schwz

Generated automatically from event-based

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

This is the main page for the Schwarz library pdf documentation. The repository is hosted on `github`. Documentation on aspects such as the build system, can be found at the # Installation Instructions page.

Modules

The structure of the Schwarz Library code is divided into different `modules` :

- Initialization : Handles the initialization of the problem and the solver.
- Communicate : Handles the communication.
- Solve : Handles the local solution and the convergence detection.
- Schwarz Class : The Classes related to the Schwarz solvers.
- Utils : Provides some basic utilities.

Chapter 2

Installation Instructions

Building

Use the standard cmake build procedure:

```
mkdir build; cd build
cmake -G "Unix Makefiles" [OPTIONS] .. && make
```

Replace `[OPTIONS]` with desired cmake options for your build. The library adds the following additional switches to control what is being built:

- `-DSCHWARZ_BUILD_BENCHMARKING={ON, OFF}` Builds some example benchmarks. Default is `ON`
- `-DSCHWARZ_BUILD_METIS={ON, OFF}` Builds with support for the `METIS` partitioner. User needs to provide the path to the installation of the `METIS` library in `METIS_DIR`, preferably as an environment variable. Default is `OFF`
- `-DSCHWARZ_BUILD_CHOLMOD={ON, OFF}` Builds with support for the `CHOLMOD` module from the Suitesparse library. User needs to set an environment variable `CHOLMOD_DIR` to the path containing the `CHOLMOD` installation. Default is `OFF`
- `-DSCHWARZ_BUILD_CUDA={ON, OFF}` Builds with CUDA support. Though Ginkgo provides most of the required CUDA support, we do need to link to CUDA for explicit setting of GPU affinities, some custom gather and scatter operations. Default is `OFF`.
- `-DSCHWARZ_BUILD_CLANG_TIDY={ON, OFF}` Builds with support for `clang-tidy` Default is `OFF`
- `-DSCHWARZ_BUILD_DEALII={ON, OFF}` Builds with support for the finite element library `deal.ii` Default is `OFF`
- `-DSCHWARZ_WITH_HWLOC={ON, OFF}` Builds with support for the hardware locality library used for binding hardware. `hwloc` is distributed as a part of the Open-MPI project. Default is `ON`
- `-DSCHWARZ_DEVEL_TOOLS={ON, OFF}` Builds with some developer tools support. Default is `ON`. In particular uses `git-cmake-format` to automatically format the source files with `clang-format`.

Tips

- If you are having CUDA problems and you are not using CUDA, then feel free to switch the CUDA module off with `-DSCHWARZ_BUILD_CUDA=off`.
- Installing CHOLMOD can be a bit annoying. TODO add some details on fixing Suitesparse compilation.
- When doing merge commits it is possible that make format does not work. You can run `cmake -DSCH↩ WARZ_DEVEL_TOOLS=OFF ..` to temporarily switch off the formatting. Please switch it on again when committing normally.

Chapter 3

Testing Instructions

Chapter 4

Benchmarking.

Benchmark example 1.

Poisson solver using Restricted Additive Schwarz with overlap.

The flag `-DSCHWARZ_BUILD_BENCHMARKING` (default `ON`) enables the example and benchmarking snippets. The following command line options are available for this example. This is setup using `gflags`.

The executable is run in the following fashion:

“‘sh [MPI_COMMAND] [MPI_OPTIONS]

Chapter 5

Module Documentation

5.1 Communicate

A module dedicated to the Communication interface in schwarz-lib.

Namespaces

- schwz::CommHelpers

  The CommHelper namespace .
- ProcessTopology

  The ProcessTopology namespace .

Classes

- class schwz::Communicate< ValueType, IndexType >

  The communication class that provides the methods for the communication between the subdomains.
- struct schwz::Metadata< ValueType, IndexType >

  The solver metadata struct.

5.1.1 Detailed Description

A module dedicated to the Communication interface in schwarz-lib.

5.2 Initialization

A module dedicated to the initialization and setup and usage of the solvers in schwarz-lib.

Namespaces

- schwz::PartitionTools

  The PartitionTools namespace .
- ProcessTopology

  The ProcessTopology namespace .

Classes

- class schwz::device_guard

  This class defines a device guard for the cuda functions and the cuda module.
- class schwz::Initialize< ValueType, IndexType >

  The initialization class that provides methods for initialization of the solver.
- struct schwz::Settings

  The struct that contains the solver settings and the parameters to be set by the user.
- struct schwz::Metadata< ValueType, IndexType >

  The solver metadata struct.

5.2.1 Detailed Description

A module dedicated to the initialization and setup and usage of the solvers in schwarz-lib.

5.3 Schwarz Class

A module dedicated to the Schwarz solver classes in schwarz-lib.

Classes

- class schwz::SolverRAS< ValueType, IndexType >

  An implementation of the solver interface using the RAS solver.
- class schwz::SchwarzBase< ValueType, IndexType >

  The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.

5.3.1 Detailed Description

A module dedicated to the Schwarz solver classes in schwarz-lib.

5.4 Solve

A module dedicated to the solvers including local solution and convergence detection in schwarz-lib.

Namespaces

- schwz::conv_tools

  The conv_tools namespace .
- schwz::SolverTools

  The SolverTools namespace .

Classes

- struct schwz::Metadata< ValueType, IndexType >

  The solver metadata struct.
- class schwz::Solve< ValueType, IndexType >

  The Solver class the provides the solver and the convergence checking methods.

5.4.1 Detailed Description

A module dedicated to the solvers including local solution and convergence detection in schwarz-lib.

5.5 Utils

A module dedicated to the utilities in schwarz-lib.

Classes

- struct schwz::Utils< ValueType, IndexType >

  The utilities class which provides some checks and basic utilities.

5.5.1 Detailed Description

A module dedicated to the utilities in schwarz-lib.

Chapter 6

Namespace Documentation

6.1 ProcessTopology Namespace Reference

The ProcessTopology namespace .

6.1.1 Detailed Description

The ProcessTopology namespace .

proc_topo

6.2 schwz Namespace Reference

The Schwarz wrappers namespace.

Namespaces

- CommHelpers

  The CommHelper namespace .
- conv_tools

  The conv_tools namespace .
- PartitionTools

  The PartitionTools namespace .
- SolverTools

  The SolverTools namespace .

Classes

- class Communicate

  The communication class that provides the methods for the communication between the subdomains.
- class device_guard

  This class defines a device guard for the cuda functions and the cuda module.
- class Initialize

  The initialization class that provides methods for initialization of the solver.
- struct Metadata

  The solver metadata struct.
- class SchwarzBase

  The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.
- struct Settings

  The struct that contains the solver settings and the parameters to be set by the user.
- class Solve

  The Solver class the provides the solver and the convergence checking methods.
- class SolverRAS

  An implementation of the solver interface using the RAS solver.
- struct Utils

  The utilities class which provides some checks and basic utilities.

6.2.1 Detailed Description

The Schwarz wrappers namespace.

6.3 schwz::CommHelpers Namespace Reference

The CommHelper namespace .

6.3.1 Detailed Description

The CommHelper namespace .

comm_helpers

6.4 schwz::conv_tools Namespace Reference

The conv_tools namespace .

6.4.1 Detailed Description

The conv_tools namespace .

conv_tools

6.5 schwz::PartitionTools Namespace Reference

The PartitionTools namespace .

6.5.1 Detailed Description

The PartitionTools namespace .

part_tools

6.6 schwz::SolverTools Namespace Reference

The SolverTools namespace .

6.6.1 Detailed Description

The SolverTools namespace .

solver_tools

Chapter 7

Class Documentation

7.1 BadDimension Class Reference

BadDimension is thrown if an operation is being applied to a LinOp with bad dimensions.

`#include <exception.hpp>`

Public Member Functions

- BadDimension (const std::string &file, int line, const std::string &func, const std::string &op_name, std::size_t op_num_rows, std::size_t op_num_cols, const std::string &clarification)

  Initializes a bad dimension error.

7.1.1 Detailed Description

BadDimension is thrown if an operation is being applied to a LinOp with bad dimensions.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BadDimension()

```
BadDimension::BadDimension (
            const std::string & file,
            int line,
            const std::string & func,
            const std::string & op_name,
            std::size_t op_num_rows,
            std::size_t op_num_cols,
            const std::string & clarification )  [inline]
```

Initializes a bad dimension error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
func	The function name where the error occurred
op_name	The name of the operator
op_num_rows	The row dimension of the operator
op_num_cols	The column dimension of the operator
clarification	An additional message further describing the error

```
115         : Error(file, line,
116                 func + ": Object " + op_name + " has dimensions [" +
117                     std::to_string(op_num_rows) + " x " +
118                     std::to_string(op_num_cols) + "]: " + clarification)
119     {}
```

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

- exception.hpp (321b91c)

7.2 schwz::Settings::comm_settings Struct Reference

The settings for the various available communication paradigms.

`#include <settings.hpp>`

Public Attributes

- bool enable_onesided = false

  Enable one-sided communication.
- bool enable_overlap = false

  Enable explicit overlap between communication and computation.
- bool enable_put = false

  Put the data to the window using MPI_Put rather than get.
- bool enable_get = true

  Get the data to the window using MPI_Get rather than put.
- bool enable_one_by_one = false

  Push each element separately directly into the buffer.
- bool enable_flush_local = false

  Use local flush.
- bool enable_flush_all = true

  Use flush all.
- bool enable_lock_local = false

  Use local locks.
- bool enable_lock_all = true

  Use lock all.

7.2.1 Detailed Description

The settings for the various available communication paradigms.

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

- settings.hpp (321b91c)

7.3 schwz::Communicate< ValueType, IndexType >::comm_struct Struct Reference

The communication struct used to store the communication data.

`#include <communicate.hpp>`

Public Attributes

- int num_neighbors_in
  The number of neighbors this subdomain has to receive data from.
- int num_neighbors_out
  The number of neighbors this subdomain has to send data to.
- int num_recv
  The total number of elements received from all neighbors.
- int num_send
  The total number of elements sent to all neighbors.
- std::shared_ptr< gko::Array< IndexType > > neighbors_in
  The neighbors this subdomain has to receive data from.
- std::shared_ptr< gko::Array< IndexType > > neighbors_out
  The neighbors this subdomain has to send data to.
- std::vector< bool > is_local_neighbor
  The bool vector which is true if the neighbors of a subdomain are in one node.
- int local_num_neighbors_in
  The number of neighbors this subdomain has to receive data from.
- int local_num_neighbors_out
  The number of neighbors this subdomain has to send data to.
- std::shared_ptr< gko::Array< IndexType > > local_neighbors_in
  The neighbors this subdomain has to receive data from.
- std::shared_ptr< gko::Array< IndexType > > local_neighbors_out
  The neighbors this subdomain has to send data to.
- std::shared_ptr< gko::Array< IndexType * > > global_put
  The array containing the number of elements that each subdomain sends from the other.
- std::shared_ptr< gko::Array< IndexType * > > local_put
  The array containing the number of elements that each subdomain sends from the other.
- std::shared_ptr< gko::Array< IndexType * > > remote_put
  The array containing the number of elements that each subdomain sends from the other.
- std::shared_ptr< gko::Array< IndexType * > > global_get
  The array containing the number of elements that each subdomain gets from the other.
- std::shared_ptr< gko::Array< IndexType * > > local_get
  The array containing the number of elements that each subdomain gets from the other.

