LUMA

1.4.0-alpha

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

---— Lattice Boltzmann @ The University of Manchester ---—

------------------------— L-U-M-A -------------------------—

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

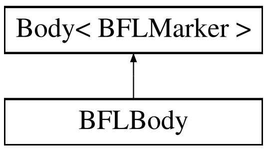
Class Documentation

5.1 BFLBody Class Reference

BFL body.

```
#include <BFLBody.h>
```

Inheritance diagram for BFLBody:



Public Member Functions

- BFLBody (void)

  Default constructor.
- ∼BFLBody (void)

  Default destructor.
- BFLBody (PCpts ∗_PCpts, GridObj ∗g_hierarchy, size_t id)

  Custom constructor to populate body from array of points.

Protected Member Functions

- void computeQ (int i, int j, int k, GridObj ∗g)

  Routine to compute wall distance Q.
- void computeQ (int i, int j, GridObj ∗g)

  Routine to compute wall distance Q.

Protected Attributes

- std::vector< std::vector< double > > Q

  Distance between adjacent lattice site and the surface of the body.

Friends

- class GridObj

5.1.1 Detailed Description

BFL body.

A BFL body is made up of a collection of BFLMarkers.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BFLBody::BFLBody (void)

Default constructor.

5.1.2.2 BFLBody::∼BFLBody (void)

Default destructor.

5.1.2.3 BFLBody::BFLBody (PCpts ∗ *_PCpts*, GridObj ∗ *g_hierarchy*, size_t *id*)

Custom constructor to populate body from array of points.

Parameters

_PCpts	pointer to point cloud data
g_hierarchy	pointer to grid hierarchy
id	ID of body in array of bodies.

5.1.3 Member Function Documentation

5.1.3.1 void BFLBody::computeQ (int *i*, int *j*, int *k*, GridObj ∗ *g*) [protected]

Routine to compute wall distance Q.

Computes Q values in 3D at a given local voxel for each application of the BFL BC. Performs a line-plane intersection algorithm for every possible triangular plane constructed out of the marker in the voxel and its nearest neighbours.

Parameters

i	local i-index of BFL voxel
j	local j-index of BFL voxel
k	local k-index of BFL voxel
g	pointer to owner grid

5.1.3.2 void BFLBody::computeQ (int *i*, int *j*, GridObj * *g*) [protected]

Routine to compute wall distance Q.

Computes Q values in 2D at a given local voxel for each application of the BFL BC. Performs a line-line intersection algorithm for each line segment either side of the voxel marker.

Parameters

i	local i-index of BFL voxel
j	local j-index of BFL voxel
g	pointer to owner grid

5.1.4 Friends And Related Function Documentation

5.1.4.1 friend class GridObj [friend]

5.1.5 Member Data Documentation

5.1.5.1 std::vector< std::vector<double> > BFLBody::Q [protected]

Distance between adjacent lattice site and the surface of the body.

There are two stores of values. Store 1 is the distance on one side of the wall and store 2 the distance on the other side. One store is appended to the other in this structure.

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

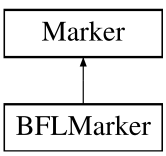
- BFLBody.h
- BFLBody.cpp

5.2 BFLMarker Class Reference

BFL marker.

```
#include <BFLMarker.h>
```

Inheritance diagram for BFLMarker:



Public Member Functions

- BFLMarker (void)

    Default constructor.
- ∼BFLMarker (void)

    Default destructor.
- BFLMarker (double x, double y, double z)

    Custom constructor with position.

Friends

- class BFLBody

Additional Inherited Members

5.2.1 Detailed Description

BFL marker.

This class declaration is for a BFL Lagrange point. A collection of these points form BFL body.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 BFLMarker::BFLMarker (void)

Default constructor.

5.2.2.2 BFLMarker::∼BFLMarker (void)

Default destructor.

5.2.2.3 BFLMarker::BFLMarker (double *x*, double *y*, double *z*)

Custom constructor with position.

Parameters

x	x-position of marker
y	y-position of marker
z	z-position of marker

5.2.3 Friends And Related Function Documentation

5.2.3.1 friend class BFLBody [friend]

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

- BFLMarker.h
- BFLMarker.cpp

5.3 Body< MarkerType > Class Template Reference

Generic body class.

`#include <Body.h>`

Public Member Functions

- Body (void)

  Default Constructor.
- ∼Body (void)

  Default destructor.
- Body (GridObj ∗g, size_t id)

  Custom constructor setting owning grid.

Protected Member Functions

- void addMarker (double x, double y, double z)

  Add marker to the body.
- MarkerData ∗ getMarkerData (double x, double y, double z)

  Retrieve marker data.
- void markerAdder (double x, double y, double z, int &curr_mark, std::vector< int > &counter)

  Downsampling marker adding method.
- bool isInVoxel (double x, double y, double z, int curr_mark)

  Determines whether a point is inside another marker's support voxel.
- bool isVoxelMarkerVoxel (double x, double y, double z)

  Determines whether a point is inside an existing marker's support voxel.

Protected Attributes

- double spacing

  Spacing of the markers in physical units.
- std::vector< MarkerType > markers

  Array of markers which make up the body.
- bool closed_surface

  Flag to specify whether or not it is a closed surface (for output)
- GridObj ∗ _Owner

  Pointer to owning grid.
- size_t id

  ID of body in array of bodies.

5.3.1 Detailed Description

template<typename MarkerType>
class Body< MarkerType >

Generic body class.

Can consist of any type of Marker so templated.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 template<typename MarkerType > Body< MarkerType >::Body (void)

Default Constructor.

5.3.2.2 template<typename MarkerType > Body< MarkerType >::~Body (void)

Default destructor.

5.3.2.3 template<typename MarkerType > Body< MarkerType >::Body (GridObj * *g*, size_t *id*)

Custom constructor setting owning grid.

Parameters

g	pointer to grid which owns this body.
id	indicates position of body in array of bodies.

5.3.3 Member Function Documentation

5.3.3.1 template<typename MarkerType > void Body< MarkerType >::addMarker (double *x*, double *y*, double *z*) [protected]

Add marker to the body.

Parameters

x	global X-position of marker.
y	global Y-position of marker.
z	global Z-position of marker.

5.3.3.2 template<typename MarkerType > MarkerData * Body< MarkerType >::getMarkerData (double *x*, double *y*, double *z*) [protected]

Retrieve marker data.

Return marker and voxel/primary support data associated with supplied global position.

Parameters

x	X-position nearest to marker to be retrieved.
y	Y-position nearest to marker to be retrieved.
z	Z-position nearest to marker to be retrieved.

Returns

MarkerData marker data structure returned. If no marker found, structure is marked as invalid.

5.3.3.3 template<typename MarkerType > bool Body< MarkerType >::isInVoxel (double *x*, double *y*, double *z*, int *curr_mark*) [protected]

Determines whether a point is inside another marker's support voxel.

Parameters

x	X-position of point.
y	Y-position of point.
z	Z-position of point.
curr_mark	ID of the marker.

Returns

true of false

5.3.3.4 template<typename MarkerType > bool Body< MarkerType >::isVoxelMarkerVoxel (double *x*, double *y*, double *z*) [protected]

Determines whether a point is inside an existing marker's support voxel.

Parameters

x	X-position of point.
y	Y-position of point.
z	Z-position of point.

Returns

true of false

5.3.3.5 template< typename MarkerType > void Body< MarkerType >::markerAdder (double *x*, double *y*, double *z*, int & *curr_mark*, std::vector< int > & *counter*) [protected]

Downsampling marker adding method.

This method tries to add a marker to body at the global location given but obeys the rules of a voxel-grid filter to ensure markers are distributed such that their spacing roughly matches the background lattice.

Parameters

x	desired global X-position of new marker.
y	desired globalY-position of new marker.
z	desired globalZ-position of new marker.
curr_mark	is a reference to the ID of last marker.
counter	is a reference to the total number of markers in the body.

5.3.4 Member Data Documentation

5.3.4.1 template<typename MarkerType> GridObj* Body< MarkerType >::_Owner [protected]

Pointer to owning grid.

5.3.4.2 template<typename MarkerType> bool Body< MarkerType >::closed_surface [protected]

Flag to specify whether or not it is a closed surface (for output)

5.3.4.3 template<typename MarkerType> size_t Body< MarkerType >::id [protected]

ID of body in array of bodies.

5.3.4.4 template<typename MarkerType> std::vector<MarkerType> Body< MarkerType >::markers [protected]

Array of markers which make up the body.

5.3.4.5 template<typename MarkerType> double Body< MarkerType >::spacing [protected]

Spacing of the markers in physical units.

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

- Body.h

5.4 MpiManager::buffer_struct Struct Reference

Structure storing buffers sizes in each direction for particular grid.

```
#include <MpiManager.h>
```

Public Attributes

- int size [L_MPI_DIRS]

    Buffer sizes for each direction.
- int level

    Grid level.
- int region

    Region number.

5.4.1 Detailed Description

Structure storing buffers sizes in each direction for particular grid.

5.4.2 Member Data Documentation

5.4.2.1 int MpiManager::buffer_struct::level

Grid level.

5.4.2.2 int MpiManager::buffer_struct::region

Region number.

5.4.2.3 int MpiManager::buffer_struct::size[L_MPI_DIRS]

Buffer sizes for each direction.

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

- MpiManager.h

5.5 GridObj Class Reference

Grid class.

```
#include <GridObj.h>
```

Public Member Functions

- GridObj ()

  Default Constructor.
- GridObj (int level)

  Serial build constructor for top level grid.
- GridObj (int RegionNumber, GridObj &pGrid)

  Constructor for a sub-grid.
- GridObj (int level, std::vector< int > local_size, std::vector< std::vector< double > > GlobalLimsPos)

  MPI constructor for top level grid.
- ∼GridObj ()

  Default Destructor.
- void LBM_initVelocity ()

  Method to initialise the lattice velocity.
- void LBM_initRho ()

  Method to initialise the lattice density.
- void LBM_initGrid ()

  Wrapper to initialise all L0 lattice quantities.
- void LBM_initGrid (std::vector< int > local_size, std::vector< std::vector< double > > GlobalLimsPos)

  Method to initialise all L0 lattice quantities.
- void LBM_initSubGrid (GridObj &pGrid)

  Method to initialise all sub-grid quantities.
- void LBM_initBoundLab ()

  Method to initialise wall and object labels on L0.
- void LBM_initSolidLab ()

  Method to initialise label-based solids.
- void LBM_initRefinedLab (GridObj &pGrid)

  Method to initialise all labels on sub-grids.
- void LBM_init_getInletProfile ()

  Method to import an input profile from a file.
- void LBM_kbcCollide (int i, int j, int k, IVector< double > &f_new)

  KBC collision operator.
- void LBM_macro (int i, int j, int k)

  Site-specific macroscopic update.
- void LBM_resetForces ()

  Method to reset body forces.
- void bc_applyBounceBack (int label, int i, int j, int k)

  Method to apply half-way bounce-back.
- void bc_applySpecReflect (int label, int i, int j, int k)

  Method to apply half-way specular reflection.
- void bc_applyRegularised (int label, int i, int j, int k)

  Method to apply regularised velocity inlet.
- void bc_applyExtrapolation (int label, int i, int j, int k)

  Method to apply extrapolation outlet.
- void bc_applyBfl (int i, int j, int k)

  Method to apply BFL bounce-back.
- void bc_applyNrbc (int i, int j, int k)

  Method to apply NRBC.
- void LBM_addSubGrid (int RegionNumber)

  Wrapper method to add sub-grid to this grid.
- void io_textout (std::string output_tag)

Verbose ASCII writer.
- void io_fgaout ()
  .fga file writer.
- void io_restart (eIOFlag IO_flag)
  Restart file read-writer.
- void io_probeOutput ()
  Probe writer.
- void io_lite (double tval, std::string Tag)
  ASCII dump of grid data.
- int io_hdf5 (double tval)
  HDF5 writer.
- void LBM_multi_opt (int subcycle=0)
  Optimised LBM multi-grid kernel.

Public Attributes

- std::vector< double > XPos
  Vector of global X positions of each site.
- std::vector< double > YPos
  Vector of global Y positions of each site.
- std::vector< double > ZPos
  Vector of global Z positions of each site.
- IVector< eType > LatTyp
  Flattened 3D array of site labels.
- double dh
  Physical lattice spacing (same for x, y and z)
- int region_number
  Region number.
- int level
  Level in embedded grid hierarchy.
- double dt
  Physical time step size.
- int t
  Number of completed iterations on this level.
- double nu
  Kinematic viscosity (in lattice units)
- double omega
  Relaxation frequency.
- double timeav_mpi_overhead
  Time-averaged time of MPI communication.
- double timeav_timestep
  Time-averaged time of a timestep.
- int N_lim
  Local size of grid in X-direction.
- int M_lim
  Local size of grid in Y-direction.
- int K_lim
  Local size of grid in Z-direction.
- double XOrigin
  Position of grid left edge.
- double YOrigin
  Position of grid bottom edge.
- double ZOrigin
  Position of grid front edge.

Friends

- class MpiManager
- class ObjectManager
- class GridUtils

5.5.1 Detailed Description

Grid class.

This class represents a grid (lattice) and is capable of owning a nested hierarchy of child grids.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 GridObj::GridObj (void)

Default Constructor.

5.5.2.2 GridObj::GridObj (int *level*)

Serial build constructor for top level grid.

Coarse limits are set to zero and then L0-specific initialiser called.

Parameters

level	always should be zero as top level grid.

5.5.2.3 GridObj::GridObj (int *RegionNumber*, GridObj & *pGrid*)

Constructor for a sub-grid.

This is not called directly but by the addSubGrid() method which first performs a check to see if a sub-grid is required.

Parameters

RegionNumber	ID indicating the region of nested refinement to which this sub-grid belongs.
pGrid	pointer to parent grid.

5.5.2.4 GridObj::GridObj (int *level*, std::vector< int > *local_size*, std::vector< std::vector< double > > *GlobalLimsPos*)

MPI constructor for top level grid.

When using MPI, this constructors a local grid which represents an appropriate portion of the top-level grid as dictated by the extent of this rank.

Parameters

level	always should be zero as top level grid.
local_size	vector indicating dimensions of local grid including halo.
GlobalLimsPos	vector indicating the global positions of the edges of this local grid core as held by the MpiManager.

5.5.2.5 GridObj::∼GridObj (void)

Default Destructor.

5.5.3 Member Function Documentation

5.5.3.1 void GridObj::bc_applyBfl (int *i*, int *j*, int *k*)

Method to apply BFL bounce-back.

Currently, assumes only 1 BFL body present on the grid.

Parameters

i	current site i-index.
j	current site j-index.
k	current site k-index.

5.5.3.2 void GridObj::bc_applyBounceBack (int *label*, int *i*, int *j*, int *k*)

Method to apply half-way bounce-back.

Parameters

label	current site label.
i	current site i-index.
j	current site j-index.
k	current site k-index.

5.5.3.3 void GridObj::bc_applyExtrapolation (int *label*, int *i*, int *j*, int *k*)

Method to apply extrapolation outlet.

Can only be applied on right-hand wall.

Parameters

label	current site label.

Parameters

i	current site i-index.
j	current site j-index.
k	current site k-index.

5.5.3.4 void GridObj::bc_applyNrbc (int *i*, int *j*, int *k*)

Method to apply NRBC.

Not implemented in this version.

Parameters

i	current site i-index.
j	current site j-index.
k	current site k-index.

5.5.3.5 void GridObj::bc_applyRegularised (int *label*, int *i*, int *j*, int *k*)

Method to apply regularised velocity inlet.

Can be applied on any wall.

Parameters

label	current site label.
i	current site i-index.
j	current site j-index.
k	current site k-index.

5.5.3.6 void GridObj::bc_applySpecReflect (int *label*, int *i*, int *j*, int *k*)

Method to apply half-way specular reflection.

Symmetry boundary condition for free-slip walls.

Parameters

label	current site label.
i	current site i-index.
j	current site j-index.
k	current site k-index.

5.5.3.7 void GridObj::io_fgaout ( )

.fga file writer.

Writes the components of the macroscopic velocity of the grid at time t and call recursively for any sub-grid. Writes the data of each subgrid in a different .fga file. .fga is the ASCII file format used by Unreal Engine 4 to read the data that populates a VectorField object. It doesn't do anything if the model is not 2D or 3D. Since .fga files can only store 3D data

5.5.3.8 int GridObj::io_hdf5 (double *tval*)

HDF5 writer.

Useful grid quantities written out as scalar arrays. Creates one $*$.h5 file per grid and data is grouped into timesteps within each file. Should be used with the merge tool at post-processing to conver to sructured VTK output readable in paraview.

