# Luxum

Generated by Doxygen 1.8.11

# Contents

# Chapter 1

# Class Index

## 1.1 Class List

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

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

# Chapter 3

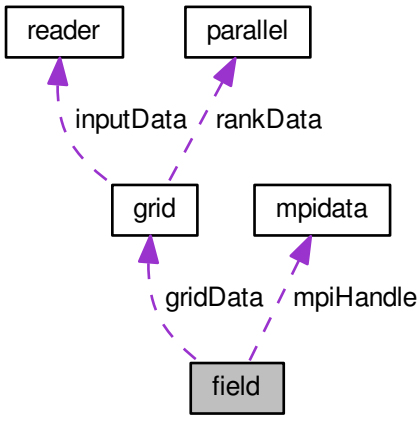
# Class Documentation

## 3.1 field Class Reference

A class which abstracts concept of a scalar field for Yee's algorithm.

`#include "lib/field.h"`

Collaboration diagram for field:




### Public Member Functions

- field (const grid &gridData, bool xStag_, bool yStag_, bool zStag_)

  *Constructor of the field class.*

### Public Attributes

- blitz::Array< double, 3 > F
- bool xStag
- bool yStag
- bool zStag
- int local_Nx
- int local_Ny
- int local_Nz
- const grid & gridData
- mpidata * mpiHandle

## 3.1.1 Detailed Description

A class which abstracts concept of a scalar field for Yee's algorithm.

Yee's algorithm requires different lattice positions for different components of E field and H field. These positions are represented by three boolean variables that keep track of whether the field component is staggered in X, Y and Z direction. Additionally, the class contains information about the array size of scalar field and constant reference to grid information.

## 3.1.2 Constructor & Destructor Documentation

### 3.1.2.1 field::field ( const grid & *gridData*, bool *xStag_*, bool *yStag_*, bool *zStag_* )

Constructor of the field class.

Initializes the data members using the input parameters and grid information. Resizes the Blitz array to required size and then initializes each element of array to zero. Also, initializes the MPI handle by calling the mpidata class.

**Parameters**

| | |
|---|---|
| *gridData* | is a constant reference to grid information. |
| *xStag_* | is a boolean representing whether the field is staggered in X direction. |
| *yStag_* | is a boolean representing whether the field is staggered in Y direction. |
| *zStag_* | is a boolean representing whether the field is staggered in Z direction. |

## 3.1.3 Member Data Documentation

### 3.1.3.1 blitz::Array<double, 3> field::F

The blitz array that contains the data of scalar field.

### 3.1.3.2 const grid& field::gridData

Array size of scalar field in Z direction.

#### 3.1.3.3 int field::local_Nx

Array size of scalar field in X direction.

#### 3.1.3.4 int field::local_Ny

Array size of scalar field in X direction.

#### 3.1.3.5 int field::local_Nz

Array size of scalar field in Y direction.

#### 3.1.3.6 mpidata* field::mpiHandle

MPI handle used to sync the field data between different MPI nodes.

#### 3.1.3.7 bool field::xStag

Boolean representing whether field is scattered in X direction.

#### 3.1.3.8 bool field::yStag

Boolean representing whether field is scattered in Y direction.

#### 3.1.3.9 bool field::zStag

Boolean representing whether field is scattered in Z direction.

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

- /home/rishabh/Luxum/lib/field.h
- /home/rishabh/Luxum/src/field.cc

## 3.2 fileio Class Reference

Class to read and write data from and to HDF5 files.

```
#include "lib/fileio.h"
```

### Public Member Functions

- fileio (const char *fileName, bool read, const grid &mesh, field *iField)

  *Constructor of the fileio class.*
- void writeHDF5 (double t)

  *Function to write scalar field data to HDF5 file.*
- void readHDF5 (std::string datasetName)

  *Function to read data from HDF5 file.*
- void closeFileio ()

  *Function to close fileio object.*

## 3.2.1 Detailed Description

Class to read and write data from and to HDF5 files.

The objects of this class can be made using pointer to a scalar field object. Once the rwHDF5 object is created, the member functions calls make it easy to read and write data.

writeHDF5() writes the current data stored in scalar field in the dataset name passed. readHDF5() reads the dataset name passed and writes in scalar field object.

## 3.2.2 Constructor & Destructor Documentation

### 3.2.2.1 fileio::fileio ( const char * *fileName,* bool *read,* const grid & *mesh,* field * *iField* )

Constructor of the fileio class.

The constructor initializes the datamembers according to the scalar field provided. The subarray that is contained in the specific MPI node is calculated according to the processor rank. The subarray is size is different for scalar field staggered differently and whether the object will be used to read or write the data.

**Parameters**

| | |
|---|---|
| *fileName* | is the name of file whose data is to be read or in which data is to be written. |
| *read* | is a boolean which is set to true if object will be used to read HDF5 file. |
| *mesh* | is a constant reference to grid information. |
| *iField* | is a reference to the scalar field object whose data is to be read or written. |

## 3.2.3 Member Function Documentation

### 3.2.3.1 void fileio::closeFileio ( )

Function to close fileio object.

Call this before ending the program. This ensures a smoothly written HDF5 file. Not calling this may cause file to get correupted.

#### 3.2.3.2 void fileio::readHDF5 ( std::string *datasetName* )

Function to read data from HDF5 file.

The function reads the dataset from HDF5 file and writes it to scalar field data. The necessary information is taken from datamember of the object calling the function.

**Parameters**

| | |
|---|---|
| *datasetName* | is name of dataset which is to be read. |

#### 3.2.3.3 void fileio::writeHDF5 ( double *t* )

Function to write scalar field data to HDF5 file.

The function write the data to scalar field. The necessary information is taken from datamember of the object calling the function.

**Parameters**

| | |
|---|---|
| *t* | is dataset number which can be changed for subsequent writes at different time steps. |

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

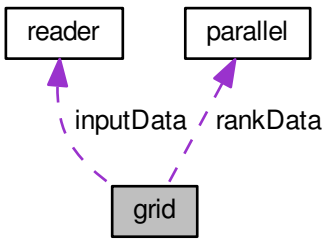
- /home/rishabh/Luxum/lib/fileio.h
- /home/rishabh/Luxum/src/fileio.cc

## 3.3 grid Class Reference

Contains information about local colloquated indices for specific MPI nodes.

`#include "lib/grid.h"`

Collaboration diagram for grid:



### Public Member Functions

- grid (const reader &solParam, parallel &parallelData)

  *Constructor of the grid class.*

### Public Attributes

- const reader & inputData
- const parallel & rankData
- int local_colloq_x
- int local_colloq_y
- int local_colloq_z
- int colloq_start_index_x
- int colloq_start_index_y
- int colloq_start_index_z
- int colloq_end_index_x
- int colloq_end_index_y
- int colloq_end_index_z
- bool isPlanar

## 3.3.1 Detailed Description

Contains information about local colloquated indices for specific MPI nodes.

The datamembers of this class contain information about how and which points are located in specific MPI node with respect to full global arrays.

## 3.3.2 Constructor & Destructor Documentation

### 3.3.2.1 grid::grid ( const reader & *solParam*, parallel & *parallelData* )

Constructor of the grid class.

Initializes the datamembers of grid class such the number of colloquated points, stating and ending index of colloquated points for specific MPI nodes. This is calculated using the information from object of parallel class which contains relevant information like number of available processors in different dimensions.

**Parameters**

| | |
|---|---|
| *solParam* | is a const reference to the global data contained in the reader class |
| *parallelData* | is a reference to the object of parallel class |

## 3.3.3 Member Data Documentation

### 3.3.3.1 int grid::colloq_end_index_x

Stores the ending global index in X-direction for colloquated points for a specific MPI node.

#### 3.3.3.2 int grid::colloq_end_index_y

Stores the ending global index in Y-direction for colloquated points for a specific MPI node.

#### 3.3.3.3 int grid::colloq_end_index_z

Stores the ending global index in Z-direction for colloquated points for a specific MPI node.

#### 3.3.3.4 int grid::colloq_start_index_x

Stores the starting global index in X-direction for colloquated points for a specific MPI node.

#### 3.3.3.5 int grid::colloq_start_index_y

Stores the starting global index in Y-direction for colloquated points for a specific MPI node.

#### 3.3.3.6 int grid::colloq_start_index_z

Stores the starting global index in Z-direction for colloquated points for a specific MPI node.

#### 3.3.3.7 const reader& grid::inputData

A const reference to the global variables stored in the reader class to access user set parameters

#### 3.3.3.8 bool grid::isPlanar

Boolean to store whether the grid is two-dimensional or not.

#### 3.3.3.9 int grid::local_colloq_x

Stores the number of colloquated points in X-direction for a specific MPI node.

#### 3.3.3.10 int grid::local_colloq_y

Stores the number of colloquated points in Y-direction for a specific MPI node.

#### 3.3.3.11 int grid::local_colloq_z

Stores the number of colloquated points in Z-direction for a specific MPI node.

#### 3.3.3.12 const parallel& grid::rankData

A reference to the global variables stored in the parallel class to access MPI related parameters

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

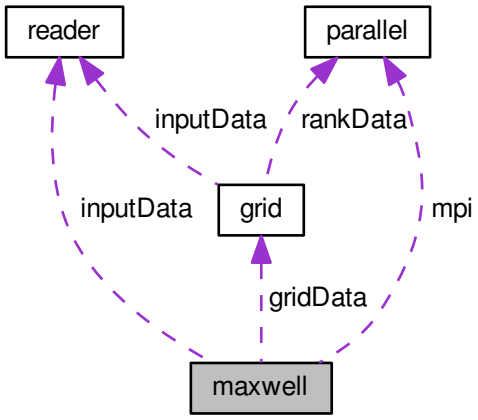
- /home/rishabh/Luxum/lib/grid.h
- /home/rishabh/Luxum/src/grid.cc

## 3.4 maxwell Class Reference

Contains many functions that implement Yee's algorithm along with various sources and boundary conditions.

