CBEMD: Parallelized MD in Various Thermodynamic Ensembles

Generated by Doxygen 1.8.2

Fri Jan 18 2013 00:07:23

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

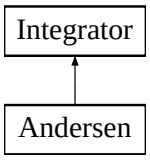
Class Documentation

4.1 Andersen Class Reference

NVT, Andersen thermostat Run (i.e. integrate) a system forward in time for a specified number of timesteps.

`#include <integrator.h>`

Inheritance diagram for Andersen:



Public Member Functions

- Andersen (double deltat, double temp, double nu)
- void **set_temp** (double temp)
- double **getTemp** () const
- double **getdt** ()
- int step (System ∗sys)
- double **nu** () const

Additional Inherited Members

4.1.1 Detailed Description

NVT, Andersen thermostat Run (i.e. integrate) a system forward in time for a specified number of timesteps.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Andersen::Andersen (double *deltat,* double *temp,* double *nu*)

Parameters

in	*deltat*	Incremental timestep
in	*temp*	Set point temperature

4.1.3 Member Function Documentation

4.1.3.1 int Andersen::step (System * *sys*) [virtual]

Step forward one timestep with the Andersen thermostat. This reports the instantaneous temperature after each step.

Parameters

in	*sys	Pointer to System to integrate.

Implements Integrator.

The documentation for this class was generated from the following files:

- integrator.h
- integrator.cpp

4.2 Atom Struct Reference

Atom class is defined as a struct so as to be easy to pass with MPI.

`#include <atom.h>`

Public Attributes

- double pos [NDIM]

  Cartesian coordinates.
- double prev_pos [NDIM]

  Cartesian coordinates for the previous position of the atom (needed for integrator)
- double vel [NDIM]

  Cartesian velocities (vx, vy, vz)
- double force [NDIM]

  Cartesian force, (fx, fy, fz)
- double mass

  Atomic mass (in reduced units)
- double diam

  Atomic diameter (in reduced units)
- int type

  Internally indexed type of this atom.
- int sys_index

  Global atom index, i.e. unique in the system.

4.2.1 Detailed Description

Atom class is defined as a struct so as to be easy to pass with MPI.

The documentation for this struct was generated from the following file:

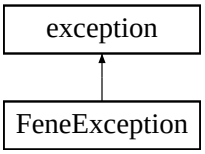
- atom.h

4.3 FeneException Class Reference

Fene exception class is thrown if there is an error.

`#include <interaction.h>`

Inheritance diagram for FeneException:



Public Member Functions

- **FeneException** (const int ind1, const int ind2, const double dist, const double r0)
- virtual const char * **what** () const throw ()

Protected Attributes

- int **ind1_**
- int **ind2_**
- double **dist_**
- double **r0_**

4.3.1 Detailed Description

Fene exception class is thrown if there is an error.

The documentation for this class was generated from the following file:

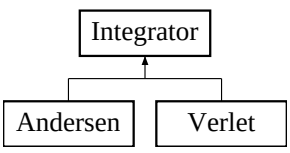
- interaction.h

4.4 Integrator Class Reference

< Abstract base class for integrators

`#include <integrator.h>`

Inheritance diagram for Integrator:



Public Member Functions

- void **set_dt** (const double dt)
- double **dt** () const
- void **set_temp** (double temp)

- double **getTemp** () const
- virtual int step (System *sys)=0

  Requires all subclasses to be able to execute a step.

Protected Attributes

- double **dt_**
- double **temp_**

4.4.1 Detailed Description

< Abstract base class for integrators

The step function should return SAFE_EXIT if successfully executed, integer error flag otherwise.

The documentation for this class was generated from the following file:

- integrator.h

4.5 Interaction Class Reference

This class stores how a pair of particles interacts.

`#include <interaction.h>`

Public Member Functions

- double force_energy (Atom *a1, Atom *a2, const vector< double > *box)

  Computes force (stored on atoms) and energy (returned)
- void set_force_energy (force_energy_ptr ife)

  Assign the potential calculator.
- void set_args (const vector< double > args)

  Assign energy and force arguments.
- force_energy_ptr check_force_energy_function () const

  Return the function for force and energy calculations.

4.5.1 Detailed Description

This class stores how a pair of particles interacts.

Returns energy and force as long as r < r_{cut}, else 0.

The documentation for this class was generated from the following file:

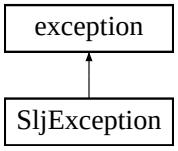
- interaction.h

4.6 SljException Class Reference

SLJ exception class is thrown if there is an error.

`#include <interaction.h>`

Inheritance diagram for SljException:



Public Member Functions

- **SljException** (const int ind1, const int ind2, const double dist, const double delta)
- virtual const char * **what** () const throw ()

Protected Attributes

- int **ind1_**
- int **ind2_**
- double **dist_**
- double **delta_**

4.6.1 Detailed Description

SLJ exception class is thrown if there is an error.

The documentation for this class was generated from the following file:

- interaction.h

4.7 System Class Reference

Public Member Functions

- System ()
- void set_box (const vector< double > new_box)

  Set the global system box size.
- vector< double > box () const

  Report system size.
- void set_T (const double T)

  Set the system temperature.
- void set_P (const double P)

  Set the system pressure.
- double T () const

  Report the temperature of the system.
- double P () const

  Report the pressure of the system.
- int total_atoms () const

  Return the number of atoms currently in this system (processor) including current ghosts.
- int natoms () const

  Return the number of atoms this system (processor) is responsible for.
- int add_atom_type (const string atom_name)

  Index an atom name.
- int atom_type (const string atom_name)

Return the internal index associated with an atom name.

- string atom_name (const unsigned int index)

  Return the name associated with an index for atom type.

- int add_bond_type (const string bond_name)

  Index a bond name.

- int bond_type (const string bond_name)

  Return the internal index associated with a bond name.

- string bond_name (const unsigned int index)

  Return the name associated with an index for bond type.

- const pair< int, int > get_bond (const int nbond)

  Return a specific bonded pair indices.

- int get_bond_type (const int nbond)

  Return the internal index of a bond.

- int nbonds ()

  Return the number of bonds in the system.

- void add_bond (const int atom1, const int atom2, const int type)
- vector< int > add_atoms (const int natoms, Atom *new_atoms)

  Add atom(s) to the system with an array of atoms.

- void add_ghost_atoms (const int natoms, Atom *new_atoms)

  Add ghost atom(s) to the system (does not update the number of atoms the processor is responsible for.

- vector< int > add_atoms (vector< Atom > *new_atoms)

  Add atom(s) to the system with an vector of atoms.

- int delete_atoms (vector< int > indices)

  Pop atoms with local indices from local storage.

- Atom * get_atom (int index)

  Get pointer to atom by local index.

- Atom copy_atom (int index)

  Report a copy of an atom.

- void set_rank (int rank)

  Record the rank this system corresponds to.

- int rank ()

  Return the rank of the system.

- void set_num_atoms (int size)

  Manually set the number of atoms in the system.

- void clear_ghost_atoms ()

  Clear ghost atoms from system.

- int gen_domain_info ()
- void set_max_rcut (const double max_rcut)

  Set the maximum cutoff radius of all interactions in the system.

- double max_rcut () const

  Return the max cutoff radius.

- void set_total_KE (const double ke)

  Set the global kinetic energy record.

- void set_total_PE (const double pe)

  Set the global potential energy record.