Parameters

tval	time value being written out.

5.5.3.9 void GridObj::io_lite (double *tval,* std::string *TAG*)

ASCII dump of grid data.

Generic ASCII writer for each rank to write out all grid data in rows into a single, unsorted file.

Parameters

tval	time value being written out.
TAG	text identifier for the data.

5.5.3.10 void GridObj::io_probeOutput ( )

Probe writer.

This routine writes the quantities at the probe locations to a single file.

5.5.3.11 void GridObj::io_restart (eIOFlag *IO_flag*)

Restart file read-writer.

This routine writes/reads the current rank's data in the custom restart file format. If the file already exists, data is appended. IB body data are also written out but no other body information at present.

Parameters

IO_flag	flag to indicate whether a write or read

5.5.3.12 void GridObj::io_textout (std::string *output_tag*)

Verbose ASCII writer.

Writes all the contents of the grid class at time t and call recursviely for any sub-grids. Writes to text file "Grids.out" by default.

Parameters

output_tag	text string added to top of output for identification.

5.5.3.13 void GridObj::LBM_addSubGrid (int *RegionNumber*)

Wrapper method to add sub-grid to this grid.

Parameters

RegionNumber	ID indicating the region of nested refinement to which this sub-grid belongs.

5.5.3.14 void GridObj::LBM_init_getInletProfile ( )

Method to import an input profile from a file.

Input data may be over- or under-sampled but it must span the physical dimensions of the inlet otherwise the software does not known how to scale the data to fit. Inlet profile is always assumed to be oriented vertically (y-direction).

5.5.3.15 void GridObj::LBM_initBoundLab ( )

Method to initialise wall and object labels on L0.

The virtual wind tunnel definitions are implemented by this method.

5.5.3.16 void GridObj::LBM_initGrid ( )

Wrapper to initialise all L0 lattice quantities.

This method wraps the MPI-specific version. It is called by the serial build and sets the MPI-specific arguments to default values before calling the full initialiser.

5.5.3.17 void GridObj::LBM_initGrid (std::vector$<$ int $>$ *local_size,* std::vector$<$ std::vector$<$ double $>$ $>$ *rank_core_edge*)

Method to initialise all L0 lattice quantities.

Parameters

local_size	local grid size on this rank including halo.
rank_core_edge	absolute positions of the rank edges (excludes overlapping halo).

5.5.3.18 void GridObj::LBM_initRefinedLab (GridObj & *pGrid*)

Method to initialise all labels on sub-grids.

Boundary labels are set by considering parent labels on overlapping sites and then assigning child labels appropriately.

Parameters

pGrid	reference to parent grid.

5.5.3.19 void GridObj::LBM_initRho ( )

Method to initialise the lattice density.

5.5.3.20 void GridObj::LBM_initSolidLab ( )

Method to initialise label-based solids.

5.5.3.21 void GridObj::LBM_initSubGrid (GridObj & *pGrid*)

Method to initialise all sub-grid quantities.

Parameters

pGrid	reference to parent grid.

5.5.3.22 void GridObj::LBM_initVelocity ( )

Method to initialise the lattice velocity.

Unless the L_NO_FLOW macro is defined, the initial velocity everywhere will be set to the values specified in the definitions file.

5.5.3.23 void GridObj::LBM_kbcCollide (int *i*, int *j*, int *k*, IVector< double > & *f_new*)

KBC collision operator.

Applies KBC collision operator using the KBC-N4 and KBC-D models in 3D and 2D, respectively.

Parameters

i	i-index of lattice site.
j	j-index of lattice site.
k	k-index of lattice site.
f_new	reference to the temporary, post-collision grid.

5.5.3.24 void GridObj::LBM_macro (int *i*, int *j*, int *k*)

Site-specific macroscopic update.

Overload of macroscopic quantity calculation to allow it to be applied to a single site as used by the MPI unpacking routine to update the values for the next collision step. This routine does not update the time-averaged quantities.

Parameters

i	i-index of lattice site.
j	j-index of lattice site.
k	k-index of lattice site.

5.5.3.25 void GridObj::LBM_multi_opt (int *subcycle* = 0)

Optimised LBM multi-grid kernel.

This kernel compresses the old kernel into a single loop in order to make it more efficient. Capabilities are current limited with this kernel with incompatible options giving unpredictable results. Use with caution.

Parameters

subcycle	sub-cycle to be performed if called from a subgrid.

5.5.3.26 void GridObj::LBM_resetForces ( )

Method to reset body forces.

Resets both Cartesian and Lattice force vectors to zero.

5.5.4 Friends And Related Function Documentation

5.5.4.1 friend class GridUtils [friend]

5.5.4.2 friend class MpiManager [friend]

5.5.4.3 friend class ObjectManager [friend]

5.5.5 Member Data Documentation

5.5.5.1 double GridObj::dh

Physical lattice spacing (same for x, y and z)

5.5.5.2 double GridObj::dt

Physical time step size.

5.5.5.3 int GridObj::K_lim

Local size of grid in Z-direction.

5.5.5.4 IVector<eType> GridObj::LatTyp

Flattened 3D array of site labels.

5.5.5.5 int GridObj::level

Level in embedded grid hierarchy.

5.5.5.6 int GridObj::M_lim

Local size of grid in Y-direction.

5.5.5.7 int GridObj::N_lim

Local size of grid in X-direction.

5.5.5.8 double GridObj::nu

Kinematic viscosity (in lattice units)

5.5.5.9 double GridObj::omega

Relaxation frequency.

5.5.5.10 int GridObj::region_number

Region number.

5.5.5.11 int GridObj::t

Number of completed iterations on this level.

5.5.5.12 double GridObj::timeav_mpi_overhead

Time-averaged time of MPI communication.

5.5.5.13 double GridObj::timeav_timestep

Time-averaged time of a timestep.

5.5.5.14 double GridObj::XOrigin

Position of grid left edge.

5.5.5.15 std::vector<double> GridObj::XPos

Vector of global X positions of each site.

5.5.5.16 double GridObj::YOrigin

Position of grid bottom edge.

5.5.5.17 std::vector<double> GridObj::YPos

Vector of global Y positions of each site.

5.5.5.18 double GridObj::ZOrigin

Position of grid front edge.

5.5.5.19 std::vector<double> GridObj::ZPos

Vector of global Z positions of each site.

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

- GridObj.h
- GridObj.cpp
- GridObj_init_grids.cpp
- GridObj_ops_boundary.cpp
- GridObj_ops_io.cpp
- GridObj_ops_lbm.cpp
- GridObj_ops_lbm_optimised.cpp

5.6 GridUnits Class Reference

GridUnits.

`#include <GridUnits.h>`

Public Member Functions

- GridUnits ()
- ∼GridUnits ()

Static Public Member Functions

- template<typename T >
  static T m2cm (const T meters)

  Convert from m to cm.
- template<typename T >
  static T ulat2uphys (T ulat, GridObj ∗currentGrid)

  Velocity in lattice units to velocity in physical units.

5.6.1 Detailed Description

GridUnits.

This class contains static methods for unit conversion (the only ones implemented are from m to cm and velocity from lattice units to m/s)

5.6.2 Constructor & Destructor Documentation

5.6.2.1 GridUnits::GridUnits ( ) `[inline]`

5.6.2.2 GridUnits::∼GridUnits ( ) `[inline]`

5.6.3 Member Function Documentation

5.6.3.1 template<typename T > static T GridUnits::m2cm (const T *meters*) `[inline]`, `[static]`

Convert from m to cm.

5.6.3.2 template<typename T > static T GridUnits::ulat2uphys (T *ulat*, GridObj ∗ *currentGrid*) `[inline]`, `[static]`

Velocity in lattice units to velocity in physical units.

Converts velocity component from lattice units to m/s. It uses the L_PHYSICAL_U introduced by the user, dh and dt. You can introduce any L_PHYSICAL_U you want, but the reference lenght (usualy the width of the domain) , the Re number and the LBM parameters will remain the same. So you will be implicitly changing the physical viscosity of your fluid when you change L_PHYSICAL_U

Parameters

ulat	Lattice velocity.
currentGrid	Pointer to the current grid.

Returns

physical velocity

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

- GridUnits.h

5.7 GridUtils Class Reference

Grid utility class.

```
#include <GridUtils.h>
```

Static Public Member Functions

- static void createOutputDirectory (std::string path_str)

    Create output directory.
- static std::vector< int > onespace (int min, int max)

    Creates a linearly-spaced vector of integers.
- static std::vector< double > linspace (double min, double max, int n)

    Creates a linearly-spaced vector of values.
- static double vecnorm (double vec[L_DIMS])

    Computes the L2 norm using the vector supplied.
- static double vecnorm (double val1, double val2)

    Computes the L2 norm using the vector components supplied.
- static double vecnorm (double val1, double val2, double val3)

    Computes the L2 norm using the vector components supplied.
- static double vecnorm (std::vector< double > vec)

    Computes the L2 norm using the vector supplied.
- static std::vector< int > getFineIndices (int coarse_i, int x_start, int coarse_j, int y_start, int coarse_k, int z_start)

    Gets the indices of the fine site given the coarse site.
- static std::vector< int > getCoarseIndices (int fine_i, int x_start, int fine_j, int y_start, int fine_k, int z_start)

    Gets the indices of the coarse site given the fine site.
- static double dotprod (std::vector< double > vec1, std::vector< double > vec2)

    Computes the scalar product of two vectors.
- static std::vector< double > subtract (std::vector< double > a, std::vector< double > b)

    Subtracts two vectors.
- static std::vector< double > add (std::vector< double > a, std::vector< double > b)

    Adds two vectors.
- static std::vector< double > vecmultiply (double scalar, std::vector< double > vec)

    Multiplies a scalar by a vector.

- static std::vector< double > crossprod (std::vector< double > vec1, std::vector< double > vec2)

  Computes vector product.
- static std::vector< double > matrix_multiply (const std::vector< std::vector< double > > &A, const std↩ ::vector< double > &x)

  Multiplies matrix A by vector x.
- static int getOpposite (int direction)

  Gets the opposite lattice direction to the one supplied.
- static void getGrid (GridObj ∗&Grids, int level, int region, GridObj ∗&ptr)

  Get a pointer to a given grid in the hierarchy.
- static bool isOverlapPeriodic (int i, int j, int k, const GridObj &pGrid)

  Finds out whether halo containng i,j,k links to neighbour rank periodically.
- static bool isOnThisRank (double x, double y, double z, eLocationOnRank loc=eNone, const GridObj ∗grid=nullptr, std::vector< int > ∗pos=nullptr)

  Finds out whether site with supplied position is on the current rank.
- static bool isOnThisRank (double xyz, eCartesianDirection dir, eLocationOnRank loc=eNone, const GridObj ∗grid=nullptr, int ∗pos=nullptr)

  Finds out whether the supplied position can be found on the current rank.
- static bool intersectsRefinedRegion (const GridObj &pGrid, int RegNum)

  Finds out whether all or part of specified refined region intersects with the space occupied by the grid provided.
- static bool isOnSenderLayer (double pos_x, double pos_y, double pos_z)

  Check whether site is on an inner (sender) halo.
- static bool isOnRecvLayer (double pos_x, double pos_y, double pos_z)

  Check whether site is on an outer (receiver) halo.
- static bool isOnSenderLayer (double site_position, eCartMinMax edge)

  Check whether site is on an inner (sender) halo.
- static bool isOnRecvLayer (double site_position, eCartMinMax edge)

  Check whether site is on an outer (receiver) halo.
- static int getMpiDirection (int offset_vector[ ])

  Get direction in MPI topology from unit vector.
- static bool isOffGrid (int i, int j, int k, const GridObj ∗g)

  Tests whether a site is on a given grid.
- static void getEnclosingVoxel (double x, double y, double z, const GridObj ∗g, std::vector< int > ∗ijk)

  Get local voxel indices on grid in which provided position lies.
- static void getEnclosingVoxel (double x, const GridObj ∗g, eCartesianDirection dir, int ∗ijk)

  Get local voxel indices on grid in which provided position lies.
- static bool isOnTransitionLayer (double pos_x, double pos_y, double pos_z, const GridObj ∗grid)

  Check whether site is on a TL.
- static bool isOnTransitionLayer (double position, eCartMinMax edge, const GridObj ∗grid)

  Check whether site is on a specific TL (to upper).
- template<typename NumType >
  static NumType vecnorm (NumType a1, NumType a2, NumType a3)

  Computes the L2-norm.
- template<typename NumType >
  static NumType vecnorm (NumType a1, NumType a2)

  Computes the L2-norm.
- template<typename NumType >
  static NumType upToZero (NumType x)

  Rounds a negative value up to zero.
- template<typename NumType >
  static NumType downToLimit (NumType x, NumType limit)

  Rounds a value greater than a limit down to this value.

- template<typename NumType >
  static NumType factorial (NumType n)

  Computes the factorial of the supplied value.
- template<typename NumType >
  static void stridedCopy (NumType *dest, NumType *src, size_t block, size_t offset, size_t stride, size_t count, size_t buf_offset=0)

  Performs a strided memcpy.

Static Public Attributes

- static std::ofstream * logfile

  Handle to output file.
- static std::string path_str

  Static string representing output path.
- static const int dir_reflect [L_DIMS *2][L_NUM_VELS]

  Array with hardcoded direction numbering for specular reflection.

5.7.1 Detailed Description

Grid utility class.

Class provides grid utilities including commonly used logical tests. This is a static class and so there is no need to instantiate it.

5.7.2 Member Function Documentation

5.7.2.1 std::vector< double > GridUtils::add (std::vector< double > *a*, std::vector< double > *b*) [static]

Adds two vectors.

Parameters

a	a vector.
b	a second vector.

Returns

vector which is a + b.

5.7.2.2 void GridUtils::createOutputDirectory (std::string *path_str*) [static]

Create output directory.

Compatible with both Windows and Linux. Filename and path passed as a single string. Returns nothing at the moment.

Parameters

path_str	full path and filename as string.

Returns

indicator of status of action.

5.7.2.3 std::vector< double > GridUtils::crossprod (std::vector< double > *a*, std::vector< double > *b*) [static]

Computes vector product.

Parameters

a	a vector.
b	a second vector.

Returns

a vector which is the cross product of a and b.

5.7.2.4 double GridUtils::dotprod (std::vector< double > *vec1*, std::vector< double > *vec2*) [static]

Computes the scalar product of two vectors.

Parameters

vec1	a vector.
vec2	a second vector.

Returns

the dot product of the two vectors.

5.7.2.5 template<typename NumType > static NumType GridUtils::downToLimit (NumType *x*, NumType *limit*) [inline], [static]

Rounds a value greater than a limit down to this value.

If value is less than or equal to the limit, return the value unchanged.

Parameters

x	value to be rounded
limit	value to be rounded down to

Returns

NumType rounded value

5.7.2.6 template< typename NumType > static NumType GridUtils::factorial (NumType *n*) [inline], [static]

Computes the factorial of the supplied value.

If n == 0 then returns 1.

Parameters

n	factorial

Returns

NumType n factorial

5.7.2.7 std::vector< int > GridUtils::getCoarseIndices (int *fine_i*, int *x_start*, int *fine_j*, int *y_start*, int *fine_k*, int *z_start*) [static]

Gets the indices of the coarse site given the fine site.

Maps the indices of a fine grid site to a corresponding coarse site on the level above.

Parameters

fine_i	local i-index of fine site to be mapped.
x_start	local x-index of start of refined region on the grid above.
fine_j	local j-index of fine site to be mapped.
y_start	local y-index of start of refined region on the grid above.
fine_k	local k-index of fine site to be mapped.
z_start	local z-index of start of refined region on the grid above.

Returns

local indices of the coarse grid site.

5.7.2.8 void GridUtils::getEnclosingVoxel (double *x*, double *y*, double *z*, const GridObj * *g*, std::vector< int > * *ijk*) [static]

Get local voxel indices on grid in which provided position lies.

Wrapper for the overload which concentates all check into a vector.

Parameters

x	x-position.
y	y-position.
z	z-position.
g	lattice on which to look for enclosing voxel.
ijk	pointer to vector where indices are to be placed.

5.7.2.9 void GridUtils::getEnclosingVoxel (double *xyz*, const GridObj ∗ *g*, eCartesianDirection *dir*, int ∗ *ijk*) [static]

Get local voxel indices on grid in which provided position lies.

Will return the 1D voxel index of the voxel on the lattice provided within which point with position (xyz) lies. This is done by rounding the position to obtain how many voxels in from the grid core edge it is, then accounting for whether the grid starts on another rank, in the halo, or further into the grid by offsetting the original index by this amount. This approach saves expensive seraches of the position vectors on each grid. This method can be used as a position -> voxel converter. The index may be off grid so it is advisable to call isOnThisRank instead.

