schwz

Generated automatically from fix-twosided

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.

The flag `-DSCHWARZ_BUILD_BENCHMARKING` (default `ON`) enables the examples and benchmarking snippets.

If `schwarz-lib` has been built with `deal.ii`, then the `deal.ii` examples, `ex_6` and `ex_9` are also built, else only the `bench_ras` example is built. The following command line options are available for this example. This is setup using `gflags`.

The executable is run in the following fashion:

```
[MPI_COMMAND] [MPI_OPTIONS] PATH_TO_EXECUTABLE [FLAGS]
```

Where `FLAGS` are the options below with the template `flag_name [type][default_value]`. For example, to set the number of iterations of the RAS solver to 100 one would add `--num_iters=100` to the executable command above.

Generic settings

- `executor` [std::string][reference] : The executor used to run the solver, one of `reference`, `cuda` or `omp`.

- `explicit_laplacian` [bool][false] : Use the explicit laplacian instead of deal.ii's matrix.

- `set_1d_laplacian_size`[uint32][16] : The number of grid points in one dimension for the 2D laplacian problem.

- `enable_random_rhs` [bool][false] : Use a random rhs instead of the default 1.0's .

- `overlap` [uint32][2] : Overlap between the domains.

- `timings_file` [std::string][null] : The filename for the timings.

- `partition` [std::string][regular] : The partitioner used. The choices are `metis`, `regular` or `regular2d`.

- `metis_objtype` [std::string][null] : The objective type to minimize for the metis partitioner. The choices are `edgecut` and `totalvol`.

- `num_threads` [uint32][1] : Number of threads to bind to a process.

- `non_symmetric_matrix` [bool][false] : Explicitly state that the matrix is non-symmetric so that the local GMRES solver is used.

- `use_mixed_precision` [bool][false] : Use mixed precision in the communication.

Input settings

- `matrix_filename` [std::string][null] : The matrix file to read the global system matrix from.

Output settings

- `enable_debug_write` [bool][false] : Enable some debugging outputs to stdout.

- `write_comm_data` [bool][false] : Write the number of sends and recvs of each subdomain to files.

- `write_perm_data` [bool][false] : Write the permutation data from CHOLMOD to a file.

- `print_config` [bool][true] : Print the configuration of the run.

- `print_matrices` [bool][false] : Print the local system matrices to a file.

- `debug` [bool][false] : Enable some possible expensive debug checks.

- `enable_logging` [bool][false] : Enable some possible expensive logging from Ginkgo.

Solver settings

Generic settings

- `num_iters` [uint32][100] : The number of outer iterations for the RAS solver.

- `set_tol` [double][1e-6] : The Outer tolerance for the RAS solver.

- `local_tol` [double][1e-12] : The Inner tolerance for the local iterative solver.

  Communication settings

- `enable_onesided` [bool][false] : Enable the onesided asynchronous communication.

- `enable_twosided` [bool][true] : Enable the twosided asynchronous communication. A dummy flag.

- `enable_one_by_one` [bool][false] : Enable putting/getting of each element in onesided communication.

- `enable_put_all_local_residual_norms` [bool][false] : Enable putting of all local residual norms"

- `enable_comm_overlap` [bool][false] : Enable overlap of communication and computation.

- `flush_type` [std::string][flush-all] : The window flush strategy. The choices are `flush-local` and `flush-all`.

- `lock_type` [std::string][lock-all] : The window lock strategy. The choices are `lock-local` and `lock-all`.

- `remote_comm_type` [std::string][get] : The type of the remote communication. `get` uses `MPI_Get` and `put` uses `MPI_Put`.

Convergence settings

- `enable_global_check` [bool][false] : Use the global convergence check for twosided.

- `global_convergence_type` [std::string][centralized-tree] : Choose the convergence detection algorithm for onesided.

- `enable_decentralized_accumulate` [bool][false] : Use accumulate strategy for decentralized convergence check..

- `enable_global_check_iter_offset` [bool][false] : Enable global convergence check only after a certain number of iterations.

Local solver settings

- `local_solver` [std::string][iterative-ginkgo] : The local solver used in the local domains. The current choices are `direct-cholmod`, `direct-ginkgo` or `iterative-ginkgo`.

- `local_factorization` [std::string][cholmod] : The factorization for the local direct solver "cholmod" or "umfpack".

- `local_reordering` [std::string][none] : The reordering for the local direct solver "none", "metis_↩ reordering" or "rcm_reordering".

- `factor_ordering_natural` [bool][false] : If true uses natural ordering instead of the default optimized ordering. This is needed for CUDA runs as the factorization ordering needs to be given to the solver.

- `enable_local_precond` [bool][false] : If true uses the Block jacobi preconditioning for the local iterative solver.

- `precond_max_block_size` [uint32][16]: Maximum size of the blocks for the block jacobi preconditioner

- `shifted_iter` [uint32][1] : The number of iterations to communicate for the local subdomains.

- `local_max_iters` [int32][-1] : The maximum number of iterations for the local iterative solver.

- `restart_iter` [uint32][1] : The restart iter for the GMRES solver.

- `reset_local_crit_iter` [int32][-1] : The RAS iteration to reset the local iteration count.

Poisson solver using Restricted Additive Schwarz with overlap.

This example runs is written within the `benchmarking/bench_ras.cpp` file. This demonstrates the basic capabilities of `schwarz-lib`. You can use it to solve the 2D Poisson equation with a 5 point stencil or solve a generic matrix by providing it a matrix file.

Examples with deal.ii

These examples use `deal.ii`'s capabilities to generate a matrix and solution is computed with the RAS method.

Possible settings are:

- `num_refine_cycles` [uint32][1][disabled] : The number of refinement cycles when used with `deal.ii`.

- `init_refine_level` [uint32][4] : The initial refinement level of the problem. This sets the initial number of dof's.

- `dealii_orig` [bool][false] : Solve with the deal.ii iterative CG instead of the RAS solver.

- `vis_sol` [bool][false] : Print the solution for visualization.