`#include "lib/maxwell.h"`

Collaboration diagram for maxwell:




### Public Member Functions

- maxwell (reader &_inputData, parallel &_mpi, grid &_gridData)

  *Constructor of the maxwell class.*
- void solve ()

  *A function which decides the routine to call depending on the choice provided by user in the yaml file.*
- void solve3d ()

  *Function that calculates the timestepping solution when grid specified is three-dimensional without absorbing boundary condition.*
- void solve3d_pml ()

  *Function that calculates the timestepping solution when grid specified is three-dimensional with absorbing boundary condition.*
- void solve_planar ()

*Function that calculates the timestepping solution when grid specified is two-dimensional without absorbing boundary condition.*

- void solve_planar_pml ()

  *Function that calculates the timestepping solution when grid specified is two-dimensional with absorbing boundary condition.*

- void plane_wave_execute (vfield *curl, int timestep)

  *This function edits the curl array provided to it adding the plane-wave values at the boundary of total field and scattered field. The function also takes timestep number to calculate the value of plane-wave theoretically.*

- void plane_wave_execute (vfield *curl1, vfield *curl2, int timestep)

  *This function is the overload to plane_wave_execute(vfield *curl, int timestep).*

- void plane_wave_initialise ()

  *This function initialises certain variables according to the grid distribution over different different processors. These variables are crucial to functioning of plane_wave_execute().*

- int sigma_fn (int dim, bool is_E, blitz::Array< double, 1 > w)

  *This function calculates and stores the polynomial graded profile of loss term sigma in variable 'w'. The function can calculate the grading profile in all directions even for rectangular grid. The direction is specified by parameter 'dim' with its value being 0, 1, and 2 for X, Y, and Z direction respectively. The function is used to calculate profile for both E and H field. The parameter 'is_E' is set to true if profile is needed for E field otherwise false.*

- int k_fn (int dim, bool is_E, blitz::Array< double, 1 > w)

  *Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for strectched PML. The profile for this function is also polynomial. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.*

- int a_fn (int dim, bool is_E, blitz::Array< double, 1 > w)

  *Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for complex PML. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.*

- int b_fn (int dim, bool is_E, blitz::Array< double, 1 > w)

  *Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for complex PML. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.*

- int c_fn (int dim, bool is_E, blitz::Array< double, 1 > w)

  *Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for complex PML. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.*

- bool check_global_limits (blitz::TinyVector< int, 3 > v, bool xstag, bool ystag)

  *This small function simply checks whether a given global index specified by parameter v lies within the MPI node calling the function. This is checked using grid data and staggered nature of the scalar field whose index is being questioned.*

- blitz::TinyVector< int, 3 > global_to_local (blitz::TinyVector< int, 3 > glob)

  *This function returns a local index to the MPI node of global index, if the result from function check_global_limits() is true. This is used to position the point source in MPI node as user will provide the index in global index terms but it will be finally required in local index terms for a specific MPI node.*

## Public Attributes

- int n_threads
- int num_timesteps
- double epsilon0
- double mew0
- double S
- double c
- double dt
- double dx
- double dy
- double dz

- double a_max
- double k_max
- double m_pml
- double ma_pml
- double sigma_optimal
- double sigma_max
- double d_pml
- double sigma
- double init
- double lambda
- reader & inputData
- parallel & mpi
- grid & gridData
- bool usePW
- bool usePML
- bool usePS
- double H_x_dir
- double H_y_dir
- double H_z_dir
- double E_x_dir
- double E_y_dir
- double E_z_dir
- double kx
- double ky
- double kz
- int p1
- int q1
- int r1
- int p2
- int q2
- int r2
- int start_x
- int start_y
- int start_z
- int end_x
- int end_y
- int end_z
- int amp_source
- int omega_source
- int Nx_source
- int Ny_source
- int Nz_source

### 3.4.1 Detailed Description

Contains many functions that implement Yee's algorithm along with various sources and boundary conditions.

The class initializes various variables according to user's choice as read by reader class. The class has various functions that implement Yee's algorithm for two and three dimensions along with different boundary condition. The class also contains various functions that collectively implement plane-wave source using Total-field scattered-field formulation.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 maxwell::maxwell ( reader & *_inputData,* parallel & *_mpi,* grid & *_gridData* )

Constructor of the maxwell class.

Initialises various variables and constants according to conditions ans values given in yaml file.

Also calculates some quantities such as direction of H field according to directions specified of wave vector 'k' and E field.

Also calls the function plane_wave_initialise(), in case plane-waves are required by the user.

**Parameters**

| | |
|---|---|
| *_inputData* | is a reference to reader class object. |
| *_mpi* | is a reference to parallel class object. |
| *_gridData* | is a reference to grid class object. |

### 3.4.3 Member Function Documentation

#### 3.4.3.1 int maxwell::a_fn ( int *dim,* bool *is_E,* blitz::Array< double, 1 > *w* )

Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for complex PML. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.

**Parameters**

| | |
|---|---|
| *dim* | specifes the dimension in which grading profile is to be calculated. |
| *is_E* | is set to true if profile is required for E field, otherwise it is set to false. |
| *w* | is the variable where the calculated grading profile will be stored. |

#### 3.4.3.2 int maxwell::b_fn ( int *dim,* bool *is_E,* blitz::Array< double, 1 > *w* )

Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for complex PML. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.

**Parameters**

| | |
|---|---|
| *dim* | specifes the dimension in which grading profile is to be calculated. |
| *is_E* | is set to true if profile is required for E field, otherwise it is set to false. |
| *w* | is the variable where the calculated grading profile will be stored. |

#### 3.4.3.3 int maxwell::c_fn ( int *dim*, bool *is_E*, blitz::Array< double, 1 > *w* )

Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for complex PML. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.

**Parameters**

| | |
|---|---|
| *dim* | specifes the dimension in which grading profile is to be calculated. |
| *is_E* | is set to true if profile is required for E field, otherwise it is set to false. |
| *w* | is the variable where the calculated grading profile will be stored. |

#### 3.4.3.4 bool maxwell::check_global_limits ( blitz::TinyVector< int, 3 > *v*, bool *xstag*, bool *ystag* )

This small function simply checks whether a given global index specified by parameter v lies within the MPI node calling the function. This is checked using grid data and staggered nature of the scalar field whose index is being questioned.

**Parameters**

| | |
|---|---|
| *v* | is a three element Blitz TinyVector which conains the global index in question. |
| *xstag* | is a boolean which contains whether the field is question is staggered in X. |
| *ystag* | is a boolean which contains whether the field is question is staggered in Y. |

#### 3.4.3.5 blitz::TinyVector< int, 3 > maxwell::global_to_local ( blitz::TinyVector< int, 3 > *glob* )

This function returns a local index to the MPI node of global index, if the result from function check_global_limits() is true. This is used to position the point source in MPI node as user will provide the index in global index terms but it will be finally required in local index terms for a specific MPI node.

**Parameters**

| | |
|---|---|
| *glob* | is a three element Blitz TinyVector which conains the global index required to be converted to local index. |

#### 3.4.3.6 int maxwell::k_fn ( int *dim*, bool *is_E*, blitz::Array< double, 1 > *w* )

Similar to sigma_fn() this function also calculates another quantity useful for PML application, specifically for strectched PML. The profile for this function is also polynomial. The function calculates the profile for all directions in rectangular grid. The input parameters are similar to sigma_fn() and serve same function.

**Parameters**

| | |
|---|---|
| *dim* | specifes the dimension in which grading profile is to be calculated. |
| *is_E* | is set to true if profile is required for E field, otherwise it is set to false. |
| *w* | is the variable where the calculated grading profile will be stored. |

#### 3.4.3.7 void maxwell::plane_wave_execute ( vfield * curl, int timestep )

This function edits the curl array provided to it adding the plane-wave values at the boundary of total field and scattered field. The function also takes timestep number to calculate the value of plane-wave theoretically.

It is enough to add the values of plane wave at the boundary of an imaginary total field cube. This keeps the plane-wave sustained throughout the cube.

**Parameters**

| | |
|---|---|
| *curl* | is the poniter to curl array which needs to edited so that it contains the plane-wave values. |
| *timestep* | is the number of time-step at which the solution is currently at. |

#### 3.4.3.8 void maxwell::plane_wave_execute ( vfield * curl1, vfield * curl2, int timestep )

This function is the overload to plane_wave_execute(vfield *curl, int timestep).

It does the exact same thing but because 3D solutions require two curl arrays instead of one, it takes both of them as input and adds the plane-wave values accordingly.

**Parameters**

| | |
|---|---|
| *curl1* | is the poniter to first curl array which needs to edited so that it contains the plane-wave values. |
| *curl2* | is the poniter to second curl array which needs to edited so that it contains the plane-wave values. |
| *timestep* | is the number of time-step at which the solution is currently at. |

#### 3.4.3.9 void maxwell::plane_wave_initialise ( )

This function initialises certain variables according to the grid distribution over different different processors. These variables are crucial to functioning of plane_wave_execute().

The variables are initialised once and then used throughout all the timesteps without change. Hence, the function is called once by the constructor if plane-wave source is reuqired.

#### 3.4.3.10 int maxwell::sigma_fn ( int dim, bool is_E, blitz::Array< double, 1 > w )

This function calculates and stores the polynomial graded profile of loss term sigma in variable 'w'. The function can calculate the grading profile in all directions even for rectangular grid. The direction is specified by parameter 'dim' with its value being 0, 1, and 2 for X, Y, and Z direction respectively. The function is used to calculate profile for both E and H field. The parameter 'is_E' is set to true if profile is needed for E field otherwise false.

**Parameters**

| | |
|---|---|
| *dim* | specifes the dimension in which grading profile is to be calculated. |
| *is_E* | is set to true if profile is required for E field, otherwise it is set to false. |
| *w* | is the variable where the calculated grading profile will be stored. |

#### 3.4.3.11 void maxwell::solve ( )

A function which decides the routine to call depending on the choice provided by user in the yaml file.

#### 3.4.3.12 void maxwell::solve3d ( )

Function that calculates the timestepping solution when grid specified is three-dimensional without absorbing boundary condition.

#### 3.4.3.13 void maxwell::solve3d_pml ( )

Function that calculates the timestepping solution when grid specified is three-dimensional with absorbing boundary condition.

#### 3.4.3.14 void maxwell::solve_planar ( )

Function that calculates the timestepping solution when grid specified is two-dimensional without absorbing boundary condition.