Parameters

xyz	x, y or z-position.
g	lattice on which to look for enclosing voxel.
dir	1D direction.
ijk	pointer to local index storage location.

5.7.2.10 std::vector< int > GridUtils::getFineIndices (int *coarse_i*, int *x_start*, int *coarse_j*, int *y_start*, int *coarse_k*, int *z_start*) [static]

Gets the indices of the fine site given the coarse site.

Maps the indices of a coarse grid site to a corresponding fine site on the level below.

Parameters

coarse←↩ _i	local i-index of coarse site to be mapped.
x_start	local x-index of start of refined region.
coarse←↩ _j	local j-index of coarse site to be mapped.
y_start	local y-index of start of refined region.
coarse←↩ _k	local k-index of coarse site to be mapped.
z_start	local z-index of start of refined region.

Returns

local indices of the fine grid site.

5.7.2.11 void GridUtils::getGrid (GridObj ∗& *Grids,* int *level,* int *region,* GridObj ∗& *ptr*) [static]

Get a pointer to a given grid in the hierarchy.

Takes a NULL pointer by reference and updates it when matching grid is found in hierarchy on this rank. If grid not found, pointer is returned without change and stays NULL. Can be used to test for the existence of a grid on a rank by passing in a NULL pointer and checking if a NULL pointer is returned.

Parameters

	Grids	x-position of site.
	level	y-position of site.
	region	z-position of site.
out	*ptr*	pointer containing address of grid in hierarchy.

5.7.2.12 int GridUtils::getMpiDirection (int *offset_vector[ ]*) [static]

Get direction in MPI topology from unit vector.

Parameters

offset_vector	unit vector pointing away from current rank.

Returns

MPI direction.

5.7.2.13 int GridUtils::getOpposite (int *direction*) [static]

Gets the opposite lattice direction to the one supplied.

This is model independent as long as the model directions are specified such that the oppoiste direction is either one vector on or one vector back in the listing depending on whether the direction supplied is even or odd.

Parameters

direction	direction to be reversed.

Returns

opposite direction in lattice model.

5.7.2.14 bool GridUtils::intersectsRefinedRegion (const GridObj & *pGrid,* int *RegNum*) [static]

Finds out whether all or part of specified refined region intersects with the space occupied by the grid provided.

Prinicpal use is for sub-grid initialisation to determine whether a sub-grid needs adding or not. This decision is made based on whether any part of the grid is covered by the discrete voxels of existing grids on the rank.

Parameters

pGrid	parent grid at appropriate level.
RegNum	region number desired.

Returns

boolean answer.

5.7.2.15 bool GridUtils::isOffGrid (int *i*, int *j*, int *k*, const GridObj ∗ *g*) [static]

Tests whether a site is on a given grid.

Parameters

i	local i-index.
j	local j-index.
k	local k-index.
g	grid on which to check.

Returns

boolean answer.

5.7.2.16 bool GridUtils::isOnRecvLayer (double *pos_x*, double *pos_y*, double *pos_z*) [static]

Check whether site is on an outer (receiver) halo.

Wrapper which checks every halo region of the rank for intersection with supplied site position.

Parameters

pos_x	x-position of site.
pos_y	y-position of site.
pos_z	z-position of site.

Returns

boolean answer.

5.7.2.17 bool GridUtils::isOnRecvLayer (double *site_position*, eCartMinMax *edge*) [static]

Check whether site is on an outer (receiver) halo.

Wrapper available which checks every halo. This method only checks the halo specified by the Cartesian direction and whether it is the left/bottom/front (minimum) or right/top/back (maximum) edge of the block.

Parameters

site_position	position of site.
edge	combination of cartesian direction and choice of edge.

Returns

boolean answer.

5.7.2.18 bool GridUtils::isOnSenderLayer (double *pos_x*, double *pos_y*, double *pos_z*) [static]

Check whether site is on an inner (sender) halo.

Wrapper which checks every halo region of the rank for intersection with supplied site position.

Parameters

pos_x	x-position of site.
pos_y	y-position of site.
pos_z	z-position of site.

Returns

boolean answer.

5.7.2.19 bool GridUtils::isOnSenderLayer (double *site_position*, eCartMinMax *edge*) [static]

Check whether site is on an inner (sender) halo.

Wrapper available which checks every halo. This method only checks the halo specified by the Cartesian direction and whether it is the left/bottom/front (minimum) or right/top/back (maximum) edge of the block.

Parameters

site_position	position of site.
edge	combination of cartesian direction and choice of edge.

Returns

boolean answer.

5.7.2.20 bool GridUtils::isOnThisRank (double *x*, double *y*, double *z*, eLocationOnRank *loc* = eNone, const GridObj ∗ *grid* = `nullptr`, std::vector< int > ∗ *pos* = `nullptr`) `[static]`

Finds out whether site with supplied position is on the current rank.

Will return true if the site is in the halo as well (send or recv). Location information provided to indicate where point is. Returns eNone enumeration if not request or if query is false. If a grid is supplied, will only return true if site is on the grid supplied. If you want to exclude the sites that belong to the halo you can call isOnRecvLayer() or isOnSenderLayer() on the same site.

Parameters

	x	x-position of site.
	y	y-position of site.
	z	z-position of site.
`out`	*pos*	pointer to the start of a vector in which local indices are returned.
	grid	grid being queried.
`out`	*loc*	description of the location of the point.

Returns

boolean answer.

5.7.2.21 bool GridUtils::isOnThisRank (double *xyz*, eCartesianDirection *dir*, eLocationOnRank *loc* = eNone, const GridObj ∗ *grid* = `nullptr`, int ∗ *pos* = `nullptr`) `[static]`

Finds out whether the supplied position can be found on the current rank.

Direction-specific version of the overload.

Parameters

	xyz	position (x, y or z)
	dir	cartesian direction of interest (x, y or z).
`out`	*loc*	description of the location of the point.
	grid	grid being queried.
`out`	*pos*	the local index of the found site.

Returns

boolean answer.

5.7.2.22 bool GridUtils::isOnTransitionLayer (double *pos_x*, double *pos_y*, double *pos_z*, const GridObj ∗ *grid*) `[static]`

Check whether site is on a TL.

Wrapper which checks every possible TL location on the grid supplied.

Parameters

pos←_x	x-position of site.
pos←_y	y-position of site.
pos←_z	z-position of site.
grid	given grid on which to check.

Returns

boolean answer.

5.7.2.23 bool GridUtils::isOnTransitionLayer (double *position*, **eCartMinMax** *edge*, const **GridObj** ∗ *grid*) [static]

Check whether site is on a specific TL (to upper).

Wrapper available which checks every TL. This method only checks the TL specified by the Cartesian direction and whether it is the left/bottom/front (minimum) or right/top/back (maximum) edge of the supplied grid.

Parameters

position	position of point.
edge	combination of cartesian direction and choice of edge.
grid	given grid on which to check.

Returns

boolean answer.

5.7.2.24 bool GridUtils::isOverlapPeriodic (int *i*, int *j*, int *k*, const **GridObj** & *g*) [static]

Finds out whether halo containng i,j,k links to neighbour rank periodically.

Checks the receiver layer containing local site i,j,k and determines from the MPI topology information whether this layer couples to an adjacent or periodic neighbour rank. I.e. if the neighbour is physically next to the rank or whether it is actaully at the other side of the domain.

Parameters

i	local i-index of recv layer site being queried.
j	local j-index of recv layer site being queried.
k	local k-index of recv layer site being queried.
g	grid on which point being queried resides.

Returns

boolean answer.

5.7.2.25 std::vector< double > GridUtils::linspace (double *min*, double *max*, int *n*) [static]

Creates a linearly-spaced vector of values.

Parameters

min	starting value of output vector.
max	ending point of output vector.
n	number of values in output vector.

Returns

a vector with n uniformly spaced values between min and max.

5.7.2.26 std::vector< double > GridUtils::matrix_multiply (const std::vector< std::vector< double > > & *A*, const std::vector< double > & *x*) [static]

Multiplies matrix A by vector x.

Parameters

A	a matrix represented as a vector or vectors.
x	a vector.

Returns

a vector which is A $*$ x.

5.7.2.27 std::vector< int > GridUtils::onespace (int *min*, int *max*) [static]

Creates a linearly-spaced vector of integers.

Parameters

min	starting value of output vector.
max	ending point of output vector.

Returns

a vector with uniformly spaced integer values between min and max.

5.7.2.28 template< typename NumType > static void GridUtils::stridedCopy (NumType * *dest*, NumType * *src*, size_t *block*, size_t *offset*, size_t *stride*, size_t *count*, size_t *buf_offset* = 0) [inline], [static]

Performs a strided memcpy.

Memcpy() is designed to copy blocks of contiguous memory. Strided copy copies a pattern of contiguous blocks.

Parameters

dest	pointer to start of destination memory.
src	pointer to start of source memory.
block	size of contiguous block.
offset	offset from the start of the soruce array.
stride	number of elements between start of first block and start of second.
count	number of blocks in pattern
buf_offset	offset from start of destination buffer to start writing. Default is zero if not supplied.

5.7.2.29 std::vector< double > GridUtils::subtract (std::vector< double > *a*, std::vector< double > *b*) [static]

Subtracts two vectors.

Parameters

a	a vector.
b	a second vector.

Returns

a vector which is a - b.

5.7.2.30 template< typename NumType > static NumType GridUtils::upToZero (NumType *x*) [inline], [static]

Rounds a negative value up to zero.

If value is positive, return the value unchanged.

Parameters

x	value to be rounded

Returns

NumType rounded value

5.7.2.31 std::vector< double > GridUtils::vecmultiply (double *scalar*, std::vector< double > *vec*) [static]

Multiplies a scalar by a vector.

Parameters

scalar	a scalar double.
vec	a vector double.

Returns

a vector which is a scalar multiplied by a vector.

5.7.2.32 double GridUtils::vecnorm (double *vec[L_DIMS]*) [static]

Computes the L2 norm using the vector supplied.

Parameters

vec	old-style C array representing a vector with the same number of number of components as the problem dimension.

Returns

the L2 norm.

5.7.2.33 double GridUtils::vecnorm (double *val1,* double *val2*) [static]

Computes the L2 norm using the vector components supplied.

Parameters

val1	first vector component.
val2	second vector component.

Returns

the L2 norm.

5.7.2.34 double GridUtils::vecnorm (double *val1,* double *val2,* double *val3*) [static]

Computes the L2 norm using the vector components supplied.

Parameters

val1	first vector component.
val2	second vector component.
val3	third vector component.

Returns

the L2 norm.

5.7.2.35 double GridUtils::vecnorm (std::vector< double > *vec*) [static]

Computes the L2 norm using the vector supplied.

Parameters

vec	C++ std::vector.

Returns

the L2 norm.

5.7.2.36 template<typename NumType > static NumType GridUtils::vecnorm (NumType *a1*, NumType *a2*, NumType *a3*) [inline], [static]

Computes the L2-norm.

Parameters

a1	first component of the vector
a2	second component of the vector
a3	third component of the vector

Returns

NumType scalar quantity

5.7.2.37 template<typename NumType > static NumType GridUtils::vecnorm (NumType *a1*, NumType *a2*) [inline], [static]

Computes the L2-norm.

Parameters

a1	first component of the vector
a2	second component of the vector

Returns

NumType scalar quantity

5.7.3 Member Data Documentation

5.7.3.1 const int GridUtils::dir_reflect [static]

Initial value:

```
=
    {
        {1, 0, 2, 3, 4, 5, 9, 8, 7, 6, 10, 11, 12, 13, 16, 17, 14, 15, 18},
        {1, 0, 2, 3, 4, 5, 9, 8, 7, 6, 10, 11, 12, 13, 16, 17, 14, 15, 18},
        {0, 1, 3, 2, 4, 5, 8, 9, 6, 7, 13, 12, 11, 10, 14, 15, 16, 17, 18},
        {0, 1, 3, 2, 4, 5, 8, 9, 6, 7, 13, 12, 11, 10, 14, 15, 16, 17, 18},
        {0, 1, 2, 3, 5, 4, 6, 7, 8, 9, 12, 13, 10, 11, 17, 16, 15, 14, 18},
        {0, 1, 2, 3, 5, 4, 6, 7, 8, 9, 12, 13, 10, 11, 17, 16, 15, 14, 18}
    }
```

Array with hardcoded direction numbering for specular reflection.

5.7.3.2 std::ofstream * GridUtils::logfile [static]

Handle to output file.

5.7.3.3 std::string GridUtils::path_str [static]

Static string representing output path.

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

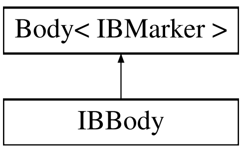
- GridUtils.h
- GridObj.cpp
- GridUtils.cpp
- main_lbm.cpp

5.8 IBBody Class Reference

Immersed boundary body.

`#include <IBBody.h>`

Inheritance diagram for IBBody:



Public Member Functions

- IBBody (void)

  Constructor which sets group ID to zero by default.
- ∼IBBody (void)

  Default destructor.
- IBBody (GridObj ∗g, size_t id)

  Constructor which assigns the owner grid.
- void addMarker (double x, double y, double z, bool flex_rigid)

  Method to add an IB marker to the body.
- virtual void markerAdder (double x, double y, double z, int &curr_mark, std::vector< int > &counter, bool flex_rigid)

  Downsampling marker adding method (overload)
- void makeBody (double radius, std::vector< double > centre, bool flex_rigid, bool moving, int group)

  Method to seed markers for a sphere / circle.
- void makeBody (std::vector< double > width_length_depth, std::vector< double > angles, std::vector< double > centre, bool flex_rigid, bool deform, int group)

  Method to seed markers for a cuboid / rectangle.
- void makeBody (int numbermarkers, std::vector< double > start_point, double fil_length, std::vector< double > angles, std::vector< int > BCs, bool flex_rigid, bool deform, int group)

  Method to seed markers for a flexible filament.
- double makeBody (std::vector< double > width_length, double angle, std::vector< double > centre, bool flex_rigid, bool deform, int group, bool plate)

  Method to seed markers for a 3D plate inclined from the XZ plane.
- void makeBody (PCpts ∗_PCpts)

  Method to build a body from a point cloud.

Protected Attributes

- bool flex_rigid

  Flag to indicate flexibility: false == rigid body; true == flexible filament.
- bool deformable

  Flag to indicate deformable body: false == rigid; true == deformable.
- int groupID

  ID of IBbody group – position updates can be driven from a flexible body in a group.
- double delta_rho

  Difference in density between fluid and solid in lattice units.
- double flexural_rigidity

  Young's modulus E ∗ Second moment of area I.
- std::vector< double > tension

  Tension between the current marker and its neighbour.
- std::vector< int > BCs

  BCs type flags (flexible bodies)

Friends

- class ObjectManager

Additional Inherited Members

5.8.1 Detailed Description

Immersed boundary body.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 IBBody::IBBody (void)

Constructor which sets group ID to zero by default.

5.8.2.2 IBBody::∼IBBody (void)

Default destructor.

5.8.2.3 IBBody::IBBody (GridObj * *g*, size_t *id*)

Constructor which assigns the owner grid.

Also sets the group ID to zero.

Parameters

g	pointer to owner grid
id	ID of body in array of bodies.

5.8.3 Member Function Documentation

5.8.3.1 void IBBody::addMarker (double *x*, double *y*, double *z*, bool *flex_rigid*)

Method to add an IB marker to the body.

Adds marker at the given position with the given moving/non-moving flag.

Parameters

x	global x-position of marker.
y	global y-position of marker.
z	global z-position of marker.
flex_rigid	flag to indicate whether marker is movable or not.

5.8.3.2 void IBBody::makeBody (double *radius*, std::vector< double > *centre*, bool *flex_rigid*, bool *deform*, int *group*)

Method to seed markers for a sphere / circle.

Parameters

radius	radius of circle/sphere.
centre	position vector of circle/sphere centre.
flex_rigid	flag to indicate whether body is flexible and requires a structural calculation.
deform	flag to indicate whether body is movable and requires relocation each time step.
group	ID indicating which group the body is part of for collective operations.

5.8.3.3 void IBBody::makeBody (std::vector< double > *width_length_depth*, std::vector< double > *angles*, std::vector< double > *centre*, bool *flex_rigid*, bool *deform*, int *group*)

Method to seed markers for a cuboid / rectangle.

Parameters

width_length_depth	principal dimensions of cuboid / rectangle.
angles	principal orientation of cuboid / rectangle w.r.t. domain axes.
centre	position vector of cuboid / rectangle centre.
flex_rigid	flag to indicate whether body is flexible and requires a structural calculation.
deform	flag to indicate whether body is movable and requires relocation each time step.
group	ID indicating which group the body is part of for collective operations.