Solving the n-dimensional Poisson equation with FEM.

The `benchmarking/dealii_ex_6.cpp` demonstrates the solution of the Poisson equation with adaptive refinement as explained on the deal.ii example documentation page

Solving the Advection equation with FEM.

The `benchmarking/dealii_ex_9.cpp` demonstrates the solution of the Advection equation with adaptive refinement as explained on the deal.ii example documentation page

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, MixedValueType >

  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, MixedValueType >

  An implementation of the solver interface using the RAS solver.

- class schwz::SchwarzBase< ValueType, IndexType, MixedValueType >

  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, MixedValueType >

    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 (35a1195)

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 (6bca89b)

7.3 schwz::Communicate< ValueType, IndexType, MixedValueType >::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.
- 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.
- gko::Array< gko::Array< IndexType > > global_put

  The array containing the number of elements that each subdomain sends from the other.
- gko::Array< gko::Array< IndexType > > local_put

  The array containing the number of elements that each subdomain sends from the other.
- gko::Array< gko::Array< IndexType > > global_get

  The array containing the number of elements that each subdomain gets from the other.
- gko::Array< 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 > > 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 (always allocated).
- std::shared_ptr< gko::matrix::Dense< MixedValueType > > mixedt_send_buffer

  The mixed 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 (always allocated).
- std::shared_ptr< gko::matrix::Dense< MixedValueType > > mixedt_recv_buffer

  The mixed precision recv buffer used for the actual communication for both one-sided and two-sided.
- 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, typename MixedValueType**>
struct schwz::Communicate< **ValueType, IndexType, MixedValueType** >**::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 , typename MixedValueType >
gko::Array<gko::Array<IndexType> > schwz::Communicate< ValueType, IndexType, MixedValueType
>::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, MixedValueType >::initialize(), schwz::SolverRAS<
ValueType, IndexType, MixedValueType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType,
MixedValueType >::setup_windows().

7.3.2.2   global_put

```
template<typename ValueType , typename IndexType , typename MixedValueType >
gko::Array<gko::Array<IndexType> > schwz::Communicate< ValueType, IndexType, MixedValueType
>::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, MixedValueType >::initialize(), schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_windows().

7.3.2.3   is_local_neighbor

```
template<typename ValueType , typename IndexType , typename MixedValueType >
std::vector<bool> schwz::Communicate< ValueType, IndexType, MixedValueType >::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, MixedValueType >::initialize(), schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_windows().

7.3.2.4   local_get

```
template<typename ValueType , typename IndexType , typename MixedValueType >
gko::Array<gko::Array<IndexType> > schwz::Communicate< ValueType, IndexType, MixedValueType
>::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::SchwarzBase< ValueType, IndexType, MixedValueType >::initialize(), schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_windows().

7.3.2.5 local_put

```
template<typename ValueType , typename IndexType , typename MixedValueType >
gko::Array<gko::Array<IndexType> > schwz::Communicate< ValueType, IndexType, MixedValueType
>::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::SchwarzBase< ValueType, IndexType, MixedValueType >::initialize(), schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_comm_buffers(), and schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_windows().

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

- communicate.hpp (6bca89b)

7.4 schwz::Communicate< ValueType, IndexType, MixedValueType > 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, const std::shared_ptr< gko::matrix::Dense< ValueType >> &prev_global_solution, std::shared_ptr< gko::matrix::Dense< ValueType >> &global_solution)=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, typename MixedValueType**>
class schwz::Communicate< **ValueType, IndexType, MixedValueType** >

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 , typename MixedValueType >
void schwz::Communicate< ValueType, IndexType, MixedValueType >::exchange_boundary (
          const Settings & settings,
          const Metadata< ValueType, IndexType > & metadata,
          const std::shared_ptr< gko::matrix::Dense< ValueType >> & prev_global_solution,
          std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution )  [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, MixedValueType >.

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

7.4.2.2 local_to_global_vector()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::Communicate< ValueType, IndexType, MixedValueType >::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.

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

```
70 {
71     using vec = gko::matrix::Dense<ValueType>;
72     auto alpha = gko::initialize<gko::matrix::Dense<ValueType>>(
73         {1.0}, settings.executor);
74     auto temp_vector = vec::create(
75         settings.executor, gko::dim<2>(metadata.local_size, 1),
76         gko::Array<ValueType>::view(
77             settings.executor, metadata.local_size,
78             &global_vector->get_values()[metadata.first_row
79                                          ->get_data()[metadata.my_rank]]),
80         1);
81
82     auto temp_vector2 = vec::create(
83         settings.executor, gko::dim<2>(metadata.local_size, 1),
84         gko::Array<ValueType>::view(settings.executor, metadata.local_size,
85                             local_vector->get_values()),
86         1);
87     if (settings.convergence_settings.convergence_crit ==
88         Settings::convergence_settings::local_convergence_crit::
89             residual_based) {
90         local_vector->add_scaled(alpha.get(), temp_vector.get());
91         temp_vector->add_scaled(alpha.get(), local_vector.get());
92     } else {
93         temp_vector->copy_from(temp_vector2.get());
94     }
95 }
```

7.4.2.3 setup_windows()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::Communicate< ValueType, IndexType, MixedValueType >::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, MixedValueType >.

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

7.4.2.4 update_boundary()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::Communicate< ValueType, IndexType, MixedValueType >::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, MixedValueType >.

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

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

- communicate.hpp (6bca89b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp (7c9a9f7)

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 (6bca89b)

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 (35a1195)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (35a1195)

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 (35a1195)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (35a1195)

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 (5a15602)

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_rhs (std::vector< ValueType > &rhs)

    Generates the 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)

    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_rhs()

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

Generates the right hand side vector.

Parameters

rhs	The rhs vector.

