

schwz

Generated automatically from event-based-new

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b># Installation Instructions</b>	<b>3</b>
<b>3</b>	<b>Testing Instructions</b>	<b>5</b>
<b>4</b>	<b>Benchmarking.</b>	<b>7</b>
<b>5</b>	<b>Module Documentation</b>	<b>11</b>
5.1	Communicate . . . . .	11
5.1.1	Detailed Description . . . . .	11
5.2	Initialization . . . . .	12
5.2.1	Detailed Description . . . . .	12
5.3	Schwarz Class . . . . .	13
5.3.1	Detailed Description . . . . .	13
5.4	Solve . . . . .	14
5.4.1	Detailed Description . . . . .	14
5.5	Utils . . . . .	15
5.5.1	Detailed Description . . . . .	15
<b>6</b>	<b>Namespace Documentation</b>	<b>17</b>
6.1	ProcessTopology Namespace Reference . . . . .	17
6.1.1	Detailed Description . . . . .	17
6.2	schwz Namespace Reference . . . . .	17
6.2.1	Detailed Description . . . . .	18
6.3	schwz::CommHelpers Namespace Reference . . . . .	18
6.3.1	Detailed Description . . . . .	18
6.4	schwz::conv_tools Namespace Reference . . . . .	18
6.4.1	Detailed Description . . . . .	18
6.5	schwz::EventHelpers Namespace Reference . . . . .	19
6.5.1	Detailed Description . . . . .	19
6.6	schwz::PartitionTools Namespace Reference . . . . .	19
6.6.1	Detailed Description . . . . .	19
6.7	schwz::SolverTools Namespace Reference . . . . .	19
6.7.1	Detailed Description . . . . .	19

<b>7</b>	<b>Class Documentation</b>	<b>21</b>
7.1	BadDimension Class Reference	21
7.1.1	Detailed Description	21
7.1.2	Constructor & Destructor Documentation	21
7.1.2.1	BadDimension()	21
7.2	schwz::Settings::comm_settings Struct Reference	22
7.2.1	Detailed Description	23
7.3	schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct Struct Reference	23
7.3.1	Detailed Description	25
7.3.2	Member Data Documentation	25
7.3.2.1	global_get	25
7.3.2.2	global_put	25
7.3.2.3	is_local_neighbor	26
7.3.2.4	local_get	26
7.3.2.5	local_put	26
7.4	schwz::Communicate< ValueType, IndexType, MixedValueType > Class Template Reference	27
7.4.1	Detailed Description	27
7.4.2	Member Function Documentation	28
7.4.2.1	exchange_boundary()	28
7.4.2.2	local_to_global_vector()	28
7.4.2.3	setup_windows()	29
7.4.2.4	update_boundary()	29
7.5	schwz::Settings::convergence_settings Struct Reference	30
7.5.1	Detailed Description	30
7.6	CudaError Class Reference	30
7.6.1	Detailed Description	31
7.6.2	Constructor & Destructor Documentation	31
7.6.2.1	CudaError()	31
7.7	CusparsError Class Reference	31
7.7.1	Detailed Description	32

7.7.2	Constructor & Destructor Documentation	32
7.7.2.1	CusparseError()	32
7.8	schwz::device_guard Class Reference	32
7.8.1	Detailed Description	32
7.9	schwz::Initialize< ValueType, IndexType > Class Template Reference	33
7.9.1	Detailed Description	33
7.9.2	Member Function Documentation	34
7.9.2.1	generate_dipole_rhs()	34
7.9.2.2	generate_rhs()	34
7.9.2.3	generate_sin_rhs()	35
7.9.2.4	partition()	35
7.9.2.5	setup_global_matrix()	37
7.9.2.6	setup_local_matrices()	39
7.9.2.7	setup_vectors()	39
7.10	schwz::Metadata< ValueType, IndexType > Struct Template Reference	40
7.10.1	Detailed Description	42
7.10.2	Member Data Documentation	42
7.10.2.1	local_solver_tolerance	42
7.10.2.2	tolerance	42
7.11	MetisError Class Reference	43
7.11.1	Detailed Description	43
7.11.2	Constructor & Destructor Documentation	43
7.11.2.1	MetisError()	43
7.12	schwz::Metadata< ValueType, IndexType >::post_process_data Struct Reference	44
7.12.1	Detailed Description	44
7.13	schwz::SchwarzBase< ValueType, IndexType, MixedValueType > Class Template Reference	44
7.13.1	Detailed Description	45
7.13.2	Constructor & Destructor Documentation	46
7.13.2.1	SchwarzBase()	46
7.13.3	Member Function Documentation	47

7.13.3.1	<a href="#">print_matrix()</a>	47
7.13.3.2	<a href="#">print_vector()</a>	47
7.13.3.3	<a href="#">run()</a>	47
7.14	<a href="#">schwz::Settings Struct Reference</a>	51
7.14.1	<a href="#">Detailed Description</a>	53
7.14.2	<a href="#">Member Data Documentation</a>	53
7.14.2.1	<a href="#">enable_logging</a>	53
7.14.2.2	<a href="#">explicit_laplacian</a>	53
7.14.2.3	<a href="#">naturally_ordered_factor</a>	53
7.14.2.4	<a href="#">norm_type</a>	53
7.14.2.5	<a href="#">thres_type</a>	54
7.15	<a href="#">schwz::Solve&lt; ValueType, IndexType, MixedValueType &gt; Class Template Reference</a>	54
7.15.1	<a href="#">Detailed Description</a>	54
7.16	<a href="#">schwz::SolverRAS&lt; ValueType, IndexType, MixedValueType &gt; Class Template Reference</a>	55
7.16.1	<a href="#">Detailed Description</a>	55
7.16.2	<a href="#">Constructor &amp; Destructor Documentation</a>	56
7.16.2.1	<a href="#">SolverRAS()</a>	56
7.16.3	<a href="#">Member Function Documentation</a>	56
7.16.3.1	<a href="#">exchange_boundary()</a>	56
7.16.3.2	<a href="#">setup_local_matrices()</a>	57
7.16.3.3	<a href="#">setup_windows()</a>	60
7.16.3.4	<a href="#">update_boundary()</a>	63
7.17	<a href="#">UmfpackError Class Reference</a>	64
7.17.1	<a href="#">Detailed Description</a>	64
7.17.2	<a href="#">Constructor &amp; Destructor Documentation</a>	64
7.17.2.1	<a href="#">UmfpackError()</a>	64
7.18	<a href="#">schwz::Utils&lt; ValueType, IndexType &gt; Struct Template Reference</a>	65
7.18.1	<a href="#">Detailed Description</a>	65

# 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 -DSCHWARZ_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 `edgcut` 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.
- `stage_through_host` [bool][false] : Enable staging transfers through host.
- `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 .*
- [schwz::EventHelpers](#)  
*The EventHelper 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 .*
- [EventHelpers](#)  
*The EventHelper 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::EventHelpers Namespace Reference

The EventHelper namespace .

### 6.5.1 Detailed Description

The EventHelper namespace .

event\_helpers

## 6.6 schwz::PartitionTools Namespace Reference

The [PartitionTools](#) namespace .

### 6.6.1 Detailed Description

The [PartitionTools](#) namespace .

part\_tools

## 6.7 schwz::SolverTools Namespace Reference

The [SolverTools](#) namespace .