- std::shared_ptr< gko::Array< IndexType * > > remote_get

  The array containing the number of elements that each subdomain gets from the other.
- std::shared_ptr< gko::Array< IndexType > > window_ids

  The RDMA window ids.
- std::shared_ptr< gko::Array< IndexType > > windows_from

  The RDMA window ids to receive data from.
- std::shared_ptr< gko::Array< IndexType > > windows_to

  The RDMA window ids to send data to.
- std::shared_ptr< gko::Array< MPI_Request > > put_request

  The put request array.
- std::shared_ptr< gko::Array< MPI_Request > > get_request

  The get request array.
- std::shared_ptr< gko::matrix::Dense< ValueType > > send_buffer

  The send buffer used for the actual communication for both one-sided and two-sided.
- std::shared_ptr< gko::matrix::Dense< ValueType > > recv_buffer

  The recv buffer used for the actual communication for both one-sided and two-sided.
- std::shared_ptr< gko::matrix::Dense< ValueType > > extra_buffer

  The extrapolation buffer used for extrapolation of values at the receiver.
- std::shared_ptr< gko::matrix::Dense< ValueType > > last_recv_bdy

  The last received boundary values for each of the in neighbors for extrapolation.
- std::shared_ptr< gko::matrix::Dense< ValueType > > curr_send_avg

  Average of values in the send buffer for each of the out neighbors.
- std::shared_ptr< gko::matrix::Dense< ValueType > > last_send_avg

  Average of values in the last send buffer for each of the out neighbors.
- std::shared_ptr< gko::matrix::Dense< ValueType > > curr_recv_avg

  Average of values in the recv buffer for each of the out neighbors.
- std::shared_ptr< gko::matrix::Dense< ValueType > > last_recv_avg

  Average of values in the last recv buffer for each of the out neighbors.
- std::shared_ptr< gko::Array< IndexType > > msg_count

  Number of messages sent.
- std::shared_ptr< gko::Array< IndexType > > last_recv_iter

  Iteration stamp of last received values.
- std::shared_ptr< gko::matrix::Dense< ValueType > > last_recv_slopes

  Last recv slopes.
- std::shared_ptr< gko::matrix::Dense< ValueType > > last_sent_slopes_avg

  Last sent slopes.
- std::shared_ptr< gko::Array< IndexType > > last_sent_iter

  Iteration stamp of last received values.
- std::shared_ptr< gko::matrix::Dense< ValueType > > thres

  Threshold.
- std::shared_ptr< gko::Array< IndexType > > get_displacements

  The displacements for the receiving of the buffer.
- std::shared_ptr< gko::Array< IndexType > > put_displacements

  The displacements for the sending of the buffer.
- MPI_Win window_recv_buffer

  The RDMA window for the recv buffer.
- MPI_Win window_send_buffer

  The RDMA window for the send buffer.
- MPI_Win window_x

  The RDMA window for the solution vector.

7.3.1 Detailed Description

template<typename ValueType, typename IndexType>
struct schwz::Communicate< ValueType, IndexType >::comm_struct

The communication struct used to store the communication data.

7.3.2 Member Data Documentation

7.3.2.1 global_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType >::comm↩
_struct::global_get
```

The array containing the number of elements that each subdomain gets from the other.

For example. global_get[p][0] contains the overall number of elements to be received to subdomain p and global↩
_put[p][i] contains the index of the solution vector to be received from subdomain p.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize(), schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

7.3.2.2 global_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType >::comm↩
_struct::global_put
```

The array containing the number of elements that each subdomain sends from the other.

For example. global_put[p][0] contains the overall number of elements to be sent to subdomain p and global_put[p][i] contains the index of the solution vector to be sent to subdomain p.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize(), schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

7.3.2.3 is_local_neighbor

```
template<typename ValueType , typename IndexType >
std::vector<bool> schwz::Communicate< ValueType, IndexType >::comm_struct::is_local_neighbor
```

The bool vector which is true if the neighbors of a subdomain are in one node.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize(), schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

7.3.2.4 local_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType >::comm↩
_struct::local_get
```

The array containing the number of elements that each subdomain gets from the other.

For example. global_get[p][0] contains the overall number of elements to be received to subdomain p and global↩
_put[p][i] contains the index of the solution vector to be received from subdomain p.

Referenced by schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

7.3.2.5 local_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType >::comm↩
_struct::local_put
```

The array containing the number of elements that each subdomain sends from the other.

For example. global_put[p][0] contains the overall number of elements to be sent to subdomain p and global_put[p][i] contains the index of the solution vector to be sent to subdomain p.

Referenced by schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

7.3.2.6 remote_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType >::comm↩
_struct::remote_get
```

The array containing the number of elements that each subdomain gets from the other.

For example. global_get[p][0] contains the overall number of elements to be received to subdomain p and global↩
_put[p][i] contains the index of the solution vector to be received from subdomain p.

Referenced by schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

7.3.2.7 remote_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType >::comm↩
_struct::remote_put
```

The array containing the number of elements that each subdomain sends from the other.

For example. global_put[p][0] contains the overall number of elements to be sent to subdomain p and global_put[p][i] contains the index of the solution vector to be sent to subdomain p.

Referenced by schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

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

- communicate.hpp (321b91c)

7.4 schwz::Communicate< ValueType, IndexType > Class Template Reference

The communication class that provides the methods for the communication between the subdomains.

`#include <communicate.hpp>`

Classes

- struct comm_struct

  The communication struct used to store the communication data.

Public Member Functions

- virtual void setup_comm_buffers ()=0

  Sets up the communication buffers needed for the boundary exchange.
- virtual void setup_windows (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &main_buffer)=0

  Sets up the windows needed for the asynchronous communication.
- virtual void exchange_boundary (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &solution, std::shared_ptr< gko::matrix↩ ::Dense< ValueType >> &last_solution, std::ofstream &fps, std::ofstream &fpr)=0

  Exchanges the elements of the solution vector.
- void local_to_global_vector (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, const std::shared_ptr< gko::matrix::Dense< ValueType >> &local_vector, std::shared_ptr< gko::matrix::↩ Dense< ValueType >> &global_vector)

  Transforms data from a local vector to a global vector.
- virtual void update_boundary (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &local_solution, const std::shared_ptr< gko↩ ::matrix::Dense< ValueType >> &local_rhs, const std::shared_ptr< gko::matrix::Dense< ValueType >> &global_solution, const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &interface_matrix)=0

  Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.
- void clear (Settings &settings)

  Clears the data.

7.4.1 Detailed Description

template<typename ValueType, typename IndexType>
class schwz::Communicate< ValueType, IndexType >

The communication class that provides the methods for the communication between the subdomains.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

Communicate

7.4.2 Member Function Documentation

7.4.2.1 exchange_boundary()

```
template<typename ValueType , typename IndexType >
void schwz::Communicate< ValueType, IndexType >::exchange_boundary (
            const Settings & settings,
            const Metadata< ValueType, IndexType > & metadata,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & solution,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & last_solution,
            std::ofstream & fps,
            std::ofstream & fpr )  [pure virtual]
```

Exchanges the elements of the solution vector.

Parameters

settings	The settings struct.
metadata	The metadata struct.
global_solution	The solution vector being exchanged between the subdomains.

Implemented in schwz::SolverRAS< ValueType, IndexType >.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::run().

7.4.2.2 local_to_global_vector()

```
template<typename ValueType , typename IndexType >
void schwz::Communicate< ValueType, IndexType >::local_to_global_vector (
            const Settings & settings,
            const Metadata< ValueType, IndexType > & metadata,
            const std::shared_ptr< gko::matrix::Dense< ValueType >> & local_vector,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & global_vector )
```

Transforms data from a local vector to a global vector.

Parameters

settings	The settings struct.
metadata	The metadata struct.
local_vector	The local vector in question.
global_vector	The global vector in question.

```
71 {
72     using vec = gko::matrix::Dense<ValueType>;
73     auto alpha = gko::initialize<gko::matrix::Dense<ValueType>>(
74         {1.0}, settings.executor);
75     auto temp_vector = vec::create(
76         settings.executor, gko::dim<2>(metadata.local_size, 1),
77         gko::Array<ValueType>::view(
78             settings.executor, metadata.local_size,
79             &global_vector->get_values()[metadata.first_row
80                                          ->get_data()[metadata.my_rank]]),
81         1);
82
83     auto temp_vector2 = vec::create(
84         settings.executor, gko::dim<2>(metadata.local_size, 1),
85         gko::Array<ValueType>::view(settings.executor, metadata.local_size,
86                                     local_vector->get_values()),
87         1);
88     if (settings.convergence_settings.convergence_crit ==
89         Settings::convergence_settings::local_convergence_crit::
90             residual_based) {
91         local_vector->add_scaled(alpha.get(), temp_vector.get());
92         temp_vector->add_scaled(alpha.get(), local_vector.get());
93     } else {
94         temp_vector->copy_from(temp_vector2.get());
95     }
96 }
```

7.4.2.3 setup_windows()

```
template<typename ValueType , typename IndexType >
void schwz::Communicate< ValueType, IndexType >::setup_windows (
             const Settings & settings,
             const Metadata< ValueType, IndexType > & metadata,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & main_buffer )  [pure virtual]
```

Sets up the windows needed for the asynchronous communication.

Parameters

settings	The settings struct.
metadata	The metadata struct.
main_buffer	The main buffer being exchanged between the subdomains.

Implemented in schwz::SolverRAS< ValueType, IndexType >.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::run().

7.4.2.4 update_boundary()

```
template<typename ValueType , typename IndexType >
void schwz::Communicate< ValueType, IndexType >::update_boundary (
            const Settings & settings,
            const Metadata< ValueType, IndexType > & metadata,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & local_solution,
            const std::shared_ptr< gko::matrix::Dense< ValueType >> & local_rhs,
            const std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution,
            const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_↩
matrix )  [pure virtual]
```

Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.

Parameters

settings	The settings struct.
metadata	The metadata struct.
local_solution	The local solution vector in the subdomain.
local_rhs	The local right hand side vector in the subdomain.
global_solution	The workspace solution vector.
global_old_solution	The global solution vector of the previous iteration.
interface_matrix	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.

Implemented in schwz::SolverRAS< ValueType, IndexType >.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::run().

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

- communicate.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp (321b91c)

7.5 schwz::Settings::convergence_settings Struct Reference

The various convergence settings available.

```
#include <settings.hpp>
```

7.5.1 Detailed Description

The various convergence settings available.

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

- settings.hpp (321b91c)

7.6 CudaError Class Reference

CudaError is thrown when a CUDA routine throws a non-zero error code.

`#include <exception.hpp>`

Public Member Functions

- CudaError (const std::string &file, int line, const std::string &func, int error_code)
  Initializes a CUDA error.