5.8.3.4 void IBBody::makeBody (int *nummarkers*, std::vector< double > *start_point*, double *fil_length*, std::vector< double > *angles*, std::vector< int > *BCs*, bool *flex_rigid*, bool *deform*, int *group*)

Method to seed markers for a flexible filament.

Parameters

nummarkers	number of markers to use for filament.
start_point	3D position vector of the start of the filament.
fil_length	length of filament in physical units.
angles	two angles representing filament inclination w.r.t. domain axes (horizontal plane and vertical plane).
BCs	vector containing start and end boundary condition types (see class definition for valid values).
flex_rigid	flag to indicate whether body is flexible and requires a structural calculation.
deform	flag to indicate whether body is movable and requires relocation each time step.
group	ID indicating which group the body is part of for collective operations.

5.8.3.5 double IBBody::makeBody (std::vector< double > *width_length*, double *angle*, std::vector< double > *centre*, bool *flex_rigid*, bool *deform*, int *group*, bool *plate*)

Method to seed markers for a 3D plate inclined from the XZ plane.

Parameters

width_length	2D vector of principal dimensions of thin plate.
angle	inclination angle from horizontal.
centre	position vector of the plate centre.
flex_rigid	flag to indicate whether body is flexible and requires a structural calculation.
deform	flag to indicate whether body is movable and requires relocation each time step.
group	ID indicating which group the body is part of for collective operations.
plate	arbitrary argument to allow overload otherwise would have the same signature as a filament builder.

5.8.3.6 void IBBody::makeBody (PCpts * *_PCpts*)

Method to build a body from a point cloud.

Flexibility and deformable properties taken from definitions.

Parameters

_PCpts	pointer to pointer cloud data.

5.8.3.7 void IBBody::markerAdder (double *x*, double *y*, double *z*, int & *curr_mark*, std::vector< int > & *counter*, bool *flex_rigid*) [virtual]

Downsampling marker adding method (overload)

This method is an overload of the method in the parent class. This version takes the flexible/rigid flag and passes it to the overloaded addMarker() method.

Parameters

x	desired global X-position of new marker.
y	desired globalY-position of new marker.
z	desired globalZ-position of new marker.
curr_mark	is a reference to the ID of last marker.
counter	is a reference to the total number of markers in the body.
flex_rigid	indicates whether markers added should form part of flexible or rigid body.

5.8.4 Friends And Related Function Documentation

5.8.4.1 friend class ObjectManager [friend]

5.8.5 Member Data Documentation

5.8.5.1 std::vector<int> IBBody::BCs [protected]

BCs type flags (flexible bodies)

5.8.5.2 bool IBBody::deformable [protected]

Flag to indicate deformable body: false == rigid; true == deformable.

5.8.5.3 double IBBody::delta_rho [protected]

Difference in density between fluid and solid in lattice units.

5.8.5.4 bool IBBody::flex_rigid [protected]

Flag to indicate flexibility: false == rigid body; true == flexible filament.

5.8.5.5 double IBBody::flexural_rigidity [protected]

Young's modulus E $*$ Second moment of area I.

5.8.5.6 int IBBody::groupID [protected]

ID of IBbody group – position updates can be driven from a flexible body in a group.

5.8.5.7 std::vector<double> IBBody::tension [protected]

Tension between the current marker and its neighbour.

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

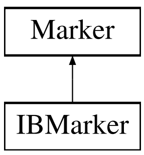
- IBBody.h
- IBBody.cpp

5.9 IBMarker Class Reference

Immersed boundary marker.

`#include <IBMarker.h>`

Inheritance diagram for IBMarker:



Public Member Functions

- IBMarker (void)

  Default constructor.
- ∼IBMarker (void)

  Default destructor.
- IBMarker (double xPos, double yPos, double zPos, bool flex_rigid=false)

  Custom constructor with position.

Protected Attributes

- std::vector< double > fluid_vel

  Fluid velocity interpolated from lattice nodes.
- std::vector< double > desired_vel

  Desired velocity at marker.
- std::vector< double > force_xyz

  Restorative force vector on marker.
- std::vector< double > position_old

  Vector containing the physical coordinates (x,y,z) of the marker at t-1. Used for moving bodies.
- std::vector< double > deltaval

  Value of delta function for a given support node.
- bool flex_rigid

  Indication as to whether marker is part of a moving or flexible body: false == rigid/fixed; true == flexible/moving.
- double epsilon

  Scaling parameter.
- double local_area

  Area associated with support node in lattice units (same for all points if from same grid and regularly spaced like LBM)
- double dilation

  Dilation parameter in lattice units (same in all directions for uniform Eulerian grid)

Friends

- class ObjectManager
- class IBBody

Additional Inherited Members

5.9.1 Detailed Description

Immersed boundary marker.

This class declaration is for an immersed boundary Lagrange point. A collection of these points form an immersed boundary body.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 IBMarker::IBMarker (void) [inline]

Default constructor.

5.9.2.2 IBMarker::∼IBMarker (void) [inline]

Default destructor.

5.9.2.3 IBMarker::IBMarker (double *xPos*, double *yPos*, double *zPos*, bool *flex_rigid* = false)

Custom constructor with position.

Parameters

xPos	x-position of marker.
yPos	y-position of marker.
zPos	z-position of marker.
flex_rigid	flag to indicate whether marker is movable or not.

5.9.3 Friends And Related Function Documentation

5.9.3.1 friend class IBBody [friend]

5.9.3.2 friend class ObjectManager [friend]

5.9.4 Member Data Documentation

5.9.4.1 std::vector<double> IBMarker::deltaval [protected]

Value of delta function for a given support node.

5.9.4.2 std::vector<double> IBMarker::desired_vel [protected]

Desired velocity at marker.

5.9.4.3 double IBMarker::dilation [protected]

Dilation parameter in lattice units (same in all directions for uniform Eulerian grid)

5.9.4.4 double IBMarker::epsilon [protected]

Scaling parameter.

5.9.4.5 bool IBMarker::flex_rigid [protected]

Indication as to whether marker is part of a moving or flexible body: false == rigid/fixed; true == flexible/moving.

5.9.4.6 std::vector<double> IBMarker::fluid_vel [protected]

Fluid velocity interpolated from lattice nodes.

5.9.4.7 std::vector<double> IBMarker::force_xyz [protected]

Restorative force vector on marker.

5.9.4.8 double IBMarker::local_area [protected]

Area associated with support node in lattice units (same for all points if from same grid and regularly spaced like LBM)

5.9.4.9 std::vector<double> IBMarker::position_old [protected]

Vector containing the physical coordinates (x,y,z) of the marker at t-1. Used for moving bodies.

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

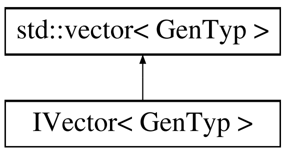
- IBMarker.h
- IBMarker.cpp

5.10 IVector< GenTyp > Class Template Reference

Index-collapsing vector class.

`#include <IVector.h>`

Inheritance diagram for IVector< GenTyp >:



Public Member Functions

- IVector ()
  Default constructor.
- ∼IVector ()
  Default destructor.
- IVector (size_t size, GenTyp val)
  Custom constructor taking type and value.
- GenTyp & operator() (size_t i, size_t j, size_t k, size_t v, size_t j_max, size_t k_max, size_t v_max)
  4D array index flatten.
- GenTyp & operator() (size_t i, size_t j, size_t k, size_t j_max, size_t k_max)
  3D array index flatten.
- GenTyp & operator() (size_t i, size_t j, size_t j_max)
  2D array index flatten.

5.10.1 Detailed Description

template<typename GenTyp>
class IVector< GenTyp >

Index-collapsing vector class.

This class has all the behaviour of std::vector but has a overriden operator() to allow automatic flattening of indices before returning a reference of value at indexed location. Needs to be able to accept different datatypes so templated.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 template<typename GenTyp> IVector< GenTyp >::IVector ( ) [inline]

Default constructor.

5.10.2.2 template<typename GenTyp> IVector< GenTyp >::∼IVector ( ) [inline]

Default destructor.

5.10.2.3 template<typename GenTyp> IVector< GenTyp >::IVector (size_t *size*, GenTyp *val*) [inline]

Custom constructor taking type and value.

Parameters

size	the desired size of vector
val	the value to fill the new vector with

5.10.3 Member Function Documentation

5.10.3.1 template<typename GenTyp> GenTyp& IVector< GenTyp >::operator() (size_t *i*, size_t *j*, size_t *k*, size_t *v*, size_t *j_max*, size_t *k_max*, size_t *v_max*) [inline]

4D array index flatten.

Override of parentheses to auto-flatten indices to a single index.

Parameters

i	the i index
j	the j index
k	the k index
v	the index in the fourth dimension
j_max	the number of j elements
k_max	the number of k elements
v_max	the number of elements in the fourth dimension

Returns

GenTyp& a reference to the value at this position in the vector

5.10.3.2 template<typename GenTyp> GenTyp& IVector< GenTyp >::operator() (size_t *i*, size_t *j*, size_t *k*, size_t *j_max*, size_t *k_max*) [inline]

3D array index flatten.

Override of parentheses to auto-flatten indices to a single index.

Parameters

i	the i index
j	the j index
k	the k index
j_max	the number of j elements
k_max	the number of k elements

Returns

GenTyp& a reference to the value at this position in the vector

5.10.3.3 template<typename GenTyp> GenTyp& IVector< GenTyp >::operator() (size_t *i*, size_t *j*, size_t *j_max*) [inline]

2D array index flatten.

Parameters

i	the i index
j	the j index
j_max	the number of j elements

Returns

GenTyp& a reference to the value at this position in the vector

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

- IVector.h

5.11 MpiManager::layer_edges Struct Reference

Structure containing absolute positions of the edges of halos.

```
#include <MpiManager.h>
```

Public Attributes

- double X [4]

    X limits.
- double Y [4]

    Y limits.
- double Z [4]

    Z limits.

5.11.1 Detailed Description

Structure containing absolute positions of the edges of halos.

Sender (inner) and receiver (outer) parts of halo are located using the convention [left_min left_max right_min right_max] for X and similar for Y and Z. Access using the enumerator eEdgeMinMax.

5.11.2 Member Data Documentation

5.11.2.1 double MpiManager::layer_edges::X[4]

X limits.

5.11.2.2 double MpiManager::layer_edges::Y[4]

Y limits.

5.11.2.3 double MpiManager::layer_edges::Z[4]

Z limits.

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

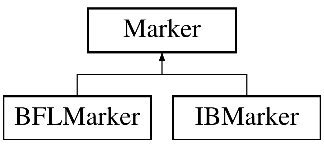
- MpiManager.h

5.12 Marker Class Reference

Generic marker class.

```
#include <Marker.h>
```

Inheritance diagram for Marker:



Public Member Functions

- Marker (void)

    Default constructor.
- ∼Marker (void)

    Default destructor.
- Marker (double x, double y, double z)

    Custom constructor which locates marker.

Public Attributes

- std::vector< double > position

    Position vector of marker location in physical units.
- std::vector< int > supp_i

    X-indices of lattice sites in support of this marker.
- std::vector< int > supp_j

    Y-indices of lattice sites in support of this marker.
- std::vector< int > supp_k

    Z-indices of lattice sites in support of this marker.
- std::vector< int > support_rank

    Array of indices indicating on which rank the given support point resides.

5.12.1 Detailed Description

Generic marker class.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Marker::Marker (void) [inline]

Default constructor.

5.12.2.2 Marker::∼Marker (void) [inline]

Default destructor.

5.12.2.3 Marker::Marker (double *x*, double *y*, double *z*) [inline]

Custom constructor which locates marker.

Parameters

x	X-position of marker in physical units
y	Y-position of marker in physical units
z	Z-position of marker in physical units

5.12.3 Member Data Documentation

5.12.3.1 std::vector<double> Marker::position

Position vector of marker location in physical units.

5.12.3.2 std::vector<int> Marker::supp_i

X-indices of lattice sites in support of this marker.

5.12.3.3 std::vector<int> Marker::supp_j

Y-indices of lattice sites in support of this marker.

5.12.3.4 std::vector<int> Marker::supp_k

Z-indices of lattice sites in support of this marker.

5.12.3.5 std::vector<int> Marker::support_rank

Array of indices indicating on which rank the given support point resides.

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

- Marker.h

5.13 MarkerData Class Reference

Container class to hold marker information.

```
#include <Body.h>
```

Public Member Functions

- MarkerData (int i, int j, int k, double x, double y, double z, int ID)

  Constructor.
- MarkerData (void)

  Default Constructor.
- ∼MarkerData (void)

  Default destructor.

Public Attributes

- int i
  i-index of primary support site
- int j
  j-index of primary support site
- int k
  k-index of primary support site
- int ID
  Marker ID (position in array of markers)
- double x
  x-position of marker
- double y
  y-position of marker
- double z
  z-position of marker

5.13.1 Detailed Description

Container class to hold marker information.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 MarkerData::MarkerData (int *i*, int *j*, int *k*, double *x*, double *y*, double *z*, int *ID*) [inline]

Constructor.

Parameters

i	i-index of primary support site
j	j-index of primary support site
k	k-index of primary support site
x	x-position of marker
y	y-position of marker
z	z-position of marker
ID	marker number in a given body

5.13.2.2 MarkerData::MarkerData (void) [inline]

Default Constructor.

Initialise with invalid marker indicator which is to set the x position to NaN.

5.13.2.3 MarkerData::∼MarkerData (void) [inline]

Default destructor.

5.13.3 Member Data Documentation

5.13.3.1 int MarkerData::i

i-index of primary support site

5.13.3.2 int MarkerData::ID

Marker ID (position in array of markers)

5.13.3.3 int MarkerData::j

j-index of primary support site

5.13.3.4 int MarkerData::k

k-index of primary support site

5.13.3.5 double MarkerData::x

x-position of marker

5.13.3.6 double MarkerData::y

y-position of marker

5.13.3.7 double MarkerData::z

z-position of marker

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

- Body.h

5.14 MpiManager Class Reference

MPI Manager class.

```
#include <MpiManager.h>
```

Classes

- struct buffer_struct

  Structure storing buffers sizes in each direction for particular grid.
- struct layer_edges

  Structure containing absolute positions of the edges of halos.
- struct phdf5_struct

  Structure for storing halo information for HDF5.

Public Member Functions

- void mpi_init ()

  Initialisation routine.
- void mpi_gridbuild ()

  Domain decomposition.
- int mpi_buildCommunicators ()

  Define writable sub-grid communicators.
- void mpi_updateLoadInfo ()

  Update the load balancing information stored in the MpiManager.
- void mpi_buffer_pack (int dir, GridObj ∗g)

  Method to pack the communication buffer.
- void mpi_buffer_unpack (int dir, GridObj ∗g)

  Method to unpack the communication buffer.
- void mpi_buffer_size ()

  Pre-calcualtion of the buffer sizes.
- void mpi_buffer_size_send (GridObj ∗&g)

  Method to pre-compute the size of the sender layer buffer.
- void mpi_buffer_size_recv (GridObj ∗&g)

  Method to pre-compute the size of the receiver layer buffer.
- void mpi_writeout_buf (std::string filename, int dir)

  Buffer ASCII writer.
- void mpi_communicate (int level, int regnum)

  Communication routine.
- int mpi_getOpposite (int direction)

  Helper method to find opposite direction in MPI topology.

Static Public Member Functions

- static MpiManager ∗ getInstance ()

  Instance creator.
- static void destroyInstance ()

  Instance destroyer.