### 6.7.1 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

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The function name where the error occurred
<i>op_name</i>	The name of the operator
<i>op_num_rows</i>	The row dimension of the operator
<i>op_num_cols</i>	The column dimension of the operator
<i>clarification</i>	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 `stage_through_host` = false  
*Stage the MPI transfers through the host.*
- 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 (c9f1d38)

## 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.*
- int [num\\_rcv](#)  
*The total number of elements received from all neighbors.*
- int [num\\_send](#)  
*The total number of elements sent to all neighbors.*
- std::shared\_ptr< gko::Array< IndexType > > [neighbors\\_in](#)  
*The neighbors this subdomain has to receive data from.*
- std::shared\_ptr< gko::Array< IndexType > > [neighbors\\_out](#)  
*The neighbors this subdomain has to send data to.*
- std::vector< bool > [is\\_local\\_neighbor](#)  
*The bool vector which is true if the neighbors of a subdomain are in one node.*
- int [local\\_num\\_neighbors\\_in](#)  
*The number of neighbors this subdomain has to receive data from.*
- int [local\\_num\\_neighbors\\_out](#)  
*The number of neighbors this subdomain has to send data to.*
- std::shared\_ptr< gko::Array< IndexType > > [local\\_neighbors\\_in](#)  
*The neighbors this subdomain has to receive data from.*
- std::shared\_ptr< gko::Array< IndexType > > [local\\_neighbors\\_out](#)  
*The neighbors this subdomain has to send data to.*
- std::shared\_ptr< gko::Array< IndexType \* > > [global\\_put](#)  
*The array containing the number of elements that each subdomain sends from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [local\\_put](#)  
*The array containing the number of elements that each subdomain sends from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [global\\_get](#)  
*The array containing the number of elements that each subdomain gets from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [local\\_get](#)  
*The array containing the number of elements that each subdomain gets from the other.*
- std::vector< IndexType > [send](#)

- The number of elements being sent to each subdomain.*

  - `std::vector< IndexType > recv`
- The number of elements being sent to each subdomain.*

  - `std::shared_ptr< gko::Array< IndexType > > window\_ids`

*The RDMA window ids.*
- The RDMA window ids to receive data from.*

  - `std::shared_ptr< gko::Array< IndexType > > windows\_from`
- The RDMA window ids to send data to.*

  - `std::shared_ptr< gko::Array< IndexType > > windows\_to`
- The put request array.*

  - `std::shared_ptr< gko::Array< MPI_Request > > put\_request`
- The get request array.*

  - `std::shared_ptr< gko::Array< MPI_Request > > get\_request`
- The send buffer used for the actual communication for both one-sided and two-sided (always allocated).*

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

  - `std::shared_ptr< gko::matrix::Dense< MixedValueType > > mixedt\_send\_buffer`
- The recv buffer used for the actual communication for both one-sided and two-sided (always allocated).*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > recv\_buffer`
- The mixed precision recv buffer used for the actual communication for both one-sided and two-sided.*

  - `std::shared_ptr< gko::matrix::Dense< MixedValueType > > mixedt\_recv\_buffer`
- The extrapolation buffer used for extrapolation of values at the receiver.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > extra\_buffer`
- The last received boundary values for each of the in neighbors for extrapolation.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > last\_rcv\_bdy`
- Average of values in the send buffer for each of the out neighbors.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > curr\_send\_avg`
- Average of values in the last send buffer for each of the out neighbors.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > last\_send\_avg`
- Average of values in the recv buffer for each of the out neighbors.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > curr\_rcv\_avg`
- Average of values in the last recv buffer for each of the out neighbors.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > last\_rcv\_avg`
- Number of messages sent.*

  - `std::shared_ptr< gko::Array< IndexType > > msg\_count`
- Iteration stamp of last received values.*

  - `std::shared_ptr< gko::Array< IndexType > > last\_rcv\_iter`
- Last rcv slopes.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > last\_rcv\_slopes`
- Last sent slopes.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > > last\_sent\_slopes\_avg`
- Iteration stamp of last received values.*

  - `std::shared_ptr< gko::Array< IndexType > > last\_sent\_iter`
- Threshold.*

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

  - `std::shared_ptr< gko::Array< IndexType > > get\_displacements`
- The displacements for the sending of the buffer.*

  - `std::shared_ptr< gko::Array< IndexType > > put\_displacements`
- The RDMA window for the recv buffer.*

  - `MPI_Win window\_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 >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType, Mixed←
ValueType >::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 >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType, Mixed←
ValueType >::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 >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType, Mixed←
ValueType >::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 >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType, Mixed←
ValueType >::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 (c9f1d38)



## 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, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_solution, std::shared\_ptr< gko::matrix::Dense< ValueType >> &prev\_event\_solution, std::ofstream &fps, std::ofstream &fpr)=0  
*Exchanges the elements of the solution vector.*
- void [local\\_to\\_global\\_vector](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, const std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_vector, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_vector)  
*Transforms data from a local vector to a global vector.*
- virtual void [update\\_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_solution, const std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_rhs, const std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_solution, const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &interface\_matrix)=0  
*Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.*
- void [clear](#) ([Settings](#) &settings)  
*Clears the data.*

#### 7.4.1 Detailed Description

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

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

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	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,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & prev_event_solution,
    std::ofstream & fps,
    std::ofstream & fpr ) [pure virtual]
```

Exchanges the elements of the solution vector.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>global_solution</i>	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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_vector</i>	The local vector in question.
<i>global_vector</i>	The global vector in question.

Referenced by [schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::run\(\)](#).

```

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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>main_buffer</i>	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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_solution</i>	The local solution vector in the subdomain.
<i>local_rhs</i>	The local right hand side vector in the subdomain.
<i>global_solution</i>	The workspace solution vector.
<i>global_old_solution</i>	The global solution vector of the previous iteration.
<i>interface_matrix</i>	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](#) (c9f1d38)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp](#) (c9f1d38)

## 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](#) (c9f1d38)

## 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

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the CUDA routine that failed
<i>error_code</i>	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 CusparsedError Class Reference

[CusparsedError](#) is thrown when a cuSPARSE routine throws a non-zero error code.

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

### Public Member Functions

- [CusparsedError](#) (const std::string &file, int line, const std::string &func, int error\_code)  
*Initializes a cuSPARSE error.*

### 7.7.1 Detailed Description

[CusparsedError](#) is thrown when a cuSPARSE routine throws a non-zero error code.

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 CusparsedError()

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

Initializes a cuSPARSE error.

##### Parameters

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the cuSPARSE routine that failed
<i>error_code</i>	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 [generate\\_dipole\\_rhs](#) (std::vector< ValueType > &rhs)  
*Generates a dipole right hand side vector.*
- void [generate\\_sin\\_rhs](#) (std::vector< ValueType > &rhs)  
*Generates a sinusoidal right hand side vector.*
- void [setup\\_global\\_matrix](#) (const std::string &filename, const gko::size\_type &oned\_laplacian\_size, std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &global\_matrix)  
*Generates the 2D global laplacian matrix.*
- void [partition](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &global\_matrix, std::vector< unsigned int > &partition\_indices)  
*The partitioning function.*
- void [setup\\_vectors](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::vector< ValueType > &rhs, std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_rhs, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_rhs, std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_solution)  
*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

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

## Initialization

## 7.9.2 Member Function Documentation

## 7.9.2.1 generate\_dipole\_rhs()

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

Generates a dipole right hand side vector.

## Parameters

<i>rhs</i>	The rhs vector.
------------	-----------------

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

```
101 {
102     auto oned_laplacian_size = metadata.oned_laplacian_size;
103
104     // Placing dipole at 1/4 and 3/4 of Y-dim at the middle of X-dim
105     for (int i = 0; i < oned_laplacian_size; i++) {
106         for (int j = 0; j < oned_laplacian_size; j++) {
107             if (i == oned_laplacian_size / 4 && j == oned_laplacian_size / 2)
108                 rhs[i * oned_laplacian_size + j] = 100.0;
109             else if (i == 3 * oned_laplacian_size / 4 &&
110                     j == oned_laplacian_size / 2)
111                 rhs[i * oned_laplacian_size + j] = -100.0;
112             else
113                 rhs[i * oned_laplacian_size + j] = 0.0;
114         }
115     }
116 }
```

## 7.9.2.2 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

<i>rhs</i>	The rhs vector.
------------	-----------------

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 }
```

## 7.9.2.3 generate\_sin\_rhs()

```

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

Generates a sinusoidal right hand side vector.

