LATField 2.0 documentation

Generated by Doxygen 1.8.3.1

Contents

Chapter 1

Main Page

1.1 Introduction

LATField 2.0, the second version of the LATfield library is a simple C++ framework for lattice based simulation on massively parallel computer.

Version 2.0 changes:

- the parallelization has been rebuilt to distribute 2 of the lattice dimensions into a 2-dimensional MPI process grid.

- A FFT wrapper for 3d cubic lattices.

- HDF5 I/O (serial and parallel).

- An I/O server has been implemented to improve I/O from compute nodes. (Beta)

1.2 Downloads

The current stable version can be downloaded at:

For user who would like to contribute to LATfield, the library git repository is on gitHub: `https://github.-com/daverio/LATfield2.git`

1.3 Getting Started

The best way to start with LATfield is to jump into the examples starting with the "getting started" example. The FFT example will explain the usage of the PlanFFT class and the IOserver example the IO server. There is a fourth example which show how to build a Poisson solver, but is currently not documented.

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 documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 cKSite Class Reference

A child of Site, built to work with the Fourier space lattices for complex to complex transforms.

```
#include <LATfield2_Site.hpp>
```

Inheritance diagram for cKSite:
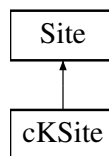
Public Member Functions

- **cKSite** (Lattice &lattice)
- **cKSite** (Lattice &lattice, long index)
- void **initialize** (Lattice &lattice)
- void **initialize** (Lattice &lattice, long index)
- cKSite **operator+** (int asked_direction)
- cKSite **operator-** (int asked_direction)
- int **coordLocal** (int asked_direction)
- int **coord** (int asked_direction)
- int **latCoord** (int direction)
- int **latCoordLocal** (int direction)
- bool **setCoord** (int ∗r_asked)
- bool **setCoord** (int x, int y, int z)

Private Attributes

- int **directions_** [3]

Additional Inherited Members

5.1.1 Detailed Description

A child of Site, built to work with the Fourier space lattices for complex to complex transforms.

A class which simplify the map of the field data array index. This class allow to get coordinate on the lattice, loop over each site of the lattice and perform displacment on the lattice.

WARNING: this site class must be used only on lattices initialized using initializeComplexFFT() method of the Lattice class.

This class have same binding that the Site class, so one can refer to the Site class for the documentation.

Examples:

> FFTs.

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

- LATfield2_Site.hpp

5.2 Field< FieldType > Class Template Reference

The Field class represent a field on a given lattice.

```
#include <LATfield2_Field.hpp>
```

Public Member Functions

- Field ()

  Constructor.
- Field (Lattice &lattice, int components=1)
- Field (Lattice &lattice, int matrixRows, int matrixCols, int symmetry=unsymmetric)
- ∼Field ()

  Destructor.
- void initialize (Lattice &lattice, int components=1)
- void initialize (Lattice &lattice, int rows, int cols, int symmetry=unsymmetric)
- void alloc ()
- void alloc (long size)
- void dealloc ()
- FieldType & operator() (long index)
- FieldType & operator() (long index, int component)
- FieldType & operator() (long index, int i, int j)
- FieldType & operator() (const Site &site)
- FieldType & operator() (const Site &site, int component)
- FieldType & operator() (const Site &site, int i, int j)
- FieldType & operator() (const cKSite &site)
- FieldType & operator() (const cKSite &site, int component)
- FieldType & operator() (const cKSite &site, int i, int j)
- FieldType & operator() (const rKSite &site)
- FieldType & operator() (const rKSite &site, int component)
- FieldType & operator() (const rKSite &site, int i, int j)
- void updateHalo ()
- void write (const string filename)
- void read (const string filename)
- void fastwrite (const string filename)
- void save (const string filename, void(∗FormatFunction)(fstream &, FieldType ∗, int)=defaultFieldSave< Field-Type >)
- void load (const string filename, void(∗FormatFunction)(fstream &, FieldType ∗, int)=defaultFieldLoad< Field-Type >)

- void [fastsave](const string filename, void(∗FormatFunction)(fstream &, FieldType ∗, int)=defaultFieldSave< FieldType >)
- void [saveHDF5](string filename)
- void [loadHDF5](string filename)
- void [saveHDF5_coarseGrain3D](string filename, int ratio)
- void [saveSliceHDF5](string filename, int xcoord, int thickness=1)
- [Lattice](#) & [lattice]()
- int [components]()
- int [rows]()
- int [cols]()
- int [symmetry]()
- FieldType ∗& [data]()

Public Attributes

- FieldType ∗ **data_**

Protected Attributes

- [Lattice](#) ∗ **lattice_**
- int **components_**
- int **rows_**
- int **cols_**
- int **symmetry_**
- unsigned int **sizeof_fieldType_**
- int **status_**
- hid_t **type_id**
- int **array_size**

Static Protected Attributes

- static int **initialized**
- static int **allocated**

Private Member Functions

- void **updateHaloComms** ()
- void **get_h5type** ()

5.2.1 Detailed Description

template< class FieldType >class Field< FieldType >

The [Field](#) class represent a field on a given lattice.

It stores the description of the field i.e. the datatype, the number of components, and the pointer to the field array in the memory.

The datatype is under user control; a field of structur or class can be also used (to be able to use the update halo method, the = operator must be defined or overloaded). However, the I/O support only native datatype and 1d array of them.

A field can be a single element, a vector of element or a 2d matrix of elements. In the case of a matrix it is possible to define the symmetry of the matrix:

LATfield2d::unsymmetric : no symmetry.

LATfield2d::symmetric : symmetric matrix (Tij = Tji)

Antisymmetric field are not yet implemented.

Examples:

> FFTs, gettingStarted, and poissonSolver.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 template<**class FieldType** > **Field**< **FieldType** >**::Field (Lattice &** *lattice,* **int** *components =* 1 **)**

Constructor of a "vector" field with initialization and allocation.

See Also

> initialize(Lattice& lattice, int components = 1)
> alloc()

Parameters

lattice	: lattice on which the field is defined
components	: number of components. The default is 1.

5.2.2.2 template<**class FieldType** > **Field**< **FieldType** >**::Field (Lattice &** *lattice,* **int** *matrixRows,* **int** *matrixCols,* **int** *symmetry =* unsymmetric **)**

Constructor of a "matrix" field with initialization and allocation.

See Also

> initialize(Lattice& lattice, int rows, int cols, int symmetry=unsymmetric)
> alloc()

Parameters

lattice	: lattice on which the field is defined
matrixRows	: matrix number of row .
matrixCols	: matrix number of colomn.
symmetry	: symmetry of the matrix, default is unsymmetric. LATfield2d::symmetric can be passed to specify the symmetry, reducing memory usage.

5.2.3 Member Function Documentation

5.2.3.1 template<**class FieldType** > **void Field**< **FieldType** >**::alloc ( )**

Memory allocation. Allocate the data_ array of this field. It allocated "components_∗lattice_->sitesLocal-Gross()∗sizeof(FieldType)" bytes. This method use malloc() to allocate the memory, in case the pointer is not allocated it will return a error message but not exiting the executable.

5.2.3.2 template<class FieldType > void Field< FieldType >::alloc (long *size*)

Memory allocation. Allocate the data_ array of this field. It allocated "size" bytes if "size" > "components_∗lattice_->sitesLocalGross()∗sizeof(FieldType)", if not it call this->alloc(). This method use malloc() to allocate the memory, in case the pointer is not allocated it will return a error message but not exiting the executable.

5.2.3.3 template<class FieldType > int Field< FieldType >::cols ( )

Returns the number of columns of the component matrix at each sites.

5.2.3.4 template<class FieldType > int Field< FieldType >::components ( )

Returns the number of components of the field at each sites.

5.2.3.5 template<class FieldType > FieldType ∗& Field< FieldType >::data ( )

Returns the pointer to the data_ array of the field.

5.2.3.6 template<class FieldType > void Field< FieldType >::dealloc ( )

Free the data_ array.

5.2.3.7 template<class FieldType > void Field< FieldType >::fastsave (const string *filename,* void(∗)(fstream &, FieldType ∗, int) *FormatFunction =* defaultFieldSave<FieldType>)

Method to write a field in ASCII. This method use serial I/O so can be very slow, but is faster than void write(const string filename). Should never be used! but it can be usefull on some architectures, where HDF5 is not installed and/or crashes the filesystem. There is no method to read back such a file. The file structur is dependent of the local geometry. This function dumps serially (in the paralle.lat_world_rank order) the data stored in each MPI process.

5.2.3.8 template<class FieldType > void Field< FieldType >::fastwrite (const string *filename*)

Method to write a field in Binary. This method uses serial I/O so can be very slow, but is faster than void save(const string filename). Should never be used! but it can be usefull on some architectures, where HDF5 is not installed and/or crashes the filesystem. There is no method to read back such a file. The file structure is dependent of the local geometry. This function dumps serially (in the paralle.lat_world_rank order) the data stored in each MPI process.

Parameters

filename,:	path to the file, from the executable folder.

5.2.3.9 template<class FieldType > void Field< FieldType >::initialize (Lattice & *lattice,* int *components =* 1)

Initialization of a "vector" field. Without allocation.

Parameters

lattice	: lattice on which the field is defined
components	: number of components. Default is 1.

5.2.3.10 **template**<**class FieldType** > **void Field**< **FieldType** >**::initialize (Lattice &** *lattice,* **int** *rows,* **int** *cols,* **int** *symmetry* **=** `unsymmetric` **)**

Initialization of a "matrix" field. Without allocation.

Parameters

lattice	: lattice on which the field is defined
matrixRows	: matrix number of row .
matrixCols	: matrix number of colomn.
symmetry	: symmetry of the matrix, default is unsymmetric. LATfield2d::symmetric can be pass to specify the symmetry.

5.2.3.11 **template**<**class FieldType** > **Lattice & Field**< **FieldType** >**::lattice (   )**

Returns a pointer to the lattice on which the field is defined.

5.2.3.12 **template**<**class FieldType** > **void Field**< **FieldType** >**::load (const string** *filename,* **void**(∗)**(fstream &, FieldType** ∗**, int)** *FormatFunction* **=** `defaultFieldLoad<FieldType>` **)**

Method to read a field in ASCII which have been written by the void write(const string filename) method.

Parameters

filename	: path to the file, from the executable folder.
Format-Function,:	format used for the writting procedure.

5.2.3.13 **template**<**class FieldType** > **void Field**< **FieldType** >**::loadHDF5 (string** *filename* **)**

Method to load a field with HDF5. To be able to use this method the flag HDF5 need to be set at compilation (-DHDF5). This method use serial HDF5 by default. For parallel HDF5 the flag -DH5_HAVE_PARALLEL must be set at compilation.

This methods will expect 1 dataset named field which contain all component of the field. If one want to use a dataset per components (named comp0 to compN) the flag -DH5_HAVE_PIXIE need to be set at compilation.

Parameters

filename	: path to the file, from the executable folder.

5.2.3.14 **template**<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (long** *index* **)** `[inline]`

Returns the value of the field stored in data_[index]. User should used operator()(const Site& site) to refer and access to the value of the field.

Parameters

index,:	displacment on the data_ array.

5.2.3.15 template$<$**class FieldType** $>$ **FieldType & Field**$<$ **FieldType** $>$**::operator() (long** *index,* **int** *component* **)**
`[inline]`

Returns the value of the field stored in data_[component + index∗components_]. User should used operator()(const Site& site, int component) to refer and access to the value of the field.

Parameters

index	: number of site to skip.
component,:	index of the desired component.

5.2.3.16 template$<$**class FieldType** $>$ **FieldType & Field**$<$ **FieldType** $>$**::operator() (long** *index,* **int** *i,* **int** *j* **)** `[inline]`

Returns the value of the field stored in data_[j∗rows_ + i + index∗components_]. In the symmetric case, it returns data_[abs(i-j) + min(i,j)∗(rows_+0.5-0.5∗min(i,j)) + index∗components_]. User should used operator()(const Site& site, int i, int j) to refer and access to the value of the field.

Parameters

index	: number of site to skip.
i	: index of the row
j	: index of the column

5.2.3.17 template$<$**class FieldType** $>$ **FieldType & Field**$<$ **FieldType** $>$**::operator() (const Site &** *site* **)** `[inline]`

Returns the value of the field at the position pointed by the Site object (data_[site.index()]). Can be used only for field with one component!

Parameters

site,:	a site instance which points to the desired lattice site.

See Also

> To have more description see the Site class documentation.

5.2.3.18 template$<$**class FieldType** $>$ **FieldType & Field**$<$ **FieldType** $>$**::operator() (const Site &** *site,* **int** *component* **)**
`[inline]`

Returns the value of a (vector) field's component at the position pointed by the Site object (data_[component + site.index()∗components_]).

Parameters

site,:	a site instance which points to the desired lattice site.
component,:	index of the desired component.

See Also

To have more description see the Site class documentation.

5.2.3.19 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const Site &** *site,* **int** *i,* **int** *j* **)**
`[inline]`

Returns the value of the (i,j) matrix component of the field at the position pointed by the Site object (data_[j∗rows_ + i + site.index∗components_]). In the symmetric case, it returns data_[abs(i-j) + min(i,j)∗(rows_+0.5-0.5∗min(i,j)) + site.index()∗components_].

Parameters

site,:	a site instance which points to the desired lattice site.
i	: index of the row
j	: index of the column

See Also

To have more description see the Site class documentation.

5.2.3.20 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const cKSite &** *site* **)** `[inline]`

Equivalent to FieldType& operator()(const Site& site) for cKsite

5.2.3.21 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const cKSite &** *site,* **int** *component* **)**
`[inline]`

Equivalent to FieldType& operator()(const Site& site, int component) for cKsite

5.2.3.22 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const cKSite &** *site,* **int** *i,* **int** *j* **)**
`[inline]`

Equivalent to FieldType& operator()(const Site& site, int i, int j) for cKsite

5.2.3.23 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const rKSite &** *site* **)** `[inline]`

Equivalent to FieldType& operator()(const Site& site) for rKsite

5.2.3.24 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const rKSite &** *site,* **int** *component* **)**
`[inline]`

Equivalent to FieldType& operator()(const Site& site, int component) for rKsite

5.2.3.25 template<**class FieldType** > **FieldType & Field**< **FieldType** >**::operator() (const rKSite &** *site,* **int** *i,* **int** *j* **)**
`[inline]`

Equivalent to FieldType& operator()(const Site& site, int i, int j) for rKsite

5.2.3.26 **template**<**class FieldType** > **void Field**< **FieldType** >**::read (const string** *filename* **)**

Method to read a field in binary which have been writen by the void write(const string filename) method.