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

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

```
90 {
91     std::uniform_real_distribution<double> unif(0.0, 1.0);
92     std::default_random_engine engine;
93     for (gko::size_type i = 0; i < rhs.size(); ++i) {
94         rhs[i] = unif(engine);
95     }
96 }
```

7.9.2.2 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, MixedValueType >::initialize().

```
284 {
285     partition_indices.resize(metadata.global_size);
286     if (metadata.my_rank == 0) {
287         auto partition_settings =
288             (Settings::partition_settings::partition_zoltan |
289              Settings::partition_settings::partition_metis |
290              Settings::partition_settings::partition_regular |
291              Settings::partition_settings::partition_regular2d |
292              Settings::partition_settings::partition_custom) &
293             settings.partition;
294
295         if (partition_settings ==
296             Settings::partition_settings::partition_zoltan) {
297             SCHWARZ_NOT_IMPLEMENTED;
298         } else if (partition_settings ==
299                 Settings::partition_settings::partition_metis) {
300             if (metadata.my_rank == 0) {
301                 std::cout << " METIS partition" << std::endl;
302             }
303             PartitionTools::PartitionMetis(
304                 settings, global_matrix, this->cell_weights,
305                 metadata.num_subdomains, partition_indices);
306         } else if (partition_settings ==
307                 Settings::partition_settings::partition_regular) {
308             if (metadata.my_rank == 0) {
309                 std::cout << " Regular 1D partition" << std::endl;
310             }
311             PartitionTools::PartitionRegular(
312                 global_matrix, metadata.num_subdomains, partition_indices);
313         } else if (partition_settings ==
314                 Settings::partition_settings::partition_regular2d) {
315             if (metadata.my_rank == 0) {
316                 std::cout << " Regular 2D partition" << std::endl;
317             }
318             PartitionTools::PartitionRegular2D(
319                 global_matrix, settings.write_debug_out,
320                 metadata.num_subdomains, partition_indices);
```

```
321            } else if (partition_settings ==
322                        Settings::partition_settings::partition_custom) {
323                // User partitions mesh manually
324                SCHWARZ_NOT_IMPLEMENTED;
325            } else {
326                SCHWARZ_NOT_IMPLEMENTED;
327            }
328        }
329 }
```

7.9.2.3 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_rhs(), and schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::initialize().

```
200 {
201     using index_type = IndexType;
202     using value_type = ValueType;
203     using mtx = gko::matrix::Csr<value_type, index_type>;
204     if (settings.matrix_filename != "null") {
205         auto input_file = std::ifstream(filename, std::ios::in);
206         if (!input_file) {
207             std::cerr << "Could not find the file \"" << filename
208                       << "\", which is required for this test.\n";
209         }
210         global_matrix =
211             gko::read<mtx>(input_file, settings.executor->get_master());
212         global_matrix->sort_by_column_index();
213         std::cout << "Matrix from file " << filename << std::endl;
214     } else if (settings.matrix_filename == "null" &&
215                settings.explicit_laplacian) {
216         std::cout << "Laplacian 2D Matrix (generated in house) " << std::endl;
217         gko::size_type global_size = oned_laplacian_size *
    oned_laplacian_size;
218
219         global_matrix = mtx::create(settings.executor->get_master(),
220                                     gko::dim<2>(global_size), 5 * global_size);
221         value_type *values = global_matrix->get_values();
222         index_type *row_ptrs = global_matrix->get_row_ptrs();
223         index_type *col_idxs = global_matrix->get_col_idxs();
224
225         std::vector<gko::size_type> exclusion_set;
226
227         std::map<IndexType, ValueType> stencil_map = {
228             {-oned_laplacian_size, -1}, {-1, -1}, {0, 4}, {1, -1},
229             {oned_laplacian_size, -1},
230         };
231         for (auto i = 2; i < global_size; ++i) {
232             gko::size_type index = (i - 1) * oned_laplacian_size;
233             if (index * index < global_size * global_size) {
234                 exclusion_set.push_back(
235                     linearize_index(index, index - 1, global_size));
236                 exclusion_set.push_back(
237                     linearize_index(index - 1, index, global_size));
```

```
238                }
239            }
240
241        std::sort(exclusion_set.begin(),
242                  exclusion_set.begin() + exclusion_set.size());
243
244        IndexType pos = 0;
245        IndexType col_idx = 0;
246        row_ptrs[0] = pos;
247        gko::size_type cur_idx = 0;
248        for (IndexType i = 0; i < global_size; ++i) {
249            for (auto ofs : stencil_map) {
250                auto in_exclusion_flag =
251                    (exclusion_set[cur_idx] ==
252                     linearize_index(i, i + ofs.first, global_size));
253                if (0 <= i + ofs.first && i + ofs.first < global_size &&
254                    !in_exclusion_flag) {
255                    values[pos] = ofs.second;
256                    col_idxs[pos] = i + ofs.first;
257                    ++pos;
258                }
259                if (in_exclusion_flag) {
260                    cur_idx++;
261                }
262                col_idx = row_ptrs[i + 1] - pos;
263            }
264            row_ptrs[i + 1] = pos;
265        }
266    } else {
267        std::cerr << " Need to provide a matrix or enable the default "
268                     "laplacian matrix."
269                  << std::endl;
270        std::exit(-1);
271    }
272 }
```

7.9.2.4 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.
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, MixedValueType >.

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

7.9.2.5   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 )
```

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, MixedValueType >::initialize().

```
339 {
340     using vec = gko::matrix::Dense<ValueType>;
341     auto my_rank = metadata.my_rank;
342     auto first_row = metadata.first_row->get_data()[my_rank];
343
344     // Copy the global rhs vector to the required executor.
345     gko::Array<ValueType> temp_rhs{settings.executor->get_master(), rhs.begin(),
346                                    rhs.end()};
347     global_rhs = vec::create(settings.executor,
348                              gko::dim<2>{metadata.global_size, 1}, temp_rhs, 1);
349
350     local_rhs =
351         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
352     // Extract the local rhs from the global rhs. Also takes into account the
353     // overlap.
354     SolverTools::extract_local_vector(settings, metadata, local_rhs.get(),
355                                       global_rhs.get(), first_row);
356
357     local_solution =
358         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
359 }
```

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

- initialization.hpp (200bbde)
- /home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp (92dbd95)

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.

- IndexType updated_max_iters

    The updated 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.
- 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.
- 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 (6bca89b)

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 (35a1195)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (35a1195)

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 (6bca89b)

7.13 schwz::SchwarzBase< ValueType, IndexType, MixedValueType > 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 > > 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, typename MixedValueType = gko↩ ::default_precision**>
class schwz::SchwarzBase< **ValueType, IndexType, MixedValueType** >

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 , typename MixedValueType >
schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::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.