## Parameters

<i>rhs</i>	The rhs vector.
------------	-----------------

References schwz::Initialize< ValueType, IndexType >::setup\_global\_matrix().

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

```

121 {
122     auto PI = (ValueType)(atan(1.0) * 4);
123     auto oned_laplacian_size = metadata.oned_laplacian_size;
124
125     // Source = sin(x)sin(y)
126     for (int i = 0; i < oned_laplacian_size; i++) {
127         for (int j = 0; j < oned_laplacian_size; j++) {
128             rhs[i * oned_laplacian_size + j] =
129                 sin(2 * PI * i / oned_laplacian_size) *
130                 sin(2 * PI * j / oned_laplacian_size);
131         }
132     }
133 }
```

## 7.9.2.4 partition()

```

template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::partition (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
```

```
const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_↵  
matrix,  
std::vector< unsigned int > & partition_indices )
```

The partitioning function.

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

## Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>global_matrix</i>	The global matrix.
<i>partition_indices</i>	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().

```

320 {
321     partition_indices.resize(metadata.global_size);
322     if (metadata.my_rank == 0) {
323         auto partition_settings =
324             (Settings::partition_settings::partition_zoltan |
325              Settings::partition_settings::partition_metis |
326              Settings::partition_settings::partition_regular |
327              Settings::partition_settings::partition_regular2d |
328              Settings::partition_settings::partition_custom) &
329             settings.partition;
330
331         if (partition_settings ==
332             Settings::partition_settings::partition_zoltan) {
333             SCHWARZ_NOT_IMPLEMENTED;
334         } else if (partition_settings ==
335                     Settings::partition_settings::partition_metis) {
336             if (metadata.my_rank == 0) {
337                 std::cout << " METIS partition" << std::endl;
338             }
339             PartitionTools::PartitionMetis(
340                 settings, global_matrix, this->cell_weights,
341                 metadata.num_subdomains, partition_indices);
342         } else if (partition_settings ==
343                     Settings::partition_settings::partition_regular) {
344             if (metadata.my_rank == 0) {
345                 std::cout << " Regular 1D partition" << std::endl;
346             }
347             PartitionTools::PartitionRegular(
348                 global_matrix, metadata.num_subdomains, partition_indices);
349         } else if (partition_settings ==
350                     Settings::partition_settings::partition_regular2d) {
351             if (metadata.my_rank == 0) {
352                 std::cout << " Regular 2D partition" << std::endl;
353             }
354             PartitionTools::PartitionRegular2D(
355                 global_matrix, settings.write_debug_out,
356                 metadata.num_subdomains, partition_indices);
357         } else if (partition_settings ==
358                     Settings::partition_settings::partition_custom) {
359             // User partitions mesh manually
360             SCHWARZ_NOT_IMPLEMENTED;
361         } else {
362             SCHWARZ_NOT_IMPLEMENTED;
363         }
364     }
365 }
```

## 7.9.2.5 setup\_global\_matrix()

```

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

Generates the 2D global laplacian matrix.

## Parameters

<i>oned_laplacian_size</i>	The size of the one d laplacian grid.
<i>global_matrix</i>	The global matrix.

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

```

236 {
237     using index_type = IndexType;
238     using value_type = ValueType;
239     using mtx = gko::matrix::Csr<value_type, index_type>;
240     if (settings.matrix_filename != "null") {
241         auto input_file = std::ifstream(filename, std::ios::in);
242         if (!input_file) {
243             std::cerr << "Could not find the file \"" << filename
244                 << "\", which is required for this test.\n";
245         }
246         global_matrix =
247             gko::read<mtx>(input_file, settings.executor->get_master());
248         global_matrix->sort_by_column_index();
249         std::cout << "Matrix from file " << filename << std::endl;
250     } else if (settings.matrix_filename == "null" &&
251         settings.explicit_laplacian) {
252         std::cout << "Laplacian 2D Matrix (generated in house) " << std::endl;
253         gko::size_type global_size = oned_laplacian_size *
254             oned_laplacian_size;
255         global_matrix = mtx::create(settings.executor->get_master(),
256             gko::dim<2>(global_size), 5 * global_size);
257         value_type *values = global_matrix->get_values();
258         index_type *row_ptrs = global_matrix->get_row_ptrs();
259         index_type *col_idxs = global_matrix->get_col_idxs();
260
261         std::vector<gko::size_type> exclusion_set;
262
263         std::map<IndexType, ValueType> stencil_map = {
264             {-oned_laplacian_size, -1}, {-1, -1}, {0, 4}, {1, -1},
265             {oned_laplacian_size, -1},
266         };
267         for (auto i = 2; i < global_size; ++i) {
268             gko::size_type index = (i - 1) * oned_laplacian_size;
269             if (index * index < global_size * global_size) {
270                 exclusion_set.push_back(
271                     linearize_index(index, index - 1, global_size));
272                 exclusion_set.push_back(
273                     linearize_index(index - 1, index, global_size));
274             }
275         }
276
277         std::sort(exclusion_set.begin(),
278             exclusion_set.begin() + exclusion_set.size());
279
280         IndexType pos = 0;
281         IndexType col_idx = 0;
282         row_ptrs[0] = pos;
283         gko::size_type cur_idx = 0;
284         for (IndexType i = 0; i < global_size; ++i) {
285             for (auto ofs : stencil_map) {
286                 auto in_exclusion_flag =
287                     (exclusion_set[cur_idx] ==
288                     linearize_index(i, i + ofs.first, global_size));
289                 if (0 <= i + ofs.first && i + ofs.first < global_size &&
290                     !in_exclusion_flag) {
291                     values[pos] = ofs.second;
292                     col_idxs[pos] = i + ofs.first;
293                     ++pos;
294                 }
295                 if (in_exclusion_flag) {
296                     cur_idx++;
297                 }
298                 col_idx = row_ptrs[i + 1] - pos;
299             }
300             row_ptrs[i + 1] = pos;
301         }
302     } else {
303         std::cerr << " Need to provide a matrix or enable the default "
304             "laplacian matrix."
305             << std::endl;
306         std::exit(-1);
307     }
308 }

```

## 7.9.2.6 setup\_local\_matrices()

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

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

## Parameters

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

Implemented in [schwz::SolverRAS< ValueType, IndexType, MixedValueType >](#).

Referenced by [schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::initialize\(\)](#).

## 7.9.2.7 setup\_vectors()

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

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

## Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_rhs</i>	The local right hand side vector in the subdomain.
<i>global_rhs</i>	The global right hand side vector.
<i>local_solution</i>	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()`.