7.6.1 Detailed Description

CudaError is thrown when a CUDA routine throws a non-zero error code.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 CudaError()

```
CudaError::CudaError (
            const std::string & file,
            int line,
            const std::string & func,
            int error_code )  [inline]
```

Initializes a CUDA error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
func	The name of the CUDA routine that failed
error_code	The resulting CUDA error code

```
137         : Error(file, line, func + ": " + get_error(error_code))
138     {}
```

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

- exception.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (321b91c)

7.7 CusparseError Class Reference

CusparseError is thrown when a cuSPARSE routine throws a non-zero error code.

`#include <exception.hpp>`

Public Member Functions

- CusparseError (const std::string &file, int line, const std::string &func, int error_code)
  Initializes a cuSPARSE error.

7.7.1 Detailed Description

CusparseError is thrown when a cuSPARSE routine throws a non-zero error code.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 CusparseError()

```
CusparseError::CusparseError (
            const std::string & file,
            int line,
            const std::string & func,
            int error_code )  [inline]
```

Initializes a cuSPARSE error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
func	The name of the cuSPARSE routine that failed
error_code	The resulting cuSPARSE error code

```
159         : Error(file, line, func + ": " + get_error(error_code))
160     {}
```

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

- exception.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (321b91c)

7.8 schwz::device_guard Class Reference

This class defines a device guard for the cuda functions and the cuda module.

`#include <device_guard.hpp>`

7.8.1 Detailed Description

This class defines a device guard for the cuda functions and the cuda module.

The guard is used to make sure that the device code is run on the correct cuda device, when run with multiple devices. The class records the current device id and uses `cudaSetDevice` to set the device id to the one being passed in. After the scope has been exited, the destructor sets the device_id back to the one before entering the scope.

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

- device_guard.hpp (321b91c)

7.9 schwz::Initialize< ValueType, IndexType > Class Template Reference

The initialization class that provides methods for initialization of the solver.

`#include <initialization.hpp>`

Public Member Functions

- void generate_random_rhs (std::vector< ValueType > &rhs)

  Generates a random right hand side vector.
- void generate_dipole_rhs (std::vector< ValueType > &rhs)

  Generates a dipole right hand side vector.
- void generate_sin_rhs (std::vector< ValueType > &rhs)

  Generates a sinusoidal right hand side vector.
- void setup_global_matrix (const std::string &filename, const gko::size_type &oned_laplacian_size, std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &global_matrix)

  Generates the 2D global laplacian matrix.
- void partition (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &global_matrix, std::vector< unsigned int > &partition_indices)

  The partitioning function.
- void setup_vectors (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std::vector< ValueType > &rhs, std::shared_ptr< gko::matrix::Dense< ValueType >> &local_rhs, std::shared_ptr< gko::matrix::Dense< ValueType >> &global_rhs, std::shared_ptr< gko::matrix::Dense< ValueType >> &local_solution, std::shared_ptr< gko::matrix::Dense< ValueType >> &last_solution)

  Setup the vectors with default values and allocate mameory if not allocated.
- virtual void setup_local_matrices (Settings &settings, Metadata< ValueType, IndexType > &metadata, std::vector< unsigned int > &partition_indices, std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &global_matrix, std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &local_matrix, std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &interface_matrix)=0

  Sets up the local and the interface matrices from the global matrix and the partition indices.

Public Attributes

- std::vector< unsigned int > partition_indices

  The partition indices containing the subdomains to which each row(vertex) of the matrix(graph) belongs to.
- std::vector< unsigned int > cell_weights

  The cell weights for the partition algorithm.

Additional Inherited Members

7.9.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::Initialize< ValueType, IndexType >

The initialization class that provides methods for initialization of the solver.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

Initialization

7.9.2 Member Function Documentation

7.9.2.1 generate_dipole_rhs()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::generate_dipole_rhs (
            std::vector< ValueType > & rhs )
```

Generates a dipole right hand side vector.

Parameters

rhs	The rhs vector.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

```
99 {
100     auto oned_laplacian_size = metadata.oned_laplacian_size;
101
102     //Placing dipole at 1/4 and 3/4 of Y-dim at the middle of X-dim
103     for (int i = 0; i < oned_laplacian_size; i++)
104     {
105         for (int j = 0; j < oned_laplacian_size; j++)
106         {
```

```
107             if (i == oned_laplacian_size / 4 && j == oned_laplacian_size / 2)
108                rhs[i * oned_laplacian_size + j] = 100.0;
109             else if (i == 3 * oned_laplacian_size / 4 && j == oned_laplacian_size / 2)
110                rhs[i * oned_laplacian_size + j] = -100.0;
111             else
112                rhs[i * oned_laplacian_size + j] = 0.0;
113         }
114     }
115 }
```

7.9.2.2 generate_random_rhs()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::generate_random_rhs (
             std::vector< ValueType > & rhs )
```

Generates a random right hand side vector.

Parameters

rhs	The rhs vector.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

```
89 {
90     std::uniform_real_distribution<double> unif(0.0, 1.0);
91     std::default_random_engine engine;
92     for (gko::size_type i = 0; i < rhs.size(); ++i) {
93         rhs[i] = unif(engine);
94     }
95 }
```

7.9.2.3 generate_sin_rhs()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::generate_sin_rhs (
             std::vector< ValueType > & rhs )
```

Generates a sinusoidal right hand side vector.

Parameters

rhs	The rhs vector.

References schwz::Initialize< ValueType, IndexType >::setup_global_matrix().

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

```
119 {
120     auto PI = (ValueType) (atan(1.0) * 4);
```

```
121     auto oned_laplacian_size = metadata.oned_laplacian_size;
122
123     //Source = sin(x)sin(y)
124     for (int i = 0; i < oned_laplacian_size; i++)
125     {
126         for (int j = 0; j < oned_laplacian_size; j++)
127         {
128             rhs[i * oned_laplacian_size + j] = sin(2 * PI * i / oned_laplacian_size) *
129                                                sin(2 * PI * j / oned_laplacian_size);
130         }
131     }
132 }
```

7.9.2.4 partition()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::partition (
             const Settings & settings,
             const Metadata< ValueType, IndexType > & metadata,
             const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_↩
matrix,
             std::vector< unsigned int > & partition_indices )
```

The partitioning function.

Allows the partition of the global matrix depending with METIS and a regular 1D decomposition.

Parameters

settings	The settings struct.
metadata	The metadata struct.
global_matrix	The global matrix.
partition_indices	The partition indices [OUTPUT].

References schwz::Metadata< ValueType, IndexType >::global_size, schwz::Metadata< ValueType, IndexType >::my_rank, schwz::Metadata< ValueType, IndexType >::num_subdomains, and schwz::Settings::write_debug_↩ out.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

```
319 {
320     partition_indices.resize(metadata.global_size);
321     if (metadata.my_rank == 0) {
322         auto partition_settings =
323             (Settings::partition_settings::partition_zoltan |
324              Settings::partition_settings::partition_metis |
325              Settings::partition_settings::partition_regular |
326              Settings::partition_settings::partition_regular2d |
327              Settings::partition_settings::partition_custom) &
328             settings.partition;
329
330         if (partition_settings ==
331             Settings::partition_settings::partition_zoltan) {
332             SCHWARZ_NOT_IMPLEMENTED;
333         } else if (partition_settings ==
334                    Settings::partition_settings::partition_metis) {
335             if (metadata.my_rank == 0) {
336                 std::cout << " METIS partition" << std::endl;
337             }
338             PartitionTools::PartitionMetis(
339                 settings, global_matrix, this->cell_weights,
340                 metadata.num_subdomains, partition_indices);
```

```
341         } else if (partition_settings ==
342                    Settings::partition_settings::partition_regular) {
343             if (metadata.my_rank == 0) {
344                 std::cout << " Regular 1D partition" << std::endl;
345             }
346             PartitionTools::PartitionRegular(
347                 global_matrix, metadata.num_subdomains, partition_indices);
348         } else if (partition_settings ==
349                    Settings::partition_settings::partition_regular2d) {
350             if (metadata.my_rank == 0) {
351                 std::cout << " Regular 2D partition" << std::endl;
352             }
353             PartitionTools::PartitionRegular2D(
354                 global_matrix, settings.write_debug_out,
355                 metadata.num_subdomains, partition_indices);
356         } else if (partition_settings ==
357                    Settings::partition_settings::partition_custom) {
358             // User partitions mesh manually
359             SCHWARZ_NOT_IMPLEMENTED;
360         } else {
361             SCHWARZ_NOT_IMPLEMENTED;
362         }
363     }
364 }
```

7.9.2.5 setup_global_matrix()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::setup_global_matrix (
              const std::string & filename,
              const gko::size_type & oned_laplacian_size,
              std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_matrix )
```

Generates the 2D global laplacian matrix.

Parameters

oned_laplacian_size	The size of the one d laplacian grid.
global_matrix	The global matrix.

Referenced by schwz::Initialize< ValueType, IndexType >::generate_sin_rhs(), and schwz::SchwarzBase< ValueType, IndexType >::initialize().

```
235 {
236     using index_type = IndexType;
237     using value_type = ValueType;
238     using mtx = gko::matrix::Csr<value_type, index_type>;
239     if (settings.matrix_filename != "null") {
240         auto input_file = std::ifstream(filename, std::ios::in);
241         if (!input_file) {
242             std::cerr << "Could not find the file \"" << filename
243                       << "\", which is required for this test.\n";
244         }
245         global_matrix =
246             gko::read<mtx>(input_file, settings.executor->get_master());
247         global_matrix->sort_by_column_index();
248         std::cout << "Matrix from file " << filename << std::endl;
249     } else if (settings.matrix_filename == "null" &&
250                settings.explicit_laplacian) {
251         std::cout << "Laplacian 2D Matrix (generated in house) " << std::endl;
252         gko::size_type global_size = oned_laplacian_size *
    oned_laplacian_size;
253
254         global_matrix = mtx::create(settings.executor->get_master(),
255                                     gko::dim<2>(global_size), 5 * global_size);
256         value_type *values = global_matrix->get_values();
257         index_type *row_ptrs = global_matrix->get_row_ptrs();
```

```
258         index_type *col_idxs = global_matrix->get_col_idxs();
259
260         std::vector<gko::size_type> exclusion_set;
261
262         std::map<IndexType, ValueType> stencil_map = {
263             {-oned_laplacian_size, -1}, {-1, -1}, {0, 4}, {1, -1},
264             {oned_laplacian_size, -1},
265         };
266         for (auto i = 2; i < global_size; ++i) {
267             gko::size_type index = (i - 1) * oned_laplacian_size;
268             if (index * index < global_size * global_size) {
269                 exclusion_set.push_back(
270                     linearize_index(index, index - 1, global_size));
271                 exclusion_set.push_back(
272                     linearize_index(index - 1, index, global_size));
273             }
274         }
275
276         std::sort(exclusion_set.begin(),
277                   exclusion_set.begin() + exclusion_set.size());
278
279         IndexType pos = 0;
280         IndexType col_idx = 0;
281         row_ptrs[0] = pos;
282         gko::size_type cur_idx = 0;
283         for (IndexType i = 0; i < global_size; ++i) {
284             for (auto ofs : stencil_map) {
285                 auto in_exclusion_flag =
286                     (exclusion_set[cur_idx] ==
287                      linearize_index(i, i + ofs.first, global_size));
288                 if (0 <= i + ofs.first && i + ofs.first < global_size &&
289                     !in_exclusion_flag) {
290                     values[pos] = ofs.second;
291                     col_idxs[pos] = i + ofs.first;
292                     ++pos;
293                 }
294                 if (in_exclusion_flag) {
295                     cur_idx++;
296                 }
297                 col_idx = row_ptrs[i + 1] - pos;
298             }
299             row_ptrs[i + 1] = pos;
300         }
301     } else {
302         std::cerr << " Need to provide a matrix or enable the default "
303                      "laplacian matrix."
304                   << std::endl;
305         std::exit(-1);
306     }
307 }
```

7.9.2.6 setup_local_matrices()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::setup_local_matrices (
             Settings & settings,
             Metadata< ValueType, IndexType > & metadata,
             std::vector< unsigned int > & partition_indices,
             std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_matrix,
             std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & local_matrix,
             std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_matrix )
[pure virtual]
```

Sets up the local and the interface matrices from the global matrix and the partition indices.