Public Attributes

- MPI_Comm world_comm

  Global MPI communicator.
- int dimensions [L_DIMS]

  Size of MPI Cartesian topology.
- int neighbour_rank [L_MPI_DIRS]

  Neighbour rank number for each direction in Cartesian topology.
- int neighbour_coords [L_DIMS][L_MPI_DIRS]

  Coordinates in MPI topology of neighbour ranks.
- MPI_Comm subGrid_comm [L_NUM_LEVELS ∗L_NUM_REGIONS]

  Communicators for sub-grid / region combinations.
- std::vector< phdf5_struct > p_data

  Vector of structures containing halo descriptors for block writing (HDF5)
- int my_rank

  Rank number.
- int num_ranks

  Total number of ranks in MPI Cartesian topology.
- int rank_coords [L_DIMS]

  Coordinates in MPI Cartesian topology.
- int global_size [3][L_NUM_LEVELS ∗L_NUM_REGIONS+1]

  Overall size of each grid (excluding halo of course).
- double global_edges [6][L_NUM_LEVELS ∗L_NUM_REGIONS+1]

  Absolute position of grid edges (excluding halo of course).
- bool subgrid_tlayer_key [6][L_NUM_LEVELS ∗L_NUM_REGIONS]

  Boolean flag array to indicate the presence of a TL on sub-grid edges.
- std::vector< int > local_size

  Dimensions of coarse lattice represented on this rank (includes halo).
- std::vector< std::vector< double > > rank_core_edge

  Absolute positions of edges of the core region represented on this rank.
- layer_edges sender_layer_pos

  Structure containing sender layer edge positions.
- layer_edges recv_layer_pos

  Structure containing receiver layer edge positions.
- GridObj ∗ Grids

  Pointer to grid hierarchy.
- std::vector< std::vector< double > > f_buffer_send

  Array of resizeable outgoing buffers used for data transfer.
- std::vector< std::vector< double > > f_buffer_recv

  Array of resizeable incoming buffers used for data transfer.
- MPI_Status recv_stat

  Status structure for Receive return information.
- MPI_Request send_requests [L_MPI_DIRS]

  Array of request structures for handles to posted ISends.
- MPI_Status send_stat [L_MPI_DIRS]

  Array of statuses for each Isend.
- std::vector< buffer_struct > buffer_send_info

  Vectors of buffer_info structures holding sender layer size info.
- std::vector< buffer_struct > buffer_recv_info

  Vectors of buffer_info structures holding receiver layer size info.
- std::ofstream ∗ logout

  Logfile handle.

Static Public Attributes

- static const int neighbour_vectors [3][26]

  Cartesian unit vectors pointing to each neighbour in Cartesian topology.

5.14.1 Detailed Description

MPI Manager class.

Class to manage all MPI apsects of the code.

5.14.2 Member Function Documentation

5.14.2.1 void MpiManager::destroyInstance ( ) [static]

Instance destroyer.

5.14.2.2 MpiManager ∗ MpiManager::getInstance ( ) [static]

Instance creator.

5.14.2.3 void MpiManager::mpi_buffer_pack (int *dir*, GridObj ∗ *g*)

Method to pack the communication buffer.

Communication buffer is packed with distribution values from the supplied grid. Amount of information is dictated by the direction of the communication being prepared.

Parameters

dir	communication direction.
g	grid doing the communication.

5.14.2.4 void MpiManager::mpi_buffer_size ( )

Pre-calcualtion of the buffer sizes.

Wrapper method for computing the buffer sizes for every grid on the rank, both sender and receiver. Must be called post-initialisation.

5.14.2.5 void MpiManager::mpi_buffer_size_recv (GridObj ∗& *g*)

Method to pre-compute the size of the receiver layer buffer.

A halo consists of a receiver (outer) and sender (inner) layer. This method computes the size of the receiver layers in each communication direction (MPI directions).

Parameters

g	grid being inspected.

5.14.2.6 void MpiManager::mpi_buffer_size_send (GridObj ∗& *g*)

Method to pre-compute the size of the sender layer buffer.

A halo consists of a receiver (outer) and sender (inner) layer. This method computes the size of the sender layers in each communication direction (MPI directions).

Parameters

g	grid being inspected.

5.14.2.7 void MpiManager::mpi_buffer_unpack (int *dir*, GridObj ∗ *g*)

Method to unpack the communication buffer.

Communication buffer is unpacked onto the supplied grid. Amount and region of unpacking is dictated by the direction of the communication taking place.

Parameters

dir	communication direction.
g	grid doing the communication.

5.14.2.8 int MpiManager::mpi_buildCommunicators ( )

Define writable sub-grid communicators.

When using HDF5 in parallel, collective IO operations require all processes to write a non-zero amount of data to the same file. This method examines availability of sub-grid and writable data on the grid (if found) and ensures it is added to a new communicator. Must be called AFTER the grids and buffers have been initialised.

5.14.2.9 void MpiManager::mpi_communicate (int *lev*, int *reg*)

Communication routine.

This method implements the communication between grids of the same level and region across MPI processes. Each call effects communication in all valid directions for the grid of the supplied level and region.

Parameters

lev	level of grid to communicate.
reg	region number of grid to communicate.

5.14.2.10 int MpiManager::mpi_getOpposite (int *direction*)

Helper method to find opposite direction in MPI topology.

The MPI directional vectors do not necessarily correspond to the lattice model direction. The MPI directional vectors are defined separately and hence there is a separate opposite finding method.

Parameters

direction	the outgoing direction whose opposite you wish to find.

5.14.2.11 void MpiManager::mpi_gridbuild ( )

Domain decomposition.

Method to decompose the domain and identify local grid sizes. Parameters defined here are used in GridObj construction.

5.14.2.12 void MpiManager::mpi_init ( )

Initialisation routine.

Method is responsible for initialising the MPI topolgy and associated data. Must be called immediately after MPI_↩ init(). For serial vuilds this gets called simply to intialise the MPIM with a basic set of grid information used by other methods.

5.14.2.13 void MpiManager::mpi_updateLoadInfo ( )

Update the load balancing information stored in the MpiManager.

This method is executed by all processes. Counts the ACTIVE cells on the current rank and pushes the information to the master (rank 0) which writes this information to an output file if required. Must be called after the grids have been built or will return zero.

5.14.2.14 void MpiManager::mpi_writeout_buf (std::string *filename,* int *dir*)

Buffer ASCII writer.

When verbose MPI logging is turned on this method will write out the communication buffer to an ASCII file.

5.14.3 Member Data Documentation

5.14.3.1 std::vector<buffer_struct> MpiManager::buffer_recv_info

Vectors of buffer_info structures holding receiver layer size info.

5.14.3.2 std::vector<buffer_struct> MpiManager::buffer_send_info

Vectors of buffer_info structures holding sender layer size info.

5.14.3.3 int MpiManager::dimensions[L_DIMS]

Size of MPI Cartesian topology.

5.14.3.4 std::vector< std::vector<double> > MpiManager::f_buffer_recv

Array of resizeable incoming buffers used for data transfer.

5.14.3.5 std::vector< std::vector<double> > MpiManager::f_buffer_send

Array of resizeable outgoing buffers used for data transfer.

5.14.3.6 double MpiManager::global_edges[6][L_NUM_LEVELS ∗L_NUM_REGIONS+1]

Absolute position of grid edges (excluding halo of course).

Since L0 can only be region = 0 this array should be accessed as [level + region_number ∗ L_NUM_LEVELS] in a loop where level cannot be 0. To retrieve L0 info, simply access [0]. The first index should be accessed using the enumerator eCartesianMinMax.

5.14.3.7 int MpiManager::global_size[3][L_NUM_LEVELS ∗L_NUM_REGIONS+1]

Overall size of each grid (excluding halo of course).

Since L0 can only be region = 0 this array should be accessed as [level + region_number ∗ L_NUM_LEVELS] in a loop where level cannot be 0. To retrieve L0 info, simply access [0].

5.14.3.8 GridObj∗ MpiManager::Grids

Pointer to grid hierarchy.

5.14.3.9 std::vector<int> MpiManager::local_size

Dimensions of coarse lattice represented on this rank (includes halo).

5.14.3.10 std::ofstream∗ MpiManager::logout

Logfile handle.

5.14.3.11 int MpiManager::my_rank

Rank number.

5.14.3.12 int MpiManager::neighbour_coords[L_DIMS][L_MPI_DIRS]

Coordinates in MPI topology of neighbour ranks.

5.14.3.13 int MpiManager::neighbour_rank[L_MPI_DIRS]

Neighbour rank number for each direction in Cartesian topology.

5.14.3.14 const int MpiManager::neighbour_vectors [static]

Initial value:

```
=
{
    { 1, -1, 1, -1, 0, 0, -1, 1, 0, 0, 1, -1, 1, -1, 0, 0, -1, 1, -1, 1, -1, 1, 0, 0, 1, -1 },
    { 0, 0, 1, -1, 1, -1, 1, -1, 0, 0, 0, 0, 1, -1, 1, -1, 1, -1, 0, 0, -1, 1, -1, 1, -1, 1 },
    { 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1 }
}
```

Cartesian unit vectors pointing to each neighbour in Cartesian topology.

Define 3D such that first 8 mimic the 2D ones. Opposites are simply the next or previous column in the array. MSVC 2013 does not support initialiser lists tagged onto the constructor although it is valid C++ so I have had to make it static even though it goes against the idea of the singleton design.

5.14.3.15 int MpiManager::num_ranks

Total number of ranks in MPI Cartesian topology.

5.14.3.16 std::vector<phdf5_struct> MpiManager::p_data

Vector of structures containing halo descriptors for block writing (HDF5)

5.14.3.17 int MpiManager::rank_coords[L_DIMS]

Coordinates in MPI Cartesian topology.

5.14.3.18 std::vector< std::vector<double> > MpiManager::rank_core_edge

Absolute positions of edges of the core region represented on this rank.

Excludes outer overlapping layer (recv layer). Rows are x,y,z start and end pairs and columns are rank number. Access the rows using the eCartMinMax enumeration.

5.14.3.19 layer_edges MpiManager::recv_layer_pos

Structure containing receiver layer edge positions.

5.14.3.20 MPI_Status MpiManager::recv_stat

Status structure for Receive return information.

5.14.3.21 MPI_Request MpiManager::send_requests[L_MPI_DIRS]

Array of request structures for handles to posted ISends.

5.14.3.22 MPI_Status MpiManager::send_stat[L_MPI_DIRS]

Array of statuses for each Isend.

5.14.3.23 layer_edges MpiManager::sender_layer_pos

Structure containing sender layer edge positions.

5.14.3.24 MPI_Comm MpiManager::subGrid_comm[L_NUM_LEVELS *L_NUM_REGIONS]

Communicators for sub-grid / region combinations.

5.14.3.25 bool MpiManager::subgrid_tlayer_key[6][L_NUM_LEVELS *L_NUM_REGIONS]

Boolean flag array to indicate the presence of a TL on sub-grid edges.

It is not a given that a sub-grid has a TL on every edge of the grid. Specifically if we have a sub-grid which is perodic (or in future, which merges with another sub-grid?). The HDF5 writer needs to know whether to exclude sites to account for TL or not so we store information here from the sub-grid initialisation. The first index should be accessed using the enumerator eCartesianMinMax. If no sub-grids present then adopts a default 6x1 size.

5.14.3.26 MPI_Comm MpiManager::world_comm

Global MPI communicator.

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

- MpiManager.h
- Mpi_buffer_pack.cpp
- Mpi_buffer_size_recv.cpp
- Mpi_buffer_size_send.cpp
- Mpi_buffer_unpk.cpp
- MpiManager.cpp

5.15 ObjectManager Class Reference

Object Manager class.

`#include <ObjectManager.h>`

Public Member Functions

- void ibm_apply ()

  Perform IBM procedure.
- void ibm_buildBody (int body_type)

  Builds a prefab immersed boundary body.
- void ibm_buildBody (PCpts *_PCpts, GridObj *owner)

  Wrapper for building a body from a point cloud.
- void ibm_initialise ()

  Initialise the array of iBodies.
- double ibm_deltaKernel (double rad, double dilation)

  Method to evaluate delta kernel at supplied location.
- void ibm_interpol (int ib)

  Interpolate velocity field onto markers.
- void ibm_spread (int ib)

  Spread restorative force back onto marker support.
- void ibm_findSupport (int ib, int m)

  Finds support points for iBody.
- void ibm_initialiseSupport (int ib, int m, int s, double estimated_position[ ])

  Initialise data associated with support points found.
- void ibm_computeForce (int ib)

  Compute restorative force at each marker in a body.
- double ibm_findEpsilon (int ib)

  Compute epsilon for a given iBody.
- void ibm_moveBodies ()

  Moves iBodies after applying IBM.
- double ibm_bicgstab (std::vector< std::vector< double > > &Amatrix, std::vector< double > &bVector, std↩ ::vector< double > &epsilon, double tolerance, int maxiterations)

  Biconjugate gradient method.
- void ibm_jacowire (int ib)

  Structural calculation of flexible cilia.
- void ibm_positionUpdate (int ib)

  Update the position of a deformable iBody.
- void ibm_positionUpdateGroup (int group)

  Update the position of a group of deformable iBodies.
- void ibm_banbks (double **a, long n, int m1, int m2, double **al, unsigned long indx[ ], double b[ ])

  Solution of a banded diagonal linear system.
- void ibm_bandec (double **a, long n, int m1, int m2, double **al, unsigned long indx[ ], double *d)

  LU decomposition of band diagonal matrix.
- void bfl_buildBody (int body_type)

  Prefab body building routine.
- void bfl_buildBody (PCpts *_PCpts)

  Wrapper for building BFL body from point cloud.
- void computeLiftDrag (int i, int j, int k, GridObj *g)

Compute forces on a rigid object.
- void io_vtkIBBWriter (double tval)

  Write IB body data to VTK file.
- void io_writeBodyPosition (int timestep)

  Write out position of immersed boundary bodies.
- void io_writeLiftDrag (int timestep)

  Write out forces on the markers of immersed boundary bodies.
- void io_restart (eIOFlag IO_flag, int level)

  Read/write IB body information to restart file.
- void io_readInCloud (PCpts ∗_PCpts, eObjectType objtype)

  Read in point cloud data.
- void io_writeForceOnObject (double tval)

  Write out the forces on a solid object.

Static Public Member Functions

- static ObjectManager ∗ getInstance ()

  Get instance method.
- static void destroyInstance ()

  Destroy instance method.
- static ObjectManager ∗ getInstance (GridObj ∗g)

  Overloaded get instance passing in pointer to grid hierarchy.

Friends

- class GridObj

5.15.1 Detailed Description

Object Manager class.

Class to manage all objects in the domain from creation through manipulation to destruction.

5.15.2 Member Function Documentation

5.15.2.1 void ObjectManager::bfl_buildBody (int *body_type*)

Prefab body building routine.

Not implemented in this version.

Parameters

body_type	type of prefab body to be built.

5.15.2.2 void ObjectManager::bfl_buildBody (PCpts * _PCpts)

Wrapper for building BFL body from point cloud.

Parameters

_PCpts	pointer to point cloud data.

5.15.2.3 void ObjectManager::computeLiftDrag (int *i*, int *j*, int *k*, GridObj * *g*)

Compute forces on a rigid object.

Uses momentum exchange to compute forces on rigid bodies. Currently working with bounce-back objects only. There is no bounding box so if we have walls in the domain they will be counted as well. Also only possible to differentiate between bodies. Lumps all bodies together.identify which body this site relates to so we can differentiate.

Parameters

i	local i-index of solid site.
j	local j-index of solid site.
k	local k-index of solid site.
g	pointer to grid on which object resides.

5.15.2.4 void ObjectManager::destroyInstance ( ) [static]

Destroy instance method.

Instance destuctor.

5.15.2.5 ObjectManager * ObjectManager::getInstance ( ) [static]

Get instance method.

Instance creator.

5.15.2.6 ObjectManager * ObjectManager::getInstance (GridObj * *g*) [static]

Overloaded get instance passing in pointer to grid hierarchy.

Instance creator with grid hierarchy assignment.

Parameters

g	pointer to grid hierarchy.

5.15.2.7 void ObjectManager::ibm_apply ( )

Perform IBM procedure.

5.15.2.8 void ObjectManager::ibm_banbks (double ∗∗ *a*, long *n*, int *m1*, int *m2*, double ∗∗ *al*, unsigned long *indx[ ]*, double *b[ ]*)

Solution of a banded diagonal linear system.

Given the arrays A, AL, and INDX as returned from ibm_bandec(), and given a right-hand side vector B[1..n], solves the band diagonal linear equations AX = B. The solution vector X overwrites B. The other input arrays are not modified, and can be left in place for successive calls with different right-hand sides. (C) Copr. 1986-92 Numerical Recipes Software ?421.1-9.

Parameters

a	array of subdiagonal and superdiagonals rows
n	size of the square matrix A
m1	number of subdiagonal rows
m2	number of superdiagonal rows
al	lower triangular matrix
indx	row permutation vector
b	right hand side vector

5.15.2.9 void ObjectManager::ibm_bandec (double ∗∗ *a*, long *n*, int *m1*, int *m2*, double ∗∗ *al*, unsigned long *indx[ ]*, double ∗ *d*)

LU decomposition of band diagonal matrix.