Parameters

filename,:	path to the file, from the executable folder.

5.2.3.27 **template**<**class FieldType** > **int Field**< **FieldType** >**::rows ( )**

Returns the number of rows of the component matrix at each sites.

5.2.3.28 **template**<**class FieldType** > **void Field**< **FieldType** >**::save (const string** *filename,* **void(**∗**)(fstream &, FieldType** ∗**, int)** *FormatFunction* **=** `defaultFieldSave<FieldType>` **)**

Method to write a field in ASCII. This method use serial I/O so can be very slow. Should never be used during production, but can be useful during development.

Parameters

filename	: path to the file, from the executable folder.
Format-Function,:	format used for the writting procedure.

5.2.3.29 **template**<**class FieldType** > **void Field**< **FieldType** >**::saveHDF5 (string** *filename* **)**

Method to write a field with HDF5. To be able to use this method the flag HDF5 need to be set at compilation (-DHDF5). This method use serial HDF5 by default. For parallel HDF5 the flag -DH5_HAVE_PARALLEL must be used at compilation.

This methods will write 1 dataset (named "/field") which contain all components of the field. If one want to use a dataset per components (named "/comp0" to "/compN") the flag -DH5_HAVE_PIXIE need to be set at compilation.

Parameters

filename	: path to the file, from the executable folder.

5.2.3.30 **template**<**class FieldType** > **void Field**< **FieldType** >**::saveHDF5_coarseGrain3D (string** *filename,* **int** *ratio* **)**

A way to save coarse grained version of the fields. To be able to use this method the flag HDF5 need to be set at compilation (-DHDF5). Work only for 3D lattice!!!

This methods will write 1 dataset (named "/field") which contain all component of the field. If one want to use a dataset per components (named "/comp0" to "/compN") the flag -DH5_HAVE_PIXIE need to be set at compilation.

Parameters

filename	: path to the file, from the executable folder.
ration	: ration of the coarse graining. Must be an integer divider of the size of each dimension of this->lattice()

5.2.3.31 template$<$**class FieldType** $>$ **void Field**$<$ **FieldType** $>$**::saveSliceHDF5 (string** *filename,* **int** *xcoord,* **int** *thickness =* `1` **)**

Save a slice perpendicular to the first coordinate, at xcoord. To be able to use this method the flag HDF5 need to be set at compilation (-DHDF5).

This methods will write 1 dataset (named "/field") which contain all component of the field. If one want to use a dataset per components (named "/comp0" to "/compN") the flag -DH5_HAVE_PIXIE need to be set at compilation.

Parameters

filename	: path to the file, from the executable folder.
xcoord	: coordinate of the slice on the first dimension of the lattice.
thickness,:	thickness of the slice, the default is 1.

5.2.3.32 template$<$**class FieldType** $>$ **int Field**$<$ **FieldType** $>$**::symmetry ( )**

returns the symmetry of the component matrix at each sites.

5.2.3.33 template$<$**class FieldType** $>$ **void Field**$<$ **FieldType** $>$**::updateHalo ( )**

Update the halo sites (ghost cells) of the field. This method us the operator = to assign values, thefor be sure that this operator is defined or correctly overloaded when using field of class or struct.

5.2.3.34 template$<$**class FieldType** $>$ **void Field**$<$ **FieldType** $>$**::write (const string** *filename* **)**

Method to write a field in binary. This method use serial I/O so can be very slow. Should never be used during production, but can be usefull during development.

Parameters

filename,:	path to the file, from the executable folder.

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

- LATfield2_Field.hpp

5.3 file_struct Struct Reference

A structure to describe a file for the I/O server (dedicated MPI processes for writing to disks)

```
#include <LATfield2_IO_server.hpp>
```

Public Attributes

- string filename

    path to the file
- char ∗ data

    data array of the file
- long long size

    size of the local part of the file
- int type

    type of the file (currently only FILETYPE_UNSTRUCTURED)

5.3.1 Detailed Description

A structure to describe a file for the I/O server (dedicated MPI processes for writing to disks)

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

- LATfield2_IO_server.hpp

5.4 Imag Class Reference

A utility class for complex arithmetic, invested from LATfield 1.0.

```
#include <Imag.hpp>
```

Public Member Functions

- **Imag** (Real a, Real b)
- Imag **operator-** ()
- Imag **operator+** (Imag z)
- Imag **operator-** (Imag z)
- Imag **operator∗** (Imag z)
- Imag **operator/** (Imag z)
- void **operator=** (Real r)
- void **operator+=** (Imag z)
- void **operator-=** (Imag z)
- void **operator∗=** (Imag z)
- void **operator+=** (Real a)
- void **operator-=** (Real a)
- void **operator∗=** (Real a)
- void **operator/=** (Real a)
- Real & **real** ()
- Real & **imag** ()
- Real **phase** ()
- Imag **conj** ()
- Real **norm** ()

Private Attributes

- Real **data** [2]
- fftwf_complex **data**
- fftw_complex **data**

Friends

- Imag **operator+** (Imag z, Real a)
- Imag **operator+** (Real a, Imag z)
- Imag **operator-** (Imag z, Real a)
- Imag **operator-** (Real a, Imag z)
- Imag **operator∗** (Imag z, Real a)
- Imag **operator∗** (Real a, Imag z)
- Imag **operator/** (Imag z, Real a)
- Imag **sin** (Imag z)
- Imag **cos** (Imag z)
- Imag **expi** (Real x)
- std::ostream & **operator**$<<$ (ostream &os, Imag z)
- std::istream & **operator**$>>$ (istream &is, Imag &z)

5.4.1 Detailed Description

A utility class for complex arithmetic, invested from LATfield 1.0.

Complex number, defined as Real[2] if FFT capability of latfield are not used, and with FFTW complex if it is use. Commun operation over complex number are also defined.

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

  • Imag.hpp

5.5 IOserver Class Reference

A class to handle the I/O using MPI process reserved for IO purpose on which the files are defined.

```
#include <LATfield2_IO_server.hpp>
```

Public Member Functions

  • void start ()

    Server method (only called by server nodes)
  • void stop ()

    Client method (only called by compute nodes)
  • int openOstream ()

    Client method (only called by compute nodes)
  • void closeOstream ()

    Client method (only called by compute nodes)
  • ioserver_file createFile (string filename)

    Client method (only called by compute nodes)
  • void closeFile (ioserver_file fileID)

    Client method (only called by compute nodes)
  • void writeBuffer (ioserver_file fileID, char ∗buffer, int size)

    Client method (only called by compute nodes)
  • void initialize (int proc_size0, int proc_size1, int IOserver_size, int IO_node_size)

Protected Attributes

  • char ∗ **dataBuffer**

Private Attributes

  • bool **serverOn_flag**
  • bool **serverReady_flag**
  • bool **ostreamFile_flag**
  • MPI_Group **world_group_**
  • MPI_Group **IO_Group_**
  • MPI_Group **computeGroup_**
  • MPI_Comm **IO_Comm_**
  • MPI_Comm **computeComm_**
  • MPI_Group **syncLineGroup_**
  • MPI_Comm **syncLineComm_**
  • MPI_Group **masterClientGroup_**
  • MPI_Comm **masterClientComm_**

- MPI_Group **IO_NodeGroup_**
- MPI_Comm **IO_NodeComm_**
- int **IO_Rank_**
- int **computeRank_**
- int **syncLineRank_**
- int **IO_NodeRank_**
- file_struct ∗ **files**
- int **IO_ClientSize_**
- int **IO_NodeSize_**
- int **IO_Node_**
- MPI_Request **sendRequest**

5.5.1 Detailed Description

A class to handle the I/O using MPI process reserved for IO purpose on which the files are defined.

This server is in beta stage, but as such a functionality is very useful, it has been added to the stable part of LA-Tfield2. An example of the usage of this class is given in the IOserver example. User should never instanciate an IOserver object. The IOserver objet (IO_Server) is instanciate within the library header

5.5.2 Member Function Documentation

5.5.2.1 void IOserver::closeFile (ioserver_file *fileID*)

Client method (only called by compute nodes)

```
Method to close a new file: fileID.
```

Parameters

ioserver_file	fileID: file to close.

Examples:

IOserver.

5.5.2.2 void IOserver::closeOstream ( )

Client method (only called by compute nodes)

Method to close the current Ostream. After the stream is closed, the server will start to write the files it have in memory.

Examples:

IOserver.

5.5.2.3 int IOserver::createFile (string *filename*)

Client method (only called by compute nodes)

```
Method to create a new file, it return the fileID.
```

Parameters

filename,:	name of the file (including the path...)

Returns

   fileID.

Examples:

   [IOserver](#).

5.5.2.4   void IOserver::initialize (int *proc_size0,* int *proc_size1,* int *IOserver_size,* int *IO_node_size*)

Initialize the I/O server, this method is called by parallel.initialize(...). Should never be used!!!

5.5.2.5   int IOserver::openOstream ( )

Client method (only called by compute nodes)

```
Method to open an Ostream. Meaning a stream from the compute to the server processes.
```

Returns

   OSTREAM_SUCCESS if the stream is open.
   OSTREAM_FAIL if the stream cannot be open.

Examples:

   [IOserver](#).

5.5.2.6   void IOserver::start ( )

Server method (only called by server nodes)

Method which is called to start the server.

Examples:

   [IOserver](#).

5.5.2.7   void IOserver::stop ( )

Client method (only called by compute nodes)

Method which is called to stop the server.

Examples:

   [IOserver](#).

5.5.2.8   void IOserver::writeBuffer (ioserver_file *fileID,* char ∗ *buffer,* int *size*)

Client method (only called by compute nodes)

```
Method to write to a file.
!!! Beta, this method work only if fileID have been created and not closed!!!
```

Parameters

fileID,:	file where to write data.
buffer,:	pointer to the buffer to add to the file fileID.
size,:	size of "buffer", in byte.

Examples:

> IOserver.

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

- LATfield2_IO_server.hpp

5.6   Lattice Class Reference

The Lattice class describe a cartesian mesh (with 2 or more dimensions). The updateHalo method of the Field class generate the periodicity.

```
#include <LATfield2_Lattice.hpp>
```

Public Member Functions

- Lattice ()
    Constructor.
- Lattice (int dim, const int ∗size, int halo)
- Lattice (int dim, const int size, int halo)
- ∼Lattice ()
    Destructor.
- void initialize (int dim, const int ∗size, int halo)
- void initialize (int dim, const int size, int halo)
- void initializeRealFFT (Lattice &lat_real, int halo)
- void initializeComplexFFT (Lattice &lat_real, int halo)
- int dim ()
- int halo ()
- int ∗ size ()
- int size (int direction)
- int ∗ sizeLocal ()
- int sizeLocal (int direction)
- long sites ()
- long sitesGross ()
- long sitesLocal ()
- long sitesLocalGross ()
- long siteFirst ()
- long siteLast ()
- long jump (int direction)
- long sitesSkip ()

- long sitesSkip2d ()
- long ∗ coordSkip ()
- void save_arch (const string filename)
- bool is_arch_saved ()

Private Attributes

- int **status_**
- int **dim_**
- int ∗ **size_**
- long **sites_**
- long **sitesGross_**
- int **halo_**
- int ∗ **sizeLocal_**
- long **sitesLocal_**
- long **sitesLocalGross_**
- long ∗ **jump_**
- long **siteFirst_**
- long **siteLast_**
- long **sitesSkip_**
- long **sitesSkip2d_**
- long **coordSkip_** [2]
- int **arch_saved_**

Static Private Attributes

- static int **initialized**

5.6.1 Detailed Description

The Lattice class describe a cartesian mesh (with 2 or more dimensions). The updateHalo method of the Field class generate the periodicity.

It store the global and local geometry of the mesh. The last 2 dimension of the lattice are scattered into the MPI processes grid.

Examples:

FFTs, gettingStarted, and poissonSolver.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Lattice::Lattice (int *dim,* const int ∗ *size,* int *halo*)

Constructor with initialization

See Also

initialize(int dim, const int∗ size, int halo);

Parameters

dim	: number of dimension
size	: array containing the size of each dimension.
halo	: size of the halo (ghost cells, same for each dimension)

5.6.2.2   Lattice::Lattice (int *dim,* const int *size,* int *halo*)

Constructor with initialization

See Also

initialize(int dim, const int size, int halo);

Parameters

dim	: number of dimension
size	: size of each dimension (same for each dimension)
halo	: size of the halo (same for each dimension)

5.6.3   Member Function Documentation

5.6.3.1   long ∗ Lattice::coordSkip (   )

Returns

∗long. Pointer to an array which store the last 2 dimensions coordinate of the first local(in this MPI process) sites. Index 0 is for dim-1, index 1 is for dim-2/

Examples:

gettingStarted.

5.6.3.2   int Lattice::dim (   )

Returns

int. Number of dimensions of the lattice.

5.6.3.3   int Lattice::halo (   )

Returns

int. Size of the halo (ghost cells).

5.6.3.4   void Lattice::initialize (int *dim,* const int ∗ *size,* int *halo*)

Initialization of a dim-dimensional lattice, the size of each dimension is set by the second parameter: int ∗size. The ghost cell number (halo) is the same for each dimension.

Parameters

dim	: number of dimension
size	: array containing the size of each dimension.
halo	: size of the halo (same for each dimension)

5.6.3.5   void Lattice::initialize (int *dim,* const int *size,* int *halo*)

Initialization of a dim-dimensional lattice, each dimension have the same size. The ghost cell number (halo) is the same for each dimension.

Parameters

dim	: number of dimension
size	: size of each dimension (same for each dimension)
halo	: size of the halo (same for each dimension)

5.6.3.6 void Lattice::initializeComplexFFT (Lattice & *lat_real,* int *halo*)

Initialization of a lattice for Fourier space in case of complex to complex transform. The Fourier space lattice size is defined according to the real space one.. The fourier space lattice have "halo" ghost cells in each dimension (which can be different than the halo of the real space lattice).

Parameters

lat_real	: pointer to a real space lattice.
halo	: size of the halo (same for each dimension)

Examples:

> FFTs.