Parameters

settings	The settings struct.
metadata	The metadata struct.
partition_indices	The array containing the partition indices.

Parameters

global_matrix	The global system matrix.
local_matrix	The local system matrix.
interface_matrix	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.
local_perm	The local permutation, obtained through RCM or METIS.

Implemented in schwz::SolverRAS< ValueType, IndexType >.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

7.9.2.7 setup_vectors()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::setup_vectors (
             const Settings & settings,
             const Metadata< ValueType, IndexType > & metadata,
             std::vector< ValueType > & rhs,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & local_rhs,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & global_rhs,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & local_solution,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & last_solution )
```

Setup the vectors with default values and allocate mameory if not allocated.

Parameters

settings	The settings struct.
metadata	The metadata struct.
local_rhs	The local right hand side vector in the subdomain.
global_rhs	The global right hand side vector.
local_solution	The local solution vector in the subdomain.

References schwz::Settings::executor, schwz::Metadata< ValueType, IndexType >::first_row, schwz::Metadata< ValueType, IndexType >::local_size_x, and schwz::Metadata< ValueType, IndexType >::my_rank.

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

```
375 {
376     using vec = gko::matrix::Dense<ValueType>;
377     auto my_rank = metadata.my_rank;
378     auto first_row = metadata.first_row->get_data()[my_rank];
379
380     // Copy the global rhs vector to the required executor.
381     gko::Array<ValueType> temp_rhs{settings.executor->get_master(), rhs.begin(),
382                                    rhs.end()};
383     global_rhs = vec::create(settings.executor,
384                              gko::dim<2>{metadata.global_size, 1}, temp_rhs, 1);
385
386     local_rhs =
387         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
388     // Extract the local rhs from the global rhs. Also takes into account the
389     // overlap.
390     SolverTools::extract_local_vector(settings, metadata, local_rhs.get(),
```

```
391                                     global_rhs.get(), first_row);
392
393     local_solution =
394         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
395
396     //contains the solution at the last event of communication
397     last_solution =
398         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
399 }
```

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

- initialization.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp (321b91c)

7.10 schwz::Metadata< ValueType, IndexType > Struct Template Reference

The solver metadata struct.

`#include <settings.hpp>`

Classes

- struct post_process_data

  The struct used for storing data for post-processing.

Public Attributes

- MPI_Comm mpi_communicator

  The MPI communicator.
- gko::size_type global_size = 0

  The size of the global matrix.
- gko::size_type oned_laplacian_size = 0

  The size of the 1 dimensional laplacian grid.
- gko::size_type local_size = 0

  The size of the local subdomain matrix.
- gko::size_type local_size_x = 0

  The size of the local subdomain matrix + the overlap.
- gko::size_type local_size_o = 0

  The size of the local subdomain matrix + the overlap.
- gko::size_type overlap_size = 0

  The size of the overlap between the subdomains.
- gko::size_type num_subdomains = 1

  The number of subdomains used within the solver.
- int my_rank

  The rank of the subdomain.
- int my_local_rank

  The local rank of the subdomain.
- int local_num_procs

  The local number of procs in the subdomain.
- int comm_size

The number of subdomains used within the solver, size of the communicator.

- int num_threads

  The number of threads used within the solver for each subdomain.
- IndexType iter_count

  The iteration count of the solver.
- ValueType tolerance

  The tolerance of the complete solver.
- ValueType local_solver_tolerance

  The tolerance of the local solver in case of an iterative solve.
- IndexType max_iters

  The maximum iteration count of the Schwarz solver.
- IndexType local_max_iters

  The maximum iteration count of the local iterative solver.
- std::string local_precond

  Local preconditioner.
- unsigned int precond_max_block_size

  The maximum block size for the preconditioner.
- ValueType current_residual_norm = -1.0

  The current residual norm of the subdomain.
- ValueType min_residual_norm = -1.0

  The minimum residual norm of the subdomain.
- ValueType constant = 0.0

  Value of constant for event threshold.
- ValueType gamma = 0.0

  Value of gamma for event threshold.
- ValueType horizon = 0.0

  Value of horizon for the event threshold.
- IndexType sent_history = 0

  Value of history at the sender.
- IndexType recv_history = 0

  Value of history at the receiver.
- IndexType comm_start_iters = 0

  Number of iterations to communicate before event comm.
- std::vector< std::tuple< int, int, int, std::string, std::vector< ValueType > > > time_struct

  The struct used to measure the timings of each function within the solver loop.
- std::vector< std::tuple< int, std::vector< std::tuple< int, int > >, std::vector< std::tuple< int, int > >, int, int > > comm_data_struct

  The struct used to measure the timings of each function within the solver loop.
- std::shared_ptr< gko::Array< IndexType > > global_to_local

  The mapping containing the global to local indices.
- std::shared_ptr< gko::Array< IndexType > > local_to_global

  The mapping containing the local to global indices.
- std::shared_ptr< gko::Array< IndexType > > overlap_row

  The overlap row indices.
- std::shared_ptr< gko::Array< IndexType > > first_row

  The starting row of each subdomain in the matrix.
- std::shared_ptr< gko::Array< IndexType > > permutation

  The permutation used for the re-ordering.
- std::shared_ptr< gko::Array< IndexType > > i_permutation

  The inverse permutation used for the re-ordering.

7.10.1 Detailed Description

template<typename ValueType, typename IndexType>
struct schwz::Metadata< ValueType, IndexType >

The solver metadata struct.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

7.10.2 Member Data Documentation

7.10.2.1 local_solver_tolerance

```
template<typename ValueType, typename IndexType>
ValueType schwz::Metadata< ValueType, IndexType >::local_solver_tolerance
```

The tolerance of the local solver in case of an iterative solve.

The residual norm reduction required.

7.10.2.2 tolerance

```
template<typename ValueType, typename IndexType>
ValueType schwz::Metadata< ValueType, IndexType >::tolerance
```

The tolerance of the complete solver.

The residual norm reduction required.

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

- settings.hpp (321b91c)

7.11 MetisError Class Reference

MetisError is thrown when a METIS routine throws a non-zero error code.

```
#include <exception.hpp>
```

Public Member Functions

- MetisError (const std::string &file, int line, const std::string &func, int error_code)
  Initializes a METIS error.

7.11.1 Detailed Description

MetisError is thrown when a METIS routine throws a non-zero error code.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 MetisError()

```
MetisError::MetisError (
            const std::string & file,
            int line,
            const std::string & func,
            int error_code )  [inline]
```

Initializes a METIS error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
func	The name of the METIS routine that failed
error_code	The resulting METIS error code

```
182         : Error(file, line, func + ": " + get_error(error_code))
183     {}
```

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

- exception.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (321b91c)

7.12 schwz::Metadata< ValueType, IndexType >::post_process_data Struct Reference

The struct used for storing data for post-processing.

```
#include <settings.hpp>
```

7.12.1 Detailed Description

template<typename ValueType, typename IndexType>
struct schwz::Metadata< ValueType, IndexType >::post_process_data

The struct used for storing data for post-processing.

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

- settings.hpp (321b91c)

7.13 schwz::SchwarzBase< ValueType, IndexType > Class Template Reference

The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.

`#include <schwarz_base.hpp>`

Public Member Functions

- SchwarzBase (Settings &settings, Metadata< ValueType, IndexType > &metadata)
  The constructor that takes in the user settings and a metadata struct containing the solver metadata.
- void initialize ()
  Initialize the matrix and vectors.
- void run (std::shared_ptr< gko::matrix::Dense< ValueType >> &solution)
  The function that runs the actual solver and obtains the final solution.
- void print_vector (const std::shared_ptr< gko::matrix::Dense< ValueType >> &vector, int subd, std::string name)
  The auxiliary function that prints a passed in vector.
- void print_matrix (const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &matrix, int rank, std::string name)
  The auxiliary function that prints a passed in CSR matrix.

Public Attributes

- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > local_matrix
  The local subdomain matrix.
- std::shared_ptr< gko::matrix::Permutation< IndexType > > local_perm
  The local subdomain permutation matrix/array.
- std::shared_ptr< gko::matrix::Permutation< IndexType > > local_inv_perm
  The local subdomain inverse permutation matrix/array.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > triangular_factor_l
  The local lower triangular factor used for the triangular solves.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > triangular_factor_u
  The local upper triangular factor used for the triangular solves.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > interface_matrix
  The local interface matrix.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > global_matrix

The global matrix.

- std::shared_ptr< gko::matrix::Dense< ValueType > > local_rhs

  The local right hand side.
- std::shared_ptr< gko::matrix::Dense< ValueType > > global_rhs

  The global right hand side.
- std::shared_ptr< gko::matrix::Dense< ValueType > > local_solution

  The local solution vector.
- std::shared_ptr< gko::matrix::Dense< ValueType > > last_solution

  The (local+overlap) solution vector at time of last event of communication The size of this vector is considered global_size to account for overlap.
- std::shared_ptr< gko::matrix::Dense< ValueType > > global_solution

  The global solution vector.
- std::vector< ValueType > local_residual_vector_out

  The global residual vector.
- std::vector< std::vector< ValueType > > global_residual_vector_out

  The local residual vector.

Additional Inherited Members

7.13.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::SchwarzBase< ValueType, IndexType >

The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.

It derives from the Initialization class, the Communication class and the Solve class all of which are templated.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 SchwarzBase()

```
template<typename ValueType , typename IndexType >
schwz::SchwarzBase< ValueType, IndexType >::SchwarzBase (
            Settings & settings,
            Metadata< ValueType, IndexType > & metadata )
```

The constructor that takes in the user settings and a metadata struct containing the solver metadata.

Parameters

settings	The settings struct.
metadata	The metadata struct.

References schwz::Settings::cuda_device_guard, schwz::Settings::executor, schwz::Settings::executor_string, schwz::Metadata< ValueType, IndexType >::local_num_procs, schwz::Metadata< ValueType, IndexType >::mpi↩ _communicator, schwz::Metadata< ValueType, IndexType >::my_local_rank, and schwz::Metadata< ValueType, IndexType >::my_rank.