- double **KE** () const
- double **U** () const

Public Attributes

- double proc_widths [NDIM]

  Width for domain decomposition.
- vector< int > final_proc_breakup

  Final domain decomposition.
- int **xyz_id** [NDIM]
- double **xyz_limits** [NDIM][2]
- int **send_table** [NNEIGHBORS]
- vector< vector< Atom > > **send_lists**
- int **send_list_size** [NNEIGHBORS]
- int **get_list_size** [NNEIGHBORS]
- vector< vector< Atom > > **get_lists**
- vector< vector< Interaction > > interact

  Interaction matrix between atoms indexed by global id's (symetric)
- vector< string > global_atom_types

  Keeps a record of every atom's type.

4.7.1 Constructor & Destructor Documentation

4.7.1.1 System::System ( )

Upon initialization, resize vectors as necessary. Set T < 0.

4.7.2 Member Function Documentation

4.7.2.1 int System::add_atom_type (const string *atom_name*)

Index an atom name.

Tries to add an atom type to the system, associating a user specified name with an internal index to reference this type in the future. This can return 3 different values:

Returns 0 if, atom_name was new and was successfully indexed.

Returns -1 if, atom_name was bad (i.e. empty string).

Returns +1 if, atom_name already exists in the system and could not be indexed again.

Parameters

in	*atom_name*	User specified name to index

4.7.2.2 vector< int > System::add_atoms (const int *natoms,* Atom * *new_atoms*)

Add atom(s) to the system with an array of atoms.

Attempt to push an atom(s) into the system. This assigns the map automatically to link the atoms global index to

the local storage location. This reallocates the internal vector that stores the atoms; if a memory error occurs during such reallocation, an error is given and the system exits.

Parameters

in	*natoms*	Length of the array of atoms to add to the system.
in	∗*new_atoms*	Pointer to an array of atoms the user has created elsewhere.

4.7.2.3 vector< int > System::add_atoms (vector< Atom > ∗ *new_atoms*)

Add atom(s) to the system with an vector of atoms.

Attempt to push an atom(s) into the system. This assigns the map automatically to link the atoms global index to the local storage location. This reallocates the internal vector that stores the atoms; if a memory error occurs during such reallocation, an error is given and the system exits.

Parameters

in	*natoms*	Length of the array of atoms to add to the system.
in	∗*new_atoms*	Pointer to an array of atoms the user has created elsewhere.

4.7.2.4 void System::add_bond (const int *atom1,* const int *atom2,* const int *type*)

Adds a new bond and associated information to the System object.

Parameters

in	*atom1*	Global index of atom1 in the bond.
in	*atom1*	Global index of atom2 in the bond.
in	*type*	Internal index associated with this bond type.

4.7.2.5 int System::add_bond_type (const string *bond_name*)

Index a bond name.

Tries to add a bond type to the system, associating a user specified name with an internal index to reference this type in the future. This can return 3 different values:

Returns 0 if, bond_name was new and was successfully indexed.

Returns -1 if, bond_name was bad (i.e. empty string).

Returns +1 if, bond_name already exists in the system and could not be indexed again.

Parameters

in	*bond_name*	User specified name to index

4.7.2.6 void System::add_ghost_atoms (const int *natoms,* Atom * *new_atoms*)

Add ghost atom(s) to the system (does not update the number of atoms the processor is responsible for.

Attempt to push ghost atom(s) into the system. This assigns the map automatically to link the atoms global index to the local storage location. This reallocates the internal vector that stores the atoms; if a memory error occurs during such reallocation, an error is given and the system exits. Does not change num_atoms_ (the number of atoms a processor is responsible for)

Parameters

in	*natoms*	Length of the array of atoms to add to the system.
in	**new_atoms*	Pointer to an array of atoms the user has created elsewhere.

4.7.2.7 string System::atom_name (const unsigned int *index*)

Return the name associated with an index for atom type.

Returns the string "NULL" if failed, else user defined name of atom.

Parameters

in	*index*	Internal index to locate and return the name associated.

4.7.2.8 int System::atom_type (const string *name*)

Return the internal index associated with an atom name.

Returns the internal index associated with this atom name; returns -1 if not found.

Parameters

in	*name*	User defined name of atom type.

4.7.2.9 string System::bond_name (const unsigned int *index*)

Return the name associated with an index for bond type.

Returns the string "NULL" if failed, else user defined name of bond.

Parameters

in	*index*	Internal index to locate and return the name associated.

4.7.2.10 int System::bond_type (const string *name*)

Return the internal index associated with a bond name.

Returns the internal index associated with this bond name; returns -1 if not found.

Parameters

in	*name*	User defined name of bond type.

4.7.2.11 void System::clear_ghost_atoms ( )

Clear ghost atoms from system.

Clears the atoms communicated from neighbouring domains from the list of atoms stored in the system leaving only the atoms the system is responsible for.

4.7.2.12 int System::delete_atoms (vector< int > *indices*)

Pop atoms with local indices from local storage.

Remove atoms from the system. Needs to sort indices because erase() operation reorders things; also, because of this it is fastest to pop from lowest to highest index. Returns the number of atoms deleted.

Parameters

in	*indices*	Vector of local indices of atoms to delete from the system

4.7.2.13 int System::gen_domain_info ( )

Generates the x,y,z ids for each processor and the absolute extents of the domain Returns 0 if successful, -1 if not.

The documentation for this class was generated from the following files:

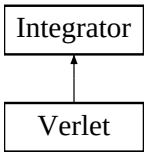
- system.h
- system.cpp

4.8 Verlet Class Reference

NVE, Verlet.

`#include <integrator.h>`

Inheritance diagram for Verlet:



Public Member Functions

- Verlet (double deltat)
- double **getTime** ()
- double **getdt** ()
- int step (System *sys)

Additional Inherited Members

4.8.1 Detailed Description

NVE, Verlet.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Verlet::Verlet (double *deltat*)

Parameters

in	*deltat*	Incremental timestep

4.8.3 Member Function Documentation

4.8.3.1 int Verlet::step (System * *sys*) [virtual]

Parameters

in,out	*sys	Pointer to System to make an integration step in.

Implements Integrator.

The documentation for this class was generated from the following files:

- integrator.h
- integrator.cpp

Chapter 5

File Documentation

5.1 andersen.cpp File Reference

Driver for MPI Version of CBEMD with Andersen Thermostat.

```
#include "CBEMD.h"
```

Functions

- int main (int argc, char ∗argv[])

5.1.1 Detailed Description

Driver for MPI Version of CBEMD with Andersen Thermostat.

Authors

{Nathan A. Mahynski, Carmeline Dsilva, Arun L. Prabhu, George Khoury, Frank Ricci, Jun Park}

5.1.2 Function Documentation

5.1.2.1 int main (int *argc,* char ∗ *argv[]*)

Parameters

∗*argv[]*	./andersen nsteps dt xml_file energy_file animation_file temperature nu The input arguments are as follows:

nsteps Number of timesteps to run for.

dt Timestep to use.

xml_file Initial configuration file (positions and velocities, etc.)

energy_file Energetics and interactions file that contains function parameters.

animation_file File to write the .xyz animation to.

temperature Desired reduced temperature.

nu Coupling constant to thermal heat bath (“collision" frequency).

5.2 atom.cpp File Reference

Functions for handling MD Atom Information.

```
#include "atom.h"
```

Functions

- void create_MPI_ATOM ()

  Creates the MPI_Atom class so it can be passed with MPI.
- void delete_MPI_atom ()

  Free the MPI type at the end of the program.