5.6.3.7 void Lattice::initializeRealFFT (Lattice & *lat_real,* int *halo*)

Initialization of a lattice for Fourier space in case of real to complex transform. The Fourier space lattice size is defined according to the real space one. The fourier space lattice have "halo" ghost cells in each dimension (which can be different than the halo of the real space lattice).

Parameters

lat_real	: pointer to a real space lattice.
halo	: size of the halo (same for each dimension)

Examples:

> FFTs, and poissonSolver.

5.6.3.8 bool Lattice::is_arch_saved ( )

Returns

> return true if the description of the lattice has been written on disk.

See Also

> save_arch(const string filename)

5.6.3.9 long Lattice::jump (int *direction*)

Function which return the number of data_ array elements to jump to move to the next site in the given direction. (does not take into account the number of component of the fields, therefor should be multiplied by Field.-components().) Should not be used by user.

Parameters

direction	: asked direction.

Returns

> long. Number of array elements to jump.

5.6.3.10   void Lattice::save‿arch (const string *filename*)

Function which save in serial and in ASCII the global and local description of the Lattice. Usefull to read a file writen by fast_save or fast_write methods of the Field class.

Parameters

filename	: filename of the architectur file.

5.6.3.11   long Lattice::siteFirst (   )

Returns

> long. Array index of the first site which is not within the halo.

5.6.3.12   long Lattice::siteLast (   )

Returns

> long. Array index of the last site which is not within the halo.

5.6.3.13   long Lattice::sites (   )

Returns

> long. Number of sites on the lattice (excluding halo sites).

Examples:

> poissonSolver.

5.6.3.14   long Lattice::sitesGross (   )

Returns

> long. Number of sites on the lattice (including halo sites).

5.6.3.15   long Lattice::sitesLocal (   )

Returns

> long. Number of sites (excluding halo sites) of the sublattice stored in this MPI process.

5.6.3.16   long Lattice::sitesLocalGross (   )

Returns

> long. Number of sites (including halo sites) of the sublattice stored in this MPI process.

5.6.3.17   long Lattice::sitesSkip ( )

Returns

long. Number of sites before first local site in lattice. Should not be used by users.

5.6.3.18   long Lattice::sitesSkip2d ( )

Returns

long. Number of sites before first local site in lattice. Should not be used by users.

5.6.3.19   int ∗ Lattice::size ( )

Returns

int∗. Pointer to the array of the size of each dimension of the lattice.

Examples:

gettingStarted, and poissonSolver.

5.6.3.20   int Lattice::size (int *direction*)

Function which returns the size of a given dimension of the lattice.

Parameters

direction	: asked dimension.

Returns

int. Global size of the lattice in the given dimension.

5.6.3.21   int ∗ Lattice::sizeLocal ( )

Returns

int∗. Pointer to the array of the size of each dimension of the sublattice stored in this MPI process.

Examples:

gettingStarted.

5.6.3.22   int Lattice::sizeLocal (int *direction*)

Function which returns the size of a given dimension of the sublattice stored in this MPI process.

Parameters

direction	: asked dimension.

Returns

int. Global size of the sublattice (of this MPI process) in the given dimension.

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

- LATfield2_Lattice.hpp

5.7 Parallel2d Class Reference

LATfield2d underliying class for paralleization.

```
#include <LATfield2_parallel2d.hpp>
```

Public Member Functions

- void initialize (int proc_size0, int proc_size1, int IO_total_size, int IO_node_size)
- void initialize (int proc_size0, int proc_size1)
- void abortForce ()
- void abortRequest ()
- void barrier ()
- template<class Type >
  void broadcast (Type &message, int from)
- template<class Type >
  void broadcast (Type *array, int len, int from)
- template<class Type >
  void broadcast_dim0 (Type &message, int from)
- template<class Type >
  void broadcast_dim0 (Type *array, int len, int from)
- template<class Type >
  void broadcast_dim1 (Type &message, int from)
- template<class Type >
  void broadcast_dim1 (Type *array, int len, int from)
- template<class Type >
  void sum (Type &number)
- template<class Type >
  void sum (Type *array, int len)
- template<class Type >
  void sum_dim0 (Type &number)
- template<class Type >
  void sum_dim0 (Type *array, int len)
- template<class Type >
  void sum_dim1 (Type &number)
- template<class Type >
  void sum_dim1 (Type *array, int len)
- template<class Type >
  void max (Type &number)
- template<class Type >
  void max (Type *array, int len)
- template<class Type >
  void max_dim0 (Type &number)
- template<class Type >
  void max_dim0 (Type *array, int len)
- template<class Type >
  void max_dim1 (Type &number)

- template<class Type >
  void max_dim1 (Type ∗array, int len)
- template<class Type >
  void min (Type &number)
- template<class Type >
  void min (Type ∗array, int len)
- template<class Type >
  void min_dim0 (Type &number)
- template<class Type >
  void min_dim0 (Type ∗array, int len)
- template<class Type >
  void min_dim1 (Type &number)
- template<class Type >
  void min_dim1 (Type ∗array, int len)
- template<class Type >
  void send (Type &message, int to)
- template<class Type >
  void send (Type ∗array, int len, int to)
- template<class Type >
  void send_dim0 (Type &message, int to)
- template<class Type >
  void send_dim0 (Type ∗array, int len, int to)
- template<class Type >
  void send_dim1 (Type &message, int to)
- template<class Type >
  void send_dim1 (Type ∗array, int len, int to)
- template<class Type >
  void receive (Type &message, int from)
- template<class Type >
  void receive (Type ∗array, int len, int from)
- template<class Type >
  void receive_dim0 (Type &message, int from)
- template<class Type >
  void receive_dim0 (Type ∗array, int len, int from)
- template<class Type >
  void receive_dim1 (Type &message, int from)
- template<class Type >
  void receive_dim1 (Type ∗array, int len, int from)
- template<class Type >
  void sendUp_dim0 (Type &bufferSend, Type &bufferRec, long len)
- template<class Type >
  void sendDown_dim0 (Type &bufferSend, Type &bufferRec, long len)
- template<class Type >
  void sendUpDown_dim0 (Type &bufferSendUp, Type &bufferRecUp, long lenUp, Type &bufferSendDown, Type &bufferRecDown, long lenDown)
- template<class Type >
  void sendUp_dim1 (Type &bufferSend, Type &bufferRec, long len)
- template<class Type >
  void sendDown_dim1 (Type &bufferSend, Type &bufferRec, long len)
- template<class Type >
  void sendUpDown_dim1 (Type &bufferSendUp, Type &bufferRecUp, long lenUp, Type &bufferSendDown, Type &bufferRecDown, long lenDown)
- int size ()
- int rank ()
- int world_size ()
- int world_rank ()

- int ∗ grid_size ()
- int ∗ grid_rank ()
- int root ()
- bool isRoot ()
- bool ∗ last_proc ()
- MPI_Comm lat_world_comm ()
- MPI_Comm ∗ dim0_comm ()
- MPI_Comm ∗ dim1_comm ()
- MPI_Group ∗ dim0_group ()
- MPI_Group ∗ dim1_group ()
- bool isIO ()

Private Attributes

- int **lat_world_size_**
- int **grid_size_** [2]
- int **lat_world_rank_**
- int **grid_rank_** [2]
- int **root_**
- bool **isRoot_**
- bool **last_proc_** [2]
- int **world_rank_**
- int **world_size_**
- MPI_Comm **world_comm_**
- MPI_Comm **lat_world_comm_**
- MPI_Comm ∗ **dim0_comm_**
- MPI_Comm ∗ **dim1_comm_**
- MPI_Group **world_group_**
- MPI_Group **lat_world_group_**
- MPI_Group ∗ **dim0_group_**
- MPI_Group ∗ **dim1_group_**
- bool **isIO_**

5.7.1 Detailed Description

LATfield2d underliying class for paralleization.

The parallel2d class is the handler of the parallelization of LATfield2d. LATfield2d distribute n-dimensional lattices into a 2-dimensional cartesian grid of MPI processes, a rod decomposition. The last dimension of the lattice is scattered into the first dimension of the process grid and the last-but-one dimension of the lattice is scattered into the second dimension of the process grid. This choice have been made to increase data locality of the ghost cells (halo), increases the efficiency of method to update them. Due to his scheme of parallelization, LATfield2d is only able to work with lattice of dimension bigger or equal to two.

The geometry of the process grid (the size of the 2 dimensions), two layers of MPI communicator and simple communication methods are enbended in the "parallel" object, which is an instance of the class Parallel. This object is instantiated but not initialized within the library header, hence the users should never declare an instance of the Parallel class. but rather use directly its pre/defined instance "parallel".

5.7.2 Member Function Documentation

5.7.2.1 void Parallel2d::abortForce ( )

Method to kill by force the executable. If one MPI process call this method the executable will be killed.

5.7.2.2 void Parallel2d::abortRequest ( )

Method to request to kill the executable. This method have to be call by every compute processes. It will wait every process had done the call before killing the executable.

5.7.2.3 void Parallel2d::barrier ( )

Method to call MPI_Barrier. This barrier is only applied on the compute processes. Every compute process have to perform the call, otherwise the executable will not continue.

Examples:

poissonSolver.

5.7.2.4 template$<$**class Type** $>$ **void Parallel2d::broadcast (Type &** *message,* **int** *from* **)**

Method to broadcast to every compute process a variable. Performed in lat_world_comm_ (compute processes world communicator).

Parameters

message,:	variable to send. the receivers will receive the value in that variable.
from,:	rank (in lat_world_comm_) of the sender.

5.7.2.5 template$<$**class Type** $>$ **void Parallel2d::broadcast (Type** $*$ *array,* **int** *len,* **int** *from* **)**

Method to broadcast to every compute process a variable array. Performed in lat_world_comm_ (compute processes world communicator).

Parameters

message	: pointer to the array to send. the receivers will receive the value in that variable.
len	: length of the array.
from	: rank (in lat_world_comm_) of the sender.

5.7.2.6 template$<$**class Type** $>$ **void Parallel2d::broadcast_dim0 (Type &** *message,* **int** *from* **)**

Method to perform a directional broadcast of a variable. The processes with grid_rank_[0]==from will broadcast the variable to every process which have same grid_rank_[1].

Parameters

message	: variable to send. the receivers will receive the value in that variable.
from	: grid_rank_[0] of the sender.

5.7.2.7 template$<$**class Type** $>$ **void Parallel2d::broadcast_dim0 (Type** $*$ *array,* **int** *len,* **int** *from* **)**

Method to perform a directional broadcast of a variable. The processes with grid_rank_[0]==from will broadcast the variable to every process which have same grid_rank_[1].

Parameters

message	: pointer to the array to send. the receivers will receive the value in that variable.

len	: length of the array.
from	: grid_rank_[0] of the sender.

5.7.2.8 template<**class Type** > **void Parallel2d::broadcast_dim1 (Type &** *message,* **int** *from* **)**

Method to perform a directional broadcast of a variable. The processes with grid_rank_[1]==from will broadcast the variable to every process which have same grid_rank_[0].

Parameters

message	: variable to send. the receivers will receive the value in that variable.
from	: grid_rank_[1] of the sender.

5.7.2.9 template<**class Type** > **void Parallel2d::broadcast_dim1 (Type** ∗ *array,* **int** *len,* **int** *from* **)**

Method to perform a directional broadcast of a variable. The processes with grid_rank_[1]==from will broadcast the variable to every process which have same grid_rank_[0].

Parameters

message	: pointer to the array to send. the receivers will receive the value in that variable.
len	: length of the array.
from	: grid_rank_[1] of the sender.

5.7.2.10 MPI_Comm ∗ **Parallel2d::dim0_comm ( )** `[inline]`

Returns

dim0_comm_ MPI_Comm array, array of directional communicator (dim 0, compute processes)

5.7.2.11 MPI_Group ∗ **Parallel2d::dim0_group ( )** `[inline]`

Returns

dim0_comm_ MPI_Group array, array of directional group (dim 0, compute processes)

5.7.2.12 MPI_Comm ∗ **Parallel2d::dim1_comm ( )** `[inline]`

Returns

dim1_comm_ MPI_Comm array, array of directional communicator (dim 1, compute processes)

5.7.2.13 MPI_Group ∗ **Parallel2d::dim1_group ( )** `[inline]`

Returns

dim1_comm_ MPI_Group array, array of directional group (dim 1, compute processes)

5.7.2.14 int∗ Parallel2d::grid_rank ( ) `[inline]`

Returns

  grid_size_ array of size 2. Rank on each dimension of the compute proceses grid.

Examples:

  gettingStarted, and IOserver.

5.7.2.15 int∗ Parallel2d::grid_size ( ) `[inline]`

Returns

  grid_size_ array of size 2. Size of each dimension of the compute processes grid.

Examples:

  gettingStarted.

5.7.2.16 void Parallel2d::initialize (int *proc_size0,* int *proc_size1,* int *IO_total_size,* int *IO_node_size*)

Overall LATfield2 initialization when the IO server is used. Should be the first call in any LATfield2 based application, as it initialize MPI (preprocessor define: -DEXTERNAL_IO)

Parameters

proc_size0	: size of the first dimension of the MPI process grid.
proc_size1	: size of the second dimension of the MPI process grid.
IO_total_size	: number of MPI process reserved for the IO server.
IO_node_size	: size of 1 goupe of process reserved for the IO server. Each group will write in a seperated file.

Examples:

  FFTs, gettingStarted, IOserver, and poissonSolver.

5.7.2.17 void Parallel2d::initialize (int *proc_size0,* int *proc_size1*)

Overall LATfield2 initialization used when the IO server is not used. Should be the first call in any LATfield2 based application, as it initialize MPI.

Parameters

proc_size0	: size of the first dimension of the MPI process grid.
proc_size1	: size of the second dimension of the MPI process grid.

5.7.2.18 bool Parallel2d::isIO ( ) `[inline]`

Returns

  isIO_ true if the process is reserved to the IO server, false if the process is a compute process.

Examples:

  IOserver.

5.7.2.19 **bool Parallel2d::isRoot ( )** `[inline]`

Returns

isRoot_ True for the compute root process, false if not the compute root process.

Examples:

poissonSolver.

5.7.2.20 **bool∗ Parallel2d::last_proc ( )** `[inline]`

Returns

last_proc_ array of size 2 containing the rank of the last process in each dimension of the compute processes grid.

5.7.2.21 **MPI_Comm Parallel2d::lat_world_comm ( )** `[inline]`

Returns

lat_world_comm_ MPI_Comm, the communicator which contains all compute processes.