#### 3.4.3.15 void maxwell::solve_planar_pml ( )

Function that calculates the timestepping solution when grid specified is two-dimensional with absorbing boundary condition.

### 3.4.4 Member Data Documentation

#### 3.4.4.1 double maxwell::a_max

PML parameter: Maximum value of a.

#### 3.4.4.2 int maxwell::amp_source

Amplitude of point source.

#### 3.4.4.3 double maxwell::c

Stores value of speed of light calculated from 'epsilon0' and 'mew0'.

#### 3.4.4.4 double maxwell::d_pml

PML parameter: Length of PML in units similar to that of dx.

#### 3.4.4.5 double maxwell::dt

Resolution in time.

#### 3.4.4.6 double maxwell::dx

Resolution in X-direction.

#### 3.4.4.7 double maxwell::dy

Resolution in Y-direction.

#### 3.4.4.8 double maxwell::dz

Resolution in Z-direction.

#### 3.4.4.9 double maxwell::E_x_dir

Coefficient representing X-direction of E-field of plane-wave.

#### 3.4.4.10 double maxwell::E_y_dir

Coefficient representing Y-direction of E-field of plane-wave.

#### 3.4.4.11 double maxwell::E_z_dir

Coefficient representing Z-direction of E-field of plane-wave.

#### 3.4.4.12 int maxwell::end_x

Stores the local X-index till where plane-wave has to go for specific MPI-node.

#### 3.4.4.13 int maxwell::end_y

Stores the local Y-index till where plane-wave has to go for specific MPI-node.

#### 3.4.4.14 int maxwell::end_z

Stores the local Z-index till where plane-wave has to go for specific MPI-node.

#### 3.4.4.15 double maxwell::epsilon0

Stores value of electric permeability of vaccum.

#### 3.4.4.16 grid& maxwell::gridData

Poiter to object of grid class.

#### 3.4.4.17 double maxwell::H_x_dir

Coefficient representing X-direction of H-field of plane-wave.

#### 3.4.4.18 double maxwell::H_y_dir

Coefficient representing Y-direction of H-field of plane-wave.

#### 3.4.4.19 double maxwell::H_z_dir

Coefficient representing Z-direction of H-field of plane-wave.

#### 3.4.4.20 double maxwell::init

Plane-wave source parameter: Offset value of Gaussian pulse with flat wave-front. It can be used to delay or advance the pulse arrival time in Total Field zone.

#### 3.4.4.21 reader& maxwell::inputData

Poiter to object of reader class.

#### 3.4.4.22 double maxwell::k_max

PML parameter: Maximum value of k.

#### 3.4.4.23 double maxwell::kx

Coefficient representing X-direction of wavevector of plane-wave.

#### 3.4.4.24 double maxwell::ky

Coefficient representing Y-direction of wavevector of plane-wave.

#### 3.4.4.25 double maxwell::kz

Coefficient representing Z-direction of wavevector of plane-wave.

#### 3.4.4.26 double maxwell::lambda

Wavelength of light.

#### 3.4.4.27 double maxwell::m_pml

PML parameter: Polynomial exponent for sigma grading.

#### 3.4.4.28 double maxwell::ma_pml

PML parameter: Polynomial exponent for a function grading.

#### 3.4.4.29 double maxwell::mew0

Stores value of magnetic permeability of vaccum.

#### 3.4.4.30 parallel& maxwell::mpi

Poiter to object of parallel class.

#### 3.4.4.31 int maxwell::n_threads

Stores the number of openMP threads per MPI node.

#### 3.4.4.32 int maxwell::num_timesteps

Number of total timesteps to be executed.

#### 3.4.4.33 int maxwell::Nx_source

Global X-index of point source location.

#### 3.4.4.34 int maxwell::Ny_source

Global Y-index of point source location.

#### 3.4.4.35 int maxwell::Nz_source

Global Z-index of point source location.

#### 3.4.4.36 int maxwell::omega_source

Angular-frequency of point source.

#### 3.4.4.37 int maxwell::p1

Global X-index marking start of Total-field zone for plane-wave.

#### 3.4.4.38 int maxwell::p2

Global X-index marking end of Total-field zone for plane-wave.

#### 3.4.4.39 int maxwell::q1

Global Y-index marking start of Total-field zone for plane-wave.

#### 3.4.4.40 int maxwell::q2

Global Y-index marking end of Total-field zone for plane-wave.

#### 3.4.4.41 int maxwell::r1

Global Z-index marking start of Total-field zone for plane-wave.

#### 3.4.4.42 int maxwell::r2

Global Z-index marking end of Total-field zone for plane-wave.

#### 3.4.4.43 double maxwell::S

Stores courant factor of Yee's algorithm.

#### 3.4.4.44 double maxwell::sigma

Plane-wave source parameter: Width of Gaussian pulse with flat wave-front.

#### 3.4.4.45 double maxwell::sigma_max

PML parameter: Maximum value of sigma function.

#### 3.4.4.46 double maxwell::sigma_optimal

PML parameter: Optimal value of sigma max for 5 to 10 cell thick PML.

#### 3.4.4.47 int maxwell::start_x

Stores the local X-index from where plane-wave has to start for specific MPI-node.

#### 3.4.4.48 int maxwell::start_y

Stores the local Y-index from where plane-wave has to start for specific MPI-node.

#### 3.4.4.49 int maxwell::start_z

Stores the local Z-index from where plane-wave has to start for specific MPI-node.

#### 3.4.4.50 bool maxwell::usePML

Boolean storing whether to use PML boundary condition or not.

#### 3.4.4.51 bool maxwell::usePS

Boolean storing whether to use point source or not.

#### 3.4.4.52 bool maxwell::usePW

Boolean storing whether to use plane-wave source or not.

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

- /home/rishabh/Luxum/lib/maxwell.h
- /home/rishabh/Luxum/src/maxwell.cc

## 3.5 mpidata Class Reference

Class to store MPI derived datatypes for individual arrays.

`#include "lib/mpidata.h"`

### Public Member Functions

- mpidata (blitz::Array< double, 3 > inputArray, bool xStag_, bool yStag_, const parallel &parallelData, const grid &gridData_)

  *Constructor of the mpidata class.*
- void createSubarrays (const bool xStag, const bool yStag, const bool zStag)

  *Function to create the subarray MPI_Datatypes.*
- void syncData ()

  *Function to sync data across all MPI nodes.*

## 3.5.1 Detailed Description

Class to store MPI derived datatypes for individual arrays.

Since the solver uses staggered grids, the data of scalar fields are stored in arrays of different sizes in each direction depending on whether the field is staggered or not in that direction. As a result, the limits of the sub-arrays to be sent across inter-processor boundaries is different for different arrays. Hence the **mpidata** class contains MPI_SUBARRAY derived datatypes to be initialized along with different fields in order to store their sub-arrays for inter-processor communication.

## 3.5.2 Constructor & Destructor Documentation

### 3.5.2.1 mpidata::mpidata ( blitz::Array< double, 3 > *inputArray,* bool *xStag,* bool *yStag,* const **parallel** & *parallelData,* const **grid** & *gridData_* )

Constructor of the mpidata class.

The short constructor of mpidata class merely resizes the array of shareRanks and MPI_Request datatypes.

**Parameters**

| | |
|---|---|
| *inputArray* | is the blitz array whose sub-arrays have to be created and synchronised across processors. |
| *xStag* | is boolean representing whether the scalar field provided is staggered in X direction. |
| *yStag* | is boolean representing whether the scalar field provided is staggered in Y direction. |
| *parallelData* | is a const reference to the global data contained in the parallel class. |
| *gridData* | is a const reference to the grid information. |

## 3.5.3 Member Function Documentation

### 3.5.3.1 void mpidata::createSubarrays ( const bool *xStag,* const bool *yStag,* const bool *zStag* )

Function to create the subarray MPI_Datatypes.

Function calculates the required subarrays that need to be shared across MPI nodes and stores them in MPI_Datatypes that can simply be used whenever syncing is required.

**Parameters**

| | |
|---|---|
| *xStag* | is boolean representing whether the scalar field provided is staggered in X direction. |
| *yStag* | is boolean representing whether the scalar field provided is staggered in Y direction. |
| *zStag* | is boolean representing whether the scalar field provided is staggered in Z direction. |

#### 3.5.3.2 void mpidata::syncData ( )

Function to sync data across all MPI nodes.

This is the core function of the mpidata class. The end slices of each sub-domain recieves data from their corresponding neighbouring sub-domains, while the interior slices of each sub-domain sends data to their corresponding neighbouring sub-domains.

All the data slices are send as subarray MPI derived data-types created in the createSubarrays function. As a result, syncData must be called only after the subarrays have been created.

The data transfer is implemented here with a mixture of blocking and non-blocking communication calls. The recieves are non-blocking, while the sends are blocking. This combination prevents inter-processor deadlock.

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

- /home/rishabh/Luxum/lib/mpidata.h
- /home/rishabh/Luxum/src/mpidata.cc

## 3.6 parallel Class Reference

Class for all the global variables and functions related to parallelization.

```
#include "lib/parallel.h"
```

### Public Member Functions

- parallel (const reader &iDat)

  *Constructor of the parallel class.*
- int findRank (int xR, int yR)

  *Function to calculate the global rank of a sub-domain using its xRank and yRank.*

### Public Attributes

- int rank
- int nProc
- blitz::Array< int, 1 > nearRanks

- const int npX
- const int npY

- int xRank
- int yRank

### 3.6.1 Detailed Description

Class for all the global variables and functions related to parallelization.

After MPI_Init, every process has its own rank. Moreover, after performing domain decomposition, each process has its own xRank and yRank to identify its position within the global computational domain. These data, along with the data to identify the neighbouring processes for inter-domain communication are stored in the **parallel** class. This class is initialized only once at the start of the solver.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 parallel::parallel ( const reader & *iDat* )

Constructor of the parallel class.

The initializing functions of MPI are called in order to get the total number of processes spawned, and the rank of each process. The xRank and yRank of each process are calculated and assigned. Finally, the ranks of neighbouring processes are found and stored in an array for use in MPI communications

**Parameters**

| | |
|---|---|
| *iDat* | is a const reference to the global data contained in the reader class |

### 3.6.3 Member Function Documentation

#### 3.6.3.1 int parallel::findRank ( int *xR*, int *yR* ) [inline]

Function to calculate the global rank of a sub-domain using its xRank and yRank.