5.2.1 Detailed Description

Functions for handling MD Atom Information.

Author

{Nathan A. Mahynski, Carmeline Dsilva}

5.2.2 Function Documentation

5.2.2.1 void create_MPI_ATOM ( )

Creates the MPI_Atom class so it can be passed with MPI.

This function creates and commits to memory the MPI_ATOM derived data type. (see example of use at https://computing.llnl.gov/tutorials/mpi/#Derived_Data_Types)

See Also

initialize, MPI_ATOM

5.2.2.2 void delete_MPI_atom ( )

Free the MPI type at the end of the program.

This function utilizes the complementary MPI_Type_free routine to mark the MPI_ATOM type for deallocation at the end of the program.

See Also

finalize

5.3 atom.h File Reference

MD Atom Information.

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

Classes

- struct Atom

  Atom class is defined as a struct so as to be easy to pass with MPI.

Functions

- void create_MPI_ATOM ()

  Creates the MPI_Atom class so it can be passed with MPI.

- void delete_MPI_atom ()

  Free the MPI type at the end of the program.

5.3.1 Detailed Description

MD Atom Information.

Author

Nathan A. Mahynski

5.3.2 Function Documentation

5.3.2.1 void create_MPI_ATOM ( )

Creates the MPI_Atom class so it can be passed with MPI.

This function creates and commits to memory the MPI_ATOM derived data type. (see example of use at https://computing.llnl.gov/tutorials/mpi/#Derived_Data_Types)

See Also

initialize, MPI_ATOM

5.3.2.2 void delete_MPI_atom ( )

Free the MPI type at the end of the program.

This function utilizes the complementary MPI_Type_free routine to mark the MPI_ATOM type for deallocation at the end of the program.

See Also

finalize

5.4 CBEMD.h File Reference

Header grouping source headers into one simple header for the driver program(s).

```
#include "atom.h"
#include "system.h"
#include "integrator.h"
#include "misc.h"
#include "global.h"
#include "read_xml.h"
#include "interaction.h"
#include "mpiatom.h"
#include "initialize.h"
#include "mpi.h"
#include <limits>
```

5.4.1 Detailed Description

Header grouping source headers into one simple header for the driver program(s).

Author

{Carmeline Dsilva}

5.5 common.h File Reference

Header that lumps all common header files into one.

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <string>
```

5.5.1 Detailed Description

Header that lumps all common header files into one.

5.6 domain_decomp.cpp File Reference

Source code containing functions for performing domain decomposition.

```
#include "domain_decomp.h"
```

Functions

- void gen_sets (const vector< int > &factors, const double box[], const int level, double &final_diff, vector< int > &final_breakup, int add)

  Given the box size and factors of nprocs, checks which combination generates the most cubic domains.
- vector< int > factorize (const int nprocs)

  Given a number returns the prime factors (does not count 1 as a prime factor)
- int init_domain_decomp (const vector< double > box, const int nprocs, double widths[], vector< int > &final_breakup)

Decomposes the box into domains for each processor to handle.

- int get_processor (const vector< double > pos, const System *sys)

  Given the co-ordinates of a point, determines within which domain the point lies.
- int get_processor (const int x_id, const int y_id, const int z_id, const vector< int > &final_breakup)
- int get_xyz_ids (const int domain_id, const vector< int > &final_breakup, int xyz_id[])

  Given a domain_id specifies the x, y, z ids of the domain.
- int gen_send_lists (System *sys)

  Generates the lists of molecules that need to be passed to other processors.
- int gen_send_table (System *sys)

  Given the rank, generates the list of its neighbours.
- void gen_goes_to (const vector< int > &is_near_border, vector< int > &goes_to, const int ndims)

  Generates the list of neighbours a particle should be sent to based on the borders its near.
- int power (int base, int exponent)

  Computes the exponentiation of an integer by an integral power.
- int communicate_skin_atoms (System *sys)

  Communicates the atoms in the skin regions of the processors to the appropriate neighbours.

5.6.1 Detailed Description

Source code containing functions for performing domain decomposition.

Author

Arun L. Prabhu

5.6.2 Function Documentation

5.6.2.1 int communicate_skin_atoms (System * *sys*)

Communicates the atoms in the skin regions of the processors to the appropriate neighbours.

Communicates the atoms in the skin regions of the processors to the appropriate neighbours

Parameters

sys	[in,out] System to be evaluated

5.6.2.2 vector<int> factorize (const int *nprocs*)

Given a number returns the prime factors (does not count 1 as a prime factor)

Given a number returns the prime factors of that number. This function does not count 1 as a prime factor.

Parameters

in	*nprocs*	number of processors

5.6.2.3 void gen_goes_to (const vector< int > & *is_near_border*, vector< int > & *goes_to*, const int *ndims*)

Generates the list of neighbours a particle should be sent to based on the borders its near.

Generates the list of neighbours a particle should be sent to based on the borders its near

Parameters

in	*is_near_border*	information about which borders an atom is near
	goes_to	[in] vector containing the ids the current atoms needs to be communicated to
in	*ndims*	the number of dimensions of the simulation

5.6.2.4 int gen_send_lists (System * *sys*)

Generates the lists of molecules that need to be passed to other processors.

Generates the lists of molecules that need to be passed to other processors

Parameters

in,out	*sys*	System to be evaluated

5.6.2.5 int gen_send_table (System * *sys*)

Given the rank, generates the list of its neighbours.

Given the rank, generates the list of its neighbours Assumes 3D system, for different number of dimensions need to make this a recursive function

Parameters

in,out	*sys*	System to be evaluated

5.6.2.6 void gen_sets (const vector< int > & *factors*, const double *box[]*, const int *level*, double & *final_diff*, vector< int > & *final_breakup*, int *add*)

Given the box size and factors of nprocs, checks which combination generates the most cubic domains.

Given the box size and factors of nprocs, this recursive function checks which combination generates the most cubic domains

Parameters

in	*factors*	list of prime factors to be used
in	*box*	dimensions of the simulation box
in	*level*	the depth of the recursion
out	*final_breakup*	the number of divisions of each dimension
in	*add*	the number of times to be spent at this level of recursion

5.6.2.7 int get_processor (const vector< double > *pos*, const System * *sys*)

Given the co-ordinates of a point, determines within which domain the point lies.

Given the coordinates of a point, determines within which domain the point lies. This function is overloaded.

Parameters

in	*pos*	the vector containing the coordinates of a point
in	*sys*	system passed to be able to utilize final domain decomposition

5.6.2.8 int get_processor (const int *x_id,* const int *y_id,* const int *z_id,* const vector< int > & *final_breakup*)

Given the coordinates of a point, determines within which domain the point lies. This function is overloaded.

Parameters

in	*x_id*	how many domains away from x_min the current domain is
in	*y_id*	how many domains away from y_min the current domain is
in	*z_id*	how many domains away from z_min the current domain is
in	*fina_breakup*	the number of divisions of each dimension

5.6.2.9 int get_xyz_ids (const int *domain_id,* const vector< int > & *final_breakup,* int *xyz_id[]*)

Given a domain_id specifies the x, y, z ids of the domain.

Given a domain_id specifies the x, y, z ids of the domain

Parameters

in	*domain_id*	domain identifier, same as the rank of the processor handling the domain
in	*final_breakup*	the number of divisions of each dimension
in	*xyz_id*	how many domains away from the lower limit of each dimension the current domain is

5.6.2.10 int init_domain_decomp (const vector< double > *box,* const int *nprocs,* double *widths[],* vector< int > & *final_breakup*)

Decomposes the box into domains for each processor to handle.

Decomposes the box into domains for each processor to handle

Parameters

in	*box*	dimensions of the simulation box
in	*nprocs*	number of processors
in	*widths*	the dimensions of each domain
out	*final_breakup*	the number of divisions of each dimension

5.6.2.11 int power (int *base,* int *exponent*)

Computes the exponentiation of an integer by an integral power.

Computes the exponentiation of an integer by an integral power

Parameters

in	*base*	the base value to be raised to a power
in	*exponent*	the exponent of the power base is raised to

5.7 domain_decomp.h File Reference

Header file for domain decomposition.