5.7.2.22 **template<class Type > void Parallel2d::max (Type &** *number* **)**

Method to find the maximum value of a variable across all the compute processes.

Parameters

number	: number to compare, the max value will be assignent to this variable.

Examples:

poissonSolver.

5.7.2.23 **template<class Type > void Parallel2d::max (Type ∗** *array,* **int** *len* **)**

Method to find the maximum value of an array across all the compute processes.

Parameters

array	: array of numbers to compare, the max value of each element will be assignent to this variable.
len	: size of the array.

5.7.2.24 **template<class Type > void Parallel2d::max_dim0 (Type &** *number* **)**

Method to find the maximum value of a variable across all the compute processes with the same grid_rank_[1].

Parameters

number	: number to compare, the max value will be assignent to this variable.

5.7.2.25 **template**<**class Type** > **void Parallel2d::max dim0 (Type** ∗ *array,* **int** *len* **)**

Method to find the maximum value of a variable across all the compute processes with the same grid_rank_[1].

Parameters

array	: number to compare, the max value will be assignment to this variable.
len	: size of the array.

5.7.2.26 **template**<**class Type** > **void Parallel2d::max dim1 (Type &** *number* **)**

Method to find the maximum value of a variable across all the compute processes with the same grid_rank_[0].

Parameters

number	: number to compare, the max value will be assignment to this variable.

5.7.2.27 **template**<**class Type** > **void Parallel2d::max dim1 (Type** ∗ *array,* **int** *len* **)**

Method to find the maximum value of a variable across all the compute processes with the same grid_rank_[0].

Parameters

array	: number to compare, the max value will be assignment to this variable.
len	: size of the array.

5.7.2.28 **template**<**class Type** > **void Parallel2d::min (Type &** *number* **)**

Method to find the minimal value of a variable across all the compute processes.

Parameters

number	: number to compare, the max value will be assignment to this variable.

Examples:

    poissonSolver.

5.7.2.29 **template**<**class Type** > **void Parallel2d::min (Type** ∗ *array,* **int** *len* **)**

Method to find the minimal value of an array across all the compute processes.

Parameters

array	: number to compare, the max value will be assignment to this variable.
len	: size of the array.

5.7.2.30 **template**<**class Type** > **void Parallel2d::min dim0 (Type &** *number* **)**

Method to find the minimal value of a variable across all the compute processes with the same grid_rank_[1].

Parameters

number	: number to compare, the max value will be assignent to this variable.

5.7.2.31   template<class Type > void Parallel2d::min_dim0 (Type ∗ *array,* int *len*)

Method to find the minimal value of a variable across all the compute processes with the same grid_rank_[1].

Parameters

array	: number to compare, the max value will be assignent to this variable.
len	: size of the array.

5.7.2.32   template<class Type > void Parallel2d::min_dim1 (Type & *number*)

Method to find the minimal value of a variable across all the compute processes with the same grid_rank_[0].

Parameters

number	: number to compare, the max value will be assignent to this variable.

5.7.2.33   template<class Type > void Parallel2d::min_dim1 (Type ∗ *array,* int *len*)

Method to find the maximum value of a variable across all the compute processes with the same grid_rank_[0].

Parameters

array	: number to compare, the max value will be assignent to this variable.
len	: size of the array.

5.7.2.34   int Parallel2d::rank ( ) `[inline]`

Returns

lat_world_rank_ rank of this process (in the compute world). This rank is set to -1 for IO processes.

Examples:

gettingStarted, and IOserver.

5.7.2.35   template<class Type > void Parallel2d::receive (Type & *message,* int *from*)

MPI receive method on the compute processes. The method calls MPI_Recv in the lat_world_comm communicator.

Parameters

message	: variable which will be assigned to the receive message.
from	: rank of the sender. (in lat_world_comm_)

5.7.2.36   template<class Type > void Parallel2d::receive (Type ∗ *array,* int *len,* int *from*)

MPI receive method on the compute processes. The method call MPI_Recv in the lat_world_comm communicator.

Parameters

message	: variable which will be assigned to the receive message.
len	: size of the array to be received.
from	: rank of the sender. (in lat_world_comm_)

5.7.2.37 template$<$**class Type** $>$ **void Parallel2d::receive_dim0 (Type &** *message,* **int** *from* **)**

MPI receive method on the compute processes. The method call MPI_Recv in the directional communicator associated with the process caller. (direction=0)

Parameters

message	: variable which will be assigned to the receive message.
from	: rank of the sender. (grid_rank_[0])

5.7.2.38 template$<$**class Type** $>$ **void Parallel2d::receive_dim0 (Type** $*$ *array,* **int** *len,* **int** *from* **)**

MPI receive method on the compute processes. The method call MPI_Recv in the directional communicator associated with the process caller. (direction=0)

Parameters

message	: variable which will be assigned to the receive message.
len	: size of the array to be received.
from	: rank of the sender. (grid_rank_[0])

5.7.2.39 template$<$**class Type** $>$ **void Parallel2d::receive_dim1 (Type &** *message,* **int** *from* **)**

MPI receive method on the compute processes. The method call MPI_Recv in the directional communicator associated with the process caller. (direction=1)

Parameters

message	: variable which will be assigned to the receive message.
from	: rank of the sender. (grid_rank_[1])

5.7.2.40 template$<$**class Type** $>$ **void Parallel2d::receive_dim1 (Type** $*$ *array,* **int** *len,* **int** *from* **)**

MPI receive method on the compute processes. The method call MPI_Recv in the directional communicator associated with the process caller. (direction=1)

Parameters

message	: variable which will be assigned to the receive message.
len	: size of the array to be received.
from	: rank of the sender. (grid_rank_[1])

5.7.2.41 int Parallel2d::root ( ) `[inline]`

Returns

> root_ the rank of the process which is the root of the compute processes grid.

5.7.2.42 **template**<**class Type** > **void Parallel2d::send (Type &** *message,* **int** *to* **)**

MPI send method on the compute processes. The method calls MPI_Send in the lat_world_comm communicator.

Parameters

message	: variable to send.
to	: rank of the receiver. (in lat_world_comm)

5.7.2.43 **template**<**class Type** > **void Parallel2d::send (Type** ∗ *array,* **int** *len,* **int** *to* **)**

MPI send method on the compute processes. The method calls MPI_Send in the lat_world_comm communicator.

Parameters

array	: variable to send.
len	: size of the array.
to	: rank of the receiver. (in lat_world_comm)

5.7.2.44 **template**<**class Type** > **void Parallel2d::send_dim0 (Type &** *message,* **int** *to* **)**

MPI send method on the compute processes. The method calls MPI_Send in the directional communicator associated with the process caller. (direction=0)

Parameters

message	: variable to send.
to	: rank of the receiver. (grid_rank_[0])

5.7.2.45 **template**<**class Type** > **void Parallel2d::send_dim0 (Type** ∗ *array,* **int** *len,* **int** *to* **)**

MPI send method on the compute processes. The method calls MPI_Send in the directional communicator associated with the process caller. (direction=0)

Parameters

array	: variable to send.
len	: size of the array.
to	: rank of the receiver. (grid_rank_[0])

5.7.2.46 **template**<**class Type** > **void Parallel2d::send_dim1 (Type &** *message,* **int** *to* **)**

MPI send method on the compute processes. The method calls MPI_Send in the directional communicator associated with the process caller. (direction=1)

Parameters

message	: variable to send.
to	: rank of the receiver. (grid_rank_[1])

5.7.2.47 template<**class Type** > **void Parallel2d::send_dim1 (Type** ∗ *array,* **int** *len,* **int** *to* **)**

MPI send method on the compute processes. The method calls MPI_Send in the directional communicator associated with the process caller. (direction=1)

Parameters

array	: variable to send.
len	: size of the array.
to	: rank of the receiver. (grid_rank_[1])

5.7.2.48 template<**class Type** > **void Parallel2d::sendDown_dim0 (Type &** *bufferSend,* **Type &** *bufferRec,* **long** *len* **)**

Method to send a message through dim0 of the processes grid. Processes of grid_rank_[0]=N will send the message to the grid_rank_[0]=N-1, with a torus topology. Therefore each process will send and receive data.

Parameters

bufferSend	: pointer to the data which will be sent.
bufferRec	: pointer to the array where the receive data will be assigned.
len	: size of the array bufferSend.

5.7.2.49 template<**class Type** > **void Parallel2d::sendDown_dim1 (Type &** *bufferSend,* **Type &** *bufferRec,* **long** *len* **)**

Method to send a message through dim1 of the processes grid. Processes of grid_rank_[1]=N will send the message to the grid_rank_[1]=N-1, with a torus topology. Therefore each process will send and receive data.

Parameters

bufferSend	: pointer to the data which will be sent.
bufferRec	: pointer to the array where the receive data will be assigned.
len	: size of the array bufferSend.

5.7.2.50 template<**class Type** > **void Parallel2d::sendUp_dim0 (Type &** *bufferSend,* **Type &** *bufferRec,* **long** *len* **)**

Method to send a message through dim0 of the process grid. Processes of grid_rank_[0]=N will send the message to the grid_rank_[0]=N+1, with a torus topology. Therefore each process will send and receive data.

Parameters

bufferSend	: pointer to the data which will be sent.
bufferRec	: pointer to the array where the receive data will be assigned.
len	: size of the array bufferSend.

5.7.2.51 template<**class Type** > **void Parallel2d::sendUp_dim1 (Type &** *bufferSend,* **Type &** *bufferRec,* **long** *len* **)**

Method to send a message through dim1 of the processes grid. Processes of grid_rank_[1]=N will send the message to the grid_rank_[1]=N+1, with a torus topology. Therefore each process will send and receive data.

Parameters

bufferSend	: pointer to the data which will be sent.
bufferRec	: pointer to the array where the receive data will be assigned.
len	: size of the array bufferSend.

5.7.2.52   template$<$**class Type** $>$ **void Parallel2d::sendUpDown_dim0 (Type &** *bufferSendUp,* **Type &** *bufferRecUp,* **long** *lenUp,*
         Type & *bufferSendDown,* **Type &** *bufferRecDown,* **long** *lenDown* **)**

Method to send 2 message through dim0 of the processes grid. Processes of grid_rank_[0]=N will send the buffer-
SendUp to the grid_rank_[0]=N+1, and the bufferSendDown to the grid_rank_[0]=N-1, with a torus topology. There-
fore each process will send and receive 2 message.

Parameters

bufferSendUp	: pointer to the data which will be sent up.
bufferRecUp	: pointer to the array where the receive down data will be assigned.
lenUp	: size of the array bufferSendUp.
bufferSendDown	: pointer to the data which will be sent down.
bufferRecDown	: pointer to the array where the receive up data will be assigned.
lenDown	: size of the array bufferSendUp.

5.7.2.53   template$<$**class Type** $>$ **void Parallel2d::sendUpDown_dim1 (Type &** *bufferSendUp,* **Type &** *bufferRecUp,* **long** *lenUp,*
         Type & *bufferSendDown,* **Type &** *bufferRecDown,* **long** *lenDown* **)**

Method to send 2 message through dim1 of the processes grid. Processes of grid_rank_[1]=N will send the buffer-
SendUp to the grid_rank_[1]=N+1, and the bufferSendDown to the grid_rank_[1]=N-1, with a torus topology. There-
fore each process will send and receive 2 message.

Parameters

bufferSendUp	: pointer to the data which will be sent up.
bufferRecUp	: pointer to the array where the receive down data will be assigned.
lenUp	: size of the array bufferSendUp.
bufferSendDown	: pointer to the data which will be sent down.
bufferRecDown	: pointer to the array where the receive up data will be assigned.
lenDown	: size of the array bufferSendUp.

5.7.2.54   int Parallel2d::size ( ) `[inline]`

Returns

   lat_world_size_ the number of MPI process (compute processes)

5.7.2.55   template$<$**class Type** $>$ **void Parallel2d::sum (Type &** *number* **)**

Method to sum a number over all the compute processes. Each process will have the result assigned in the input
variable.

Parameters

number	: variable to sum.

Examples:

   poissonSolver.

5.7.2.56   template$<$**class Type** $>$ **void Parallel2d::sum (Type** $*$ *array,* **int** *len* **)**

Method to sum an array of number over all the compute processes. Each process will have the result assigned in
the input array.

Parameters

number	: pointer to the array to sum.
len	: size of the array.

5.7.2.57   template<**class Type** > **void Parallel2d::sum_dim0 (Type &** *number* **)**

Method to perform a sum of a number over all the compute processes with same grid_rank_[1]. Each process will have the result assigned in the input variable.

Parameters

number	: variable to sum.

5.7.2.58   template<**class Type** > **void Parallel2d::sum_dim0 (Type** ∗ *array,* **int** *len* **)**

Method to perform a sum of a number over all the compute processes with same grid_rank_[1]. Each process will have the result assigned in the input array.

Parameters

number	: pointer to the array to sum.
len	: size of the array.

5.7.2.59   template<**class Type** > **void Parallel2d::sum_dim1 (Type &** *number* **)**

Method to perform a sum of a number over all the compute processes with same grid_rank_[0]. Each process will have the result assigned in the input variable.

Parameters

number,:	variable to sum.

5.7.2.60   template<**class Type** > **void Parallel2d::sum_dim1 (Type** ∗ *array,* **int** *len* **)**

Method to perform a sum of a number over all the compute processes with same grid_rank_[0]. Each process will have the result assigned in the input array.

Parameters

number	: pointer to the array to sum.
len	: size of the array.

5.7.2.61   int Parallel2d::world_rank ( ) `[inline]`

Returns

world_rank_ rank of this process (in the world = compute + IOserver)

Examples:

IOserver.

5.7.2.62 int Parallel2d::world_size ( ) `[inline]`

Returns

world_size_ the number of MPI process (compute + [IOserver])

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

- [LATfield2_parallel2d.hpp]

5.8 PlanFFT< compType > Class Template Reference

Class which handle Fourier transforms of fields (real/complex, single/double precision) on cubic lattices. (currently implemented only for 3d)

```
#include <LATfield2_PlanFFT.hpp>
```

Public Member Functions

- [PlanFFT] ()

    Constructor.