```
76      : Initialize<ValueType, IndexType>(settings, metadata),
77        settings(settings),
78        metadata(metadata)
79  {
80      using vec_itype = gko::Array<IndexType>;
81      metadata.my_local_rank =
82          Utils<ValueType, IndexType>::get_local_rank(metadata.mpi_communicator);
83      metadata.local_num_procs = Utils<ValueType, IndexType>::get_local_num_procs(
84          metadata.mpi_communicator);
85      auto my_local_rank = metadata.my_local_rank;
86      if (settings.executor_string == "omp") {
87          settings.executor = gko::OmpExecutor::create();
88          auto exec_info =
89              static_cast<gko::OmpExecutor *>(settings.executor.get())
90                  ->get_exec_info();
91          exec_info->bind_to_core(metadata.my_local_rank);
92
93      } else if (settings.executor_string == "cuda") {
94          int num_devices = 0;
95  #if SCHW_HAVE_CUDA
96          SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
97  #else
98          SCHWARZ_NOT_IMPLEMENTED;
99  #endif
100         Utils<ValueType, IndexType>::assert_correct_cuda_devices(
101             num_devices, metadata.my_rank);
102         settings.executor = gko::CudaExecutor::create(
103             my_local_rank, gko::OmpExecutor::create(), false);
104         auto exec_info = static_cast<gko::OmpExecutor *>(
105                         settings.executor->get_master().get())
106                         ->get_exec_info();
107         exec_info->bind_to_core(my_local_rank);
108         settings.cuda_device_guard =
109             std::make_shared<schwz::device_guard>(my_local_rank);
```

```
110
111            std::cout << " Rank " << metadata.my_rank << " with local rank "
112                    << my_local_rank << " has "
113                    << (static_cast<gko::CudaExecutor *>(settings.executor.get()))
114                          ->get_device_id()
115                    << " id of gpu" << std::endl;
116        MPI_Barrier(metadata.mpi_communicator);
117    } else if (settings.executor_string == "reference") {
118        settings.executor = gko::ReferenceExecutor::create();
119        auto exec_info =
120            static_cast<gko::ReferenceExecutor *>(settings.executor.get())
121                ->get_exec_info();
122        exec_info->bind_to_core(my_local_rank);
123    }
124 }
```

7.13.3 Member Function Documentation

7.13.3.1 print_matrix()

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32, typename
MixedValueType = gko::default_precision>
void schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::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, typename
MixedValueType = gko::default_precision>
void schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::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 , typename MixedValueType >
void schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::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::Communicate< ValueType, IndexType, MixedValueType >::exchange_boundary(), schwz::←
Settings::executor, schwz::Settings::executor_string, schwz::SchwarzBase< ValueType, IndexType, MixedValue←
Type >::global_matrix, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::global_rhs, schwz←
::SchwarzBase< ValueType, IndexType, MixedValueType >::global_solution, schwz::SchwarzBase< ValueType,
IndexType, MixedValueType >::interface_matrix, schwz::SchwarzBase< ValueType, IndexType, MixedValueType
>::local_inv_perm, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::local_matrix, schwz::←
SchwarzBase< ValueType, IndexType, MixedValueType >::local_perm, schwz::SchwarzBase< ValueType, Index←
Type, MixedValueType >::local_rhs, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::local_←
solution, schwz::Communicate< ValueType, IndexType, MixedValueType >::local_to_global_vector(), schwz::←
Communicate< ValueType, IndexType, MixedValueType >::setup_windows(), schwz::SchwarzBase< ValueType,
IndexType, MixedValueType >::triangular_factor_l, schwz::SchwarzBase< ValueType, IndexType, MixedValueType
>::triangular_factor_u, schwz::Communicate< ValueType, IndexType, MixedValueType >::update_boundary(), and
schwz::Settings::write_iters_and_residuals.

```
321 {
322     using vec_vtype = gko::matrix::Dense<ValueType>;
323     if (!solution.get()) {
324         solution =
325             vec_vtype::create(settings.executor->get_master(),
326                               gko::dim<2>(this->metadata.global_size, 1));
327     }
328     MixedValueType dummy1 = 0.0;
329     ValueType dummy2 = 1.0;
330
331     if (metadata.my_rank == 0) {
332         std::cout << " MixedValueType: " << typeid(dummy1).name()
333                   << " ValueType: " << typeid(dummy2).name() << std::endl;
334     }
335     // The main solution vector
336     std::shared_ptr<vec_vtype> global_solution = vec_vtype::create(
337         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
338     // The previous iteration solution vector
339     std::shared_ptr<vec_vtype> prev_global_solution =
340         vec_vtype::create(this->settings.executor->get_master(),
341                           gko::dim<2>(this->metadata.global_size, 1));
342     // A work vector.
343     std::shared_ptr<vec_vtype> work_vector = vec_vtype::create(
344         settings.executor, gko::dim<2>(2 * this->metadata.local_size_x, 1));
345     // An initial guess.
346     std::shared_ptr<vec_vtype> init_guess = vec_vtype::create(
347         settings.executor, gko::dim<2>(this->metadata.local_size_x, 1));
348     // init_guess->copy_from(local_rhs.get());
349
350     if (settings.executor_string == "omp") {
351         ValueType sum_rhs = std::accumulate(
352             local_rhs->get_values(),
353             local_rhs->get_values() + local_rhs->get_size()[0], 0.0);
354         std::cout << " Rank " << this->metadata.my_rank << " sum local rhs "
355                   << sum_rhs << std::endl;
356     }
357
358     // std::vector<IndexType> local_converged_iter_count;
359
360     // Setup the windows for the onesided communication.
```

