd);
00107
00108
                       discDragy[I] += fd; //disc-disc drag
00109
               uSum += 0.5 * uVal;
00110
               virSum += 0.5 * dval,

virSum += 0.5 * fcVal * rr;

rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00111
00112
00113
00114
               }else if (m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)) {
                 dr[1] = rx[I] - rx[J] - shift[1];
dr[2] = ry[I] - ry[J] - shift[2];
rr = Sqr(dr[1]) + Sqr(dr[2]);
RadiusIJ = atomRadius[I] + atomRadius[J];
SqrRadiusIJ = Sqr(RadiusIJ);
00115
00116
00117
00118
00120
                 if(rr < SqrRadiusIJ) {</pre>
00121
               r = sqrt(rr);
00122
               ri = 1.0/r;
                        rrinv = 1.0/r;
00123
                        vr[1] = vx[I] - vx[J];
vr[2] = vy[I] - vy[J];
00124
00125
00126
               RadiusIJInv = 1.0/RadiusIJ;
00127
               uVal = Sqr(1.0 - r * RadiusIJInv);
               00128
00129
00130
00131
               f = fcVal * dr[1];
               fd = fdVal * dr[1];
ax[I] += (f + fd);
00132
00133
00134
                       discDragx[I] += fd; //disc-disc drag
00135
               f = fcVal * dr[2];
00136
                       fd = fdVal * dr[2];
00137
00138
               ay[I] += (f + fd);
00139
                       discDragy[I] += fd; //disc-disc drag
00140
               uSum += 0.5 * uVal;
00141
               virSum += 0.5 * dval;

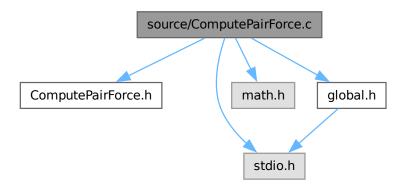
rfAtom += 0.5 * fcVal * rr;

rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00142
00143
00144
00145
00146
                   J = cellList[J];
00147
            I = cellList[I];
00148
00149
```

```
00150
00151 }
00152 }
00153 }
```

## 3.30 source/ComputePairForce.c File Reference

```
#include "ComputePairForce.h"
#include <stdio.h>
#include <math.h>
#include "global.h"
Include dependency graph for ComputePairForce.c:
```



### **Functions**

• void ComputePairForce (int normFlag)

## 3.30.1 Function Documentation

### 3.30.1.1 ComputePairForce()

```
void ComputePairForce (
          int normFlag )
```

## Definition at line 27 of file ComputePairForce.c.

```
00027

00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;

00029 int n, i, j;

00030 uVal = 0.0; uSumPair = 0.0;

00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;

00032

00033 for(n = 1; n <= nAtom; n ++) {

00034 ax[n] = 0.0;

00035 ay[n] = 0.0;

00036 discDragx[n] = 0.0;

00037 discDragy[n] = 0.0;

00038 }

00039 for(n = 1; n <= nPairTotal; n ++) {

00040 PairID[n] = 0;
```

```
00041 Pairatom1[n] = 0;
00042 Pairatom2[n] = 0;
00043 PairXij[n] = 0.0;
00044 PairYij[n] = 0.0;
00045 }
00046
00047
00048 \text{ Kn} = 1.0;
00049 double vr[NDIM+1], fdVal, rri;
00050 nPairActive = 0;
00051 double meff;
00052 \text{ meff} = 0.0;
00053 int atomIDi, atomIDj;
00054 //int processThisPair = 1;
00055
00056 for(i=1;i<=nAtomInterface;i++){</pre>
00057 for(j=i+1; j<=nAtomInterface; j++) {
        atomIDi = atomIDInterface[i];
atomIDj = atomIDInterface[j];
00058
00060
         if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
00061
         rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
00062
         RadiusIJ = 0.0;
00063
00064
         dr[1] = rx[atomIDi] - rx[atomIDj];
00065
         if(dr[1] >= regionH[1])
          dr[1] -= region[1];
00066
00067
         else if(dr[1] < -regionH[1])</pre>
00068
            dr[1] += region[1];
00069
00070
         dr[2] = ry[atomIDi] - ry[atomIDj];
          if(dr[2] >= regionH[2]) {
  dr[1] -= shearDisplacement;
00071
00072
00073
            if(dr[1] < -regionH[1]) dr[1] += region[1];
00074
            dr[2] -= region[2];
          }else if(dr[2] < -regionH[2]) {
dr[1] += shearDisplacement;</pre>
00075
00076
00077
            if(dr[1] >= regionH[1]) dr[1] -= region[1];
00078
           dr[2] += region[2];
00079
08000
         rr = Sqr(dr[1]) + Sqr(dr[2]);
RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
SqrRadiusIJ = Sqr(RadiusIJ);
00081
00082
00083
          if(rr < SqrRadiusIJ){</pre>
00084
00085
          r = sqrt(rr);
00086
           ri = 1.0/r;
00087
          rri = 1.0/rr;
00088
          RadiusIJInv = 1.0/RadiusIJ;
00089
           strech = (RadiusIJ - r);
          uVal = 0.5 * Kn * Sqr(strech);
00090
00091
00092
           //NormFlag
00093
           if(normFlag == 1){
            strech = strech * RadiusIJInv;
uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00094
00095
00096
00097
00098
           fcVal = Kn * strech * ri;
          vr[1] = vx[atomIDi] - vx[atomIDj];
vr[2] = vy[atomIDi] - vy[atomIDj];
00099
00100
00101
00102
           nPairActive++;
00103
           PairID[nPairActive] = nPairActive;
           Pairatom1[nPairActive] = atomIDi;
Pairatom2[nPairActive] = atomIDj;
00104
00105
          PairXij[nPairActive] = dr[1];
PairYij[nPairActive] = dr[2];
00106
00107
00108
00109
           //DampFlag = 1
           if(DampFlag == 1){
00110
00111
           meff = (atomMass[atomIDi]**atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
00112
           fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00113
           discDragx[atomIDi] = fdVal * dr[1]; //disc-disc drag
00114
          discDragy[atomIDi] = fdVal * dr[2]; //disc-disc drag
discDragy[atomIDj] = -fdVal * dr[1]; //disc-disc drag
discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00115
00116
00117
           discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00118
          discDragx[nPairActive] = discDragx[atomIDi];
discDragy[nPairActive] = discDragy[atomIDi];
00119
00120
00121
           ax[atomIDi] += (fcVal + fdVal) * dr[1];
ay[atomIDi] += (fcVal + fdVal) * dr[2];
00123
00124
          ax[atomIDj] += -(fcVal + fdVal) * dr[1];
ay[atomIDj] += -(fcVal + fdVal) * dr[2];
00125
00126
00127
```

```
00128
00129
         //DampFlag = 2
         else if (DampFlag == 2) {
00130
          discDragx[atomIDi] = -gamman * vr[1]; //disc-disc drag
discDragy[atomIDi] = -gamman * vr[2]; //disc-disc drag
discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00131
00132
00133
           discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00134
00135
           discDragx[nPairActive] = discDragx[atomIDi];
discDragy[nPairActive] = discDragy[atomIDi];
00136
00137
00138
00139
          ax[atomIDi] += (fcVal * dr[1] - gamman * vr[1]);
ay[atomIDi] += (fcVal * dr[2] - gamman * vr[2]);
00140
00141
00142
           ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00143
           ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00144
00145
         //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00147
         else if(DampFlag == 3){
00148
          //Track compression velocity
         DeltaXijNew = dr[1];
DeltaYijNew = dr[2];
00149
00150
          if(stepCount == 0) { // Initialization step
00151
00152
           DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
           DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00153
00154
00155
           DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
00156
           DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
00157
           DeltaVXij = DeltaXij / deltaT;
DeltaVYij = DeltaYij / deltaT;
00158
00159
00160
00161
           // Update history for next step
           DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00162
00163
00164
00165
           discDragx[atomIDi] = -gamman * DeltaVXij; //disc-disc drag
           discDragy[atomIDi] = -gamman * DeltaVYij; //disc-disc drag
discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
00166
00167
00168
           discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag
00169
00170
           discDragx[nPairActive] = discDragx[atomIDil:
           discDragy[nPairActive] = discDragy[atomIDi];
00171
00172
00173
           ax[atomIDi] += (fcVal * dr[1] - gamman * DeltaVXij);
          ay[atomIDi] += (fcVal * dr[2] - gamman * DeltaVYij);
ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
00174
00175
          ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
00176
00177
00178
          //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force = \frac{1}{2}
00179
      Hookean Interaction + relative velocity drag
00180
         uSumPair += 0.5 * uVal:
         virSumPair += 0.5 * (fcVal + fdVal) * rr;
00181
         virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00182
00184
         virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00185
00186
         else { //Resetting the distance between two discs when they are not in contact
          DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
00187
00188
00189
           DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
00190
          DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
00191
00192
00193
00194
        }
00195 }
```

References atomIDInterface, atomMass, atomRadius, ax, ay, DampFlag, deltaT, DeltaVXij, DeltaVYij, DeltaXij, DeltaXij, DeltaXij, DeltaXij, DeltaYij, DeltaYij, DeltaYijOldPair, discDragx, discDragy, gamman, isBonded, Kn, nAtom, nAtomInterface, NDIM, nPairActive, nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, RadiusIJ, RadiusIJInv, region, regionH, rx, ry, shearDisplacement, Sqr, SqrRadiusIJ, stepCount, strech, uSumPair, virSumPairxy, virSumPairxy, virSumPairxy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.31 ComputePairForce.c

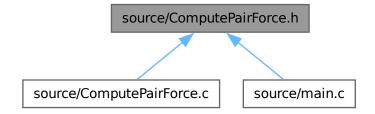
```
00002
        * This file is part of Lamina.
00003
00004
        \star Lamina is free software: you can redistribute it and/or modify
        * it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
00005
00006
00007
        * (at your option) any later version.
00009
        * Lamina is distributed in the hope that it will be useful,
00010 \,\,\star\, but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 \,\,\star\, MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include "ComputePairForce.h"
00022
00023 #include<stdio.h>
00024 #include<math.h>
00025 #include"global.h"
00027 void ComputePairForce(int normFlag) {
00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;
00029 int n, i, j;
00030 uVal = 0.0;
                        uSumPair = 0.0;
00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;
00032
00033 for(n = 1 ; n <= nAtom ; n ++) {
00034 ax[n] = 0.0;
00035 ay[n] = 0.0;
00036 discDragx[n] = 0.0;
00037 discDragy[n] = 0.0;
00038 }
00039 for(n = 1; n <= nPairTotal; n ++) {
00040 PairID[n] = 0;
00041 Pairatom1[n] = 0;
00042 Pairatom2[n] = 0;
00043 PairXij[n] = 0.0;
00044 PairYij[n] = 0.0;
00045 }
00046
00047
00048 \text{ Kn} = 1.0;
00049 double vr[NDIM+1], fdVal, rri;
00050 nPairActive = 0;
00051 double meff;
00052 \text{ meff} = 0.0;
00053 int atomIDi, atomIDj;
00054 //int processThisPair = 1;
00055
00056 for(i=1;i<=nAtomInterface;i++){
00057 for (j=i+1; j<=nAtomInterface; j++) {
        atomIDi = atomIDInterface[i];
atomIDj = atomIDInterface[j];
00059
00060
         if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
```

```
rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
00062
00063
00064
          dr[1] = rx[atomIDi] - rx[atomIDj];
00065
          if(dr[1] >= regionH[1])
          dr[1] -= region[1];
else if(dr[1] < -regionH[1])</pre>
00066
00068
             dr[1] += region[1];
00069
00070
          dr[2] = ry[atomIDi] - ry[atomIDj];
           if(dr[2] >= regionH[2]) {
  dr[1] -= shearDisplacement;
00071
00072
             dr[1] - sheartispiacement,
if(dr[1] < -regionH[1]) dr[1] += region[1];
dr[2] -= region[2];</pre>
00073
00074
00075
          }else if(dr[2] < -regionH[2]){</pre>
00076
           dr[1] += shearDisplacement;
00077
            if(dr[1] >= regionH[1]) dr[1] -= region[1];
00078
           dr[2] += region[2];
00079
08000
00081
          rr = Sqr(dr[1]) + Sqr(dr[2]);
          RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
SqrRadiusIJ = Sqr(RadiusIJ);
00082
00083
00084
          if (rr < SqrRadiusIJ) {</pre>
00085
           r = sqrt(rr);
           ri = 1.0/r;
00086
00087
            rri = 1.0/rr;
00088
           RadiusIJInv = 1.0/RadiusIJ;
00089
            strech = (RadiusIJ - r);
           uVal = 0.5 * Kn * Sqr(strech);
00090
00091
00092
            //NormFlag
00093
            if(normFlag == 1){
00094
             strech = strech * RadiusIJInv;
             uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00095
00096
00097
            fcVal = Kn * strech * ri;
vr[1] = vx[atomIDi] - vx[atomIDj];
vr[2] = vy[atomIDi] - vy[atomIDj];
00098
00099
00100
00101
00102
            nPairActive++;
            PairID[nPairActive] = nPairActive;
00103
            Pairatom1[nPairActive] = atomIDi;
Pairatom2[nPairActive] = atomIDj;
00104
00105
00106
            PairXij[nPairActive] = dr[1];
00107
            PairYij[nPairActive] = dr[2];
00108
00109
            //DampFlag = 1
            if(DampFlag == 1){
00110
00111
            meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
00112
            fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00113
           discDragx[atomIDi] = fdVal * dr[1]; //disc-disc drag
discDragy[atomIDi] = fdVal * dr[2]; //disc-disc drag
discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00114
00115
00116
            discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00118
            discDragx[nPairActive] = discDragx[atomIDi];
discDragy[nPairActive] = discDragy[atomIDi];
00119
00120
00121
00122
00123
            ax[atomIDi] += (fcVal + fdVal) * dr[1];
           ay[atomIDi] += (fcVal + fdVal) * dr[2];
ax[atomIDj] += -(fcVal + fdVal) * dr[1];
ay[atomIDj] += -(fcVal + fdVal) * dr[2];
00124
00125
00126
00127
00128
00129
          //DampFlag = 2
          else if (DampFlag == 2) {
00130
           discDragx[atomIDi] = -gamman * vr[1]; //disc-disc drag
discDragy[atomIDi] = -gamman * vr[2]; //disc-disc drag
discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00131
00132
00133
00134
00135
00136
            discDragx[nPairActive] = discDragx[atomIDi];
00137
            discDragy[nPairActive] = discDragy[atomIDi];
00138
00139
00140
           ax[atomIDi] += (fcVal * dr[1] - gamman * vr[1]);
ay[atomIDi] += (fcVal * dr[2] - gamman * vr[2]);
ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00141
00142
00143
           ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00144
00145
          //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00146
00147
          else if (DampFlag == 3) {
```

```
//Track compression velocity
          DeltaXijNew = dr[1];
DeltaYijNew = dr[2];
00149
00150
          if(stepCount == 0) { // Initialization step
00151
           DeltaXijOldPair[atomIDi] [atomIDj] = DeltaXijNew;
00152
           DeltaYijOldPair[atomIDi] [atomIDj] = DeltaYijNew;
00153
00154
00155
           DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
00156
00157
           DeltaVXij = DeltaXij / deltaT;
DeltaVYij = DeltaYij / deltaT;
00158
00159
00160
00161
            // Update history for next step
00162
           DeltaXijOldPair[atomIDi] [atomIDj] = DeltaXijNew;
           DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00163
00164
           discDragx[atomIDi] = -gamman * DeltaVXij; //disc-disc drag
discDragy[atomIDi] = -gamman * DeltaVYij; //disc-disc drag
discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
00165
00166
00167
00168
           discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag
00169
           discDragx[nPairActive] = discDragx[atomIDi];
discDragy[nPairActive] = discDragy[atomIDi];
00170
00171
00172
00173
           ax[atomIDi] += (fcVal * dr[1] - gamman * DeltaVXij);
           ay[atomIDi] += (fcVal * dr[2] - gamman * DeltaVYij);
ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
00174
00175
           ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
00176
00177
00178
00179
          //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
       Hookean Interaction + relative velocity drag
00180
          uSumPair += 0.5 * uVal;
          virSumPair += 0.5 * (fcVal + fdVal) * rr;
virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00181
00182
00183
          virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00185
00186
          else { //Resetting the distance between two discs when they are not in contact
           DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
00187
00188
           DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
00189
00190
           DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
00191
            }
00192
00193
00194
         }
00195 }
00196
00197
00198
```

## 3.32 source/ComputePairForce.h File Reference

This graph shows which files directly or indirectly include this file:



### **Functions**

void ComputePairForce (int normFlag)

### 3.32.1 Function Documentation

### 3.32.1.1 ComputePairForce()

```
void ComputePairForce (
                   int normFlag )
Definition at line 27 of file ComputePairForce.c.
00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;
00029 int n, i,j;
00030 uVal = 0.0;
                         uSumPair = 0.0;
00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairxy = 0.0; virSumPairxy = 0.0;
00032
00033 for (n = 1 ; n \le nAtom ; n ++) {
00034 ax[n] = 0.0;
00035 ay[n] = 0.0;
00036 discDragx[n] = 0.0;
00037 discDragy[n] = 0.0;
00038 }
00039 for(n = 1; n <= nPairTotal; n ++) {
00040 PairID[n] = 0;
00041 Pairatom1[n] = 0;
00042 Pairatom2[n] = 0;
00043 PairXij[n] = 0.0;
00044 PairYij[n] = 0.0;
00045 }
00046
00047
00048 \text{ Kn} = 1.0;
00049 double vr[NDIM+1], fdVal, rri;
00050 nPairActive = 0;
00051 double meff;
00052 \text{ meff} = 0.0;
00053 int atomIDi, atomIDj;
00054 //int processThisPair = 1;
00055
00056 for(i=1;i<=nAtomInterface;i++) {</pre>
00057    for(j=i+1; j<=nAtomInterface; j++) {
00058     atomIDi = atomIDInterface[i];</pre>
00059
          atomIDj = atomIDInterface[j];
          if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
00060
00061
00062
          RadiusIJ = 0.0;
00063
00064
          dr[1] = rx[atomIDi] - rx[atomIDj];
          if(dr[1] >= regionH[1])
dr[1] -= region[1];
else if(dr[1] < -regionH[1])</pre>
00065
00066
00067
00068
            dr[1] += region[1];
00069
00070
          dr[2] = ry[atomIDi] - ry[atomIDj];
          if(dr[2] >= regionH[2]) {
  dr[1] -= shearDisplacement;
00071
00072
             if(dr[1] < -regionH[1]) dr[1] += region[1];
dr[2] -= region[2];</pre>
00073
00074
00075
          }else if(dr[2] < -regionH[2]) {</pre>
          dr[1] += shearDisplacement;
  if(dr[1] >= regionH[1]) dr[1] -= region[1];
00076
00077
00078
           dr[2] += region[2];
00079
08000
00081
          rr = Sqr(dr[1]) + Sqr(dr[2]);
          RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
SqrRadiusIJ = Sqr(RadiusIJ);
00082
00083
          if (rr < SqrRadiusIJ) {</pre>
00084
00085
           r = sqrt(rr);
00086
           ri = 1.0/r;
            rri = 1.0/rr;
00087
           RadiusIJInv = 1.0/RadiusIJ;
strech = (RadiusIJ - r);
00088
00089
           uVal = 0.5 * Kn * Sqr(strech);
00090
00091
```

//NormFlag

00092

```
if(normFlag == 1){
00094
             strech = strech * RadiusIJInv;
00095
             uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00096
00097
00098
            fcVal = Kn * strech * ri;
            vr[1] = vx[atomIDi] - vx[atomIDj];
00099
00100
            vr[2] = vy[atomIDi] - vy[atomIDj];
00101
            nPairActive++;
00102
            PairID[nPairActive] = nPairActive;
00103
            Pairatom1[nPairActive] = atomIDi;
Pairatom2[nPairActive] = atomIDj;
00104
00105
00106
            PairXij[nPairActive] = dr[1];
00107
            PairYij[nPairActive] = dr[2];
00108
00109
            //DampFlag = 1
00110
            if(DampFlag == 1){
            meff = (atomMass[atomIDi]**atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
00111
00112
            fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00113
00114
            discDragx[atomIDi] = fdVal * dr[1]; //disc-disc drag
           discDragy[atomIDi] = fdVal * dr[2]; //disc-disc drag
discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00115
00116
00117
            discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00118
00119
            discDragx[nPairActive] = discDragx[atomIDi];
00120
            discDragy[nPairActive] = discDragy[atomIDi];
00121
00122
00123
           ax[atomIDi] += (fcVal + fdVal) * dr[1];
           ay[atomIDi] += (fcVal + fdVal) * dr[2];

ax[atomIDj] += -(fcVal + fdVal) * dr[1];

ay[atomIDj] += -(fcVal + fdVal) * dr[2];
00124
00125
00126
00127
00128
00129
          //DampFlag = 2
          else if(DampFlag == 2){
00130
           discDragx[atomIDi] = -gamman * vr[1]; //disc-disc drag
discDragy[atomIDi] = -gamman * vr[2]; //disc-disc drag
00131
00132
           discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00133
00134
00135
00136
            discDragx[nPairActive] = discDragx[atomIDi];
            discDragy[nPairActive] = discDragy[atomIDi];
00137
00138
00139
           ax[atomIDi] += (fcVal * dr[1] - gamman * vr[1]);
ay[atomIDi] += (fcVal * dr[2] - gamman * vr[2]);
ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00140
00141
00142
           ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00143
00144
00145
          //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version else if (DampFlag == 3) {
00146
00147
           //Track compression velocity
00148
           DeltaXijNew = dr[1];
00149
          DeltaYijNew = dr[2];
00150
00151
           if(stepCount == 0) { // Initialization step
           DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00152
00153
00154
00155
           DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
00156
00157
           DeltaVXij = DeltaXij / deltaT;
DeltaVYij = DeltaYij / deltaT;
00158
00159
00160
00161
            // Update history for next step
            DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00162
00163
            DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00164
           discDragx[atomIDi] = -gamman * DeltaVXij; //disc-disc drag
discDragy[atomIDi] = -gamman * DeltaVYij; //disc-disc drag
discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
00165
00166
00167
            discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag
00168
00169
           discDragx[nPairActive] = discDragx[atomIDi];
discDragy[nPairActive] = discDragy[atomIDi];
00170
00171
00172
00173
            ax[atomIDi] += (fcVal * dr[1] - gamman * DeltaVXij);
           aw[atomIDi] += (fcVal * dr[2] - gamman * DeltaVYij);
ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
00174
00175
00176
00177
00178
00179
          //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
```

```
Hookean Interaction + relative velocity drag
           uSumPair += 0.5 * uVal;
virSumPair += 0.5 * (fcVal + fdVal) * rr;
virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
virSumPairxy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00180
00181
00182
00183
00184
00185
00186
            else { //Resetting the distance between two discs when they are not in contact
             DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
00187
00188
00189
             DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
00190
             DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
00191
00192
00193
00194
00195 }
```

References atomIDInterface, atomMass, atomRadius, ax, ay, DampFlag, deltaT, DeltaVXij, DeltaVYij, DeltaXij, DeltaXij, DeltaXij, DeltaXij, DeltaYijOldPair, discDragx, discDragy, gamman, isBonded, Kn, nAtom, nAtomInterface, NDIM, nPairActive, nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, RadiusIJ, RadiusIJInv, region, regionH, rx, ry, shearDisplacement, Sqr, SqrRadiusIJ, stepCount, strech, uSumPair, virSumPairxx, virSumPairxy, virSumPairyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



# 3.33 ComputePairForce.h

### Go to the documentation of this file.

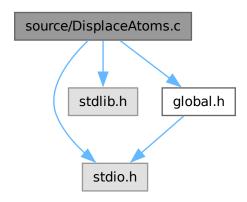
```
00001 #ifndef COMPUTE_PAIR_FORCE_H
00002 #define COMPUTE_PAIR_FORCE_H
00003
00004 void ComputePairForce(int normFlag);
00005
00006 #endif
00007
```

# 3.34 source/DisplaceAtoms.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "global.h"
```

Include dependency graph for DisplaceAtoms.c:



## **Functions**

· void DisplaceAtoms ()

## 3.34.1 Function Documentation

## 3.34.1.1 DisplaceAtoms()

```
void DisplaceAtoms ( )
```

Definition at line 25 of file DisplaceAtoms.c.

```
00025 {
00026 int n;
00027 for(n = 1; n <= nAtom; n ++) {
00028 if(molID[n] == 2) {
00029 rx[n] += DeltaX;
00030 ry[n] += DeltaY;
00031 } } }
```

References DeltaX, DeltaY, molID, nAtom, rx, and ry.

Referenced by main().

Here is the caller graph for this function:



3.35 DisplaceAtoms.c 53

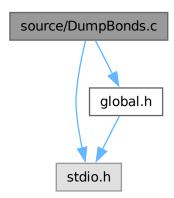
## 3.35 DisplaceAtoms.c

### Go to the documentation of this file.

```
00001 /*
00002
       * This file is part of Lamina.
00004
       * Lamina is free software: you can redistribute it and/or modify
00006 * the Free Software Foundation, either version 3 of the License, or
00007 \,\,\star\, (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
       * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \,\, \star GNU General Public License for more details. 00013 \,\, \star
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void DisplaceAtoms(){
00026 int n;
        for (n = 1; n <= nAtom; n ++) {</pre>
00027
00028
         if (molID[n] == 2) {
        rx[n] += DeltaX;
00029
00030
          ry[n] += DeltaY;
00031 } } }
```

# 3.36 source/DumpBonds.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for DumpBonds.c:
```



### **Functions**

• void DumpBonds ()

### 3.36.1 Function Documentation

### 3.36.1.1 **DumpBonds()**

```
void DumpBonds ( )
```

Definition at line 24 of file DumpBonds.c.

```
00025
           //Trajectory file in LAMMPS dump format for OVITO visualization fprintf(fpbond, "ITEM: TIMESTEP\n");
00026
00027
           fprintf(fpbond, "%lf\n",timeNow);
fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
00028
           fprintf(fpbond, "%d\n",nBond);
fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
00030
00031
           fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
00032
00033
00034
00035
           fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEqul nodeDragx1
        nodeDragy1\n");
00036
           for(n=1; n<=nBond; n++)
fprintf(fpbond, "%d %d %d %d %0.16lf %0.16lf %0.16lf %0.16lf \n", BondID[n], BondType[n], atoml[n],</pre>
00037
00038
        atom2[n],
00039
            BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
00040
```

References atom1, atom2, BondID, BondLength, BondType, fpbond, nBond, nodeDragx, nodeDragy, regionH, ro, and timeNow.

Referenced by main().

Here is the caller graph for this function:



# 3.37 DumpBonds.c

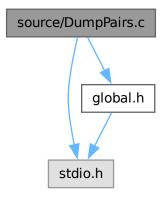
```
00001 /*
00002
         * This file is part of Lamina.
00003
00004
         * Lamina is free software: you can redistribute it and/or modify
         * it under the terms of the GNU General Public License as published by * the Free Software Foundation, either version 3 of the License, or
00005
00006
00007
         \star (at your option) any later version.
00008
00009
         \star Lamina is distributed in the hope that it will be useful,
         * but WITHOUT ANY WARRANTY; without even the implied warranty of

* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012 * GNU General Public License for more details.
00013 *
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
         Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021 #include<stdio.h>
```

```
00022 #include"global.h"
00023
00024 void DumpBonds() {
               int n;
00025
               int n;
//Trajectory file in LAMMPS dump format for OVITO visualization
fprintf(fpbond, "ITEM: TIMESTEP\n");
fprintf(fpbond, "%if\n",timeNow);
fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
fprintf(fpbond, "%d\n",nBond);
fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
fprintf(fpbond, "%If %If xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[2], regionH[2]);
fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
fprintf(fpbond, "%If %If zlo zhi\n", -0.1, 0.1);
fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atoml atom2 BondLength BondLengthEqul nodeDragx1
nodeDragy1\n");
00026
00027
00029
00030
00031
00032
00033
00034
00035
            nodeDragy1\n");
00036
                 for (n=1; n<=nBond; n++) fprintf(fpbond, "%d %d %d %0.161f %0.161f %0.161f %0.161f \n", BondID[n], BondType[n], atoml[n],
00037
00038
            atom2[n],
00039
                  BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
00040
00041
00042
00043
```

# 3.38 source/DumpPairs.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for DumpPairs.c:
```



### **Functions**

• void DumpPairs ()

## 3.38.1 Function Documentation

## 3.38.1.1 DumpPairs()

```
void DumpPairs ( )
```

Definition at line 25 of file DumpPairs.c.

```
00026
              //Trajectory file in LAMMPS dump format for OVITO visualization fprintf(fppair, "ITEM: TIMESTEP\n"); fprintf(fppair, "%lf\n",timeNow);
00027
00028
00029
             tprintf(fppair, "%lf\n",timeNow);
fprintf(fppair, "ITEM: NUMBER OF ENTRIES\n");
fprintf(fppair, "%d\n",nPairActive);
fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
fprintf(fppair, ""ITEM: ENTRIES index there's income.
00031
00032
00033
00034
00035
              fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00036
00037
00038
              for(n=1; n<=nPairActive; n++)</pre>
00039
               fprintf(fppair, "%d %d %0.161f %0.161f %0.161f %0.161f\n", PairID[n], Pairatom1[n],
          Pairatom2[n].
00040
                PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00041
00042
```

References discDragx, discDragy, fppair, nPairActive, Pairatom1, Pairatom2, PairID, PairXij, PairYij, regionH, and timeNow.

Referenced by main().