```

375 {
376     using vec = gko::matrix::Dense<ValueType>;
377     auto my_rank = metadata.my_rank;
378     auto first_row = metadata.first_row->get_data()[my_rank];
379
380     // Copy the global rhs vector to the required executor.
381     gko::Array<ValueType> temp_rhs{settings.executor->get_master(), rhs.begin(),
382                                   rhs.end()};
383     global_rhs = vec::create(settings.executor,
384                              gko::dim<2>{metadata.global_size, 1}, temp_rhs, 1);
385
386     local_rhs =
387         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
388     // Extract the local rhs from the global rhs. Also takes into account the
389     // overlap.
390     SolverTools::extract_local_vector(settings, metadata, local_rhs.get(),
391                                       global_rhs.get(), first_row);
392
393     local_solution =
394         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
395 }
```

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

- `initialization.hpp` (c9f1d38)
- `/home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp` (c9f1d38)

## 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.*

  - ValueType `constant` = 0.0
- Value of constant for event threshold Relevant for cgammak threshold.*

  - ValueType `gamma` = 0.0
- Value of gamma for event threshold Relevant for cgammak threshold.*

  - ValueType `horizon` = 0.0
- Value of horizon for the event threshold Relevant for slope-based threshold.*

  - ValueType `decay_param` = 0.0
- Value of decay parameter for the event threshold Relevant for slope-based threshold.*

  - IndexType `sent_history` = 0
- Value of history at the sender.*

  - IndexType `recv_history` = 0
- Value of history at the receiver.*

  - IndexType `comm_start_iters` = 0
- Number of iterations to communicate before event comm.*

  - std::vector< std::tuple< int, int, int, std::string, std::vector< ValueType > > > `time_struct`
- The struct used to measure the timings of each function within the solver loop.*

- `std::vector< std::tuple< int, std::vector< std::tuple< int, int > >, std::vector< std::tuple< int, int > >, int, int > >` [comm\\_data\\_struct](#)  
The struct used to measure the timings of each function within the solver loop.
- `std::shared_ptr< gko::Array< IndexType > >` [global\\_to\\_local](#)  
The mapping containing the global to local indices.
- `std::shared_ptr< gko::Array< IndexType > >` [local\\_to\\_global](#)  
The mapping containing the local to global indices.
- `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

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	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 (c9f1d38)



## 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

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the METIS routine that failed
<i>error_code</i>	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 (c9f1d38)

## 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 > >` [prev\\_event\\_solution](#)  
*The (local+overlap) solution vector at time of previous event of communication The size of this vector is considered global\_size to account for overlap.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [global\\_solution](#)  
*The global solution vector.*
- `std::vector< ValueType >` [local\\_residual\\_vector\\_out](#)  
*The global residual vector.*
- `std::vector< std::vector< ValueType > >` [global\\_residual\\_vector\\_out](#)  
*The local residual vector.*

## Additional Inherited Members

### 7.13.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32, 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

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	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

<i>settings</i>	The settings struct.
<i>metadata</i>	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     } else if (settings.executor_string == "cuda") {
93         int num_devices = 0;
94         #if SCHW_HAVE_CUDA
95         SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
96         #else
97         SCHWARZ_NOT_IMPLEMENTED;
98         #endif
99         Utils<ValueType, IndexType>::assert_correct_cuda_devices(
100             num_devices, metadata.my_rank);
101         settings.executor = gko::CudaExecutor::create(
102             my_local_rank, gko::OmpExecutor::create(), false);
103         auto exec_info = static_cast<gko::OmpExecutor *>(
104             settings.executor->get_master().get())
105             ->get_exec_info();
106         exec_info->bind_to_core(my_local_rank);
107         settings.cuda_device_guard =
108             std::make_shared<schwz::device_guard>(my_local_rank);
109
110         std::cout << " Rank " << metadata.my_rank << " with local rank "
111             << my_local_rank << " has "
112             << (static_cast<gko::CudaExecutor *>(settings.executor.get()))
113             ->get_device_id()
114             << " id of gpu" << std::endl;
115         MPI_Barrier(metadata.mpi_communicator);
116     } else if (settings.executor_string == "reference") {
117         settings.executor = gko::ReferenceExecutor::create();
118         auto exec_info =
119             static_cast<gko::ReferenceExecutor *>(settings.executor.get())
120             ->get_exec_info();
121         exec_info->bind_to_core(my_local_rank);
122     }
123 }
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

<i>matrix</i>	The matrix to be printed.
<i>subd</i>	The subdomain on which the vector exists.
<i>name</i>	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

<i>vector</i>	The vector to be printed.
<i>subd</i>	The subdomain on which the vector exists.
<i>name</i>	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

<i>solution</i>	The solution vector.
-----------------	----------------------

References `schwz::Settings::debug_print`, `schwz::Communicate< ValueType, IndexType, MixedValueType >::exchange_boundary()`, `schwz::Settings::executor`, `schwz::Settings::executor_string`, `schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::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, IndexType, MixedValueType >::local_rhs`, `schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::local_solution`, `schwz::Communicate< ValueType, IndexType, MixedValueType >::local_to_global_vector()`, `schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::msg_count`, `schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::neighbors_out`, `schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::num_neighbors_out`, `schwz::Settings::overlap`, `schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::prev_event_solution`, `schwz::Communicate< ValueType, IndexType, MixedValueType >::setup_windows()`, `schwz::Settings::comm_settings::stage_through_host`, `schwz::Settings::thres_type`, `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`.

```

335 {
336     using vec_vtype = gko::matrix::Dense<ValueType>;
337     if (!solution.get()) {
338         solution =
339             vec_vtype::create(settings.executor->get_master(),
340                             gko::dim<2>(this->metadata.global_size, 1));
341     }
342     MixedValueType dummy1 = 0.0;
343     ValueType dummy2 = 1.0;
344
345     auto num_neighbors_out = this->comm_struct.num_neighbors_out;
346     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
347
348     if (metadata.my_rank == 0) {
349         std::cout << " MixedValueType: " << typeid(dummy1).name()
350                 << " ValueType: " << typeid(dummy2).name() << std::endl;
351     }
352     // The main solution vector
353     std::shared_ptr<vec_vtype> global_solution = vec_vtype::create(
354         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
355     // The main solution vector on the host
356     std::shared_ptr<vec_vtype> host_global_solution;
357     if (settings.comm_settings.stage_through_host) {
358         host_global_solution =
359             vec_vtype::create(this->settings.executor->get_master(),
360                             gko::dim<2>(this->metadata.global_size, 1));
361     }
362     // The solution vector at the previous event of communication
363     std::shared_ptr<vec_vtype> prev_event_solution = vec_vtype::create(
364         settings.executor, gko::dim<2>(metadata.global_size, 1));
365
366     // A work vector.
367     std::shared_ptr<vec_vtype> work_vector = vec_vtype::create(
368         settings.executor, gko::dim<2>(2 * this->metadata.local_size_x, 1));
369
370     // An initial guess.
371     std::shared_ptr<vec_vtype> init_guess = vec_vtype::create(
372         settings.executor, gko::dim<2>(this->metadata.local_size_x, 1));
373     // init_guess->copy_from(local_rhs.get());
374
375     if (settings.executor_string == "omp" && settings.debug_print) {
376         ValueType sum_rhs = std::accumulate(
377             local_rhs->get_values(),
378             local_rhs->get_values() + local_rhs->get_size()[0], 0.0);
379         std::cout << " Rank " << this->metadata.my_rank << " sum local rhs "
380                 << sum_rhs << std::endl;
381     }
382
383     // Setup the windows for the onesided communication.
384     if (settings.comm_settings.stage_through_host) {

```