```
#include "common.h"
#include "system.h"
#include "mpi.h"
```

Functions

- void gen_sets (const vector< int > &factors, const double box[], const int level, double &final_diff, vector< int > &final_breakup, int add)

  Given the box size and factors of nprocs, checks which combination generates the most cubic domains.
- vector< int > factorize (const int nprocs)

  Given a number returns the prime factors (does not count 1 as a prime factor)
- int init_domain_decomp (const vector< double > box, const int nprocs, double widths[], vector< int > &final_breakup)

  Decomposes the box into domains for each processor to handle.
- int get_processor (const vector< double > pos, const System *sys)

  Given the co-ordinates of a point, determines within which domain the point lies.
- int get_processor_id (const int x_id, const int y_id, const int z_id, const vector< int > &final_breakup)

  Given the x, y, z ids of a domain, determines the domain id (useful for locating neighbouring domains)
- int get_xyz_ids (const int domain_id, const vector< int > &final_breakup, int xyz_id[])

  Given a domain_id specifies the x, y, z ids of the domain.
- int gen_send_lists (System *sys)

  Generates the lists of molecules that need to be passed to other processors.
- int gen_send_table (System *sys)

  Given the rank, generates the list of its neighbours.
- void gen_goes_to (const vector< int > &is_near_border, vector< int > &goes_to, const int ndims)

  Generates the list of neighbours a particle should be sent to based on the borders its near.
- int power (int base, int exponent)

  Computes the exponentiation of an integer by an integral power.
- int communicate_skin_atoms (System *sys)

  Communicates the atoms in the skin regions of the processors to the appropriate neighbours.

5.7.1 Detailed Description

Header file for domain decomposition.

Author

Arun L. Prabhu

5.7.2 Function Documentation

5.7.2.1 int communicate_skin_atoms (System * *sys*)

Communicates the atoms in the skin regions of the processors to the appropriate neighbours.

Communicates the atoms in the skin regions of the processors to the appropriate neighbours

Parameters

sys	[in,out] System to be evaluated

5.7.2.2 vector<int> factorize (const int *nprocs*)

Given a number returns the prime factors (does not count 1 as a prime factor)

Given a number returns the prime factors of that number. This function does not count 1 as a prime factor.

Parameters

in	*nprocs*	number of processors

5.7.2.3 void gen_goes_to (const vector< int > & *is_near_border,* vector< int > & *goes_to,* const int *ndims*)

Generates the list of neighbours a particle should be sent to based on the borders its near.

Generates the list of neighbours a particle should be sent to based on the borders its near

Parameters

in	*is_near_border*	information about which borders an atom is near
	goes_to	[in] vector containing the ids the current atoms needs to be communicated to
in	*ndims*	the number of dimensions of the simulation

5.7.2.4 int gen_send_lists (System * *sys*)

Generates the lists of molecules that need to be passed to other processors.

Generates the lists of molecules that need to be passed to other processors

Parameters

in,out	*sys*	System to be evaluated

5.7.2.5 int gen_send_table (System * *sys*)

Given the rank, generates the list of its neighbours.

Given the rank, generates the list of its neighbours Assumes 3D system, for different number of dimensions need to make this a recursive function

Parameters

in,out	*sys*	System to be evaluated

5.7.2.6 void gen_sets (const vector< int > & *factors,* const double *box[],* const int *level,* double & *final_diff,* vector< int > & *final_breakup,* int *add*)

Given the box size and factors of nprocs, checks which combination generates the most cubic domains.

Given the box size and factors of nprocs, this recursive function checks which combination generates the most cubic domains

Parameters

in	*factors*	list of prime factors to be used
in	*box*	dimensions of the simulation box
in	*level*	the depth of the recursion
out	*final_breakup*	the number of divisions of each dimension
in	*add*	the number of times to be spent at this level of recursion

5.7.2.7 int get_processor (const vector< double > *pos*, const System * *sys*)

Given the co-ordinates of a point, determines within which domain the point lies.

Given the coordinates of a point, determines within which domain the point lies. This function is overloaded.

Parameters

in	*pos*	the vector containing the coordinates of a point
in	*sys*	system passed to be able to utilize final domain decomposition

5.7.2.8 int get_xyz_ids (const int *domain_id*, const vector< int > & *final_breakup*, int *xyz_id[]*)

Given a domain_id specifies the x, y, z ids of the domain.

Given a domain_id specifies the x, y, z ids of the domain

Parameters

in	*domain_id*	domain identifier, same as the rank of the processor handling the domain
in	*final_breakup*	the number of divisions of each dimension
in	*xyz_id*	how many domains away from the lower limit of each dimension the current domain is

5.7.2.9 int init_domain_decomp (const vector< double > *box*, const int *nprocs*, double *widths[]*, vector< int > & *final_breakup*)

Decomposes the box into domains for each processor to handle.

Decomposes the box into domains for each processor to handle

Parameters

in	*box*	dimensions of the simulation box
in	*nprocs*	number of processors
in	*widths*	the dimensions of each domain
out	*final_breakup*	the number of divisions of each dimension

5.7.2.10 int power (int *base*, int *exponent*)

Computes the exponentiation of an integer by an integral power.

Computes the exponentiation of an integer by an integral power

Parameters

in	*base*	the base value to be raised to a power
in	*exponent*	the exponent of the power base is raised to

5.8 force_calc.cpp File Reference

Source code for force calculation.

```
#include "force_calc.h"
```

Functions

- int send_atoms (System *sys)

  Move atoms between processors (domains)
- int force_calc (System *sys)

  Calculates the forces between the particles in the system.

5.8.1 Detailed Description

Source code for force calculation.

Authors

{Frank Ricci, Jun Park, Nathan Mahynski, Carmeline Dsilva, Arun Prabhu, George Khoury}

5.8.2 Function Documentation

5.8.2.1 int force_calc (System * *sys*)

Calculates the forces between the particles in the system.

Returns SAFE_EXIT if successful, else returns an error flag.

Parameters

in	*sys	Pointer to system for which to evaluate the forces

5.8.2.2 int send_atoms (System * *sys*)

Move atoms between processors (domains)

This function sends the atoms that have left the domain of the processor to the relevant neighboring processor. If the atom has moved more than one domain width, it returns an error flag, else returns SAFE_EXIT for success.

Parameters

*sys	[in] Pointer to system for which to move the atoms from

5.9 force_calc.h File Reference

Header for force_calc function.