Here is the caller graph for this function:



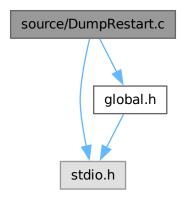
# 3.39 DumpPairs.c

```
00001 /*
       * This file is part of Lamina.
00002
00003
       \star Lamina is free software: you can redistribute it and/or modify
00004
        * it under the terms of the GNU General Public License as published by
        * the Free Software Foundation, either version 3 of the License, or
00006
00007
        * (at your option) any later version.
00008 *
00009 \star Lamina is distributed in the hope that it will be useful, 00010 \star but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013
00014 \,\,\star\,\, You should have received a copy of the GNU General Public License
00015
        * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
        Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void DumpPairs(){
00026
         //Trajectory file in LAMMPS dump format for OVITO visualization fprintf(fppair, "ITEM: TIMESTEP\n"); fprintf(fppair, "%lf\n",timeNow); fprintf(fppair, "ITEM: NUMBER OF ENTRIES\n");
00027
00028
00029
00030
         fprintf(fppair, "%d\n", nPairActive);
```

```
fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00033
00034
00035
00036
00037
             for(n=1; n<=nPairActive; n++)
fprintf(fppair, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf \n", PairID[n], Pairatom1[n],</pre>
00039
         Pairatom2[n],
00040
              PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00041
00042
00043
00044
00045
```

## 3.40 source/DumpRestart.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for DumpRestart.c:
```



### **Functions**

• void DumpRestart ()

### 3.40.1 Function Documentation

### 3.40.1.1 DumpRestart()

```
fprintf(stderr, "Error opening file %s for writing\n", DUMP);
 00032
                                                return;
 00033
 00034
                                               00035
 00036
                                                fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
fprintf(fpDUMP, "nBondType %d\n", nBondType);
 00038
 00039
                                                 fprintf(fpDUMP, "region[1] %0.141f\n", region[1]);
 00040
                                                fprintf(fpDUMP, "region[2] %0.141f\n", region[2]);
 00041
00042
 00043
                                                  int n;
 00044
                                                fprintf(fpDUMP, "Atoms\n");
 00045
                                               for(n = 1; n <= nAtom; n ++)
00046
                                                    fprintf(fpDUMP, "%d %d %0 .21f %0.161f %0.161f %0.161f %0.161f \n", atomID[n], molID[n], figure (figure for the figure for t
                                 atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n]);
00047
 00048
 00049
                                              fprintf(fpDUMP, "Bonds\n");
 00050
                                                   for (n=1; n<=nBond; n++)</pre>
                                              fprintf(fpDUMP, "%d %d %d %0 .21f %0.161f\n", BondID[n], BondType[n], atoml[n], atom2[n], kb[n], holding fine the state of the state 
 00051
                                 ro[n]);
 00052
 00053
                                                fclose(fpDUMP);
 00054 }
```

References atom1, atom2, atomID, atomRadius, atomType, BondID, BondType, kb, molID, nAtom, nAtomType, nBond, nBondType, prefix, region, ro, rx, ry, timeNow, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



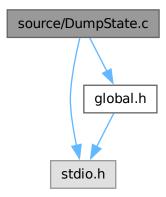
# 3.41 DumpRestart.c

```
00001 /*
00002
       \star This file is part of Lamina.
00003
00004 * Lamina is free software: you can redistribute it and/or modify
00005
        * it under the terms of the GNU General Public License as published by
        * the Free Software Foundation, either version 3 of the License, or
00007
        \star (at your option) any later version.
80000
00009
        * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
       \star GNU General Public License for more details.
00013
00014 \,\, You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
       Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include <stdio.h>
00023 #include "global.h"
00024
00025 void DumpRestart() {
```

```
00026 char DUMP[256];
00027 FILE *fpDUMP;
00028 sprintf(DUMP, "%s.Restart", prefix);
00029 fpDUMP = fopen(DUMP, "w");
00030 if(fpDUMP == NULL) {
00031 fprintf(stderr, "Error opening file %s for writing\n", DUMP);
          return;
00033 }
00034
          fprintf(fpDUMP, "timeNow %lf\n", timeNow);
fprintf(fpDUMP, "nAtom %d\n", nAtom);
fprintf(fpDUMP, "nBond %d\n", nBond);
fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
fprintf(fpDUMP, "nBondType %d\n", nBondType);
fprintf(fpDUMP, "region[1] %0.141f\n", region[1]);
fprintf(fpDUMP, "region[2] %0.141f\n", region[2]);
00035
00036
00037
00038
00039
00040
00041
00042
00043
           int n;
00044
           fprintf(fpDUMP, "Atoms\n");
00045
           for(n = 1; n <= nAtom; n ++)</pre>
            fprintf(fpDUMP, "%d %d %d %0.21f %0.161f %0.161f %0.161f %0.161f\n", atomID[n], molID[n],
        atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n]);
00047
00048
00049
          fprintf(fpDUMP, "Bonds\n");
          for (n=1; n<=nBond; n++)
00051 fprintf(fpDUMP,
                                   "%d %d %d %0.21f %0.161f\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n],
00052
00053
          fclose(fpDUMP);
00054 }
00055
```

# 3.42 source/DumpState.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for DumpState.c:
```



### **Functions**

• void DumpState ()

#### 3.42.1 Function Documentation

## 3.42.1.1 DumpState()

```
void DumpState ( )
```

Definition at line 25 of file DumpState.c.

```
00025
          char DUMP[256];
00026
00027
          FILE *fpDUMP;
00028
          sprintf(DUMP,
                               "%s.STATE", prefix);
          fpDUMP = fopen(DUMP, "w");
if(fpDUMP == NULL) {
  fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00029
00030
00031
00032
           return:
00033
00034
           00035
00036
           fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
00037
           fprintf(fpDUMP, "%d\n", nAtom);
fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp \n");
00038
00039
           fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp \n");
fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fpDUMP, "%lf %lf zlo zhi\n", -0.1, 0.1);
fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00040
00041
00042
00043
00044
           int n;
           for (n = 1; n <= nAtom; n++) {
   fprintf(fpDUMP, "%d\t %d\t %d\t %0.21f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t</pre>
00045
00047
            atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00048
00049
           fclose(fpDUMP);
00050 }
```

References atomID, atomRadius, atomType, ax, ay, molID, nAtom, prefix, regionH, rx, ry, timeNow, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



# 3.43 DumpState.c

## Go to the documentation of this file.

```
00002
       * This file is part of Lamina.
00003
00004
       * Lamina is free software: you can redistribute it and/or modify
00005
       * it under the terms of the GNU General Public License as published by
       * the Free Software Foundation, either version 3 of the License, or
00006
       \star (at your option) any later version.
80000
00009
       * Lamina is distributed in the hope that it will be useful,
       * but WITHOUT ANY WARRANTY; without even the implied warranty of * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012
       * GNU General Public License for more details.
00013
       * You should have received a copy of the GNU General Public License
```

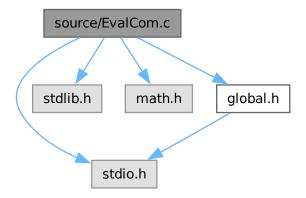
```
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
          Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include <stdio.h>
00023 #include "global.h"
00024
00025 void DumpState() {
00026 char DUMP[256];

00027 FILE *fpDUMP;

00028 sprintf(DUMP, "%s.STATE", prefix);
00029 fpDUMP = fopen(DUMP, "w");
00032
           return:
00033 }
00034
          fprintf(fpDUMP, "ITEM: TIMESTEP\n");
fprintf(fpDUMP, "%lf\n",timeNow);
fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
fprintf(fpDUMP, "%d\n",nAtom);
fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
00035
00036
00037
00038
00039
           fprintf(fpDUMP, "lem: BOX BOUNDS pp pp pp \n");
fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fpDUMP, "%lf %lf zlo zhi\n", -0.1, 0.1);
fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00040
00041
00042
00043
00044
            int n;
           for (n = 1; n <= nAtom; n++) {
    fprintf(fpDUMP, "%d\t %d\t %d\t %0.21f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t</pre>
00045
00046
00047
             atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00048
            fclose(fpDUMP);
00049
00050 }
00051
```

# 3.44 source/EvalCom.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
Include dependency graph for EvalCom.c:
```



#### **Functions**

· void EvalCom ()

#### 3.44.1 Function Documentation

#### 3.44.1.1 EvalCom()

```
void EvalCom ( )
Definition at line 27 of file EvalCom.c.
00027
00028
        int n;
ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00029
00030
        TotalMass = 0.0;
00031
00032
        for (n=1; n<=nAtom; n++) {</pre>
00033
        if(molID[n] == 2){
         ComX += atomMass[n] * rxUnwrap[n];
ComY += atomMass[n] * ryUnwrap[n];
00034
00035
00036
         TotalMass += atomMass[n];
00037
00038
        ComX = ComX/TotalMass;
ComY = ComY/TotalMass;
00039
00040
00041
00042
         if(timeNow == 0.0){
         ComX0 = ComX; ComY0 = ComY;
00044
00045
         ComXRatio = ComX/ComX0; ComYRatio = ComY/ComY0;
```

References atomMass, ComX, ComX0, ComXRatio, ComY, ComY0, ComYRatio, molID, nAtom, rxUnwrap, ryUnwrap, timeNow, and TotalMass.

Referenced by main().

00046 }

Here is the caller graph for this function:



# 3.45 EvalCom.c

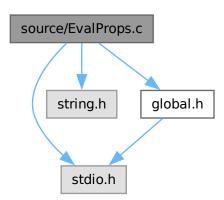
# Go to the documentation of this file.

```
00001 /*
00002 * This file is part of Lamina.
00003 *
00004 * Lamina is free software: you can redistribute it and/or modify
00005 * it under the terms of the GNU General Public License as published by
00006 * the Free Software Foundation, either version 3 of the License, or
00007 * (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
```

```
00013 *
       * You should have received a copy of the GNU General Public License * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00014
00015
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void EvalCom(){
00028 int n;
00029 ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00030 TotalMass = 0.0;
00031
00032 for (n=1; n<=nAtom; n++) {
00033
        if(molID[n] == 2){
         ComX += atomMass[n] * rxUnwrap[n];
ComY += atomMass[n] * ryUnwrap[n];
00034
00035
00036
         TotalMass += atomMass[n];
00037
00038
00039
         ComX = ComX/TotalMass;
00040
         ComY = ComY/TotalMass;
00041
00042
         if(timeNow == 0.0){
         ComX0 = ComX; ComY0 = ComY;
00043
00044
00045
         ComXRatio = ComX/ComX0; ComYRatio = ComY/ComY0;
00046 }
00047
00048
00049
```

# 3.46 source/EvalProps.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "global.h"
Include dependency graph for EvalProps.c:
```



#### **Functions**

• void EvalProps ()

#### 3.46.1 Function Documentation

#### 3.46.1.1 EvalProps()

```
void EvalProps ( )
```

Definition at line 26 of file EvalProps.c.

```
00027
        real v, vv;
00028
        virSum = 0.0;
00029
        vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
00030
        vvSum = 0.;
00031
        int n;
00032
        for (n = 1; n <= nAtom; n++) {</pre>
00033
00034
        vv = 0.;
00035
        // Initialize v with a default value to avoid "uninitialized" warning.
00036
         v = 0.0;
        // X direction velocity
if (strcmp(solver, "Verlet") == 0) {
00037
00038
        v = vx[n];
} else if (strcmp(solver, "LeapFrog") == 0) {
v = vx[n] - 0.5 * deltaT * ax[n];
00039
00040
00041
00042
00043
          vSum += v;
          vv += Sqr(v);
vSumX += v;
00044
00045
          // Y direction velocity
00046
00047
          if (strcmp(solver, "Verlet") == 0) {
          v = vy[n];
00048
00049
          } else if (strcmp(solver, "LeapFrog") == 0) {
00050
          v = vy[n] - 0.5 * deltaT * ay[n];
00051
00052
          vSum += v:
          vSumY += v;
00053
          vv += Sqr(v);
00054
00055
          vvSum += vv;
00056
00057
00058
        kinEnergy = 0.5 * vvSum / nAtom ;
00059
         uSumPairPerAtom = uSumPair / nAtom ;
00060 BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
      bond energy
00061
         potEnergy = uSumPairPerAtom + BondEnergyPerAtom ;
        totEnergy = kinEnergy + potEnergy;
virSumxx = virSumPairxx + virSumBondxx;
virSumyy = virSumPairyy + virSumBondyy;
00062
00063
00064
         virSumxy = virSumPairxy + virSumBondxy;
virSum = virSumPair + virSumBond;
00065
00066
00067
         pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00068
00069 }
```

References ax, ay, BondEnergyPerAtom, deltaT, density, kinEnergy, nAtom, NDIM, potEnergy, pressure, solver, Sqr, TotalBondEnergy, totEnergy, uSumPair, uSumPairPerAtom, virSum, virSumBond, virSumBondxx, virSumBondyy, virSumPair, virSumPairxx, virSumPairxy, virSumPairyy, virSumxx, virSumxy, virSumyy, vSum, vSumX, vSumY, vvSum, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



3.47 EvalProps.c 65

# 3.47 EvalProps.c

```
Go to the documentation of this file.
```

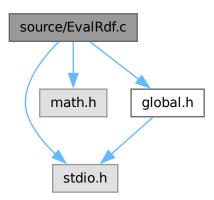
```
00002
       * This file is part of Lamina.
00003
00004
       * Lamina is free software: you can redistribute it and/or modify
00005
       * it under the terms of the GNU General Public License as published by
       * the Free Software Foundation, either version 3 of the License, or
00007
       * (at your option) any later version.
00008 *
00009 \,\,\star\,\, Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
00014
      * You should have received a copy of the GNU General Public License
00015 \star along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include <stdio.h>
00023 #include <string.h>
00024 #include "global.h"
00026 void EvalProps() {
00027 real v, vv;
00028 virSum = 0.0;
00029 vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
00030 vvSum = 0.;
00031 int n;
00032
00033 for (n = 1; n \le nAtom; n++) {
00034
        vv = 0.;
        // Initialize v with a default value to avoid "uninitialized" warning.
00035
00036
        v = 0.0;
        // X direction velocity
00038
         if (strcmp(solver, "Verlet") == 0) {
00039
          v = vx[n];
        } else if (strcmp(solver, "LeapFrog") == 0) {
v = vx[n] - 0.5 * deltaT * ax[n];
00040
00041
        }
00042
00043
          vSum += v;
          vv += Sqr(v);
00044
          vSumX += v;
00045
          // Y direction velocity
if (strcmp(solver, "Verlet") == 0) {
00046
00047
         v = vy[n];
} else if (strcmp(solver, "LeapFrog") == 0) {
00048
00049
          v = vy[n] - 0.5 * deltaT * ay[n];
00050
00051
          vSum += v;
00052
00053
          vSumY += v:
          vv += Sqr(v);
00054
00055
          vvSum += vv;
00056
00057
00058
         kinEnergy = 0.5 * vvSum / nAtom ;
         uSumPairPerAtom = uSumPair / nAtom ;
00059
        BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
00060
      bond energy
00061 potEnergy = uSumPairPerAtom + BondEnergyPerAtom;
00062
         totEnergy = kinEnergy + potEnergy;
        virSumxx = virSumPairxx + virSumBondxx;
virSumyy = virSumPairyy + virSumBondyy;
00063
00064
         virSumxy = virSumPairxy + virSumBondxy;
00065
00066
        virSum = virSumPair + virSumBond;
        pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00067
00068
00069 }
00070
```

## 3.48 source/EvalRdf.c File Reference

```
#include <stdio.h>
#include <math.h>
```

```
#include "global.h"
```

Include dependency graph for EvalRdf.c:



#### **Functions**

• void EvalRdf ()

## 3.48.1 Function Documentation

# 3.48.1.1 EvalRdf()

```
void EvalRdf ( )
```

#### Definition at line 26 of file EvalRdf.c.

```
00026
           real dr[NDIM+1], deltaR, normFac, rr, rrRange;
00027
           int j1, j2, n;
countRdf ++;
00028
00029
           if(countRdf == 1) {
  for(n = 1 ; n <= sizeHistRdf ; n ++)</pre>
00030
00031
00032
                 histRdf[n] = 0.;
00033
          rrRange = Sqr(rangeRdf);
deltaR = rangeRdf / sizeHistRdf;
for(j1 = 1; j1 <= nAtom - 1; j1 ++) {
   for(j2 = j1 + 1; j2 <= nAtom; j2 ++) {</pre>
00034
00035
00036
00037
00038
              dr[1] = rx[j1] - rx[j2];
  if(fabs(dr[1]) > regionH[1])
dr[1] -= SignR(region[1], dr[1]);
00039
00040
00041
00042
                 dr[2] = ry[j1] - ry[j2];
if(fabs(dr[2]) > regionH[2])
00043
00044
00045
              dr[2] -= SignR(region[2], dr[2]);
00046
00047
                rr = Sqr(dr[1]) + Sqr(dr[2]);
00048
00049
                 if(rr < rrRange){</pre>
00050
              n = (int)(sqrt(rr)/deltaR) + 1;
00051
              histRdf[n] ++;
00052
00053
00054
00055
           if(countRdf == limitRdf){
```

3.49 EvalRdf.c 67

```
normFac = region[1] *region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
          for (n = 1 ; n <= sizeHistRdf ; n ++)</pre>
00058
00059
            histRdf[n] *= normFac/(n-0.5);
          // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
00060
00061
          real rBin;
00062
          int n:
00063
          fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
00064
          for(n = 1; n <= sizeHistRdf; n ++) {</pre>
00065
          rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
00066
            fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00067
00068
       }
00069
00070 }
```

References countRdf, fprdf, histRdf, limitRdf, nAtom, NDIM, rangeRdf, region, regionH, rx, ry, SignR, sizeHistRdf, Sqr, and timeNow.

## 3.49 EvalRdf.c

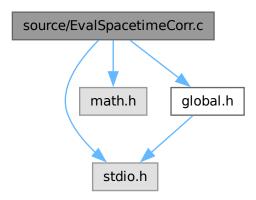
#### Go to the documentation of this file.

```
00001 /*
00002
       * This file is part of Lamina.
00003
00004
       * Lamina is free software: you can redistribute it and/or modify
00005
       \star it under the terms of the GNU General Public License as published by
00006 \star the Free Software Foundation, either version 3 of the License, or
00007 \star (at your option) any later version.
80000
00009
       * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \,\,\star\, GNU General Public License for more details.
00013 *
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void EvalRdf() {
00027
        real dr[NDIM+1], deltaR, normFac, rr, rrRange;
         int j1, j2, n;
countRdf ++;
00028
00029
00030
         if(countRdf == 1){
          for(n = 1; n <= sizeHistRdf; n ++)
histRdf[n] = 0.;</pre>
00031
00032
00033
00034
         rrRange = Sqr(rangeRdf);
         deltaR = rangeRdf / sizeHistRdf;
for(j1 = 1; j1 <= nAtom - 1; j1 ++) {
  for(j2 = j1 + 1; j2 <= nAtom; j2 ++) {</pre>
00035
00036
00037
00038
              dr[1] = rx[j1] - rx[j2];
if(fabs(dr[1]) > regionH[1])
00039
00040
00041
            dr[1] -= SignR(region[1], dr[1]);
00042
              dr[2] = ry[j1] - ry[j2];
if(fabs(dr[2]) > regionH[2])
00043
00044
            dr[2] -= SignR(region[2], dr[2]);
00045
00046
00047
              rr = Sqr(dr[1]) + Sqr(dr[2]);
00048
00049
             if(rr < rrRange){
00050
            n = (int)(sqrt(rr)/deltaR) + 1;
00051
           histRdf[n] ++;
00052
              }
00053
00054
00055
00056
         if(countRdf == limitRdf){
           normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf);
for(n = 1; n <= sizeHistRdf; n ++)</pre>
00057
00058
              histRdf[n] *= normFac/(n-0.5);
```

# 3.50 source/EvalSpacetimeCorr.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for EvalSpacetimeCorr.c:



#### **Functions**

• void EvalSpacetimeCorr ()

# 3.50.1 Function Documentation

## 3.50.1.1 EvalSpacetimeCorr()

```
void EvalSpacetimeCorr ( )
```

# Definition at line 26 of file EvalSpacetimeCorr.c.

```
00026

00027 real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
00028 real COSA, SINA, COSV, SINV;
00029 int j, m, n, nb, ni, nv;
00030 real kMin = 2. * M_PI / region[1];
00031 real kMax = M_PI;
00032 real deltaK = (kMax - kMin) / nFunCorr;
```

```
for (j = 1; j <= 2*nFunCorr; j++)
  cfVal[j] = 0.;</pre>
00034
00035
00036
00037
           for (n = 1; n \le nAtom; n++) {
00038
             i = 1;
             COSA = cos(kMin*rx[n]);
00040
             SINA = sin(kMin*rx[n]);
             for (m = 1; m <= nFunCorr; m++) {
  if (m == 1) {</pre>
00041
00042
             cosV = cos(deltaK*rx[n]);
00043
             sinV = sin(deltaK*rx[n]);
00044
             cosV0 = cosV;
00045
00046
               }else if(m == 2){
00047
             cosV1 = cosV;
             sinV1 = sinV;
cosV = 2.*cosV0*cosV1-1;
00048
00049
             sinV = 2.*cosV0*sinV1;
00050
00051
              }else{
             cosV2 = cosV1;
00052
00053
             sinV2 = sinV1;
             cosV1 = cosV;
00054
00055
             sinV1 = sinV;
             cosV = 2.*cosV0*cosV1-cosV2;
00056
00057
             sinV = 2.*cosV0*sinV1-sinV2;
00058
00059
                COSV = COSA*cosV - SINA*sinV;
00060
                SINV = SINA*cosV + COSA*sinV;
                cfVal[j] += COSV;
cfVal[j+1] += SINV;
00061
00062
00063
                j += 2;
00064
             }
00065
00066
          for (nb = 1; nb <= nBuffCorr; nb++) {
  indexCorr[nb] += 1;
  if (indexCorr[nb] <= 0) continue;</pre>
00067
00068
00069
             ni = nFunCorr * (indexCorr[nb] - 1);
             if (indexCorr[nb] == 1) {
   for (j = 1; j <= 2*nFunCorr; j++)
   cfOrg[nb][j] = cfVal[j];</pre>
00071
00072
00073
00074
00075
00076
             for (j = 1; j <= nFunCorr; j++)</pre>
00077
               spacetimeCorr[nb][ni + j] = 0.;
00078
00079
             j = 1;
             for (m = 1; m <= nFunCorr; m++) {</pre>
00080
00081
               nv = m + ni;
                spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
00082
00083
                j += 2;
00084
00085
00086
00087
00088
           // ACCUMULATE SPACETIME CORRELATIONS
           for (nb = 1; nb <= nBuffCorr; nb++) {</pre>
00090
           if (indexCorr[nb] == nValCorr) {
               for (j = 1; j <= nFunCorr*nValCorr; j++)
   spacetimeCorrAv[j] += spacetimeCorr[nb][j];</pre>
00091
00092
               indexCorr[nb] = 0.;
00093
00094
               countCorrAv ++;
00095
               if (countCorrAv == limitCorrAv) {
00096
                 for (j = 1; j <= nFunCorr*nValCorr; j++)</pre>
00097
               spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
                 pacetimeCorrav[]] /= (nAtom*limitcorrav);
fprintf(fpdnsty, "NDIM %d\n", NDIM);
fprintf(fpdnsty, "nAtom %d\n", nAtom);
fprintf(fpdnsty, "region %lf\n", region[1]);
fprintf(fpdnsty, "nFunCorr %d\n", nFunCorr);
fprintf(fpdnsty, "limitCorrav %d\n", limitCorrav);
fprintf(fpdnsty, "stepCorr %d\n", stepCorr);
fprintf(fpdnsty, "nValCorr %d\n", nValCorr);
fprintf(fpdnsty, "NvalCorr %d\n", nValCorr);
fprintf(fpdnsty, "deltaT %lf\n", deltaT);
00098
00099
00100
00101
00102
00103
00104
                 fprintf(fpdnsty, "deltaT %lf\n", deltaT);
00105
00106
                 real tVal;
                  for (n = 1; n <= nValCorr; n++) {</pre>
00107
               tVal = (n-1) *stepCorr*deltaT;
00108
00109
               fprintf (fpdnsty, "%e\t", tVal);
              00110
00111
00112
00113
00114
00115
00116
                 countCorrAv = 0.;
00117
                 for (j = 1; j <= nFunCorr*nValCorr; j++)</pre>
00118
               spacetimeCorrAv[j] = 0.;
00119
```

```
00120
00121 }
00122 }
```

References cfOrg, cfVal, countCorrAv, deltaT, fpdnsty, indexCorr, limitCorrAv, nAtom, nBuffCorr, NDIM, nFunCorr, nValCorr, region, rx, spacetimeCorr, spacetimeCorrAv, and stepCorr.

# 3.51 EvalSpacetimeCorr.c

#### Go to the documentation of this file.

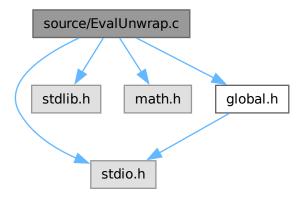
```
00002
       * This file is part of Lamina.
00003
00004 \,* Lamina is free software: you can redistribute it and/or modify
       * it under the terms of the GNU General Public License as published by
00005
00006
       * the Free Software Foundation, either version 3 of the License, or
00007 * (at your option) any later version.
80000
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details.
00013 *
00014 \,\,\star\,\, You should have received a copy of the GNU General Public License
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void EvalSpacetimeCorr (){
        real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
real COSA, SINA, COSV, SINV;
00027
00028
        int j, m, n, nb, ni, nv;
real kMin = 2. * M_PI / region[1];
real kMax = M_PI;
00029
00030
00031
        real deltaK = (kMax - kMin) / nFunCorr;
00033
00034
        for (j = 1; j <= 2*nFunCorr; j++)</pre>
00035
          cfVal[j] = 0.;
00036
00037
        for (n = 1; n \le nAtom; n++) {
00038
           i = 1;
00039
           COSA = cos(kMin*rx[n]);
00040
           SINA = sin(kMin*rx[n]);
00041
           for (m = 1; m <= nFunCorr; m++) {</pre>
00042
            if(m == 1){
           cosV = cos(deltaK*rx[n]);
00043
00044
           sinV = sin(deltaK*rx[n]);
00045
           cosV0 = cosV;
00046
             }else if(m == 2){
           cosV1 = cosV;
sinV1 = sinV;
00047
00048
00049
           cosV = 2.*cosV0*cosV1-1;
           sinV = 2.*cosV0*sinV1;
00050
00051
             }else{
00052
           cosV2 = cosV1;
           sinV2 = sinV1;
00053
           cosV1 = cosV;
00054
           sinV1 = sinV;
00055
00056
           cosV = 2.*cosV0*cosV1-cosV2;
           sinV = 2.*cosV0*sinV1-sinV2;
00057
00058
00059
              COSV = COSA*cosV - SINA*sinV;
             SINV = SINA*cosV + COSA*sinV;
00060
             cfVal[j] += COSV;
cfVal[j+1] += SINV;
00061
00062
00063
             j += 2;
00064
00065
00066
00067
         00068
00069
           if (indexCorr[nb] <= 0) continue;</pre>
           ni = nFunCorr * (indexCorr[nb] - 1);
```

```
if (indexCorr[nb] == 1) {
              for (j = 1; j <= 2*nFunCorr; j++)
cfOrg[nb][j] = cfVal[j];</pre>
00072
00073
00074
00075
              for (j = 1; j <= nFunCorr; j++)
   spacetimeCorr[nb][ni + j] = 0.;</pre>
00076
00077
00078
00079
               for (m = 1; m <= nFunCorr; m++) {</pre>
08000
                nv = m + ni;
00081
00082
                 spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
00083
                 j += 2;
00084
00085
00086
00087
00088
           // ACCUMULATE SPACETIME CORRELATIONS
           for (nb = 1; nb <= nBuffCorr; nb++) {</pre>
00089
00090
            if (indexCorr[nb] == nValCorr) {
                for (j = 1; j <= nFunCorr*nValCorr; j++)
   spacetimeCorrAv[j] += spacetimeCorr[nb][j];</pre>
00091
00092
00093
                indexCorr[nb] = 0.;
00094
                countCorrAv ++:
00095
                if (countCorrAv == limitCorrAv) {
00096
                   for (j = 1; j <= nFunCorr*nValCorr; j++)</pre>
00097
                spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
                   pacetimeCorrav[]] /= (nAtom*limitcorrav);
fprintf(fpdnsty, "NDIM %d\n", NDIM);
fprintf(fpdnsty, "nAtom %d\n", nAtom);
fprintf(fpdnsty, "region %lf\n", region[1]);
fprintf(fpdnsty, "nFunCorr %d\n", nFunCorr);
fprintf(fpdnsty, "limitCorrav %d\n", limitCorrav);
fprintf(fpdnsty, "stepCorr %d\n", stepCorr);
fprintf(fpdnsty, "nValCorr %d\n", nValCorr);
fprintf(fpdnsty, "NvalCorr %d\n", nValCorr);
fprintf(fpdnsty, "deltaT %lf\n", deltaT);
00098
00099
00100
00101
00102
00103
00104
                   \texttt{fprintf(fpdnsty,"deltaT \$lf\n", deltaT);}
00105
                real tVal;
for (n = 1; n <= nValCorr; n++) {
tVal = (n-1)*stepCorr*deltaT;</pre>
00106
00107
00109
                fprintf (fpdnsty, "%e\t", tVal);
                00110
00111
00112
00113
00114
00115
00116
                   countCorrAv = 0.;
00117
                   for (j = 1; j <= nFunCorr*nValCorr; j++)</pre>
00118
                spacetimeCorrAv[j] = 0.;
00119
00120
00121
           }
00122 }
```

# 3.52 source/EvalUnwrap.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for EvalUnwrap.c:



# **Functions**

• void EvalUnwrap ()

# 3.52.1 Function Documentation

#### 3.52.1.1 **EvalUnwrap()**

```
void EvalUnwrap ( )
```

Definition at line 27 of file EvalUnwrap.c.

References ImageX, ImageY, nAtom, region, rx, rxUnwrap, ry, and ryUnwrap.

Referenced by main().

Here is the caller graph for this function:



3.53 EvalUnwrap.c 73

# 3.53 EvalUnwrap.c

#### Go to the documentation of this file.

```
00002
          * This file is part of Lamina.
00003
00004~\star~Lamina is free software: you can redistribute it and/or modify 00005~\star~it under the terms of the GNU General Public License as published by 00006~\star~the Free Software Foundation, either version 3 of the License, or
          * (at your option) any later version.
00007
00008 *
00009 \,\star\, Lamina is distributed in the hope that it will be useful,
00010 ^{\star} but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 ^{\star} MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <math.h>
00025 #include "global.h"
00026
00027 void EvalUnwrap() {
00028 int n;
00020 int n,

00020 for (n = 1; n <= nAtom; n++) {

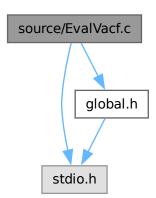
00030 rxUnwrap[n] = rx[n] + ImageX[n] * region[1];

00031 ryUnwrap[n] = ry[n] + ImageY[n] * region[2];

00032 }
00033 }
00034
```

# 3.54 source/EvalVacf.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for EvalVacf.c:
```



### **Functions**

- void AccumVacf ()
- void EvalVacf ()

# 3.54.1 Function Documentation

#### 3.54.1.1 AccumVacf()

```
void AccumVacf ( )
```

Definition at line 27 of file AccumVacf.c.

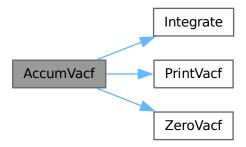
```
00027
00028
         double fac;
         int j, nb;

for(nb = 1; nb <= nBuffAcf; nb ++) {
00029
00030
           if(indexAcf[nb] == nValAcf) {
  for(j = 1 ; j <= nValAcf; j ++) {
    viscAcfAv[j] += viscAcf[nb][j];</pre>
00031
00032
00033
00034
00035
           indexAcf[nb] = 0;
00036
           countAcfAv ++;
           if(countAcfAv == limitAcfAv){
00037
            fac = 1./(kinEnergy*region[1]*region[2]*limitAcfAv);
viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
00038
00039
00040
            PrintVacf();
            ZeroVacf();
00041
00042 } } }
```

References countAcfAv, deltaT, indexAcf, Integrate(), kinEnergy, limitAcfAv, nBuffAcf, nValAcf, PrintVacf(), region, stepAcf, viscAcf, viscAcfAv, viscAcfInt, and ZeroVacf().

Referenced by EvalVacf().

Here is the call graph for this function:



Here is the caller graph for this function:



3.55 EvalVacf.c 75

### 3.54.1.2 EvalVacf()

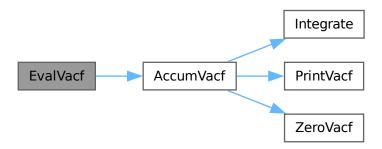
```
void EvalVacf ( )
```

#### Definition at line 26 of file EvalVacf.c.

```
00026
         int n, nb, ni;
double viscVec = 0.;
00027
00028
00029
         double v[3];
00030
         for (n = 1 ; n <= nAtom ; n ++) {</pre>
            v[1] = vx[n] - 0.5*ax[n]*deltaT;
v[2] = vy[n] - 0.5*ay[n]*deltaT;
00031
00032
00033
            viscVec += v[1]*v[2];
00034
00035
         viscVec += rfAtom;
         for(nb = 1 ; nb <= nBuffAcf ; nb ++) {</pre>
00036
00037
            indexAcf[nb] ++;
            if(indexAcf[nb] <= 0) continue;
if(indexAcf[nb] == 1) {</pre>
00038
00039
00040
              viscAcfOrg[nb] = viscVec;
00041
00042
            ni = indexAcf[nb];
00043
            viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
00044
00045
         AccumVacf():
00046 }
```

References AccumVacf(), ax, ay, deltaT, indexAcf, nAtom, nBuffAcf, rfAtom, viscAcf, viscAcfOrg, vx, and vy.

Here is the call graph for this function:



# 3.55 EvalVacf.c

# Go to the documentation of this file.

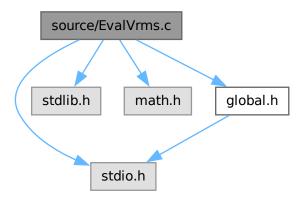
```
00001 /*
00002
        * This file is part of Lamina.
00003
         * Lamina is free software: you can redistribute it and/or modify * it under the terms of the GNU General Public License as published by
00004
00005
00006
          \star the Free Software Foundation, either version 3 of the License, or
00007
          \star (at your option) any later version.
80000
00009
        * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
         * GNU General Public License for more details.
00013
00014 \,\, You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
```

```
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00025 void AccumVacf();
00026 void EvalVacf(){
00027
         int n, nb, ni;
00028
         double viscVec = 0.;
         double visovec = 0.;
double v[3];
for(n = 1; n <= nAtom; n ++) {
  v[1] = vx[n] - 0.5*ax[n]*deltaT;
  v[2] = vy[n] - 0.5*ay[n]*deltaT;</pre>
00029
00030
00031
00032
            viscVec += v[1] *v[2];
00033
00034
00035
          viscVec += rfAtom;
for(nb = 1; nb <= nBuffAcf; nb ++) {</pre>
00036
            indexAcf[nb] ++;
00037
            if(indexAcf[nb] <= 0) continue;
if(indexAcf[nb] == 1) {</pre>
00038
00039
00040
               viscAcfOrg[nb] = viscVec;
00041
            ni = indexAcf[nb];
00042
00043
            viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
00044
00045
         AccumVacf();
00046 }
```

# 3.56 source/EvalVrms.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for EvalVrms.c:



### **Functions**

• void EvalVrms ()

3.57 EvalVrms.c 77

## 3.56.1 Function Documentation

#### 3.56.1.1 EvalVrms()

```
void EvalVrms ( )
Definition at line 27 of file EvalVrms.c.
00027
00028
         int n;
         VSqr = 0.0;
00029
00030
         VMeanSqr = 0.0;
00031
         VRootMeanSqr = 0.0;
00032
00033
         for (n = 1 ; n <= nAtom ; n ++) {
VSqr += Sqr(vx[n]) + Sqr(vy[n]);</pre>
00034
00035
00036
         VMeanSqr = VSqr/nAtom;
00037
         VRootMeanSqr = sqrt(VMeanSqr);
00038
```

References nAtom, Sqr, VMeanSqr, VRootMeanSqr, VSqr, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.57 EvalVrms.c

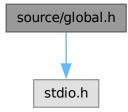
#### Go to the documentation of this file.