```

385         this->setup_windows(this->settings, this->metadata,
386                             host_global_solution);
387     } else {
388         this->setup_windows(this->settings, this->metadata, global_solution);
389     }
390
391     const auto solver_settings =
392         (Settings::local_solver_settings::direct_solver_cholmod |
393          Settings::local_solver_settings::direct_solver_umfpack |
394          Settings::local_solver_settings::direct_solver_ginkgo |
395          Settings::local_solver_settings::iterative_solver_dealii |
396          Settings::local_solver_settings::iterative_solver_ginkgo) &
397         settings.local_solver;
398     if (settings.comm_settings.stage_through_host) {
399         host_global_solution->copy_from(gko::lend(global_solution));
400     }
401
402     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
403             global_residual_norm = 0.0, global_residual_norm0 = -1.0;
404     metadata.iter_count = 0;
405     int num_converged_procs = 0;
406
407     std::string rank_string = std::to_string(metadata.my_rank);
408     if (metadata.my_rank < 10) {
409         rank_string = "0" + std::to_string(metadata.my_rank);
410     }
411
412     std::ofstream fps; // file for sending log
413     std::ofstream fpr; // file for receiving log
414     if (settings.debug_print) {
415         // Opening files for event logs
416         fps.open("send" + rank_string + ".txt");
417         fpr.open("recv" + rank_string + ".txt");
418     }
419
420     if (metadata.my_rank == 0) {
421         std::cout << "Send history - " << metadata.sent_history
422                 << ", Recv history - " << metadata.recv_history << std::endl;
423         std::cout << "Thres type - " << settings.thres_type << std::endl;
424         std::cout << "Overlap - " << settings.overlap << std::endl;
425     }
426
427     auto start_time = std::chrono::steady_clock::now();
428
429     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
430         // Exchange the boundary values. The communication part.
431         if (settings.comm_settings.stage_through_host) {
432             host_global_solution->copy_from(gko::lend(global_solution));
433             // By staging through host just transfer the host_global_solution
434             // instead of on device global_solution
435             MEASURE_ELAPSED_FUNC_TIME(
436                 this->exchange_boundary(settings, metadata,
437                                         host_global_solution,
438                                         prev_event_solution, fps, fpr),
439                 0, metadata.my_rank, boundary_exchange, metadata.iter_count);
440             global_solution->copy_from(gko::lend(host_global_solution));
441         } else {
442             MEASURE_ELAPSED_FUNC_TIME(
443                 this->exchange_boundary(settings, metadata, global_solution,
444                                         prev_event_solution, fps, fpr),
445                 0, metadata.my_rank, boundary_exchange, metadata.iter_count);
446         }
447
448         // Update the boundary and interior values after the exchanging from
449         // other processes.
450         MEASURE_ELAPSED_FUNC_TIME(
451             this->update_boundary(settings, metadata, this->
452 local_solution,
453                                     this->local_rhs, global_solution,
454                                     this->interface_matrix),
455             1, metadata.my_rank, boundary_update, metadata.iter_count);
456
457         // Check for the convergence of the solver.
458         // num_converged_procs = 0;
459         MEASURE_ELAPSED_FUNC_TIME(
460             (Solve<ValueType, IndexType, MixedValueType>::check_convergence(
461                 settings, metadata, this->comm_struct, this->convergence_vector,
462                 global_solution, this->local_solution, this->
463 local_matrix,
464                                     work_vector, local_residual_norm, local_residual_norm0,
465                                     global_residual_norm, global_residual_norm0,
466                                     num_converged_procs)),
467             2, metadata.my_rank, convergence_check, metadata.iter_count);
468
469         // break if the solution diverges.
470         if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
471             std::cout << " Rank " << metadata.my_rank << " diverged in "

```

```

470         << metadata.iter_count << " iters " << std::endl;
471         std::exit(-1);
472     }
473
474     // break if all processes detect that all other processes have
475     // converged otherwise continue iterations.
476     if (num_converged_procs == metadata.num_subdomains) {
477         break;
478     } else {
479         MEASURE_ELAPSED_FUNC_TIME(
480             (Solve<ValueType, IndexType, MixedValueType>::local_solve(
481                 settings, metadata, this->local_matrix,
482                 this->triangular_factor_l, this->
triangular_factor_u,
483                 this->local_perm, this->local_inv_perm, work_vector,
484                 init_guess, this->local_solution)),
485             3, metadata.my_rank, local_solve, metadata.iter_count);
486
487         // Gather the local vector into the locally global vector for
488         // communication.
489         MEASURE_ELAPSED_FUNC_TIME(
490             (Communicate<ValueType, IndexType, MixedValueType>::
491                 local_to_global_vector(settings, metadata,
492                 this->local_solution,
493                 global_solution)),
494             4, metadata.my_rank, expand_local_vec, metadata.iter_count);
495     }
496 }
497 MPI_Barrier(MPI_COMM_WORLD);
498 auto elapsed_time = std::chrono::duration<ValueType>(
499     std::chrono::steady_clock::now() - start_time);
500
501 if (settings.debug_print) {
502     // Closing event log files
503     fps.close();
504     fpr.close();
505 }
506
507 // adding 1 to include the 0-th iteration
508 // metadata.iter_count = metadata.iter_count + 1;
509
510 // number of messages a PE would send without event-based
511 int noevent_msg_count = metadata.iter_count * num_neighbors_out;
512
513 int total_events = 0;
514
515 // Printing msg count
516 if (settings.debug_print) {
517     for (int k = 0; k < num_neighbors_out; k++) {
518         std::cout << " Rank: " << metadata.my_rank << " to "
519             << neighbors_out[k] << " : "
520             << this->comm_struct.msg_count->get_data()[k];
521         total_events += this->comm_struct.msg_count->get_data()[k];
522     }
523     std::cout << std::endl;
524 }
525
526 // Total no of messages in all PEs
527 MPI_Allreduce(MPI_IN_PLACE, &total_events, 1, MPI_INT, MPI_SUM,
528     MPI_COMM_WORLD);
529 MPI_Allreduce(MPI_IN_PLACE, &noevent_msg_count, 1, MPI_INT, MPI_SUM,
530     MPI_COMM_WORLD);
531
532 if (metadata.my_rank == 0) {
533     std::cout << "Total number of events - " << total_events << std::endl;
534     std::cout << "Total number of msgs without event - "
535         << noevent_msg_count << std::endl;
536 }
537
538 // Write the residuals and iterations to files
539 if (settings.write_iters_and_residuals &&
540     solver_settings ==
541     Settings::local_solver_settings::iterative_solver_ginkgo) {
542     std::string filename = "iter_res_" + rank_string + ".csv";
543     write_iters_and_residuals(
544         metadata.num_subdomains, metadata.my_rank,
545         metadata.post_process_data.local_residual_vector_out.size(),
546         metadata.post_process_data.local_residual_vector_out,
547         metadata.post_process_data.local_converged_iter_count,
548         metadata.post_process_data.local_converged_resnorm,
549         metadata.post_process_data.local_timestamp, filename);
550 }
551 if (num_converged_procs < metadata.num_subdomains) {
552     std::cout << "Rank " << metadata.my_rank << " did not converge in "
553         << metadata.iter_count << " iterations." << std::endl;
554 } else {
555     std::cout << " Rank " << metadata.my_rank << " converged in "

```



```

556         << metadata.iter_count << " iterations " << std::endl;
557     ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
558         residual_norm = -1.0;
559
560     // Compute the final residual norm. Also gathers the solution from all
561     // subdomains.
562     Solve<ValueType, IndexType, MixedValueType>::compute_residual_norm(
563         settings, metadata, global_matrix, global_rhs, global_solution,
564         mat_norm, rhs_norm, sol_norm, residual_norm);
565     gather_comm_data<ValueType, IndexType, MixedValueType>(
566         metadata.num_subdomains, this->comm_struct,
567         metadata.comm_data_struct);
568     // clang-format off
569     if (metadata.my_rank == 0)
570     {
571         std::cout
572             << " residual norm " << residual_norm << "\n"
573             << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
574             << " Time taken for solve " << elapsed_time.count()
575             << std::endl;
576     }
577     // clang-format on
578 }
579 if (metadata.my_rank == 0) {
580     solution->copy_from(global_solution.get());
581 }
582
583 // Communicate<ValueType, IndexType>::clear(settings);
584 }

```

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

- schwarz\_base.hpp (c9f1d38)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz\_base.cpp (2f81ed4)

## 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.*
- `std::string rhs_type = "ones"`  
*Flag to enable a random rhs.*
- `std::string thres_type = "cgammak"`  
*Flag to choose thres type.*
- `std::string norm_type = "L1"`  
*Flag to choose norm type.*
- `bool print_matrices = false`  
*Flag to enable printing of matrices.*
- `bool debug_print = false`  
*Flag to enable some debug printing.*
- `bool non_symmetric_matrix = false`  
*Is the matrix non-symmetric ? , Use GMRES for local solves.*
- `int restart_iter = 1`  
*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.

#### 7.14.2.4 norm\_type

```
std::string schwz::Settings::norm_type = "L1"
```

Flag to choose norm type.

Choices are "L1" or "L2"

Referenced by `schwz::SolverRAS< ValueType, IndexType, MixedValueType >::setup_windows()`.

### 7.14.2.5 thres\_type

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

Flag to choose thres type.

Choices are "cgammak" or "slope"

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

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

- settings.hpp (c9f1d38)

## 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

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

### [Solve](#)

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

- solve.hpp (c9f1d38)
- /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, std::shared\_ptr< gko::matrix::Dense< ValueType >> &[global\\_solution](#), std::shared\_ptr< gko::matrix::Dense< ValueType >> &[prev\\_event\\_solution](#), std::ofstream &fps, std::ofstream &fpr) override  
*Exchanges the elements of the solution vector.*
- void [update\\_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &[local\\_solution](#), const std::shared\_ptr< gko::matrix::Dense< ValueType >> &[local\\_rhs](#), const std::shared\_ptr< gko::matrix::Dense< ValueType >> &[global\\_solution](#), const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &[interface\\_matrix](#)) override  
*Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.*

### Additional Inherited Members

#### 7.16.1 Detailed Description

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

An implementation of the solver interface using the RAS solver.

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>data</i>	The additional data struct.