```
#include "system.h"
#include "atom.h"
#include "interaction.h"
```

Functions

- int force_calc (System *sys)

  Calculates the forces between the particles in the system.
- int send_atoms (System *sys)

  Move atoms between processors (domains)

5.9.1 Detailed Description

Header for force_calc function.

Author

{Frank Ricci, Jun Park}

5.9.2 Function Documentation

5.9.2.1 int force_calc (System * *sys*)

Calculates the forces between the particles in the system.

Returns SAFE_EXIT if successful, else returns an error flag.

Parameters

in	*sys	Pointer to system for which to evaluate the forces

5.9.2.2 int send_atoms (System * *sys*)

Move atoms between processors (domains)

This function sends the atoms that have left the domain of the processor to the relevant neighboring processor. If the atom has moved more than one domain width, it returns an error flag, else returns SAFE_EXIT for success.

Parameters

*sys	[in] Pointer to system for which to move the atoms from

5.10 global.h File Reference

Global variables.

```
#include <exception>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <string>
```

Enumerations

- enum NAME_SIZES { **ATOM_NAME_LENGTH** = 100, **BOND_NAME_LENGTH** = 200, **MYERR_FLAG_SIZE** = 1000 }

  Set maximum size for user-defined names of things.

- enum RETURN_FLAGS { **SAFE_EXIT** = 0, **BAD_MEM** = 1, **ILLEGAL_VALUE** = 2, **MPI_FAIL** = 3, **INTEGRATE_FAIL** = 4, **FILE_ERROR** = 5, **BAD_EXIT** = 6 }

  Set error flags returned if a failure condition is met.

Variables

- const double WCA_CUTOFF = pow(2.0, 1/6.0)

  The Weeks-Chandler-Andersen cutoff length is a fixed number that is used in numerous places commonly in MD so it is precomputed here.
- MPI_Datatype MPI_ATOM

  MPI_ATOM is made visible to all routines.
- const int NDIM = 3

  Number of dimensions in the system (3D), but future work may include 2D as well. This leaves the door open for this.
- const int NNEIGHBORS = 26

  Nearest neighbors for 3D Domain decomposition.
- const int PARALLELDIM = 0

  dimension along which to do the domain decomposition

5.10.1 Detailed Description

Global variables.

Authors

{Nathan A. Mahynski, George Khoury}

5.11 initialize.cpp File Reference

Initialization routines for System object, as well as finalization of MPI.

```
#include "initialize.h"
```

Functions

- int initialize_from_files (const string xml_filename, const string energy_filename, System *sys)

  Parse an XML and energy file to obtain atom and interaction information.
- int start_mpi (int argc, char *argv[])

  Call MPI_Init and start the MPI ensuring it began successfully.
- int end_mpi ()

  Finalize MPI when program finishes successfully.
- int abort_mpi ()

  Abort MPI in case of failure.

5.11.1 Detailed Description

Initialization routines for System object, as well as finalization of MPI.

Author

Nathan A. Mahynski

5.11.2 Function Documentation

5.11.2.1 int abort_mpi ( )

Abort MPI in case of failure.

Abort the MPI as a result of a failure, freeing MPI_ATOM as necessary.

5.11.2.2 int end_mpi ( )

Finalize MPI when program finishes successfully.

Finalize MPI and clean up derived data types.

5.11.2.3 int initialize_from_files (const string *xml_filename*, const string *energy_filename*, System * *sys*)

Parse an XML and energy file to obtain atom and interaction information.

Parse an XML and energy file to obtain atom and interaction information. Returns 0 if successful, non-zero if failure. Operates in a "cascade" between ranks so that each processor (if MPI is used) opens and initializes from the input file in order.

Parameters

in	*xml_filename*	Name of coordinate file to open and read.
in	*energy_filename*	Name of file containing bonds, pair potential parameters, etc.
in,out	**sys*	Pointer to System object to store this information in.

5.11.2.4 int start_mpi (int *argc*, char * *argv[]*)

Call MPI_Init and start the MPI ensuring it began successfully.

Call MPI_Init and start the MPI ensuring it began successfully. Returns SAFE_EXIT if successful.

Parameters

in	*argc*	Number of arguments in *argv[].
in	**argv[]*	Array of character arguments.

5.12 integrator.cpp File Reference

MD Integrator(s) Information.

```
#include "integrator.h"
```

Functions

- int run (System *sys, Integrator *integrator, const int timesteps, const string outfile)

5.12.1 Detailed Description

MD Integrator(s) Information.

Authors

{George Khoury, Carmeline Dsilva, Nathan A. Mahynski}

5.12.2 Function Documentation

5.12.2.1 int run (System * *sys*, Integrator * *integrator*, const int *timesteps*, const string *outfile*)

Function returns SAFE_EXIT if successful, error flag otherwise.

Parameters

in,out	*sys	Pointer to System to integrate
in	*integrator	Pointer to Integrator to use
in	*timesteps*	Number of timesteps to integrate over
in	*outfile*	Name of file to print animation to

5.13 integrator.h File Reference

MD Integrator(s) Information.

```
#include "system.h"
#include "misc.h"
#include "mpi.h"
#include "atom.h"
#include "global.h"
#include "force_calc.h"
#include "read_xml.h"
#include <boost/tr1/random.hpp>
```

Classes

- class Integrator

  < *Abstract base class for integrators*
- class Verlet

  NVE, Verlet.
- class Andersen

  NVT, Andersen thermostat Run (i.e. integrate) a system forward in time for a specified number of timesteps.

Functions

- int run (System *sys, Integrator *integrator, const int timesteps, const string outfile)

5.13.1 Detailed Description

MD Integrator(s) Information.

Author

{Nathan A. Mahynski, George Khoury}

5.13.2 Function Documentation

5.13.2.1 int run (System * *sys*, Integrator * *integrator*, const int *timesteps*, const string *outfile*)

Function returns SAFE_EXIT if successful, error flag otherwise.

Parameters

in,out	*sys	Pointer to System to integrate
in	*integrator	Pointer to Integrator to use
in	*timesteps*	Number of timesteps to integrate over
in	*outfile*	Name of file to print animation to

5.14 interaction.cpp File Reference

Source code for interaction functions.