```
00001 /*
00002 * This file is part of Lamina.
00003
00004
        * Lamina is free software: you can redistribute it and/or modify
00005
        \star it under the terms of the GNU General Public License as published by
00006 \star the Free Software Foundation, either version 3 of the License, or
00007
       * (at your option) any later version.
00008 *
00009
       * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \star GNU General Public License for more details.
00013 *
* You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
        Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void EvalVrms(){
00028
       int n;
00029
         VSqr = 0.0;
```

```
00030    VMeanSqr = 0.0;
00031    VRootMeanSqr = 0.0;
00032
00033    for(n = 1 ; n <= nAtom ; n ++) {
00034     VSqr += Sqr(vx[n]) + Sqr(vy[n]);
00035    }
00036    VMeanSqr = VSqr/nAtom;
00037    VRootMeanSqr = sqrt(VMeanSqr);
00038    }
00039
00040
00041</pre>
```

# 3.58 source/global.h File Reference

```
#include <stdio.h>
Include dependency graph for global.h:
```



This graph shows which files directly or indirectly include this file:

## Macros

- #define EXTERN extern
- #define NDIM 2
- #define Sqr(x) ((x) \* (x))
- #define SignR(x, y) (((y) >= 0) ? (x) : (- (x)))

# **Typedefs**

• typedef double real

#### **Variables**

- double \* rx
- double \* ry
- double \* vx
- double \* vy
- double \* ax
- double \* ay
- double \* speed
- double region [2+1]
- double regionH [2+1]
- double deltaT
- double timeNow
- double potEnergy
- double kinEnergy
- double totEnergy
- · double density
- · double pressure
- double rCut
- · double kappa
- double uSum
- · double virSum
- double svirSum
- double vSum
- double vSumX
- double vSumY
- double vvSum
- double sPotEnergy
- double sKinEnergy
- double sTotEnergy
- · double sPressure
- double ssPotEnergy
- double ssKinEnergy
- double ssTotEnergy
- double ssPressure
- int initUcell [2+1]
- · int moreCycles
- int nAtom
- · int stepAvg
- · int stepCount
- · int stepEquil
- · int stepLimit
- int stepTraj
- int stepDump
- double RadiusIJ
- double SqrRadiusIJ
- double RadiusIJInv
- int nAtomType
- int \* atomType
- int \* atomID
- double \* atomRadius
- double \* atomMass
- double TotalMass
- int nBond
- int nBondType

- int \* atom1
- int \* atom2
- int \* BondID
- int \* BondType
- double \* kb
- double \* ro
- double \* BondEnergy
- double \* BondLength
- double TotalBondEnergy
- double BondEnergyPerAtom
- · double gamman
- double \* discDragx
- double \* discDragy
- double \* nodeDragx
- double \* nodeDragy
- · double strain
- double strainRate
- · double shearDisplacement
- double shearVelocity
- · double VSqr
- double VMeanSqr
- double VRootMeanSqr
- · double ComX
- · double ComY
- double ComX0
- double ComY0
- · double ComXRatio
- · double ComYRatio
- double HaltCondition
- · double DeltaY
- double DeltaX
- int \* ImageX
- int \* ImageY
- double \* rxUnwrap
- double \* ryUnwrap
- int nAtomInterface
- int nDiscInterface
- int nAtomBlock
- int \* atomIDInterface
- double Kn
- · double fx
- · double fy
- · double FyBylx
- double fxByfy
- · int DampFlag
- · double strech
- int dumpPairFlag
- int nPairTotal
- int nPairActive
- int \* PairID
- int \* Pairatom1
- int \* Pairatom2
- double \* PairXij
- double \* PairYij
- char solver [128]

- char xBoundary [10]
- char yBoundary [10]
- · char thermo
- double \* DeltaXijOld
- double \* DeltaYijOld
- double DeltaXijNew
- · double DeltaYijNew
- double DeltaXij
- · double DeltaYij
- · double DeltaVXii
- · double DeltaVYij
- double \*\* DeltaXijOldPair
- double \*\* DeltaYijOldPair
- int \* mollD
- int \*\* isBonded
- int \* cellList
- int cells [2+1]
- · int rank
- · int size
- · int master
- double \* fax
- double \* fay
- · double fuSum
- · double fvirSum
- double frfAtom
- · double uSumPair
- · double uSumPairPerAtom
- double virSumPair
- double virSumPairxx
- · double virSumPairyy
- double virSumPairxy
- · double virSumBond
- double virSumBondxx
- · double virSumBondyy
- double virSumBondxy
- double virSumxx
- double virSumyy
- double virSumxy
- int freezeAtomType
- double \*\* cfOrg
- double \*\* spacetimeCorr
- double \* cfVal
- double \* spacetimeCorrAv
- int \* indexCorr
- int countCorrAv
- int limitCorrAv
- int nBuffCorr
- · int nFunCorr
- · int nValCorr
- · int stepCorr
- double rfAtom
- double \* indexAcf
- double \*\* viscAcf
- double \* viscAcfOrg
- double \* viscAcfAv

- double viscAcfInt
- int nValAcf
- int nBuffAcf
- int stepAcf
- · int countAcfAv
- · int limitAcfAv
- double \* histRdf
- · double rangeRdf
- · int countRdf
- · int limitRdf
- int sizeHistRdf
- int stepRdf
- char \* prefix
- char result [250]
- FILE \* fpresult
- char xyz [256]
- FILE \* fpxyz
- char bond [256]
- FILE \* fpbond
- char dump [256]
- FILE \* fpdump
- char dnsty [256]
- FILE \* fpdnsty
- char visc [256]
- FILE \* fpvisc
- char rdf [256]
- FILE \* fprdf
- char vrms [256]
- FILE \* fpvrms
- char stress [256]
- FILE \* fpstress
- char momentum [256]
- FILE \* fpmomentum
- char com [256]
- FILE \* fpcom
- char pair [256]
- FILE \* fppair

# 3.58.1 Macro Definition Documentation

## 3.58.1.1 EXTERN

#define EXTERN extern

Definition at line 8 of file global.h.

# 3.58.1.2 NDIM

#define NDIM 2

Definition at line 13 of file global.h.

Referenced by ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), EvalProps(), EvalRdf(), EvalSpacetimeCorr(), and LeapfrogStep().

### 3.58.1.3 SignR

```
#define SignR(  x, \\  y ) \mbox{ (((y) >= 0) ? (x) : (- (x)))}
```

Definition at line 15 of file global.h.

Referenced by EvalRdf(), and LeapfrogStep().

#### 3.58.1.4 Sqr

```
#define Sqr( x ) ((x) * (x))
```

Definition at line 14 of file global.h.

Referenced by AccumProps(), BrownianStep(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), EvalProps(), EvalRdf(), EvalVrms(), and LeapfrogStep().

# 3.58.2 Typedef Documentation

## 3.58.2.1 real

```
typedef double real
```

Definition at line 11 of file global.h.

## 3.58.3 Variable Documentation

# 3.58.3.1 atom1

```
int* atom1 [extern]
```

Referenced by Close(), ComputeBondForce(), DumpBonds(), DumpRestart(), and Init().

### 3.58.3.2 atom2

```
int * atom2
```

Definition at line 34 of file global.h.

Referenced by Close(), ComputeBondForce(), DumpBonds(), DumpRestart(), and Init().

#### 3.58.3.3 atomID

```
int* atomID [extern]
```

Referenced by Close(), DumpRestart(), DumpState(), Init(), and Trajectory().

#### 3.58.3.4 atomIDInterface

```
int* atomIDInterface [extern]
```

Referenced by Close(), ComputePairForce(), and Init().

#### 3.58.3.5 atomMass

```
double* atomMass [extern]
```

Referenced by Close(), ComputePairForce(), EvalCom(), and Init().

#### 3.58.3.6 atomRadius

```
double* atomRadius [extern]
```

Referenced by ApplyBoundaryCond(), Close(), ComputeForcesCells(), ComputePairForce(), DumpRestart(), DumpState(), Init(), and Trajectory().

# 3.58.3.7 atomType

```
int* atomType [extern]
```

Referenced by ApplyDrivingForce(), Close(), DumpRestart(), DumpState(), Init(), Trajectory(), and VelocityVerletStep().

## 3.58.3.8 ax

double \* ax

Definition at line 17 of file global.h.

Referenced by ApplyDrivingForce(), ApplyForce(), ApplyViscous(), BrownianStep(), Close(), ComputeBondForce(), ComputeForceScells(), ComputePairForce(), DumpState(), EvalProps(), EvalVacf(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

#### 3.58.3.9 ay

double \* ay

Definition at line 17 of file global.h.

Referenced by ApplyDrivingForce(), ApplyForce(), ApplyViscous(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DumpState(), EvalProps(), EvalVacf(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

#### 3.58.3.10 bond

```
char bond[256] [extern]
```

Referenced by main().

## 3.58.3.11 BondEnergy

```
double* BondEnergy [extern]
```

Referenced by ComputeBondForce(), and Init().

## 3.58.3.12 BondEnergyPerAtom

```
double BondEnergyPerAtom
```

Definition at line 38 of file global.h.

Referenced by EvalProps(), and PrintSummary().

#### 3.58.3.13 BondID

```
int* BondID [extern]
```

Referenced by Close(), DumpBonds(), DumpRestart(), and Init().

#### 3.58.3.14 BondLength

```
double * BondLength
```

Definition at line 37 of file global.h.

Referenced by ComputeBondForce(), DumpBonds(), and Init().

# 3.58.3.15 BondType

```
int * BondType
```

Definition at line 35 of file global.h.

Referenced by Close(), DumpBonds(), DumpRestart(), and Init().

## 3.58.3.16 cellList

```
int* cellList [extern]
```

Referenced by Close(), ComputeForcesCells(), and Init().

## 3.58.3.17 cells

```
int cells[2+1]
```

Definition at line 78 of file global.h.

Referenced by ComputeForcesCells(), and Init().

# 3.58.3.18 cfOrg

```
double** cfOrg [extern]
```

Referenced by AllocArrays(), Close(), and EvalSpacetimeCorr().

## 3.58.3.19 cfVal

```
double * cfVal
```

Definition at line 89 of file global.h.

Referenced by AllocArrays(), Close(), and EvalSpacetimeCorr().

#### 3.58.3.20 com

```
char com[256] [extern]
```

Referenced by main().

# 3.58.3.21 ComX

```
double ComX [extern]
```

Referenced by EvalCom(), and PrintCom().

# 3.58.3.22 ComX0

double ComX0

Definition at line 44 of file global.h.

Referenced by EvalCom().

## 3.58.3.23 ComXRatio

double ComXRatio

Definition at line 44 of file global.h.

Referenced by EvalCom().

## 3.58.3.24 ComY

double ComY

Definition at line 44 of file global.h.

Referenced by EvalCom(), and PrintCom().

#### 3.58.3.25 ComY0

double ComY0

Definition at line 44 of file global.h.

Referenced by EvalCom().

#### 3.58.3.26 ComYRatio

double ComYRatio

Definition at line 44 of file global.h.

Referenced by EvalCom().

# 3.58.3.27 countAcfAv

int countAcfAv

Definition at line 95 of file global.h.

Referenced by AccumVacf(), and ZeroVacf().

## 3.58.3.28 countCorrAv

int countCorrAv

Definition at line 90 of file global.h.

Referenced by EvalSpacetimeCorr(), and SetupJob().

#### 3.58.3.29 countRdf

```
int countRdf [extern]
```

Referenced by EvalRdf(), and SetupJob().

#### 3.58.3.30 DampFlag

```
int DampFlag [extern]
```

Referenced by ComputeBondForce(), ComputePairForce(), and Init().

#### 3.58.3.31 deltaT

double deltaT

Definition at line 20 of file global.h.

Referenced by AccumVacf(), BrownianStep(), ComputeBondForce(), ComputePairForce(), EvalProps(), EvalSpacetimeCorr(), EvalVacf(), Init(), LeapfrogStep(), main(), PrintVacf(), and VelocityVerletStep().

#### 3.58.3.32 DeltaVXij

```
double DeltaVXij
```

Definition at line 70 of file global.h.

Referenced by ComputeBondForce(), and ComputePairForce().

## 3.58.3.33 DeltaVYij

double DeltaVYij

Definition at line 70 of file global.h.

Referenced by ComputeBondForce(), and ComputePairForce().

### 3.58.3.34 DeltaX

double DeltaX

Definition at line 46 of file global.h.

Referenced by DisplaceAtoms(), and Init().

# 3.58.3.35 DeltaXij

```
double DeltaXij [extern]
```

Referenced by ComputeBondForce(), and ComputePairForce().

### 3.58.3.36 DeltaXijNew

```
double DeltaXijNew [extern]
```

Referenced by ComputeBondForce(), and ComputePairForce().

#### 3.58.3.37 DeltaXijOld

```
double* DeltaXijOld [extern]
```

Referenced by Close(), ComputeBondForce(), and Init().

## 3.58.3.38 DeltaXijOldPair

```
double** DeltaXijOldPair [extern]
```

Referenced by Close(), ComputePairForce(), and Init().

#### 3.58.3.39 DeltaY

```
double DeltaY [extern]
```

Referenced by DisplaceAtoms(), and Init().

# 3.58.3.40 DeltaYij

```
double DeltaYij
```

Definition at line 70 of file global.h.

Referenced by ComputeBondForce(), and ComputePairForce().

## 3.58.3.41 DeltaYijNew

```
double DeltaYijNew
```

Definition at line 69 of file global.h.

Referenced by ComputeBondForce(), and ComputePairForce().

# 3.58.3.42 DeltaYijOld

```
double * DeltaYijOld
```

Definition at line 68 of file global.h.

Referenced by Close(), ComputeBondForce(), and Init().

## 3.58.3.43 DeltaYijOldPair

```
double ** DeltaYijOldPair
```

Definition at line 71 of file global.h.

Referenced by Close(), ComputePairForce(), and Init().

## 3.58.3.44 density

```
double density
```

Definition at line 21 of file global.h.

Referenced by EvalProps(), and Init().

#### 3.58.3.45 discDragx

```
double* discDragx [extern]
```

Referenced by ComputeForcesCells(), ComputePairForce(), DumpPairs(), and Init().

# 3.58.3.46 discDragy

```
double * discDragy
```

Definition at line 40 of file global.h.

Referenced by ComputeForcesCells(), ComputePairForce(), DumpPairs(), and Init().

## 3.58.3.47 dnsty

```
char dnsty[256] [extern]
```

## 3.58.3.48 dump

```
char dump[256] [extern]
```

# 3.58.3.49 dumpPairFlag

```
int dumpPairFlag [extern]
```

## 3.58.3.50 fax

```
double* fax [extern]
```

Referenced by Close(), and Init().

### 3.58.3.51 fay

```
double * fay
```

Definition at line 80 of file global.h.

Referenced by Close(), and Init().

## 3.58.3.52 fpbond

```
FILE* fpbond [extern]
```

Referenced by DumpBonds(), and main().

#### 3.58.3.53 fpcom

```
FILE* fpcom [extern]
```

Referenced by Init(), main(), and PrintCom().

## 3.58.3.54 fpdnsty

```
FILE* fpdnsty [extern]
```

Referenced by EvalSpacetimeCorr().

# 3.58.3.55 fpdump

```
FILE* fpdump [extern]
```

### 3.58.3.56 fpmomentum

```
FILE* fpmomentum [extern]
```

Referenced by PrintMomentum().

# 3.58.3.57 fppair

```
FILE* fppair [extern]
```

Referenced by DumpPairs(), and main().

## 3.58.3.58 fprdf

```
FILE* fprdf [extern]
```

Referenced by EvalRdf().

## 3.58.3.59 fpresult

```
FILE* fpresult [extern]
```

Referenced by ApplyBoundaryCond(), HaltConditionCheck(), Init(), main(), and PrintSummary().

## 3.58.3.60 fpstress

```
FILE* fpstress [extern]
```

Referenced by PrintStress().

# 3.58.3.61 fpvisc

```
FILE* fpvisc [extern]
```

Referenced by PrintVacf().

#### 3.58.3.62 fpvrms

```
FILE* fpvrms [extern]
```

Referenced by Init(), main(), and PrintVrms().

# 3.58.3.63 fpxyz

```
FILE* fpxyz [extern]
```

Referenced by main(), and Trajectory().

## 3.58.3.64 freezeAtomType

```
int freezeAtomType [extern]
```

Referenced by Init(), and VelocityVerletStep().

# 3.58.3.65 frfAtom

double frfAtom

Definition at line 80 of file global.h.

## 3.58.3.66 fuSum

double fuSum

Definition at line 80 of file global.h.

## 3.58.3.67 fvirSum

```
double fvirSum
```

Definition at line 80 of file global.h.

#### 3.58.3.68 fx

```
double fx [extern]
```

Referenced by ApplyForce().

#### 3.58.3.69 fxByfy

```
double fxByfy
```

Definition at line 52 of file global.h.

Referenced by ApplyForce(), and Init().

## 3.58.3.70 fy

```
double fy
```

Definition at line 52 of file global.h.

Referenced by ApplyForce().

## 3.58.3.71 FyBylx

```
double FyBylx
```

Definition at line 52 of file global.h.

Referenced by ApplyForce(), and Init().

## 3.58.3.72 gamman

```
double gamman [extern]
```

Referenced by ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), and Init().

## 3.58.3.73 HaltCondition

```
double HaltCondition [extern]
```

Referenced by HaltConditionCheck(), and Init().

## 3.58.3.74 histRdf

```
double* histRdf [extern]
```

Referenced by AllocArrays(), and EvalRdf().

## 3.58.3.75 ImageX

```
int* ImageX [extern]
```

Referenced by Close(), EvalUnwrap(), Init(), and VelocityVerletStep().

## 3.58.3.76 ImageY

```
int * ImageY
```

Definition at line 47 of file global.h.

Referenced by Close(), EvalUnwrap(), Init(), and VelocityVerletStep().

#### 3.58.3.77 indexAcf

```
double* indexAcf [extern]
```

Referenced by AccumVacf(), AllocArrays(), Close(), EvalVacf(), and InitVacf().

# 3.58.3.78 indexCorr

```
int* indexCorr [extern]
```

Referenced by AllocArrays(), Close(), EvalSpacetimeCorr(), and SetupJob().

#### 3.58.3.79 initUcell

```
int initUcell[2+1] [extern]
```

### 3.58.3.80 isBonded

```
int** isBonded [extern]
```

Referenced by Close(), ComputePairForce(), and Init().

#### 3.58.3.81 kappa

double kappa

Definition at line 21 of file global.h.

Referenced by Init(), and LeapfrogStep().

#### 3.58.3.82 kb

```
double* kb [extern]
```

Referenced by Close(), ComputeBondForce(), DumpRestart(), and Init().

## 3.58.3.83 kinEnergy

double kinEnergy

Definition at line 20 of file global.h.

Referenced by AccumProps(), AccumVacf(), EvalProps(), and PrintSummary().

## 3.58.3.84 Kn

double Kn [extern]

Referenced by ComputePairForce().

#### 3.58.3.85 limitAcfAv

int limitAcfAv

Definition at line 95 of file global.h.

Referenced by AccumVacf(), and Init().

## 3.58.3.86 limitCorrAv

int limitCorrAv

Definition at line 90 of file global.h.

Referenced by EvalSpacetimeCorr(), and Init().

## 3.58.3.87 limitRdf

```
int limitRdf
```

Definition at line 99 of file global.h.

Referenced by EvalRdf(), and Init().

#### 3.58.3.88 master

```
int master
```

Definition at line 79 of file global.h.

#### 3.58.3.89 molID

```
int* molID [extern]
```

Referenced by ApplyForce(), Close(), DisplaceAtoms(), DumpRestart(), DumpState(), EvalCom(), Init(), and Trajectory().

#### 3.58.3.90 momentum

```
char momentum[256] [extern]
```

## 3.58.3.91 moreCycles

int moreCycles

Definition at line 24 of file global.h.

Referenced by main().

## 3.58.3.92 nAtom

int nAtom

Definition at line 24 of file global.h.

Referenced by ApplyBoundaryCond(), ApplyDrivingForce(), ApplyForce(), ApplyLeesEdwardsBoundaryCond(), ApplyShear(), ApplyViscous(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DisplaceAtoms(), DumpRestart(), DumpState(), EvalCom(), EvalProps(), EvalRdf(), EvalSpacetimeCorr(), EvalUnwrap(), EvalVacf(), EvalVrms(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

#### 3.58.3.93 nAtomBlock

int nAtomBlock

Definition at line 49 of file global.h.

Referenced by ApplyForce(), and Init().

## 3.58.3.94 nAtomInterface

```
int nAtomInterface [extern]
```

Referenced by ComputePairForce(), and Init().

## 3.58.3.95 nAtomType

```
int nAtomType [extern]
```

Referenced by DumpRestart(), and Init().

#### 3.58.3.96 nBond

```
int nBond [extern]
```

Referenced by ComputeBondForce(), DumpBonds(), DumpRestart(), and Init().

## 3.58.3.97 nBondType

int nBondType

Definition at line 33 of file global.h.

Referenced by DumpRestart(), and Init().

## 3.58.3.98 nBuffAcf

int nBuffAcf

Definition at line 95 of file global.h.

Referenced by AccumVacf(), AllocArrays(), Close(), EvalVacf(), Init(), and InitVacf().

### 3.58.3.99 nBuffCorr

int nBuffCorr

Definition at line 90 of file global.h.

 $Referenced \ by \ AllocArrays(), \ Close(), \ EvalSpacetimeCorr(), \ Init(), \ and \ SetupJob().$ 

## 3.58.3.100 nDiscInterface

```
int nDiscInterface
```

Definition at line 49 of file global.h.

Referenced by Init().

#### 3.58.3.101 nFunCorr

```
int nFunCorr
```

Definition at line 90 of file global.h.

Referenced by AllocArrays(), EvalSpacetimeCorr(), Init(), and SetupJob().

## 3.58.3.102 nodeDragx

```
double * nodeDragx
```

Definition at line 40 of file global.h.

Referenced by ComputeBondForce(), DumpBonds(), and Init().

## 3.58.3.103 nodeDragy

```
double * nodeDragy
```

Definition at line 40 of file global.h.

Referenced by ComputeBondForce(), DumpBonds(), and Init().

## 3.58.3.104 nPairActive

```
int nPairActive
```

Definition at line 58 of file global.h.

 $Referenced \ by \ Compute Pair Force(), \ and \ Dump Pairs().$ 

#### 3.58.3.105 nPairTotal

```
int nPairTotal [extern]
```

Referenced by ComputePairForce(), and Init().

#### 3.58.3.106 nValAcf

```
int nValAcf [extern]
```

Referenced by AccumVacf(), AllocArrays(), Init(), InitVacf(), PrintVacf(), and ZeroVacf().

#### 3.58.3.107 nValCorr

int nValCorr

Definition at line 90 of file global.h.

Referenced by AllocArrays(), EvalSpacetimeCorr(), Init(), and SetupJob().

## 3.58.3.108 pair

```
char pair[256] [extern]
```

Referenced by main().

## 3.58.3.109 Pairatom1

```
int * Pairatom1
```

Definition at line 59 of file global.h.

Referenced by Close(), ComputePairForce(), DumpPairs(), and Init().

#### 3.58.3.110 Pairatom2

```
int * Pairatom2
```

Definition at line 59 of file global.h.

Referenced by Close(), ComputePairForce(), DumpPairs(), and Init().

## 3.58.3.111 PairID

```
int* PairID [extern]
```

Referenced by Close(), ComputePairForce(), DumpPairs(), and Init().

## 3.58.3.112 PairXij

```
double* PairXij [extern]
```

Referenced by Close(), ComputePairForce(), DumpPairs(), and Init().

## 3.58.3.113 PairYij

```
double * PairYij
```

Definition at line 60 of file global.h.

Referenced by Close(), ComputePairForce(), DumpPairs(), and Init().

#### 3.58.3.114 potEnergy

```
double potEnergy
```

Definition at line 20 of file global.h.

Referenced by AccumProps(), EvalProps(), and PrintSummary().

## 3.58.3.115 prefix

```
char* prefix [extern]
```

Definition at line 13 of file main.c.

Referenced by DumpRestart(), DumpState(), and main().

## 3.58.3.116 pressure

```
double pressure
```

Definition at line 21 of file global.h.

Referenced by AccumProps(), EvalProps(), PrintStress(), and PrintSummary().

## 3.58.3.117 RadiusIJ

```
double RadiusIJ [extern]
```

Referenced by ComputeForcesCells(), and ComputePairForce().

## 3.58.3.118 RadiuslJInv

double RadiusIJInv

Definition at line 26 of file global.h.

Referenced by ComputeForcesCells(), and ComputePairForce().

#### 3.58.3.119 rangeRdf

double rangeRdf

Definition at line 98 of file global.h.

Referenced by EvalRdf(), and Init().

#### 3.58.3.120 rank

```
int rank [extern]
```

Referenced by ComputeForcesCells().

#### 3.58.3.121 rCut

double rCut

Definition at line 21 of file global.h.

Referenced by Init(), and LeapfrogStep().

#### 3.58.3.122 rdf

```
char rdf[256] [extern]
```

### 3.58.3.123 region

```
double region[2+1] [extern]
```

Referenced by AccumVacf(), ApplyBoundaryCond(), ApplyLeesEdwardsBoundaryCond(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DumpRestart(), EvalRdf(), EvalSpacetimeCorr(), EvalUnwrap(), Init(), LeapfrogStep(), and VelocityVerletStep().

## 3.58.3.124 regionH

```
double regionH[2+1]
```

Definition at line 20 of file global.h.

Referenced by ApplyBoundaryCond(), ApplyForce(), ApplyLeesEdwardsBoundaryCond(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DumpBonds(), DumpPairs(), DumpState(), EvalRdf(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

#### 3.58.3.125 result

```
char result[250] [extern]
```

Referenced by main().

#### 3.58.3.126 rfAtom

```
double rfAtom [extern]
```

Referenced by ComputeForcesCells(), and EvalVacf().

#### 3.58.3.127 ro

```
double * ro
```

Definition at line 36 of file global.h.

Referenced by Close(), ComputeBondForce(), DumpBonds(), DumpRestart(), and Init().

#### 3.58.3.128 rx

```
double* rx [extern]
```

Referenced by ApplyBoundaryCond(), ApplyLeesEdwardsBoundaryCond(), ApplyShear(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DisplaceAtoms(), DumpRestart(), DumpState(), EvalRdf(), EvalSpacetimeCorr(), EvalUnwrap(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

## 3.58.3.129 rxUnwrap

```
double* rxUnwrap [extern]
```

Referenced by Close(), EvalCom(), EvalUnwrap(), and Init().

#### 3.58.3.130 ry

```
double * ry
```

Definition at line 17 of file global.h.

Referenced by ApplyBoundaryCond(), ApplyLeesEdwardsBoundaryCond(), ApplyShear(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DisplaceAtoms(), DumpRestart(), DumpState(), EvalRdf(), EvalUnwrap(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

### 3.58.3.131 ryUnwrap

```
double * ryUnwrap
```

Definition at line 48 of file global.h.

Referenced by Close(), EvalCom(), EvalUnwrap(), and Init().

## 3.58.3.132 shearDisplacement

```
double shearDisplacement [extern]
```

Referenced by ApplyLeesEdwardsBoundaryCond(), ComputeBondForce(), ComputePairForce(), and Init().

## 3.58.3.133 shearVelocity

```
double shearVelocity
```

Definition at line 42 of file global.h.

Referenced by Init().

#### 3.58.3.134 size

int size

Definition at line 79 of file global.h.

Referenced by ComputeForcesCells().

## 3.58.3.135 sizeHistRdf

int sizeHistRdf

Definition at line 99 of file global.h.

Referenced by AllocArrays(), EvalRdf(), and Init().

## 3.58.3.136 sKinEnergy

double sKinEnergy

Definition at line 21 of file global.h.

Referenced by AccumProps().

## 3.58.3.137 solver

```
char solver[128] [extern]
```

Referenced by EvalProps(), and Init().

## 3.58.3.138 spacetimeCorr

```
double ** spacetimeCorr
```

Definition at line 89 of file global.h.

Referenced by AllocArrays(), Close(), and EvalSpacetimeCorr().

#### 3.58.3.139 spacetimeCorrAv

```
double * spacetimeCorrAv
```

Definition at line 89 of file global.h.

Referenced by AllocArrays(), Close(), EvalSpacetimeCorr(), and SetupJob().

## 3.58.3.140 speed

```
double* speed [extern]
```

Referenced by Close(), and Init().

#### 3.58.3.141 sPotEnergy

double sPotEnergy

Definition at line 21 of file global.h.

Referenced by AccumProps().

#### 3.58.3.142 sPressure

double sPressure

Definition at line 22 of file global.h.

Referenced by AccumProps().

## 3.58.3.143 SqrRadiusIJ

double SqrRadiusIJ

Definition at line 26 of file global.h.

Referenced by ComputeForcesCells(), and ComputePairForce().

## 3.58.3.144 ssKinEnergy

double ssKinEnergy

Definition at line 22 of file global.h.

Referenced by AccumProps().

## 3.58.3.145 ssPotEnergy

double ssPotEnergy

Definition at line 22 of file global.h.

Referenced by AccumProps().

#### 3.58.3.146 ssPressure

double ssPressure

Definition at line 22 of file global.h.

Referenced by AccumProps().

## 3.58.3.147 ssTotEnergy

double ssTotEnergy

Definition at line 22 of file global.h.

Referenced by AccumProps().

## 3.58.3.148 stepAcf

int stepAcf

Definition at line 95 of file global.h.

Referenced by AccumVacf(), Init(), and PrintVacf().

## 3.58.3.149 stepAvg

int stepAvg

Definition at line 24 of file global.h.

Referenced by AccumProps(), Init(), and main().

#### 3.58.3.150 stepCorr

```
int stepCorr
```

Definition at line 90 of file global.h.

Referenced by EvalSpacetimeCorr(), and Init().

## 3.58.3.151 stepCount

```
int stepCount
```

Definition at line 24 of file global.h.

Referenced by BrownianStep(), ComputeBondForce(), ComputePairForce(), HaltConditionCheck(), LeapfrogStep(), main(), and SetupJob().

## 3.58.3.152 stepDump

```
int stepDump
```

Definition at line 24 of file global.h.

Referenced by Init(), and main().

#### 3.58.3.153 stepEquil

```
int stepEquil
```

Definition at line 24 of file global.h.

Referenced by BrownianStep(), Init(), and LeapfrogStep().

## 3.58.3.154 stepLimit

```
int stepLimit
```

Definition at line 24 of file global.h.

Referenced by Init(), and main().

#### 3.58.3.155 stepRdf

int stepRdf

Definition at line 99 of file global.h.

Referenced by Init().

## 3.58.3.156 stepTraj

```
int stepTraj
```

Definition at line 24 of file global.h.

Referenced by Init(), and main().

## 3.58.3.157 sTotEnergy

```
double sTotEnergy
```

Definition at line 22 of file global.h.

Referenced by AccumProps().

## 3.58.3.158 strain

```
double strain [extern]
```

Referenced by ApplyShear(), and Init().

#### 3.58.3.159 strainRate

double strainRate

Definition at line 41 of file global.h.

Referenced by Init().

## 3.58.3.160 strech