Given an n by n band diagonal matrix A with m1 subdiagonal rows and m2 superdiagonal rows, compactly stored in the array A[1..n][1..m1+m2+1] , this routine constructs an LU decomposition of a rowwise permutation of A. The upper triangular matrix replaces A, while the lower triangular matrix is returned in AL[1..n][1..m1]. INDX[1..n] is an output vector which records the row permutation effected by the partial pivoting; D is output as +/-1 depending on whether the number of row interchanges was even or odd, respectively. This routine is used in combination with ibm_banbks() to solve band-diagonal sets of equations. Once the matrix A has been decomposed, any number of right-hand sides can be solved in turn by repeated calls to ibm_banbks(). (C) Copr. 1986-92 Numerical Recipes Software ?421.1-9.

Parameters

a	array of subdiagonal and superdiagonals rows
n	size of the square matrix A
m1	number of subdiagonal rows
m2	number of superdiagonal rows
al	lower triangular matrix
indx	row permutation vector
d	odd or even number of row interchanges

5.15.2.10 double ObjectManager::ibm_bicgstab (std::vector< std::vector< double > > & *Amatrix*, std::vector< double > & *bVector*, std::vector< double > & *epsilon*, double *tolerance*, int *maxiterations*)

Biconjugate gradient method.

Biconjugate gradient stabilised method of solving a linear system Ax = b. Solution is performed iteratively.

Parameters

Amatrix	the A matrix in the linear system.
bVector	the b vector in the linear system.
epsilon	epsilon paramters for each marker.
tolerance	tolerance of solution.
maxiterations	maximum number of iterations.

Returns

the minimum residual achieved by the solver.

5.15.2.11 void ObjectManager::ibm_buildBody (int *body_type*)

Builds a prefab immersed boundary body.

Parameters

body_type	type of body to be built.

5.15.2.12 void ObjectManager::ibm_buildBody (PCpts * *_PCpts*, GridObj * *owner*)

Wrapper for building a body from a point cloud.

Parameters

_PCpts	pointer to point cloud data.
owner	pointer to the grid on which the body is to be placed.

5.15.2.13 void ObjectManager::ibm_computeForce (int *ib*)

Compute restorative force at each marker in a body.

Parameters

ib	iBody being operated on.

5.15.2.14 double ObjectManager::ibm_deltaKernel (double *radius*, double *dilation*)

Method to evaluate delta kernel at supplied location.

Radius and dilation must be in the same units.

Parameters

radius	location at which kernel should be evaluated.
dilation	width of kernel function.

Returns

value of kernel function.

5.15.2.15 double ObjectManager::ibm_findEpsilon (int *ib*)

Compute epsilon for a given iBody.

Parameters

ib	iBody being operated on.

5.15.2.16 void ObjectManager::ibm_findSupport (int *ib*, int *m*)

Finds support points for iBody.

Support for given marker in given body is sought on the owning grid.

Parameters

ib	body under consideration.
m	marker whose support is to be found.

5.15.2.17 void ObjectManager::ibm_initialise ( )

Initialise the array of iBodies.

Computes support and epsilon values.

5.15.2.18 void ObjectManager::ibm_initialiseSupport (int *ib*, int *m*, int *s*, double *estimated_position[ ]*)

Initialise data associated with support points found.

Finds and stores the delta values of the support points.

Parameters

ib	iBody being operated on.
m	marker of interest.
s	support point of interest.
estimate_position	vector containing the estimated position of the support point.

5.15.2.19 void ObjectManager::ibm_interpol (int *ib*)

Interpolate velocity field onto markers.

Parameters

ib	iBody being operated on.

5.15.2.20 void ObjectManager::ibm_jacowire (int *ib*)

Structural calculation of flexible cilia.

Models the structural behaviour of a thin wire using Euler-Bernoulli beam elements. Only implemented for one simply supported end and one free end at present.

Parameters

ib	index of body to which calculation is to be applied.

5.15.2.21 void ObjectManager::ibm_moveBodies ( )

Moves iBodies after applying IBM.

Wrapper for relocating markers of an iBody be calling appropriate positional update routine.

5.15.2.22 void ObjectManager::ibm_positionUpdate (int *ib*)

Update the position of a deformable iBody.

Wrapper for applying external forcing or structural calculations to iBodies marked as deformable. Updates support on completion.

Parameters

ib	index of body to which calculation is to be applied.

5.15.2.23 void ObjectManager::ibm_positionUpdateGroup (int *group*)

Update the position of a group of deformable iBodies.

Updates the position of a group of non-flexible moving (deformable) bodies by using the first flexible body in the group as the driver. Must be called after all previous positional update routines have been called.

Parameters

group	group ID to be updated.

5.15.2.24 void ObjectManager::ibm_spread (int *ib*)

Spread restorative force back onto marker support.

Parameters

ib	iBody being operated on.

5.15.2.25 void ObjectManager::io_readInCloud (PCpts * *_PCpts,* eObjectType *objtype*)

Read in point cloud data.

Input data must be in tab separated, 3-column format in the input directory.

Parameters

_PCpts	pointer to empty point cloud data container.
objtype	type of object to be read in.

5.15.2.26 void ObjectManager::io_restart (eIOFlag *IO_flag,* int *level*)

Read/write IB body information to restart file.

Parameters

IO_flag	flag indicating write (true) or read (false).
level	level of the grid begin written/read

5.15.2.27 void ObjectManager::io_vtkIBBWriter (double *tval*)

Write IB body data to VTK file.

Currently can only write out un-closed bodies like filaments.

Parameters

tval	time value at which the write out is being performed.

5.15.2.28 void ObjectManager::io_writeBodyPosition (int *timestep*)

Write out position of immersed boundary bodies.

Parameters

timestep	timestep at which the write out is being performed.

5.15.2.29 void ObjectManager::io_writeForceOnObject (double *tval*)

Write out the forces on a solid object.

Writes out the forces on solid objects in the domain computed using momentum exchange. Each rank writes its own file. Output is a CSV file.

Parameters

tval	time value at which write out is taking place.

5.15.2.30 void ObjectManager::io_writeLiftDrag (int *timestep*)

Write out forces on the markers of immersed boundary bodies.

Parameters

timestep	timestep at which the write out is being performed.

5.15.3 Friends And Related Function Documentation

5.15.3.1 friend class GridObj `[friend]`

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

- ObjectManager.h
- ObjectManager.cpp
- ObjectManager_init_bflbody.cpp
- ObjectManager_init_ibmbody.cpp
- ObjectManager_ops_ibm.cpp
- ObjectManager_ops_ibmflex.cpp
- ObjectManager_ops_io.cpp

5.16 PCpts Class Reference

Class to hold point cloud data.

`#include <PCpts.h>`

Public Member Functions

- PCpts (void)

    Default constructor.
- ~PCpts (void)

    Default destructor.

Public Attributes

- std::vector< double > x

    Vector of X positions.
- std::vector< double > y

    Vector of Y positions.
- std::vector< double > z

    Vector of Z positions.

5.16.1 Detailed Description

Class to hold point cloud data.

A container class for hold the X, Y and Z positions of points in a point cloud.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 PCpts::PCpts (void) `[inline]`

Default constructor.

5.16.2.2 PCpts::~PCpts (void) `[inline]`

Default destructor.

5.16.3 Member Data Documentation

5.16.3.1 std::vector<double> PCpts::x

Vector of X positions.

5.16.3.2 std::vector< double > PCpts::y

Vector of Y positions.

5.16.3.3 std::vector< double > PCpts::z

Vector of Z positions.

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

- PCpts.h

5.17 MpiManager::phdf5_struct Struct Reference

Structure for storing halo information for HDF5.

`#include <MpiManager.h>`

Public Attributes

- int i_start
  Starting i-index for writable region.
- int i_end
  Ending i-index for writable region.
- int j_start
  Starting j-index for writable region.
- int j_end
  Ending j-index for writable region.
- int k_start
  Starting k-index for writable region.
- int k_end
  Ending k-index for writable region.
- int level
  Grid level to which these data correspond.
- int region
  Region number to which these data correspond.
- unsigned int writable_data_count = 0
  Writable data count.

5.17.1 Detailed Description

Structure for storing halo information for HDF5.

Structure also stores the amount of writable data on the grid.

5.17.2 Member Data Documentation

5.17.2.1 int MpiManager::phdf5_struct::i_end

Ending i-index for writable region.

5.17.2.2 int MpiManager::phdf5_struct::i_start

Starting i-index for writable region.

5.17.2.3 int MpiManager::phdf5_struct::j_end

Ending j-index for writable region.

5.17.2.4 int MpiManager::phdf5_struct::j_start

Starting j-index for writable region.

5.17.2.5 int MpiManager::phdf5_struct::k_end

Ending k-index for writable region.

5.17.2.6 int MpiManager::phdf5_struct::k_start

Starting k-index for writable region.

5.17.2.7 int MpiManager::phdf5_struct::level

Grid level to which these data correspond.

5.17.2.8 int MpiManager::phdf5_struct::region

Region number to which these data correspond.

5.17.2.9 unsigned int MpiManager::phdf5_struct::writable_data_count = 0

Writable data count.

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

- MpiManager.h

Chapter 6

File Documentation

6.1 BFLBody.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/BFLBody.h"
#include "../inc/MpiManager.h"
#include "../inc/PCpts.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.2 BFLBody.h File Reference

```
#include "stdafx.h"
#include "Body.h"
#include "BFLMarker.h"
```

Classes

- class BFLBody

    BFL body.

6.3 BFLMarker.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/BFLMarker.h"
#include "../inc/GridUtils.h"
```

6.4 BFLMarker.h File Reference

```
#include "stdafx.h"
#include "Marker.h"
```

Classes

- class BFLMarker

  BFL marker.

6.5 Body.h File Reference

```
#include "stdafx.h"
#include "GridUtils.h"
```

Classes

- class MarkerData

  Container class to hold marker information.
- class Body< MarkerType >

  Generic body class.

6.6 definitions.h File Reference