```
74     : Initialize<ValueType, IndexType>(settings, metadata),
75       settings(settings),
76       metadata(metadata)
77 {
78     using vec_itype = gko::Array<IndexType>;
79     using vec_vecshared = gko::Array<IndexType *>;
80     metadata.my_local_rank =
81         Utils<ValueType, IndexType>::get_local_rank(metadata.mpi_communicator);
82     metadata.local_num_procs = Utils<ValueType, IndexType>::get_local_num_procs(
83         metadata.mpi_communicator);
84     auto my_local_rank = metadata.my_local_rank;
85     if (settings.executor_string == "omp") {
86         settings.executor = gko::OmpExecutor::create();
87         auto exec_info =
88             static_cast<gko::OmpExecutor *>(settings.executor.get())
89                 ->get_exec_info();
90         exec_info->bind_to_core(metadata.my_local_rank);
91
92     } else if (settings.executor_string == "cuda") {
93         int num_devices = 0;
94 #if SCHW_HAVE_CUDA
95         SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
96 #else
97         SCHWARZ_NOT_IMPLEMENTED;
98 #endif
99         Utils<ValueType, IndexType>::assert_correct_cuda_devices(
100             num_devices, metadata.my_rank);
101         settings.executor = gko::CudaExecutor::create(
102             my_local_rank, gko::OmpExecutor::create());
103         auto exec_info = static_cast<gko::OmpExecutor *>(
104                              settings.executor->get_master().get())
105                              ->get_exec_info();
106         exec_info->bind_to_core(my_local_rank);
107         settings.cuda_device_guard =
108             std::make_shared<schwz::device_guard>(my_local_rank);
109
110         std::cout << " Rank " << metadata.my_rank << " with local rank "
111                   << my_local_rank << " has "
112                   << (static_cast<gko::CudaExecutor *>(settings.executor.get()))
113                          ->get_device_id()
114                   << " id of gpu" << std::endl;
115         MPI_Barrier(metadata.mpi_communicator);
116     } else if (settings.executor_string == "reference") {
117         settings.executor = gko::ReferenceExecutor::create();
118         auto exec_info =
119             static_cast<gko::ReferenceExecutor *>(settings.executor.get())
120                 ->get_exec_info();
121         exec_info->bind_to_core(my_local_rank);
122     }
123 }
```

7.13.3 Member Function Documentation

7.13.3.1 print_matrix()

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void schwz::SchwarzBase< ValueType, IndexType >::print_matrix (
```

```
const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & matrix,
int rank,
std::string name )
```

The auxiliary function that prints a passed in CSR matrix.

Parameters

matrix	The matrix to be printed.
subd	The subdomain on which the vector exists.
name	The name of the matrix as a string.

7.13.3.2 print_vector()

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void schwz::SchwarzBase< ValueType, IndexType >::print_vector (
            const std::shared_ptr< gko::matrix::Dense< ValueType >> & vector,
            int subd,
            std::string name )
```

The auxiliary function that prints a passed in vector.

Parameters

vector	The vector to be printed.
subd	The subdomain on which the vector exists.
name	The name of the vector as a string.

7.13.3.3 run()

```
template<typename ValueType , typename IndexType >
void schwz::SchwarzBase< ValueType, IndexType >::run (
            std::shared_ptr< gko::matrix::Dense< ValueType >> & solution )
```

The function that runs the actual solver and obtains the final solution.

Parameters

solution	The solution vector.

References schwz::Settings::debug_print, schwz::Communicate< ValueType, IndexType >::exchange_boundary(), schwz::Settings::executor, schwz::SchwarzBase< ValueType, IndexType >::global_matrix, schwz::Schwarz↩ Base< ValueType, IndexType >::global_rhs, schwz::SchwarzBase< ValueType, IndexType >::global_solution, schwz::SchwarzBase< ValueType, IndexType >::interface_matrix, schwz::SchwarzBase< ValueType, Index↩ Type >::last_solution, schwz::SchwarzBase< ValueType, IndexType >::local_inv_perm, schwz::SchwarzBase< ValueType, IndexType >::local_matrix, schwz::SchwarzBase< ValueType, IndexType >::local_perm, schwz↩ ::SchwarzBase< ValueType, IndexType >::local_rhs, schwz::SchwarzBase< ValueType, IndexType >::local↩ _solution, schwz::Communicate< ValueType, IndexType >::comm_struct::msg_count, schwz::Communicate< ValueType, IndexType >::comm_struct::neighbors_out, schwz::Communicate< ValueType, IndexType >::comm↩ _struct::num_neighbors_out, schwz::Settings::overlap, schwz::Communicate< ValueType, IndexType >::setup↩ _windows(), schwz::Settings::thres_type, schwz::SchwarzBase< ValueType, IndexType >::triangular_factor_l,

schwz::SchwarzBase< ValueType, IndexType >::triangular_factor_u, schwz::Communicate< ValueType, IndexType >::update_boundary(), and schwz::Settings::write_iters_and_residuals.

```
326 {
327     using vec_vtype = gko::matrix::Dense<ValueType>;
328     if (!solution.get()) {
329         solution =
330             vec_vtype::create(settings.executor->get_master(),
331                               gko::dim<2>(this->metadata.global_size, 1));
332     }
333     // The main solution vector
334     std::shared_ptr<vec_vtype> global_solution = vec_vtype::create(
335         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
336
337     auto num_neighbors_out = this->comm_struct.num_neighbors_out;
338     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
339
340     // The last communicated solution vector
341     std::shared_ptr<vec_vtype> last_solution = vec_vtype::create(
342         settings.executor, gko::dim<2>(metadata.global_size, 1));
343
344     // A work vector.
345     std::shared_ptr<vec_vtype> work_vector = vec_vtype::create(
346         settings.executor, gko::dim<2>(2 * this->metadata.local_size_x, 1));
347     // An initial guess.
348     std::shared_ptr<vec_vtype> init_guess = vec_vtype::create(
349         settings.executor, gko::dim<2>(this->metadata.local_size_x, 1));
350     init_guess->copy_from(local_rhs.get());
351
352     // Setup the windows for the onesided communication.
353     this->setup_windows(this->settings, this->metadata, global_solution);
354
355     const auto solver_settings =
356         (Settings::local_solver_settings::direct_solver_cholmod |
357          Settings::local_solver_settings::direct_solver_umfpack |
358          Settings::local_solver_settings::direct_solver_ginkgo |
359          Settings::local_solver_settings::iterative_solver_dealii |
360          Settings::local_solver_settings::iterative_solver_ginkgo) &
361         settings.local_solver;
362
363     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
364               global_residual_norm = 0.0, global_residual_norm0 = -1.0;
365     metadata.iter_count = 0;
366     int num_converged_procs = 0;
367
368     std::ofstream fps;  // file for sending log
369     std::ofstream fpr;  // file for receiving log
370
371     if (settings.debug_print) {
372         // Opening files for event logs
373         char send_name[30], recv_name[30], pe_str[3];
374         sprintf(pe_str, "%d", metadata.my_rank);
375
376         strcpy(send_name, "send");
377         strcat(send_name, pe_str);
378         strcat(send_name, ".txt");
379
380         strcpy(recv_name, "recv");
381         strcat(recv_name, pe_str);
382         strcat(recv_name, ".txt");
383
384         fps.open(send_name);
385         fpr.open(recv_name);
386     }
387
388     if (metadata.my_rank == 0) {
389         std::cout << "Send history - " << metadata.sent_history << ", Recv history - "
390                   << metadata.recv_history << std::endl;
391         std::cout << "Thres type - " << settings.thres_type << std::endl;
392         std::cout << "Overlap - " << settings.overlap << std::endl;
393     }
394
395     auto start_time = std::chrono::steady_clock::now();
396
397     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
398         // Exchange the boundary values. The communication part.
399         MEASURE_ELAPSED_FUNC_TIME(
400             this->exchange_boundary(settings, metadata, global_solution,
401                                     last_solution, fps, fpr),
402             0, metadata.my_rank, boundary_exchange, metadata.iter_count);
403
404         // Update the boundary and interior values after the exchanging from
405         // other processes.
406         MEASURE_ELAPSED_FUNC_TIME(
407             this->update_boundary(settings, metadata, this->
```

```
    local_solution,
408                                 this->local_rhs, global_solution,
409                                 this->interface_matrix),
410             1, metadata.my_rank, boundary_update, metadata.iter_count);
411
412         if (settings.debug_print) {
413             //fps << metadata.iter_count << ", " << local_residual_norm
414             //    << std::endl;
415         }
416
417         // Check for the convergence of the solver.
418         // num_converged_procs = 0;
419         MEASURE_ELAPSED_FUNC_TIME(
420             (Solve<ValueType, IndexType>::check_convergence(
421                 settings, metadata, this->comm_struct, this->convergence_vector,
422                 global_solution, this->local_solution, this->
    local_matrix,
423                 work_vector, local_residual_norm, local_residual_norm0,
424                 global_residual_norm, global_residual_norm0,
425                 num_converged_procs)),
426             2, metadata.my_rank, convergence_check, metadata.iter_count);
427
428         // break if the solution diverges.
429         if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
430             std::cout << " Rank " << metadata.my_rank << " diverged in "
431                       << metadata.iter_count << " iters " << std::endl;
432             std::exit(-1);
433         }
434
435         // break if all processes detect that all other processes have
436         // converged otherwise continue iterations.
437         if (num_converged_procs == metadata.num_subdomains) {
438             break;
439         } else {
440             MEASURE_ELAPSED_FUNC_TIME(
441                 (Solve<ValueType, IndexType>::local_solve(
442                     settings, metadata, this->local_matrix,
443                     this->triangular_factor_l, this->
    triangular_factor_u,
444                     this->local_perm, this->local_inv_perm, work_vector,
445                     init_guess, this->local_solution)),
446                 3, metadata.my_rank, local_solve, metadata.iter_count);
447             // Gather the local vector into the locally global vector for
448             // communication.
449             MEASURE_ELAPSED_FUNC_TIME(
450                 (Communicate<ValueType, IndexType>::local_to_global_vector
    (
451                     settings, metadata, this->local_solution, global_solution)),
452                 4, metadata.my_rank, expand_local_vec, metadata.iter_count);
453         }
454     }
455
456     MPI_Barrier(MPI_COMM_WORLD);
457     auto elapsed_time = std::chrono::duration<ValueType>(
458         std::chrono::steady_clock::now() - start_time);
459
460     if (settings.debug_print) {
461         // Closing event log files
462         fps.close();
463         fpr.close();
464     }
465
466     // adding 1 to include the 0-th iteration
467     metadata.iter_count = metadata.iter_count + 1;
468
469     // number of messages a PE would send without event-based
470     int noevent_msg_count = metadata.iter_count * num_neighbors_out;
471
472     int total_events = 0;
473
474     // Printing msg count
475     for (int k = 0; k < num_neighbors_out; k++) {
476         std::cout << " Rank: " << metadata.my_rank << " to " << neighbors_out[k]
477                   << " : " << this->comm_struct.msg_count->get_data()[k];
478         total_events += this->comm_struct.msg_count->get_data()[k];
479     }
480     std::cout << std::endl;
481
482     // Total no of messages in all PEs
483     MPI_Allreduce(MPI_IN_PLACE, &total_events, 1, MPI_INT, MPI_SUM,
484                   MPI_COMM_WORLD);
485     MPI_Allreduce(MPI_IN_PLACE, &noevent_msg_count, 1, MPI_INT, MPI_SUM,
486                   MPI_COMM_WORLD);
487
488     if (metadata.my_rank == 0) {
489         std::cout << "Total number of events - " << total_events << std::endl;
490         std::cout << "Total number of msgs without event - "
```