```
double strech [extern]
```

 $Referenced \ by \ Compute Bond Force(), \ and \ Compute Pair Force().$ 

## 3.58.3.161 stress

```
char stress[256] [extern]
```

## 3.58.3.162 svirSum

double svirSum

Definition at line 21 of file global.h.

Referenced by AccumProps().

## 3.58.3.163 thermo

```
char thermo [extern]
```

Referenced by Init(), and LeapfrogStep().

#### 3.58.3.164 timeNow

double timeNow

Definition at line 20 of file global.h.

Referenced by DumpBonds(), DumpPairs(), DumpRestart(), DumpState(), EvalCom(), EvalRdf(), Init(), main(), PrintCom(), PrintMomentum(), PrintStress(), PrintSummary(), PrintVrms(), and Trajectory().

## 3.58.3.165 TotalBondEnergy

```
double TotalBondEnergy [extern]
```

Referenced by ComputeBondForce(), and EvalProps().

#### 3.58.3.166 TotalMass

double TotalMass

Definition at line 31 of file global.h.

Referenced by EvalCom().

## 3.58.3.167 totEnergy

double totEnergy

Definition at line 20 of file global.h.

Referenced by AccumProps(), EvalProps(), and PrintSummary().

#### 3.58.3.168 uSum

double uSum

Definition at line 21 of file global.h.

Referenced by ComputeForcesCells().

#### 3.58.3.169 uSumPair

```
double uSumPair [extern]
```

Referenced by ComputePairForce(), and EvalProps().

#### 3.58.3.170 uSumPairPerAtom

double uSumPairPerAtom

Definition at line 83 of file global.h.

Referenced by EvalProps(), and PrintSummary().

#### 3.58.3.171 virSum

double virSum

Definition at line 21 of file global.h.

Referenced by AccumProps(), ComputeForcesCells(), EvalProps(), and PrintSummary().

## 3.58.3.172 virSumBond

double virSumBond [extern]

Referenced by ComputeBondForce(), and EvalProps().

#### 3.58.3.173 virSumBondxx

double virSumBondxx

Definition at line 84 of file global.h.

Referenced by ComputeBondForce(), and EvalProps().

## 3.58.3.174 virSumBondxy

double virSumBondxy

Definition at line 84 of file global.h.

Referenced by ComputeBondForce(), and EvalProps().

## 3.58.3.175 virSumBondyy

```
double virSumBondyy
```

Definition at line 84 of file global.h.

Referenced by ComputeBondForce(), and EvalProps().

#### 3.58.3.176 virSumPair

```
double virSumPair
```

Definition at line 83 of file global.h.

Referenced by ComputePairForce(), and EvalProps().

#### 3.58.3.177 virSumPairxx

```
double virSumPairxx
```

Definition at line 83 of file global.h.

Referenced by ComputePairForce(), and EvalProps().

## 3.58.3.178 virSumPairxy

```
double virSumPairxy
```

Definition at line 83 of file global.h.

Referenced by ComputePairForce(), and EvalProps().

## 3.58.3.179 virSumPairyy

```
double virSumPairyy
```

Definition at line 83 of file global.h.

Referenced by ComputePairForce(), and EvalProps().

#### 3.58.3.180 virSumxx

```
double virSumxx [extern]
```

Referenced by EvalProps(), and PrintStress().

#### 3.58.3.181 virSumxy

double virSumxy

Definition at line 85 of file global.h.

Referenced by EvalProps(), and PrintStress().

## 3.58.3.182 virSumyy

double virSumyy

Definition at line 85 of file global.h.

Referenced by EvalProps(), and PrintStress().

#### 3.58.3.183 visc

```
char visc[256] [extern]
```

#### 3.58.3.184 viscAcf

double \*\* viscAcf

Definition at line 94 of file global.h.

Referenced by AccumVacf(), AllocArrays(), Close(), and EvalVacf().

#### 3.58.3.185 viscAcfAv

```
double * viscAcfAv
```

Definition at line 94 of file global.h.

Referenced by AccumVacf(), AllocArrays(), Close(), PrintVacf(), and ZeroVacf().

#### 3.58.3.186 viscAcfInt

double viscAcfInt

Definition at line 94 of file global.h.

Referenced by AccumVacf(), and PrintVacf().

## 3.58.3.187 viscAcfOrg

```
double * viscAcfOrg
```

Definition at line 94 of file global.h.

Referenced by AllocArrays(), Close(), and EvalVacf().

#### 3.58.3.188 VMeanSqr

```
double VMeanSqr
```

Definition at line 43 of file global.h.

Referenced by EvalVrms().

#### 3.58.3.189 vrms

```
char vrms[256] [extern]
```

Referenced by main().

## 3.58.3.190 VRootMeanSqr

```
double VRootMeanSqr
```

Definition at line 43 of file global.h.

Referenced by EvalVrms(), main(), and PrintVrms().

## 3.58.3.191 VSqr

```
double VSqr [extern]
```

Referenced by EvalVrms().

## 3.58.3.192 vSum

double vSum

Definition at line 21 of file global.h.

Referenced by EvalProps(), and PrintSummary().

#### 3.58.3.193 vSumX

double vSumX

Definition at line 21 of file global.h.

Referenced by EvalProps(), and PrintMomentum().

#### 3.58.3.194 vSumY

double vSumY

Definition at line 21 of file global.h.

Referenced by EvalProps(), and PrintMomentum().

#### 3.58.3.195 vvSum

double vvSum

Definition at line 21 of file global.h.

Referenced by EvalProps(), and LeapfrogStep().

### 3.58.3.196 vx

double \* vx

Definition at line 17 of file global.h.

Referenced by ApplyBoundaryCond(), ApplyDrivingForce(), ApplyViscous(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DumpRestart(), DumpState(), EvalProps(), EvalVacf(), EvalVrms(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

#### 3.58.3.197 vy

double \* vy

Definition at line 17 of file global.h.

Referenced by ApplyBoundaryCond(), ApplyDrivingForce(), ApplyViscous(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DumpRestart(), DumpState(), EvalProps(), EvalVacf(), EvalVrms(), Init(), LeapfrogStep(), Trajectory(), and VelocityVerletStep().

#### 3.58.3.198 xBoundary

char xBoundary[10] [extern]

Referenced by ApplyBoundaryCond(), and Init().

#### 3.58.3.199 xyz

```
char xyz[256] [extern]
```

Referenced by main().

#### 3.58.3.200 yBoundary

```
char yBoundary[10]
```

Definition at line 64 of file global.h.

Referenced by ApplyBoundaryCond(), and Init().

# 3.59 global.h

#### Go to the documentation of this file.

```
00001 #ifndef GLOBAL_H
00002 #define GLOBAL H
00003 #include <stdio.h> // Required for FILE*
00004
00005 #ifdef DEFINE_GLOBALS
00006
          #define EXTERN
00007 #else
8,000
         #define EXTERN extern
00009 #endif
00010
00011 typedef double real;
00012
00013 #define NDIM 2
00014 \#define Sqr(x) ((x) * (x))
00015 #define SignR(x, y) (((y) \geq= 0) ? (x) : (- (x)))
00016
00017 EXTERN double *rx, *ry, *vx, *vy, *ax, *ay;
00018 EXTERN double *speed;
00019
00020 EXTERN double region[NDIM+1], regionH[NDIM+1], deltaT, timeNow, potEnergy, kinEnergy, totEnergy, 00021 density, pressure, rCut, kappa, uSum, virSum, svirSum, vSum, vSumX, vSumY, vvSum, sPotEnergy,
      sKinEnergy,
00022 sTotEnergy, sPressure, ssPotEnergy, ssKinEnergy, ssTotEnergy, ssPressure;
00023
00024 EXTERN int
                     initUcell[NDIM+1], moreCycles, nAtom, stepAvg, stepCount, stepEquil, stepLimit,
      stepTraj, stepDump;
00025
00026 EXTERN double RadiusIJ, SqrRadiusIJ, RadiusIJInv;
00027 EXTERN int
                     nAtomType;
00028 EXTERN int
00029 EXTERN int
                      *atomID;
00030 EXTERN double *atomRadius;
00031 EXTERN double *atomMass, TotalMass;
00032
00033 EXTERN int
                      nBond, nBondType;
00034 EXTERN int
                      *atom1, *atom2;
00035 EXTERN int
                      *BondID, *BondType;
00036 EXTERN double
                      *kb, *ro;
                      *BondEnergy, *BondLength;
00037 EXTERN double
00038 EXTERN double
                      TotalBondEnergy, BondEnergyPerAtom;
00039 EXTERN double gamman;
                     *discDragx, *discDragy, *nodeDragx, *nodeDragy;
00040 EXTERN double
00041 EXTERN double
                     strain, strainRate;
00042 EXTERN double
                      shearDisplacement, shearVelocity;
00043 EXTERN double
                      VSqr, VMeanSqr, VRootMeanSqr;
                      ComX, ComY, ComX0, ComY0, ComXRatio, ComYRatio; HaltCondition;
00044 EXTERN double
00045 EXTERN double
00046 EXTERN double DeltaY, DeltaX;
00047 EXTERN int
                      *ImageX, *ImageY;
00048 EXTERN double
                      *rxUnwrap, *ryUnwrap;
00049 EXTERN int
                      nAtomInterface, nDiscInterface, nAtomBlock;
00050 EXTERN int.
                      *atomIDInterface;
00051 EXTERN double Kn;
00052 EXTERN double fx, fy, FyBylx, fxByfy;
00053 EXTERN int
                      DampFlag;
```

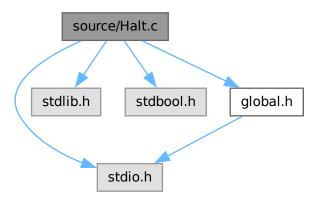
3.59 global.h 115

```
00054 EXTERN double strech;
00055
00056 //For dumping the pair interaction data
00057 EXTERN int
                      dumpPairFlag;
00061
00062
00063 EXTERN char
                    solver[128];
00064 EXTERN char
                    xBoundary[10], yBoundary[10];
00065 EXTERN char
                      thermo:
00066
00067 //For damping as in PRL, 130, 178203 (2023)
00068 EXTERN double *DeltaXijOld, *DeltaYijOld;
00069 EXTERN double DeltaXijNew, DeltaYijNew;
00070 EXTERN double DeltaXij, DeltaYij, DeltaVXij, DeltaVYij;
00071 EXTERN double **DeltaXijOldPair, **DeltaYijOldPair;
00073 //For molecule-ID as per LAMMPS, helpful!
00074 EXTERN int *molID;
00075 EXTERN int
                     **isBonded:
00076
00077 //Following three for MPI only
                   *cellList, cells[NDIM+1];
00078 EXTERN int
00079 EXTERN int
                      rank, size, master;
00080 EXTERN double *fax, *fay, fuSum, fvirSum, frfAtom;
00081
00082 //For thermodynamic properties
00083 EXTERN double uSumPair, uSumPairPerAtom, virSumPair, virSumPairxx, virSumPairxy;
00084 EXTERN double virSumBond, virSumBondxx, virSumBondyy, virSumBondxy;
00085 EXTERN double virSumxx, virSumyy, virSumxy;
00086 EXTERN int
                      freezeAtomType;
00087
00088 // Spacetime Correlations
00089 EXTERN double **cfOrg, **spacetimeCorr, *cfVal, *spacetimeCorrAv;
00090 EXTERN int *indexCorr, countCorrAv, limitCorrAv, nBuffCorr, nFunCorr, nValCorr, stepCorr;
00092 // Viscosity
00093 EXTERN double rfAtom, frfAtom;
00094 EXTERN double *indexAcf, **viscAcf, *viscAcfOrg, *viscAcfAv, viscAcfInt;
                      nValAcf, nBuffAcf, stepAcf, countAcfAv, limitAcfAv;
00095 EXTERN int.
00096
00097 // Radial distribution function
00098 EXTERN double *histRdf, rangeRdf;
00099 EXTERN int countRdf, limitRdf, sizeHistRdf, stepRdf;
00100
00101
00102 // Output files prefixes
00103 EXTERN char
                        *prefix:
00104
00105 EXTERN char
                      result[250];
00106 EXTERN FILE
                      *fpresult;
00107
00108 EXTERN char
                      xyz[256];
00109 EXTERN FILE
                      *fpxyz;
00111 EXTERN char
                      bond[256];
00112 EXTERN FILE
                      *fpbond;
00113
00114
00115 EXTERN char
                      dump[256];
00116 EXTERN FILE
                      *fpdump;
00117
00118 EXTERN char
                      dnsty[256];
00119 EXTERN FILE
                      *fpdnsty;
00120
00121 EXTERN char
                      visc[256];
00122 EXTERN FILE
                      *fpvisc:
00123
00124 EXTERN char
                      rdf[256];
00125 EXTERN FILE
                      *fprdf;
00126
00127 EXTERN char
                      vrms[2561;
00128 EXTERN FILE
                      *fpvrms:
00129
00130 EXTERN char
                      stress[256];
00131 EXTERN FILE
                      *fpstress;
00132
00133 EXTERN char
                      momentum[256]:
00134 EXTERN FILE
                      *fpmomentum;
00135
00136 EXTERN char
                      com[256];
00137 EXTERN FILE
                      *fpcom;
00138
00139 EXTERN char
                      pair[256];
00140 EXTERN FILE
                      *fppair;
```

```
00141
00142 #endif // GLOBALEXTERN_H
```

## 3.60 source/Halt.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "global.h"
Include dependency graph for Halt.c:
```



#### **Functions**

00034 }

• bool HaltConditionCheck (double value, int stepCount)

## 3.60.1 Function Documentation

## 3.60.1.1 HaltConditionCheck()

bool HaltConditionCheck (

References fpresult, HaltCondition, and stepCount.

00033 return false; // Halt condition not met

3.61 Halt.c 117

Referenced by main().

Here is the caller graph for this function:



## 3.61 Halt.c

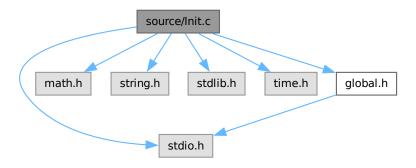
#### Go to the documentation of this file.

```
00002
        * This file is part of Lamina.
00003
         \star Lamina is free software: you can redistribute it and/or modify \star it under the terms of the GNU General Public License as published by
00004
00005
00006
        * the Free Software Foundation, either version 3 of the License, or
        * (at your option) any later version.
80000
00009 \,\,\star\,\, Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details.
00014 \,\star\, You should have received a copy of the GNU General Public License
00015
        * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <stdbool.h>
00025 #include "global.h"
00027 bool HaltConditionCheck(double value, int stepCount) {
00028
          if (value <= HaltCondition && value != 0) {</pre>
00029
         fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.10f\n", stepCount, value); return true; // Signal that the halt condition is met
00030
00031
00033 return false; // Halt condition not met
00034 }
00035
```

## 3.62 source/Init.c File Reference

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
```

#include "global.h"
Include dependency graph for Init.c:



#### **Functions**

• void Init ()

## 3.62.1 Function Documentation

#### 3.62.1.1 Init()

```
void Init ( )
```

#### Definition at line 29 of file Init.c.

```
00029
             char dummy[128];
00030
00031
             char inputConfig[128];
00032
             FILE *fp;
00033
             fp = fopen("input-data", "r");
             fscanf(fp, "%s %s", dummy, inputConfig);
fscanf(fp, "%s %s", dummy, solver);
00034
00035
             fscanf(fp, "%s %s", dummy, solver);
fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
fscanf(fp, "%s %d", dummy, &DampFlag);
fscanf(fp, "%s %d", dummy, &freezeAtomType);
fscanf(fp, "%s %lf", dummy, &rCut);
00036
00037
00038
00039
             fscanf(fp, "%s %lf", dummy, &gamman);
fscanf(fp, "%s %lf", dummy, &kappa);
fscanf(fp, "%s %lf", dummy, &deltaT);
fscanf(fp, "%s %lf", dummy, &strain);
fscanf(fp, "%s %lf", dummy, &fyBylx);
00040
00041
00042
00043
00044
             fscanf(fp, "%s %lf", dummy, &fxByfy);
fscanf(fp, "%s %lf", dummy, &DeltaY);
fscanf(fp, "%s %lf", dummy, &DeltaX);
00045
00046
00047
             fscanf(fp, "%s %lf", dummy, &HaltCondition);
00048
             fscanf(fp, "%s %d", dummy, &stepAvg);
00049
             fscanf(fp, "%s %d", dummy, &stepEquil);
fscanf(fp, "%s %d", dummy, &stepLimit);
00050
00051
             fscanf(fp, "%s %d", fscanf(fp, "%s %d", fscanf(fp, "%s %d",
00052
                                                   dummy, &stepDump);
00053
                                                   dummy, &stepTraj);
00054
                                                   dummy, &limitCorrAv);
             fscanf(fp, "%s %d",
00055
                                                   dummy, &nBuffCorr);
             fscanf(fp, "%s %d",
00056
                                                   dummy, &nFunCorr);
             fscanf(fp, "%s %d", fscanf(fp, "%s %d",
00057
                                                   dummy, &nValCorr);
00058
                                                   dummy, &stepCorr);
            fscanf(fp, "%s %d", dummy, &limitAcfAv
fscanf(fp, "%s %d", dummy, &limitAcfAcf);
fscanf(fp, "%s %d", dummy, &nBuffAcf);
fscanf(fp, "%s %d", dummy, &nValAcf);
fscanf(fp, "%s %d", dummy, &stepAcf);
fscanf(fp, "%s %lf", dummy, &rangeRdf);
00059
                                                   dummy, &limitAcfAv);
00060
                                                   dummy, &nBuffAcf);
00061
00062
00063
             fscanf(fp, "%s %d", dummy, &limitRdf);
```

```
fscanf(fp, "%s %d", dummy, &sizeHistRdf);
fscanf(fp, "%s %d", dummy, &stepRdf);
00066
00067
00068
         fclose(fp);
00069
         FILE *fpSTATE;
if((fpSTATE = fopen(inputConfig, "r")) ==NULL) {
00070
00071
         printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00072
         exit(0);
00073
00074
         fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
fscanf(fpSTATE, "%s %d", dummy, &nAtom);
00075
00076
         fscanf(fpSTATE, "%s %d",
00077
                                       dummy, &nBond);
         fscanf(fpSTATE, "%s %d", dummy, &nAtomType);
00078
         fscanf(fpSTATE, "%s %d", dummy, &nBondType);
fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00079
00080
00081
00082
00083
         region[2] *= 1.5; //Remove this when put on GitHub
00084
00085
         density = nAtom/(region[1]*region[2]);
         cells[1] = region[1] / rCut;
cells[2] = region[2] / rCut;
00086
00087
         cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00088
00089
         regionH[1] = 0.5*region[1];
         regionH[2] = 0.5*region[2];
00090
00091
         //strain information
00092
00093
         strainRate = strain/deltaT;
00094
         shearDisplacement = strain * region[2];
00095
         shearVelocity = strainRate * region[2];
00096
00097
00098
         rx = (double*)malloc((nAtom + 1) * sizeof(double));
         ry = (double*)malloc((nAtom + 1) * sizeof(double));
vx = (double*)malloc((nAtom + 1) * sizeof(double));
00099
00100
         vy = (double*)malloc((nAtom + 1) * sizeof(double));
00101
         ax = (double*)malloc((nAtom + 1) * sizeof(double));
00103
         ay = (double*)malloc((nAtom + 1) * sizeof(double));
         fax = (double*)malloc((nAtom + 1) * sizeof(double));
fay = (double*)malloc((nAtom + 1) * sizeof(double));
00104
00105
         atomID = (int*)malloc((nAtom+1) * sizeof(int));
00106
00107
         atomType = (int*)malloc((nAtom+1) * sizeof(int));
00108
         atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
         atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
00109
00110
         speed = (double*)malloc((nAtom + 1) * sizeof(double));
00111
         atom1 = (int*)malloc((nBond+1)*sizeof(int));
00112
         atom2 = (int*)malloc((nBond+1)*sizeof(int));
00113
         BondID = (int*)malloc((nBond+1)*sizeof(int));
00114
         BondType = (int*)malloc((nBond+1)*sizeof(int));
00115
         kb = (double*)malloc((nBond+1)*sizeof(double));
00116
         ro = (double*)malloc((nBond+1)*sizeof(double));
00117
         BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
         BondLength = (double*) malloc((nBond+1)*sizeof(double));
00118
         discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00119
00120
         nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00121
00122
         nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
         ImageY = (int*)malloc((nAtom+1) * sizeof(int));
ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00123
00124
         rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00125
00126
00127
         DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
         DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
00128
00129
         DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
         DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00130
         for(int n = 0; n <= nAtom; n++) {
  DeltaXijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));</pre>
00131
00132
          DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00133
00134
00135
         molID = (int*)malloc((nAtom+1) * sizeof(int));
00136
         for(n = 1; n <= nAtom; n ++) {
  atomMass[n] = 1.0;</pre>
00137
00138
00139
00140
00141
         fscanf(fpSTATE, "%s\n", dummy);
         for(n = 1; n <= nAtom; n ++)
fscanf(fpSTATE, "%d %d %d %lf %lf %lf %lf %lf %lf \n", &atomID[n], &molID[n], &atomType[n],</pre>
00142
00143
      &atomRadius[n], &rx[n], &ry[n], &vx[n], &vy[n]);
00144
00145
00146
         fscanf(fpSTATE, "%s\n", dummy);
00147
         for (n=1; n<=nBond; n++)</pre>
00148
          &ro[n]);
00149
```

```
fclose(fpSTATE);
00151
00152
           //2D-List of bonded atoms. This is used to remove pair interaction
00153
           //calculation for the bonded atoms
              isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00154
             for (int i = 0; i <= nAtom; i++) {</pre>
00155
               isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00156
                for (int j = 0; j <= nAtom; j++) {
   isBonded[i][j] = 0;</pre>
00157
00158
00159
            }
00160
00161
             for (n = 1; n <= nBond; n++) {
  int i = atom1[n];</pre>
00162
00163
                int j = atom2[n];
00164
                isBonded[i][j] = 1;
isBonded[j][i] = 1; // symmetric
00165
00166
00167 }
00168
00169
           //For thermostate, update in final version
00170 thermo = 'C';
00171
00172
00173 // List the interface atoms
00174 nAtomInterface = 0;
00175 nAtomBlock = 0;
00176
           nDiscInterface = 0;
00177
           double InterfaceWidth, bigDiameter;
00178
           bigDiameter = 2.8;
           InterfaceWidth = 5.0 * bigDiameter;
00179
00180
00181
           for (n = 1; n <= nAtom; n++) {</pre>
00182
            if(fabs(ry[n]) < InterfaceWidth){</pre>
00183
             nAtomInterface++;
00184
             if(molID[n] == 2){
00185
00186
             nAtomBlock++;
00187
00188
             if(atomRadius[n] != 0.0){
00189
             nDiscInterface++;
00190
00191
              atomIDInterface = (int*)malloc((nAtomInterface+1)*sizeof(int));
00192
00193
00194
00195
             m = 1;
00196
             for (n=1; n<=nAtom; n++) {</pre>
00197
              if(fabs(ry[n]) < InterfaceWidth){</pre>
00198
              atomIDInterface[m] = atomID[n];
00199
              m++;
00200
              } }
00201
00202
             nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
             PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00203
00204
00205
             Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
             PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00206
00207
             PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00208
                00209
00210
00211
00212
00213
                fprintf(fpresult, "nBond\t\t\t\d\n", nBond);
fprintf(fpresult, "nAtomBlock\t\t\d\n", nAtomBlock);
fprintf(fpresult, "nAtomInterface\t\d\n", nAtomInterface);
fprintf(fpresult, "nDiscInterface\t\t\d\n", nDiscInterface);
fprintf(fpresult, "gamman\t\t\t\d\0.6g\n", gamman);
fprintf(fpresult, "strain\t\t\d\0.6g\n", strain);
fprintf(fpresult, "strainRate\t\t\d\0.6g\n", strainRate);

00214
00215
00216
00217
00218
00219
                fprintf(fpresult, "strainRate\t\t\80.6g\n", strainRate);
fprintf(fpresult, "FyBylx\t\t\80.6g\n", FyBylx);
fprintf(fpresult, "fxByfy\t\t\80.6g\n", fxByfy);
fprintf(fpresult, "DeltaY\t\t\80.6g\n", DeltaY);
fprintf(fpresult, "DeltaX\t\t\80.6g\n", DeltaX);
fprintf(fpresult, "HaltCondition\t\80.6g\n", HaltCondition);
fprintf(fpresult, "kappa\t\t\$0.6g\n", kappa);
00220
00221
00222
00223
00224
00225
                fprintf(fpresult, "kappa\t\t\t\sq\n", kappa);
fprintf(fpresult, "density\t\t\sq\n", density);
fprintf(fpresult, "rCut\t\t\t\sq\n", rCut);
fprintf(fpresult, "deltaT\t\t\sq\n", rCut);
fprintf(fpresult, "stepEquil\t\t\sq\n", deltaT);
fprintf(fpresult, "stepEquil\t\t\sq\n", stepEquil);
fprintf(fpresult, "stepLimit\t\t\sq\n", stepLimit);
fprintf(fpresult, "region[1]\t\t\sq\n", region[1]);
fprintf(fpresult, "region[2]\t\t\sq\n", region[2]);
fprintf(fpresult, "cells[1]\t\t\sq\n", cells[1]);
fprintf(fpresult, "cells[2]\t\t\sq\n", cells[2]);
fprintf(fpresult, "solver\t\t\sq\n", solver);
fprintf(fpresult, "boundary\t\t\sq\n", xBoundary, yBoundary);
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
```

3.63 Init.c 121

```
fprintf(fpresult, "DampFlag\t\t%d\n", DampFlag);
00238
00239
            00240
00241
       PairEnergyPerAtom BondEnergyPerAtom Press VirialSum\n");
            fprintf(fpvrms, "#timeNow\tVrms \n");
fprintf(fpcom, "#timeNow\tComX\tComY\n");
00242
00243
00244
00245 \slash //Uncomment the following as per your acquirement
            fprintf(fpstress, "strain
fprintf(fpstress, "region[1]
00246
                                                          f^n, strain);
                                                          %lf\n", region[1]);
%lf\n", region[2]);
00247
            fprintf(fpstress, "region[2] %1\n", region[2]);
fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00248
00249
00250
            fprintf(fpmomentum, "#timeNow Px Py\n");
00251 */
00252
           if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
  (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00253
            (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
00255
       allowed.\n", xBoundary, yBoundary);
exit(EXIT_FAILURE); // Exit with failure status
00256
00257
00258
00259 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, BondEnergy, BondID, BondLength, BondType, cellList, cells, DampFlag, deltaT, DeltaX, DeltaXijOld, DeltaXijOldPair, DeltaY, DeltaYijOld, DeltaYijOldPair, density, discDragx, discDragy, fax, fay, fpcom, fpresult, fpvrms, freezeAtomType, fxByfy, FyBylx, gamman, HaltCondition, ImageX, ImageY, isBonded, kappa, kb, limitAcfAv, limitCorrAv, limitRdf, molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nBuffAcf, nBuffCorr, nDiscInterface, nFunCorr, nodeDragx, nodeDragy, nPairTotal, nValAcf, nValCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rangeRdf, rCut, region, regionH, ro, rx, rxUnwrap, ry, ryUnwrap, shearDisplacement, shearVelocity, sizeHistRdf, solver, speed, stepAcf, stepAvg, stepCorr, stepDump, stepEquil, stepLimit, stepRdf, stepTraj, strain, strainRate, thermo, timeNow, vx, vy, xBoundary, and yBoundary.

Referenced by main().