- [PlanFFT] ([Field]< compType > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF_PL-ACE)
- void [initialize] ([Field]< compType > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF-_PLACE)
- [PlanFFT] ([Field]< double > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF_PLACE)
- void [initialize] ([Field]< double > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF_PL-ACE)
- **PlanFFT** ([Field]< compType > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF_PL-ACE)
- void **initialize** ([Field]< compType > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF-_PLACE)
- **PlanFFT** ([Field]< float > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF_PLACE)
- void **initialize** ([Field]< float > *rfield, [Field]< compType > *kfield, const int mem_type=FFT_OUT_OF_PLA-CE)
- void [execute] (int fft_type)

Private Member Functions

- void **transpose_0_2** (fftwf_complex *in, fftwf_complex *out, int dim_i, int dim_j, int dim_k)
- void **transpose_0_2_last_proc** (fftwf_complex *in, fftwf_complex *out, int dim_i, int dim_j, int dim_k)
- void **implement_local_0_last_proc** (fftwf_complex *in, fftwf_complex *out, int proc_dim_i, int proc_dim_j, int proc_dim_k, int proc_size)
- void **transpose_1_2** (fftwf_complex *in, fftwf_complex *out, int dim_i, int dim_j, int dim_k)
- void **transpose_back_0_3** (fftwf_complex *in, fftwf_complex *out, int r2c, int local_r2c, int local_size_j, int local_size_k, int proc_size, int halo, int components, int comp)
- void **implement_0** (fftwf_complex *in, fftwf_complex *out, int r2c_size, int local_size_j, int local_size_k, int halo, int components, int comp)
- void **b_arrange_data_0** (fftwf_complex *in, fftwf_complex *out, int dim_i, int dim_j, int dim_k, int khalo, int components, int comp)
- void **b_transpose_back_0_1** (fftwf_complex *in, fftwf_complex *out, int r2c, int local_r2c, int local_size_j, int local_size_k, int proc_size)
- void **b_implement_0** (fftwf_complex *in, fftwf_complex *out, int r2c_size, int local_size_j, int local_size_k)
- void [transpose_0_2] (fftw_complex *in, fftw_complex *out, int dim_i, int dim_j, int dim_k)

transpostion fonction

- void **transpose_0_2_last_proc** (fftw_complex ∗in, fftw_complex ∗out, int dim_i, int dim_j, int dim_k)
- void **implement_local_0_last_proc** (fftw_complex ∗in, fftw_complex ∗out, int proc_dim_i, int proc_dim_j, int proc_dim_k, int proc_size)
- void **transpose_1_2** (fftw_complex ∗in, fftw_complex ∗out, int dim_i, int dim_j, int dim_k)
- void **transpose_back_0_3** (fftw_complex ∗in, fftw_complex ∗out, int r2c, int local_r2c, int local_size_j, int local_size_k, int proc_size, int halo, int components, int comp)
- void **implement_0** (fftw_complex ∗in, fftw_complex ∗out, int r2c_size, int local_size_j, int local_size_k, int halo, int components, int comp)
- void **b_arrange_data_0** (fftw_complex ∗in, fftw_complex ∗out, int dim_i, int dim_j, int dim_k, int khalo, int components, int comp)
- void **b_transpose_back_0_1** (fftw_complex ∗in, fftw_complex ∗out, int r2c, int local_r2c, int local_size_j, int local_size_k, int proc_size)
- void **b_implement_0** (fftw_complex ∗in, fftw_complex ∗out, int r2c_size, int local_size_j, int local_size_k)

Private Attributes

- bool **status_**
- bool **type_**
- int **mem_type_**
- int **components_**
- int **rSize_** [3]
- int **kSize_** [3]
- int **rJump_** [3]
- int **kJump_** [3]
- int **rSizeLocal_** [3]
- int **kSizeLocal_** [3]
- int **r2cSize_**
- int **r2cSizeLocal_**
- int **r2cSizeLocal_as_**
- int **rHalo_**
- int **kHalo_**
- float ∗ **rData_**
- fftwf_complex ∗ **cData_**
- fftwf_complex ∗ **kData_**
- fftwf_complex ∗ **temp_**
- fftwf_complex ∗ **temp1_**
- fftwf_plan **fPlan_i_**
- fftwf_plan **fPlan_j_**
- fftwf_plan **fPlan_k_**
- fftwf_plan **fPlan_k_real_**
- fftwf_plan **bPlan_i_**
- fftwf_plan **bPlan_j_**
- fftwf_plan **bPlan_j_real_**
- fftwf_plan **bPlan_k_**
- double ∗ **rData_**
- fftw_complex ∗ **cData_**
- fftw_complex ∗ **kData_**
- fftw_complex ∗ **temp_**
- fftw_complex ∗ **temp1_**
- fftw_plan **fPlan_i_**
- fftw_plan **fPlan_j_**
- fftw_plan **fPlan_k_**
- fftw_plan **fPlan_k_real_**

- fftw_plan **bPlan_i_**
- fftw_plan **bPlan_j_**
- fftw_plan **bPlan_j_real_**
- fftw_plan **bPlan_k_**

Static Private Attributes

- static bool **R2C**
- static bool **C2C**
- static bool **initialized**

5.8.1 Detailed Description

template<class compType>class PlanFFT< compType >

Class which handle Fourier transforms of fields (real/complex, single/double precision) on cubic lattices. (currently implemented only for 3d)

This class allow to perform Fourier transform of real and complex fields. See the FFTs example to have have a short intro of the usage. See the PoissonSolver example for more advanced usage (as linking several field to the same Fourier image)

One should understand that first a plan is created then executed (in the FFTW fashion). The plan links two fields, one on Fourier space, one on real space. Boths field will be allocated by the planer when the plan is initilized. But need to be initialized before passing them to the planer.

One need to be carefull to corretly define the lattice and field.

See Also

    void Lattice::initializeRealFFT(Lattice & lat_real, int halo);
    void Lattice::initializeComplexFFT(Lattice & lat_real, int halo);
    cKSite class
    rKSite class

Examples:

    FFTs, and poissonSolver.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 template<class compType > PlanFFT< compType >::PlanFFT (Field< compType > * *rfield,* Field< compType > * *kfield,* const int *mem_type =* FFT_OUT_OF_PLACE)

Constructor with initialization for complex to complex transform.

Parameters

rfield	: real space field
kfield	: Fourier space field
mem_type	: memory type (FFT_OUT_OF_PLACE or FFT_IN_PLACE). In place mean that both Fourier and real space field point to the same data array.

See Also

    initialize(Field<compType>* rfield,Field<compType>* kfield,const int mem_type = FFT_OUT_OF_PLACE);

5.8.2.2 template<**class compType** > **PlanFFT**< **compType** >**::PlanFFT (Field**< **double** > ∗ *rfield,* **Field**< **compType** > ∗ *kfield,* **const int** *mem_type =* FFT_OUT_OF_PLACE **)**

Constructor with initialization for real to complex transform.

Parameters

rfield	: real space field
kfield	: Fourier space field
mem_type	: memory type (FFT_OUT_OF_PLACE or FFT_IN_PLACE). In place mean that both Fourier and real space fields point to the same data array.

See Also

initialize(Field<compType>∗ rfield,Field<compType>∗ kfield,const int mem_type = FFT_OUT_OF_PLACE);

5.8.3 Member Function Documentation

5.8.3.1 template<**class compType** > **void PlanFFT**< **compType** >**::execute (int** *fft_type* **)**

Execute the Fourier transform.

Parameters

fft_type	: dirrection of the transform. Can be FFT_BACKWARD or FFT_FORWARD.

5.8.3.2 template<**class compType** > **void PlanFFT**< **compType** >**::initialize (Field**< **compType** > ∗ *rfield,* **Field**< **compType** > ∗ *kfield,* **const int** *mem_type =* FFT_OUT_OF_PLACE **)**

Initialization for complex to complex transform.

Parameters

rfield	: real space field
kfield	: Fourier space field
mem_type	: memory type (FFT_OUT_OF_PLACE or FFT_IN_PLACE). In place mean that both Fourier and real space field point to the same data array.

5.8.3.3 template<**class compType** > **void PlanFFT**< **compType** >**::initialize (Field**< **double** > ∗ *rfield,* **Field**< **compType** > ∗ *kfield,* **const int** *mem_type =* FFT_OUT_OF_PLACE **)**

Initialization for real to complex transform.

Parameters

rfield	: real space field
kfield	: Fourier space field
mem_type	: memory type (FFT_OUT_OF_PLACE or FFT_IN_PLACE). In place mean that both Fourier and real space fields point to the same data array.

5.8.3.4 template<**class compType** > **void PlanFFT**< **compType** >**::transpose_0_2 (fftw_complex** ∗ *in,* **fftw_complex** ∗ *out,* **int** *dim_i,* **int** *dim_j,* **int** *dim_k* **)** `[private]`

transpostion fonction

forward real to complex

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

- LATfield2_PlanFFT.hpp

5.9 rKSite Class Reference

A child of Site, built to work with the Fourier space lattices for real to complex transforms.

```
#include <LATfield2_Site.hpp>
```

Inheritance diagram for rKSite:

```
        Site
         ▲
         |
       rKSite
```
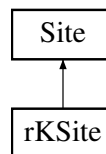
Public Member Functions

- **rKSite** (Lattice &lattice)
- **rKSite** (Lattice &lattice, long index)
- void **initialize** (Lattice &lattice)
- void **initialize** (Lattice &lattice, long index)
- rKSite **operator+** (int asked_direction)
- rKSite **operator-** (int asked_direction)
- int **coordLocal** (int asked_direction)
- int **coord** (int asked_direction)
- int **latCoord** (int direction)
- int **latCoordLocal** (int direction)
- bool **setCoord** (int ∗r_asked)
- bool **setCoord** (int x, int y, int z)

Private Attributes

- int **directions_** [3]

Additional Inherited Members

5.9.1 Detailed Description

A child of Site, built to work with the Fourier space lattices for real to complex transforms.

A class which simplifies the map of the field data array index. This class allow to get coordinate on the lattice, loop over each site of the lattice and access neighboring lattices sites

WARNING: the rKSite class must be used only on lattices initialized using initializeRealFFT() method of the Lattice class.

This class has same binding as the Site class, please refer to the Site class for the documentation.

Examples:

FFTs, and poissonSolver.

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

- LATfield2_Site.hpp

5.10 SettingsFile Class Reference

A utility class designed to make reading in runtime parameter values easier.

```
#include <LATfield2_SettingsFile.hpp>
```

Public Member Functions

- SettingsFile ()

    Constructor.
- SettingsFile (const std::string filename, const int mode, const int argc=0, char ∗∗argv=NULL)
- ∼SettingsFile ()

    desctructor
- void open (const std::string filename, const int mode, const int argc=0, char ∗∗argv=NULL)
- void close ()
- void create (const std::string filename)
- template<class TemplateClass >
    void read (const std::string parameter_name, TemplateClass &parameter)
- template<class TemplateClass >
    void add (const std::string parameter_name, const TemplateClass &parameter)
- template<class TemplateClass >
    void write (const std::string parameter_name, const TemplateClass &parameter)

Static Public Attributes

- static int **noCreate**
- static int **autoCreate**

Private Member Functions

- bool **search** (const std::string search_string)

Private Attributes

- std::string **filename_**
- std::fstream **file_**
- std::stringstream **stream_**
- int **mode_**
- bool **isRoot_**

5.10.1 Detailed Description

A utility class designed to make reading in runtime parameter values easier.

If the command-line arguments are input via optional inputs on either the constructor or open member function, then these take precedent: they are effectively last in the file.

Note, when used with std::string objects, only one word is allowed per setting, ie. spaces are not allowed. This is because of the way that the $>>$ operator works for this class. This fits nicely with the command-line override, however.

Note that the string specified followed by = is searched for in the file and then the input read. If one setting name is also the end of another that precedes it in the file then the wrong one will be read.

Only the primary MPI process is able to create or add to the setting file. Further processes will be sent the file contents via MPI. To use this class in serial code the preprocessor defintion SERIAL must be set. This flag have not been remove to allow users to use it outside LATfield2. LATfield2 have no serial version, therefor setting prepocessor flag -DSERIAL should never be used with LATfield2.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 SettingsFile::SettingsFile (const std::string *filename,* const int *mode,* const int *argc =* 0 *,* char $**$ *argv =* NULL)

Constructor + open a file

Parameters

filename	: path to the file.
mode	: noCreate (the read method will exit if the parameter does not exist) or autoCreate (read will add the missing parameter).
argc	: additionnal argument number.
argv	: pointer to the additionnal arguments.

5.10.3 Member Function Documentation

5.10.3.1 template<class TemplateClass > void SettingsFile::add (const std::string *parameter_name,* const TemplateClass & *parameter*)

Method to add a parameter to the settings file. The new parameter will be just added to the end of the file, even if it already exists.

Parameters

parameter_-name,:	string containing the name of the parameter.
parameter,:	pointer to the value of the parameter.

5.10.3.2 void SettingsFile::close ( )

Close the current settings file

5.10.3.3 void SettingsFile::create (const std::string *filename*)

Create a new settings file and open it.

Parameters

filename,:	path to the file.

5.10.3.4 void SettingsFile::open (const std::string *filename,* const int *mode,* const int *argc =* 0 *,* char $**$ *argv =* NULL)

Open an existinge settings file

Parameters

filename	: path to the file
mode	: noCreate (the read method will exit if the parameter does not exist) or autoCreate (read will add the missing parameter).
argc	: additional argument number.
argv	: pointer to the additional arguments.

5.10.3.5 **template**<**class TemplateClass** > **void SettingsFile::read (const std::string** *parameter_name,* **TemplateClass &** *parameter* **)**

Method to read a parameter.

Parameters

parameter_- *name*	: string containing the name of the parameter. If the parameter does not exite and the mode autocreate is set, this method will add the parameter to the settings file with the current value of "parameter". In the case the mode is set to nocreate, then read will exit for security, for this reason in it is always advise to set the mode to nocreate for production runs.
parameter	: pointer to the variable where the parameter will be assigned.

5.10.3.6 **template**<**class TemplateClass** > **void SettingsFile::write (const std::string** *parameter_name,* **const TemplateClass &** *parameter* **)**

Method to write a parameter in the settings file. If the parameter_name exist, it will overwrite the parameter. And if it does not exist in the file, it will be added at the end of the file.

Parameters

parameter_- *name,:*	string containing the name of the parameter
parameter,:	pointer to the value of the parameter.