```
#include "interaction.h"
```

Functions

- double slj (Atom *atom1, Atom *atom2, const vector< double > *box, const vector< double > *args)
  Shifted Lennard-Jones Force.
- double harmonic (Atom *a1, Atom *a2, const vector< double > *box, const vector< double > *args)
  Computes force and energy of a Harmonic bond.
- double fene (Atom *a1, Atom *a2, const vector< double > *box, const vector< double > *args)
  Computes force and energy of a Fene bond.

5.14.1 Detailed Description

Source code for interaction functions.

Authors

{Nathan A. Mahynski, George Khoury}

5.14.2 Function Documentation

5.14.2.1 double fene (Atom * *a1*, Atom * *a2*, const vector< double > * *box*, const vector< double > * *args*)

Computes force and energy of a Fene bond.

Finitely Extensible Non-linear Elastic Bond (FENE) The Fene bond potential is given by:

$$U(r) = -\frac{1}{2}kr_0^2 ln\left(1-\left(\frac{r-\Delta}{r_0}\right)^2\right)+U_{WCA}$$

Where the short range repulsion is provided by the WCA potential:

$$U_{WCA} = 4\varepsilon\left(\left(\frac{\sigma}{r-\Delta}\right)^{12}-\left(\frac{\sigma}{r-\Delta}\right)^{6}\right)+\varepsilon r < 2^{1/6}\sigma+\Delta = 0 r-\Delta \geq 2^{1/6}\sigma$$

is usually set such that $ = (d_i+d_j)/2-1$ where d_i is the diameter of species i, but the user may decide on other parameters.

Parameters

in,out	*atom1	Pointer to first atom
in,out	*atom2	Pointer to second atom
in	*box	Pointer to vector of box size
in	*args	Vector of arguments <epsilon, sigma, delta, k, r0>

5.14.2.2 double harmonic (Atom * *a1*, Atom * *a2*, const vector< double > * *box*, const vector< double > * *args*)

Computes force and energy of a Harmonic bond.

Harmonic Bond The Harmonic bond potential is given by:

$$U(r) = \frac{1}{2}k(r - r_0)^2$$

Parameters

in,out	*atom1	Pointer to first atom
in,out	*atom2	Pointer to second atom
in	*box	Pointer to vector of box size
in	*args	Vector of arguments <k, r0>

5.14.2.3 double slj (Atom * *atom1*, Atom * *atom2*, const vector< double > * *box*, const vector< double > * *args*)

Shifted Lennard-Jones Force.

Computes force and energy of Shifted Lennard-Jones interaction.

This is the same as standard LJ if = 0. This is generally useful for systems with large size asymmetries. The energy U(r) is returned:

$$U(r) = 4\varepsilon\left(\left(\frac{\sigma}{r-\Delta}\right)^{12} - \left(\frac{\sigma}{r-\Delta}\right)^{6}\right) + U_{shift} r - \Delta < r_{cut} = 0 r - \Delta \geq r_{cut}$$

Forces are added to atoms:

$$F_i = -\frac{U}{r}\frac{r}{x_i} = -\frac{U}{r}\frac{x_i}{r} r - \Delta < r_{cut} = 0 r - \Delta \geq r_{cut}$$

Parameters

in,out	*atom1	Pointer to first atom
in,out	*atom2	Pointer to second atom
in	*box	Pointer to vector of box size
in	*args	Pointer to vector of arguments <epsilon, sigma, delta, U_{shift}, rcut^2>

5.15 interaction.h File Reference

Header file for interaction information.

```
#include "atom.h"
#include "misc.h"
#include "global.h"
```

Classes

- class Interaction

  This class stores how a pair of particles interacts.
- class FeneException

  Fene exception class is thrown if there is an error.
- class SljException

  SLJ exception class is thrown if there is an error.

Typedefs

- typedef double(∗ **force_energy_ptr**)(Atom ∗a1, Atom ∗a2, const vector< double > ∗box, const vector< double > ∗args)

Functions

- double slj (Atom ∗a1, Atom ∗a2, const vector< double > ∗box, const vector< double > ∗args)

  Computes force and energy of Shifted Lennard-Jones interaction.
- double fene (Atom ∗a1, Atom ∗a2, const vector< double > ∗box, const vector< double > ∗args)

  Computes force and energy of a Fene bond.
- double harmonic (Atom ∗a1, Atom ∗a2, const vector< double > ∗box, const vector< double > ∗args)

  Computes force and energy of a Harmonic bond.

5.15.1 Detailed Description

Header file for interaction information.

Author

Nathan A. Mahynski

5.15.2 Function Documentation

5.15.2.1 double fene (Atom ∗ *a1,* Atom ∗ *a2,* const vector< double > ∗ *box,* const vector< double > ∗ *args*)

Computes force and energy of a Fene bond.

Finitely Extensible Non-linear Elastic Bond (FENE) The Fene bond potential is given by:

$$U(r) = -\frac{1}{2}kr_0^2 ln\left(1-\left(\frac{r-\Delta}{r_0}\right)^2\right)+U_{WCA}$$

Where the short range repulsion is provided by the WCA potential:

$$U_{WCA} = 4\varepsilon\left(\left(\frac{\sigma}{r-\Delta}\right)^{12}-\left(\frac{\sigma}{r-\Delta}\right)^{6}\right)+\varepsilon r < 2^{1/6}\sigma+\Delta = 0 r-\Delta \geq 2^{1/6}\sigma$$

is usually set such that $ = (d_i+d_j)/2-1$ where d_i is the diameter of species i, but the user may decide on other parameters.

Parameters

in,out	∗*atom1*	Pointer to first atom
in,out	∗*atom2*	Pointer to second atom
in	∗*box*	Pointer to vector of box size
in	∗*args*	Vector of arguments <epsilon, sigma, delta, k, r0>

5.15.2.2 double harmonic (Atom ∗ *a1,* Atom ∗ *a2,* const vector< double > ∗ *box,* const vector< double > ∗ *args*)

Computes force and energy of a Harmonic bond.

Harmonic Bond The Harmonic bond potential is given by:

$$U(r) = \frac{1}{2}k(r-r_0)^2$$

Parameters

in,out	*atom1	Pointer to first atom
in,out	*atom2	Pointer to second atom
in	*box	Pointer to vector of box size
in	*args	Vector of arguments <k, r0>

5.15.2.3 double slj (Atom * *atom1*, Atom * *atom2*, const vector< double > * *box*, const vector< double > * *args*)

Computes force and energy of Shifted Lennard-Jones interaction.

Computes force and energy of Shifted Lennard-Jones interaction.

This is the same as standard LJ if = 0. This is generally useful for systems with large size asymmetries. The energy U(r) is returned:

$$U(r) = 4\varepsilon\left(\left(\frac{\sigma}{r-\Delta}\right)^{12} - \left(\frac{\sigma}{r-\Delta}\right)^{6}\right) + U_{shift} r-\Delta < r_{cut} = 0 r-\Delta \geq r_{cut}$$

Forces are added to atoms:

$$F_i = -\frac{U}{r}\frac{r}{x_i} = -\frac{U}{r}\frac{x_i}{r} r-\Delta < r_{cut} = 0 r-\Delta \geq r_{cut}$$

Parameters

in,out	*atom1	Pointer to first atom
in,out	*atom2	Pointer to second atom
in	*box	Pointer to vector of box size
in	*args	Pointer to vector of arguments <epsilon, sigma, delta, U_{shift}, rcut^2>

5.16 misc.cpp File Reference

Source code for miscellaneous routines.