```
52      : SchwarzBase<ValueType, IndexType, MixedValueType>(settings, metadata)
53 {}
```

## 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,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & prev_event_solution,
    std::ofstream & fps,
    std::ofstream & fpr ) [override], [virtual]
```

Exchanges the elements of the solution vector.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>global_solution</i>	The solution vector being exchanged between the subdomains.

Implements [schwz::Communicate< ValueType, IndexType, MixedValueType >](#).

References [schwz::Settings::comm\\_settings::enable\\_onesided](#), [schwz::SchwarzBase< ValueType, IndexType,](#)

MixedValueType >::global\_solution, and schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::prev↵\_event\_solution.

```

1257 {
1258     if (settings.comm_settings.enable_onesided) {
1259         exchange_boundary_onesided<ValueType, IndexType, MixedValueType>(
1260             settings, metadata, this->comm_struct, global_solution,
1261             prev_event_solution, fps, fpr);
1262     } else {
1263         exchange_boundary_twosided<ValueType, IndexType, MixedValueType>(
1264             settings, metadata, this->comm_struct, global_solution);
1265     }
1266 }

```

### 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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>partition_indices</i>	The array containing the partition indices.
<i>global_matrix</i>	The global system matrix.
<i>local_matrix</i>	The local system matrix.
<i>interface_matrix</i>	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.
<i>local_perm</i>	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, IndexType >::local\\_to\\_global](#), [schwz::Metadata< ValueType, IndexType >::my\\_rank](#), [schwz::Metadata< ValueType, IndexType >::num\\_subdomains](#), [schwz::Settings::overlap](#), [schwz::Metadata< ValueType, IndexType >::overlap↵\\_row](#), [schwz::Metadata< ValueType, IndexType >::overlap\\_size](#), and [schwz::Metadata< ValueType, IndexType >::permutation](#).

```

63 {
64     using mtx = gko::matrix::Csr<ValueType, IndexType>;
65     using vec_type = gko::Array<IndexType>;
66     using perm_type = gko::matrix::Permutation<IndexType>;
67     using arr = gko::Array<IndexType>;
68     auto my_rank = metadata.my_rank;
69     auto comm_size = metadata.comm_size;
70     auto num_subdomains = metadata.num_subdomains;
71     auto global_size = metadata.global_size;
72     auto mpi_itype = schwz::mpi::get_mpi_datatype(*partition_indices.data());
73
74     MPI_Bcast(partition_indices.data(), global_size, mpi_itype, 0,
75             MPI_COMM_WORLD);
76
77     std::vector<IndexType> local_p_size(num_subdomains);
78     auto global_to_local = metadata.global_to_local->get_data();
79     auto local_to_global = metadata.local_to_global->get_data();
80
81     auto first_row = metadata.first_row->get_data();
82     auto permutation = metadata.permutation->get_data();
83     auto i_permutation = metadata.i_permutation->get_data();
84
85     auto nb = (global_size + num_subdomains - 1) /
num_subdomains;
86     auto partition_settings =
87     (Settings::partition_settings::partition_zoltan |
88     Settings::partition_settings::partitionmetis |
89     Settings::partition_settings::partition_regular |
90     Settings::partition_settings::partition_regular2d |
91     Settings::partition_settings::partition_custom) &
92     settings.partition;
93
94     IndexType *gmat_row_ptrs = global_matrix->get_row_ptrs();
95     IndexType *gmat_col_idxs = global_matrix->get_col_idxs();
96     ValueType *gmat_values = global_matrix->get_values();
97
98     // default local p size set for 1 subdomain.
99     first_row[0] = 0;
100     for (auto p = 0; p < num_subdomains; ++p) {
101         local_p_size[p] = std::min(global_size - first_row[p], nb);
102         first_row[p + 1] = first_row[p] + local_p_size[p];
103     }
104
105     if (partition_settings == Settings::partition_settings::partitionmetis ||
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
136     auto gmat_temp = mtx::create(settings.executor->get_master(),
137                                 global_matrix->get_size(),
138                                 global_matrix->get_num_stored_elements());
139
140     auto nnz = 0;
141     gmat_temp->get_row_ptrs()[0] = 0;
142     for (auto row = 0; row < metadata.global_size; ++row) {
143         for (auto col = gmat_row_ptrs[permutation[row]];
144              col < gmat_row_ptrs[permutation[row] + 1]; ++col) {
145             gmat_temp->get_col_idxs()[nnz] =
146                 i_permutation[gmat_col_idxs[col]];
147             gmat_temp->get_values()[nnz] = gmat_values[col];
148             nnz++;

```



```

149         }
150         gmat_temp->get_row_ptrs()[row + 1] = nnz;
151     }
152     global_matrix->copy_from(gmat_temp.get());
153 }
154
155
156 for (auto i = 0; i < global_size; i++) {
157     global_to_local[i] = 0;
158     local_to_global[i] = 0;
159 }
160 auto num = 0;
161 for (auto i = first_row[my_rank]; i < first_row[
my_rank + 1]; i++) {
162     global_to_local[i] = 1 + num;
163     local_to_global[num] = i;
164     num++;
165 }
166
167 IndexType old = 0;
168 for (auto k = 1; k < settings.overlap; k++) {
169     auto now = num;
170     for (auto i = old; i < now; i++) {
171         for (auto j = gmat_row_ptrs[local_to_global[i]];
172             j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
173             if (global_to_local[gmat_col_idxs[j]] == 0) {
174                 local_to_global[num] = gmat_col_idxs[j];
175                 global_to_local[gmat_col_idxs[j]] = 1 + num;
176                 num++;
177             }
178         }
179     }
180     old = now;
181 }
182 metadata.local_size = local_p_size[my_rank];
183 metadata.local_size_x = num;
184 metadata.local_size_o = global_size;
185 auto local_size = metadata.local_size;
186 auto local_size_x = metadata.local_size_x;
187
188 metadata.overlap_size = num - metadata.local_size;
189 auto host_ov_row = gko::Array<IndexType>::view(
190     settings.executor->get_master(), metadata.overlap_size,
191     &(metadata.local_to_global->get_data()[metadata.local_size]));
192 metadata.overlap_row = vec_itype(settings.executor, metadata.overlap_size);
193 metadata.overlap_row = host_ov_row;
194
195 auto nnz_local = 0;
196 auto nnz_interface = 0;
197
198 for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
199     for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; j++) {
200         if (global_to_local[gmat_col_idxs[j]] != 0) {
201             nnz_local++;
202         } else {
203             std::cout << " debug: invalid edge?" << std::endl;
204         }
205     }
206 }
207 auto temp = 0;
208 for (auto k = 0; k < metadata.overlap_size; k++) {
209     temp = host_ov_row.get_data()[k];
210     for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1]; j++) {
211         if (global_to_local[gmat_col_idxs[j]] != 0) {
212             nnz_local++;
213         } else {
214             nnz_interface++;
215         }
216     }
217 }
218
219 std::shared_ptr<mtx> local_matrix_compute;
220 local_matrix_compute = mtx::create(settings.executor->get_master(),
221     gko::dim<2>(local_size_x), nnz_local);
222 IndexType *lmat_row_ptrs = local_matrix_compute->get_row_ptrs();
223 IndexType *lmat_col_idxs = local_matrix_compute->get_col_idxs();
224 ValueType *lmat_values = local_matrix_compute->get_values();
225
226 std::shared_ptr<mtx> interface_matrix_compute;
227 if (nnz_interface > 0) {
228     interface_matrix_compute =
229         mtx::create(settings.executor->get_master(),
230             gko::dim<2>(local_size_x), nnz_interface);
231 } else {
232     interface_matrix_compute = mtx::create(settings.executor->get_master());
233 }
234

```

```

235     IndexType *imat_row_ptrs = interface_matrix_compute->get_row_ptrs();
236     IndexType *imat_col_idxs = interface_matrix_compute->get_col_idxs();
237     ValueType *imat_values = interface_matrix_compute->get_values();
238
239     num = 0;
240     nnz_local = 0;
241     auto nnz_interface_temp = 0;
242     lmat_row_ptrs[0] = nnz_local;
243     if (nnz_interface > 0) {
244         imat_row_ptrs[0] = nnz_interface_temp;
245     }
246     // Local interior matrix
247     for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
248         for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; ++j) {
249             if (global_to_local[gmat_col_idxs[j]] != 0) {
250                 lmat_col_idxs[nnz_local] =
251                     global_to_local[gmat_col_idxs[j]] - 1;
252                 lmat_values[nnz_local] = gmat_values[j];
253                 nnz_local++;
254             }
255         }
256         if (nnz_interface > 0) {
257             imat_row_ptrs[num + 1] = nnz_interface_temp;
258         }
259         lmat_row_ptrs[num + 1] = nnz_local;
260         num++;
261     }
262
263     // Interface matrix
264     if (nnz_interface > 0) {
265         nnz_interface = 0;
266         for (auto k = 0; k < metadata.overlap_size; k++) {
267             temp = host_ov_row.get_data()[k];
268             for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1];
269                 j++) {
270                 if (global_to_local[gmat_col_idxs[j]] != 0) {
271                     lmat_col_idxs[nnz_local] =
272                         global_to_local[gmat_col_idxs[j]] - 1;
273                     lmat_values[nnz_local] = gmat_values[j];
274                     nnz_local++;
275                 } else {
276                     imat_col_idxs[nnz_interface] = gmat_col_idxs[j];
277                     imat_values[nnz_interface] = gmat_values[j];
278                     nnz_interface++;
279                 }
280             }
281             lmat_row_ptrs[num + 1] = nnz_local;
282             imat_row_ptrs[num + 1] = nnz_interface;
283             num++;
284         }
285     }
286     auto now = num;
287     for (auto i = old; i < now; i++) {
288         for (auto j = gmat_row_ptrs[local_to_global[i]];
289             j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
290             if (global_to_local[gmat_col_idxs[j]] == 0) {
291                 local_to_global[num] = gmat_col_idxs[j];
292                 global_to_local[gmat_col_idxs[j]] = 1 + num;
293                 num++;
294             }
295         }
296     }
297
298     local_matrix_compute->sort_by_column_index();
299     interface_matrix_compute->sort_by_column_index();
300
301     local_matrix = mtx::create(settings.executor);
302     local_matrix->copy_from(gko::lend(local_matrix_compute));
303     interface_matrix = mtx::create(settings.executor);
304     interface_matrix->copy_from(gko::lend(interface_matrix_compute));
305 }

```

### 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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>main_buffer</i>	The main buffer being exchanged between the subdomains.

Implements [schwz::Communicate< ValueType, IndexType, MixedValueType >](#).

References [schwz::Metadata< ValueType, IndexType >::comm\\_start\\_iters](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::curr\\_rcv\\_avg](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::curr\\_send\\_avg](#), [schwz::Settings::debug\\_print](#), [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::extra\\_buffer](#), [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, IndexType, MixedValueType >::global\\_solution](#), [schwz::Metadata< ValueType, IndexType >::horizon](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::is\\_local\\_neighbor](#), [schwz::Metadata< ValueType, IndexType >::iter\\_count](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::last\\_rcv\\_avg](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::last\\_rcv\\_iter](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::last\\_send\\_avg](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::last\\_sent\\_iter](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::local\\_get](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::local\\_neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::local\\_neighbors\\_out](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::local\\_num\\_neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::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\\_rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::mixedt\\_send\\_buffer](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::msg\\_count](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::neighbors\\_out](#), [schwz::Settings::norm\\_type](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::num\\_neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::num\\_neighbors\\_out](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::num\\_rcv](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::num\\_send](#), [schwz::Metadata< ValueType, IndexType >::num\\_subdomains](#), [schwz::SchwarzBase< ValueType, IndexType, MixedValueType >::prev\\_event\\_solution](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::put\\_displacements](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::put\\_request](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::send\\_buffer](#), [schwz::Settings::comm\\_settings::stage\\_through\\_host](#), [schwz::Settings::thres\\_type](#), [schwz::Settings::use\\_mixed\\_precision](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::window\\_rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::window\\_send\\_buffer](#), and [schwz::Communicate< ValueType, IndexType, MixedValueType >::comm\\_struct::window\\_x](#).

```
718 {
719     using vec_itype = gko::Array<IndexType>;
```

```

720     using vec_vtype = gko::matrix::Dense<ValueType>;
721     auto num_subdomains = metadata.num_subdomains;
722     auto local_size_o = metadata.local_size_o;
723     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
724     auto global_get = this->comm_struct.global_get->get_data();
725     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
726     auto global_put = this->comm_struct.global_put->get_data();
727
728     // set displacement for the MPI buffer
729     auto get_displacements = this->comm_struct.get_displacements->get_data();
730     auto put_displacements = this->comm_struct.put_displacements->get_data();
731     {
732         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
733         tmp_num_comm_elems[0] = 0;
734         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
735             if ((global_get[j])[0] > 0) {
736                 int p = neighbors_in[j];
737                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
738             }
739         }
740         for (auto j = 0; j < num_subdomains; j++) {
741             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
742         }
743
744         auto mpi_itype = schwz::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
745         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
746                     1, mpi_itype, MPI_COMM_WORLD);
747     }
748
749     {
750         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
751         tmp_num_comm_elems[0] = 0;
752         for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
753             if ((global_put[j])[0] > 0) {
754                 int p = neighbors_out[j];
755                 tmp_num_comm_elems[p + 1] = (global_put[j])[0];
756             }
757         }
758         for (auto j = 0; j < num_subdomains; j++) {
759             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
760         }
761
762         auto mpi_itype = schwz::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
763         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
764                     1, mpi_itype, MPI_COMM_WORLD);
765     }
766
767     // setup windows
768     if (settings.comm_settings.enable_onesided) {
769         // Onesided
770         MPI_Win_create(main_buffer->get_values(),
771                       main_buffer->get_size()[0] * sizeof(ValueType),
772                       sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
773                       &(this->comm_struct.window_x));
774     }
775
776
777     if (settings.comm_settings.enable_onesided) {
778         // MPI_Alloc_mem ? Custom allocator ? TODO
779         MPI_Win_create(this->local_residual_vector->get_values(),
780                       (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
781                       MPI_INFO_NULL, MPI_COMM_WORLD,
782                       &(this->window_residual_vector));
783         std::vector<IndexType> zero_vec(num_subdomains, 0);
784         gko::Array<IndexType> temp_array(settings.executor->get_master(),
785                                         zero_vec.begin(), zero_vec.end());
786         this->convergence_vector = std::shared_ptr<vec_itype>(
787             new vec_itype(settings.executor->get_master(), temp_array),
788             std::default_delete<vec_itype>());
789         this->convergence_sent = std::shared_ptr<vec_itype>(
790             new vec_itype(settings.executor->get_master(), num_subdomains),
791             std::default_delete<vec_itype>());
792         this->convergence_local = std::shared_ptr<vec_itype>(
793             new vec_itype(settings.executor->get_master(), num_subdomains),
794             std::default_delete<vec_itype>());
795         MPI_Win_create(this->convergence_vector->get_data(),
796                       (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
797                       MPI_INFO_NULL, MPI_COMM_WORLD,
798                       &(this->window_convergence));
799     }
800
801     if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
802         // Lock all windows.
803         if (settings.comm_settings.enable_get &&
804             settings.comm_settings.enable_lock_all) {
805             MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
806         }
807     }