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

- LATfield2_SettingsFile.hpp

5.11 Site Class Reference

A class for referencing values of an instance of the field class at a given point on a lattice.

```
#include <LATfield2_Site.hpp>
```

Inheritance diagram for Site:
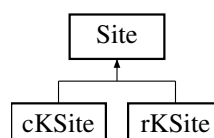
Public Member Functions

- Site ()

Constructor.

- Site (Lattice &lattice)
- Site (Lattice &lattice, long index)
- void initialize (Lattice &lattice)
- void initialize (Lattice &lattice, long index)
- void first ()
- bool test ()
- void next ()
- void haloFirst ()
- bool haloTest ()
- void haloNext ()
- Site operator+ (int direction)
- Site operator- (int direction)
- void indexAdvance (long number)
- long index () const
- void setIndex (long new_index)
- int coord (int direction)
- int coordLocal (int direction)
- bool setCoord (int *r)
- bool setCoord (int x, int y, int z)
- Lattice & lattice ()

Protected Attributes

- Lattice * **lattice_**
- long **index_**

5.11.1 Detailed Description

A class for referencing values of an instance of the field class at a given point on a lattice.

A class which simplify the map of the field data array index. This class allow to get coordinate on the lattice, loop over each site of the lattice and perform displacment on the lattice.

The site class encapsulates the mapping between the coordinate on the lattice and the index of the Field::data_ array which store the value of an instance of the Field class. It also contain method to loop over each site of the lattice, and to perform spacial displacement on the lattice.

Examples:

FFTs, gettingStarted, and poissonSolver.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Site::Site (Lattice & *lattice*)

Constructor with initialization.

Parameters

lattice	: the lattice on which the Site is defined.

See Also

initialize(Lattice& lattice)

5.11.2.2 Site::Site (Lattice & *lattice,* **long** *index* **)**

Constructor with initialization.

Parameters

lattice	: the lattice on which the Site is defined.
index	: set the current index of the field.

See Also

> initialize(Lattice& lattice, long index)

5.11.3 Member Function Documentation

5.11.3.1 int Site::coord (int *direction* **)**

Method which return the site coordinate of a give dimension

Parameters

direction	: label of the coordinate.

Returns

> site coordinate of the "direction" dimension

Examples:

> gettingStarted, and poissonSolver.

5.11.3.2 int Site::coordLocal (int *direction* **)**

Method which return the local site coordinate of a give dimension

Parameters

direction	: label of the coordinate.

Returns

> site local coordinate of the "direction" dimension

5.11.3.3 void Site::first ( )

Method to set the Site to the first site which is not within the halo. This method is used for loopping over the all lattice sites:

for(site.first();site.test();site.next());

See Also

> test()
> next()

Examples:

> gettingStarted, and poissonSolver.

5.11.3.4   void Site::haloFirst (   )

Method to set the Site to the first site which is within the halo. This method is used for loopping over the all halo sites:

for(site.haloFirst();site.haloTest();site.haloNext());

See Also

> haloTest()
> haloNext()

5.11.3.5   void Site::haloNext (   )

Method to jump to the next index which is in the halo. This method is used for loopping over the all halo sites:

for(site.haloFirst();site.haloTest();site.haloNext());

See Also

> haloFirst()
> haloTest()

5.11.3.6   bool Site::haloTest (   )

Method to test if the Site have a smaller or equal index than the last index within the halo. This method is used for loopping over the all halo sites:

for(site.haloFirst();site.haloTest();site.haloNext());

See Also

> haloFirst()
> haloNext()

5.11.3.7   long Site::index (   ) const

Returns

> this method return the current index pointed by the site.

5.11.3.8   void Site::indexAdvance (long *number*)

Method which add "number" to the current index.

5.11.3.9   void Site::initialize (Lattice & *lattice*)

Initialization.

Parameters

lattice	: the lattice on which the Site is defined.

5.11.3.10   void Site::initialize (Lattice & *lattice,* **long** *index* **)**

Constructor with initialization.

Parameters

lattice	: the lattice on which the Site is defined.
index	: set the current index of the field.

5.11.3.11   Lattice & Site::lattice (   )

Returns

Returns the pointer to the lattice on which the site is defined.

5.11.3.12   void Site::next (   )

Method to jump to the next index which is not in the halo. This method is used for loopping over the all lattice sites:

for(site.first();site.test();site.next());

See Also

first()
test()

Examples:

gettingStarted, and poissonSolver.

5.11.3.13   Site Site::operator+ (**int** *direction* **)**

Overloaded operator + The + operator is used to make a displacement of +1 site the the asked direction.

Parameters

direction	: direction of the displacement

5.11.3.14   Site Site::operator- (**int** *direction* **)**

Overloaded operator - The - operator is used to make a displacement of -1 site the the asked direction.

Parameters

direction	: direction of the displacement

5.11.3.15   bool Site::setCoord (**int** $*$ *r* **)**

Method to set the site to a given coordinate.

Parameters

r	: array which contain the coordinate. The array size must be equal to the number of dimension of the lattice

Returns

    True: if the coordinate is local. False: if the local part of the lattice does not have this coordinate.

5.11.3.16   bool Site::setCoord (int *x,* int *y* = 0 *,* int *z* = 0)

Method to set the site to a given coordinate for 3d lattices.

Parameters

x	: coordinate of the 0 dimension.
y	: coordinate of the 1 dimension.
z	: coordinate of the 2 dimension.

Returns

    True: if the coordinate is local. False: if the local part of the lattice does not have this coordinate.

5.11.3.17   void Site::setIndex (long *new_index*)

Method to set the current index of the site.

Parameters

new_index,:	the site index is set to new_index.

5.11.3.18   bool Site::test ( )

Method to test if the Site have a smaller or equal index than the last index not within the halo. This method is used for loopping over the all lattice sites:

for(site.first();site.test();site.next());

See Also

    first()
    next()

Examples:

    gettingStarted, and poissonSolver.

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

- LATfield2_Site.hpp

5.12   temporaryMemFFT Class Reference

A class wich handle the additional memory needed by the class PlanFFT; Should never be used, internal temporary memroy handler, if real need, can be hacked, but could comflict with the PlanFFT!

```
#include <LATfield2_PlanFFT.hpp>
```

Public Member Functions

- **temporaryMemFFT** (long size)
- int **setTemp** (long size)
- fftwf_complex ∗ **temp1** ()
- fftwf_complex ∗ **temp2** ()
- fftw_complex ∗ **temp1** ()
- fftw_complex ∗ **temp2** ()

Private Attributes

- fftwf_complex ∗ **temp1_**
- fftwf_complex ∗ **temp2_**
- fftw_complex ∗ **temp1_**
- fftw_complex ∗ **temp2_**
- long **allocated_**

5.12.1   Detailed Description

A class wich handle the additional memory needed by the class PlanFFT; Should never be used, internal temporary memroy handler, if real need, can be hacked, but could comflict with the PlanFFT!

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

- LATfield2_PlanFFT.hpp

Chapter 6

File Documentation

6.1 Imag.hpp File Reference

Imag.hpp Contains the Imag class definition.

Classes

- class Imag

  A utility class for complex arithmetic, invested from LATfield 1.0.

Typedefs

- typedef float Real

  real numbers

Functions

- Imag **expi** (Real x)

6.1.1 Detailed Description

Imag.hpp Contains the Imag class definition.

Author

Neil Bevis

6.2 int2string.hpp File Reference

int2string a small function to convert integer to string

Functions

- string int2string (int number, int max=999, bool zeropad=true)

  integer to string method

6.2.1 Detailed Description

int2string a small function to convert integer to string

Author

Neil Bevis

6.2.2 Function Documentation

6.2.2.1 string int2string (int *number,* int *max =* 999*,* bool *zeropad =* true)

integer to string method

Parameters

number	: int, input integer
max	: int, max number (to specify number of digit in the string.
zeropad	: bool, true will pad zeros, fals will not. default is true.

Returns

string.

Examples:

IOserver.

6.3 LATfield2.hpp File Reference

LATfield2 header.

Variables

- IOserver **IO_Server**
- Parallel2d **parallel**

6.3.1 Detailed Description

LATfield2 header.

Author

David Daverio,Neil Bevis

6.4 LATfield2_Field.hpp File Reference

LATfield2_Field.hpp contain the class Field definition.

Classes

- class Field< FieldType >

    The Field class represent a field on a given lattice.

Functions

- template< class FieldType >
  void **defaultFieldSave** (fstream &file, FieldType *siteData, int components)
- template< class FieldType >
  void **defaultFieldLoad** (fstream &file, FieldType *siteData, int components)

Variables

- int **symmetric**
- int **unsymmetric**

6.4.1 Detailed Description

LATfield2_Field.hpp contain the class Field definition.

Author

David Daverio, Neil Bevis

6.5 LATfield2_IO_server.hpp File Reference

LATfield2_IO_server.hpp contains the class IOserver definition.

Classes

- struct file_struct

  A structure to describe a file for the I/O server (dedicated MPI processes for writing to disks)
- class IOserver

  A class to handle the I/O using MPI process reserved for IO purpose on which the files are defined.

Typedefs

- typedef int **ioserver_file**

6.5.1 Detailed Description

LATfield2_IO_server.hpp contains the class IOserver definition.

Author

David Daverio

6.6 LATfield2_Lattice.hpp File Reference

LATfield2_Lattice.hpp contains the class Lattice definition.

Classes

- class Lattice

  The Lattice class describe a cartesian mesh (with 2 or more dimensions). The updateHalo method of the Field class generate the periodicity.

6.6.1 Detailed Description

LATfield2_Lattice.hpp contains the class Lattice definition.

Author

David Daverio, Neil Bevis

6.7 LATfield2_parallel2d.hpp File Reference

LATfield2_parallel2d.hpp contains the class Parallel2d definition.

Classes

- class Parallel2d

    LATfield2d underliying class for paralleization.

Functions

- **if** (world_rank_==0)
- **MPI_Comm_group** (world_comm_,&world_group_)
- **MPI_Group_range_incl** (world_group_, 1,&rang,&lat_world_group_)
- **MPI_Comm_create** (world_comm_, lat_world_group_,&lat_world_comm_)
- **MPI_Group_rank** (lat_world_group_,&comm_rank)
- **if** (comm_rank!=MPI_UNDEFINED)
- **if** (grid_rank_[0]==grid_size_[0]-1) last_proc_[0]
- **if** (grid_rank_[1]==grid_size_[1]-1) last_proc_[1]
- IO_Server **initialize** (proc_size0, proc_size1, IO_total_size, IO_node_size)
- **if** (lat_world_rank_==0)
- **MPI_Comm_group** (lat_world_comm_,&lat_world_group_)
- **for** (j=0;j< grid_size_[1];j++)
- **if** (root_==lat_world_rank_) isRoot_

Variables

- void Parallel2d::initialize(int proc_size0, int proc_size1, int IO_total_size, int IO_node_size) void Parallel2d **grid_size_** [1]
- **dim0_comm_**
- **dim1_comm_**
- **dim0_group_**
- **dim1_group_**
- int **rang** [3]
- int **i**
- int **j**
- int **comm_rank**
- **else**
- **root_**
- **grid_rank_** [1]
- **isIO_**
- else **last_proc_** [0]
- else **isRoot_**

6.7.1   Detailed Description

LATfield2_parallel2d.hpp contains the class Parallel2d definition.

Author

David Daverio

6.8   LATfield2_PlanFFT.hpp File Reference

LATfield2_PlanFFT.hpp contains the class PlanFFT definition.

Classes

- class temporaryMemFFT

    A class wich handle the additional memory needed by the class PlanFFT; Should never be used, internal temporary memroy handler, if real need, can be hacked, but could comflict with the PlanFFT!

- class PlanFFT< compType >

    Class which handle Fourier transforms of fields (real/complex, single/double precision) on cubic lattices. (currently implemented only for 3d)

Variables

- const int **FFT_FORWARD**
- const int **FFT_BACKWARD**
- const int **FFT_IN_PLACE**
- const int **FFT_OUT_OF_PLACE**
- temporaryMemFFT **tempMemory**

6.8.1   Detailed Description

LATfield2_PlanFFT.hpp contains the class PlanFFT definition.

Author

David Daverio

6.9   LATfield2_save_hdf5.h File Reference

LATfield2_save_hdf5.h contains the definition of the function used for hdf5 i/o.

Functions

- int **save_hdf5_externC** (char ∗data, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, int comp, hid_t array_type, int array_size, string filename_str)
- int **load_hdf5_externC** (char ∗data, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, string filename_str)
- template<class fieldType >
    int **save_hdf5** (fieldType ∗data, hid_t type_id, int array_size, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, int comp, string filename_str)
- template<class fieldType >
    int **load_hdf5** (fieldType ∗data, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, int comp, string filename_str)

6.9.1 Detailed Description

LATfield2_save_hdf5.h contains the definition of the function used for hdf5 i/o.

Author

David Daverio

6.10 LATfield2_save_hdf5_pixie.h File Reference

LATfield2_save_hdf5_pixie.h contains the definition of the function used for hdf5 i/o compatible to pixie reader.

Functions

- int **save_hdf5_externC** (char ∗data, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, int comp, hid_t array_type, int array_size, string filename_str)
- int **load_hdf5_externC** (char ∗data, long file_offset[2], int ∗size, int ∗sizeLocal, int comp, int halo, int lat_dim, string filename_str)
- template<class fieldType >
  int **save_hdf5** (fieldType ∗data, hid_t type_id, int array_size, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, int comp, string filename_str)
- template<class fieldType >
  int **load_hdf5** (fieldType ∗data, long file_offset[2], int ∗size, int ∗sizeLocal, int halo, int lat_dim, int comp, string filename_str)

6.10.1 Detailed Description

LATfield2_save_hdf5_pixie.h contains the definition of the function used for hdf5 i/o compatible to pixie reader.

Author

David Daverio

6.11 LATfield2_SettingsFile.hpp File Reference

LATfield2_SettingsFile.hpp contain the class SettingsFile definition.

Classes

- class SettingsFile
  A utility class designed to make reading in runtime parameter values easier.

6.11.1 Detailed Description

LATfield2_SettingsFile.hpp contain the class SettingsFile definition.

Author

N. Bevis

6.12   LATfield2_Site.hpp File Reference

LATfield2_Site.hpp contains the Site, rKSite, and cKSite definition.

Classes

- class Site

    A class for referencing values of an instance of the field class at a given point on a lattice.
- class cKSite

    A child of Site, built to work with the Fourier space lattices for complex to complex transforms.
- class rKSite

    A child of Site, built to work with the Fourier space lattices for real to complex transforms.