Here is the caller graph for this function:



## 3.63 Init.c

#### Go to the documentation of this file.

```
00001 /*
00002
       * This file is part of Lamina.
00003
00004
       * Lamina is free software: you can redistribute it and/or modify
00005
         it under the terms of the {\tt GNU} General Public License as published by
00006
       * the Free Software Foundation, either version 3 of the License, or
00007
       \star (at your option) any later version.
80000
       * Lamina is distributed in the hope that it will be useful,
       * but WITHOUT ANY WARRANTY; without even the implied warranty of
00010
00011
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
       \star GNU General Public License for more details.
00013
00014
      * You should have received a copy of the GNU General Public License
00015
       * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
```

```
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include<string.h>
00025 #include<stdlib.h>
00026 #include <time.h>
00027 #include"global.h"
00028
00029 void Init(){
00030 char dummy[128];
00031
            char inputConfig[128];
00032
            FILE *fp;
            fp = fopen("input-data","r");
fscanf(fp, "%s %s", dummy, inputConfig);
fscanf(fp, "%s %s", dummy, solver);
00033
00034
           fscanf(fp, "%s %s", dummy, xBoundary, yBoundary);
fscanf(fp, "%s %d", dummy, &DampFlag);
fscanf(fp, "%s %d", dummy, &freezeAtomType);
fscanf(fp, "%s %d", dummy, &rCut);
00036
00037
00038
00039
           fscanf(fp, "%s %lf", dummy, &gamman);
fscanf(fp, "%s %lf", dummy, &kappa);
fscanf(fp, "%s %lf", dummy, &deltaT);
00040
00041
00042
           fscanf(fp, "%s %lf", dummy, &strain);
fscanf(fp, "%s %lf", dummy, &fyBylx);
fscanf(fp, "%s %lf", dummy, &fxByfy);
fscanf(fp, "%s %lf", dummy, &DeltaY);
00043
00044
00045
00046
            fscanf(fp, "%s %lf", dummy, &DeltaX);
00047
           fscanf(fp, "%s %lf", dummy, &DeltaX);
fscanf(fp, "%s %lf", dummy, &HaltCondition);
fscanf(fp, "%s %d", dummy, &stepAvg);
fscanf(fp, "%s %d", dummy, &stepEquil);
fscanf(fp, "%s %d", dummy, &stepLimit);
fscanf(fp, "%s %d", dummy, &stepDump);
00048
00049
00050
00051
00052
            fscanf(fp, "%s %d", dummy, &stepEmpl;
fscanf(fp, "%s %d", dummy, &limitCorrAv);
00053
           fscanf(fp, "%s %d", fscanf(fp, "%s %d", fscanf(fp, "%s %d", fscanf(fp, "%s %d",
00055
                                             dummy, &nBuffCorr);
00056
                                             dummy, &nFunCorr);
00057
                                             dummy, &nValCorr);
00058
                                             dummy, &stepCorr);
            fscanf(fp, "%s %d",
00059
                                             dummy, &limitAcfAv);
           fscanf(fp, "%s %d", dummy, &limitAcfAv);
fscanf(fp, "%s %d", dummy, &nBuffAcf);
fscanf(fp, "%s %d", dummy, &nValAcf);
fscanf(fp, "%s %d", dummy, &stepAcf);
fscanf(fp, "%s %d", dummy, &rangeRdf);
fscanf(fp, "%s %d", dummy, &limitRdf);
fscanf(fp, "%s %d", dummy, &sizeHistRdf);
fscanf(fp, "%s %d", dummy, &stepRdf);
00060
00061
00062
00063
00064
00065
00066
00067
00068
            fclose(fp);
00069
            FILE *fpSTATE;
            if((fpSTATE = fopen(inputConfig, "r")) ==NULL) {
printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00070
00071
00072
            exit(0);
00073
00074
            00075
00076
            fscanf(fpSTATE, "%s %d",
00077
                                                      dummy, &nBond);
            fscanf(fpSTATE, "%s %d", dummy, &nAtomType);
00078
            fscanf(fpSTATE, "%s %d", dummy, &nBondType);
fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00079
08000
00081
00082
00083
            region[2] *= 1.5; //Remove this when put on GitHub
00084
00085
            density = nAtom/(region[1]*region[2]);
            cells[1] = region[1] / rCut;
cells[2] = region[2] / rCut;
00086
00087
00088
            cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
            regionH[1] = 0.5*region[1];
regionH[2] = 0.5*region[2];
00089
00090
00091
00092
            //strain information
00093
            strainRate = strain/deltaT;
00094
            shearDisplacement = strain * region[2];
            shearVelocity = strainRate * region[2];
00095
00096
            int n:
00097
00098
            rx = (double*)malloc((nAtom + 1) * sizeof(double));
00099
            ry = (double*)malloc((nAtom + 1) * sizeof(double));
            vx = (double*)malloc((nAtom + 1) * sizeof(double));
00100
           vy = (double*)malloc((nAtom + 1) * sizeof(double));
ax = (double*)malloc((nAtom + 1) * sizeof(double));
ay = (double*)malloc((nAtom + 1) * sizeof(double));
00101
00102
00103
```

3.63 Init.c 123

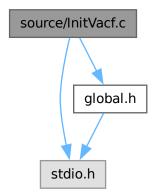
```
fax = (double*)malloc((nAtom + 1) * sizeof(double));
              fay = (double*)malloc((nAtom + 1) * sizeof(double));
00105
00106
              atomID = (int*)malloc((nAtom+1) * sizeof(int));
             atomType = (int*)malloc((nAtom+1) * sizeof(int));
00107
00108
             atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
             atomMass = (double*) malloc((nAtom + 1) * sizeof(double));
00109
             speed = (double*)malloc((nAtom + 1) * sizeof(double));
00110
00111
              atom1 = (int*)malloc((nBond+1)*sizeof(int));
00112
              atom2 = (int*)malloc((nBond+1)*sizeof(int));
00113
             BondID = (int*)malloc((nBond+1)*sizeof(int));
             BondType = (int*)malloc((nBond+1)*sizeof(int));
00114
00115
             kb = (double*)malloc((nBond+1)*sizeof(double));
00116
              ro = (double*)malloc((nBond+1)*sizeof(double));
00117
              BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00118
              BondLength = (double*) malloc((nBond+1)*sizeof(double));
             discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00119
00120
             nodeDragx = (double*) malloc((nAtom + 1) * sizeof(double));
00121
             nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00123
              ImageX = (int*)malloc((nAtom+1) * sizeof(int));
              ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00124
             rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00125
00126
             DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
00127
00128
             DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
              DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00129
00130
              DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00131
              for(int n = 0; n \le nAtom; n++) {
00132
               DeltaXijOldPair[n] = (double*) malloc((nAtom+1) * sizeof(double));
              DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00133
00134
00135
             molID = (int*)malloc((nAtom+1) * sizeof(int));
00136
00137
              for(n = 1; n <= nAtom; n ++) {</pre>
00138
               atomMass[n] = 1.0;
00139
00140
              fscanf(fpSTATE, "%s\n", dummy);
00142
             for(n = 1; n <= nAtom; n ++)</pre>
               fscanf(fpSTATE, "%d %d %d %lf %lf %lf %lf %lf %lf \n", &atomID[n], &molID[n], &atomType[n], \\
00143
          &atomRadius[n], &rx[n], &ry[n], &vx[n], &vy[n]);
00144
00145
00146
              fscanf(fpSTATE, "%s\n", dummy);
              for (n=1; n<=nBond; n++)</pre>
00148
               fscanf(fpSTATE, "%d %d %d %d %lf %lf <math>n", \&BondID[n], \&BondType[n], \&atoml[n], \&atoml[
         &ro[n]);
00149
00150
             fclose(fpSTATE);
00151
00152
            //2D-List of bonded atoms. This is used to remove pair interaction
            //calculation for the bonded atoms
00153
00154
               isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
             00155
00156
00157
00158
00159
00160
00161
             for (n = 1; n <= nBond; n++) {</pre>
00162
               int i = atom1[n];
00163
00164
                 int j = atom2[n];
                 isBonded[i][j] = 1;
00165
00166
                 isBonded[j][i] = 1; // symmetric
00167 }
00168
00169
            //For thermostate, update in final version
00170 thermo = 'C';
00172
00173 // List the interface atoms
00174 nAtomInterface = 0;
00175 nAtomBlock = 0;
00176 nDiscInterface = 0;
00177
           double InterfaceWidth, bigDiameter;
           bigDiameter = 2.8;
00178
00179
           InterfaceWidth = 5.0 * bigDiameter;
00180
00181
            for (n = 1: n \le nAtom: n++) {
             if (fabs(ry[n]) < InterfaceWidth) {</pre>
00182
             nAtomInterface++;
00183
00184
00185
              if(molID[n] == 2){
00186
             nAtomBlock++;
00187
00188
              if (atomRadius[n] != 0.0) {
```

```
nDiscInterface++;
00190
00191
00192
                   atomIDInterface = (int*)malloc((nAtomInterface+1)*sizeof(int));
00193
00194
                 int m:
                 m = 1;
00195
00196
                  for (n=1; n<=nAtom; n++) {</pre>
00197
                  if(fabs(ry[n]) < InterfaceWidth){</pre>
00198
                   atomIDInterface[m] = atomID[n];
00199
                  m++;
00200
                  } }
00201
                  nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00202
00203
                  PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
                  Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00204
00205
                  PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00206
                  PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
                     fprintf(fpresult, "------\n");
fprintf(fpresult, "------PARAMETERS-----\n");
fprintf(fpresult, "------\n");
fprintf(fpresult, "nAtom\t\t\t\t\d\n", nAtom);
fprintf(fpresult, "nBond\t\t\t\d\n", nBond);
fprintf(fpresult, "nAtomBlock\t\t\d\n", nAtomBlock);
fprintf(fpresult, "nAtomInterface\t\d\n", nAtomInterface);
fprintf(fpresult, "nDiscInterface\t\d\n", nDiscInterface);
fprintf(fpresult, "gamman\t\t\t\0.6g\n", gamman);
fprintf(fpresult, "strain\t\t\0.6g\n", strain;
fprintf(fpresult, "strainRate\t\0.6g\n", strainRate);
fprintf(fpresult, "FyBylx\t\t\0.6g\n", FyBylx);
fprintf(fpresult, "fxByfy\t\t\0.6g\n", fxByfy);
fprintf(fpresult, "fxByfy\t\t\0.6g\n", fxByfy);
fprintf(fpresult, "DeltaY\t\t\0.6g\n", DeltaY);
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
                      fprint(fpresult, "DeltaY\t\t\t\0.6g\n", DeltaY);
fprintf(fpresult, "DeltaX\t\t\$0.6g\n", DeltaY);
fprintf(fpresult, "DeltaX\t\t\$0.6g\n", DeltaX);
fprintf(fpresult, "HaltCondition\t\$0.6g\n", HaltCondition);
fprintf(fpresult, "kappa\t\t\$g\n", kappa);
fprintf(fpresult, "density\t\t\$g\n", density);
00222
00223
00224
00225
                      fprintf(fpresult, "density\t\t\t\g\n", density);
fprintf(fpresult, "rCut\t\t\t\g\n", rCut);
fprintf(fpresult, "deltaT\t\t\t\g\n", deltaT);
fprintf(fpresult, "stepEquil\t\t\g\n", deltaT);
fprintf(fpresult, "stepEquil\t\t\g\n", stepEquil);
fprintf(fpresult, "stepLimit\t\t\g\n", stepLimit);
fprintf(fpresult, "region[1]\t\t\g\0.161f\n", region[1]);
fprintf(fpresult, "region[2]\t\t\g\0.161f\n", region[2]);
00227
00228
00229
00230
00231
00232
                      fprintf(fpresult, region[2]\t\t\sd\n", cells[1]);
fprintf(fpresult, "cells[1]\t\t\sd\n", cells[1]);
fprintf(fpresult, "cells[2]\t\t\sd\n", cells[2]);
fprintf(fpresult, "solver\t\t\s\n", solver);
fprintf(fpresult, "boundary\t\t\s\s\n", xBoundary, yBoundary);
fprintf(fpresult, "DampFlag\t\s\d\n", DampFlag);
00233
00234
00235
00236
00237
00238
00239
                       00240
00241
           PairEnergyPerAtom BondEnergyPerAtom Press VirialSum\n");
fprintf(fpvrms, "#timeNow\tVrms \n");
fprintf(fpcom, "#timeNow\tComX\tComY\n");
00242
00243
00244
00245 /\star //Uncomment the following as per your acquirement
                     //Uncomment the following as per your acquirement
fprintf(fpstress, "strain %lf\n", strain);
fprintf(fpstress, "region[1] %lf\n", region[1]);
fprintf(fpstress, "region[2] %lf\n", region[2]);
fprintf(fpstress, "#timeNow virSumxx virSumxy virSumxy pressure\n");
00246
00247
00248
00249
00250
                       fprintf(fpmomentum, "#timeNow Px Py\n");
00251 */
00252
           if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
   (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
   fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
   allowed.\n", xBoundary, yBoundary);
   exit(EXIT_FAILURE); // Exit with failure status
00253
00254
00255
00256
00257
00258
00259 }
```

## 3.64 source/InitVacf.c File Reference

```
#include <stdio.h>
#include "global.h"
```

Include dependency graph for InitVacf.c:



## **Functions**

- void ZeroVacf ()
- void InitVacf ()

## 3.64.1 Function Documentation

## 3.64.1.1 InitVacf()

```
void InitVacf ( )
```

Definition at line 26 of file InitVacf.c.

References indexAcf, nBuffAcf, nValAcf, and ZeroVacf().

Referenced by SetupJob().

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.64.1.2 ZeroVacf()

```
void ZeroVacf ( )
```

### Definition at line 25 of file ZeroVacf.c.

Referenced by InitVacf().

Here is the caller graph for this function:



## 3.65 InitVacf.c

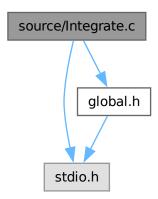
### Go to the documentation of this file.

```
00001 /*
00002
         * This file is part of Lamina.
00003
00004
         \star Lamina is free software: you can redistribute it and/or modify
         * it under the terms of the GNU General Public License as published by * the Free Software Foundation, either version 3 of the License, or
00005
00006
         * (at your option) any later version.
00007
80000
00009
         \star Lamina is distributed in the hope that it will be useful,
         * but WITHOUT ANY WARRANTY; without even the implied warranty of * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00010
00011
00012 * GNU General Public License for more details.
00013
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
         Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
```

```
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void ZeroVacf();
00026 void InitVacf() {
00027    int nb;
00028    for(nb = 1; nb <= nBuffAcf; nb ++)
00029        indexAcf[nb] = -(nb-1)*nValAcf/nBuffAcf;
00030    ZeroVacf();
00031 }</pre>
```

# 3.66 source/Integrate.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for Integrate.c:
```



#### **Functions**

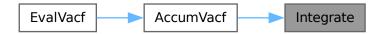
• double Integrate (double \*f, int nf)

## 3.66.1 Function Documentation

### 3.66.1.1 Integrate()

Referenced by AccumVacf().

Here is the caller graph for this function:



# 3.67 Integrate.c

#### Go to the documentation of this file.

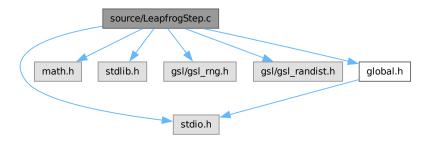
```
00002
         * This file is part of Lamina.
00003
         \star Lamina is free software: you can redistribute it and/or modify \star it under the terms of the GNU General Public License as published by
00004
00005
00006
          * the Free Software Foundation, either version 3 of the License, or
         * (at your option) any later version.
80000
00009 \,\,\star\,\, Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details.
00014 \,\star\, You should have received a copy of the GNU General Public License
00015
         * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 double Integrate(double *f, int nf) {
00025 double s;
00027 int i;
00028 s = 0.5*(f[1] + f[nf]);
00029 for(i = 2; i <= nf - 1; i ++)
00030 s += f[i];
00031 return(s);
00032 }
00033
```

# 3.68 source/LeapfrogStep.c File Reference

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
```

```
#include "global.h"
```

Include dependency graph for LeapfrogStep.c:



#### **Functions**

void LeapfrogStep (char thermo, gsl\_rng \*rnd)

## 3.68.1 Function Documentation

#### 3.68.1.1 LeapfrogStep()

## Definition at line 28 of file LeapfrogStep.c.

```
00028
00029 double temperature, GAMMA;
00030 \text{ GAMMA} = 100;
00031
00032 double *TValSum;
00033 TValSum = (double*)malloc((nAtom + 1) * sizeof(double));
00034
00035
         if(stepCount <= stepEquil){
  double gSum, varS, massS;
  temperature = 1./GAMMA;</pre>
00036
00037
00038
00039
          if(stepCount == 1) varS = 0.;
double A, S1, S2, T;
00040
00041
00042
            int n;
            S1 = 0.; S2 = 0.; qSum = 0.; massS = 0.1;
00043
00044
00045
            vvSum = 0.;
            double halfdt = 0.5*deltaT;
00046
            for (n = 1; n <= nAtom; n++) {
    T = vx[n] + halfdt * ax[n];
00047
00048
              S1 += T * ax[n];
00049
00050
              S2 += Sqr(T);
00051
00052
              T = vy[n] + halfdt * ay[n];
              S1 += T * ay[n];
00053
00054
              S2 += Sqr(T);
00055
             vvSum += (Sqr(vx[n]) + Sqr(vy[n]));
00056
00057
00058
            A = -S1 / S2;
            S2 = vvSum;
00059
00060
            double C = 1 + A*deltaT;
double D = deltaT * (1 + 0.5 * A * deltaT);
00061
00062
00063
```

```
00064
            int i,j;
00065
            real dr[NDIM+1], r, rr, ri, rrCut;
00066
            double vv;
00067
            double uVal, AA, AASum;
00068
00069
            double TVal;
00070
00071
            double deno, VVSum;
            deno = 0.;
VVSum = 0.;
00072
00073
           AASum = 0.;
00074
00075
00076
         for (n=1; n<=nAtom; n++)</pre>
00077
             TValSum[n] = 0.;
00078
00079
         rrCut = Sqr(rCut);
08000
00081 /*****Calculating Configarational temperature*****/
00082 //Solving the equation of motion here
00083 if(thermo == 'C'){
00084 for(i = 1 ; i <= nAtom; i ++) {
            for(j = i+1; j <= nAtom; j ++) {
  dr[1] = rx[i] - rx[j];</pre>
00085
00086
00087
              if(fabs(dr[1]) > regionH[1])
00088
           dr[1] -= SignR(region[1], dr[1]);
00089
            dr[2] = ry[i] - ry[j];
if(fabs(dr[2]) > regionH[2])
00090
00091
00092
            dr[2] -= SignR(region[2], dr[2]);
00093
             rr = Sqr(dr[1]) + Sqr(dr[2]);
00094
00095
             if(rr < rrCut ){
00096
             r = sqrt(rr);
00097
             ri = 1/r;
00098
             uVal = ri*exp(-kappa*r);
00099
             TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
TValSum[i] += TVal;
00100
00102
             TValSum[j] += TVal;
00103
00104
             AA = Sqr(ax[i]) + Sqr(ay[i]);
00105
             AASum += AA;
00106
             vv = Sqr(vx[i]) + Sqr(vv[i]);
00107
             VVSum += vv;
             deno += TValSum[i];
00108
00109 }
00110
             double gSumconfig, varSconfig, massSconfig;
if(stepCount == 1) varSconfig = 0.;
gSumconfig = 0.; massSconfig = 2.0;
00111
00112
00113
00114
00115
             gSumconfig = (AASum/temperature - deno)/massSconfig;
             varSconfig += deltaT*gSumconfig;
00116
00117
          /*****Configarational Nose-Hoover thermostat****/
for (n = 1; n <= nAtom; n++) {
   vx[n] += deltaT * ax[n];
   rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);</pre>
00118
00119
00121
              rvy[n] += deltaT * ay[n];
ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
00122
00123
00124
             /****Kinetic Nose-Hoover thermostat****/
00125
         }else if(thermo == 'N') {
  gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
00126
00127
00128
            varS += deltaT*gSum;
00129
          for (n = 1; n <= nAtom; n++) {</pre>
              vx[n] += deltaT * (ax[n] - varS *vx[n]);
rx[n] += deltaT * vx[n];
vy[n] += deltaT * (ay[n] - varS *vy[n]);
00130
00131
00132
              ry[n] += deltaT * vy[n];
00133
00134
00135
              /*****for \ Gaussian \ thermostat****/
vx[n] = C * vx[n] + D * ax[n];
rx[n] += deltaT * vx[n];
00138
00139
00140
              vy[n] = C * vy[n] + D * ay[n];
00141
              ry[n] += deltaT * vy[n];
00142
          }else if (thermo == 'I') {
00143
         double nu = 0.03066;
00144
         double var = sqrt(2*nu/(GAMMA*deltaT));
00145
00146
         double scale = 1. + nu*deltaT/2.;
00147
         double scale_v = 2./scale - 1.;
00148
         double scale_f = deltaT/scale;
00149
         int n;
         for (n = 1; n <= nAtom; n ++) {
00150
```

3.69 LeapfrogStep.c 131

```
vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
              rx[n] += deltaT * vx[n];
00153
              vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
00154
              ry[n] += deltaT * vy[n];
00155
00156
00157
       }else{
00158
00159
           for (n = 1 ; n <= nAtom ; n ++) {</pre>
             vx[n] += deltaT * ax[n];
rx[n] += deltaT * vx[n];
vy[n] += deltaT * ay[n];
00160
00161
00162
              ry[n] += deltaT * vy[n];
00163
00164
00165
        }
00166 }
```

References ax, ay, deltaT, kappa, nAtom, NDIM, rCut, region, regionH, rx, ry, SignR, Sqr, stepCount, stepEquil, thermo, vvSum, vx, and vy.

# 3.69 LeapfrogStep.c

#### Go to the documentation of this file.

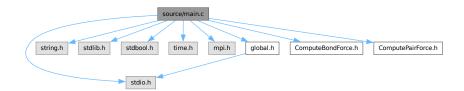
```
00002
       * This file is part of Lamina.
00003
       \star Lamina is free software: you can redistribute it and/or modify
00004
       * it under the terms of the GNU General Public License as published by
00005
       * the Free Software Foundation, either version 3 of the License, or
00006
        * (at your option) any later version.
80000
00009
       * Lamina is distributed in the hope that it will be useful,
00010 \star but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 \star MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
       * You should have received a copy of the GNU General Public License
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00020 #include<stdio.h>
00021 #include<math.h>
00022 #include<stdlib.h>
00023 #include <gsl/gsl_rng.h>
00025 #include <gs1/gs1_ing.h>
00024 #include <gs1/gs1_randist.h>
00025 #include"global.h"
00026
00027
00028 void LeapfrogStep(char thermo, gsl_rng * rnd){
00029 double temperature, GAMMA;
00030 GAMMA = 100;
00032 double *TValSum;
00033 TValSum = (double*)malloc((nAtom + 1) * sizeof(double));
00034
00035
00036
         if(stepCount <= stepEquil){
00037
         double gSum, varS, massS;
           temperature = 1./GAMMA;
00039
00040
          if(stepCount == 1) varS = 0.;
00041
           double A, S1, S2, T;
00042
           int n;
00043
           S1 = 0.; S2 = 0.; gSum = 0.; massS = 0.1;
00044
00045
00046
           double halfdt = 0.5*deltaT;
00047
            for (n = 1; n <= nAtom; n++) {</pre>
             T = vx[n] + halfdt * ax[n];
S1 += T * ax[n];
00048
00049
00050
             S2 += Sqr(T);
00051
00052
              T = vy[n] + halfdt * ay[n];
             S1 += T * ay[n];
S2 += Sqr(T);
00053
00054
            vvSum += (Sqr(vx[n]) + Sqr(vy[n]));
00055
00056
00057
```

```
A = -S1 / S2;
00059
            S2 = vvSum;
00060
           double C = 1 + A*deltaT;
double D = deltaT * (1 + 0.5 * A * deltaT);
00061
00062
00063
00064
00065
            real dr[NDIM+1], r, rr, ri, rrCut;
00066
            double vv;
00067
            double uVal, AA, AASum;
00068
           double TVal;
00069
00070
00071
            double deno, VVSum;
00072
            deno = 0.;
           VVSum = 0.;
AASum = 0.;
00073
00074
00075
00076
         for (n=1; n<=nAtom; n++)</pre>
00077
             TValSum[n] = 0.;
00078
00079
         rrCut = Sqr(rCut);
08000
00081 /*****Calculating Configarational temperature*****/
00082 //Solving the equation of motion here
00083 if(thermo == 'C'){
00084 for(i = 1 ; i <= nAtom; i ++) {
          for(j = i+1; j <= nAtom; j ++) {
  dr[1] = rx[i] - rx[j];</pre>
00085
00086
00087
              if(fabs(dr[1]) > regionH[1])
00088
            dr[1] -= SignR(region[1], dr[1]);
00089
00090
             dr[2] = ry[i] - ry[j];
00091
              if(fabs(dr[2]) > regionH[2])
00092
            dr[2] -= SignR(region[2], dr[2]);
00093
00094
             rr = Sqr(dr[1]) + Sqr(dr[2]);
             if(rr < rrCut ) {</pre>
00096
             r = sqrt(rr);
00097
             ri = 1/r;
00098
             uVal = ri*exp(-kappa*r);
00099
             TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
TValSum[i] += TVal;
TValSum[j] += TVal;
00100
00101
00102
00103
00104
             AA = Sqr(ax[i]) + Sqr(ay[i]);
00105
             AASum += AA;
             vv = Sqr(vx[i]) + Sqr(vy[i]);
00106
00107
             VVSum += vv;
             deno += TValSum[i];
00108
00109 }
00110
             double gSumconfig, varSconfig, massSconfig;
if(stepCount == 1) varSconfig = 0.;
00111
00112
00113
             gSumconfig = 0.; massSconfig = 2.0;
00114
00115
             gSumconfig = (AASum/temperature - deno)/massSconfig;
00116
             varSconfig += deltaT*gSumconfig;
00117
00118
              /*****Configarational Nose-Hoover thermostat****/
          for (n = 1; n <= nAtom; n++) {
    vx[n] += deltaT * ax[n];
    rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);</pre>
00119
00120
00121
00122
              vy[n] += deltaT * ay[n];
              ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
00123
00124
              /****Kinetic Nose-Hoover thermostat****/
00125
         }else if(thermo == 'N') {
  gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
00126
00127
00128
            varS += deltaT*gSum;
           for (n = 1; n <= nAtom; n++) {</pre>
00129
              rx[n] += deltaT * (ax[n] - varS *vx[n]);
rx[n] += deltaT * vx[n];
vy[n] += deltaT * (ay[n] - varS *vy[n]);
ry[n] += deltaT * vy[n];
00130
00131
00132
00133
00134
        /*****for Gaussian thermostat****/
}else if(thermo == 'G'){
00135
00136
              for (n = 1; n <= nAtom; n++) {
00137
              vx[n] = C * vx[n] + D * ax[n];
rx[n] += deltaT * vx[n];
00138
00139
00140
              vy[n] = C * vy[n] + D * ay[n];
00141
              ry[n] += deltaT * vy[n];
00142
          }else if (thermo == 'L') {
00143
00144
         double nu = 0.03066;
```

```
double var = sqrt(2*nu/(GAMMA*deltaT));
          double scale = 1. + nu*deltaT/2.;
double scale_v = 2./scale - 1.;
double scale_f = deltaT/scale;
00146
00147
00148
00149
          int n;
          for (n = 1; n <= nAtom; n ++) {
00150
00151
               vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
00152
               rx[n] += deltaT * vx[n];
00153
               vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
00154
               ry[n] += deltaT * vy[n];
00155
            }
00156
00157 }else{
00158
00159
             for (n = 1 ; n <= nAtom ; n ++) {</pre>
               vx[n] += deltaT * ax[n];
rx[n] += deltaT * vx[n];
vy[n] += deltaT * ay[n];
ry[n] += deltaT * vy[n];
00160
00161
00162
00163
00164
00165 }
00166 }
00167
```

# 3.70 source/main.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <mpi.h>
#include "global.h"
#include "ComputeBondForce.h"
#include "ComputePairForce.h"
Include dependency graph for main.c:
```



## **Macros**

• #define DEFINE\_GLOBALS

# **Functions**

- void Init ()
- void SetupJob ()
- void EvalSpacetimeCorr ()
- void Trajectory ()
- void DumpState ()
- void ComputeForcesCells ()
- void LeapfrogStep ()

- void BrownianStep ()
- void ApplyBoundaryCond ()
- void EvalProps ()
- void EvalVacf ()
- void EvalRdf ()
- void AccumProps (int icode)
- void PrintSummary ()
- void PrintVrms ()
- void DumpBonds ()
- void VelocityVerletStep (int icode)
- void ApplyForce ()
- void ApplyDrivingForce ()
- void ApplyShear ()
- void ApplyLeesEdwardsBoundaryCond ()
- void PrintStress ()
- void Close ()
- void PrintMomentum ()
- void DisplaceAtoms ()
- void DumpRestart ()
- bool HaltConditionCheck (double value, int stepCount)
- void EvalCom ()
- void PrintCom ()
- void EvalVrms ()
- · void EvalUnwrap ()
- void DumpPairs ()
- void ApplyViscous ()
- int main (int argc, char \*\*argv)

## **Variables**

char \* prefix = NULL

# 3.70.1 Macro Definition Documentation

## 3.70.1.1 DEFINE\_GLOBALS

#define DEFINE\_GLOBALS

Definition at line 7 of file main.c.

#### 3.70.2 Function Documentation

#### 3.70.2.1 AccumProps()

```
void AccumProps (
                            int icode )
Definition at line 25 of file AccumProps.c.
00025
00026
              if(icode == 0){}
00027 sPotEnergy = ssPotEnergy = 0.;
00028 sKinEnergy = ssKinEnergy = 0.;
00029 sPressure = ssPressure = 0.;
00030 sTotEnergy = ssTotEnergy = 0.;
00031 svirSum = 0.;

00032 }else if(icode == 1) {

00033 sPotEnergy += potEnergy;

00034 ssPotEnergy += Sqr(potEnergy);

00035 sKinEnergy += kinEnergy;
00036 ssKinEnergy += Sqr(kinEnergy);
00030 sshithlergy += 3qr(krimhergy),

00037 sTotEnergy += totEnergy;

00038 ssTotEnergy += Sqr(totEnergy);

00039 sPressure += pressure;

00040 ssPressure += Sqr(pressure);
00040 ssPressure += Sqr(pressure);
00041 svirSum += virSum;
00042 }else if(icode == 2) {
00043 sPotEnergy /= stepAvg;
00044 ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045 sTotEnergy /= stepAvg;
00046 stotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047 sKinEnergy /= stepAvg;
00048 ssKinEnergy /= stepAvg;
00048 ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049 sPressure /= stepAvg;
00050 ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051 svirSum /= stepAvg;
00052 } }
```

#### 3.70.2.2 ApplyBoundaryCond()

void ApplyBoundaryCond ( )

Definition at line 27 of file ApplyBoundaryCond.c.

```
00027
00028
         for (n = 1 ; n <= nAtom ; n ++) {</pre>
00029
00030
          if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){
                                                                                                // P.B.C along x and y axis
          rx[n] -= region[1]*rint(rx[n]/region[1]);
ry[n] -= region[2]*rint(ry[n]/region[2]);
} else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0) { //R.B.C. along x and y
00031
00032
00033
      axis
00034
             if((rx[n] + atomRadius[n]) >= regionH[1]){
00035
                 rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n];
00036
             \inf((rx[n]-atomRadius[n]) < -regionH[1]){
00037
                rx[n] = -0.999999 * regionH[1] + atomRadius[n]; vx[n] = -vx[n];
00038
00039
            if((rv[n] + atomRadius[n])>= regionH[2]){
                 ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n];
00040
00041
             }if((ry[n]-atomRadius[n]) < -regionH[2]){</pre>
00042
                ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n];
00043
            else if (strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "r") == 0){ //P.B.C. along x and R.B.C
00044
      along v axis
            rx[n] -= region[1]*rint(rx[n]/region[1]);
00046
            if((ry[n] + atomRadius[n]) >= regionH[2]){
             ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
}if((ry[n] - atomRadius[n]) < -regionH[2]){</pre>
00047
00048
                ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n];
00049
00050
         } }
00051
           else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){      //R.B.C. along x and P.B.C
       along y axis
00052
           if((rx[n] + atomRadius[n]) >= regionH[1]){
              \begin{array}{lll} & rx[n] = 0.999999*regionH[1] - atomRadius[n]; \ vx[n] = -vx[n]; \\ & if((rx[n] - atomRadius[n]) < -regionH[1]) \{ \\ & rx[n] = -0.999999*regionH[1] + atomRadius[n]; \ vx[n] = -vx[n]; \\ \end{array} 
00053
00054
00055
00056
00057
             ry[n] -= region[2]*rint(ry[n]/region[2]);
```

References atomRadius, fpresult, nAtom, region, regionH, rx, ry, vx, vy, xBoundary, and yBoundary.

Referenced by main().

Here is the caller graph for this function:



#### 3.70.2.3 ApplyDrivingForce()

```
void ApplyDrivingForce ( )
```

00025

Definition at line 25 of file ApplyDrivingForce.c.

```
00026
        int n:
        double Vxblock, Vyblock;
00028 double Vxsubstrate, Vysubstrate;
00029  Vxblock = 0.0;  Vyblock = 0.0;
00030  Vxsubstrate = 0.0;  Vysubstrate = 0.0;
00031 double gammav;
00032 gammav = 0.0;
00033
00034
        double count_substrate = 0;
00035
        double count_block = 0;
00036
        for(n = 1; n <= nAtom; n ++) {
  if(atomType[n] == 1 || atomType[n] == 2) {
  Vxsubstrate += vx[n]; Vysubstrate += vy[n];</pre>
00037
00038
00039
00040
         count_substrate++;
00041
00042
          if(atomType[n] == 3 || atomType[n] == 4){
00043
         Vxblock += vx[n]; Vyblock += vy[n];
         count_block++;
00044
00045
         } }
00046
00047
          if(count_substrate > 0) {
           Vxsubstrate /= count_substrate;
Vysubstrate /= count_substrate;
00048
00049
00050
00051
00052
         if(count_block > 0) {
00053
           Vxblock /= count_block;
00054
          Vyblock /= count_block;
00055
00056
00057
         for (n = 1 ; n <= nAtom; n ++) {</pre>
00058
         if (atomType[n] == 1 || atomType[n] == 2) {
00059
         ax[n] += -gammav * (vx[n] - Vxsubstrate);
00060
          ay[n] += -gammav * (vy[n] - Vysubstrate);
00061
00062
         if(atomType[n] == 3 || atomType[n] == 4){
         ax[n] += -gammav * (vx[n] - Vxblock);
ay[n] += -gammav * (vy[n] - Vyblock);
00063
00064
        } } }
```

References atomType, ax, ay, nAtom, vx, and vy.

## 3.70.2.4 ApplyForce()

```
void ApplyForce ( )
```

Definition at line 25 of file ApplyForce.c.

```
00025 {
00026 int n;
00027 double lx;
00028 lx = regionH[1];
00029 fy = (FyBylx * lx)/nAtomBlock;
00030 fx = fxByfy * fy;
00031 for (n = 1; n <= nAtom; n ++) {
00032 if (molID[n] == 2) {
00033 ax[n] += fx;
00034 ay[n] -= fy;
00035 } }
```

References ax, ay, fx, fxByfy, fy, FyBylx, molID, nAtom, nAtomBlock, and regionH.

Referenced by main().

Here is the caller graph for this function:



#### 3.70.2.5 ApplyLeesEdwardsBoundaryCond()

```
void ApplyLeesEdwardsBoundaryCond ( )
```

Definition at line 25 of file ApplyLeesEdwardsBoundaryCond.c.

```
00025
00026
00027 for (n = 1; n <= nAtom; n++) {
00028 //PBC along x-direction
00034 //LEBC along y-direction
if(rx[n] < -regionH[1]) rx[n] += region[1];
//vx[n] -= shearVelocity;
ry[n] -= region[2];</pre>
00037
00038
00039
00040
        }else if(ry[n] < -regionH[2]){</pre>
00041
         rx[n] += shearDisplacement;
         if(rx[n] >= regionH[1]) rx[n] -= region[1];
//vx[n] += shearVelocity;
ry[n] += region[2];
00042
00043
00044
00045
00046
00047 }
```

References nAtom, region, regionH, rx, ry, and shearDisplacement.

## 3.70.2.6 ApplyShear()

```
void ApplyShear ( )
```

Definition at line 25 of file ApplyShear.c.

```
00025 {
00026 int n;
00027 for(n = 1; n <= nAtom; n ++) {
00028 rx[n] += strain * ry[n];
00029 //vx[n] += stranRate * ry[n];
00030 } }
```

References nAtom, rx, ry, and strain.

#### 3.70.2.7 ApplyViscous()

```
void ApplyViscous ( )
```

Definition at line 25 of file ApplyViscous.c.