```
491                   << noevent_msg_count << std::endl;
492     }
493
494     std::cout << " Rank " << metadata.my_rank << " converged in "
495               << metadata.iter_count << " iters " << std::endl;
496     ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
497               residual_norm = -1.0;
498
499
500     // Write the residuals and iterations to files
501     if (settings.write_iters_and_residuals &&
502         solver_settings ==
503             Settings::local_solver_settings::iterative_solver_ginkgo) {
504         std::string rank_string = std::to_string(metadata.my_rank);
505         if (metadata.my_rank < 10) {
506             rank_string = "0" + std::to_string(metadata.my_rank);
507         }
508         std::string filename = "iter_res_" + rank_string + ".csv";
509         write_iters_and_residuals(
510             metadata.num_subdomains, metadata.my_rank,
511             metadata.post_process_data.local_residual_vector_out.size(),
512             metadata.post_process_data.local_residual_vector_out,
513             metadata.post_process_data.local_converged_iter_count,
514             metadata.post_process_data.local_converged_resnorm, filename);
515     }
516
517     // Compute the final residual norm. Also gathers the solution from all
518     // subdomains.
519     Solve<ValueType, IndexType>::compute_residual_norm(
520         settings, metadata, global_matrix, global_rhs, global_solution,
521         mat_norm, rhs_norm, sol_norm, residual_norm);
522     gather_comm_data<ValueType, IndexType>(
523         metadata.num_subdomains, this->comm_struct, metadata.comm_data_struct);
524     // clang-format off
525     if (metadata.my_rank == 0)
526       {
527         std::cout
528               << " residual norm " << residual_norm << "\n"
529               << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
530               << " Time taken for solve " << elapsed_time.count()
531               << std::endl;
532         if (num_converged_procs < metadata.num_subdomains)
533           {
534             std::cout << " Did not converge in " << metadata.iter_count
535                       << " iterations."
536                       << std::endl;
537           }
538       }
539
540     // clang-format on
541     if (metadata.my_rank == 0) {
542         solution->copy_from(global_solution.get());
543     }
544     // Communicate<ValueType, IndexType>::clear(settings);
545 }
```

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

- schwarz_base.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz_base.cpp (321b91c)

7.14 schwz::Settings Struct Reference

The struct that contains the solver settings and the parameters to be set by the user.

`#include <settings.hpp>`

Classes

- struct comm_settings

  The settings for the various available communication paradigms.
- struct convergence_settings

  The various convergence settings available.

Public Types

- enum partition_settings

  The partition algorithm to be used for partitioning the matrix.
- enum local_solver_settings

  The local solver algorithm for the local subdomain solves.

Public Attributes

- std::string executor_string

  The string that contains the ginkgo executor paradigm.
- std::shared_ptr< gko::Executor > executor = gko::ReferenceExecutor::create()

  The ginkgo executor the code is to be executed on.
- std::shared_ptr< device_guard > cuda_device_guard

  The ginkgo executor the code is to be executed on.
- gko::int32 overlap = 2

  The overlap between the subdomains.
- std::string matrix_filename = "null"

  The string that contains the matrix file name to read from .
- bool explicit_laplacian = true

  Flag if the laplacian matrix should be generated within the library.
- std::string rhs_type = "ones"

  Flag to enable a random rhs.
- std::string thres_type = "cgammak"

  Flag to choose thres type.
- bool print_matrices = false

  Flag to enable printing of matrices.
- bool debug_print = false

  Flag to enable some debug printing.
- bool non_symmetric_matrix = false

  Is the matrix non-symmetric ? , Use GMRES for local solves.
- unsigned int restart_iter = 1u

  The restart iter for the GMRES solver.
- bool naturally_ordered_factor = false

  Disables the re-ordering of the matrix before computing the triangular factors during the CHOLMOD factorization.
- std::string metis_objtype

  This setting defines the objective type for the metis partitioning.
- bool use_precond = false

  Enable the block jacobi local preconditioner for the local solver.
- bool write_debug_out = false

  Enable the writing of debug out to file.
- bool write_iters_and_residuals = false

  Enable writing the iters and residuals to a file.
- bool write_perm_data = false

  Enable the local permutations from CHOLMOD to a file.
- int shifted_iter = 1

  Iteration shift for node local communication.
- std::string factorization = "cholmod"

  The factorization for the local direct solver.
- std::string reorder

  The reordering for the local solve.

7.14.1 Detailed Description

The struct that contains the solver settings and the parameters to be set by the user.

settings

7.14.2 Member Data Documentation

7.14.2.1 explicit_laplacian

```
bool schwz::Settings::explicit_laplacian = true
```

Flag if the laplacian matrix should be generated within the library.

If false, an external matrix and rhs needs to be provided

Referenced by schwz::SchwarzBase< ValueType, IndexType >::initialize().

7.14.2.2 naturally_ordered_factor

```
bool schwz::Settings::naturally_ordered_factor = false
```

Disables the re-ordering of the matrix before computing the triangular factors during the CHOLMOD factorization.

Note

This is mainly to allow compatibility with GPU solution.

7.14.2.3 thres_type

```
std::string schwz::Settings::thres_type = "cgammak"
```

Flag to choose thres type.

Choices are "cgammak" or "slope"

Referenced by schwz::SchwarzBase< ValueType, IndexType >::run(), and schwz::SolverRAS< ValueType, IndexType >::setup_windows().

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

- settings.hpp (321b91c)

7.15 schwz::Solve< ValueType, IndexType > Class Template Reference

The Solver class the provides the solver and the convergence checking methods.

`#include <solve.hpp>`

Additional Inherited Members

7.15.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::Solve< ValueType, IndexType >

The Solver class the provides the solver and the convergence checking methods.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

Solve

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

- solve.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/solve.cpp (321b91c)

7.16 schwz::SolverRAS< ValueType, IndexType > Class Template Reference

An implementation of the solver interface using the RAS solver.

`#include <restricted_schwarz.hpp>`

Public Member Functions

- SolverRAS (Settings &settings, Metadata< ValueType, IndexType > &metadata)

  The constructor that takes in the user settings and a metadata struct containing the solver metadata.
- void setup_local_matrices (Settings &settings, Metadata< ValueType, IndexType > &metadata, std::vector< unsigned int > &partition_indices, std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &global_↩ matrix, std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &local_matrix, std::shared_ptr< gko↩ ::matrix::Csr< ValueType, IndexType >> &interface_matrix) override

  Sets up the local and the interface matrices from the global matrix and the partition indices.
- void setup_comm_buffers () override

  Sets up the communication buffers needed for the boundary exchange.
- void setup_windows (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std↩ ::shared_ptr< gko::matrix::Dense< ValueType >> &main_buffer) override

Sets up the windows needed for the asynchronous communication.

- void exchange_boundary (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &solution, std::shared_ptr< gko::matrix::↩ Dense< ValueType >> &last_solution, std::ofstream &fps, std::ofstream &fpr) override

  Exchanges the elements of the solution vector.

- void update_boundary (const Settings &settings, const Metadata< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &local_solution, const std::shared_ptr< gko::matrix↩ ::Dense< ValueType >> &local_rhs, const std::shared_ptr< gko::matrix::Dense< ValueType >> &global↩ _solution, const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &interface_matrix) override

  Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.

Additional Inherited Members

7.16.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::SolverRAS< ValueType, IndexType >

An implementation of the solver interface using the RAS solver.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 SolverRAS()

```
template<typename ValueType , typename IndexType >
schwz::SolverRAS< ValueType, IndexType >::SolverRAS (
            Settings & settings,
            Metadata< ValueType, IndexType > & metadata )
```

The constructor that takes in the user settings and a metadata struct containing the solver metadata.

Parameters

settings	The settings struct.
metadata	The metadata struct.
data	The additional data struct.

```
48     : SchwarzBase<ValueType, IndexType>(settings, metadata)
49 {}
```

7.16.3 Member Function Documentation

7.16.3.1 exchange_boundary()

```
template<typename ValueType , typename IndexType >
void schwz::SolverRAS< ValueType, IndexType >::exchange_boundary (
            const Settings & settings,
            const Metadata< ValueType, IndexType > & metadata,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & solution,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & last_solution,
            std::ofstream & fps,
            std::ofstream & fpr )  [override], [virtual]
```

Exchanges the elements of the solution vector.

Parameters

settings	The settings struct.
metadata	The metadata struct.
global_solution	The solution vector being exchanged between the subdomains.

Implements schwz::Communicate< ValueType, IndexType >.

References schwz::Settings::comm_settings::enable_onesided, schwz::SchwarzBase< ValueType, IndexType >::global_solution, and schwz::SchwarzBase< ValueType, IndexType >::last_solution.

```
1154 {
1155     if (settings.comm_settings.enable_onesided) {
1156         exchange_boundary_onesided<ValueType, IndexType>(
1157             settings, metadata, this->comm_struct, global_solution,
1158             last_solution, fps, fpr);
1159     } else {
1160         exchange_boundary_twosided<ValueType, IndexType>(
1161             settings, metadata, this->comm_struct, global_solution);
1162     }
1163 }
```

7.16.3.2 setup_local_matrices()

```
template<typename ValueType , typename IndexType >
void schwz::SolverRAS< ValueType, IndexType >::setup_local_matrices (
            Settings & settings,
            Metadata< ValueType, IndexType > & metadata,
            std::vector< unsigned int > & partition_indices,
            std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_matrix,
            std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & local_matrix,
            std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_matrix )
[override], [virtual]
```

Sets up the local and the interface matrices from the global matrix and the partition indices.

Parameters

settings	The settings struct.
metadata	The metadata struct.
partition_indices	The array containing the partition indices.
global_matrix	The global system matrix.
local_matrix	The local system matrix.
interface_matrix	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.
local_perm	The local permutation, obtained through RCM or METIS.

Implements schwz::Initialize< ValueType, IndexType >.

References schwz::Metadata< ValueType, IndexType >::comm_size, schwz::Settings::executor, schwz::Metadata< ValueType, IndexType >::first_row, schwz::SchwarzBase< ValueType, IndexType >::global_matrix, schwz::Metadata< ValueType, IndexType >::global_size, schwz::Metadata< ValueType, IndexType >::global_to_local, schwz::Metadata< ValueType, IndexType >::i_permutation, schwz::SchwarzBase< ValueType, IndexType >::interface_matrix, schwz::SchwarzBase< ValueType, IndexType >::local_matrix, schwz::Metadata< ValueType, IndexType >::local_size, schwz::Metadata< ValueType, IndexType >::local_size_o, schwz::Metadata< ValueType, IndexType >::local_size_x, schwz::Metadata< ValueType, IndexType >::local_to_global, schwz::Metadata< ValueType, IndexType >::my_rank, schwz::Metadata< ValueType, IndexType >::num_subdomains, schwz::Settings::overlap, schwz::Metadata< ValueType, IndexType >::overlap_row, schwz::Metadata< ValueType, IndexType >::overlap_size, and schwz::Metadata< ValueType, IndexType >::permutation.