```
361     this->setup_windows(this->settings, this->metadata, global_solution);
362
363     const auto solver_settings =
364         (Settings::local_solver_settings::direct_solver_cholmod |
365          Settings::local_solver_settings::direct_solver_umfpack |
366          Settings::local_solver_settings::direct_solver_ginkgo |
367          Settings::local_solver_settings::iterative_solver_dealii |
368          Settings::local_solver_settings::iterative_solver_ginkgo) &
369         settings.local_solver;
370     prev_global_solution->copy_from(gko::lend(global_solution));
371
372     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
373              global_residual_norm = 0.0, global_residual_norm0 = -1.0;
374     metadata.iter_count = 0;
375     auto start_time = std::chrono::steady_clock::now();
376     int num_converged_procs = 0;
377
378     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
379         // Exchange the boundary values. The communication part.
380         prev_global_solution->copy_from(gko::lend(global_solution));
381         MEASURE_ELAPSED_FUNC_TIME(
382             this->exchange_boundary(settings, metadata, global_solution,
383                                     prev_global_solution),
384             0, metadata.my_rank, boundary_exchange, metadata.iter_count);
385         global_solution->copy_from(gko::lend(prev_global_solution));
386
387         // Update the boundary and interior values after the exchanging from
388         // other processes.
389         MEASURE_ELAPSED_FUNC_TIME(
390             this->update_boundary(settings, metadata, this->
    local_solution,
391                                   this->local_rhs, global_solution,
392                                   this->interface_matrix),
393             1, metadata.my_rank, boundary_update, metadata.iter_count);
394
395         // Check for the convergence of the solver.
396         // num_converged_procs = 0;
397         MEASURE_ELAPSED_FUNC_TIME(
398             (Solve<ValueType, IndexType, MixedValueType>::check_convergence(
399                 settings, metadata, this->comm_struct, this->convergence_vector,
400                 global_solution, this->local_solution, this->
    local_matrix,
401                 work_vector, local_residual_norm, local_residual_norm0,
402                 global_residual_norm, global_residual_norm0,
403                 num_converged_procs)),
404             2, metadata.my_rank, convergence_check, metadata.iter_count);
405
406         // break if the solution diverges.
407         if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
408             std::cout << " Rank " << metadata.my_rank << " diverged in "
409                       << metadata.iter_count << " iters " << std::endl;
410             std::exit(-1);
411         }
412
413         // break if all processes detect that all other processes have
414         // converged otherwise continue iterations.
415         if (num_converged_procs == metadata.num_subdomains) {
416             break;
417         } else {
418             MEASURE_ELAPSED_FUNC_TIME(
419                 (Solve<ValueType, IndexType, MixedValueType>::local_solve(
420                     settings, metadata, this->local_matrix,
421                     this->triangular_factor_l, this->
    triangular_factor_u,
422                     this->local_perm, this->local_inv_perm, work_vector,
423                     init_guess, this->local_solution)),
424                 3, metadata.my_rank, local_solve, metadata.iter_count);
425
426             // Gather the local vector into the locally global vector for
427             // communication.
428             MEASURE_ELAPSED_FUNC_TIME(
429                 (Communicate<ValueType, IndexType, MixedValueType>::
430                     local_to_global_vector(settings, metadata,
431                                            this->local_solution,
432                                            global_solution)),
433                 4, metadata.my_rank, expand_local_vec, metadata.iter_count);
434         }
435     }
436     MPI_Barrier(MPI_COMM_WORLD);
437     auto elapsed_time = std::chrono::duration<ValueType>(
438         std::chrono::steady_clock::now() - start_time);
439     // Write the residuals and iterations to files
440     if (settings.write_iters_and_residuals &&
441         solver_settings ==
442             Settings::local_solver_settings::iterative_solver_ginkgo) {
443         std::string rank_string = std::to_string(metadata.my_rank);
444         if (metadata.my_rank < 10) {
```

```
445                rank_string = "0" + std::to_string(metadata.my_rank);
446            }
447            std::string filename = "iter_res_" + rank_string + ".csv";
448            write_iters_and_residuals(
449                metadata.num_subdomains, metadata.my_rank,
450                metadata.post_process_data.local_residual_vector_out.size(),
451                metadata.post_process_data.local_residual_vector_out,
452                metadata.post_process_data.local_converged_iter_count,
453                metadata.post_process_data.local_converged_resnorm,
454                metadata.post_process_data.local_timestamp, filename);
455        }
456        if (num_converged_procs < metadata.num_subdomains) {
457            std::cout << "Rank " << metadata.my_rank << " did not converge in "
458                    << metadata.iter_count << " iterations." << std::endl;
459        } else {
460            std::cout << " Rank " << metadata.my_rank << " converged in "
461                    << metadata.iter_count << " iterations " << std::endl;
462            ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
463                    residual_norm = -1.0;
464
465            // Compute the final residual norm. Also gathers the solution from all
466            // subdomains.
467            Solve<ValueType, IndexType, MixedValueType>::compute_residual_norm(
468                settings, metadata, global_matrix, global_rhs, global_solution,
469                mat_norm, rhs_norm, sol_norm, residual_norm);
470            gather_comm_data<ValueType, IndexType, MixedValueType>(
471                metadata.num_subdomains, this->comm_struct,
472                metadata.comm_data_struct);
473            // clang-format off
474            if (metadata.my_rank == 0)
475              {
476                std::cout
477                   << " residual norm " << residual_norm << "\n"
478                   << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
479                   << " Time taken for solve " << elapsed_time.count()
480                   << std::endl;
481              }
482            // clang-format on
483        }
484        if (metadata.my_rank == 0) {
485            solution->copy_from(global_solution.get());
486        }
487
488        // Communicate<ValueType, IndexType>::clear(settings);
489 }
```

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

- schwarz_base.hpp (ca00a18)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz_base.cpp (6bca89b)

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.
- bool use_mixed_precision = false

  Flag if mixed precision should be used.
- bool enable_random_rhs = false

  Flag to enable a random rhs.
- 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.
- int reset_local_crit_iter = -1

  The global iter at which to reset the local solver criterion.
- 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 enable_logging = false

  Flag to enable logging for local iterative solvers.
- 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 enable_logging