```
00025 {
00026 int n;
00027 double gammav;
00028 gammav = 1.0;
00029 for(n = 1; n <= nAtom; n ++) {
00030 ax[n] += -gammav * vx[n];
00031 ay[n] += -gammav * vy[n];
00032 } }
```

References ax, ay, nAtom, vx, and vy.

## 3.70.2.8 BrownianStep()

void BrownianStep ( )

Definition at line 26 of file BrownianStep.c.

```
00026
            if (stepCount <= stepEquil) {</pre>
00027
00028
              double A, S1, S2, T;
00029
              int n;
00030
              S1 = 0.; S2 = 0;
              double halfdt = 0.5*deltaT;
for (n = 1; n <= nAtom; n++) {</pre>
00031
00032
                T = vx[n] + halfdt * ax[n];

S1 += T * ax[n];
00033
00034
00035
                S2 += Sqr(T);
00036
                 T = vy[n] + halfdt * ay[n];
S1 += T * ay[n];
S2 += Sqr(T);
00037
00038
00039
00040
00041
              A = -S1 / S2;
              double C = 1 + A*deltaT;
double D = deltaT * (1 + 0.5 * A * deltaT);
00042
00043
              for (n = 1; n <= nAtom; n++) {
  vx[n] = C * vx[n] + D * ax[n];
  rx[n] += deltaT * vx[n];
  vy[n] = C * vy[n] + D * ay[n];
  ry[n] += deltaT * vy[n];</pre>
00044
00045
00046
00047
00048
00049
00050
           }else{
00051
                 int n;
                 //SETTING TEMP = 0.0
00052
00053
                if (stepCount == stepEquil+1) {
                for (n = 1; n <= nAtom; n ++) {
    vx[n] = 0.0;
    vy[n] = 0.0;</pre>
00054
00055
00056
00057
00058
                double zeta = 1.0;
                double dx, dy;

for(n = 1; n <= nAtom; n ++) {
00059
00060
00061
                 dx = rx[n];
```

References ax, ay, deltaT, nAtom, rx, ry, Sqr, stepCount, stepEquil, vx, and vy.

## 3.70.2.9 Close()

```
void Close ( )
```

```
00027
        free(ry);
00028
        free(vx);
00029
        free(vy);
00030
        free(ax);
00031
        free(ay);
00032
00033
        free(fay);
00034
        free(cellList);
00035
00036
        free(atomID); free(atomType); free(atomRadius); free(atomMass);
        free(speed);
00038
        free(atom1); free(atom2); free(BondID);
00039
        free(BondType); free(kb); free(ro);
        free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00040
        free(atomIDInterface);
00041
00042
        free(PairID); free(Pairatom1); free(Pairatom2);
00043
        free(PairXij); free(PairYij);
00044
00045
        free (DeltaXijOld);
00046
        free (DeltaYijOld);
00047
00048
        free (molID);
00049
00050
        for (n = 0; n <= nAtom; n++) {</pre>
00051
         free(isBonded[n]);
00052
00053
         free(isBonded);
00054
00055
00056
00057
        for (n = 0; n <= nAtom; n++) {</pre>
00058
         free(DeltaXijOldPair[n]);
00059
         free (DeltaYijOldPair[n]);
00060
00061
          free(DeltaXijOldPair);
          free (DeltaYijOldPair);
00062
00063
00064
        for (n = 0; n <= nBuffCorr; n++) {</pre>
00065
         free(cfOrg[n]);
00066
          free(spacetimeCorr[n]);
00067
00068
        free (cfOrg);
00069
        free(spacetimeCorr);
        free(cfVal);
00070
00071
        free(indexCorr);
00072
        free(spacetimeCorrAv);
00073
00074
        free(indexAcf);
00075
        free(viscAcfOrg);
        free(viscAcfAv);
for(n = 0 ; n <= nBuffAcf ; n ++)</pre>
00076
00077
          free(viscAcf[n]);
00078
00079
        free(viscAcf);
08000
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, BondID, BondType, cellList, cfOrg, cfVal, DeltaXijOld, DeltaXijOldPair, DeltaYijOld, DeltaYijOldPair, fax, fay, ImageX,

ImageY, indexAcf, indexCorr, isBonded, kb, molID, nAtom, nBuffAcf, nBuffCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, ro, rx, rxUnwrap, ry, ryUnwrap, spacetimeCorr, spacetimeCorrAv, speed, viscAcf, viscAcfAv, viscAcfOrg, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



### 3.70.2.10 ComputeForcesCells()

void ComputeForcesCells ( )

```
Definition at line 25 of file ComputeForcesCells.c.
```

```
00025
         double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
int iofX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
00026
00027
00028
00029
               iofY[] = {0, 0, 0, 1, 1, 1, 0, -1, -1, -1};
00030
         invWid[1] = cells[1]/region[1];
invWid[2] = cells[2]/region[2];
00031
00032
00033
00034
         for(n = nAtom+1; n <= nAtom+cells[1]*cells[2]; n++)</pre>
           cellList[n] = 0;
00036
      c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) +
nAtom+ 1;
00037
00038
           cellList[n] = cellList[c];
cellList[c] = n;
00039
00040
00041
00042
00043
          for (n = 1 ; n \le nAtom ; n ++) {
          ax[n] = 0.;
00044
           ay[n] = 0.;
00045
00046
00047
00048
         uSum = 0.0;
         virSum = 0.0;
rfAtom = 0.0;
00049
00050
00051
         RadiusIJ = 0.0;
00052
00053
         gamman = 1.0;
00054
         double vr[NDIM+1], fd, fdVal, rrinv;
00055
         rrinv = 0.0;
00056
         fd = 0.0;
00057
         fdVal = 0.0;
00058
00059
         int start = 1 + rank*(cells[2]/size);
00060
         int end = (rank+1)*(cells[2]/size);
00061
          for(m1Y = start ; m1Y \le end ; m1Y ++){
00062
           for(m1X = 1; m1X <= cells[1]; m1X ++) {
    m1 = (m1Y-1) * cells[1] + m1X + nAtom;
    for(offset = 1; offset <= 9; offset ++) {</pre>
00063
00064
00065
00066
            m2X = m1X + iofX[offset]; shift[1] = 0.;
00067
            if(m2X > cells[1]){
00068
              m2X = 1; shift[1] = region[1];
00069
            else if (m2X == 0) {
00070
              m2X = cells[1]; shift[1] = -region[1];
00071
            m2Y = m1Y + iofY[offset]; shift[2] = 0.;
```

```
if (m2Y > cells[2]) {
                     m2Y = 1; shift[2] = region[2];
}else if(m2Y == 0){
00074
00075
00076
                       m2Y = cells[2]; shift[2] = -region[2];
00077
00078
                    m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079
                     I = cellList[m1];
00080
                     while (I > 0) {
00081
                        J = cellList[m2];
00082
                         while (J > 0) {
                              if (m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)) {
00083
                                dr[1] = rx[I] - rx[J] - shift[1];
dr[2] = ry[I] - ry[J] - shift[2];
rr = Sqr(dr[1]) + Sqr(dr[2]);
00084
00085
00086
00087
                                  RadiusIJ = atomRadius[I] + atomRadius[J];
00088
                                  SqrRadiusIJ = Sqr(RadiusIJ);
00089
                                 if (rr < SgrRadiusIJ) {
                              r = sqrt(rr);
00090
00091
                              ri = 1.0/r;
00092
                                              rrinv = 1.0/rr;
                                              vr[1] = vx[I] - vx[J];
vr[2] = vy[I] - vy[J];
00093
00094
                              RadiusIJInv = 1.0/RadiusIJ;
00095
                              two individual in
00096
00097
                                              fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00098
00099
                              00100
00101
                              ax[I] += (f + fd);
00102
00103
                                              discDragx[I] += fd; //disc-disc drag
00104
00105
                              f = fcVal * dr[2];
00106
                                              fd = fdVal * dr[2];
                              ay[I] += (f + fd);
00107
                                              discDragy[I] += fd; //disc-disc drag
00108
00109
00110
                              uSum += 0.5 * uVal;
00111
                              virSum += 0.5 * fcVal * rr;
00112
                              rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00113
00114
                              }else if (m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)) {
                                dr[1] = rx[I] - rx[J] - shift[1];

dr[2] = ry[I] - ry[J] - shift[2];
00115
00116
00117
                                  rr = Sqr(dr[1]) + Sqr(dr[2]);
00118
                                  RadiusIJ = atomRadius[I] + atomRadius[J];
00119
                                 SqrRadiusIJ = Sqr(RadiusIJ);
00120
                                 if(rr < SqrRadiusIJ){</pre>
                             r = sqrt(rr);
00121
00122
                              ri = 1.0/r;
00123
                                              rrinv = 1.0/r;
                                              vr[1] = vx[I] - vx[J];
vr[2] = vy[I] - vy[J];
00124
00125
                             RadiusIJInv = 1.0/RadiusIJ;
uVal = Sqr(1.0 - r * RadiusIJInv);
fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
    fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00126
00127
00128
00130
00131
                              f = fcVal * dr[1];
                              fd = fdVal * dr[1];
ax[I] += (f + fd);
00132
00133
                                              discDragx[I] += fd; //disc-disc drag
00134
00135
00136
                              f = fcVal * dr[2];
00137
                                              fd = fdVal * dr[2];
00138
                              ay[I] += (f + fd);
00139
                                              discDragy[I] += fd; //disc-disc drag
00140
00141
                              uSum += 0.5 * uVal;
                              virSum += 0.5 * fcVal * rr;
00142
00143
                              rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00144
00145
                                      J = cellList[J];
00146
00147
00148
                          I = cellList[I];
00149
                   }
00150
00151
               }
00152
00153 }
```

References atomRadius, ax, ay, cellList, cells, discDragx, discDragy, gamman, nAtom, NDIM, RadiusIJ, RadiusIJInv, rank, region, regionH, rfAtom, rx, ry, size, Sqr, SqrRadiusIJ, uSum, virSum, vx, and vy.

### 3.70.2.11 DisplaceAtoms()

```
void DisplaceAtoms ( )
```

Definition at line 25 of file DisplaceAtoms.c.

```
00025 int n;

00027 for(n = 1; n <= nAtom; n ++) {

00028 if(molID[n] == 2) {

00029 rx[n] += DeltaX;

00030 ry[n] += DeltaY;

00031 } }
```

References DeltaX, DeltaY, molID, nAtom, rx, and ry.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.12 DumpBonds()

```
void DumpBonds ( )
```

Definition at line 24 of file DumpBonds.c.

```
00025
             //Trajectory file in LAMMPS dump format for OVITO visualization
fprintf(fpbond, "ITEM: TIMESTEP\n");
fprintf(fpbond, "%lf\n",timeNow);
fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
00026
00027
00028
00029
             fprintf(fpbond, "lTEM: NUMBER OF ENTRIES\n");
fprintf(fpbond, "%d\n", nBond);
fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
00030
00031
00032
00033
00034
              fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEqul nodeDragx1
00035
         nodeDragy1\n");
00036
              for(n=1; n<=nBond; n++)
fprintf(fpbond, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", BondID[n], BondType[n], atoml[n],</pre>
00037
00038
          atom2[n],
00039
               BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
```

References atom1, atom2, BondID, BondLength, BondType, fpbond, nBond, nodeDragx, nodeDragy, regionH, ro, and timeNow.

Referenced by main().

Here is the caller graph for this function:



#### 3.70.2.13 DumpPairs()

```
void DumpPairs ( )
Definition at line 25 of file DumpPairs.c.
00025
00026
                 //Trajectory file in LAMMPS dump format for OVITO visualization
              //Trajectory file in LAMMPS dump format for OVITO visualization fprintf (fppair, "ITEM: TIMESTEP\n"); fprintf (fppair, "%lf\n",timeNow); fprintf (fppair, "ITEM: NUMBER OF ENTRIES\n"); fprintf (fppair, "%d\n",nPairActive); fprintf (fppair, "%d\n",nPairActive); fprintf (fppair, "ITEM: BOX BOUNDS pp ff pp\n"); fprintf (fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]); fprintf (fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]); fprintf (fppair, "%lf %lf zlo zhi\n", -0.1, 0.1); fprintf (fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00028
00029
00030
00031
00032
00034
00035
00036
00037
00038
               for (n=1; n<=nPairActive; n++)</pre>
                 fprintf(fppair, "%d %d %d %0.161f %0.161f %0.161f %0.161f \n", PairID[n], Pairatom1[n],
00039
            Pairatom2[n],
00040
                 PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00041
00042
```

References discDragx, discDragy, fppair, nPairActive, Pairatom1, Pairatom2, PairID, PairXij, PairYij, regionH, and timeNow.

Referenced by main().

Here is the caller graph for this function:



### 3.70.2.14 DumpRestart()

```
void DumpRestart ( )
```

Definition at line 25 of file DumpRestart.c.

```
00025
00026
         char DUMP[256];
00027
         FILE *fpDUMP;
         sprintf(DUMP, "%s.Restart", prefix);
00029
         fpDUMP = fopen(DUMP, "w");
00030
         if(fpDUMP == NULL) {
00031
          fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032
          return;
00033
00034
         fprintf(fpDUMP, "timeNow %lf\n", timeNow);
fprintf(fpDUMP, "nAtom %d\n", nAtom);
fprintf(fpDUMP, "nBond %d\n", nBond);
fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
fprintf(fpDUMP, "nBondType %d\n", nBondType);
fprintf(fpDUMP, "region[1] %0.141f\n", region[1]);
00035
00036
00037
00038
00039
00040
00041
          fprintf(fpDUMP, "region[2] %0.141f\n", region[2]);
00042
00043
          fprintf(fpDUMP, "Atoms\n");
for(n = 1; n <= nAtom; n ++)</pre>
00044
00045
           fprintf(fpDUMP, "%d %d %d %0.21f %0.161f %0.161f %0.161f %0.161f\n", atomID[n], molID[n],
00046
       atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n]);
00047
00048
          fprintf(fpDUMP, "Bonds\n");
00049
00050
          for(n=1; n<=nBond; n++)
fprintf(fpDUMP, "%d %d %d %0.21f %0.161f\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n],</pre>
00051
       ro[n]);
00052
00053
          fclose(fpDUMP);
00054 }
```

References atom1, atom2, atomID, atomRadius, atomType, BondID, BondType, kb, molID, nAtom, nAtomType, nBond, nBondType, prefix, region, ro, rx, ry, timeNow, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



#### 3.70.2.15 DumpState()

```
void DumpState ( )
```

Definition at line 25 of file DumpState.c.

```
00033
00034
              fprintf(fpDUMP, "ITEM: TIMESTEP\n");
fprintf(fpDUMP, "%lf\n",timeNow);
fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
fprintf(fpDUMP, "%d\n",nAtom);
fprintf(fpDUMP, "NITEM: BOX BOUNDS pp pp pp\n");
fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
fprintf(fpDUMP, "%lf %lf zlo zhi\n", -0.1, 0.1);
fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
int n:
00035
00036
00037
00038
00040
00041
00042
00043
00044
               int n;
               for (n = 1; n <= nAtom; n++) {</pre>
00045
                 fprintf(fpDUMP, "%d\t %d\t %0.21f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t
          %0.16lf\n",
00047
                atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00048
00049
               fclose(fpDUMP);
00050 }
```

References atomID, atomRadius, atomType, ax, ay, molID, nAtom, prefix, regionH, rx, ry, timeNow, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



#### 3.70.2.16 EvalCom()

```
void EvalCom ( )
```

Definition at line 27 of file EvalCom.c.

```
00028
00029 ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00030 TotalMass = 0.0;
00031
00032
       for (n=1; n<=nAtom; n++) {</pre>
00033
       if(molID[n] == 2){
00034
        ComX += atomMass[n] * rxUnwrap[n];
00035
       ComY += atomMass[n] * ryUnwrap[n];
00036
       TotalMass += atomMass[n];
00037
00038
00039
        ComX = ComX/TotalMass;
00040
       ComY = ComY/TotalMass;
00041
00042
        if(timeNow == 0.0){
00043
       ComX0 = ComX; ComY0 = ComY;
00044
00045
        ComXRatio = ComX/ComX0;
                                 ComYRatio = ComY/ComY0;
00046
```

References atomMass, ComX, ComX0, ComXRatio, ComY, ComY0, ComYRatio, molID, nAtom, rxUnwrap, ryUnwrap, timeNow, and TotalMass.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.17 EvalProps()

```
void EvalProps ( )
```

Definition at line 26 of file EvalProps.c.

```
00026
00027
       real v, vv;
       virSum = 0.0;
00029
       vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
00030
       vvSum = 0.;
00031
       int n:
00032
00033
       for (n = 1; n <= nAtom; n++) {</pre>
00035
        // Initialize v with a default value to avoid "uninitialized" warning.
00036
        v = 0.0;
00037
        // X direction velocity
        if (strcmp(solver, "Verlet") == 0) {
00038
00039
        v = vx[n];
} else if (strcmp(solver, "LeapFrog") == 0) {
00041
          v = vx[n] - 0.5 * deltaT * ax[n];
00042
00043
         vSum += v;
00044
         vv += Sqr(v);
         vSumX += v;
00045
         // Y direction velocity
00046
00047
          if (strcmp(solver, "Verlet") == 0) {
00048
         v = vy[n];
         } else if (strcmp(solver, "LeapFrog") == 0) {
v = vy[n] - 0.5 * deltaT * ay[n];
00049
00050
00051
00052
         vSum += v;
00053
         vSumY += v;
         vv += Sqr(v);
00054
00055
         vvSum += vv;
00056
00057
00058
        kinEnergy = 0.5 * vvSum / nAtom ;
        uSumPairPerAtom = uSumPair / nAtom ;
        BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
     bond energy
00061
        potEnergy = uSumPairPerAtom + BondEnergyPerAtom ;
00062
        totEnergy = kinEnergy + potEnergy;
        virSumxx = virSumPairxx + virSumBondxx;
virSumyy = virSumPairyy + virSumBondyy;
00063
00064
00065
        virSumxy = virSumPairxy + virSumBondxy;
00066
        virSum = virSumPair + virSumBond;
00067
        pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00068
00069 }
```

References ax, ay, BondEnergyPerAtom, deltaT, density, kinEnergy, nAtom, NDIM, potEnergy, pressure, solver, Sqr, TotalBondEnergy, totEnergy, uSumPair, uSumPairPerAtom, virSum, virSumBond, virSumBondx, virSumBondy, virSumPair, virSumPairx, virSumPairxy, virSumPairy, virSumPairy, virSumPairy, virSumPairy, virSumy, virSumy, vvSum, vXumY, vvSum, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.18 EvalRdf()

```
void EvalRdf ( )
```

Definition at line 26 of file EvalRdf.c.

```
00026
         real dr[NDIM+1], deltaR, normFac, rr, rrRange;
         int j1, j2, n;
countRdf ++;
00028
00029
00030
         if(countRdf == 1) {
           for (n = 1 ; n <= sizeHistRdf ; n ++)
  histRdf[n] = 0.;</pre>
00031
00032
00033
00034
         rrRange = Sqr(rangeRdf);
         deltaR = rangeRdf / sizeHistRdf;
for(j1 = 1; j1 <= nAtom - 1; j1 ++) {</pre>
00035
00036
            for(j2 = j1 + 1 ; j2 <= nAtom ; j2 ++) {
00037
00038
              dr[1] = rx[j1] - rx[j2];
if(fabs(dr[1]) > regionH[1])
00039
00040
00041
            dr[1] -= SignR(region[1], dr[1]);
00042
              dr[2] = ry[j1] - ry[j2];
if(fabs(dr[2]) > regionH[2])
00043
00044
00045
            dr[2] -= SignR(region[2], dr[2]);
00046
00047
              rr = Sqr(dr[1]) + Sqr(dr[2]);
00048
00049
              if(rr < rrRange){</pre>
00050
           n = (int) (sqrt(rr)/deltaR) + 1;
00051
           histRdf[n] ++;
00052
              }
00053
           }
00054
00055
00056
         if(countRdf == limitRdf){
           normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf);
for(n = 1; n <= sizeHistRdf; n ++)</pre>
00057
00058
00059
              histRdf[n] *= normFac/(n-0.5);
00060
            // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
00061
            real rBin;
00062
            int n;
            fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
00063
            for(n = 1; n <= sizeHistRdf; n ++) {
  rBin = (n - 0.5)*rangeRdf/sizeHistRdf;</pre>
00064
00065
00066
              fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00067
00068
         }
00069
00070 }
```

References countRdf, fprdf, histRdf, limitRdf, nAtom, NDIM, rangeRdf, region, regionH, rx, ry, SignR, sizeHistRdf, Sqr, and timeNow.

## 3.70.2.19 EvalSpacetimeCorr()

void EvalSpacetimeCorr ( )

```
Definition at line 26 of file EvalSpacetimeCorr.c.
00026
           real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
real COSA, SINA, COSV, SINV;
00027
00028
           int j, m, n, nb, ni, nv;
real kMin = 2. * M_PI / region[1];
00029
00030
00031
           real kMax = M_PI;
00032
           real deltaK = (kMax - kMin) / nFunCorr;
00033
          for (j = 1; j <= 2*nFunCorr; j++)
  cfVal[j] = 0.;</pre>
00034
00035
00037
           for (n = 1; n <= nAtom; n++) {</pre>
00038
              j = 1;
             COSA = cos(kMin*rx[n]);
SINA = sin(kMin*rx[n]);
00039
00040
00041
              for (m = 1; m <= nFunCorr; m++) {</pre>
00042
                if(m == 1) {
00043
              cosV = cos(deltaK*rx[n]);
              sinV = sin(deltaK*rx[n]);
00044
00045
              cosV0 = cosV;
               }else if(m == 2){
00046
00047
              cosV1 = cosV;
00048
              sinV1 = sinV;
00049
              cosV = 2.*cosV0*cosV1-1;
00050
              sinV = 2.*cosV0*sinV1;
              }else{
cosV2 = cosV1;
00051
00052
00053
              sinV2 = sinV1;
              cosV1 = cosV;
00054
00055
              sinV1 = sinV;
00056
              cosV = 2.*cosV0*cosV1-cosV2;
00057
              sinV = 2.*cosV0*sinV1-sinV2;
00058
                 COSV = COSA*cosV - SINA*sinV;
00059
                 SINV = SINA*cosV + COSA*sinV;
00060
00061
                 cfVal[j] += COSV;
00062
                 cfVal[j+1] += SINV;
                 j += 2;
00063
00064
00065
00066
00067
           for (nb = 1; nb <= nBuffCorr; nb++) {</pre>
00068
             indexCorr[nb] += 1;
00069
              if (indexCorr[nb] <= 0) continue;</pre>
             ini = nFunCorr * (indexCorr[nb] - 1);
if (indexCorr[nb] == 1) {
  for (j = 1; j <= 2*nFunCorr; j++)
  cfOrg[nb][j] = cfVal[j];</pre>
00070
00071
00072
00073
00074
00075
00076
              for (j = 1; j <= nFunCorr; j++)</pre>
                spacetimeCorr[nb][ni + j] = 0.;
00077
00078
00079
08000
              for (m = 1; m <= nFunCorr; m++) {</pre>
00081
                nv = m + ni;
00082
                 spacetimeCorr[nb][nv] \ += \ cfVal[j] \ \star \ cfOrg[nb][j] \ + \ cfVal[j+1] \ \star \ cfOrg[nb][j+1];
00083
                 j += 2;
00084
00085
00087
00088
           // ACCUMULATE SPACETIME CORRELATIONS
           for (nb = 1; nb <= nBuffCorr; nb++) {
  if (indexCorr[nb] == nValCorr) {</pre>
00089
00090
               for (j = 1; j <= nFunCorr*nValCorr; j++)
    spacetimeCorrAv[j] += spacetimeCorr[nb][j];</pre>
00091
00092
00093
                indexCorr[nb] = 0.;
00094
                countCorrAv ++;
               if (countCorrAv == limitCorrAv) {
00095
               for (j = 1; j <= nFunCorr*nValCorr; j++)
spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
fprintf(fpdnsty, "NDIM %d\n", NDIM);
fprintf(fpdnsty, "nAtom %d\n", nAtom);</pre>
00096
00097
00098
00099
                  fprintf(fpdnsty, "region %\lfn, region[1]);
fprintf(fpdnsty, "nFunCorr %d\n", nFunCorr);
fprintf(fpdnsty, "limitCorrAv %d\n", limitCorrAv);
fprintf(fpdnsty, "stepCorr %d\n", stepCorr);
fprintf(fpdnsty, "nValCorr %d\n", nValCorr);
00100
00101
00102
00103
00104
00105
                  fprintf(fpdnsty, "deltaT %lf\n", deltaT);
```

```
real tVal;
              for (n = 1; n <= nValCorr; n++){
tVal = (n-1)*stepCorr*deltaT;
fprintf (fpdnsty, "%e\t", tVal);</pre>
00107
00108
00109
00110
              int nn = nFunCorr*(n-1);
              for (j = 1; j <= nFunCorr; j ++)
    fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
    fprintf (fpdnsty, "\n");</pre>
00111
00112
00113
00114
00115
                countCorrAv = 0.;
00116
                  for (j = 1; j <= nFunCorr*nValCorr; j++)</pre>
00117
               spacetimeCorrAv[j] = 0.;
00118
00119
00120
00121 }
00122 }
```

References cfOrg, cfVal, countCorrAv, deltaT, fpdnsty, indexCorr, limitCorrAv, nAtom, nBuffCorr, NDIM, nFunCorr, nValCorr, region, rx, spacetimeCorr, spacetimeCorrAv, and stepCorr.

## 3.70.2.20 EvalUnwrap()

```
void EvalUnwrap ( )
```

Definition at line 27 of file EvalUnwrap.c.

```
00027 {
00028 int n;
00029 for (n = 1; n <= nAtom; n++) {
00030 rxUnwrap[n] = rx[n] + ImageX[n] * region[1];
00031 ryUnwrap[n] = ry[n] + ImageY[n] * region[2];
00032 }
00033 }
```

References ImageX, ImageY, nAtom, region, rx, rxUnwrap, ry, and ryUnwrap.

Referenced by main().

Here is the caller graph for this function:



#### 3.70.2.21 EvalVacf()

```
void EvalVacf ( )
```

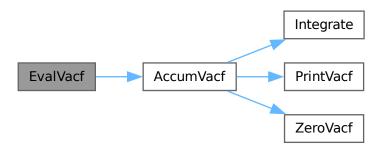
Definition at line 26 of file EvalVacf.c.

```
00026
00027
         int n, nb, ni;
00028
         double viscVec = 0.;
00029
        double v[3];
00030
        for(n = 1 ; n <= nAtom ; n ++) {</pre>
         v[1] = vx[n] - 0.5*ax[n]*deltaT;
v[2] = vy[n] - 0.5*ay[n]*deltaT;
00031
00032
          viscVec += v[1]*v[2];
00033
00034
00035
        viscVec += rfAtom;
```

```
for (nb = 1; nb <= nBuffAcf; nb ++) {
          indexAcf[nb] ++;
if(indexAcf[nb] <= 0)continue;
if(indexAcf[nb] == 1){</pre>
00037
00038
00039
             viscAcfOrg[nb] = viscVec;
00040
00041
00042
           ni = indexAcf[nb];
00043
           viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
00044
00045
         AccumVacf();
00046 }
```

References AccumVacf(), ax, ay, deltaT, indexAcf, nAtom, nBuffAcf, rfAtom, viscAcf, viscAcfOrg, vx, and vy.

Here is the call graph for this function:



## 3.70.2.22 EvalVrms()

```
void EvalVrms ( )
```

Definition at line 27 of file EvalVrms.c.

```
00027
00028
           int n;
00029
          Vsqr = 0.0;
          VMeanSqr = 0.0;
VRootMeanSqr = 0.0;
00030
00031
00032
00033
          for(n = 1 ; n <= nAtom ; n ++) {
VSqr += Sqr(vx[n]) + Sqr(vy[n]);</pre>
00035
00036
          VMeanSqr = VSqr/nAtom;
00037
          VRootMeanSqr = sqrt(VMeanSqr);
00038
```

References nAtom, Sqr, VMeanSqr, VRootMeanSqr, VSqr, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.23 HaltConditionCheck()

```
bool HaltConditionCheck (
              double value,
              int stepCount )
Definition at line 27 of file Halt.c.
00028
        if(value <= HaltCondition && value != 0) {
00029
        fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.10f\n", stepCount, value);
00030
00031
                           // Signal that the halt condition is met
        return true;
00032
00033
       return false; // Halt condition not met
00034 }
```

References fpresult, HaltCondition, and stepCount.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.24 Init()

```
void Init ( )
```