```

```

807         if (settings.comm_settings.enable_put &&
808             settings.comm_settings.enable_lock_all) {
809             MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
810         }
811         if (settings.comm_settings.enable_one_by_one &&
812             settings.comm_settings.enable_lock_all) {
813             MPI_Win_lock_all(0, this->comm_struct.window_x);
814         }
815         MPI_Win_lock_all(0, this->window_residual_vector);
816         MPI_Win_lock_all(0, this->window_convergence);
817     }
818 }

```

#### 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

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_solution</i>	The local solution vector in the subdomain.
<i>local_rhs</i>	The local right hand side vector in the subdomain.
<i>global_solution</i>	The workspace solution vector.
<i>global_old_solution</i>	The global solution vector of the previous iteration.
<i>interface_matrix</i>	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](#).

```

1277 {
1278     using vec_vtype = gko::matrix::Dense<ValueType>;
1279     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
1280         {1.0}, settings.executor);
1281     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
1282         {-1.0}, settings.executor);
1283     auto local_size_x = metadata.local_size_x;
1284     local_solution->copy_from(local_rhs.get());
1285     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
1286         auto temp_solution = vec_vtype::create(
1287             settings.executor, local_solution->get_size(),
1288             gko::Array<ValueType>::view(settings.executor,

```

```

1289                                     local_solution->get_size()[0],
1290                                     global_solution->get_values()),
1291                                     1);
1292     interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
1293                             local_solution.get());
1294 }
1295 }

```

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

- `restricted_schwarz.hpp` (c9f1d38)
- `/home/runner/work/schwarz-lib/schwarz-lib/source/restricted_schwarz.cpp` (2f81ed4)

## 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

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the METIS routine that failed
<i>error_code</i>	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

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	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

- BadDimension, [21](#)
  - BadDimension, [21](#)
- Communicate, [11](#)
- CudaError, [30](#)
  - CudaError, [31](#)
- CusparsError, [31](#)
  - CusparsError, [32](#)
- enable\_logging
  - schwz::Settings, [53](#)
- exchange\_boundary
  - schwz::Communicate, [28](#)
  - schwz::SolverRAS, [56](#)
- explicit\_laplacian
  - schwz::Settings, [53](#)
- generate\_dipole\_rhs
  - schwz::Initialize, [34](#)
- generate\_rhs
  - schwz::Initialize, [34](#)
- generate\_sin\_rhs
  - schwz::Initialize, [35](#)
- global\_get
  - schwz::Communicate::comm\_struct, [25](#)
- global\_put
  - schwz::Communicate::comm\_struct, [25](#)
- Initialization, [12](#)
- is\_local\_neighbor
  - schwz::Communicate::comm\_struct, [25](#)
- local\_get
  - schwz::Communicate::comm\_struct, [26](#)
- local\_put
  - schwz::Communicate::comm\_struct, [26](#)
- local\_solver\_tolerance
  - schwz::Metadata, [42](#)
- local\_to\_global\_vector
  - schwz::Communicate, [28](#)
- MetisError, [43](#)
  - MetisError, [43](#)
- naturally\_ordered\_factor
  - schwz::Settings, [53](#)
- norm\_type
  - schwz::Settings, [53](#)
- partition
  - schwz::Initialize, [35](#)
- print\_matrix
  - schwz::SchwarzBase, [47](#)
- print\_vector
  - schwz::SchwarzBase, [47](#)
- ProcessTopology, [17](#)
- run
  - schwz::SchwarzBase, [47](#)
- Schwarz Class, [13](#)
- SchwarzBase
  - schwz::SchwarzBase, [46](#)
- schwz, [17](#)
- schwz::CommHelpers, [18](#)
- schwz::Communicate
  - exchange\_boundary, [28](#)
  - local\_to\_global\_vector, [28](#)
  - setup\_windows, [29](#)
  - update\_boundary, [29](#)
- schwz::Communicate< ValueType, IndexType, Mixed↔  
ValueType >, [27](#)
- schwz::Communicate< ValueType, IndexType, Mixed↔  
ValueType >::comm\_struct, [23](#)
- schwz::Communicate::comm\_struct
  - global\_get, [25](#)
  - global\_put, [25](#)
  - is\_local\_neighbor, [25](#)
  - local\_get, [26](#)
  - local\_put, [26](#)
- schwz::EventHelpers, [19](#)
- schwz::Initialize
  - generate\_dipole\_rhs, [34](#)
  - generate\_rhs, [34](#)
  - generate\_sin\_rhs, [35](#)
  - partition, [35](#)
  - setup\_global\_matrix, [37](#)
  - setup\_local\_matrices, [39](#)
  - setup\_vectors, [39](#)
- schwz::Initialize< ValueType, IndexType >, [33](#)
- schwz::Metadata
  - local\_solver\_tolerance, [42](#)
  - tolerance, [42](#)
- schwz::Metadata< ValueType, IndexType >, [40](#)
- schwz::Metadata< ValueType, IndexType >::post\_↔  
process\_data, [44](#)
- schwz::PartitionTools, [19](#)
- schwz::SchwarzBase
  - print\_matrix, [47](#)
  - print\_vector, [47](#)
  - run, [47](#)

- SchwarzBase, [46](#)
- schwz::SchwarzBase< ValueType, IndexType, Mixed↔  
ValueType >, [44](#)
- schwz::Settings, [51](#)
  - enable\_logging, [53](#)
  - explicit\_laplacian, [53](#)
  - naturally\_ordered\_factor, [53](#)
  - norm\_type, [53](#)
  - thres\_type, [53](#)
- schwz::Settings::comm\_settings, [22](#)
- schwz::Settings::convergence\_settings, [30](#)
- schwz::Solve< ValueType, IndexType, MixedValueType  
>, [54](#)
- schwz::SolverRAS< ValueType, IndexType, Mixed↔  
ValueType >, [55](#)
- schwz::SolverRAS
  - exchange\_boundary, [56](#)
  - setup\_local\_matrices, [57](#)
  - setup\_windows, [60](#)
  - SolverRAS, [56](#)
  - update\_boundary, [63](#)
- schwz::SolverTools, [19](#)
- schwz::Utils< ValueType, IndexType >, [65](#)
- schwz::conv\_tools, [18](#)
- schwz::device\_guard, [32](#)
- setup\_global\_matrix
  - schwz::Initialize, [37](#)
- setup\_local\_matrices
  - schwz::Initialize, [39](#)
  - schwz::SolverRAS, [57](#)
- setup\_vectors
  - schwz::Initialize, [39](#)
- setup\_windows
  - schwz::Communicate, [29](#)
  - schwz::SolverRAS, [60](#)
- Solve, [14](#)
- SolverRAS
  - schwz::SolverRAS, [56](#)
- thres\_type
  - schwz::Settings, [53](#)
- tolerance
  - schwz::Metadata, [42](#)
- UmfpackError, [64](#)
  - UmfpackError, [64](#)
- update\_boundary
  - schwz::Communicate, [29](#)
  - schwz::SolverRAS, [63](#)
- Utils, [15](#)