```
bool schwz::Settings::enable_logging = false
```

Flag to enable logging for local iterative solvers.

Note: Probably will have a significant performance hit.

7.14.2.2 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, MixedValueType >::initialize().

7.14.2.3 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.

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

- settings.hpp (6bca89b)

7.15 schwz::Solve< ValueType, IndexType, MixedValueType > 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, typename MixedValueType = gko**↩
::default_precision>
class schwz::Solve< **ValueType, IndexType, MixedValueType** >

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 (228ce7a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/solve.cpp (92dbd95)

7.16 schwz::SolverRAS< ValueType, IndexType, MixedValueType > 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,
  const std::shared_ptr< gko::matrix::Dense< ValueType >> &prev_global_solution, std::shared_ptr< gko←↩
  ::matrix::Dense< ValueType >> &global_solution) 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, typename MixedValueType = gko←↩
::default_precision>
class schwz::SolverRAS< ValueType, IndexType, MixedValueType >

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 , typename MixedValueType >
schwz::SolverRAS< ValueType, IndexType, MixedValueType >::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.

```
51     : SchwarzBase<ValueType, IndexType, MixedValueType>(settings, metadata)
52 {}
```

7.16.3 Member Function Documentation

7.16.3.1 exchange_boundary()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::SolverRAS< ValueType, IndexType, MixedValueType >::exchange_boundary (
            const Settings & settings,
            const Metadata< ValueType, IndexType > & metadata,
            const std::shared_ptr< gko::matrix::Dense< ValueType >> & prev_global_solution,
            std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution ) [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, MixedValueType >.

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

```
951 {
952     if (settings.comm_settings.enable_onesided) {
953         exchange_boundary_onesided<ValueType, IndexType, MixedValueType>(
954             settings, metadata, this->comm_struct, prev_global_solution,
955             global_solution);
956     } else {
957         exchange_boundary_twosided<ValueType, IndexType, MixedValueType>(
958             settings, metadata, this->comm_struct, prev_global_solution,
959             global_solution);
960     }
961 }
```

7.16.3.2 setup_local_matrices()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::SolverRAS< ValueType, IndexType, MixedValueType >::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, MixedValueType >← ::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, MixedValueType >::interface_matrix, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::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, Index← Type >::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.

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

```
106             partition_settings ==
107                 Settings::partition_settings::partition_regular2d) {
108         if (num_subdomains > 1) {
109             for (auto p = 0; p < num_subdomains; p++) {
110                 local_p_size[p] = 0;
111             }
112             for (auto i = 0; i < global_size; i++) {
113                 local_p_size[partition_indices[i]]++;
114             }
115             first_row[0] = 0;
116             for (auto p = 0; p < num_subdomains; ++p) {
117                 first_row[p + 1] = first_row[p] + local_p_size[p];
118             }
119             // permutation
120             for (auto i = 0; i < global_size; i++) {
121                 permutation[first_row[partition_indices[i]]] = i;
122                 first_row[partition_indices[i]]++;
123             }
124             for (auto p = num_subdomains; p > 0; p--) {
125                 first_row[p] = first_row[p - 1];
126             }
127             first_row[0] = 0;
128
129             // iperm
130             for (auto i = 0; i < global_size; i++) {
131                 i_permutation[permutation[i]] = i;
132             }
133         }
134
135         auto gmat_temp = mtx::create(settings.executor->get_master(),
136                                      global_matrix->get_size(),
137                                      global_matrix->get_num_stored_elements());
138
139         auto nnz = 0;
140         gmat_temp->get_row_ptrs()[0] = 0;
141         for (auto row = 0; row < metadata.global_size; ++row) {
142             for (auto col = gmat_row_ptrs[permutation[row]];
143                  col < gmat_row_ptrs[permutation[row] + 1]; ++col) {
144                 gmat_temp->get_col_idxs()[nnz] =
145                     i_permutation[gmat_col_idxs[col]];
146                 gmat_temp->get_values()[nnz] = gmat_values[col];
147                 nnz++;
148             }
149             gmat_temp->get_row_ptrs()[row + 1] = nnz;
150         }
151         global_matrix->copy_from(gmat_temp.get());
152     }
153
154
155     for (auto i = 0; i < global_size; i++) {
156         global_to_local[i] = 0;
157         local_to_global[i] = 0;
158     }
159     auto num = 0;
160     for (auto i = first_row[my_rank]; i < first_row[
    my_rank + 1]; i++) {
161         global_to_local[i] = 1 + num;
162         local_to_global[num] = i;
163         num++;
164     }
165
166     IndexType old = 0;
167     for (auto k = 1; k < settings.overlap; k++) {
168         auto now = num;
169         for (auto i = old; i < now; i++) {
170             for (auto j = gmat_row_ptrs[local_to_global[i]];
171                  j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
172                 if (global_to_local[gmat_col_idxs[j]] == 0) {
173                     local_to_global[num] = gmat_col_idxs[j];
174                     global_to_local[gmat_col_idxs[j]] = 1 + num;
175                     num++;
176                 }
177             }
178         }
179         old = now;
180     }
181     metadata.local_size = local_p_size[my_rank];
182     metadata.local_size_x = num;
183     metadata.local_size_o = global_size;
184     auto local_size = metadata.local_size;
185     auto local_size_x = metadata.local_size_x;
186
187     metadata.overlap_size = num - metadata.local_size;
188     auto host_ov_row = gko::Array<IndexType>::view(
189         settings.executor->get_master(), metadata.overlap_size,
190         &(metadata.local_to_global->get_data()[metadata.local_size]));
191     metadata.overlap_row = vec_itype(settings.executor, metadata.overlap_size);
```