#### Definition at line 29 of file Init.c.

```
00030
            char dummy[128];
            char inputConfig[128];
FILE *fp;
00031
00032
00033
            fp = fopen("input-data", "r");
           fp = fopen("input-data", "r");
fscanf(fp, "%s %s", dummy, inputConfig);
fscanf(fp, "%s %s", dummy, solver);
fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
fscanf(fp, "%s %d", dummy, &DampFlag);
fscanf(fp, "%s %d", dummy, &freezeAtomType);
fscanf(fp, "%s %lf", dummy, &rCut);
00034
00035
00036
00037
00038
00039
            fscanf(fp, "%s %lf", dummy, &gamman);
fscanf(fp, "%s %lf", dummy, &kappa);
fscanf(fp, "%s %lf", dummy, &deltaT);
00040
00041
00042
            fscanf(fp, "%s %lf", dummy, &strain);
fscanf(fp, "%s %lf", dummy, &strain);
fscanf(fp, "%s %lf", dummy, &FyBylx);
00043
00044
            fscanf(fp, "%s %lf", dummy, &fxByfy);
00045
            fscanf(fp, "%s %lf", dummy, &DeltaY);
00046
            fscanf(fp, "%s %lf", dummy, &DeltaX);
fscanf(fp, "%s %lf", dummy, &HaltCondition);
fscanf(fp, "%s %d", dummy, &stepAvg);
00047
00048
00049
            fscanf(fp, "%s %d",
00050
                                              dummy, &stepEquil);
            fscanf(fp, "%s %d",
00051
                                              dummy, &stepLimit);
            fscanf(fp, "%s %d", fscanf(fp, "%s %d",
00052
                                               dummy, &stepDump);
00053
                                              dummy, &stepTraj);
            fscanf(fp, "%s %d", fscanf(fp, "%s %d",
00054
                                               dummy, &limitCorrAv);
00055
                                              dummy, &nBuffCorr);
            fscanf(fp, "%s %d",
00056
                                               dummy, &nFunCorr);
            fscanf(fp, "%s %d", fscanf(fp, "%s %d",
00057
                                              dummy, &nValCorr);
00058
                                              dummy, &stepCorr);
00059
            fscanf(fp, "%s %d", dummy, &limitAcfAv);
```

```
fscanf(fp, "%s %d", dummy, &nBuffAcf);
fscanf(fp, "%s %d", dummy, &nValAcf);
fscanf(fp, "%s %d", dummy, &stepAcf);
00061
00062
          fscanf(fp, "%s %lf", dummy, &rangeRdf);
00063
          fscanf(fp, "%s %d", dummy, &limitRdf);
fscanf(fp, "%s %d", dummy, &sizeHistRdf);
00064
00065
                                     dummy, &sizeHistRdf);
          fscanf(fp, "%s %d", dummy, &stepRdf);
00066
00067
          fclose(fp);
00068
00069
         FILE *fpSTATE;
          if((fpSTATE = fopen(inputConfig,"r")) ==NULL) {
00070
00071
          printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00072
          exit(0);
00073
00074
          fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
fscanf(fpSTATE, "%s %d", dummy, &nAtom);
fscanf(fpSTATE, "%s %d", dummy, &nBond);
00075
00076
00077
          fscanf(fpSTATE, "%s %d", dummy, &nAtomType);
          fscanf(fpSTATE, "%s %d", dummy, &nBondType);
fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00079
00080
00081
00082
00083
          region[2] *= 1.5: //Remove this when put on GitHub
00084
00085
          density = nAtom/(region[1]*region[2]);
          cells[1] = region[1] / rCut;
cells[2] = region[2] / rCut;
00086
00087
00088
          cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
          regionH[1] = 0.5*region[1];
00089
          regionH[2] = 0.5*region[2];
00090
00091
00092
          //strain information
00093
          strainRate = strain/deltaT;
          shearDisplacement = strain * region[2];
shearVelocity = strainRate * region[2];
00094
00095
00096
          int n;
00098
                (double*)malloc((nAtom + 1) * sizeof(double));
         ry = (double*)malloc((nAtom + 1) * sizeof(double));
vx = (double*)malloc((nAtom + 1) * sizeof(double));
00099
00100
          vy = (double*)malloc((nAtom + 1) * sizeof(double));
00101
          ax = (double*)malloc((nAtom + 1) * sizeof(double));
00102
          ay = (double*)malloc((nAtom + 1) * sizeof(double));
00103
          fax = (double*)malloc((nAtom + 1) * sizeof(double));
fay = (double*)malloc((nAtom + 1) * sizeof(double));
00104
00105
00106
          atomID = (int*)malloc((nAtom+1) * sizeof(int));
00107
          atomType = (int*)malloc((nAtom+1) * sizeof(int));
          atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00108
00109
          atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
          speed = (double*)malloc((nAtom + 1) * sizeof(double));
00110
00111
          atom1 = (int*)malloc((nBond+1)*sizeof(int));
00112
          atom2 = (int*)malloc((nBond+1)*sizeof(int));
00113
          BondID = (int*)malloc((nBond+1)*sizeof(int));
          BondType = (int*)malloc((nBond+1)*sizeof(int));
00114
          kb = (double*)malloc((nBond+1)*sizeof(double));
00115
          ro = (double*)malloc((nBond+1)*sizeof(double));
          BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00117
00118
          BondLength = (double*) malloc((nBond+1)*sizeof(double));
          discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00119
00120
          nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00121
00122
          ImageY = (int*)malloc((nAtom+1) * sizeof(int));
ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00123
00124
          rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00125
00126
          DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
00127
00128
00129
          DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00130
          DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00131
          for(int n = 0; n \le nAtom; n++) {
          DeltaXijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00132
00133
00134
00135
          molID = (int*)malloc((nAtom+1) * sizeof(int));
00136
00137
          for(n = 1; n <= nAtom; n ++) {</pre>
00138
           atomMass[n] = 1.0;
00139
00140
          fscanf(fpSTATE, "%s\n", dummy);
          for(n = 1; n <= nAtom; n ++)
fscanf(fpSTATE, "%d %d %d %lf %lf %lf %lf %lf \n", &atomID[n], &molID[n], &atomType[n],
00142
00143
       &atomRadius[n], &rx[n], &ry[n], &vx[n], &vy[n]);
00144
00145
```

```
00146
         fscanf(fpSTATE, "%s\n", dummy);
         for(n=1; n<=nBond; n++)
fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
00147
00148
      &ro[n]);
00149
00150
         fclose(fpSTATE);
00152
        //2D-List of bonded atoms. This is used to remove pair interaction
00153
        //calculation for the bonded atoms
          isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00154
         for (int i = 0; i <= nAtom; i++) {</pre>
00155
          isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
for (int j = 0; j <= nAtom; j++) {
   isBonded[i][j] = 0;</pre>
00156
00157
00158
00159
00160
00161
00162
         for (n = 1; n <= nBond; n++) {</pre>
          int i = atom1[n];
00163
            int j = atom2[n];
00164
           isBonded[i][j] = 1;
isBonded[j][i] = 1; // symmetric
00165
00166
00167 }
00168
00169
        //For thermostate, update in final version
00170 thermo = 'C';
00171
00172
00173 // List the interface atoms
00174 nAtomInterface = 0;
00175 nAtomBlock = 0;
00176 nDiscInterface = 0;
00177
        double InterfaceWidth, bigDiameter;
00178
        bigDiameter = 2.8;
00179
        InterfaceWidth = 5.0 * bigDiameter;
00180
00181 for(n = 1; n <= nAtom; n++) {
        if (fabs(ry[n]) < InterfaceWidth) {</pre>
00182
00183
         nAtomInterface++;
00184
00185
         if(molID[n] == 2){
00186
         nAtomBlock++:
00187
00188
         if (atomRadius[n] != 0.0) {
00189
         nDiscInterface++;
00190
00191
00192
          atomIDInterface = (int*)malloc((nAtomInterface+1)*sizeof(int));
00193
00194
         int m:
00195
         m = 1;
00196
         for (n=1; n<=nAtom; n++) {</pre>
00197
          if(fabs(ry[n]) < InterfaceWidth) {</pre>
00198
          atomIDInterface[m] = atomID[n];
00199
          m++;
00200
          } }
00201
00202
         nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00203
         PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
         Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00204
00205
00206
         PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00207
         PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00208
00209
            fprintf(fpresult, "----\n");
           00210
                                                                          ----\n");
00211
00212
00213
            fprintf(fpresult, "nAtomBlock\t\t\s\d\n", nAtomBlock);
fprintf(fpresult, "nAtomInterface\t\s\d\n", nAtomInter
00214
            fprintf(fpresult, "nAtomInterface\t\%d\n", nAtomInterface);
fprintf(fpresult, "nDiscInterface\t\t\%d\n", nDiscInterface);
00215
           00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
           fprintf(fpresult, "censity\t\t\t\g\n", density);
fprintf(fpresult, "rCut\t\t\t\g\n", rCut);
fprintf(fpresult, "deltaT\t\t\t\g\n", deltaT);
fprintf(fpresult, "stepEquil\t\t\t\g\n", stepEquil);
fprintf(fpresult, "stepLimit\t\t\t\g\n", stepLimit);
fprintf(fpresult, "region[1]\t\t\t\s\0.161\n", region[1]);
00227
00228
00229
00230
00231
```

```
00232
                 fprintf(fpresult, "region[2]\t\t\t\0.16lf\n", region[2]);
                fprintf(fpresult, region[2]\t\t\t\sd\n", cells[1]);
fprintf(fpresult, "cells[1]\t\t\t\sd\n", cells[1]);
fprintf(fpresult, "cells[2]\t\t\t\sd\n", cells[2]);
fprintf(fpresult, "solver\t\t\t\s\n", solver);
fprintf(fpresult, "boundary\t\t\t\s\s\n", xBoundary, yBoundary);
fprintf(fpresult, "DampFlag\t\t\sd\n", DampFlag);
00233
00234
00235
00236
00237
00239
                 00240
00241
         PairEnergyPerAtom BondEnergyPerAtom Press VirialSum\n");
fprintf(fpvrms, "#timeNow\tVrms \n");
fprintf(fpcom, "#timeNow\tComX\tComY\n");
00242
00243
00244
00245 /\star // Uncomment the following as per your acquirement
                 fprintf(fpstress, "strain
fprintf(fpstress, "region[1]
                                                                         %lf\n", strain);
%lf\n", region[1]);
%lf\n", region[2]);
00246
00247
                 fprintf(fpstress, "region[1] %11\n', region[1]);
fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00248
                 fprintf(fpmomentum, "#timeNow Px Py\n");
00250
00251 */
00252
        if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
  (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
  fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
  allowed.\n", xBoundary, yBoundary);
00253
00254
00255
00256
                exit(EXIT_FAILURE); // Exit with failure status
00257
00258
00259 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, BondEnergy, BondID, BondLength, BondType, cellList, cells, DampFlag, deltaT, DeltaX, DeltaXijOld, DeltaXijOldPair, DeltaY, DeltaYijOld, DeltaYijOldPair, density, discDragx, discDragy, fax, fay, fpcom, fpresult, fpvrms, freezeAtomType, fxByfy, FyBylx, gamman, HaltCondition, ImageX, ImageY, isBonded, kappa, kb, limitAcfAv, limitCorrAv, limitRdf, molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nBuffAcf, nBuffCorr, nDiscInterface, nFunCorr, nodeDragx, nodeDragy, nPairTotal, nValAcf, nValCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rangeRdf, rCut, region, regionH, ro, rx, rxUnwrap, ry, ryUnwrap, shearDisplacement, shearVelocity, sizeHistRdf, solver, speed, stepAcf, stepAvg, stepCorr, stepDump, stepEquil, stepLimit, stepRdf, stepTraj, strain, strainRate, thermo, timeNow, vx, vy, xBoundary, and yBoundary.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.25 LeapfrogStep()

```
void LeapfrogStep ( )
```

### 3.70.2.26 main()

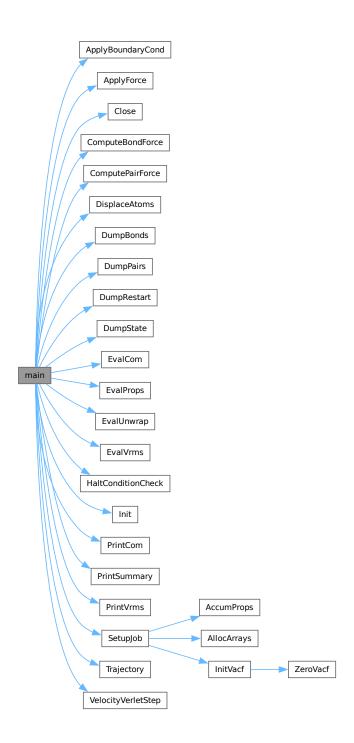
```
int main (
          int argc,
          char ** argv )
```

```
Definition at line 51 of file main.c.
00052
         time_t t1 = 0, t2;
         if (argc < 2) {
00053
         fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
00054
00055
00057
           int prefix_size = snprintf(NULL, 0, "../output/ss", argv[1]) + 1; // +1 for the null terminator
00058
          prefix = malloc(prefix_size);
           if(prefix == NULL) {
  fprintf(stderr, "Memory allocation failed\n");
00059
00060
00061
            return 1:
00062
00063
         // Write the formatted string into the allocated space
snprintf(prefix, prefix_size, "../output/%s", argv[1]);
sprintf(result, "%s.result", prefix);
fpresult = fopen(result, "w");
sprintf(xyz, "%s.xyz", prefix);
fpxyz = fopen(xyz, "w");
sprintf(vrms, "%s.vrms", prefix);
fpvrms = fopen(vrms, "w");
sprintf(bond, "%s.bond", prefix);
fpbond = fopen(bond, "w");
sprintf(com, "%s.com", prefix);
fpcom = fopen(com, "w");
sprintf(pair, "%s.pair", prefix);
fppair = fopen(pair, "w");
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
          /\star //Uncomment the following as per your acquirement
          sprintf(dnsty, "%s.curr-dnsty", prefix);
08000
00081
          fpdnsty = fopen(dnsty, "w");
00082
          sprintf(visc, "%s.viscosity", prefix);
          fpvisc = fopen(visc, "w");
sprintf(rdf, "%s.rdf", prefix);
fprdf = fopen(rdf, "w");
sprintf(stress, "%s.stress", prefix);
fpstress = fopen(stress, "w");
00083
00084
00085
00086
          sprintf(momentum, "%s.momentum", prefix);
fpmomentum = fopen(momentum, "w");
00088
00089
00090
00091
00092
          Init();
00093
          SetupJob();
00094
          t1 = time(NULL);
00095
          moreCycles = 1;
00096
          timeNow = 0.0;
00097
          if(timeNow == 0.0) {
           DisplaceAtoms();
00098
00099
            ComputePairForce(1);
            ComputeBondForce();
00100
00101
            ApplyForce();
00102
            DumpBonds();
00103
            DumpPairs();
00104
            Trajectory();
00105
            EvalUnwrap();
            ApplyBoundaryCond();
00107
            EvalProps();
00108
            EvalVrms();
00109
            EvalCom();
00110
            PrintVrms():
00111
            PrintCom();
00112
            PrintSummary();
00113
00114
00115 //Here starts the main loop of the program
00116
          while (moreCycles) {
           if(stepLimit == 0) {
00117
00118
           exit(0);
00119
00120
00121
            stepCount ++;
00122
            timeNow = stepCount * deltaT; //for adaptive step size: timeNow += deltaT
00123
00124
            VelocityVerletStep(1);
            EvalUnwrap();
00125
00126
            ApplyBoundaryCond();
00127
            ComputePairForce(1);
00128
            ComputeBondForce();
            ApplyForce();
00129
            VelocityVerletStep(2);
00130
            ApplyBoundaryCond();
00131
00132
            EvalProps();
00133
            EvalVrms();
00134
            EvalCom();
            if(stepCount % stepAvg == 0){
00135
             PrintSummary();
00136
```

```
00137
         PrintVrms();
00138
         PrintCom();
00139
00140
        if(stepCount % stepTraj == 0){
00141
         Trajectory();
00142
         DumpBonds();
         DumpPairs();
00144
00145
        if(stepCount % stepDump == 0){
         00146
00147
00148
        if(HaltConditionCheck(VRootMeanSqr, stepCount)) {
00149
00150
         DumpRestart();  // Save the current state for input
                            // Save the current state for config
00151
         DumpState();
00152
         break; // Exit the loop when the halt condition is met
00153
00154
00155
         if(stepCount >= stepLimit)
00156
          moreCycles = 0;
00157
00158
00159
       t2 = time(NULL);
00160
00161
       fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
       fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));
00162
00163
00164
       fclose(fpresult);
00165
       fclose(fpxyz);
00166
       fclose(fpvrms);
00167
       fclose(fpbond);
00168
       fclose(fppair);
00169
       fclose(fpcom);
00170
00171 /\star//\text{Uncomment} the following as per your acquirement
00172
       fclose(fpdnsty);
00173
       fclose(fpvisc);
       fclose(fprdf);
00175
       fclose(fpstress);
00176
     fclose(fpmomentum);
00177 */
00178
00179
       free (prefix);
00180
       Close();
00181
       return 0;
00182 }
```

References ApplyBoundaryCond(), ApplyForce(), bond, Close(), com, ComputeBondForce(), ComputePairForce(), deltaT, DisplaceAtoms(), DumpBonds(), DumpPairs(), DumpRestart(), DumpState(), EvalCom(), EvalProps(), EvalUnwrap(), EvalVrms(), fpbond, fpcom, fppair, fpresult, fpvrms, fpxyz, HaltConditionCheck(), Init(), moreCycles, pair, prefix, PrintCom(), PrintSummary(), PrintVrms(), result, SetupJob(), stepAvg, stepCount, stepDump, stepLimit, stepTraj, timeNow, Trajectory(), VelocityVerletStep(), vrms, VRootMeanSqr, and xyz.

Here is the call graph for this function:



# 3.70.2.27 PrintCom()

void PrintCom ( )

Definition at line 28 of file PrintCom.c. 00028

```
00029 fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY); 00030 fflush(fpcom); 00031 }
```

References ComX, ComY, fpcom, and timeNow.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.28 PrintMomentum()

```
void PrintMomentum ( )
```

Definition at line 25 of file PrintMomentum.c.

```
00025 { 00026 \quad \text{fprintf(fpmomentum, "%0.41f}\t%0.161f}\t%0.161f\n", timeNow, vSumX, vSumY); \\ 00027 \quad \text{fflush(fpmomentum);} \\ 00028 \ \}
```

References fpmomentum, timeNow, vSumX, and vSumY.

## 3.70.2.29 PrintStress()

```
void PrintStress ( )
```

Definition at line 25 of file PrintStress.c.

```
00025 {
00026 fprintf(fpstress, "%0.41f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\n", timeNow, virSumxx, virSumyy, virSumxy, pressure);
00027 fflush(fpstress);
00028 }
```

References fpstress, pressure, timeNow, virSumxx, virSumxy, and virSumyy.

## 3.70.2.30 PrintSummary()

```
void PrintSummary ( )
```

Definition at line 4 of file PrintSummary.c.

```
00005 fprintf(fpresult, "%0.41f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161
```

References BondEnergyPerAtom, fpresult, kinEnergy, potEnergy, pressure, timeNow, totEnergy, uSumPairPerAtom, virSum, and vSum.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.31 PrintVrms()

```
void PrintVrms ( )
```

Definition at line 27 of file PrintVrms.c.

References fpvrms, timeNow, and VRootMeanSqr.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.32 SetupJob()

```
void SetupJob ( )
```

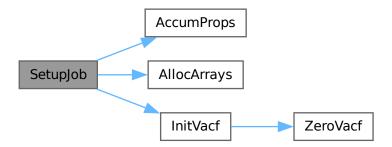
Definition at line 27 of file SetupJob.c.

```
00028
          AllocArrays();
00029
          AccumProps(0);
         InitVacf();
stepCount = 0;
// INITIALISE SPACETIME CORRELATIONS
00030
00031
00032
00033
          int n;
         for (n = 1; n <= nBuffCorr; n++)
  indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;</pre>
00034
00035
00036
00037
         countCorrAv = 0.;
00038
00039
         for (n = 1; n <= nFunCorr*nValCorr; n++)</pre>
00040
            spacetimeCorrAv[n] = 0.;
00041
00042
          //RDF
          countRdf = 0;
00043
00044 }
```

References AccumProps(), AllocArrays(), countCorrAv, countRdf, indexCorr, InitVacf(), nBuffCorr, nFunCorr, nValCorr, spacetimeCorrAv, and stepCount.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.70.2.33 Trajectory()

```
Definition at line 25 of file Trajectory.c.

Definition at line 25 of file 25 of file Trajectory.c.

Definition at line 25 of file Trajectory.c.

Definition at line 25 of file 25 of fi
```

References atomID, atomRadius, atomType, ax, ay, fpxyz, moIID, nAtom, regionH, rx, ry, timeNow, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



## 3.70.2.34 VelocityVerletStep()

## Definition at line 26 of file VelocityVerletStep.c.

```
00026
00027 int n;
00028 if(icode == 1){
        for (n= 1; n <= nAtom; n++) {</pre>
00030
         if(atomType[n] != freezeAtomType) {
         vx[n] += ax[n] * 0.5 * deltaT;
vy[n] += ay[n] * 0.5 * deltaT;
rx[n] += vx[n] * deltaT;
00031
00032
00033
00034
          ry[n] += vy[n] * deltaT;
00035
00036
          //Calculating the image flags here
         if (rx[n] >= regionH[1]) {
  rx[n] -= region[1];
00037
00038
          ImageX[n]++;
} else if (rx[n] < -regionH[1]) {
rx[n] += region[1];</pre>
00039
00040
00041
00042
          ImageX[n]--;
00043
00044
          if (ry[n] \ge regionH[2]) {
00045
           ry[n] -= region[2];
           ImageY[n]++;
00046
00047
           } else if (ry[n] < -regionH[2]) {</pre>
00048
           ry[n] += region[2];
```

References atomType, ax, ay, deltaT, freezeAtomType, ImageX, ImageY, nAtom, region, regionH, rx, ry, vx, and vv.

Referenced by main().

Here is the caller graph for this function:



## 3.70.3 Variable Documentation

#### 3.70.3.1 prefix

```
char* prefix = NULL
```

Definition at line 13 of file main.c.

Referenced by DumpRestart(), DumpState(), and main().

# 3.71 main.c

# Go to the documentation of this file.

```
00001 #include<stdio.h>
00002 #include<string.h>
00003 #include<stdlib.h>
00004 #include <stdbool.h>
00005 #include <time.h>
00006 #include <mpi.h>
00007 #define DEFINE_GLOBALS
00008 #include "global.h"
00009 #include "ComputeBondForce.h"
00010 #include "ComputePairForce.h"
00011
00012
00013 char *prefix = NULL; // Definition of prefix
00014
00015 void Init();
00016 void SetupJob();
00017 void EvalSpacetimeCorr();
00018 void Trajectory();
00019 void DumpState();
00020 void ComputeForcesCells();
00021 void LeapfrogStep();
00022 void BrownianStep();
00023 void ApplyBoundaryCond();
00024 void EvalProps();
00025 void EvalVacf();
```

3.71 main.c 163

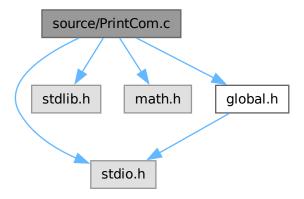
```
00026 void EvalRdf();
00027 void AccumProps(int icode);
00028 void PrintSummary();
00029 void PrintVrms();
00030 //void ComputeBondForce();
00031 void DumpBonds();
00032 void VelocityVerletStep(int icode);
00033 void ApplyForce();
00034 void ApplyDrivingForce();
00035 void ApplyShear();
00036 void ApplyLeesEdwardsBoundaryCond();
00037 void PrintStress();
00038 void Close();
00039 //void ComputePairForce(int normFlag);
00040 void PrintMomentum();
00041 void DisplaceAtoms();
00042 void DumpRestart();
00043 bool HaltConditionCheck(double value, int stepCount);
00044 void EvalCom();
00045 void PrintCom();
00046 void EvalVrms();
00047 void EvalUnwrap();
00048 void DumpPairs();
00049 void ApplyViscous();
00050
00051 int main(int argc, char **argv) {
00052 time_t t1 = 0, t2;
00053
         if (argc < 2) {</pre>
00054 fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
00055
         return 1:
00056 }
00057
          int prefix_size = snprintf(NULL, 0, "../output/%s", argv[1]) + 1; // +1 for the null terminator
00058
          prefix = malloc(prefix_size);
00059
           if(prefix == NULL) {
00060
           fprintf(stderr, "Memory allocation failed\n");
00061
            return 1:
00062
00064
            // Write the formatted string into the allocated space
          // Write the formatted string into the allocated space
snprintf(prefix, prefix_size, "../output/%s", argv[1]);
sprintf(result, "%s.result", prefix);
fpresult = fopen(result, "w");
sprintf(xyz, "%s.xyz", prefix);
fpxyz = fopen(xyz, "w");
sprintf(vrms, "%s.vrms", prefix);
fpvrms = fopen(vrms, "w");
sprintf(bond, "%s.bond", prefix);
00065
00066
00067
00068
00069
00070
00071
          sprintf(bond, "%s.bond", prefix);
fpbond = fopen(bond, "w");
sprintf(com, "%s.com", prefix);
00072
00073
00074
          fpcom = fopen(com, "w");
00075
          sprintf(pair, "%s.pair", prefix);
fppair = fopen(pair, "w");
00076
00077
00078
          /\star //Uncomment the following as per your acquirement sprintf(dnsty, "%s.curr-dnsty", prefix);
00079
00080
          fpdnsty = fopen(dnsty, "w");
00081
          sprintf(visc, "%s.viscosity", prefix);
00082
          sprint(visc, "%s.viscosity", prefix);
fpvisc = fopen(visc, "w");
sprintf(rdf, "%s.rdf", prefix);
fprdf = fopen(rdf, "w");
sprintf(stress, "%s.stress", prefix);
fpstress = fopen(stress, "w");
sprintf(momentum, "%s.momentum", prefix);
00083
00084
00085
00086
00087
00088
00089
          fpmomentum = fopen(momentum, "w");
00090
00091
00092
          Init();
          SetupJob();
t1 = time(NULL);
00093
00094
00095
          moreCycles = 1;
00096
          timeNow = 0.0;
00097
          if(timeNow == 0.0) {
           DisplaceAtoms();
00098
00099
            ComputePairForce(1);
00100
            ComputeBondForce();
00101
            ApplyForce();
00102
            DumpBonds();
00103
            DumpPairs();
00104
            Trajectory();
            EvalUnwrap();
00105
00106
            ApplyBoundaryCond();
00107
            EvalProps();
00108
            EvalVrms();
00109
            EvalCom();
            PrintVrms();
00110
00111
            PrintCom();
00112
            PrintSummarv():
```

```
00113
00114
00115 //Here starts the main loop of the program
00116
        while (moreCycles) {
00117
        if(stepLimit == 0){
00118
         exit(0);
00119
00120
00121
         stepCount ++;
         timeNow = stepCount * deltaT; //for adaptive step size: timeNow += deltaT
00122
00123
00124
         VelocityVerletStep(1):
00125
         EvalUnwrap();
00126
         ApplyBoundaryCond();
00127
         ComputePairForce(1);
00128
         ComputeBondForce();
00129
         ApplyForce();
         VelocityVerletStep(2);
00130
         ApplyBoundaryCond();
00131
00132
         EvalProps();
00133
         EvalVrms();
00134
         EvalCom();
         if(stepCount % stepAvg == 0) {
00135
00136
          PrintSummary();
00137
          PrintVrms();
00138
          PrintCom();
00139
00140
         if(stepCount % stepTraj == 0){
00141
          Trajectory();
00142
          DumpBonds();
00143
          DumpPairs();
00144
00145
         if(stepCount % stepDump == 0){
          00146
00147
00148
00149
         if(HaltConditionCheck(VRootMeanSqr, stepCount)) {
          DumpRestart();  // Save the current state for input
00150
00151
          DumpState();
                              // Save the current state for config
00152
          break; // Exit the loop when the halt condition is met
00153
00154
          if(stepCount >= stepLimit)
moreCycles = 0;
00155
00156
00157
00158
00159
       t2 = time(NULL);
00160
       fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));
00161
00162
00163
00164
00165
        fclose(fpxyz);
00166
       fclose(fpvrms);
00167
        fclose(fpbond);
00168
        fclose(fppair);
00169
        fclose(fpcom);
00170
00171 / \star / / \text{Uncomment} the following as per your acquirement
00172 fclose(fpdnsty);
00173
       fclose(fpvisc);
00174
       fclose(fprdf);
00175
       fclose(fpstress);
00176
       fclose(fpmomentum);
00177 */
00178
00179
       free (prefix);
00180
       Close();
00181
       return 0:
00182 }
```

# 3.72 source/PrintCom.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for PrintCom.c:



# **Functions**

• void PrintCom ()

# 3.72.1 Function Documentation

## 3.72.1.1 PrintCom()

```
void PrintCom ( )
```

Definition at line 28 of file PrintCom.c.

```
00028 {
00029 fprintf(fpcom, "%0.41f\t%0.161f\t%0.161f\n", timeNow, ComX, ComY);
00030 fflush(fpcom);
00031 }
00031
```

References ComX, ComY, fpcom, and timeNow.

Referenced by main().

Here is the caller graph for this function:



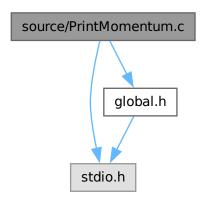
# 3.73 PrintCom.c

## Go to the documentation of this file.

```
00002
           \star This file is part of Lamina.
00003
00004
           \star Lamina is free software: you can redistribute it and/or modify \star it under the terms of the GNU General Public License as published by
00005
00006
           * the Free Software Foundation, either version 3 of the License, or
00007
           * (at your option) any later version.
80000
00000 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 * GNU General Public License for more details.
00013 *
00014 \,\, You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022
00023 #include<stdio.h>
00024 #include<stdlib.h>
00025 #include<math.h>
00026 #include"global.h"
00027
00028 void PrintCom() {
00029 fprintf(fpcom, "%0.41f\t%0.161f\t%0.161f\n", timeNow, ComX, ComY);
00030 fflush(fpcom);
00032
00033
00034
```

# 3.74 source/PrintMomentum.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for PrintMomentum.c:
```



# **Functions**

• void PrintMomentum ()

3.75 PrintMomentum.c 167

#### 3.74.1 Function Documentation

#### 3.74.1.1 PrintMomentum()

References fpmomentum, timeNow, vSumX, and vSumY.

# 3.75 PrintMomentum.c

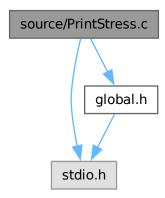
#### Go to the documentation of this file.

```
00002 * This file is part of Lamina.
00003
         * Lamina is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
00004
00005
00006 \star the Free Software Foundation, either version 3 of the License, or
00007
         \star (at your option) any later version.
00008 *
00009 * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details.
00013 *
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void PrintMomentum() {
00026 fprintf(fpmomentum, "%0.41f\t%0.161f\t%0.161f\n", timeNow, vSumX, vSumY);
00027 fflush(fpmomentum);
00028 }
```

## 3.76 source/PrintStress.c File Reference

```
#include <stdio.h>
#include "global.h"
```

Include dependency graph for PrintStress.c:



#### **Functions**

• void PrintStress ()

#### 3.76.1 Function Documentation

# 3.76.1.1 PrintStress()

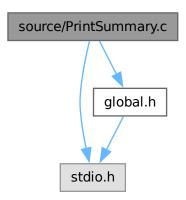
References fpstress, pressure, timeNow, virSumxx, virSumxy, and virSumyy.

#### 3.77 PrintStress.c

```
00002 * This file is part of Lamina.
00003
00004
          \star Lamina is free software: you can redistribute it and/or modify
          \star it under the terms of the GNU General Public License as published by
00005
          * the Free Software Foundation, either version 3 of the License, or
00006
00007
          \star (at your option) any later version.
80000
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 00012 * GNU General Public License for more details
00009
         * Lamina is distributed in the hope that it will be useful,
00013 *
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
```

# 3.78 source/PrintSummary.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for PrintSummary.c:
```



## **Functions**

• void PrintSummary ()

#### 3.78.1 Function Documentation

#### 3.78.1.1 PrintSummary()

00008 }

References BondEnergyPerAtom, fpresult, kinEnergy, potEnergy, pressure, timeNow, totEnergy, uSumPairPerAtom, virSum, and vSum.

Referenced by main().

Here is the caller graph for this function:



# 3.79 PrintSummary.c

Go to the documentation of this file.

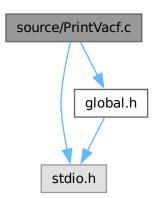
```
00001 #include<stdio.h>
00002 #include"global.h"

00003

00004 void PrintSummary() {
00005 fprintf(fpresult, "%0.41f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0.161f\t%0
```

# 3.80 source/PrintVacf.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for PrintVacf.c:
```



3.81 PrintVacf.c 171

#### **Functions**

void PrintVacf ()

# 3.80.1 Function Documentation

#### 3.80.1.1 PrintVacf()

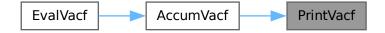
```
void PrintVacf ( )
```

#### Definition at line 25 of file PrintVacf.c.

References deltaT, fpvisc, nValAcf, stepAcf, viscAcfAv, and viscAcfInt.

Referenced by AccumVacf().

Here is the caller graph for this function:



# 3.81 PrintVacf.c

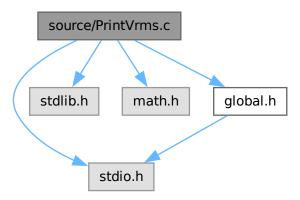
```
00001 /*
00002
        * This file is part of Lamina.
00003
00004
        * Lamina is free software: you can redistribute it and/or modify
00005
        \star it under the terms of the GNU General Public License as published by
00006
        \star the Free Software Foundation, either version 3 of the License, or
00007
        \star (at your option) any later version.
80000
00009
        * Lamina is distributed in the hope that it will be useful,
        * but WITHOUT ANY WARRANTY; without even the implied warranty of
00010
00011
        * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        \star GNU General Public License for more details.
00013
00014 \,\, You should have received a copy of the GNU General Public License 00015 \,\, * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
```

```
00024
00025 void PrintVacf(){
00026    double tVal;
00027    int j;
00028    fprintf(fpvisc, "viscosity acf\n");
00029    for(j = 1 ; j <= nValAcf ; j ++){
00030         tVal = (j-1)*stepAcf*deltaT;
00031    fprintf(fpvisc, "%lf\t %lf\t %lf\n", tVal, viscAcfAv[j], viscAcfAv[j]/viscAcfAv[l]);
00032    }
00033    fprintf(fpvisc, "viscosity acf integral : %lf\n", viscAcfInt);
00034 }
00035
00036</pre>
```

# 3.82 source/PrintVrms.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for PrintVrms.c:



## **Functions**

• void PrintVrms ()

#### 3.82.1 Function Documentation

## 3.82.1.1 PrintVrms()

```
void PrintVrms ( )

Definition at line 27 of file PrintVrms.c.
00027
00028 fprintf(fpvrms, "%0.41f\t%0.161f\n", timeNow, VRootMeanSqr);
00029 fflush(fpvrms);
00030 }
```

3.83 PrintVrms.c 173

References fpvrms, timeNow, and VRootMeanSqr.

Referenced by main().