6.12.1   Detailed Description

LATfield2_Site.hpp contains the Site, rKSite, and cKSite definition.

Author

David Daveio, Neil Bevis

Chapter 7

Example Documentation

7.1 FFTs

This is a simple example of the usage of the FFT wrapper of LATfield2. The wrapper uses the FFTW library to perform 1d serial FFTs. And so FFTW 2 (or higher) need to be installed on the system.

```cpp
#include <iostream>
#include "LATfield2.hpp"

using namespace LATfield2;


int main(int argc, char **argv)
{
    int n,m;
    int BoxSize = 64;
    int halo = 1;
    int khalo =0;
    int dim = 3;
    int comp = 1;
    double sigma2=0.5;
    double res =0.5;


    for (int i=1 ; i < argc ; i++ ){
        if ( argv[i][0] != '-' )
            continue;
        switch(argv[i][1]) {
            case 'n':
                n = atoi(argv[++i]);
                break;
            case 'm':
                m =  atoi(argv[++i]);
                break;
            case 'b':
                BoxSize = atoi(argv[++i]);
                break;
        }
    }

    parallel.initialize(n,m);


    Lattice lat(dim,BoxSize,halo);


    //Real to complex fourier transform

    Lattice latKreal;
    latKreal.initializeRealFFT(lat, khalo);

    Site x(lat);
    rKSite kReal(latKreal);

    Field<Real> phi;
    phi.initialize(lat,comp);

    Field<Imag> phiK;
    phiK.initialize(latK,comp);
```

```
    PlanFFT<Imag> planReal(&phi,&phiK);

    planReal.execute(FFT_FORWARD);

    planReal.execute(FFT_BACKWARD);

    //complex to complex fourier transform

    Lattice latKcomplex;
    latKcomplex.initializeComplexFFT(lat, khalo);

    Site x(lat);
    cKSite kComplex(latKreal);

    Field<Imag> rho;
    rho.initialize(lat,comp);

    Field<Imag> rhoK;
    rhoK.initialize(latK,comp);

    PlanFFT<Imag> planComplex(&rho,&rhoK);

    planComplex.execute(FFT_FORWARD);

    planComplex.execute(FFT_BACKWARD);


}
```

7.1.1 Compile and Run

Go to the LATfield2/examples folder. and compile this example with e.g. mpic++:

Double precision:

```
mpic++ -o fft_exec fft.cpp -I../ -DFFT3D -lfftw
```

Single precision:

```
mpic++ -o fft_exec fft.cpp -I../ -DFFT3D -DSINGLE -lfftwf
```

It can be executed using (here using "mpirun -np 4" to run with 4 processes):

```
mpirun -np 4 ./fft_exec -n 2 -m 2
```

The executable will not return anything, and the FFT are perform on fields which have not been assigned. It exist just to show the usage of the FFT wrapper of LATfield2.

7.1.2 Going through the code

7.1.2.1 Declaration of the lattice in real space.

The lattice in real space is declared as usual.

```
    Lattice lat(dim,BoxSize,halo);
```

7.1.2.2 Real to Complex (r2c) Fourier transform

First thing to do is to declare the lattice in Fourier space. For a r2c FFT it is done using the initializeRealFFT method of the Lattice class.

```
    Lattice latKreal;
    latKreal.initializeRealFFT(lat, khalo);
```

At that point, one should be aware that the lattice in Fourier space does not have the same data distribution in a single process and not the same distribution in the cluster. Indeed the local part of the lattice which is stored in a process is not the same, and the local part have not anymore the same data distribution in the memory. Therefor one should use different site objet for both lattice. For r2c transform, the Fourier space site is the rKSite class.

```
Site x(lat);
rKSite kReal(latKreal);
```

Once both lattice (real and Fourier one) are initialized, one can declare the fields. The PlanFFT class will allocate the needed memory both field and some temporary memory. Therefor the field need to be initialized but not allocated.

For the real field one should use the Real datatype (which is a redefinition of float or double, depending the precision requirement)

```
Field<Real> phi;
phi.initialize(lat,comp);
```

For the Fourier field one should use the Imag datatype (which is a redefinition of fftwf_complex or fftw_complex, depending the precision requirement)

```
Field<Imag> phiK;
phiK.initialize(latK,comp);
```

Once both field are initialized, the planner can be declared and initialized:

```
PlanFFT<Imag> planReal(&phi,&phiK);
```

One should notice that the reference of the field has to be passed as the initialization of the planner will allocate the fields.

Then you can perform the Fourier transform using the execute method of the PlanFFT class.

```
planReal.execute(FFT_FORWARD);
planReal.execute(FFT_BACKWARD);
```

7.1.2.3 Complex to Complex (c2c) Fourier transform

First thing to do is to declare the lattice in Fourier space. For a c2c FFT it is done using the initializeComplexFFT method of the Lattice class.

```
Lattice latKcomplex;
latKcomplex.initializeComplexFFT(lat, khalo);
```

At that point, one should be aware that the lattice in Fourier space does not have the same data distribution in a single process and not the same distribution in the cluster. Indeed the local part of the lattice which is stored in a process is not the same, and the local part have not anymore the same data distribution in the memory. Therefor one should use different site objet for both lattice. For c2c transform, the Fourier space site is the cKSite class.

```
Site x(lat);
cKSite kComplex(latKreal);
```

Once both lattice (real and Fourier one) are initialized, one can declare the fields. The PlanFFT class will allocate the needed memory both field and some temporary memory. Therefor the field need to be initialized but not allocated.

For both the real and the fourier fields one should use the Imag datatype (which is a redefinition of fftwf_complex or fftw_complex, depending the precision requirement)

```
    Field<Imag> rho;
    rho.initialize(lat,comp);


    Field<Imag> rhoK;
    rhoK.initialize(latK,comp);
```

Once both fields are initialized, the planner can be declared and initialized:

```
    PlanFFT<Imag> planComplex(&rho,&rhoK);
```

One should notice that the reference of the fields has to be passed as the initialization of the planner will allocate the field.

Then you can perform the fourier transform using the execute method of the PlanFFT class.

```
    planComplex.execute(FFT_FORWARD);
    planComplex.execute(FFT_BACKWARD);
```

7.1.2.4 important points

One should notice that the wrapper is based on the FFTW library, more precisely the 1d serial transform of FFTW. Therefore to know exactly what is computed the best is to refer to the FFTW documentation and look at the 1d version of FFTW.

http://www.fftw.org/doc/What-FFTW-Really-Computes.html

Secondly, one should notice that FFTW forward methods will return a result which have been multiply by the size of the array. Therefor, to have the correct result once should divide by Lattice.sites() the result given by the wrapper. This is not done within the wrapper to return the same result as FFTW.

7.2 gettingStarted

A very simple example which aims to describe the main features of LATfield2. This example is composed by 5 blocks of code. First the parallel object is initialized, and this must be the first operation within a code which use LATfield2 as it initializes MPI. Secondly a Lattice object is declared. Thirdly Field objects are declared on the Lattice. Fourth the some operations are performed on the fields, then finally the field are written on disk.

```
#include "LATfield2.hpp"
using namespace LATfield2;


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

    //-------- Initilization of the parallel object ---------
    int n,m;

    for (int i=1 ; i < argc ; i++ ){
        if ( argv[i][0] != '-' )
            continue;
        switch(argv[i][1]) {
            case 'n':
                n = atoi(argv[++i]);
                break;
            case 'm':
                m =  atoi(argv[++i]);
                break;
        }
    }

    parallel.initialize(n,m);
```

```
    COUT << "Parallel grid size: ("<<parallel.grid_size()[0]<<","<<parallel.
      grid_size()[1]<<"). "<<endl;
    //----------------------   end   -----------------------


    //------------   Declaration of a Lattice   --------------
    int dim = 3;
    int latSize[dim] = {25,57,32};
    int halo = 1;
    Lattice lat(dim,latSize,halo);

    COUT << "Lattice size: ("<< lat.size(0)<<","<< lat.size(1)<<","<< lat.
      size(2)<<");"<<endl;
    cout << "Process ranks: "<< parallel.rank()<<",("<< parallel.grid_rank()[0]<<","<<parallel
      .grid_rank()[1]<< "); ";
    cout << "Local lattice size: ("<< lat.sizeLocal(0)<<","<< lat.
      sizeLocal(1)<<","<< lat.sizeLocal(2)<<"); ";
    cout << "Coordinate of the first local point: (0,"<< lat.coordSkip()[1] <<","<< lat.
      coordSkip()[0] <<")."<<endl;
    //----------------------   end   -----------------------


    //-----------   Declaration of the Fields   --------------
    Field<Real> rho(lat);
    Field<Real> gradPhi(lat,3);

    Field<Real> phi;
    phi.initialize(lat);
    phi.alloc();
    //----------------------   end   -----------------------

    //-------------   Operations on Fields   ----------------
    Site x(lat);

    double x2;
    for(x.first();x.test();x.next())
    {
        x2  = pow(0.5 + x.coord(0) - lat.size(0)/2.,2.);
        x2 += pow(0.5 + x.coord(1) - lat.size(1)/2.,2.);
        x2 += pow(0.5 + x.coord(2) - lat.size(2)/2.,2.);
        phi(x) = exp(-x2 * 2.);
    }

    phi.updateHalo();

    for(x.first();x.test();x.next())
    {

        gradPhi(x,0) = (phi(x+0)-phi(x-0));
        gradPhi(x,1) = (phi(x+1)-phi(x-1));
        gradPhi(x,2) = (phi(x+2)-phi(x-2));

        rho(x)=0;
        for(int i=0;i<3;i++)rho(x) += phi(x+i) - 2 * phi(x) + phi(x-i);

    }
    //----------------------   end   -----------------------

    //-----------------   writing fields   ------------------
    string str_filename = "./test_phi";
#ifdef HDF5
    str_filename += ".h5";
    phi.saveHDF5(str_filename);
    str_filename = "test_phi_slice.h5";
    phi.saveSliceHDF5(str_filename,0,3);
#else
    str_filename += ".txt";
    phi.write(str_filename);
#endif


    //-------------------------------------------------------
}
```

7.2.1 Compile and Run

Go to the LATfield2/examples folder. and compile this example with e.g. mpic++:

```
mpic++ -o getStart gettingStarted.cpp -I../ -DHDF5 -lhdf5
```

This will compile using HDF5 for the Field I/O. If you have not HDF5 installed then you should compile with:

```
mpic++ -o getStart gettingStarted.cpp -I../
```

It can be executed using (here using "mpirun -np 4" to run with 4 process):

```
mpirun -np 4 ./getStart -n 2 -m 2
```

The executable will prompt the following text:

```
Parallel grid size: (2,2).
Lattice size: (25,57,32);
Process ranks: 0,(0,0); Local lattice size: (25,28,16); First local point coordinate: (0,0,0).
Process ranks: 1,(1,0); Local lattice size: (25,28,16); First local point coordinate: (0,0,16).
Process ranks: 2,(0,1); Local lattice size: (25,29,16); First local point coordinate: (0,28,0).
Process ranks: 3,(1,1); Local lattice size: (25,29,16); First local point coordinate: (0,28,16).
```

The first line gives the size of the 2 dimension of the parallel grid. The second line gives the size of a Lattice. Then each process output its ranks and the description of the local part of the lattice (the part of the lattice which is stored on the given process).

7.2.2 Going through the code

7.2.2.1 Parallel initialization

The first operation to perform within any code which use LATfield2 is to initialize the parallel object by giving the size of the 2 dimensions of the parallel grid. This numbers depend on the number of process used to run, therefor it is advised to read the numbers from the executable arguments. Here this 2 integer are given using "-n XX -m YY":

```cpp
#include "LATfield2.hpp"
using namespace LATfield2;


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

    //-------- Initilization of the parallel object ---------
    int n,m;

    for (int i=1 ; i < argc ; i++ ){
        if ( argv[i][0] != '-' )
            continue;
        switch(argv[i][1]) {
            case 'n':
                n = atoi(argv[++i]);
                break;
            case 'm':
                m =  atoi(argv[++i]);
                break;
        }
    }
```

Then the parallel object is initialized passing the two integers n and m:

```cpp
    parallel.initialize(n,m);
```

This will initialize the parallel grid by calling MPI_initialize and will also construct all MPI communicators needed by the library.

After this, one can output the size of the parallel grid:

```cpp
    COUT << "Parallel grid size: ("<<parallel.grid_size()[0]<<","<<parallel.
      grid_size()[1]<<"). "<<endl;
```

The command COUT is a definition of

```
if(parallel.isRoot())cout
```

and therefore only the root process (the one with rank 0) will perform the output.

7.2.2.2 Lattice declaration

```
//------------   Declaration of a Lattice   -------------
```

LATfield2 can work with lattice of N-dimensions, with N larger or equal to 2. In this example we will declare a lattice with 3 dimensions.

```
int dim = 3;
int latSize[dim] = {25,57,32};
int halo = 1;
Lattice lat(dim,latSize,halo);
```

Note that the halo (number of ghost cells) is a single integer, as each dimension of the lattice have the same halo within LATfield2. Also note that if boxSize is an integer (instead of a pointer) this will initialize the lattice with each dimension with the same size.

Next the code outputs the size of the 3 dimension using COUT.

```
COUT << "Lattice size: ("<< lat.size(0)<<","<< lat.size(1)<<","<< lat.size(2)<<");"<<endl;
```

Then the code outputs on a single line the local description of the lattice. First the MPI ranks of each process, first the world rank then the 2d ranks (the position with the parallel grid). Then the size of the local part of the lattice which is stored in this given process. Finally the offset to this local part in respect to the global lattice.

```
cout << "Process ranks: "<< parallel.rank()<<",("<< parallel.grid_rank()[0]<<","<<parallel
  .grid_rank()[1]<< "); ";
cout << "Local lattice size: ("<< lat.sizeLocal(0)<<","<< lat.sizeLocal(1)<<","<< lat.sizeLocal(2)<<");
  ";
cout << "Coordinate of the first local point: (0,"<< lat.coordSkip()[1] <<","<< lat.coordSkip()[0] <<"
  )."<<endl;
```

7.2.2.3 Field declaration

```
//-----------   Declaration of the Fields   --------------
```

In this simple example we want to work with 3 fields, ϕ, ρ and $\nabla\phi$ which are named in the code phi, rho and gradphi. One can declare, initialize and allocate field using:

```
Field<Real> rho(lat);
Field<Real> gradPhi(lat,3);
```

In the case that declaration should be separately performed (as within a class declaration, or when several fields point to the same array and therefore should not be allocated), one can use the following 3 command:

```
Field<Real> phi;
phi.initialize(lat);
phi.alloc();
```

7.2.2.4 Field opperations

```
//-------------    Operations on Fields    ---------------
```

First, we would like to initialize the value of ϕ to be a gaussian defined as $\phi = e^{-2(\vec{x}-\vec{x}_0)^2}$. To do so we have to declare a Site object on the lattice of ϕ. This object will allow to work with coordinate.

```
Site x(lat);
```

Then it is possible to loop over all site using the usual for loop:

```
double x2;
for(x.first();x.test();x.next())
{
    x2  = pow(0.5 + x.coord(0) - lat.size(0)/2.,2.);
    x2 += pow(0.5 + x.coord(1) - lat.size(1)/2.,2.);
    x2 += pow(0.5 + x.coord(2) - lat.size(2)/2.,2.);
    phi(x) = exp(-x2 * 2.);
}
```

In that loop the exponential is built with $x_0 = latticeSize/2 + 0.5$ this mean that the gaussian is centered at the center of the lattice center cell. The coord method return the coordinate of the site object in lattice units.

Then some spacial derivative are computed to build $gradPhi = \vec{\nabla}\phi$ and $\rho = \Delta\phi$. The first thing to do is to update the ghost cells (halo) of ϕ.

```
phi.updateHalo();
```

Then the derivative can be performed.

```
for(x.first();x.test();x.next())
{

    gradPhi(x,0) = (phi(x+0)-phi(x-0));
    gradPhi(x,1) = (phi(x+1)-phi(x-1));
    gradPhi(x,2) = (phi(x+2)-phi(x-2));

    rho(x)=0;
    for(int i=0;i<3;i++)rho(x) += phi(x+i) - 2 * phi(x) + phi(x-i);

}
```

This shows how to work with file with multiple components (gradPhi), and also exhibits the usage of the Site object for displacement in the Lattice. The operator + and - are overloaded to provide an extremely easy way to move on the lattice. Both operator will return the neighbor site in the direction specified after the operator.

7.2.2.5 Field I/O

```
//-----------------    writing fields    ------------------
```

This method allow to write a file with different file format. Using HDF5 or using ASCII format.

```
    string str_filename = "./test_phi";
#ifdef HDF5
    str_filename += ".h5";
    phi.saveHDF5(str_filename);
    str_filename = "test_phi_slice.h5";
    phi.saveSliceHDF5(str_filename,0,3);
#else
    str_filename += ".txt";
    phi.write(str_filename);
#endif
```

7.3 IOserver

This is very simple example to show the usage of the I/O server. It writes two very simple files in ASCII, in which each compute core writes a single line. Such usage are clearly unrealistic, and speed gain are only found at much larger files.

The server works as follow. It needs to be started only on the I/O cores. Then the compute core will process their usual computation. Once some I/O has to be performed, the compute cores have to open an "ostream" to the I/O cores, then files can be created. Only one file can be open simultaneously currently, and there is no method to reopen a file which have been closed. This feature will be added, in the next update of the server. Once all data have been transferred to the I/O cores, one can close the "ostream". This operation will launch the writing to disks procedure and the server will be in a busy state until all data has been written to disks.

```cpp
#include <iostream>
#include "LATfield2.hpp"

using namespace LATfield2;


int main(int argc, char **argv)
{
    int n,m;
    int io_size;
    int io_groupe_size;


    for (int i=1 ; i < argc ; i++ ){
        if ( argv[i][0] != '-' )
            continue;
        switch(argv[i][1]) {
            case 'n':
                n = atoi(argv[++i]);
                break;
            case 'm':
                m =  atoi(argv[++i]);
                break;
            case 'i':
                io_size =  atoi(argv[++i]);
                break;
            case 'g':
                io_groupe_size = atoi(argv[++i]);
                break;
        }
    }

    parallel.initialize(n,m,io_size,io_groupe_size);

    if(parallel.isIO()) IO_Server.start();
    else
    {
        string filename = "./testfile";
        ioserver_file file;

        string sentence;
        sentence = "I am the " + int2string(parallel.world_rank(),99999) + " MPI
    process. My rank is ";
        sentence += int2string(parallel.rank(),99999) + " in the compute group. I have the
    position (";
        sentence += int2string(parallel.grid_rank()[0],99999) + "," +
    int2string(parallel.grid_rank()[1],99999);
        sentence += ") in the processes grid." ;

        char * sendbuffer;
        sendbuffer = (char*)malloc(sentence.size()+1);
        for(int i=0;i<sentence.size();i++)sendbuffer[i]=sentence[i];
        sendbuffer[sentence.size()]='\n';


        while(IO_Server.openOstream()==OSTREAM_FAIL)usleep(50);

        file = IO_Server.createFile(filename);
        IO_Server.writeBuffer(file,sendbuffer,sentence.size()+1);
        IO_Server.closeFile(file);

        IO_Server.closeOstream();

        IO_Server.stop();
    }
```

```
}
```

7.3.1 Compile and Run

Go to the LATfield2/examples folder, and compile this example with e.g. mpic++:

```
mpic++ -o ioserver_exec IOserver.cpp -I../ -DEXTERNAL_IO
```

It can be executed using (here using "mpirun -np 4" to run with 4 process):

```
mpirun -np 24 ./ioserver_exec -n 4 -m 4 -i 8 -g 4
```

The n and m parameter are as usual parameters to initialize the parallel object. Then 2 additional parameters are passed. First -i which is the total number of MPI processes of the IO server, then -g is the number of IO process which write data in a single file. Therefor n∗m+i need to be equal to the total number of MPI processes used by the job and i/g must be an integer.

7.3.2 Going through the code

7.3.2.1 Parallel initialization and server launch

The initialization of the parallel object is performed as without the IO server, the only difference is that two addition parameter are passed to the parallel object. The number of process of the IO server, and the size of one group of the server, which is the number of process which write in the same disk. It is advised to set the group size to be an integer multiple of the number of core on a node.

```cpp
#include <iostream>
#include "LATfield2.hpp"

using namespace LATfield2;


int main(int argc, char **argv)
{
    int n,m;
    int io_size;
    int io_groupe_size;

    for (int i=1 ; i < argc ; i++ ){
        if ( argv[i][0] != '-' )
            continue;
        switch(argv[i][1]) {
            case 'n':
                n = atoi(argv[++i]);
                break;
            case 'm':
                m =  atoi(argv[++i]);
                break;
            case 'i':
                io_size =  atoi(argv[++i]);
                break;
            case 'g':
                io_groupe_size = atoi(argv[++i]);
                break;
        }
    }

    parallel.initialize(n,m,io_size,io_groupe_size);
```

Once the parallel object is initialized, the parallel object contains a list of compute and IO cores. The method isIO() return true for IO process and false for compute ones. Basicaly, a IO process has to perform only one operation, launching the server. This is perform using the start() method.

```
    if(parallel.isIO()) IO_Server.start();
```

7.3.2.2 Compute processes

Then the part of the code executed by the compute process need to be in the else of the previous if, or in a block within

```
if(!parallel.isIO()){...}.
```

In this simple example, each process will write a simple sentence which contains its position in the process grid.

```
    else
    {
        string filename = "./testfile";
        ioserver_file file;

        string sentence;
        sentence = "I am the " + int2string(parallel.world_rank(),99999) + " MPI
    process. My rank is ";
        sentence += int2string(parallel.rank(),99999) + " in the compute group. I have the
    position (";
        sentence += int2string(parallel.grid_rank()[0],99999) + "," +
    int2string(parallel.grid_rank()[1],99999);
        sentence += ") in the processes grid." ;

        char * sendbuffer;
        sendbuffer = (char*)malloc(sentence.size()+1);
        for(int i=0;i<sentence.size();i++)sendbuffer[i]=sentence[i];
        sendbuffer[sentence.size()]='\n';
```

In real applications the construction of the data which need to be send should be done only when the file is open, and sent to the server by several message (order 5 is the most efficient). This allow a much better usage of the IOserver.

The first step to start a transfer from to the IOserver is to open an ostream to the server on the compute processes. The server has no method to wait until ready. Ostream can be open only if the server is in the ready state. The server can be in busy state for two reasons. First, it has not finishin launching, and secondly the server is currently writing a file.

```
        while(IO_Server.openOstream()==OSTREAM_FAIL)usleep(50);
```

Once a stream is open one can open a file. Currently it is possible to open 5 file within one ostream, but not simultaneously (the next version of the server will be able to deal with multiple file simultaneously). Currently, there is no method to open an existing file. The only method which open a file is createFile, which will create a file or trunk it if existing.

```
        file = IO_Server.createFile(filename);
```

Once a file is open, data can be transferred to the IOserver using the write method. In this method the size of the sendBuffer is given in bytes, and the buffer is expected to be a pointer to a char array. (so if not a char array, typecast it: (char∗))

```
        IO_Server.writeBuffer(file,sendbuffer,sentence.size()+1);
```

When all data have been send to the file, one can close the file. Which is very important, as when a file is open, the server will continuously look for data send by the compute process to this file.

```
        IO_Server.closeFile(file);
```

Once every data has been sent to the file where it should be written, the ostream must be closed, which launches the transfer from the server to the disks. At that moment the server will turn its state to busy. Once the data are written on the disk a new ostream can be open.

```
IO_Server.closeOstream();
```

One need to be aware that to correctly terminate MPI, the server has to shut down. Otherwise the process of the IOserver will never stop to listen for an instruction to open a ostream.

The call to stop the server is performed by the compute process:

```
IO_Server.stop();
```

7.3.3 Known Issues

There is one known issue which arises from the desire to maximise speed. The server can start to write a file before it has finished receiving all the messages from the compute nodes. However, it cannot start the writing procedure before all message have been sent by the compute nodes. It means that some messages can still be in the network memory during the writing procedure. This can end with some data which should be written in the file of snapshot n instead being written into the file of snapshot n+1. We regard this as an acceptable risk for the rewards of increased efficiency of the server. It can be a problem for last snapshot, as there is not a subsequent write procedure. To avoid losing some data, one should open a file and call a write just before shutting down the server, without sending any message from the compute node. This issue is only present when the compute node executable is not very well balanced, and therefore should be very rare.

7.4 poissonSolver

A simple poisson solver which verifies its result. The code will be documented in a close future, but even without documentation it is a useful example

```cpp
#include <iostream>
#include "LATfield2.hpp"

using namespace LATfield2;


int main(int argc, char **argv)
{
    int n,m;
    int BoxSize = 64;
    int halo = 1;
    int khalo =0;
    int dim = 3;
    int comp = 1;
    double sigma2=1.0;
    double res =0.5;


    for (int i=1 ; i < argc ; i++ ){
        if ( argv[i][0] != '-' )
            continue;
        switch(argv[i][1]) {
            case 'n':
                n = atoi(argv[++i]);
                break;
            case 'm':
                m =  atoi(argv[++i]);
                break;
            case 'b':
                BoxSize = atoi(argv[++i]);
                break;
        }
```

```
    }

    parallel.initialize(n,m);


    double res2 =res*res;
    double renormFFT;



    Lattice lat(dim,BoxSize,halo);
    Lattice latK;
    latK.initializeRealFFT(lat, khalo);

    Site x(lat);
    rKSite k(latK);

    Field<Real> phi;
    phi.initialize(lat,comp);
    Field<Imag> phiK;
    phiK.initialize(latK,comp);
    PlanFFT<Imag> planPhi(&phi,&phiK);


    Field<Real> rhoVerif(lat,comp);
    Field<Real> rho;
    rho.initialize(lat,comp);
    Field<Imag> rhoK;
    rhoK.initialize(latK,comp);
    PlanFFT<Imag> planRho(&rho,&rhoK);


    renormFFT=(double)lat.sites();

    //fill rho with a gaussian:

    double mean = 0.;

    sigma2 = BoxSize*BoxSize/9.;

    for(x.first();x.test();x.next())
    {
        double x2 = pow(0.5l + x.coord(0) - lat.size(0)/2,2);
        x2 += pow(0.5l + x.coord(1) - lat.size(1)/2,2);
        x2 += pow(0.5l + x.coord(2) - lat.size(2)/2,2);
        rho(x)= 1.0 * exp(-x2/sigma2) + 0.1;
    mean += rho(x);
    }

    parallel.sum(mean);
    mean /= (double) lat.sites();

    planRho.execute(FFT_FORWARD);


    k.first();
    if(parallel.isRoot())
    {
        phiK(k)=0.0;
        k.next();
    }
    for(;k.test();k.next())
    {
        phiK(k)= rhoK(k) /
        ( 2.0 *(cos(2.0*M_PI*k.coord(0)/BoxSize)
            + cos(2.0*M_PI*k.coord(1)/BoxSize)
            + cos(2.0*M_PI*k.coord(2)/BoxSize)-3.0)/res2 );
    }

    planPhi.execute(FFT_BACKWARD);

    phi.updateHalo();

    for(x.first();x.test();x.next())
    {
        rhoVerif(x) =  (phi(x+0) - 2 * phi(x) + phi(x-0))/res2;
        rhoVerif(x) += (phi(x+1) - 2 * phi(x) + phi(x-1))/res2;
        rhoVerif(x) += (phi(x+2) - 2 * phi(x) + phi(x-2))/res2;
    }



    double error;

    double maxError=0;
```

```
    double minError=20;
    double averageError=0;

    for(x.first();x.test();x.next())
    {
        error =  (fabs( rho(x)- (mean + rhoVerif(x)/renormFFT) ) ) / fabs(rho(x));
        averageError+=error;
        if(minError>error)minError=error;
        if(maxError<error)maxError=error;

    }

    parallel.max(maxError);
    parallel.min(minError);
    parallel.sum(averageError);

    averageError/=renormFFT;

    parallel.barrier();

    COUT<<"Min error: "<<minError<<", Max error: "<<maxError<<" , Average error: "<<averageError<<endl;

#ifdef SINGLE
#define TOLERANCE 1.0e-6
#else
#define TOLERANCE 1.0e-11
#endif
    if (maxError > TOLERANCE) exit(max(1, 1 + (int) fabs(log10(maxError))));
    else exit(0);
}
```