```
192        metadata.overlap_row = host_ov_row;
193
194        auto nnz_local = 0;
195        auto nnz_interface = 0;
196
197        for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
198            for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; j++) {
199                if (global_to_local[gmat_col_idxs[j]] != 0) {
200                    nnz_local++;
201                } else {
202                    std::cout << " debug: invalid edge?" << std::endl;
203                }
204            }
205        }
206        auto temp = 0;
207        for (auto k = 0; k < metadata.overlap_size; k++) {
208            temp = host_ov_row.get_data()[k];
209            for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1]; j++) {
210                if (global_to_local[gmat_col_idxs[j]] != 0) {
211                    nnz_local++;
212                } else {
213                    nnz_interface++;
214                }
215            }
216        }
217
218        std::shared_ptr<mtx> local_matrix_compute;
219        local_matrix_compute = mtx::create(settings.executor->get_master(),
220                                    gko::dim<2>(local_size_x), nnz_local);
221        IndexType *lmat_row_ptrs = local_matrix_compute->get_row_ptrs();
222        IndexType *lmat_col_idxs = local_matrix_compute->get_col_idxs();
223        ValueType *lmat_values = local_matrix_compute->get_values();
224
225        std::shared_ptr<mtx> interface_matrix_compute;
226        if (nnz_interface > 0) {
227            interface_matrix_compute =
228                mtx::create(settings.executor->get_master(),
229                            gko::dim<2>(local_size_x), nnz_interface);
230        } else {
231            interface_matrix_compute = mtx::create(settings.executor->get_master());
232        }
233
234        IndexType *imat_row_ptrs = interface_matrix_compute->get_row_ptrs();
235        IndexType *imat_col_idxs = interface_matrix_compute->get_col_idxs();
236        ValueType *imat_values = interface_matrix_compute->get_values();
237
238        num = 0;
239        nnz_local = 0;
240        auto nnz_interface_temp = 0;
241        lmat_row_ptrs[0] = nnz_local;
242        if (nnz_interface > 0) {
243            imat_row_ptrs[0] = nnz_interface_temp;
244        }
245        // Local interior matrix
246        for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
247            for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; ++j) {
248                if (global_to_local[gmat_col_idxs[j]] != 0) {
249                    lmat_col_idxs[nnz_local] =
250                        global_to_local[gmat_col_idxs[j]] - 1;
251                    lmat_values[nnz_local] = gmat_values[j];
252                    nnz_local++;
253                }
254            }
255            if (nnz_interface > 0) {
256                imat_row_ptrs[num + 1] = nnz_interface_temp;
257            }
258            lmat_row_ptrs[num + 1] = nnz_local;
259            num++;
260        }
261
262        // Interface matrix
263        if (nnz_interface > 0) {
264            nnz_interface = 0;
265            for (auto k = 0; k < metadata.overlap_size; k++) {
266                temp = host_ov_row.get_data()[k];
267                for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1];
268                     j++) {
269                    if (global_to_local[gmat_col_idxs[j]] != 0) {
270                        lmat_col_idxs[nnz_local] =
271                            global_to_local[gmat_col_idxs[j]] - 1;
272                        lmat_values[nnz_local] = gmat_values[j];
273                        nnz_local++;
274                    } else {
275                        imat_col_idxs[nnz_interface] = gmat_col_idxs[j];
276                        imat_values[nnz_interface] = gmat_values[j];
277                        nnz_interface++;
278                    }
```

```
279                }
280            lmat_row_ptrs[num + 1] = nnz_local;
281            imat_row_ptrs[num + 1] = nnz_interface;
282            num++;
283        }
284    }
285    auto now = num;
286    for (auto i = old; i < now; i++) {
287        for (auto j = gmat_row_ptrs[local_to_global[i]];
288             j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
289            if (global_to_local[gmat_col_idxs[j]] == 0) {
290                local_to_global[num] = gmat_col_idxs[j];
291                global_to_local[gmat_col_idxs[j]] = 1 + num;
292                num++;
293            }
294        }
295    }
296
297    local_matrix_compute->sort_by_column_index();
298    interface_matrix_compute->sort_by_column_index();
299
300    local_matrix = mtx::create(settings.executor);
301    local_matrix->copy_from(gko::lend(local_matrix_compute));
302    interface_matrix = mtx::create(settings.executor);
303    interface_matrix->copy_from(gko::lend(interface_matrix_compute));
304 }
```

7.16.3.3 setup_windows()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::SolverRAS< ValueType, IndexType, MixedValueType >::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, MixedValueType >.

References 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, MixedValueType >::comm_struct::get_displacements,
schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::get_request, schwz::←
Communicate< ValueType, IndexType, MixedValueType >::comm_struct::global_get, schwz::Communicate<
ValueType, IndexType, MixedValueType >::comm_struct::global_put, schwz::SchwarzBase< ValueType, Index←
Type, MixedValueType >::global_solution, schwz::Communicate< ValueType, IndexType, MixedValueType >←
::comm_struct::is_local_neighbor, schwz::Metadata< ValueType, IndexType >::iter_count, schwz::Communicate<
ValueType, IndexType, MixedValueType >::comm_struct::local_get, schwz::Communicate< ValueType, IndexType,
MixedValueType >::comm_struct::local_neighbors_in, schwz::Communicate< ValueType, IndexType, Mixed←
ValueType >::comm_struct::local_neighbors_out, schwz::Communicate< ValueType, IndexType, MixedValue←
Type >::comm_struct::local_num_neighbors_in, schwz::Communicate< ValueType, IndexType, MixedValue←
Type >::comm_struct::local_num_neighbors_out, schwz::Communicate< ValueType, IndexType, MixedValueType

>::comm_struct::local_put, schwz::Metadata< ValueType, IndexType >::local_size_o, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::mixedt_recv_buffer, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::mixedt_send_buffer, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::neighbors_in, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::neighbors_out, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_↩ struct::num_neighbors_in, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct↩ ::num_neighbors_out, schwz::Metadata< ValueType, IndexType >::num_subdomains, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::put_displacements, schwz::Communicate< Value↩ Type, IndexType, MixedValueType >::comm_struct::put_request, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::recv_buffer, schwz::Communicate< ValueType, IndexType, MixedValueType >↩ ::comm_struct::send_buffer, schwz::Settings::use_mixed_precision, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::window_recv_buffer, schwz::Communicate< ValueType, IndexType, Mixed↩ ValueType >::comm_struct::window_send_buffer, and schwz::Communicate< ValueType, IndexType, Mixed↩ ValueType >::comm_struct::window_x.