Here is the caller graph for this function:



## 3.83 PrintVrms.c

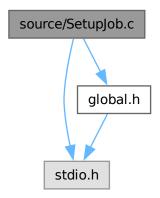
#### Go to the documentation of this file.

```
\star This file is part of Lamina.
00002
00003
        * Lamina is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
00004
00005
        * the Free Software Foundation, either version 3 of the License, or
00007
         * (at your option) any later version.
00008 *
00009 \,\star\, Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
        * GNU General Public License for more details.
00013 *
00014 \,\,\star\,\, You should have received a copy of the GNU General Public License
00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void PrintVrms() {
00028 fprintf(fpvrms, "%0.41f\t%0.161f\n", timeNow, VRootMeanSqr);
00029 fflush(fpvrms);
00030 }
00031
00032
00033
```

# 3.84 source/SetupJob.c File Reference

```
#include <stdio.h>
#include "global.h"
```

Include dependency graph for SetupJob.c:



#### **Functions**

- void AllocArrays ()
- void AccumProps (int icode)
- · void InitVacf ()
- void SetupJob ()

#### 3.84.1 Function Documentation

#### 3.84.1.1 AccumProps()

```
void AccumProps ( int\ \textit{icode}\ )
```

#### Definition at line 25 of file AccumProps.c.

```
00025
if(icode == 0) {
00026    if(icode == 0) {
00027         sPotEnergy = ssPotEnergy = 0.;
00028         sKinEnergy = ssKinEnergy = 0.;
00029         sPressure = ssPressure = 0.;
00030         sTotEnergy = ssTotEnergy = 0.;
00031         svirSum = 0.;
00032         }else    if(icode == 1) {
00033         sPotEnergy += potEnergy;
00034         sPotEnergy += Sqr(potEnergy);
00035         sKinEnergy += Sqr(kinEnergy);
00036         ssKinEnergy += Sqr(kinEnergy);
00037         stotEnergy += Sqr(totEnergy);
00038         ssTotEnergy += Sqr(totEnergy);
00040         spressure += pressure;
00040         ssPressure += Sqr(pressure);
00041         svirSum += virSum;
00042         }else    if(icode == 2) {
00043         spotEnergy /= stepAvg;
00044         ssPotEnergy /= stepAvg;
00045         stotEnergy /= stepAvg;
00046         ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047         sKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00048         srotEnergy = stepAvg;
00049         spressure /= stepAvg;
00050         ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051         svirSum /= stepAvg;
00051
```

```
00052 } }
```

References kinEnergy, potEnergy, pressure, sKinEnergy, sPotEnergy, sPressure, Sqr, ssKinEnergy, ssPotEnergy, ssPressure, ssTotEnergy, stepAvg, sTotEnergy, svirSum, totEnergy, and virSum.

Referenced by SetupJob().

Here is the caller graph for this function:



#### 3.84.1.2 AllocArrays()

```
void AllocArrays ( )
```

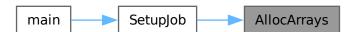
Definition at line 25 of file AllocArrays.c.

```
00026
00027
        // SPACETIME CORRELATIONS
00028 cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00029 for (n = 0; n <= nBuffCorr; n++)</pre>
        cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00031
00032 cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00033
        indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00034
00035 spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00036 for (n = 0; n <= nBuffCorr; n++)
00037 spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00038
00039 spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00040 // VISCOSITY
00041 indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00042 viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00043 for (n = 0 ; n \le nBuffAcf ; n ++)
00044
        viscAcf[n] = (double *) malloc((nValAcf+1)*sizeof(double));
00045
00046  viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00047  viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00048
00050
         histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
00051 }
```

References cfOrg, cfVal, histRdf, indexAcf, indexCorr, nBuffAcf, nBuffCorr, nFunCorr, nValAcf, nValCorr, sizeHistRdf, spacetimeCorr, spacetimeCorrAv, viscAcf, viscAcfAv, and viscAcfOrg.

Referenced by SetupJob().

Here is the caller graph for this function:



#### 3.84.1.3 InitVacf()

```
void InitVacf ( )
```

Definition at line 26 of file InitVacf.c.

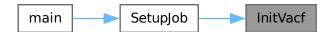
References indexAcf, nBuffAcf, nValAcf, and ZeroVacf().

Referenced by SetupJob().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.84.1.4 SetupJob()

```
void SetupJob ( )
```

Definition at line 27 of file SetupJob.c.

```
00027
00028
          AllocArrays();
00029
          AccumProps(0);
          InitVacf();
stepCount = 0;
00030
00031
          // INITIALISE SPACETIME CORRELATIONS
00032
00033
          int n;
          for (n = 1; n <= nBuffCorr; n++)
  indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;</pre>
00034
00035
00036
00037
00038
00039
          for (n = 1; n <= nFunCorr*nValCorr; n++)
    spacetimeCorrAv[n] = 0.;</pre>
00040
00041
00042
```

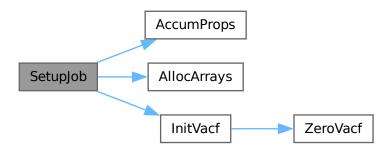
3.85 SetupJob.c 177

```
00043 countRdf = 0;
00044 }
```

References AccumProps(), AllocArrays(), countCorrAv, countRdf, indexCorr, InitVacf(), nBuffCorr, nFunCorr, nValCorr, spacetimeCorrAv, and stepCount.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



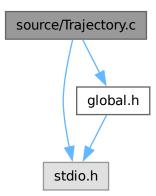
# 3.85 SetupJob.c

```
00001 /*
00002
       * This file is part of Lamina.
00003
00004
        \star Lamina is free software: you can redistribute it and/or modify
        * it under the terms of the GNU General Public License as published by
00005
        * the Free Software Foundation, either version 3 of the License, or
00006
00007
        * (at your option) any later version.
80000
00009
       \star Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of 00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
        * GNU General Public License for more details.
00013
00014
       \star You should have received a copy of the GNU General Public License
       * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00015
00016
        Copyright (C) 2025 Harish Charan, University of Durham, UK
00017
00018
00019
```

```
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 void AllocArrays();
00025 void AccumProps(int icode);
00026 void InitVacf();
00027 void SetupJob(){
00028 AllocArrays();
00029 AccumProps(0);
         AccumProps(0);
InitVacf();
stepCount = 0;
00030
00031
00032
          // INITIALISE SPACETIME CORRELATIONS
00033
         for (n = 1; n <= nBuffCorr; n++)
  indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;</pre>
00034
00035
00036
00037
         countCorrAv = 0.;
00038
00039
         for (n = 1; n <= nFunCorr*nValCorr; n++)</pre>
00040
           spacetimeCorrAv[n] = 0.;
00041
00042
          //RDF
00043
         countRdf = 0;
00044 }
```

# 3.86 source/Trajectory.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for Trajectory.c:
```



#### **Functions**

• void Trajectory ()

#### 3.86.1 Function Documentation

## 3.86.1.1 Trajectory()

void Trajectory ( )

3.87 Trajectory.c 179

Definition at line 25 of file Trajectory.c.

References atomID, atomRadius, atomType, ax, ay, fpxyz, molID, nAtom, regionH, rx, ry, timeNow, vx, and vy.

Referenced by main().

Here is the caller graph for this function:



# 3.87 Trajectory.c

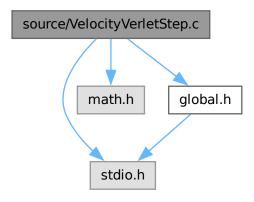
```
00001 /*
00002
           * This file is part of Lamina.
00003
00004
           * Lamina is free software: you can redistribute it and/or modify
00005
           \star it under the terms of the GNU General Public License as published by
00006
           \star the Free Software Foundation, either version 3 of the License, or
00007
           \star (at your option) any later version.
80000
00009
           * Lamina is distributed in the hope that it will be useful,
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
           * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 \,\star\, GNU General Public License for more details.
00013
00014 * You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00017
           Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void Trajectory(){
00026 int n; $^{\circ}$ 00027 //Trajectory file in LAMMPS dump format for OVITO visualization
//Trajectory file in LAMMPS dump format for OVITO visualization
00028 fprintf(fpxyz, "ITEM: TIMESTEP\n");
00029 fprintf(fpxyz, "%lf\n",timeNow);
00030 fprintf(fpxyz, "ITEM: NUMBER OF ATOMS\n");
00031 fprintf(fpxyz, "%d\n",nAtom);
00032 fprintf(fpxyz, "ITEM: BOX BOUNDS pp ff pp\n");
00033 fprintf(fpxyz, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034 fprintf(fpxyz, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035 fprintf(fpxyz, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035 fprintf(fpxyz, "%lf %lf zlo zhi\n", -0.1, 0.1);
```

```
00036    fprintf(fpxyz, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00037    for(n=1; n<=nAtom; n++)
00038         fprintf(fpxyz, "%d\t %d\t %d\t %0.21f\t %0.161f\t %0.161f
```

# 3.88 source/VelocityVerletStep.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for VelocityVerletStep.c:



#### **Functions**

void VelocityVerletStep (int icode)

## 3.88.1 Function Documentation

#### 3.88.1.1 VelocityVerletStep()

## Definition at line 26 of file VelocityVerletStep.c.

```
00026 {
00027 int n;
00028 if(icode == 1) {
00029 for (n= 1; n <= nAtom; n++) {
00030 if(atomType[n] != freezeAtomType) {
00031 vx[n] += ax[n] * 0.5 * deltaT;
00032 vy[n] += ay[n] * 0.5 * deltaT;
00033 rx[n] += vx[n] * deltaT;
00034 ry[n] += vy[n] * deltaT;
```

```
00035
00036
         //Calculating the image flags here
00037
        if (rx[n] >= regionH[1]) {
         rx[n] -= region[1];
00038
         ImageX[n]++;
} else if (rx[n] < -regionH[1]) {
rx[n] += region[1];</pre>
00039
00040
00041
00042
        ImageX[n]--;
00043
        if (ry[n] >= regionH[2]) {
00044
         ry[n] -= region[2];
00045
         ImageY[n]++;
00046
00047
         } else if (ry[n] < -regionH[2]) {</pre>
00048
         ry[n] += region[2];
00049
         ImageY[n]--;
00050
        else if(icode == 2){
00051
        for (n = 1; n <= nAtom; n++) {</pre>
00052
        if (atomType[n] != freezeAtomType) {
         vx[n] += ax[n] * 0.5 * deltaT;
00055
         vy[n] += ay[n] * 0.5 * deltaT;
00056 } } }
```

References atomType, ax, ay, deltaT, freezeAtomType, ImageX, ImageY, nAtom, region, regionH, rx, ry, vx, and vv.

Referenced by main().

Here is the caller graph for this function:



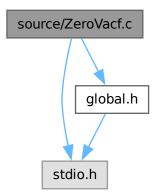
# 3.89 VelocityVerletStep.c

```
00001 /*
00002 * This file is part of Lamina.
00004
       * Lamina is free software: you can redistribute it and/or modify
00005
        \star it under the terms of the GNU General Public License as published by
00006
        \star the Free Software Foundation, either version 3 of the License, or
00007
        \star (at your option) any later version.
80000
       * Lamina is distributed in the hope that it will be useful,
00009
00010 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012
       * GNU General Public License for more details.
00013 *
* You should have received a copy of the GNU General Public License 00015 * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
        Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void VelocityVerletStep(int icode){
00027 int n;
00028 if(icode == 1){
00029 for (n= 1; n <= nAtom; n++) {
```

```
if(atomType[n] != freezeAtomType){
              ractions | - race |
vx[n] += ax[n] * 0.5 * deltaT;
vy[n] += ay[n] * 0.5 * deltaT;
rx[n] += vx[n] * deltaT;
ry[n] += vy[n] * deltaT;
00031
00032
00033
00034
00035
              //Calculating the image flags here
if (rx[n] >= regionH[1]) {
00036
00037
00038
               rx[n] -= region[1];
               ImageX[n]++;
} else if (rx[n] < -regionH[1]) {
rx[n] += region[1];</pre>
00039
00040
00041
00042
              ImageX[n]--;
00043
              if (ry[n] >= regionH[2]) {
00044
              ry[n] -= region[2];
ry[n] -= region[2];
ImageY[n]++;
} else if (ry[n] < -regionH[2]) {
ry[n] += region[2];</pre>
00045
00046
00047
00048
00049
                ImageY[n]--;
              } } }
else if(icode == 2){
00050
00051
             for(n = 1; n <= nAtom; n++) {
  if(atomType[n] != freezeAtomType) {
    vx[n] += ax[n] * 0.5 * deltaT;
    vy[n] += ay[n] * 0.5 * deltaT;</pre>
00052
00053
00054
00056 } } }
00057
```

# 3.90 source/ZeroVacf.c File Reference

```
#include <stdio.h>
#include "global.h"
Include dependency graph for ZeroVacf.c:
```



#### **Functions**

• void ZeroVacf ()

3.91 ZeroVacf.c 183

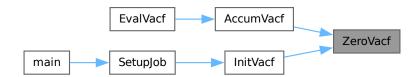
#### 3.90.1 Function Documentation

#### 3.90.1.1 ZeroVacf()

References countAcfAv, nValAcf, and viscAcfAv.

Referenced by AccumVacf(), and InitVacf().

Here is the caller graph for this function:



# 3.91 ZeroVacf.c

```
00002 * This file is part of Lamina.
00003
00004
       \star Lamina is free software: you can redistribute it and/or modify
00005
      * it under the terms of the GNU General Public License as published by
00006
       * the Free Software Foundation, either version 3 of the License, or
00007
      \star (at your option) any later version.
80000
00009
      \star Lamina is distributed in the hope that it will be useful,
00010 \,\star\, but WITHOUT ANY WARRANTY; without even the implied warranty of
* GNU General Public License for more details.
00012
00013 *
00014 \,\,\star\,\, You should have received a copy of the GNU General Public License
00015
       * along with Lamina. If not, see <a href="https://www.gnu.org/licenses/">https://www.gnu.org/licenses/</a>.
00016
00017
      Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void ZeroVacf() {
00026 int j;
00027 countAcfAv= 0;
00028 for(j = 1 ; j <= nValAcf ; j ++)
00029
       viscAcfAv[j] = 0.;
00030 }
```

# Index

AccumProps	atom1
AccumProps.c, 8	global.h, 83
main.c, 135	atom2
SetupJob.c, 174	global.h, 83
AccumProps.c	atomID
AccumProps, 8	global.h, 83
AccumVacf	atomIDInterface
AccumVacf.c, 10	global.h, 83
EvalVacf.c, 74	atomMass
AccumVacf.c	global.h, 84
AccumVacf, 10	atomRadius
Integrate, 11	global.h, 84
PrintVacf, 11	atomType
ZeroVacf, 12	global.h, 84
AllocArrays	ax
AllocArrays.c, 14	global.h, 84
SetupJob.c, 175	ay
AllocArrays.c	global.h, 84
AllocArrays, 14	
ApplyBoundaryCond	bond
ApplyBoundaryCond.c, 16	global.h, 84
main.c, 135	BondEnergy
ApplyBoundaryCond.c	global.h, <mark>85</mark>
ApplyBoundaryCond, 16	BondEnergyPerAtom
ApplyDrivingForce	global.h, <mark>85</mark>
ApplyDrivingForce.c, 18	BondID
main.c, 136	global.h, <mark>85</mark>
ApplyDrivingForce.c	BondLength
ApplyDrivingForce, 18	global.h, <mark>85</mark>
ApplyForce	BondType
ApplyForce.c, 20	global.h, <mark>85</mark>
main.c, 136	BrownianStep
ApplyForce.c	BrownianStep.c, 27
ApplyForce, 20	main.c, 138
ApplyLeesEdwardsBoundaryCond	BrownianStep.c
ApplyLeesEdwardsBoundaryCond.c, 22	BrownianStep, 27
main.c, 137	
ApplyLeesEdwardsBoundaryCond.c	cellList
ApplyLeesEdwardsBoundaryCond, 22	global.h, 85
ApplyShear	cells
ApplyShear.c, 24	global.h, 85
main.c, 137	cfOrg
ApplyShear.c	global.h, 86
ApplyShear, 24	cfVal
ApplyViscous	global.h, 86
ApplyViscous.c, 25	Close
main.c, 138	Close.c, 29
ApplyViscous.c	main.c, 139
ApplyViscous, 25	Close.c
· · ·	Close, 29

com	global.h, 89
global.h, 86	DeltaY
ComputeBondForce	global.h, 89
ComputeBondForce.c, 32	DeltaYij
ComputeBondForce.h, 36	global.h, 89
Compute Bond Force 32	DeltaYijNew
ComputeBondForce, 32 ComputeBondForce.h	global.h, 89 DeltaYijOld
ComputeBondForce, 36	global.h, 89
ComputeForcesCells	DeltaYijOldPair
ComputeForcesCells.c, 39	global.h, 89
main.c, 140	density
ComputeForcesCells.c	global.h, 90
ComputeForcesCells, 39	discDragx
ComputePairForce	global.h, 90
ComputePairForce.c, 43	discDragy
ComputePairForce.h, 49	global.h, 90
ComputePairForce.c	DisplaceAtoms
ComputePairForce, 43	DisplaceAtoms.c, 52
ComputePairForce.h	main.c, 141
ComputePairForce, 49	DisplaceAtoms.c
ComX	DisplaceAtoms, 52
global.h, 86	dnsty
ComX0	global.h, 90
global.h, 86	dump
ComXRatio	global.h, 90
global.h, 86	DumpBonds
ComY	DumpBonds.c, 54
global.h, 86	main.c, 142
ComY0	DumpBonds.c
global.h, 87	DumpBonds, 54
ComYRatio	dumpPairFlag
global.h, 87	global.h, 90
countAcfAv	DumpPairs
global.h, 87	DumpPairs.c, 55
countCorrAv	main.c, 143
global.h, 87	DumpPairs.c
countRdf	DumpPairs, 55
global.h, 87	DumpRestart
DampFlag	DumpRestart.c, 57
global.h, 87	main.c, 143
DEFINE_GLOBALS	DumpRestart.c
main.c, 134	DumpRestart, 57
deltaT	DumpState DumpState.c, 60
global.h, 88	main.c, 144
DeltaVXij	DumpState.c
global.h, 88	DumpState, 60
DeltaVYij	Dampotato, 00
global.h, 88	EvalCom
DeltaX	EvalCom.c, 62
global.h, 88	main.c, 145
DeltaXij	EvalCom.c
global.h, 88	EvalCom, 62
DeltaXijNew	EvalProps
global.h, 88	EvalProps.c, 64
DeltaXijOld	main.c, 146
global.h, 89	EvalProps.c
DeltaXijOldPair	EvalProps, 64

EvalRdf	frfAtom
EvalRdf.c, 66	global.h, 92
main.c, 147	fuSum
EvalRdf.c	global.h, 92
EvalRdf, 66	fvirSum
EvalSpacetimeCorr	global.h, 92
EvalSpacetimeCorr.c, 68	fx
main.c, 147	global.h, 93
EvalSpacetimeCorr.c	fxByfy
EvalSpacetimeCorr, 68	global.h, <mark>93</mark>
EvalUnwrap	fy
EvalUnwrap.c, 72	global.h, 93
main.c, 149	FyBylx
EvalUnwrap.c	global.h, 93
EvalUnwrap, 72	giobaiii, oo
EvalVacf	gamman
	global.h, 93
EvalVacf.c, 74	global.h
main.c, 149	•
EvalVacf.c	atom1, 83
AccumVacf, 74	atom2, 83
EvalVacf, 74	atomID, 83
EvalVrms	atomIDInterface, 83
EvalVrms.c, 77	atomMass, 84
main.c, 150	atomRadius, 84
EvalVrms.c	atomType, 84
	ax, 84
EvalVrms, 77	ay, <mark>84</mark>
EXTERN	bond, 84
global.h, 82	
<i>t</i>	BondEnergy, 85
fax	BondEnergyPerAtom, 85
global.h, 90	BondID, 85
fay	BondLength, 85
global.h, 90	BondType, 85
fpbond	cellList, 85
global.h, 91	cells, 85
fpcom	cfOrg, 86
global.h, 91	cfVal, 86
fpdnsty	com, 86
global.h, 91	ComX, 86
-	ComX0, 86
fpdump	
global.h, 91	ComXRatio, 86
fpmomentum	ComY, 86
global.h, 91	ComY0, 87
fppair	ComYRatio, 87
global.h, 91	countAcfAv, 87
fprdf	countCorrAv, 87
global.h, 91	countRdf, 87
fpresult	DampFlag, 87
global.h, 91	deltaT, 88
fpstress	DeltaVXij, 88
•	DeltaVYij, 88
global.h, 92	•
fpvisc	DeltaX, 88
global.h, 92	DeltaXij, 88
fpvrms	DeltaXijNew, 88
global.h, 92	DeltaXijOld, 89
fpxyz	DeltaXijOldPair, 89
global.h, 92	DeltaY, 89
freezeAtomType	DeltaYij, 89
global.h, 92	DeltaYijNew, 89
<del></del>	

DeltaYijOld, 89	nBuffCorr, 97
DeltaYijOldPair, 89	NDIM, 82
density, 90	nDiscInterface, 97
discDragx, 90	nFunCorr, 98
discDragy, 90	nodeDragx, 98
dnsty, 90	nodeDragy, 98
dump, 90	nPairActive, 98
dumpPairFlag, 90	nPairTotal, 98
EXTERN, 82	nValAcf, 98
fax, 90	nValCorr, 99
fay, 90	pair, 99
-	Pairatom1, 99
fpbond, 91	,
fpcom, 91	Pairatom2, 99
fpdnsty, 91	PairID, 99
fpdump, 91	PairXij, 99
fpmomentum, 91	PairYij, 99
fppair, 91	potEnergy, 100
fprdf, 91	prefix, 100
fpresult, 91	pressure, 100
fpstress, 92	RadiusIJ, 100
fpvisc, 92	RadiuslJInv, 100
fpvrms, 92	rangeRdf, 100
fpxyz, 92	rank, 101
freezeAtomType, 92	rCut, 101
frfAtom, 92	rdf, 101
fuSum, 92	real, 83
fvirSum, 92	region, 101
fx, 93	regionH, 101
fxByfy, 93	result, 101
fy, 93	rfAtom, 101
FyBylx, 93	ro, 102
gamman, 93	rx, 102
HaltCondition, 93	rxUnwrap, 102
	•
histRdf, 93	ry, 102
ImageX, 94	ryUnwrap, 102
ImageY, 94	shearDisplacement, 102
indexAcf, 94	shearVelocity, 103
indexCorr, 94	SignR, 82
initUcell, 94	size, 103
isBonded, 94	sizeHistRdf, 103
kappa, 94	sKinEnergy, 103
kb, 95	solver, 103
kinEnergy, 95	spacetimeCorr, 103
Kn, 95	spacetimeCorrAv, 104
limitAcfAv, 95	speed, 104
limitCorrAv, 95	sPotEnergy, 104
limitRdf, 95	sPressure, 104
master, 96	Sqr, <mark>83</mark>
molID, 96	SqrRadiusIJ, 104
momentum, 96	ssKinEnergy, 104
moreCycles, 96	ssPotEnergy, 105
nAtom, 96	ssPressure, 105
nAtomBlock, 96	ssTotEnergy, 105
nAtomInterface, 97	stepAcf, 105
nAtomType, 97	stepAvg, 105
• •	
nBond, 97	stepCorr, 105
nBondType, 97	stepCount, 106
nBuffAcf, 97	stepDump, 106

stepEquil,	106	ImageX
stepLimit,	106	global.h, 94
stepRdf, 10	06	ImageY
stepTraj, 10	06	global.h, 94
sTotEnergy	y, 107	indexAcf
strain, 107		global.h, 94
strainRate,	, 107	indexCorr
strech, 107	7	global.h, 94
stress, 107	7	Init
svirSum, 1	07	Init.c, 118
thermo, 10	17	main.c, 151
timeNow, 1	108	Init.c
TotalBondE	Energy, 108	Init, 118
TotalMass,		initUcell
totEnergy,		global.h, 94
uSum, 108		InitVacf
uSumPair,		InitVacf.c, 125
	PerAtom, 109	SetupJob.c, 175
virSum, 10		InitVacf.c
virSumBon		InitVacf, 125
virSumBon		ZeroVacf, 126
virSumBon		Integrate
virSumBon	-	AccumVacf.c, 11
virSumPair		Integrate.c, 127
virSumPair		Integrate.c
virSumPair		3
	• 1	Integrate, 127 isBonded
virSumPair		
virSumxx,		global.h, 94
virSumxy,		kappa
virSumyy,	111	global.h, 94
visc, 111		kb
viscAcf, 11		
viscAcfAv,		global.h, 95
viscAcfInt,		kinEnergy
viscAcfOrg		global.h, 95
VMeanSqr	, 112	Kn
vrms, 112	_	global.h, 95
VRootMea	nSqr, 112	Lamina: A Molecular Dynamics Package, 1
VSqr, 112		LeapfrogStep
vSum, 112		LeapfrogStep.c, 129
vSumX, 11		main.c, 154
vSumY, 11		LeapfrogStep.c
vvSum, 11	3	LeapfrogStep, 129
vx, 113		limitAcfAv
vy, 113		
xBoundary	v, 113	global.h, 95 limitCorrAv
xyz, 113		
yBoundary	v, 114	global.h, 95
		limitRdf
Halt.c		global.h, 95
	ionCheck, 116	main
HaltCondition		
global.h, 9		main.c, 154
HaltConditionCh	heck	main.c
Halt.c, 116		AccumProps, 135
main.c, 150	0	ApplyBoundaryCond, 135
histRdf		ApplyDrivingForce, 136
global.h, 9	3	ApplyForce, 136
		ApplyLeesEdwardsBoundaryCond, 137

ApplyShear, 137	global.h, 97
ApplyViscous, 138	nFunCorr
BrownianStep, 138	global.h, 98
Close, 139	nodeDragx
ComputeForcesCells, 140	global.h, 98
DEFINE_GLOBALS, 134	nodeDragy
DisplaceAtoms, 141	global.h, <mark>98</mark>
DumpBonds, 142	nPairActive
DumpPairs, 143	global.h, <mark>98</mark>
DumpRestart, 143	nPairTotal
DumpState, 144	global.h, <mark>98</mark>
EvalCom, 145	nValAcf
EvalProps, 146	global.h, <mark>98</mark>
EvalRdf, 147	nValCorr
EvalSpacetimeCorr, 147	global.h, 99
EvalUnwrap, 149	nair
EvalVacf, 149	pair
EvalVrms, 150	global.h, 99 Pairatom1
HaltConditionCheck, 150	
Init, 151	global.h, 99 Pairatom2
LeapfrogStep, 154	
main, 154	global.h, 99 PairID
prefix, 162	. •
PrintCom, 157	global.h, 99
PrintMomentum, 158	PairXij
PrintStress, 158	global.h, 99
PrintSummary, 158	PairYij
PrintVrms, 159	global.h, 99
SetupJob, 159	potEnergy
Trajectory, 160	global.h, 100
VelocityVerletStep, 161	prefix
master	global.h, 100
global.h, 96	main.c, 162
molID	pressure
global.h, 96	global.h, 100 PrintCom
momentum	
global.h, 96	main.c, 157 PrintCom.c, 165
moreCycles	PrintCom.c
global.h, 96	PrintCom, 165
nAtom	PrintMomentum
global.h, 96	main.c, 158
nAtomBlock	PrintMomentum.c, 167
global.h, 96	PrintMomentum.c
nAtomInterface	PrintMomentum, 167
global.h, 97	PrintStress
nAtomType	main.c, 158
global.h, 97	PrintStress.c, 168
nBond	PrintStress.c
global.h, 97	PrintStress, 168
nBondType	PrintSummary
global.h, 97	main.c, 158
nBuffAcf	PrintSummary.c, 169
global.h, 97	PrintSummary.c
nBuffCorr	PrintSummary, 169
global.h, 97	i innounnally, 103
MM (M (II ) I I ) M (II ) M (I	PrintVacf
	PrintVacf AccumVacf c 11
NDIM	AccumVacf.c, 11

PrintVacf, 171	sKinEnergy
PrintVrms	global.h, 103
main.c, 159	solver
PrintVrms.c, 172	global.h, 103
PrintVrms.c	source/AccumProps.c, 7, 8
PrintVrms, 172	source/AccumVacf.c, 9, 12
	source/AllocArrays.c, 13, 14
RadiusIJ	source/ApplyBoundaryCond.c, 15, 17
global.h, 100	source/ApplyDrivingForce.c, 17, 19
RadiuslJInv	source/ApplyForce.c, 20, 21
global.h, 100	source/ApplyLeesEdwardsBoundaryCond.c, 21, 23
rangeRdf	source/ApplyShear.c, 23, 24
global.h, 100	source/ApplyViscous.c, 25, 26
rank	source/BrownianStep.c, 26, 27
global.h, 101	source/Close.c, 28, 30
rCut	source/ComputeBondForce.c, 31, 34
global.h, 101	source/ComputeBondForce.h, 36, 38
rdf	source/ComputeForcesCells.c, 38, 41
global.h, 101	source/ComputePairForce.c, 43, 46
README.md, 7	source/ComputePairForce.h, 48, 51
real	source/DisplaceAtoms.c, 51, 53
global.h, <mark>83</mark>	source/DumpBonds.c, 53, 54
region	source/DumpPairs.c, 55, 56
global.h, 101	source/DumpRestart.c, 57, 58
regionH	source/DumpState.c, 59, 60
global.h, 101	source/EvalCom.c, 61, 62
result	source/EvalProps.c, 63, 65
global.h, 101	source/EvalRdf.c, 65, 67
rfAtom	source/EvalSpacetimeCorr.c, 68, 70
global.h, 101	source/EvalUnwrap.c, 71, 73
ro	source/EvalVacf.c, 73, 75
global.h, 102	source/EvalVrms.c, 76, 77
rx	source/global.h, 78, 114
global.h, 102	source/Halt.c, 116, 117
rxUnwrap	source/Init.c, 117, 121
global.h, 102	source/InitVacf.c, 124, 126
ry	source/Integrate.c, 127, 128
global.h, 102	source/LeapfrogStep.c, 128, 131
ryUnwrap	source/main.c, 133, 162
global.h, 102	source/PrintCom.c, 164, 166
SetupJob	source/PrintMomentum.c, 166, 167
main.c, 159	source/PrintStress.c, 167, 168
SetupJob.c, 176	source/PrintSummary.c, 169, 170
SetupJob.c	source/PrintVacf.c, 170, 171
AccumProps, 174	source/PrintVrms.c, 172, 173
AllocArrays, 175	source/SetupJob.c, 173, 177
InitVacf, 175	source/Trajectory.c, 178, 179
SetupJob, 176	source/VelocityVerletStep.c, 180, 181
shearDisplacement	source/ZeroVacf.c, 182, 183
global.h, 102	spacetimeCorr global.h, 103
shearVelocity	spacetimeCorrAv
global.h, 103	global.h, 104
SignR	speed
global.h, 82	global.h, 104
size	sPotEnergy
global.h, 103	global.h, 104
sizeHistRdf	sPressure
global.h, 103	5. 1000dio

global.h, 104	uSum
Sqr	global.h, 108
global.h, 83	uSumPair
SqrRadiusIJ	global.h, 108
global.h, 104	uSumPairPerAtom
ssKinEnergy	global.h, 109
global.h, 104	giobaini, 100
ssPotEnergy	VelocityVerletStep
global.h, 105	main.c, 161
	VelocityVerletStep.c, 180
ssPressure	Velocity VerletStep.c
global.h, 105	VelocityVerletStep, 180
ssTotEnergy	virSum
global.h, 105	global.h, 109
stepAcf	virSumBond
global.h, 105	
stepAvg	global.h, 109
global.h, 105	virSumBondxx
stepCorr	global.h, 109
global.h, 105	virSumBondxy
stepCount	global.h, 109
global.h, 106	virSumBondyy
stepDump	global.h, 109
global.h, 106	virSumPair
stepEquil	global.h, 110
global.h, 106	virSumPairxx
stepLimit	global.h, 110
global.h, 106	virSumPairxy
stepRdf	global.h, 110
global.h, 106	virSumPairyy
_	global.h, 110
stepTraj	virSumxx
global.h, 106	global.h, 110
sTotEnergy	virSumxy
global.h, 107	global.h, 110
strain	virSumyy
global.h, 107	global.h, 111
strainRate	visc
global.h, 107	
strech	global.h, 111
global.h, 107	viscAcf
stress	global.h, 111
global.h, 107	viscAcfAv
svirSum	global.h, 111
global.h, 107	viscAcfInt
	global.h, 111
thermo	viscAcfOrg
global.h, 107	global.h, 111
timeNow	VMeanSqr
global.h, 108	global.h, 112
TotalBondEnergy	vrms
global.h, 108	global.h, 112
TotalMass	VRootMeanSqr
global.h, 108	global.h, 112
totEnergy	VSqr
global.h, 108	global.h, 112
Trajectory	vSum
main.c, 160	
	global.h, 112
Trajectory.c, 178	vSumX
Trajectory.c	global.h, 112
Trajectory, 178	vSumY

```
global.h, 113
vvSum
    global.h, 113
VX
    global.h, 113
vy
    global.h, 113
xBoundary
    global.h, 113
xyz
    global.h, 113
yBoundary
    global.h, 114
ZeroVacf
    AccumVacf.c, 12
    InitVacf.c, 126
    ZeroVacf.c, 183
ZeroVacf.c
    ZeroVacf, 183
```