```
59 {
60     using mtx = gko::matrix::Csr<ValueType, IndexType>;
61     using vec_itype = gko::Array<IndexType>;
62     using perm_type = gko::matrix::Permutation<IndexType>;
63     using arr = gko::Array<IndexType>;
64     auto my_rank = metadata.my_rank;
65     auto comm_size = metadata.comm_size;
66     auto num_subdomains = metadata.num_subdomains;
67     auto global_size = metadata.global_size;
68     auto mpi_itype = boost::mpi::get_mpi_datatype(*partition_indices.data());
69
70     MPI_Bcast(partition_indices.data(), global_size, mpi_itype, 0,
71               MPI_COMM_WORLD);
72
73     std::vector<IndexType> local_p_size(num_subdomains);
74     auto global_to_local = metadata.global_to_local->get_data();
75     auto local_to_global = metadata.local_to_global->get_data();
76
77     auto first_row = metadata.first_row->get_data();
78     auto permutation = metadata.permutation->get_data();
79     auto i_permutation = metadata.i_permutation->get_data();
80
81     auto nb = (global_size + num_subdomains - 1) /
    num_subdomains;
82     auto partition_settings =
83         (Settings::partition_settings::partition_zoltan |
84          Settings::partition_settings::partition_metis |
85          Settings::partition_settings::partition_regular |
86          Settings::partition_settings::partition_regular2d |
87          Settings::partition_settings::partition_custom) &
88         settings.partition;
89
90     IndexType *gmat_row_ptrs = global_matrix->get_row_ptrs();
91     IndexType *gmat_col_idxs = global_matrix->get_col_idxs();
92     ValueType *gmat_values = global_matrix->get_values();
93
94     // default local p size set for 1 subdomain.
95     first_row[0] = 0;
96     for (auto p = 0; p < num_subdomains; ++p) {
97         local_p_size[p] = std::min(global_size - first_row[p], nb);
98         first_row[p + 1] = first_row[p] + local_p_size[p];
99     }
100
101     if (partition_settings == Settings::partition_settings::partition_metis ||
102         partition_settings ==
103             Settings::partition_settings::partition_regular2d) {
104         if (num_subdomains > 1) {
```

```
105             for (auto p = 0; p < num_subdomains; p++) {
106                 local_p_size[p] = 0;
107             }
108             for (auto i = 0; i < global_size; i++) {
109                 local_p_size[partition_indices[i]]++;
110             }
111             first_row[0] = 0;
112             for (auto p = 0; p < num_subdomains; ++p) {
113                 first_row[p + 1] = first_row[p] + local_p_size[p];
114             }
115             // permutation
116             for (auto i = 0; i < global_size; i++) {
117                 permutation[first_row[partition_indices[i]]] = i;
118                 first_row[partition_indices[i]]++;
119             }
120             for (auto p = num_subdomains; p > 0; p--) {
121                 first_row[p] = first_row[p - 1];
122             }
123             first_row[0] = 0;
124 
125             // iperm
126             for (auto i = 0; i < global_size; i++) {
127                 i_permutation[permutation[i]] = i;
128             }
129         }
130 
131         auto gmat_temp = mtx::create(settings.executor->get_master(),
132                                      global_matrix->get_size(),
133                                      global_matrix->get_num_stored_elements());
134 
135         auto nnz = 0;
136         gmat_temp->get_row_ptrs()[0] = 0;
137         for (auto row = 0; row < metadata.global_size; ++row) {
138             for (auto col = gmat_row_ptrs[permutation[row]];
139                  col < gmat_row_ptrs[permutation[row] + 1]; ++col) {
140                 gmat_temp->get_col_idxs()[nnz] =
141                     i_permutation[gmat_col_idxs[col]];
142                 gmat_temp->get_values()[nnz] = gmat_values[col];
143                 nnz++;
144             }
145             gmat_temp->get_row_ptrs()[row + 1] = nnz;
146         }
147         global_matrix->copy_from(gmat_temp.get());
148     }
149     for (auto i = 0; i < global_size; i++) {
150         global_to_local[i] = 0;
151         local_to_global[i] = 0;
152     }
153     auto num = 0;
154     for (auto i = first_row[my_rank]; i < first_row[
    my_rank + 1]; i++) {
155         global_to_local[i] = 1 + num;
156         local_to_global[num] = i;
157         num++;
158     }
159 
160     IndexType old = 0;
161     for (auto k = 1; k < settings.overlap; k++) {
162         auto now = num;
163         for (auto i = old; i < now; i++) {
164             for (auto j = gmat_row_ptrs[local_to_global[i]];
165                  j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
166                 if (global_to_local[gmat_col_idxs[j]] == 0) {
167                     local_to_global[num] = gmat_col_idxs[j];
168                     global_to_local[gmat_col_idxs[j]] = 1 + num;
169                     num++;
170                 }
171             }
172         }
173         old = now;
174     }
175     metadata.local_size = local_p_size[my_rank];
176     metadata.local_size_x = num;
177     metadata.local_size_o = global_size;
178     auto local_size = metadata.local_size;
179     auto local_size_x = metadata.local_size_x;
180 
181     metadata.overlap_size = num - metadata.local_size;
182     metadata.overlap_row = std::shared_ptr<vec_itype>(
183         new vec_itype(gko::Array<IndexType>::view(
184             settings.executor, metadata.overlap_size,
185             &(metadata.local_to_global->get_data()[metadata.local_size]))),
186         std::default_delete<vec_itype>());
187 
188     auto nnz_local = 0;
189     auto nnz_interface = 0;
190 
```

```
191     for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
192         for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; j++) {
193             if (global_to_local[gmat_col_idxs[j]] != 0) {
194                 nnz_local++;
195             } else {
196                 std::cout << " debug: invalid edge?" << std::endl;
197             }
198         }
199     }
200     auto temp = 0;
201     for (auto k = 0; k < metadata.overlap_size; k++) {
202         temp = metadata.overlap_row->get_data()[k];
203         for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1]; j++) {
204             if (global_to_local[gmat_col_idxs[j]] != 0) {
205                 nnz_local++;
206             } else {
207                 nnz_interface++;
208             }
209         }
210     }
211
212     std::shared_ptr<mtx> local_matrix_compute;
213     local_matrix_compute = mtx::create(settings.executor->get_master(),
214                                        gko::dim<2>(local_size_x), nnz_local);
215     IndexType *lmat_row_ptrs = local_matrix_compute->get_row_ptrs();
216     IndexType *lmat_col_idxs = local_matrix_compute->get_col_idxs();
217     ValueType *lmat_values = local_matrix_compute->get_values();
218
219     std::shared_ptr<mtx> interface_matrix_compute;
220     if (nnz_interface > 0) {
221         interface_matrix_compute =
222             mtx::create(settings.executor->get_master(),
223                         gko::dim<2>(local_size_x), nnz_interface);
224     } else {
225         interface_matrix_compute = mtx::create(settings.executor->get_master());
226     }
227
228     IndexType *imat_row_ptrs = interface_matrix_compute->get_row_ptrs();
229     IndexType *imat_col_idxs = interface_matrix_compute->get_col_idxs();
230     ValueType *imat_values = interface_matrix_compute->get_values();
231
232     num = 0;
233     nnz_local = 0;
234     auto nnz_interface_temp = 0;
235     lmat_row_ptrs[0] = nnz_local;
236     if (nnz_interface > 0) {
237         imat_row_ptrs[0] = nnz_interface_temp;
238     }
239
240     // Local interior matrix
241     for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
242         for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; ++j) {
243             if (global_to_local[gmat_col_idxs[j]] != 0) {
244                 lmat_col_idxs[nnz_local] =
245                     global_to_local[gmat_col_idxs[j]] - 1;
246                 lmat_values[nnz_local] = gmat_values[j];
247                 nnz_local++;
248             }
249         }
250         if (nnz_interface > 0) {
251             imat_row_ptrs[num + 1] = nnz_interface_temp;
252         }
253         lmat_row_ptrs[num + 1] = nnz_local;
254         num++;
255     }
256
257     // Interface matrix
258     if (nnz_interface > 0) {
259         nnz_interface = 0;
260         for (auto k = 0; k < metadata.overlap_size; k++) {
261             temp = metadata.overlap_row->get_data()[k];
262             for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1];
263                  j++) {
264                 if (global_to_local[gmat_col_idxs[j]] != 0) {
265                     lmat_col_idxs[nnz_local] =
266                         global_to_local[gmat_col_idxs[j]] - 1;
267                     lmat_values[nnz_local] = gmat_values[j];
268                     nnz_local++;
269                 } else {
270                     imat_col_idxs[nnz_interface] = gmat_col_idxs[j];
271                     imat_values[nnz_interface] = gmat_values[j];
272                     nnz_interface++;
273                 }
274             }
275             lmat_row_ptrs[num + 1] = nnz_local;
276             imat_row_ptrs[num + 1] = nnz_interface;
277             num++;
```

```
278         }
279     }
280     auto now = num;
281     for (auto i = old; i < now; i++) {
282         for (auto j = gmat_row_ptrs[local_to_global[i]];
283              j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
284             if (global_to_local[gmat_col_idxs[j]] == 0) {
285                 local_to_global[num] = gmat_col_idxs[j];
286                 global_to_local[gmat_col_idxs[j]] = 1 + num;
287                 num++;
288             }
289         }
290     }
291
292     local_matrix = mtx::create(settings.executor);
293     local_matrix->copy_from(gko::lend(local_matrix_compute));
294     interface_matrix = mtx::create(settings.executor);
295     interface_matrix->copy_from(gko::lend(interface_matrix_compute));
296
297     local_matrix->sort_by_column_index();
298     interface_matrix->sort_by_column_index();
299 }
```

7.16.3.3 setup_windows()