```
#include "misc.h"
```

Functions

- void flag_error (const char *msg, const char *file, const int line)

  Report an error message.
- void flag_notify (const char *msg, const char *file, const int line)

  Report a notification.
- FILE * mfopen (const char *filename, const char *opt)

  Safely open a file.
- vector< double > pbc (const vector< double > coords, const vector< double > box)

  Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.
- vector< double > pbc (const double *coords, const vector< double > box)

  Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.
- double min_image_dist2 (const vector< double > coords1, const vector< double > coords2, const vector< double > box)

  Return the square of the minimum image distance between 2 coordinate vectors.
- double min_image_dist2 (const Atom *a1, const Atom *a2, const vector< double > *box, double *xyz)

  Returns the square of the minimum image distance between 2 atoms, also returns the minimum image distance vector xyz that points from atom1 to atom2.

- double unifRand ()

  Generate a random number between 0 and 1, returns a uniform number in [0,1].
- double unifRand (double a, double b)

  Generate a random number in a real interval.

5.16.1 Detailed Description

Source code for miscellaneous routines.

Author

Nathan A. Mahynski

5.16.2 Function Documentation

5.16.2.1 void flag_error (const char * *msg*, const char * *file*, const int *line*)

Report an error message.

Error messages are sent to stderr.

Parameters

in	*msg	Character string to print out.
in	*file	**FILE** this function is called from.
in	*line*	**LINE** this function is called from.

5.16.2.2 void flag_notify (const char * *msg*, const char * *file*, const int *line*)

Report a notification.

Notification messages are sent to stderr.

Parameters

in	*msg	Character string to print out.
in	*file	**FILE** this function is called from.
in	*line*	**LINE** this function is called from.

5.16.2.3 FILE* mfopen (const char * *filename*, const char * *opt*)

Safely open a file.

Tries to open a file, if it fails it returns a NULL pointer and alerts the user with an error message. If it succeeds it returns the file pointer.

Parameters

in	*filename	Character name of file to open.
in	*opt	File option ("r","w","rw+",etc.).

5.16.2.4 double min_image_dist2 (const vector< double > *coords1,* const vector< double > *coords2,* const vector< double > *box*)

Return the square of the minimum image distance between 2 coordinate vectors.

Parameters

in	*coords1*	Vector of cartesian coordinates of one atom
in	*coords2*	Vector of cartesian coordinates of the other atom
in	*box*	Vector of cartesian coordinates of the box

5.16.2.5 double min_image_dist2 (const Atom * *a1,* const Atom * *a2,* const vector< double > * *box,* double * *xyz*)

Returns the square of the minimum image distance between 2 atoms, also returns the minimum image distance vector xyz that points from atom1 to atom2.

Parameters

in	**a1*	Pointer to one atom
in	**a2*	Pointer to the other atom
in	*box*	Vector of cartesian coordinates of the box
in,out	**xyz*	Array of xyz displacements to be returned to the user (length 3)

5.16.2.6 vector<double> pbc (const vector< double > *coords,* const vector< double > *box*)

Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.

If this routine fails, it returns an empty vector (size = 0).

Parameters

in	*coords*	Vector of cartesian coordinates.
in	*box*	Vector of box dimensions (L_x, L_y, L_z).

5.16.2.7 vector<double> pbc (const double * *coords,* const vector< double > *box*)

Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.

If this routine fails, it returns an empty vector (size = 0).

Parameters

in	**coords*	Array of cartesian coordinates.
in	*box*	Vector of box dimensions (L_x, L_y, L_z).

5.16.2.8 double unifRand (double *a,* double *b*)

Generate a random number in a real interval.

Parameters

a	Lower end point of the interval
b	Upper end of the interval

5.17 misc.h File Reference

Header for Miscellaneous Routines.

```
#include <map>
#include <assert.h>
#include "atom.h"
#include "global.h"
```

Functions

- void flag_error (const char *msg, const char *file, const int line)

  Report an error message.
- void flag_notify (const char *msg, const char *file, const int line)

  Report a notification.
- FILE * mfopen (const char *filename, const char *opt)

  Safely open a file.
- vector< double > pbc (const vector< double > coords, const vector< double > box)

  Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.
- vector< double > pbc (const double *coords, const vector< double > box)

  Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.
- double min_image_dist2 (const vector< double > coords1, const vector< double > coords2, const vector< double > box)

  Return the square of the minimum image distance between 2 coordinate vectors.
- double min_image_dist2 (const Atom *a1, const Atom *a2, const vector< double > *box, double *xyz)

  Returns the square of the minimum image distance between 2 atoms, also returns the minimum image distance vector xyz that points from atom1 to atom2.
- double unifRand ()

  Generate a random number between 0 and 1, returns a uniform number in [0,1].
- double unifRand (double a, double b)

  Generate a random number in a real interval.

5.17.1 Detailed Description

Header for Miscellaneous Routines.

Author

Nathan A. Mahynski

5.17.2 Function Documentation

5.17.2.1 void flag_error (const char * *msg*, const char * *file*, const int *line*)

Report an error message.

Error messages are sent to stderr.

Parameters

in	*msg	Character string to print out.
in	*file	**FILE** this function is called from.
in	*line*	**LINE** this function is called from.

5.17.2.2 void flag_notify (const char * *msg*, const char * *file*, const int *line*)

Report a notification.

Notification messages are sent to stderr.

Parameters

in	*msg	Character string to print out.
in	*file	**FILE** this function is called from.
in	*line*	**LINE** this function is called from.

5.17.2.3 FILE* mfopen (const char * *filename*, const char * *opt*)

Safely open a file.

Tries to open a file, if it fails it returns a NULL pointer and alerts the user with an error message. If it succeeds it returns the file pointer.

Parameters

in	*filename	Character name of file to open.
in	*opt	File option ("r","w","rw+",etc.).

5.17.2.4 double min_image_dist2 (const vector< double > *coords1*, const vector< double > *coords2*, const vector< double > *box*)

Return the square of the minimum image distance between 2 coordinate vectors.

Parameters

in	*coords1*	Vector of cartesian coordinates of one atom
in	*coords2*	Vector of cartesian coordinates of the other atom
in	*box*	Vector of cartesian coordinates of the box

5.17.2.5 double min_image_dist2 (const Atom * *a1*, const Atom * *a2*, const vector< double > * *box*, double * *xyz*)

Returns the square of the minimum image distance between 2 atoms, also returns the minimum image distance vector xyz that points from atom1 to atom2.

Parameters

in	*a1	Pointer to one atom
in	*a2	Pointer to the other atom
in	*box*	Vector of cartesian coordinates of the box
in,out	*xyz	Array of xyz displacements to be returned to the user (length 3)

5.17.2.6 vector<double> pbc (const vector< double > *coords*, const vector< double > *box*)

Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.

If this routine fails, it returns an empty vector (size = 0).

Parameters

in	*coords*	Vector of cartesian coordinates.
in	*box*	Vector of box dimensions (L_x, L_y, L_z).

5.17.2.7 vector<double> pbc (const double * *coords*, const vector< double > *box*)

Returns the equivalent cartesian coordinates back in the simulation box assuming periodic boundaries.

If this routine fails, it returns an empty vector (size = 0).

Parameters

in	*coords	Array of cartesian coordinates.
in	*box*	Vector of box dimensions (L_x, L_y, L_z).

5.17.2.8 double unifRand (double *a*, double *b*)

Generate a random number in a real interval.

Parameters

a	Lower end point of the interval
b	Upper end of the interval

5.18 mpiatom.h File Reference

Variables

- MPI_Datatype MPI_ATOM

  MPI_ATOM is made visible to all routines.

5.18.1 Detailed Description

Author

Nathan A. Mahynski

5.19 read_interaction.cpp File Reference

Source code to read in interaction parameters from a parameter file.

```
#include "read_interaction.h"
```

Functions

- int read_interactions (const string filename, System *sys)

  Function to read in interaction parameters and store them into the system.
- force_energy_ptr get_fn (const string name, vector< double > *args, double *r_cut_max)

  Factory function to return a force_energy_ptr given a type of interaction name.

5.19.1 Detailed Description

Source code to read in interaction parameters from a parameter file.

Author

{Nathan A. Mahynski, Carmeline Dsilva}

5.19.2 Function Documentation

5.19.2.1 force_energy_ptr get_fn (const string *name*, vector< double > * *args*, double * *r_cut_max*)

Factory function to return a force_energy_ptr given a type of interaction name.

Essentially a factory, given a name, return the force_energy_ptr associated with it. Also check the arguments that will be passed to it are in acceptable range.

Parameters

in	*name*	Name of interaction, i.e. "fene" or "slj"
in	**args*	Pointer to vector of arguments that will be passed to this interaction later on
in,out	**r_cut_max*	Maximum cutoff radius for interactions

5.19.2.2 int read_interactions (const string *filename*, System * *sys*)

Function to read in interaction parameters and store them into the system.

Parameters

in	*filename*	name of file containing the interaction parameters to be processed
in,out	**sys*	Pointer to system

5.20 read_interaction.h File Reference

Header file for reading interactions in.