The inline function computes the global rank while also checking if the input xRank and yRank lie within the limits prescribed for the solver.

**Parameters**

| | |
|---|---|
| *xR* | is the integer value of the sub-domain's xRank |
| *yR* | is the integer value of the sub-domain's yRank |

**Returns**

The integer value of the rank of the sub-domain; MPI_PROC_NULL if the xRank or yRank lie beyond limits

### 3.6.4 Member Data Documentation

#### 3.6.4.1 blitz::Array<int, 1> parallel::nearRanks

Array of ranks of the 4 neighbouring sub-domains - Left, Right, Front, Back

#### 3.6.4.2 int parallel::nProc

The total number of cores available for computation

#### 3.6.4.3 const int parallel::npX

npX and npY indicates the number of sub-domain divisions along the X and Y directions respectively

#### 3.6.4.4 const int parallel::npY

#### 3.6.4.5 int parallel::rank

The MPI rank of each sub-domain

#### 3.6.4.6 int parallel::xRank

xRank and yRank indicates the rank in terms of sub-domain divisions along the X and Y directions respectively. Like the global rank variable, these values also start from 0 to npX -1 and npY - 1 respectively.

#### 3.6.4.7 int parallel::yRank

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

- /home/rishabh/Luxum/lib/parallel.h
- /home/rishabh/Luxum/src/parallel.cc

## 3.7 reader Class Reference

Contains all the global variables set by the user through the yaml file.

```
#include "lib/reader.h"
```

### Public Member Functions

- reader ()

  *Contructor to reader class.*
- void readYAML ()

  *Function to open the yaml file and read the parameters.*
- void checkData ()

  *Function to perform a check on the consistency of user-set parameters.*

### Public Attributes

- int npX
- int npY
- double dx
- double dy
- double dz
- double S
- int num_timesteps
- int Nx
- int Ny
- int Nz
- int n_threads
- bool isPlanar
- bool usePW
- bool usePML
- bool usePS
- double kx_pw
- double ky_pw
- double kz_pw
- double Ex_pw
- double Ey_pw
- double Ez_pw
- double off_pw
- int x1_pw
- int y1_pw
- int z1_pw
- int x2_pw
- int y2_pw
- int z2_pw
- int Nx_source
- int Ny_source
- int Nz_source
- double amp_source
- double omega_source
- double d_pml
- double sigma
- double sigma_m
- double kappa
- double a_max
- double a_m

#### 3.7.1 Detailed Description

Contains all the global variables set by the user through the yaml file.

The class reads the paramters.yaml file and stores all the simulation paramters in publicly accessible constants. The class also has a function to check the consistency of the user set paramters and throw exceptions. The class is best initialized as a constant to prevent inadvertent tampering of the global variables it contains.

### 3.7.2 Constructor & Destructor Documentation

#### 3.7.2.1 reader::reader ( )

Contructor to reader class.

The brief constructor calls two functions - one to read the YAML file and other to validate the read arguments and abort the program if any dispute is found.

### 3.7.3 Member Function Documentation

#### 3.7.3.1 void reader::checkData ( )

Function to perform a check on the consistency of user-set parameters.

In order to catch potential errors early on, a few basic checks are performed here to validate the paramters set by the user. Additional checks to be performed on the paramters can be added to this function if necessary.

#### 3.7.3.2 void reader::readYAML ( )

Function to open the yaml file and read the parameters.

The function opens the parameters.yaml file and reads the simulation parameters into its member variables that are publicly accessible.

### 3.7.4 Member Data Documentation

#### 3.7.4.1 double reader::a_m

PML parameter: Polynomial exponent for a function grading.

#### 3.7.4.2 double reader::a_max

PML parameter: Maximum value of a.

#### 3.7.4.3 double reader::amp_source

Amplitude of point source.

#### 3.7.4.4 double reader::d_pml

PML parameter: Length of PML in units similar to that of dx.

#### 3.7.4.5 double reader::dx

Resolution in X-direction.

#### 3.7.4.6 double reader::dy

Resolution in Y-direction.

#### 3.7.4.7 double reader::dz

Resolution in Z-direction.

#### 3.7.4.8 double reader::Ex_pw

Coefficient representing X-direction of E-field of plane-wave.

#### 3.7.4.9 double reader::Ey_pw

Coefficient representing Y-direction of E-field of plane-wave.

#### 3.7.4.10 double reader::Ez_pw

Coefficient representing Z-direction of E-field of plane-wave.

#### 3.7.4.11 bool reader::isPlanar

Boolean to store whether the solution is planar or two-dimensional.

#### 3.7.4.12 double reader::kappa

PML parameter: Maximum value of kappa.

#### 3.7.4.13 double reader::kx_pw

Coefficient representing X-direction of wavevector of plane-wave.

#### 3.7.4.14 double reader::ky_pw

Coefficient representing Y-direction of wavevector of plane-wave.

#### 3.7.4.15 double reader::kz_pw

Coefficient representing Z-direction of wavevector of plane-wave.

#### 3.7.4.16 int reader::n_threads

Stores the number of openMP threads per MPI node.

#### 3.7.4.17 int reader::npX

Number of processors in X direction.

#### 3.7.4.18 int reader::npY

Number of processors in Y direction.

#### 3.7.4.19 int reader::num_timesteps

Number of total timesteps to be executed.

#### 3.7.4.20 int reader::Nx

Number of grid points in X-direction.

#### 3.7.4.21 int reader::Nx_source

Global X-index of point source location.

#### 3.7.4.22 int reader::Ny

Number of grid points in Y-direction.

#### 3.7.4.23 int reader::Ny_source

Global Y-index of point source location.

#### 3.7.4.24 int reader::Nz

Number of grid points in Z-direction.

#### 3.7.4.25 int reader::Nz_source

Global Z-index of point source location.

#### 3.7.4.26 double reader::off_pw

Offset for plane-wave pulse. Can be used to hasten or delay arrival of pulse.

#### 3.7.4.27 double reader::omega_source

Angular-frequency of point source.

#### 3.7.4.28 double reader::S

Stores courant factor of Yee's algorithm.

#### 3.7.4.29 double reader::sigma

Plane-wave source parameter: Width of Gaussian pulse with flat wave-front.

#### 3.7.4.30 double reader::sigma_m

PML parameter: Polynomial exponent for sigma grading.

#### 3.7.4.31 bool reader::usePML

Boolean storing whether to use PML boundary condition or not.

#### 3.7.4.32 bool reader::usePS

Boolean storing whether to use point source or not.

#### 3.7.4.33 bool reader::usePW

Boolean storing whether to use plane-wave source or not.

#### 3.7.4.34 int reader::x1_pw

Global X-index marking start of Total-field zone for plane-wave.

#### 3.7.4.35 int reader::x2_pw

Global X-index marking end of Total-field zone for plane-wave.

#### 3.7.4.36 int reader::y1_pw

Global Y-index marking start of Total-field zone for plane-wave.

#### 3.7.4.37 int reader::y2_pw

Global Y-index marking end of Total-field zone for plane-wave.

#### 3.7.4.38 int reader::z1_pw

Global Z-index marking start of Total-field zone for plane-wave.

#### 3.7.4.39 int reader::z2_pw

Global Z-index marking end of Total-field zone for plane-wave.

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

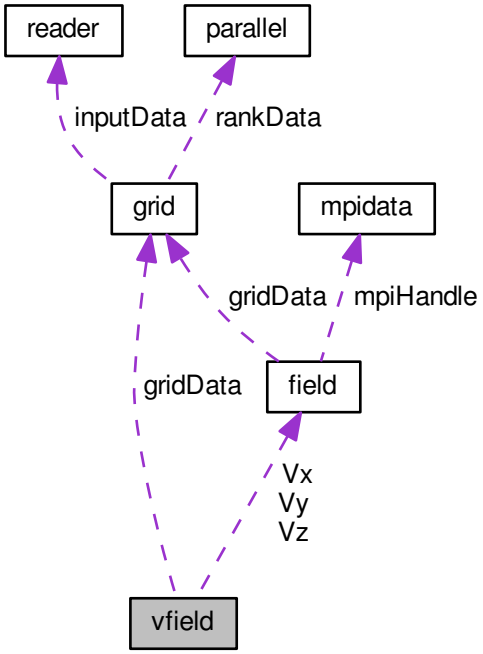
- /home/rishabh/Luxum/lib/reader.h
- /home/rishabh/Luxum/src/reader.cc

## 3.8 vfield Class Reference

A class which abstracts concept of a vector field for Yee's algorithm.

```
#include "lib/vfield.h"
```

Collaboration diagram for vfield:




## Public Member Functions

- vfield (const grid &gridData, bool isFaceCentered_)

  *Constructor of the vfield class.*
- void curl_3d (vfield *curl)

  *Calculates curl of vector field of corresponding to object calling it.*
- void curl_3d_adv (vfield *curl1, vfield *curl2)

  *Calculates curl of vector field of corresponding to object calling it.*
- void curl_planar (vfield *curl)

  *Calculates curl of vector field of corresponding to object calling it for planar grid.*
- void curl_planar_adv (vfield *curl1, vfield *curl2)

  *Calculates curl of vector field of corresponding to object calling it for planar grid.*

## Public Attributes

- field * Vx
- field * Vy
- field * Vz
- const grid & gridData
- bool isFaceCentered
- bool isPlanar

### 3.8.1 Detailed Description

A class which abstracts concept of a vector field for Yee's algorithm.

Yee's algorithm requires E and H vector fields at specific lattice points - face center and edge center. This class abtracts these fields out. The type of field can be specified using input parameter whether it is face centered or not.

The class also defines actions that can be performed on these vector fields. For now, the actions are only limited curl functions, but more can be easily added.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 vfield::vfield ( const grid & *gridData,* bool *isFaceCentered_* )

Constructor of the vfield class.

Initializes the data members, especially the pointers to scalar field objects are intialised according to need.