```
#include <time.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <iomanip>
#include <math.h>
#include <string>
#include <mpi.h>
```

Macros

- #define LUMA_VERSION "1.4.0-alpha"

  LUMA version.
- #define L_HDF_DEBUG

  Write some HDF5 debugging information.
- #define L_PI 3.14159265358979323846

  PI definition.
- #define L_BUILD_FOR_MPI

  Enable MPI features in build.

- #define L_OUT_EVERY 100

    How many timesteps before whole grid output.
- #define L_OUT_EVERY_FORCES 1

    Specific output frequency of body forces.
- #define L_OUTPUT_PRECISION 5

    Precision of output (for text writers)
- #define L_HDF5_OUTPUT

    HDF5 dump on output.
- #define L_PROBE_OUT_FREQ 250

    Write out frequency of probe output.
- #define L_GRAVITY_ON
- #define L_GRAVITY_FORCE 0.0001

    Expression for the gravity force.
- #define L_GRAVITY_DIRECTION eXDirection

    Gravity direction (specify using enumeration)
- #define L_NO_FLOW

    Initialise the domain with no flow.
- #define L_RESTART_OUT_FREQ 5000

    Frequency of write out of restart file.
- #define L_CSMAG 0.07
- #define L_TIMESTEPS 10

    Number of time steps to run simulation for.
- #define L_MPI_XCORES 2

    Number of MPI ranks to divide domain into in X direction.
- #define L_MPI_YCORES 2
- #define L_MPI_ZCORES 2
- #define L_DIMS 3

    Number of dimensions to the problem.
- #define L_RESOLUTION 20

    Number of lattice sites per unit length.
- #define L_BX 2.0

    End of domain-x.
- #define L_BY 1.0

    End of domain-y.
- #define L_BZ 1.0

    End of domain-z.
- #define L_PHYSICAL_U 0.2

    Reference velocity of the real fluid to model [m/s].
- #define L_UREF 0.04

    Reference velocity for scaling.
- #define L_UMAX L_UREF*1.5

    Max velocity of inlet profile.
- #define L_UX0 0.04

    Initial/inlet x-velocity.
- #define L_UY0 0.0

    Initial/inlet y-velocity.
- #define L_UZ0 0.0

    Initial/inlet z-velocity.
- #define L_RHOIN 1

    Initial density.
- #define L_RE 150

Desired Reynolds number.

- #define L_IB_ON_LEV 0

  Grid level for immersed boundary object (0 if no refined regions, -1 if no IBM)
- #define L_IB_ON_REG 0

  Grid region for immersed boundary object (0 if no refined regions, -1 if no IBM)
- #define L_VTK_BODY_WRITE

  Write out the bodies to a VTK file.
- #define L_IBB_FROM_FILE

  Build immersed bodies from a point cloud file.
- #define L_IBB_ON_GRID_LEV L_IB_ON_LEV

  Provide grid level on which object should be added.
- #define L_IBB_ON_GRID_REG L_IB_ON_REG

  Provide grid region on which object should be added.
- #define L_START_IBB_X 0.4

  Start X of object bounding box.
- #define L_START_IBB_Y 0.35

  Start Y of object bounding box.
- #define L_CENTRE_IBB_Z 0.5

  Centre of object bounding box in Z direction.
- #define L_IBB_LENGTH 0.2

  The object input is scaled based on this dimension.
- #define L_IBB_SCALE_DIRECTION eXDirection

  Scale in this direction (specify as enumeration)
- #define L_IBB_REF_LENGTH 0.2

  Reference length to be used in the definition of Reynolds number.
- #define L_NUM_MARKERS 31

  Number of Lagrange points to use when building a prefab body (approximately)
- #define L_IBB_MOVABLE false

  Default deformable property of body to be built (whether it moves or not)
- #define L_IBB_FLEXIBLE false

  Whether a structural calculation needs to be performed on the body.
- #define L_IBB_X 0.2

  X Position of body centre.
- #define L_IBB_Y 0.2

  Y Position of body centre.
- #define L_IBB_Z 0.0

  Z Position of body centre.
- #define L_IBB_W 0.5

  Width (x) of IB body.
- #define L_IBB_L 0.5

  Length (y) of IB body.
- #define L_IBB_D 0.5

  Depth (z) of IB body.
- #define L_IBB_R 0.05

  Radius of IB body.
- #define L_IBB_FILAMENT_LENGTH 0.5

  Length of filament.
- #define L_IBB_FILAMENT_START_X 0.2

  Start X position of the filament.
- #define L_IBB_FILAMENT_START_Y 0.5

  Start Y position of the filament.

- #define L_IBB_FILAMENT_START_Z 0.5

  Start Z position of the filament.
- #define L_IBB_ANGLE_VERT 90

  Inclination of filament in XY plane.
- #define L_IBB_ANGLE_HORZ 0

  Inclination of filament in XZ plane.
- #define L_FILAMENT_START_BC 2

  Type of boundary condition at filament start: 0 == free; 1 = simply supported; 2 == clamped.
- #define L_FILAMENT_END_BC 0

  Type of boundary condition at filament end: 0 == free; 1 = simply supported; 2 == clamped.
- #define L_IBB_DELTA_RHO 1.0

  Difference in density (lattice units) between solid and fluid.
- #define L_IBB_EI 2.0

  Flexural rigidity (lattice units) of filament.
- #define L_WALLS_ON

  Turn on no-slip walls (default is top, bottom, front, back unless L_WALLS_ON_2D is used)
- #define L_WALL_THICKNESS_BOTTOM (L_BX/L_N)

  Thickness of wall.
- #define L_WALL_THICKNESS_TOP (L_BX/L_N)

  Thickness of top wall.
- #define L_WALL_THICKNESS_FRONT (L_BX/L_N)

  Thickness of front (3D) wall.
- #define L_WALL_THICKNESS_BACK (L_BX/L_N)

  Thickness of back (3D) wall.
- #define L_SOLID_BLOCK_ON

  Add solid block to the domain.
- #define L_BLOCK_ON_GRID_LEV 2

  Provide grid level on which block should be added.
- #define L_BLOCK_ON_GRID_REG 0

  Provide grid region on which block should be added.
- #define L_BLOCK_MIN_X 0.9

  Start of object/wall in x-direction.
- #define L_BLOCK_MAX_X 1.1

  End of object/wall in x-direction.
- #define L_BLOCK_MIN_Y 0.4

  Start of object/wall in y-direction.
- #define L_BLOCK_MAX_Y 0.6

  End of object/wall in y-direction.
- #define L_BLOCK_MIN_Z 0.3

  Start of object/wall in z-direction.
- #define L_BLOCK_MAX_Z 0.7

  End of object/wall in z-direction.
- #define L_OBJECT_ON_GRID_LEV 2

  Provide grid level on which object should be added.
- #define L_OBJECT_ON_GRID_REG 0

  Provide grid region on which object should be added.
- #define L_START_OBJECT_X 0.7

  Start of object bounding box in X direction.
- #define L_START_OBJECT_Y 0.4

  Start of object bounding box in Y direction.
- #define L_CENTRE_OBJECT_Z 0.5

Centre of object bounding box in Z direction.
- #define L_OBJECT_LENGTH 0.2

  The object input is scaled based on this dimension.
- #define L_OBJECT_SCALE_DIRECTION eXDirection

  Scale in this direction (specify as enumeration)
- #define L_OBJECT_REF_LENGTH 0.2

  Reference length to be used in the definition of Reynolds number.
- #define L_BFL_ON_GRID_LEV 2

  Provide grid level on which BFL body should be added.
- #define L_BFL_ON_GRID_REG 0

  Provide grid region on which BFL body should be added.
- #define L_START_BFL_X 0.9

  Start of object bounding box in X direction.
- #define L_START_BFL_Y 0.4

  Start of object bounding box in Y direction.
- #define L_CENTRE_BFL_Z 0.5

  Centre of object bounding box in Z direction.
- #define L_BFL_LENGTH 0.2

  The BFL object input is scaled based on this dimension.
- #define L_BFL_SCALE_DIRECTION eXDirection

  Scale in this direction (specify as enumeration)
- #define L_BFL_REF_LENGTH 0.2

  Reference length to be used in the definition of Reynolds number.
- #define L_NUM_LEVELS 2

  Levels of refinement (0 = coarse grid only)
- #define L_NUM_REGIONS 1

  Number of refined regions (can be arbitrary if L_NUM_LEVELS = 0)
- #define L_N (L_BX ∗ L_RESOLUTION)
- #define L_M (L_BY ∗ L_RESOLUTION)
- #define L_K (L_BZ ∗ L_RESOLUTION)
- #define L_NUM_VELS 19

  Number of lattice velocities.
- #define L_MPI_DIRS 26

  Number of MPI directions.

Variables

- static const int cNumProbes [3] = {3, 3, 3}

  Number of probes in each direction (x, y, z)
- static const double cProbeLimsX [2] = {0.1, 0.2}

  Limits of X plane for array of probes.
- static const double cProbeLimsY [2] = {0.1, 0.2}

  Limits of Y plane for array of probes.
- static const double cProbeLimsZ [2] = {0.1, 0.2}

  Limits of Z plane for array of probes.
- static double cRefStartX [L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.5 }, { 0.6 } }
- static double cRefEndX [L_NUM_LEVELS][L_NUM_REGIONS] = { { 1.5 }, { 1.4 } }
- static double cRefStartY [L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.2 }, { 0.3 } }
- static double cRefEndY [L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.8 }, { 0.7 } }
- static double cRefStartZ [L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.1 }, { 0.25 } }
- static double cRefEndZ [L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.9 }, { 0.75 } }

6.6.1 Macro Definition Documentation

6.6.1.1 #define L_BFL_LENGTH 0.2

The BFL object input is scaled based on this dimension.

6.6.1.2 #define L_BFL_ON_GRID_LEV 2

Provide grid level on which BFL body should be added.

6.6.1.3 #define L_BFL_ON_GRID_REG 0

Provide grid region on which BFL body should be added.

6.6.1.4 #define L_BFL_REF_LENGTH 0.2

Reference length to be used in the definition of Reynolds number.

6.6.1.5 #define L_BFL_SCALE_DIRECTION eXDirection

Scale in this direction (specify as enumeration)

6.6.1.6 #define L_BLOCK_MAX_X 1.1

End of object/wall in x-direction.

6.6.1.7 #define L_BLOCK_MAX_Y 0.6

End of object/wall in y-direction.

6.6.1.8 #define L_BLOCK_MAX_Z 0.7

End of object/wall in z-direction.

6.6.1.9 #define L_BLOCK_MIN_X 0.9

Start of object/wall in x-direction.

6.6.1.10 #define L_BLOCK_MIN_Y 0.4

Start of object/wall in y-direction.

6.6.1.11 #define L_BLOCK_MIN_Z 0.3

Start of object/wall in z-direction.

6.6.1.12 #define L_BLOCK_ON_GRID_LEV 2

Provide grid level on which block should be added.

6.6.1.13 #define L_BLOCK_ON_GRID_REG 0

Provide grid region on which block should be added.

6.6.1.14 #define L_BUILD_FOR_MPI

Enable MPI features in build.

6.6.1.15 #define L_BX 2.0

End of domain-x.

6.6.1.16 #define L_BY 1.0

End of domain-y.

6.6.1.17 #define L_BZ 1.0

End of domain-z.

6.6.1.18 #define L_CENTRE_BFL_Z 0.5

Centre of object bounding box in Z direction.

6.6.1.19 #define L_CENTRE_IBB_Z 0.5

Centre of object bounding box in Z direction.

6.6.1.20 #define L_CENTRE_OBJECT_Z 0.5

Centre of object bounding box in Z direction.

6.6.1.21 #define L_CSMAG 0.07

6.6.1.22 #define L_DIMS 3

Number of dimensions to the problem.

6.6.1.23 #define L_FILAMENT_END_BC 0

Type of boundary condition at filament end: 0 == free; 1 = simply supported; 2 == clamped.

6.6.1.24 #define L_FILAMENT_START_BC 2

Type of boundary condition at filament start: 0 == free; 1 = simply supported; 2 == clamped.

6.6.1.25 #define L_GRAVITY_DIRECTION eXDirection

Gravity direction (specify using enumeration)

6.6.1.26 #define L_GRAVITY_FORCE 0.0001

Expression for the gravity force.

6.6.1.27 #define L_GRAVITY_ON

Turn on gravity force

6.6.1.28 #define L_HDF5_OUTPUT

HDF5 dump on output.

6.6.1.29 #define L_HDF_DEBUG

Write some HDF5 debugging information.

6.6.1.30 #define L_IB_ON_LEV 0

Grid level for immersed boundary object (0 if no refined regions, -1 if no IBM)

6.6.1.31 #define L_IB_ON_REG 0

Grid region for immersed boundary object (0 if no refined regions, -1 if no IBM)

6.6.1.32 #define L_IBB_ANGLE_HORZ 0

Inclination of filament in XZ plane.

6.6.1.33 #define L_IBB_ANGLE_VERT 90

Inclination of filament in XY plane.

6.6.1.34 #define L_IBB_D 0.5

Depth (z) of IB body.

6.6.1.35 #define L_IBB_DELTA_RHO 1.0

Difference in density (lattice units) between solid and fluid.

6.6.1.36 #define L_IBB_EI 2.0

Flexural rigidity (lattice units) of filament.

6.6.1.37 #define L_IBB_FILAMENT_LENGTH 0.5

Length of filament.

6.6.1.38 #define L_IBB_FILAMENT_START_X 0.2

Start X position of the filament.

6.6.1.39 #define L_IBB_FILAMENT_START_Y 0.5

Start Y position of the filament.

6.6.1.40 #define L_IBB_FILAMENT_START_Z 0.5

Start Z position of the filament.

6.6.1.41 #define L_IBB_FLEXIBLE false

Whether a structural calculation needs to be performed on the body.

6.6.1.42 #define L_IBB_FROM_FILE

Build immersed bodies from a point cloud file.

6.6.1.43 #define L_IBB_L 0.5

Length (y) of IB body.

6.6.1.44 #define L_IBB_LENGTH 0.2

The object input is scaled based on this dimension.

6.6.1.45 #define L_IBB_MOVABLE false

Default deformable property of body to be built (whether it moves or not)

6.6.1.46 #define L_IBB_ON_GRID_LEV L_IB_ON_LEV

Provide grid level on which object should be added.

6.6.1.47 #define L_IBB_ON_GRID_REG L_IB_ON_REG

Provide grid region on which object should be added.

6.6.1.48 #define L_IBB_R 0.05

Radius of IB body.

6.6.1.49 #define L_IBB_REF_LENGTH 0.2

Reference length to be used in the definition of Reynolds number.

6.6.1.50 #define L_IBB_SCALE_DIRECTION eXDirection

Scale in this direction (specify as enumeration)

6.6.1.51 #define L_IBB_W 0.5

Width (x) of IB body.

6.6.1.52 #define L_IBB_X 0.2

X Position of body centre.

6.6.1.53 #define L_IBB_Y 0.2

Y Position of body centre.

6.6.1.54 #define L_IBB_Z 0.0

Z Position of body centre.

6.6.1.55 #define L_K (L_BZ ∗ L_RESOLUTION)

6.6.1.56 #define L_M (L_BY ∗ L_RESOLUTION)

6.6.1.57 #define L_MPI_DIRS 26

Number of MPI directions.

6.6.1.58 #define L_MPI_XCORES 2

Number of MPI ranks to divide domain into in X direction.

6.6.1.59 #define L_MPI_YCORES 2

Number of MPI ranks to divide domain into in Y direction

6.6.1.60 #define L_MPI_ZCORES 2

Number of MPI ranks to divide domain into in Z direction. Set to 1 if doing a 2D problem when using custom MPI sizes

6.6.1.61 #define L_N (L_BX ∗ L_RESOLUTION)

6.6.1.62 #define L_NO_FLOW

Initialise the domain with no flow.

6.6.1.63 #define L_NUM_LEVELS 2

Levels of refinement (0 = coarse grid only)

6.6.1.64 #define L_NUM_MARKERS 31

Number of Lagrange points to use when building a prefab body (approximately)

6.6.1.65 #define L_NUM_REGIONS 1

Number of refined regions (can be arbitrary if L_NUM_LEVELS = 0)

6.6.1.66 #define L_NUM_VELS 19

Number of lattice velocities.

6.6.1.67 #define L_OBJECT_LENGTH 0.2

The object input is scaled based on this dimension.

6.6.1.68 #define L_OBJECT_ON_GRID_LEV 2

Provide grid level on which object should be added.

6.6.1.69 #define L_OBJECT_ON_GRID_REG 0

Provide grid region on which object should be added.

6.6.1.70 #define L_OBJECT_REF_LENGTH 0.2

Reference length to be used in the definition of Reynolds number.

6.6.1.71 #define L_OBJECT_SCALE_DIRECTION eXDirection

Scale in this direction (specify as enumeration)

6.6.1.72 #define L_OUT_EVERY 100

How many timesteps before whole grid output.

6.6.1.73 #define L_OUT_EVERY_FORCES 1

Specific output frequency of body forces.

6.6.1.74 #define L_OUTPUT_PRECISION 5

Precision of output (for text writers)

6.6.1.75 #define L_PHYSICAL_U 0.2

Reference velocity of the real fluid to model [m/s].

6.6.1.76 #define L_PI 3.14159265358979323846

PI definition.

6.6.1.77 #define L_PROBE_OUT_FREQ 250

Write out frequency of probe output.

6.6.1.78 #define L_RE 150

Desired Reynolds number.

6.6.1.79 #define L_RESOLUTION 20

Number of lattice sites per unit length.

6.6.1.80 #define L_RESTART_OUT_FREQ 5000

Frequency of write out of restart file.

6.6.1.81 #define L_RHOIN 1

Initial density.

6.6.1.82 #define L_SOLID_BLOCK_ON

Add solid block to the domain.

6.6.1.83 #define L_START_BFL_X 0.9

Start of object bounding box in X direction.

6.6.1.84 #define L_START_BFL_Y 0.4

Start of object bounding box in Y direction.

6.6.1.85 #define L_START_IBB_X 0.4

Start X of object bounding box.

6.6.1.86 #define L_START_IBB_Y 0.35

Start Y of object bounding box.

6.6.1.87 #define L_START_OBJECT_X 0.7

Start of object bounding box in X direction.

6.6.1.88 #define L_START_OBJECT_Y 0.4

Start of object bounding box in Y direction.

6.6.1.89 #define L_TIMESTEPS 10

Number of time steps to run simulation for.

6.6.1.90 #define L_UMAX L_UREF*1.5

Max velocity of inlet profile.

6.6.1.91 #define L_UREF 0.04

Reference velocity for scaling.

6.6.1.92 #define L_UX0 0.04

Initial/inlet x-velocity.

6.6.1.93 #define L_UY0 0.0

Initial/inlet y-velocity.

6.6.1.94 #define L_UZ0 0.0

Initial/inlet z-velocity.

6.6.1.95 #define L_VTK_BODY_WRITE

Write out the bodies to a VTK file.

6.6.1.96 #define L_WALL_THICKNESS_BACK (L_BX/L_N)

Thickness of back (3D) wall.

6.6.1.97 #define L_WALL_THICKNESS_BOTTOM (L_BX/L_N)

Thickness of wall.

6.6.1.98 #define L_WALL_THICKNESS_FRONT (L_BX/L_N)

Thickness of front (3D) wall.

6.6.1.99 #define L_WALL_THICKNESS_TOP (L_BX/L_N)

Thickness of top wall.

6.6.1.100 #define L_WALLS_ON

Turn on no-slip walls (default is top, bottom, front, back unless L_WALLS_ON_2D is used)

6.6.1.101 #define LUMA_VERSION "1.4.0-alpha"

LUMA version.

6.6.2 Variable Documentation

6.6.2.1 const int cNumProbes[3] = {3, 3, 3} `[static]`

Number of probes in each direction (x, y, z)

6.6.2.2 const double cProbeLimsX[2] = {0.1, 0.2} `[static]`

Limits of X plane for array of probes.

6.6.2.3 const double cProbeLimsY[2] = {0.1, 0.2} [static]

Limits of Y plane for array of probes.

6.6.2.4 const double cProbeLimsZ[2] = {0.1, 0.2} [static]

Limits of Z plane for array of probes.

6.6.2.5 double cRefEndX[L_NUM_LEVELS][L_NUM_REGIONS] = { { 1.5 }, { 1.4 } } [static]

6.6.2.6 double cRefEndY[L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.8 }, { 0.7 } } [static]

6.6.2.7 double cRefEndZ[L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.9 }, { 0.75 } } [static]

6.6.2.8 double cRefStartX[L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.5 }, { 0.6 } } [static]

6.6.2.9 double cRefStartY[L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.2 }, { 0.3 } } [static]

6.6.2.10 double cRefStartZ[L_NUM_LEVELS][L_NUM_REGIONS] = { { 0.1 }, { 0.25 } } [static]

6.7 GridObj.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/MpiManager.h"
#include "../inc/GridUtils.h"
```

6.8 GridObj.h File Reference

```
#include "stdafx.h"
#include "IVector.h"
```

Classes

- class GridObj

  Grid class.

Enumerations

- enum eType {
  eSolid, eFluid, eRefined, eTransitionToCoarser,
  eTransitionToFiner, eBFL, eSymmetry, eInlet,
  eOutlet, eRefinedSolid, eRefinedSymmetry, eRefinedInlet }

  Lattice typing labels.
- enum eBCType {
  eBCAll, eBCSolidSymmetry, eBCInlet, eBCOutlet,
  eBCInletOutlet, eBCBFL }

  Flag for indicating which BCs to apply.
- enum eIOFlag { eWrite, eRead }

  Flag for indicating write or read action for IO methods.

6.8.1 Enumeration Type Documentation

6.8.1.1 enum **eBCType**

Flag for indicating which BCs to apply.

Enumerator

eBCAll Apply all BCs.
eBCSolidSymmetry Apply just solid and symmetry BCs.
eBCInlet Apply just inlet BCs.
eBCOutlet Apply just outlet BCs.
eBCInletOutlet Apply inlet and outlet BCs.
eBCBFL Apply just BFL BCs.

6.8.1.2 enum **eIOFlag**

Flag for indicating write or read action for IO methods.

Enumerator

eWrite Write to file.
eRead Read from file.

6.8.1.3 enum **eType**

Lattice typing labels.

Enumerator

eSolid Rigid, solid site.
eFluid Fluid site.
eRefined Fluid site which is represented on a finer grid.
eTransitionToCoarser Fluid site coupled to a coarser grid.
eTransitionToFiner Fluid site coupled to a finer grid.
eBFL Site containing a BFL marker.
eSymmetry Symmetry boundary.
eInlet Inlet boundary.
eOutlet Outlet boundary.
eRefinedSolid Rigid, solid site represented on a finer grid.
eRefinedSymmetry Symmtery boundary represented on a finer grid.
eRefinedInlet Inlet site represented on a finer grid.

6.9 GridObj_init_grids.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/MpiManager.h"
#include "../inc/GridUtils.h"
```

6.10 GridObj_ops_boundary.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/BFLBody.h"
#include "../inc/ObjectManager.h"
#include "../inc/GridUtils.h"
```

6.11 GridObj_ops_io.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/MpiManager.h"
#include "../inc/ObjectManager.h"
#include "../inc/GridUtils.h"
#include "../inc/hdf5luma.h"
#include "../inc/GridUnits.h"
```

6.12 GridObj_ops_lbm.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/IVector.h"
#include "../inc/ObjectManager.h"
#include "../inc/MpiManager.h"
#include "../inc/GridUtils.h"
```

6.13 GridObj_ops_lbm_optimised.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/ObjectManager.h"
#include "../inc/MpiManager.h"
#include "../inc/GridUtils.h"
```

6.14 GridUnits.h File Reference

```
#include "../inc/GridObj.h"
```

Classes

- class GridUnits

  GridUnits.

6.15 GridUtils.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridUtils.h"
#include "../inc/MpiManager.h"
#include "../inc/GridObj.h"
```

6.16 GridUtils.h File Reference

```
#include "stdafx.h"
#include "GridObj.h"
```

Classes

- class GridUtils

  Grid utility class.

Enumerations

- enum eLocationOnRank { eNone, eCore, eHalo }

  Enumeration indicating the location of a site when queried using isOnThisRank()
- enum eCartesianDirection { eXDirection, eYDirection, eZDirection }

  Enumeration for directional options.
- enum eMinMax { eMinimum, eMaximum }

  Enumeration for minimum and maximum.
- enum eCartMinMax {
  eXMin, eXMax, eYMin, eYMax,
  eZMin, eZMax }

  Enumeration for the combination of eCartesianDirection and eMinMax as these are often used together to index arrays.
- enum eEdgeMinMax { eLeftMin, eLeftMax, eRightMin, eRightMax }

  Enumeration for the combination of Left and Right min and max edges.

6.16.1 Enumeration Type Documentation

6.16.1.1 enum eCartesianDirection

Enumeration for directional options.

Enumerator

- ***eXDirection*** X-direction.
- ***eYDirection*** Y-direction.
- ***eZDirection*** Z-direction.

6.16.1.2 enum eCartMinMax

Enumeration for the combination of eCartesianDirection and eMinMax as these are often used together to index arrays.

Enumerator

- ***eXMin***
- ***eXMax***
- ***eYMin***
- ***eYMax***
- ***eZMin***
- ***eZMax***

6.16.1.3 enum eEdgeMinMax

Enumeration for the combination of Left and Right min and max edges.

Enumerator

- ***eLeftMin***
- ***eLeftMax***
- ***eRightMin***
- ***eRightMax***

6.16.1.4 enum eLocationOnRank

Enumeration indicating the location of a site when queried using isOnThisRank()

Enumerator

- ***eNone*** No information provided (default).
- ***eCore*** Site on core (including send layer).
- ***eHalo*** Site in halo (recv layer).

6.16.1.5 enum eMinMax

Enumeration for minimum and maximum.

Some utility methods need to know whether they should be looking at or for a maximum or minimum edge of a grid so we use this enumeration to specify.

Enumerator

eMinimum Minimum.

eMaximum Maximum.

6.17 hdf5luma.h File Reference

```
#include "stdafx.h"
#include "hdf5.h"
#include "MpiManager.h"
```

Macros

- #define H5_BUILT_AS_DYNAMIC_LIB
- #define HDF5_EXT_ZLIB
- #define HDF5_EXT_SZIP

Enumerations

- enum eHdf5SlabType {
  eScalar, eVector, eProductVector, ePosX,
  ePosY, ePosZ }

  Defines the type of storage arrangement of the variable in memory.

Functions

- template<typename T >
  void hdf5_writeDataSet (hid_t &memspace, hid_t &filespace, hid_t &dataset_id, eHdf5SlabType slab_type, int N_lim, int M_lim, int K_lim, GridObj *g, T *data, hid_t hdf_datatype, bool *TL_present, int TL_thickness, MpiManager::phdf5_struct hdf_data)

  Helper method to write out using HDF5.

6.17.1 Macro Definition Documentation

6.17.1.1 #define H5_BUILT_AS_DYNAMIC_LIB

6.17.1.2 #define HDF5_EXT_SZIP

6.17.1.3 #define HDF5_EXT_ZLIB

6.17.2 Enumeration Type Documentation

6.17.2.1 enum eHdf5SlabType

Defines the type of storage arrangement of the variable in memory.

The write wrapper can then extract the data from memeory and write it to an HDF5 file using a particular hyperslab selection.

Enumerator

- ***eScalar*** 2/3D data – One variable per grid site
- ***eVector*** 2/3D data – L_DIMS variables per grid site
- ***eProductVector*** 1D data – 3∗L_DIMS-3 variables per grid site
- ***ePosX*** 1D data – Single L_dim vector per dimension
- ***ePosY*** 1D data – Single L_dim vector per dimension
- ***ePosZ*** 1D data – Single L_dim vector per dimension

6.17.3 Function Documentation

6.17.3.1 template<typename T > void hdf5_writeDataSet (hid_t & *memspace*, hid_t & *filespace*, hid_t & *dataset_id*, eHdf5SlabType *slab_type*, int *N_lim*, int *M_lim*, int *K_lim*, GridObj ∗ *g*, T ∗ *data*, hid_t *hdf_datatype*, bool ∗ *TL_present*, int *TL_thickness*, MpiManager::phdf5_struct *hdf_data*)

Helper method to write out using HDF5.

Automatically selects the correct slab arrangement and buffers the data accordingly before writing to structured file.

Parameters

memspace	memory dataspace id.
filespace	file dataspace id.
dataset_id	dataset id.
slab_type	slab type enum.
N_lim	number of X-direction sites on the local grid.
M_lim	number of Y-direction sites on the local grid.
K_lim	number of Z-direction sites on the local grid.
g	pointer to grid which we are writing out.
data	pointer to the start of the array to be written.
hdf_datatype	HDF5 datatype being written.
TL_present	pointer to array of flags indicating whether a lower TL is present on this grid in given direction so offset in file can be computed.
TL_thickness	the thickness of the TL on this grid level in local lattice units.
hdf_data	the data structure containing information about local halos.

6.18 IBBody.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/IBBody.h"
#include "../inc/IBMarker.h"
#include "../inc/PCpts.h"
#include "../inc/GridUtils.h"
#include "../inc/ObjectManager.h"
```

6.19 IBBody.h File Reference

```
#include "stdafx.h"
#include "Body.h"
```

Classes

- class IBBody

  Immersed boundary body.

6.20 IBMarker.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/IBMarker.h"
#include "../inc/GridUtils.h"
```

6.21 IBMarker.h File Reference

```
#include "stdafx.h"
#include "Marker.h"
```

Classes

- class IBMarker

  Immersed boundary marker.

6.22 IVector.h File Reference

```
#include "stdafx.h"
```

Classes

- class IVector< GenTyp >

    Index-collapsing vector class.

6.23 main_lbm.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/MpiManager.h"
#include "../inc/ObjectManager.h"
#include "../inc/GridUtils.h"
#include "../inc/PCpts.h"
```

Functions

- int main (int argc, char *argv[ ])

    Entry point for the application.

6.23.1 Function Documentation

6.23.1.1 int main (int *argc,* char * *argv[ ]*)

Entry point for the application.

6.24 Marker.h File Reference

```
#include "stdafx.h"
```

Classes

- class Marker

    Generic marker class.

6.25 Mpi_buffer_pack.cpp File Reference

```
#include "../inc/stdafx.h"
#include <mpi.h>
#include "../inc/MpiManager.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.26 Mpi_buffer_size_recv.cpp File Reference

```
#include "../inc/stdafx.h"
#include <mpi.h>
#include "../inc/MpiManager.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.27 Mpi_buffer_size_send.cpp File Reference

```
#include "../inc/stdafx.h"
#include <mpi.h>
#include "../inc/MpiManager.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.28 Mpi_buffer_unpk.cpp File Reference

```
#include "../inc/stdafx.h"
#include <mpi.h>
#include "../inc/MpiManager.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.29 MpiManager.cpp File Reference

```
#include "../inc/stdafx.h"
#include <mpi.h>
#include "../inc/MpiManager.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.30 MpiManager.h File Reference

```
#include "stdafx.h"
```

Classes

- class MpiManager

  MPI Manager class.
- struct MpiManager::phdf5_struct

  Structure for storing halo information for HDF5.
- struct MpiManager::layer_edges

  Structure containing absolute positions of the edges of halos.
- struct MpiManager::buffer_struct

  Structure storing buffers sizes in each direction for particular grid.

Macros

- #define range_i_left i = 0; i < GridUtils::downToLimit((int)pow(2, g->level + 1), N_lim); i++

  For loop definition for left halo.
- #define range_j_down j = 0; j < GridUtils::downToLimit((int)pow(2, g->level + 1), M_lim); j++

  For loop definition for bottom halo.
- #define range_k_front k = 0; k < GridUtils::downToLimit((int)pow(2, g->level + 1), K_lim); k++

  For loop definition for front halo.
- #define range_i_right i = GridUtils::upToZero(N_lim - (int)pow(2, g->level + 1)); i < N_lim; i++

  For loop definition for right halo.
- #define range_j_up j = GridUtils::upToZero(M_lim - (int)pow(2, g->level + 1)); j < M_lim; j++

  For loop definition for top halo.
- #define range_k_back k = GridUtils::upToZero(K_lim - (int)pow(2, g->level + 1)); k < K_lim; k++

  For loop definition for back halo.

6.30.1 Macro Definition Documentation

6.30.1.1 #define range_i_left i = 0; i < **GridUtils::downToLimit**((int)pow(2, g->level + 1), N_lim); i++

For loop definition for left halo.

6.30.1.2 #define range_i_right i = **GridUtils::upToZero**(N_lim - (int)pow(2, g->level + 1)); i < N_lim; i++

For loop definition for right halo.

6.30.1.3 #define range_j_down j = 0; j < **GridUtils::downToLimit**((int)pow(2, g->level + 1), M_lim); j++

For loop definition for bottom halo.

6.30.1.4 #define range_j_up j = **GridUtils::upToZero**(M_lim - (int)pow(2, g->level + 1)); j < M_lim; j++

For loop definition for top halo.

6.30.1.5 #define range_k_back k = GridUtils::upToZero(K_lim - (int)pow(2, g->level + 1)); k < K_lim; k++

For loop definition for back halo.

6.30.1.6 #define range_k_front k = 0; k < GridUtils::downToLimit((int)pow(2, g->level + 1), K_lim); k++

For loop definition for front halo.

6.31 ObjectManager.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/ObjectManager.h"
#include "../inc/GridObj.h"
#include "../inc/GridUtils.h"
```

6.32 ObjectManager.h File Reference

```
#include "stdafx.h"
#include "IVector.h"
#include "IBMarker.h"
#include "IBBody.h"
#include "BFLBody.h"
```

Classes

- class ObjectManager

  Object Manager class.

Enumerations

- enum eObjectType { eBBBCloud, eBFLCloud, eIBBCloud }

  Specifies the type of body being processed.

6.32.1 Enumeration Type Documentation

6.32.1.1 enum eObjectType

Specifies the type of body being processed.

Enumerator

eBBBCloud Bounce-back body.

eBFLCloud BFL body.

eIBBCloud Immersed boundary body.

6.33 ObjectManager_init_bflbody.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/ObjectManager.h"
```

6.34 ObjectManager_init_ibmbody.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/ObjectManager.h"
```

6.35 ObjectManager_ops_ibm.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/ObjectManager.h"
#include "../inc/MpiManager.h"
#include "../inc/GridUtils.h"
```

6.36 ObjectManager_ops_ibmflex.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/GridObj.h"
#include "../inc/ObjectManager.h"
#include "../inc/MpiManager.h"
```

Macros

- #define SWAP(a, b) {dum=(a);(a)=(b);(b)=dum;}

    Pointer swap definition.
- #define TINY 1.0e-20

    Definition of small number (could use numerics since this is C++ but nevermind)
- #define SWAP(a, b) {dum=(a);(a)=(b);(b)=dum;}

    Pointer swap definition.

6.36.1 Macro Definition Documentation

6.36.1.1 #define SWAP(*a*, *b*) {dum=(a);(a)=(b);(b)=dum;}

Pointer swap definition.

6.36.1.2 #define SWAP(*a*, *b*) {dum=(a);(a)=(b);(b)=dum;}

Pointer swap definition.

6.36.1.3 #define TINY 1.0e-20

Definition of small number (could use numerics since this is C++ but nevermind)

6.37 ObjectManager_ops_io.cpp File Reference

```
#include "../inc/stdafx.h"
#include "../inc/ObjectManager.h"
#include "../inc/GridUtils.h"
#include "../inc/PCpts.h"
#include "../inc/GridObj.h"
#include "../inc/MpiManager.h"
```

6.38 PCpts.h File Reference

```
#include "stdafx.h"
```

Classes

- class PCpts

  Class to hold point cloud data.

6.39 stdafx.cpp File Reference

```
#include "../inc/stdafx.h"
```

Variables

- const int c [3][L_NUM_VELS]

  Lattice velocities.
- const int c_opt [L_NUM_VELS][3]

  Lattice velocities optimised arrangement.
- const double w [L_NUM_VELS]

  Quadrature weights.
- const double cs = 1.0 / sqrt(3.0)

  Lattice sound speed.

6.39.1 Variable Documentation

6.39.1.1 const int c[3][L_NUM_VELS]

Initial value:

```
=
{
    {1, -1,  0,  0,  0,  0,  1, -1,  1, -1,  0,  0,  0,  0,  1, -1, -1,  1,  0 },
    {0,  0,  1, -1,  0,  0,  1, -1, -1,  1,  1, -1,  1, -1,  0,  0,  0,  0,  0 },
    {0,  0,  0,  0,  1, -1,  0,  0,  0,  0,  1, -1, -1,  1,  1, -1,  1, -1,  0 }
}
```

Lattice velocities.

6.39.1.2 const int c_opt[L_NUM_VELS][3]

Initial value:

```
=
{
    { 1, 0, 0 },
    { -1, 0, 0 },
    { 0, 1, 0 },
    { 0, -1, 0 },
    { 0, 0, 1 },
    { 0, 0, -1 },
    { 1, 1, 0 },
    { -1, -1, 0 },
    { 1, -1, 0 },
    { -1, 1, 0 },
    { 0, 1, 1 },
    { 0, -1, -1 },
    { 0, 1, -1 },
    { 0, -1, 1 },
    { 1, 0, 1 },
    { -1, 0, -1 },
    { -1, 0, 1 },
    { 1, 0, -1 },
    { 0, 0, 0, }
}
```

Lattice velocities optimised arrangement.

6.39.1.3 const double cs = 1.0 / sqrt(3.0)

Lattice sound speed.

6.39.1.4 const double w[L_NUM_VELS]

Initial value:

```
=
{1.0/18.0, 1.0/18.0, 1.0/18.0, 1.0/18.0, 1.0/18.0, 1.0/18.0,
1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.
      0, 1.0/36.0,
1.0/3.0}
```

Quadrature weights.

6.40 stdafx.h File Reference

```
#include <algorithm>
#include <cmath>
#include <vector>
#include <iostream>
#include <fstream>
#include <sstream>
#include <numeric>
#include <assert.h>
#include <stdlib.h>
#include <cstring>
#include <stdio.h>
#include "definitions.h"
```

Macros

- #define DEPRECATED
- #define L_IS_NAN std::isnan

    Not a Number declaration (Unix)
- #define SQ(x) ((x) ∗ (x))
- #define L_DACTION_WRITE_OUT_FORCES
- #define LUMA_FAILED 12345

    Error definition.
- #define L_ERROR errorfcn

Functions

- void errorfcn (std::string msg, std::ofstream ∗logfile)

    Fatal Error function.

Variables

- const int c [3][L_NUM_VELS]

    Lattice velocities.
- const int c_opt [L_NUM_VELS][3]

    Lattice velocities optimised arrangement.
- const double w [L_NUM_VELS]

    Quadrature weights.
- const double cs

    Lattice sound speed.

6.40.1 Macro Definition Documentation

6.40.1.1 #define DEPRECATED

6.40.1.2 #define L_DACTION_WRITE_OUT_FORCES

6.40.1.3 #define L_ERROR errorfcn

Error function shorthand

6.40.1.4 #define L_IS_NAN std::isnan

Not a Number declaration (Unix)

6.40.1.5 #define LUMA_FAILED 12345

Error definition.

6.40.1.6 #define SQ(*x*) ((x) ∗ (x))

6.40.2 Function Documentation

6.40.2.1 void errorfcn (std::string *msg*, std::ofstream ∗ *logfile*) [inline]

Fatal Error function.

Writes error to the user and further information to the supplied logfile. Inlined since this header is included everywhere.

Parameters

msg	string to be printed to the log file.
logfile	pointer to the logfile where the message is to be written.

6.40.3 Variable Documentation

6.40.3.1 const int c[3][L_NUM_VELS]

Lattice velocities.

6.40.3.2 const int c_opt[L_NUM_VELS][3]

Lattice velocities optimised arrangement.

6.40.3.3 const double cs

Lattice sound speed.

6.40.3.4 const double w[L_NUM_VELS]

Quadrature weights.

Index