```
template<typename ValueType , typename IndexType >
void schwz::SolverRAS< ValueType, IndexType >::setup_windows (
             const Settings & settings,
             const Metadata< ValueType, IndexType > & metadata,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & main_buffer )  [override],
[virtual]
```

Sets up the windows needed for the asynchronous communication.

Parameters

settings	The settings struct.
metadata	The metadata struct.
main_buffer	The main buffer being exchanged between the subdomains.

Implements schwz::Communicate< ValueType, IndexType >.

References schwz::Metadata< ValueType, IndexType >::comm_start_iters, schwz::Metadata< ValueType, Index↩ Type >::constant, schwz::Communicate< ValueType, IndexType >::comm_struct::curr_recv_avg, schwz::↩ Communicate< ValueType, IndexType >::comm_struct::curr_send_avg, schwz::Settings::debug_print, schwz↩ ::Settings::comm_settings::enable_flush_all, schwz::Settings::comm_settings::enable_flush_local, schwz::↩ Settings::comm_settings::enable_get, schwz::Settings::comm_settings::enable_lock_all, schwz::Settings::comm↩ _settings::enable_one_by_one, schwz::Settings::comm_settings::enable_onesided, schwz::Settings::comm↩ _settings::enable_overlap, schwz::Settings::comm_settings::enable_put, schwz::Settings::executor, schwz::↩ Communicate< ValueType, IndexType >::comm_struct::extra_buffer, schwz::Metadata< ValueType, Index↩ Type >::gamma, schwz::Communicate< ValueType, IndexType >::comm_struct::get_displacements, schwz::↩ Communicate< ValueType, IndexType >::comm_struct::get_request, schwz::Communicate< ValueType, Index↩ Type >::comm_struct::global_get, schwz::Communicate< ValueType, IndexType >::comm_struct::global_put, schwz::SchwarzBase< ValueType, IndexType >::global_solution, schwz::Metadata< ValueType, IndexType >↩ ::horizon, schwz::Communicate< ValueType, IndexType >::comm_struct::is_local_neighbor, schwz::Metadata< ValueType, IndexType >::iter_count, schwz::Communicate< ValueType, IndexType >::comm_struct::last_recv↩ _avg, schwz::Communicate< ValueType, IndexType >::comm_struct::last_recv_bdy, schwz::Communicate< ValueType, IndexType >::comm_struct::last_recv_iter, schwz::Communicate< ValueType, IndexType >::comm_↩ struct::last_recv_slopes, schwz::Communicate< ValueType, IndexType >::comm_struct::last_send_avg, schwz::↩

Communicate< ValueType, IndexType >::comm_struct::last_sent_iter, schwz::Communicate< ValueType, IndexType >::comm_struct::last_sent_slopes_avg, schwz::SchwarzBase< ValueType, IndexType >::last_solution, schwz::Communicate< ValueType, IndexType >::comm_struct::local_get, schwz::Communicate< ValueType, IndexType >::comm_struct::local_neighbors_in, schwz::Communicate< ValueType, IndexType >::comm_struct::local_neighbors_out, schwz::Communicate< ValueType, IndexType >::comm_struct::local_num_neighbors_in, schwz::Communicate< ValueType, IndexType >::comm_struct::local_num_neighbors_out, schwz::Communicate< ValueType, IndexType >::comm_struct::local_put, schwz::Metadata< ValueType, IndexType >::local_size_o, schwz::Communicate< ValueType, IndexType >::comm_struct::msg_count, schwz::Communicate< ValueType, IndexType >::comm_struct::neighbors_in, schwz::Communicate< ValueType, IndexType >::comm_struct::neighbors_out, schwz::Communicate< ValueType, IndexType >::comm_struct::num_neighbors_in, schwz::Communicate< ValueType, IndexType >::comm_struct::num_neighbors_out, schwz::Communicate< ValueType, IndexType >::comm_struct::num_recv, schwz::Communicate< ValueType, IndexType >::comm_struct::num_send, schwz::Metadata< ValueType, IndexType >::num_subdomains, schwz::Communicate< ValueType, IndexType >::comm_struct::put_displacements, schwz::Communicate< ValueType, IndexType >::comm_struct::put_request, schwz::Communicate< ValueType, IndexType >::comm_struct::recv_buffer, schwz::Metadata< ValueType, IndexType >::recv_history, schwz::Communicate< ValueType, IndexType >::comm_struct::remote_get, schwz::Communicate< ValueType, IndexType >::comm_struct::remote_put, schwz::Communicate< ValueType, IndexType >::comm_struct::send_buffer, schwz::Metadata< ValueType, IndexType >::sent_history, schwz::Communicate< ValueType, IndexType >::comm_struct::thres, schwz::Settings::thres_type, schwz::Communicate< ValueType, IndexType >::comm_struct::window_recv_buffer, schwz::Communicate< ValueType, IndexType >::comm_struct::window_send_buffer, and schwz::Communicate< ValueType, IndexType >::comm_struct::window_x.

```
604 {
605     using vec_itype = gko::Array<IndexType>;
606     using vec_vtype = gko::matrix::Dense<ValueType>;
607     auto num_subdomains = metadata.num_subdomains;
608     auto local_size_o = metadata.local_size_o;
609     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
610     auto global_get = this->comm_struct.global_get->get_data();
611     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
612     auto global_put = this->comm_struct.global_put->get_data();
613 
614     // set displacement for the MPI buffer
615     auto get_displacements = this->comm_struct.get_displacements->get_data();
616     auto put_displacements = this->comm_struct.put_displacements->get_data();
617     {
618         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
619         tmp_num_comm_elems[0] = 0;
620         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
621             if ((global_get[j])[0] > 0) {
622                 int p = neighbors_in[j];
623                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
624             }
625         }
626         for (auto j = 0; j < num_subdomains; j++) {
627             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
628         }
629 
630         auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
631         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
632                      1, mpi_itype, MPI_COMM_WORLD);
633     }
634 
635     {
636         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
637         tmp_num_comm_elems[0] = 0;
638         for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
639             if ((global_put[j])[0] > 0) {
640                 int p = neighbors_out[j];
641                 tmp_num_comm_elems[p + 1] = (global_put[j])[0];
642             }
643         }
644         for (auto j = 0; j < num_subdomains; j++) {
645             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
646         }
647 
648         auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
649         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
650                      1, mpi_itype, MPI_COMM_WORLD);
651     }
652 
653     // setup windows
654     if (settings.comm_settings.enable_onesided) {
655         // Onesided
656         MPI_Win_create(main_buffer->get_values(),
657                        main_buffer->get_size()[0] * sizeof(ValueType),
```

```
658                         sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
659                         &(this->comm_struct.window_x));
660     }
661
662     if (settings.comm_settings.enable_onesided) {
663         // MPI_Alloc_mem ? Custom allocator ?  TODO
664         MPI_Win_create(this->local_residual_vector->get_values(),
665                         (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
666                         MPI_INFO_NULL, MPI_COMM_WORLD,
667                         &(this->window_residual_vector));
668         std::vector<IndexType> zero_vec(num_subdomains, 0);
669         gko::Array<IndexType> temp_array{settings.executor->get_master(),
670                                          zero_vec.begin(), zero_vec.end()};
671         this->convergence_vector = std::shared_ptr<vec_itype>(
672             new vec_itype(settings.executor->get_master(), temp_array),
673             std::default_delete<vec_itype>());
674         this->convergence_sent = std::shared_ptr<vec_itype>(
675             new vec_itype(settings.executor->get_master(), num_subdomains),
676             std::default_delete<vec_itype>());
677         this->convergence_local = std::shared_ptr<vec_itype>(
678             new vec_itype(settings.executor->get_master(), num_subdomains),
679             std::default_delete<vec_itype>());
680         MPI_Win_create(this->convergence_vector->get_data(),
681                         (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
682                         MPI_INFO_NULL, MPI_COMM_WORLD,
683                         &(this->window_convergence));
684     }
685
686     if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
687         // Lock all windows.
688         if (settings.comm_settings.enable_get &&
689             settings.comm_settings.enable_lock_all) {
690             MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
691         }
692         if (settings.comm_settings.enable_put &&
693             settings.comm_settings.enable_lock_all) {
694             MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
695         }
696         if (settings.comm_settings.enable_one_by_one &&
697             settings.comm_settings.enable_lock_all) {
698             MPI_Win_lock_all(0, this->comm_struct.window_x);
699         }
700         MPI_Win_lock_all(0, this->window_residual_vector);
701         MPI_Win_lock_all(0, this->window_convergence);
702     }
703 }
```

7.16.3.4 update_boundary()

```
template<typename ValueType , typename IndexType >
void schwz::SolverRAS< ValueType, IndexType >::update_boundary (
             const Settings & settings,
             const Metadata< ValueType, IndexType > & metadata,
             std::shared_ptr< gko::matrix::Dense< ValueType >> & local_solution,
             const std::shared_ptr< gko::matrix::Dense< ValueType >> & local_rhs,
             const std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution,
             const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_↩
matrix )  [override], [virtual]
```

Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.

Parameters

settings	The settings struct.
metadata	The metadata struct.
local_solution	The local solution vector in the subdomain.
local_rhs	The local right hand side vector in the subdomain.
global_solution	The workspace solution vector.
global_old_solution	The global solution vector of the previous iteration.
interface_matrix	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.

Implements schwz::Communicate< ValueType, IndexType >.

References schwz::Settings::executor, schwz::SchwarzBase< ValueType, IndexType >::global_solution, schwz::SchwarzBase< ValueType, IndexType >::interface_matrix, schwz::SchwarzBase< ValueType, IndexType >::local_rhs, schwz::Metadata< ValueType, IndexType >::local_size_x, schwz::SchwarzBase< ValueType, IndexType >::local_solution, schwz::Metadata< ValueType, IndexType >::num_subdomains, and schwz::Settings::overlap.

```
1174 {
1175     using vec_vtype = gko::matrix::Dense<ValueType>;
1176     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
1177         {1.0}, settings.executor);
1178     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
1179         {-1.0}, settings.executor);
1180     auto local_size_x = metadata.local_size_x;
1181     local_solution->copy_from(local_rhs.get());
1182     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
1183         auto temp_solution = vec_vtype::create(
1184             settings.executor, local_solution->get_size(),
1185             gko::Array<ValueType>::view(settings.executor,
1186                                         local_solution->get_size()[0],
1187                                         global_solution->get_values()),
1188             1);
1189         interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
1190                                 local_solution.get());
1191     }
1192 }
```

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

- restricted_schwarz.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/restricted_schwarz.cpp (321b91c)

7.17 UmfpackError Class Reference

UmfpackError is thrown when a METIS routine throws a non-zero error code.

`#include <exception.hpp>`

Public Member Functions

- UmfpackError (const std::string &file, int line, const std::string &func, int error_code)

  Initializes a METIS error.

7.17.1 Detailed Description

UmfpackError is thrown when a METIS routine throws a non-zero error code.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 UmfpackError()

```
UmfpackError::UmfpackError (
            const std::string & file,
            int line,
            const std::string & func,
            int error_code )  [inline]
```

Initializes a METIS error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
func	The name of the METIS routine that failed
error_code	The resulting METIS error code

```
205         : Error(file, line, func + ": " + get_error(error_code))
206     {}
```

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

- exception.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (321b91c)

7.18 schwz::Utils< ValueType, IndexType > Struct Template Reference

The utilities class which provides some checks and basic utilities.

`#include <utils.hpp>`

7.18.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
struct schwz::Utils< ValueType, IndexType >

The utilities class which provides some checks and basic utilities.

Template Parameters

ValueType	The type of the floating point values.
IndexType	The type of the index type values.

Utils

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

- utils.hpp (321b91c)
- /home/runner/work/schwarz-lib/schwarz-lib/source/utils.cpp (321b91c)

Index