**Parameters**

| | |
|---|---|
| *gridData* | is a constant reference to grid information. |
| *isFaceCentered_* | is a boolean representing whether the vector field is face centered. |

### 3.8.3 Member Function Documentation

#### 3.8.3.1 void vfield::curl_3d ( vfield * *curl* )

Calculates curl of vector field of corresponding to object calling it.

The function takes one argument of vfield class where it stores the curl. The curl of a face centered vector field will be edge centered and vice-versa. The curl is only calculated if the vfield object passed and vfield object calling are of different types.

**Parameters**

| | |
|---|---|
| *curl* | is a pointer to vfield object where the calculated curl will be stored. |

#### 3.8.3.2 void vfield::curl_3d_adv ( vfield * *curl1,* vfield * *curl2* )

Calculates curl of vector field of corresponding to object calling it.

The function takes two argument of vfield class where it stores two parts of the curl. The two parts of the curl correspond to two final values corresponding to different dimensions which need to be subtracted to get the original curl in a given dimension. The first parameter contains the positive part of the curl. The second parameter contains the negative part of the curl. Actual curl = curl1 - curl2

The curl of a face centered vector field will be edge centered and vice-versa. The curl is only calculated if the vfield object passed and vfield object calling are of different types.

**Parameters**

| | |
|---|---|
| *curl1* | is a pointer to vfield object where positive part of the calculated curl will be stored. |
| *curl2* | is a pointer to vfield object where negative part of the calculated curl will be stored. |

#### 3.8.3.3 void vfield::curl_planar ( vfield * *curl* )

Calculates curl of vector field of corresponding to object calling it for planar grid.

The function takes one argument of vfield class where it stores the curl. The curl of a face centered vector field will be edge centered and vice-versa. The curl is only calculated if the vfield object passed and vfield object calling are of different types.

**Parameters**

| | |
|---|---|
| *curl* | is a pointer to vfield object where the calculated curl will be stored. |

#### 3.8.3.4 void vfield::curl_planar_adv ( vfield * *curl1*, vfield * *curl2* )

Calculates curl of vector field of corresponding to object calling it for planar grid.

The function takes two argument of vfield class where it stores two parts of the curl. The two parts of the curl correspond to two final values corresponding to different dimensions which need to be subtracted to get the original curl in a given dimension. The first parameter contains the positive part of the curl. The second parameter contains the negative part of the curl. Actual curl = curl1 - curl2

The curl of a face centered vector field will be edge centered and vice-versa. The curl is only calculated if the vfield object passed and vfield object calling are of different types.

**Parameters**

| | |
|---|---|
| *curl1* | is a pointer to vfield object where positive part of the calculated curl will be stored. |
| *curl2* | is a pointer to vfield object where negative part of the calculated curl will be stored. |

### 3.8.4 Member Data Documentation

#### 3.8.4.1 const grid& vfield::gridData

A constant reference to the grid information.

#### 3.8.4.2 bool vfield::isFaceCentered

Boolean representing whether the vector field is face centered.

#### 3.8.4.3 bool vfield::isPlanar

Boolean representing whether the grid is planar or two dimensional.

#### 3.8.4.4 field* vfield::Vx

Pointer to scalar field object which contains X component.

#### 3.8.4.5 field* vfield::Vy

Pointer to scalar field object which contains Y component.

#### 3.8.4.6 field* vfield::Vz

Pointer to scalar field object which contains Z component.

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

- /home/rishabh/Luxum/lib/vfield.h
- /home/rishabh/Luxum/src/vfield.cc

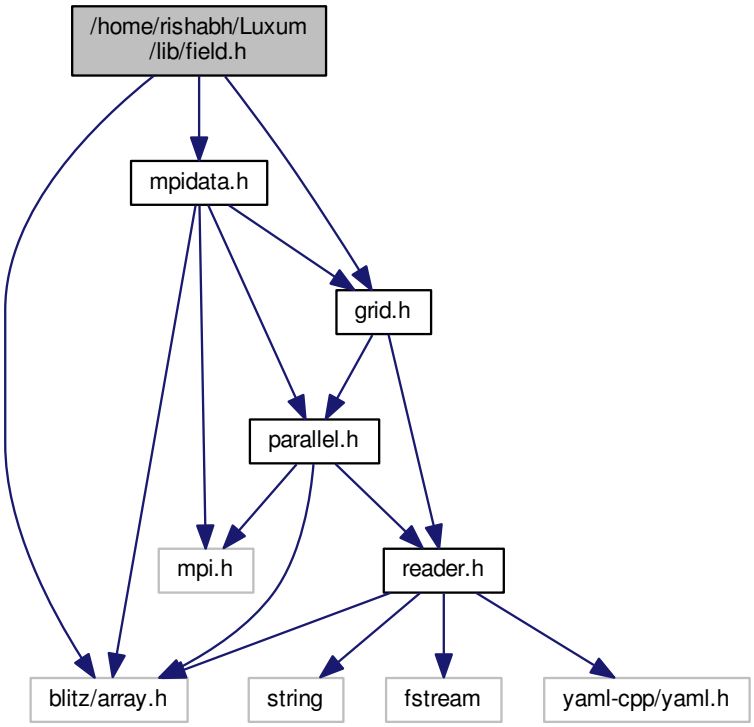
# Chapter 4

# File Documentation

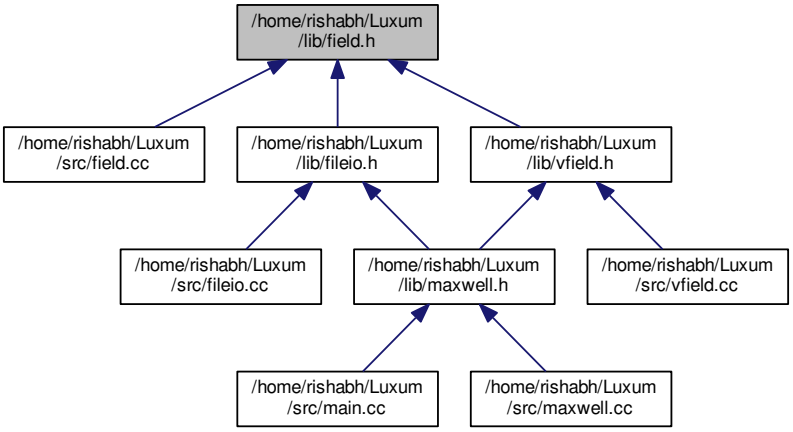
## 4.1 /home/rishabh/Luxum/lib/field.h File Reference

```
#include <blitz/array.h>
#include "mpidata.h"
#include "grid.h"
```

Include dependency graph for field.h:



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




### Classes

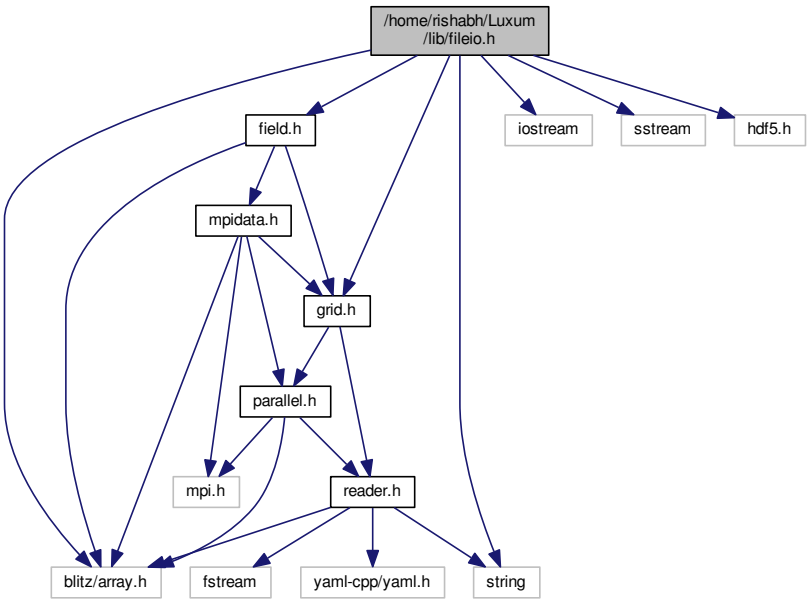
- class field

  *A class which abstracts concept of a scalar field for Yee's algorithm.*

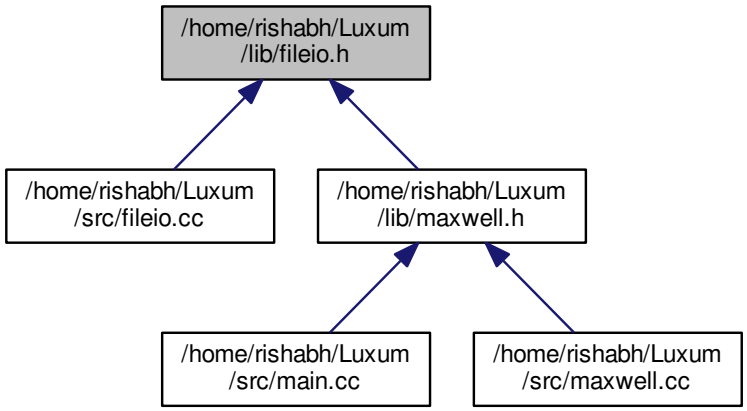
## 4.2 /home/rishabh/Luxum/lib/fileio.h File Reference

```
#include <blitz/array.h>
#include <iostream>
#include <string>
#include <sstream>
#include "field.h"
#include "grid.h"
#include "hdf5.h"
```

Include dependency graph for fileio.h:




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




## Classes

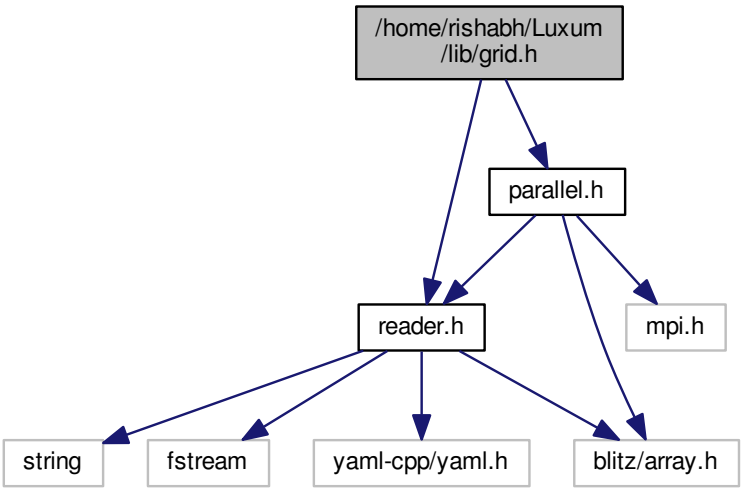
- class fileio

  *Class to read and write data from and to HDF5 files.*

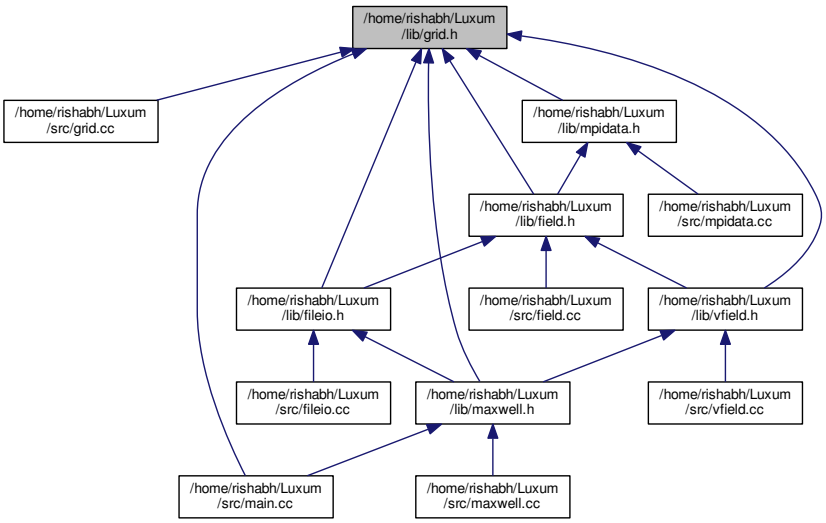
## 4.3 /home/rishabh/Luxum/lib/grid.h File Reference

```
#include "reader.h"
#include "parallel.h"
```

Include dependency graph for grid.h:




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




### Classes

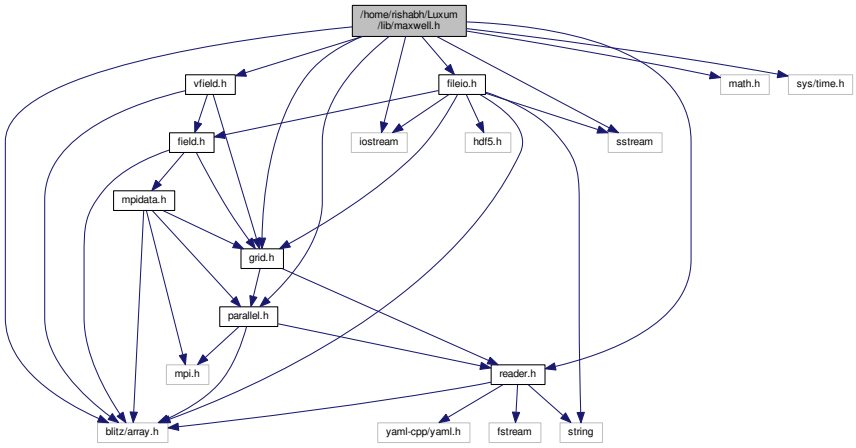
- class grid

  *Contains information about local colloquated indices for specific MPI nodes.*

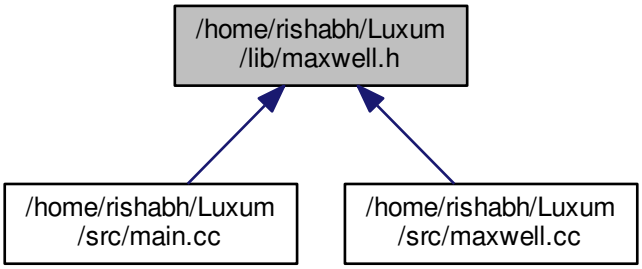
## 4.4 /home/rishabh/Luxum/lib/maxwell.h File Reference

```
#include <iostream>
#include <math.h>
#include <blitz/array.h>
#include "parallel.h"
#include "reader.h"
#include "grid.h"
#include "vfield.h"
#include "fileio.h"
#include <sstream>
#include <sys/time.h>
```

Include dependency graph for maxwell.h:




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




### Classes

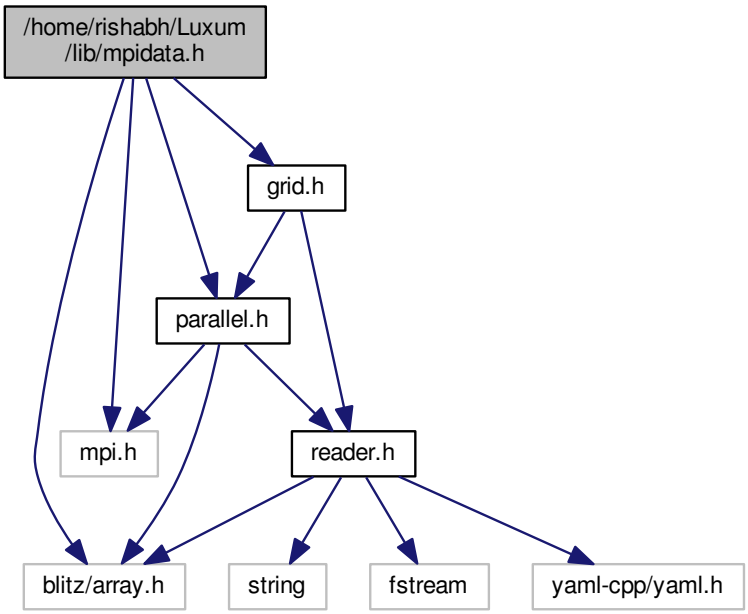
- class maxwell

  *Contains many functions that implement Yee's algorithm along with various sources and boundary conditions.*

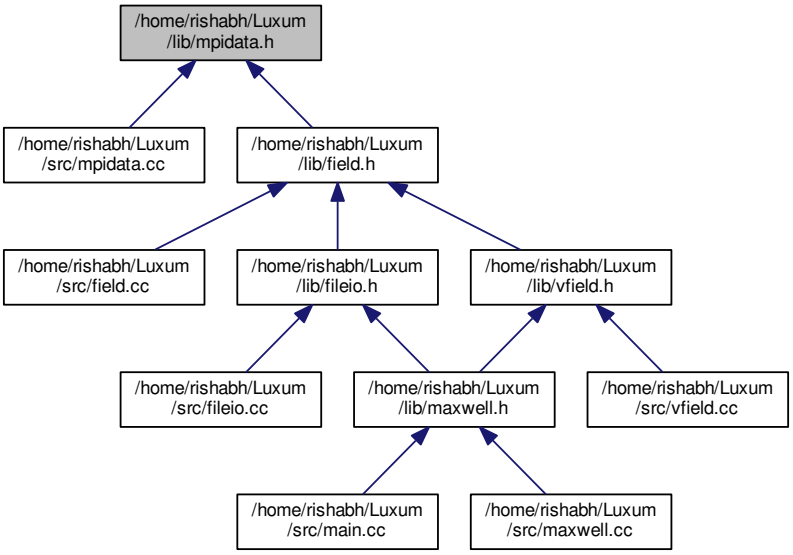
## 4.5 /home/rishabh/Luxum/lib/mpidata.h File Reference

```
#include <blitz/array.h>
#include <mpi.h>
#include "parallel.h"
#include "grid.h"
```

Include dependency graph for mpidata.h:



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




### Classes

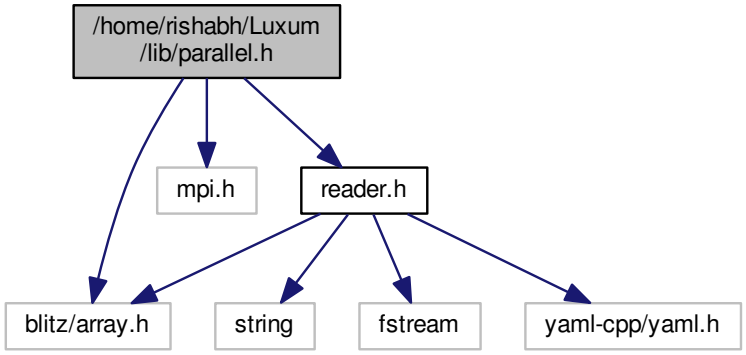
- class mpidata

  *Class to store MPI derived datatypes for individual arrays.*

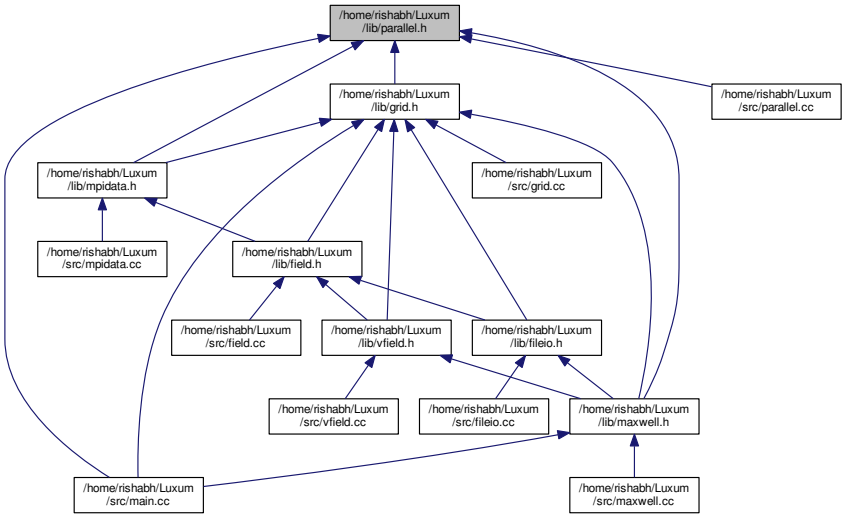
## 4.6 /home/rishabh/Luxum/lib/parallel.h File Reference

```
#include <blitz/array.h>
#include <mpi.h>
#include "reader.h"
```