```
570 {
571     using vec_itype = gko::Array<IndexType>;
572     using vec_vtype = gko::matrix::Dense<ValueType>;
573     auto num_subdomains = metadata.num_subdomains;
574     auto local_size_o = metadata.local_size_o;
575     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
576     auto global_get = this->comm_struct.global_get->get_data();
577     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
578     auto global_put = this->comm_struct.global_put->get_data();
579
580     // set displacement for the MPI buffer
581     auto get_displacements = this->comm_struct.get_displacements->get_data();
582     auto put_displacements = this->comm_struct.put_displacements->get_data();
583     {
584         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
585         tmp_num_comm_elems[0] = 0;
586         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
587             if ((global_get[j])[0] > 0) {
588                 int p = neighbors_in[j];
589                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
590             }
591         }
592         for (auto j = 0; j < num_subdomains; j++) {
593             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
594         }
595
596         auto mpi_itype = schwz::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
597         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
598                      1, mpi_itype, MPI_COMM_WORLD);
599     }
600
601     {
602         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
603         tmp_num_comm_elems[0] = 0;
604         for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
605             if ((global_put[j])[0] > 0) {
606                 int p = neighbors_out[j];
607                 tmp_num_comm_elems[p + 1] = (global_put[j])[0];
608             }
609         }
610         for (auto j = 0; j < num_subdomains; j++) {
611             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
612         }
613
614         auto mpi_itype = schwz::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
615         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
616                      1, mpi_itype, MPI_COMM_WORLD);
617     }
618
619     // setup windows
620     if (settings.comm_settings.enable_onesided) {
621         // Onesided
622         MPI_Win_create(main_buffer->get_values(),
623                        main_buffer->get_size()[0] * sizeof(ValueType),
624                        sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
625                        &(this->comm_struct.window_x));
626     }
627
628
629     if (settings.comm_settings.enable_onesided) {
630         // MPI_Alloc_mem ? Custom allocator ?  TODO
631         MPI_Win_create(this->local_residual_vector->get_values(),
632                        (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
633                        MPI_INFO_NULL, MPI_COMM_WORLD,
```

```
634                            &(this->window_residual_vector));
635          std::vector<IndexType> zero_vec(num_subdomains, 0);
636          gko::Array<IndexType> temp_array{settings.executor->get_master(),
637                                   zero_vec.begin(), zero_vec.end()};
638          this->convergence_vector = std::shared_ptr<vec_itype>(
639              new vec_itype(settings.executor->get_master(), temp_array),
640              std::default_delete<vec_itype>());
641          this->convergence_sent = std::shared_ptr<vec_itype>(
642              new vec_itype(settings.executor->get_master(), num_subdomains),
643              std::default_delete<vec_itype>());
644          this->convergence_local = std::shared_ptr<vec_itype>(
645              new vec_itype(settings.executor->get_master(), num_subdomains),
646              std::default_delete<vec_itype>());
647          MPI_Win_create(this->convergence_vector->get_data(),
648                         (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
649                         MPI_INFO_NULL, MPI_COMM_WORLD,
650                         &(this->window_convergence));
651      }
652
653      if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
654          // Lock all windows.
655          if (settings.comm_settings.enable_get &&
656              settings.comm_settings.enable_lock_all) {
657              MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
658          }
659          if (settings.comm_settings.enable_put &&
660              settings.comm_settings.enable_lock_all) {
661              MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
662          }
663          if (settings.comm_settings.enable_one_by_one &&
664              settings.comm_settings.enable_lock_all) {
665              MPI_Win_lock_all(0, this->comm_struct.window_x);
666          }
667          MPI_Win_lock_all(0, this->window_residual_vector);
668          MPI_Win_lock_all(0, this->window_convergence);
669      }
670 }
```

7.16.3.4 update_boundary()

```
template<typename ValueType , typename IndexType , typename MixedValueType >
void schwz::SolverRAS< ValueType, IndexType, MixedValueType >::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, MixedValueType >.

References schwz::Settings::executor, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::global←
_solution, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::interface_matrix, schwz::Schwarz←
Base< ValueType, IndexType, MixedValueType >::local_rhs, schwz::Metadata< ValueType, IndexType >::local←
_size_x, schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::local_solution, schwz::Metadata<
ValueType, IndexType >::num_subdomains, and schwz::Settings::overlap.

```
972 {
973     using vec_vtype = gko::matrix::Dense<ValueType>;
974     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
975         {1.0}, settings.executor);
976     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
977         {-1.0}, settings.executor);
978     auto local_size_x = metadata.local_size_x;
979     local_solution->copy_from(local_rhs.get());
980     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
981         auto temp_solution = vec_vtype::create(
982             settings.executor, local_solution->get_size(),
983             gko::Array<ValueType>::view(settings.executor,
984                                         local_solution->get_size()[0],
985                                         global_solution->get_values()),
986             1);
987         interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
988                                 local_solution.get());
989     }
990 }
```

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

- restricted_schwarz.hpp (7c9a9f7)
- /home/runner/work/schwarz-lib/schwarz-lib/source/restricted_schwarz.cpp (6bca89b)

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 (35a1195)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (35a1195)

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 (1cd0e3b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/utils.cpp (f366659)

Index