```
#include <boost/algorithm/string.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>
#include "mpi.h"
#include "misc.h"
#include "atom.h"
#include "system.h"
#include "global.h"
```

Functions

- int read_interactions (const string filename, System *sys)

  Function to read in interaction parameters and store them into the system.

- force_energy_ptr get_fn (const string name, vector< double > *args, double *r_cut_max)

  Factory function to return a force_energy_ptr given a type of interaction name.

5.20.1 Detailed Description

Header file for reading interactions in.

5.20.2 Function Documentation

5.20.2.1 force_energy_ptr get_fn (const string *name*, vector< double > * *args*, double * *r_cut_max*)

Factory function to return a force_energy_ptr given a type of interaction name.

Essentially a factory, given a name, return the force_energy_ptr associated with it. Also check the arguments that will be passed to it are in acceptable range.

Parameters

in	*name*	Name of interaction, i.e. "fene" or "slj"
in	**args*	Pointer to vector of arguments that will be passed to this interaction later on
in,out	**r_cut_max*	Maximum cutoff radius for interactions

5.20.2.2 int read_interactions (const string *filename*, System * *sys*)

Function to read in interaction parameters and store them into the system.

Parameters

in	*filename*	name of file containing the interaction parameters to be processed
in,out	**sys*	Pointer to system

5.21 read_xml.cpp File Reference

I/O for XML file format.

```
#include "read_xml.h"
```

Functions

- int read_xml (const string filename, System *sys)

  Read an xml file to initialize a System object.
- int print_xml (const string filename, const System *sys)

  Print the current state of the system to a .xml file.
- int write_xyz (const string filename, const System *sys, const int timestep, const bool wrap_pos)

  Write an animation file (.xyz)

5.21.1 Detailed Description

I/O for XML file format.

Author

Nathan A. Mahynski

5.21.2 Function Documentation

5.21.2.1 int print_xml (const string *filename*, const System * *sys*)

Print the current state of the system to a .xml file.

Print atom information to an xml file. Returns SAFE_EXIT if successful, else an error flag if failure. This only operates on the master node when multiple processors are used, the rest pause and are sequentially informed to send information as needed.

Parameters

in	*filename*	Name of file to open and read.
in,out	**sys*	Pointer to System object for the main node to stores its information at.

5.21.2.2 int read_xml (const string *filename*, System * *sys*)

Read an xml file to initialize a System object.

Parse an XML file to obtain atom information. Initializes a System object but only stores information that belongs to this processor rank according to domain decomposition. Returns SAFE_EXIT if successful, else returns an error flag.

Parameters

in	*filename*	Name of file to open and read.
in,out	**sys*	Pointer to System object to store its information at.

5.21.2.3 int write_xyz (const string *filename*, const System * *sys*, const int *timestep*, const bool *wrap_pos*)

Write an animation file (.xyz)

Write an xyz file that stores the coordinates of all the atoms in the simulation If timestep=0, then the file is created, or overwritten if it exists already. If timestep>0, then the current information is appended to the existing file. This file can then be read by VMD or another visualization program to produce animations.

Parameters

in	*filename*	Name of file to open and write to.
in	**sys*	System object where the atoms are stored
in	*timestep*	Current timestep of the simulation
in	*wrap_pos*	Whether the positions should be written in unwrapped (wrap_pos=false) or wrapped (wrap_pos=true) coordinates

5.22 read_xml.h File Reference

I/O for XML.

```
#include <boost/algorithm/string.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>
#include "mpi.h"
#include "misc.h"
#include "atom.h"
#include "system.h"
#include "global.h"
#include "read_interaction.h"
```

Functions

- int read_xml (const string filename, System *sys)

  Read an xml file to initialize a System object.
- int print_xml (const string filename, const System *sys)

  Print the current state of the system to a .xml file.
- int write_xyz (const string filename, const System *sys, const int timestep, const bool wrap_pos)

  Write an animation file (.xyz)

5.22.1 Detailed Description

I/O for XML.

Author

Nathan A. Mahynski

5.22.2 Function Documentation

5.22.2.1 int print_xml (const string *filename,* const System * *sys*)

Print the current state of the system to a .xml file.

Print atom information to an xml file. Returns SAFE_EXIT if successful, else an error flag if failure. This only operates on the master node when multiple processors are used, the rest pause and are sequentially informed to send information as needed.

Parameters

in	*filename*	Name of file to open and read.
in,out	**sys*	Pointer to System object for the main node to stores its information at.

5.22.2.2 int read_xml (const string *filename,* System * *sys*)

Read an xml file to initialize a System object.

Parse an XML file to obtain atom information. Initializes a System object but only stores information that belongs to this processor rank according to domain decomposition. Returns SAFE_EXIT if successful, else returns an error flag.

Parameters

in	*filename*	Name of file to open and read.
in,out	**sys*	Pointer to System object to store its information at.

5.22.2.3 int write_xyz (const string *filename,* const System * *sys,* const int *timestep,* const bool *wrap_pos*)

Write an animation file (.xyz)

Write an xyz file that stores the coordinates of all the atoms in the simulation If timestep=0, then the file is created, or overwritten if it exists already. If timestep>0, then the current information is appended to the existing file. This file can then be read by VMD or another visualization program to produce animations.

Parameters

in	*filename*	Name of file to open and write to.
in	*∗sys*	System object where the atoms are stored
in	*timestep*	Current timestep of the simulation
in	*wrap_pos*	Whether the positions should be written in unwrapped (wrap_pos=false) or wrapped (wrap_pos=true) coordinates

5.23 system.cpp File Reference

Source code for the System object.

```
#include "system.h"
```

5.23.1 Detailed Description

Source code for the System object.

Author

Nathan A. Mahynski

5.24 system.h File Reference

Header for MD System Information.

```
#include <map>
#include "atom.h"
#include "misc.h"
#include "interaction.h"
#include <list>
#include <algorithm>
#include "global.h"
```

Classes

- class System

5.24.1 Detailed Description

Header for MD System Information.

Author

Nathan A. Mahynski

5.25 verlet.cpp File Reference

Driver for MPI Version of CBEMD verlet.

```
#include "CBEMD.h"
```

Functions

- int main (int argc, char ∗argv[])

5.25.1 Detailed Description

Driver for MPI Version of CBEMD verlet.

Authors

{Nathan A. Mahynski, Carmeline Dsilva, Arun L. Prabhu, George Khoury, Frank Ricci, Jun Park}

5.25.2 Function Documentation

5.25.2.1 int main (int *argc,* char ∗ *argv[]*)

Parameters

∗*argv[]*	./verlet nsteps dt xml_file energy_file animation_file The input arguments are as follows:

nsteps Number of timesteps to run for.

dt Timestep to use.

xml_file Initial configuration file (positions and velocities, etc.)

energy_file Energetics and interactions file that contains function parameters.

animation_file File to write the .xyz animation to.