Include dependency graph for parallel.h:



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




### Classes

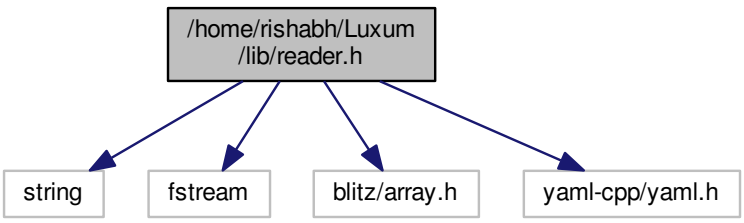
- class parallel

  *Class for all the global variables and functions related to parallelization.*

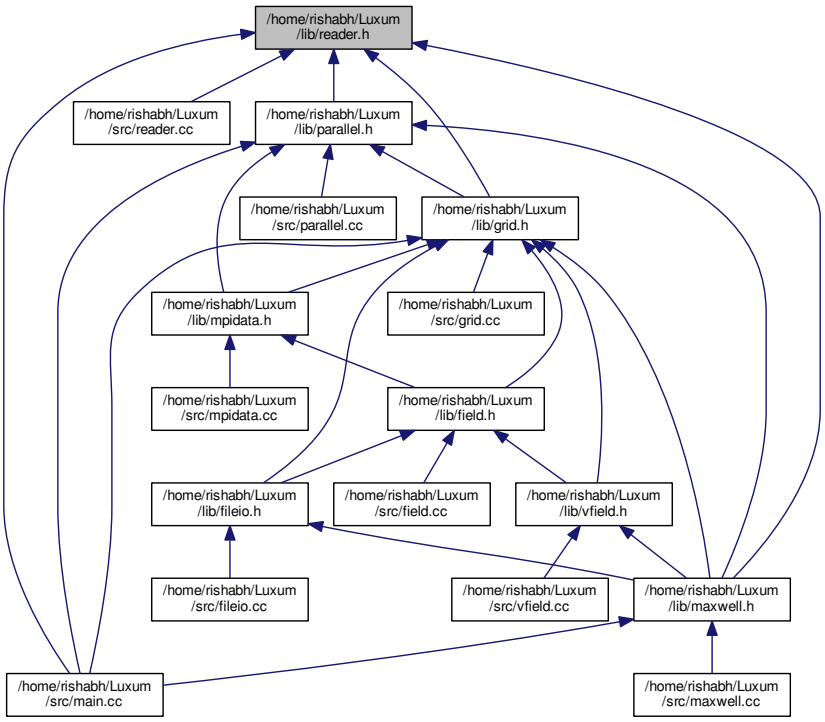
## 4.7 /home/rishabh/Luxum/lib/reader.h File Reference

```
#include <string>
#include <fstream>
#include <blitz/array.h>
#include <yaml-cpp/yaml.h>
```

Include dependency graph for reader.h:



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




## Classes

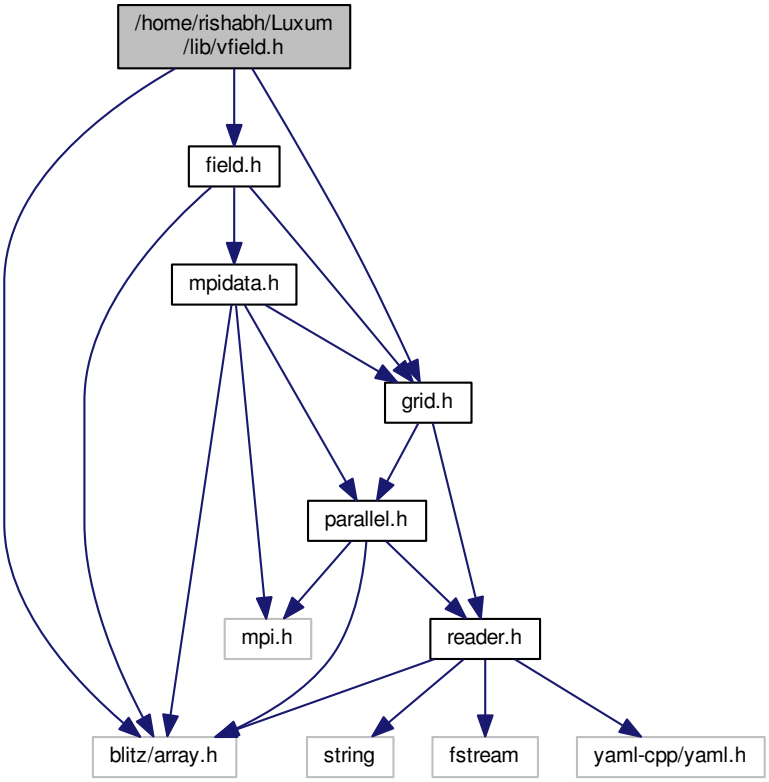
- class reader

  *Contains all the global variables set by the user through the yaml file.*

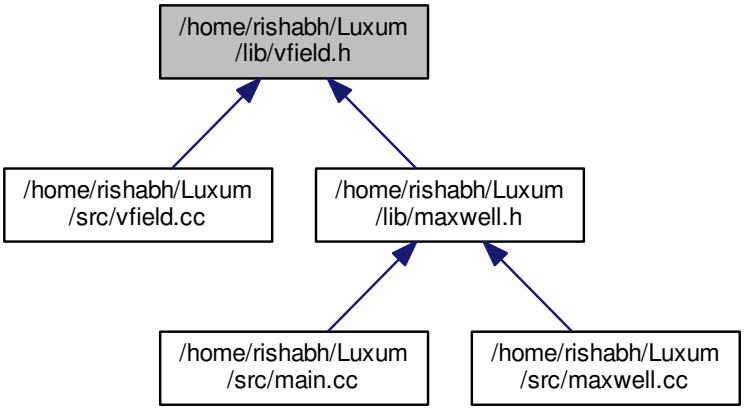
# 4.8 /home/rishabh/Luxum/lib/vfield.h File Reference

```
#include <blitz/array.h>
#include "field.h"
#include "grid.h"
```

Include dependency graph for vfield.h:




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



### Classes

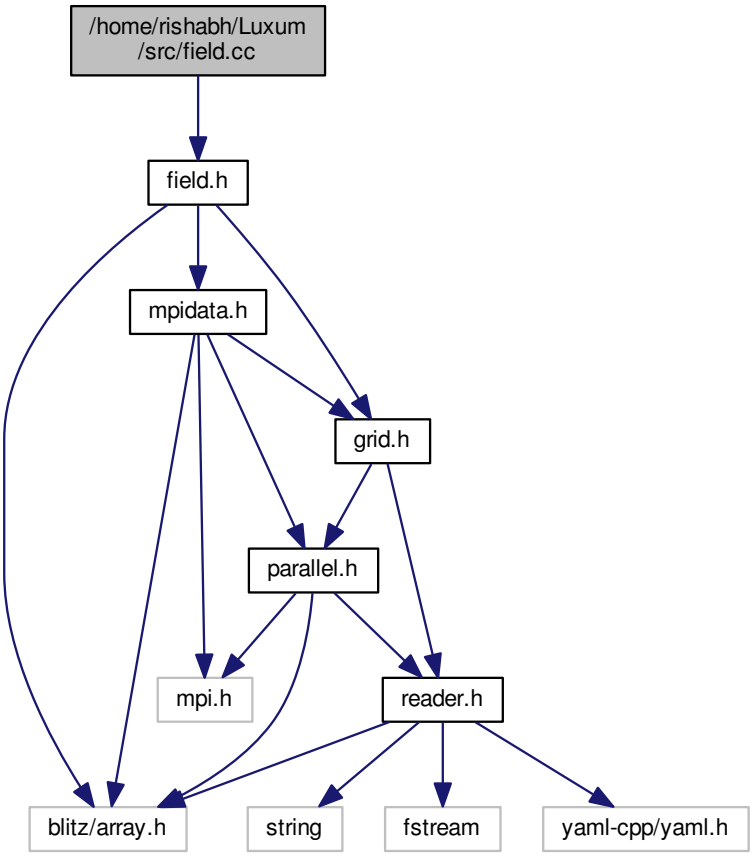
- class vfield

  *A class which abstracts concept of a vector field for Yee's algorithm.*

## 4.9 /home/rishabh/Luxum/src/field.cc File Reference

```
#include "field.h"
```

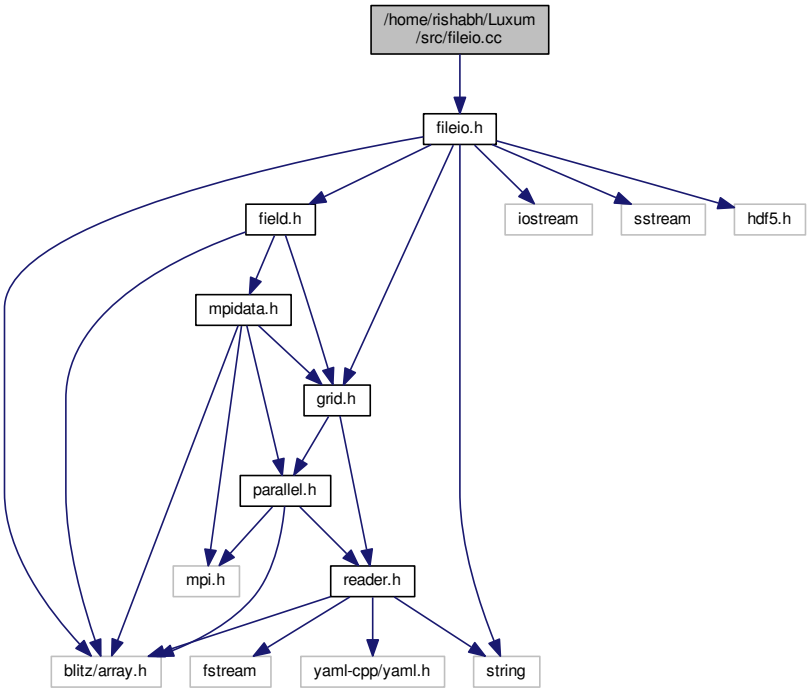
Include dependency graph for field.cc:




## 4.10 /home/rishabh/Luxum/src/fileio.cc File Reference

```
#include "fileio.h"
```

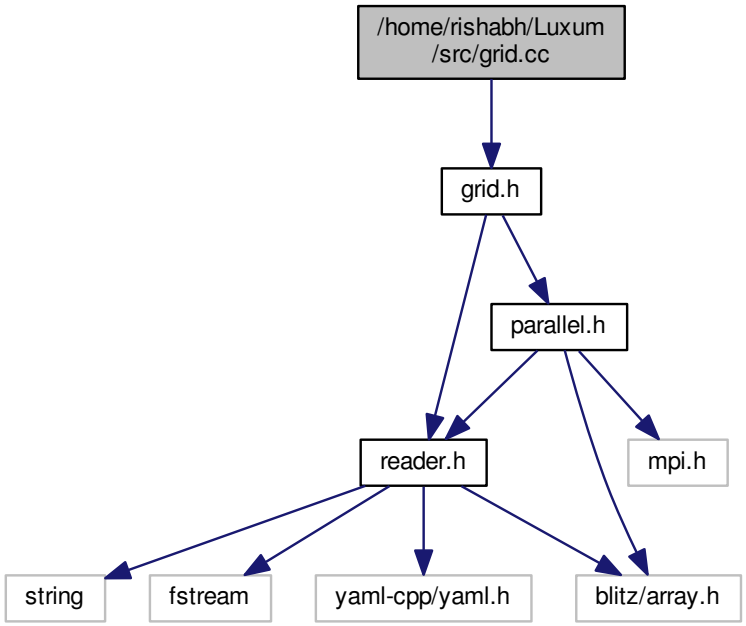
Include dependency graph for fileio.cc:




## 4.11 /home/rishabh/Luxum/src/grid.cc File Reference

```
#include "grid.h"
```

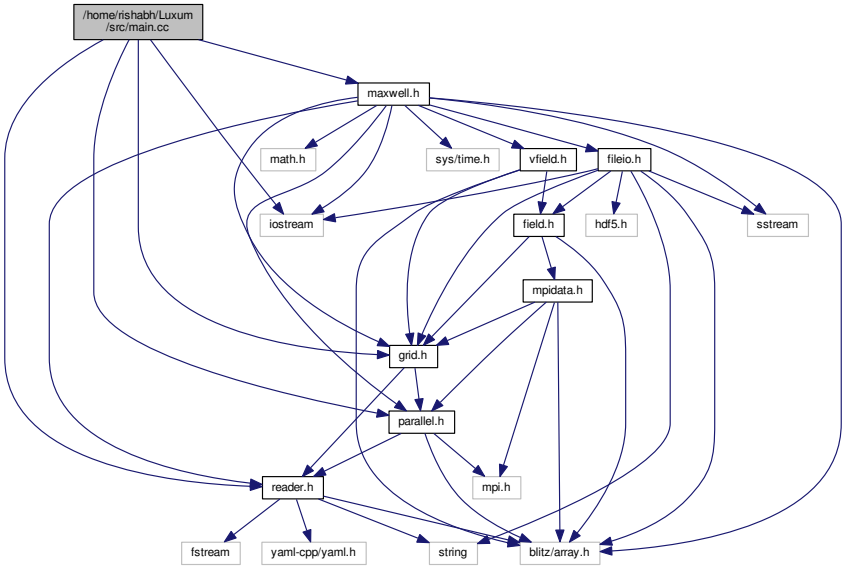
Include dependency graph for grid.cc:




## 4.12 /home/rishabh/Luxum/src/main.cc File Reference

```
#include <iostream>
#include "parallel.h"
#include "reader.h"
#include "grid.h"
#include "maxwell.h"
```

Include dependency graph for main.cc:




### Functions

- int main ()

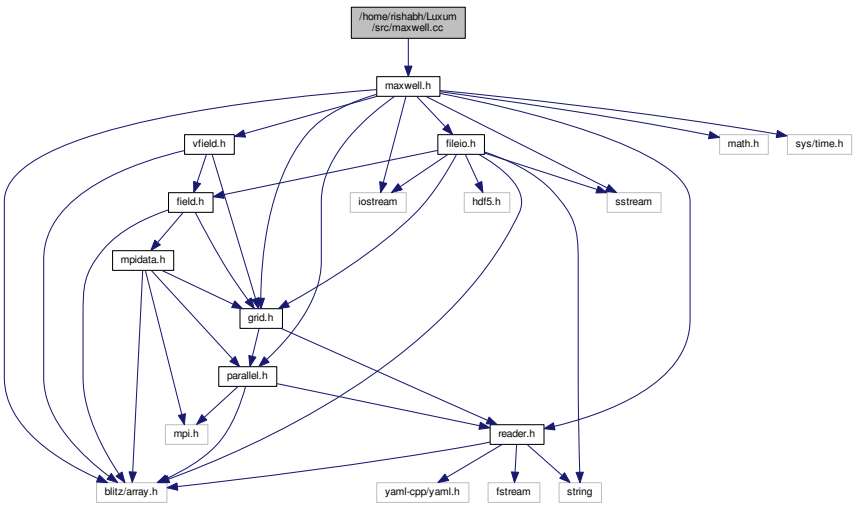
### 4.12.1 Function Documentation

#### 4.12.1.1 int main ( )

## 4.13 /home/rishabh/Luxum/src/maxwell.cc File Reference

```
#include "maxwell.h"
```

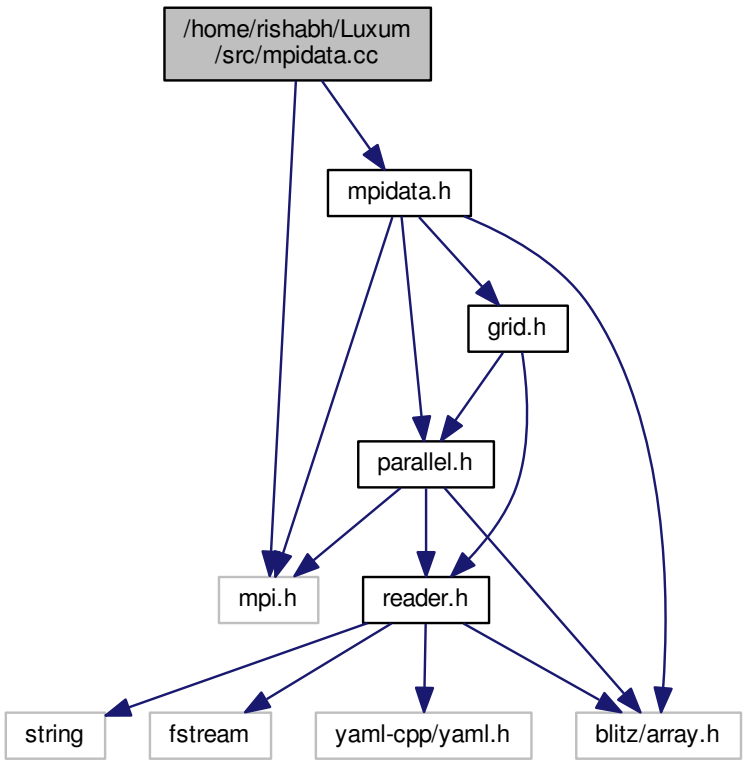
Include dependency graph for maxwell.cc:



## 4.14 /home/rishabh/Luxum/src/mpidata.cc File Reference

```
#include "mpidata.h"
#include <mpi.h>
```

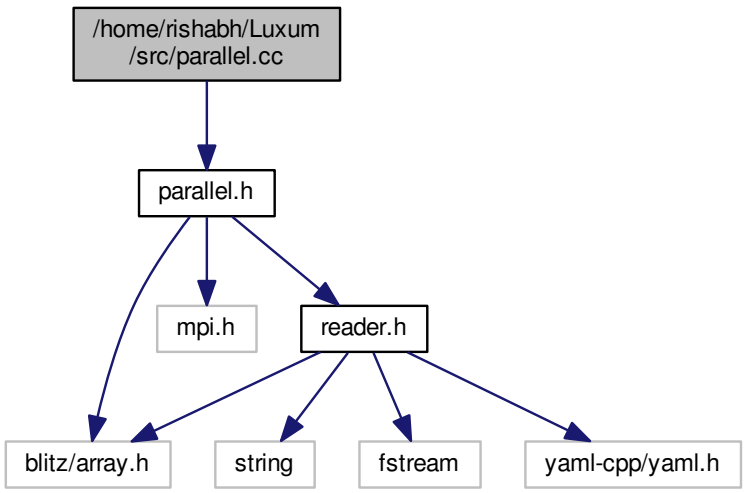
Include dependency graph for mpidata.cc:




## 4.15 /home/rishabh/Luxum/src/parallel.cc File Reference

```
#include "parallel.h"
```

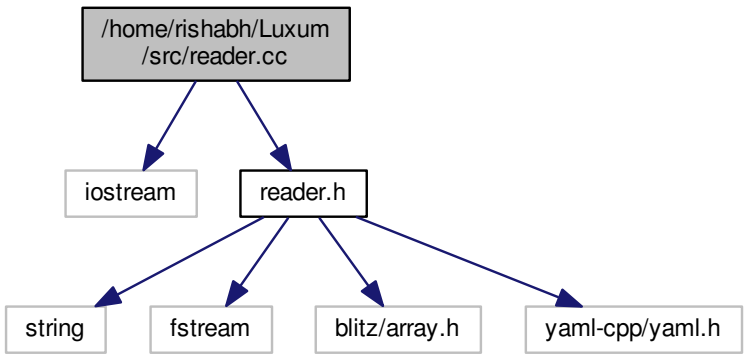
Include dependency graph for parallel.cc:




## 4.16 /home/rishabh/Luxum/src/reader.cc File Reference

```
#include <iostream>
#include "reader.h"
```

Include dependency graph for reader.cc:




## 4.17 /home/rishabh/Luxum/src/vfield.cc File Reference

```
#include <iostream>
```

```
#include "vfield.h"
```

Include dependency graph for vfield.cc: