

schwz

Generated automatically from event-based-new

Generated by Doxygen 1.8.13

Contents

1	Main Page	1
2	# Installation Instructions	3
3	Testing Instructions	5
4	Benchmarking.	7
5	Module Documentation	11
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
6	Namespace Documentation	17
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

7	Class Documentation	21
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
7.3.2.6	remote_get	26
7.3.2.7	remote_put	27
7.4	schwz::Communicate< ValueType, IndexType, MixedValueType > Class Template Reference	27
7.4.1	Detailed Description	28
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()	30
7.5	schwz::Settings::convergence_settings Struct Reference	30
7.5.1	Detailed Description	30
7.6	CudaError Class Reference	31
7.6.1	Detailed Description	31
7.6.2	Constructor & Destructor Documentation	31
7.6.2.1	CudaError()	31

7.7	CuspsetError Class Reference	32
7.7.1	Detailed Description	32
7.7.2	Constructor & Destructor Documentation	32
7.7.2.1	CuspsetError()	32
7.8	schwz::device_guard Class Reference	33
7.8.1	Detailed Description	33
7.9	schwz::Initialize< ValueType, IndexType > Class Template Reference	33
7.9.1	Detailed Description	34
7.9.2	Member Function Documentation	34
7.9.2.1	generate_dipole_rhs()	34
7.9.2.2	generate_rhs()	35
7.9.2.3	generate_sin_rhs()	35
7.9.2.4	partition()	36
7.9.2.5	setup_global_matrix()	37
7.9.2.6	setup_local_matrices()	38
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	43
7.10.2.1	local_solver_tolerance	43
7.10.2.2	tolerance	43
7.11	MetisError Class Reference	43
7.11.1	Detailed Description	43
7.11.2	Constructor & Destructor Documentation	44
7.11.2.1	MetisError()	44
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	45
7.13.1	Detailed Description	46
7.13.2	Constructor & Destructor Documentation	46

7.13.2.1	SchwarzBase()	46
7.13.3	Member Function Documentation	47
7.13.3.1	print_matrix()	47
7.13.3.2	print_vector()	48
7.13.3.3	run()	48
7.14	schwz::Settings Struct Reference	51
7.14.1	Detailed Description	53
7.14.2	Member Data Documentation	53
7.14.2.1	enable_logging	53
7.14.2.2	explicit_laplacian	53
7.14.2.3	naturally_ordered_factor	54
7.14.2.4	norm_type	54
7.14.2.5	thres_type	54
7.15	schwz::Solve< ValueType, IndexType, MixedValueType > Class Template Reference	54
7.15.1	Detailed Description	54
7.16	schwz::SolverRAS< ValueType, IndexType, MixedValueType > Class Template Reference	55
7.16.1	Detailed Description	55
7.16.2	Constructor & Destructor Documentation	56
7.16.2.1	SolverRAS()	56
7.16.3	Member Function Documentation	56
7.16.3.1	exchange_boundary()	56
7.16.3.2	setup_local_matrices()	57
7.16.3.3	setup_windows()	61
7.16.3.4	update_boundary()	63
7.17	UmfpackError Class Reference	64
7.17.1	Detailed Description	64
7.17.2	Constructor & Destructor Documentation	64
7.17.2.1	UmfpackError()	64
7.18	schwz::Utils< ValueType, IndexType > Struct Template Reference	65
7.18.1	Detailed Description	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.
- `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 `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 (640e6d0)

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 * > > [remote_put](#)
The array containing the number of elements that each subdomain sends from the other.
- std::shared_ptr< gko::Array< IndexType * > > [global_get](#)
The array containing the number of elements that each subdomain gets from the other.
- std::shared_ptr< gko::Array< IndexType * > > [local_get](#)

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

 - `std::shared_ptr< gko::Array< IndexType * > >` [remote_get](#)
- The array containing the number of elements that each subdomain gets from the other.*

 - `std::shared_ptr< gko::Array< IndexType > >` [window_ids](#)
- The RDMA window ids.*

 - `std::shared_ptr< gko::Array< IndexType > >` [windows_from](#)
- The RDMA window ids to receive data from.*

 - `std::shared_ptr< gko::Array< IndexType > >` [windows_to](#)
- The RDMA window ids to send data to.*

 - `std::shared_ptr< gko::Array< MPI_Request > >` [put_request](#)
- The put request array.*

 - `std::shared_ptr< gko::Array< MPI_Request > >` [get_request](#)
- The get request array.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [send_buffer](#)
- The send buffer used for the actual communication for both one-sided and two-sided (always allocated).*

 - `std::shared_ptr< gko::matrix::Dense< MixedValueType > >` [mixedt_send_buffer](#)
- The mixed send buffer used for the actual communication for both one-sided and two-sided.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [recv_buffer](#)
- The recv buffer used for the actual communication for both one-sided and two-sided (always allocated).*

 - `std::shared_ptr< gko::matrix::Dense< MixedValueType > >` [mixedt_recv_buffer](#)
- The mixed precision recv buffer used for the actual communication for both one-sided and two-sided.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [extra_buffer](#)
- The extrapolation buffer used for extrapolation of values at the receiver.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [last_recv_bdy](#)
- The last received boundary values for each of the in neighbors for extrapolation.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [curr_send_avg](#)
- Average of values in the send buffer for each of the out neighbors.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [last_send_avg](#)
- Average of values in the last send buffer for each of the out neighbors.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [curr_recv_avg](#)
- Average of values in the recv buffer for each of the out neighbors.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [last_recv_avg](#)
- Average of values in the last recv buffer for each of the out neighbors.*

 - `std::shared_ptr< gko::Array< IndexType > >` [msg_count](#)
- Number of messages sent.*

 - `std::shared_ptr< gko::Array< IndexType > >` [last_recv_iter](#)
- Iteration stamp of last received values.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [last_recv_slopes](#)
- Last recv slopes.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [last_sent_slopes_avg](#)
- Last sent slopes.*

 - `std::shared_ptr< gko::Array< IndexType > >` [last_sent_iter](#)
- Iteration stamp of last received values.*

 - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [thres](#)
- Threshold.*

 - `std::shared_ptr< gko::Array< IndexType > >` [get_displacements](#)
- The displacements for the receiving of the buffer.*

 - `std::shared_ptr< gko::Array< IndexType > >` [put_displacements](#)
- The displacements for the sending of the buffer.*

 - `MPI_Win` [window_recv_buffer](#)
- The RDMA window for the recv buffer.*

- MPI_Win [window_send_buffer](#)
The RDMA window for the send buffer.
- MPI_Win [window_x](#)
The RDMA window for the solution vector.

7.3.1 Detailed Description

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

7.3.2.6 remote_get

```
template<typename ValueType , typename IndexType , typename MixedValueType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType, Mixed←
ValueType >::comm_struct::remote_get
```

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

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

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

7.3.2.7 remote_put

```
template<typename ValueType , typename IndexType , typename MixedValueType >
std::shared_ptr<gko::Array<IndexType *> > schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::remote_put
```

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

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

Referenced by `schwz::SolverRAS< ValueType, IndexType, 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` (640e6d0)

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

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

```
#include <communicate.hpp>
```

Classes

- struct `comm_struct`
The communication struct used to store the communication data.

Public Member Functions

- virtual void `setup_comm_buffers` ()=0
Sets up the communication buffers needed for the boundary exchange.
- virtual void `setup_windows` (const `Settings` &settings, const `Metadata`< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &main_buffer)=0
Sets up the windows needed for the asynchronous communication.
- virtual void `exchange_boundary` (const `Settings` &settings, const `Metadata`< ValueType, IndexType > &metadata, const std::shared_ptr< gko::matrix::Dense< ValueType >> &prev_global_solution, std::shared_ptr< gko::matrix::Dense< ValueType >> &global_solution, 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,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & prev_global_solution,
    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 {
72     using vec = gko::matrix::Dense<ValueType>;
73     auto alpha = gko::initialize<gko::matrix::Dense<ValueType>>(
74         {1.0}, settings.executor);
75     auto temp_vector = vec::create(
76         settings.executor, gko::dim<2>(metadata.local_size, 1),
77         gko::Array<ValueType>::view(
78             settings.executor, metadata.local_size,
79             &global_vector->get_values()[metadata.first_row
80                                     ->get_data()[metadata.my_rank]]),
81         1);
82
83     auto temp_vector2 = vec::create(
84         settings.executor, gko::dim<2>(metadata.local_size, 1),
85         gko::Array<ValueType>::view(settings.executor, metadata.local_size,
86                                     local_vector->get_values()),
87         1);
88     if (settings.convergence_settings.convergence_crit ==
89         Settings::convergence_settings::local_convergence_crit::
90             residual_based) {
91         local_vector->add_scaled(alpha.get(), temp_vector.get());
92         temp_vector->add_scaled(alpha.get(), local_vector.get());
93     } else {
94         temp_vector->copy_from(temp_vector2.get());
95     }
96 }

```

7.4.2.3 setup_windows()

```

template<typename ValueType , typename IndexType , 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](#) (640e6d0)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp](#) (640e6d0)

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](#) (640e6d0)

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 memory 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.

Parameters

<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 mameory 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 (640e6d0)
- /home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp (640e6d0)

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.
- std::shared_ptr< gko::Array< IndexType > > [overlap_row](#)
The overlap row indices.
- std::shared_ptr< gko::Array< IndexType > > [first_row](#)
The starting row of each subdomain in the matrix.
- std::shared_ptr< gko::Array< IndexType > > [permutation](#)
The permutation used for the re-ordering.
- std::shared_ptr< gko::Array< IndexType > > [i_permutation](#)
The inverse permutation used for the re-ordering.

7.10.1 Detailed Description

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

The solver metadata struct.

Template Parameters

<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 (640e6d0)

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 (640e6d0)

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     using vec_vecshared = gko::Array<IndexType *>;
82     metadata.my_local_rank =
83         Utils<ValueType, IndexType>::get_local_rank(metadata.mpi_communicator);
84     metadata.local_num_procs = Utils<ValueType, IndexType>::get_local_num_procs(
85         metadata.mpi_communicator);
86     auto my_local_rank = metadata.my_local_rank;
```

```

87     if (settings.executor_string == "omp") {
88         settings.executor = gko::OmpExecutor::create();
89         auto exec_info =
90             static_cast<gko::OmpExecutor *>(settings.executor.get())
91             ->get_exec_info();
92         exec_info->bind_to_core(metadata.my_local_rank);
93     }
94     else if (settings.executor_string == "cuda") {
95         int num_devices = 0;
96 #if SCHW_HAVE_CUDA
97         SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
98 #else
99         SCHWARZ_NOT_IMPLEMENTED;
100 #endif
101         Utils<ValueType, IndexType>::assert_correct_cuda_devices(
102             num_devices, metadata.my_rank);
103         settings.executor = gko::CudaExecutor::create(
104             my_local_rank, gko::OmpExecutor::create());
105         auto exec_info = static_cast<gko::OmpExecutor *>(
106             settings.executor->get_master().get())
107             ->get_exec_info();
108         exec_info->bind_to_core(my_local_rank);
109         settings.cuda_device_guard =
110             std::make_shared<schwz::device_guard>(my_local_rank);
111
112         std::cout << " Rank " << metadata.my_rank << " with local rank "
113             << my_local_rank << " has "
114             << (static_cast<gko::CudaExecutor *>(settings.executor.get()))
115             ->get_device_id()
116             << " id of gpu" << std::endl;
117         MPI_Barrier(metadata.mpi_communicator);
118     }
119     else if (settings.executor_string == "reference") {
120         settings.executor = gko::ReferenceExecutor::create();
121         auto exec_info =
122             static_cast<gko::ReferenceExecutor *>(settings.executor.get())
123             ->get_exec_info();
124         exec_info->bind_to_core(my_local_rank);
125     }

```

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::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`.

```
328 {
329     using vec_vtype = gko::matrix::Dense<ValueType>;
330     if (!solution.get()) {
331         solution =
332             vec_vtype::create(settings.executor->get_master(),
```

```

333         gko::dim<2>(this->metadata.global_size, 1));
334     }
335     MixedValueType dummy1 = 0.0;
336     ValueType dummy2 = 1.0;
337
338     auto num_neighbors_out = this->comm_struct.num_neighbors_out;
339     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
340
341     if (metadata.my_rank == 0) {
342         std::cout << " MixedValueType: " << typeid(dummy1).name()
343             << " ValueType: " << typeid(dummy2).name() << std::endl;
344     }
345     // The main solution vector
346     std::shared_ptr<vec_vtype> global_solution = vec_vtype::create(
347         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
348
349     // The previous iteration solution vector
350     std::shared_ptr<vec_vtype> prev_global_solution = vec_vtype::create(
351         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
352
353     // The solution vector at the previous event of communication
354     std::shared_ptr<vec_vtype> prev_event_solution = vec_vtype::create(
355         settings.executor, gko::dim<2>(metadata.global_size, 1));
356
357     // A work vector.
358     std::shared_ptr<vec_vtype> work_vector = vec_vtype::create(
359         settings.executor, gko::dim<2>(2 * this->metadata.local_size_x, 1));
360
361     // An initial guess.
362     std::shared_ptr<vec_vtype> init_guess = vec_vtype::create(
363         settings.executor, gko::dim<2>(this->metadata.local_size_x, 1));
364     // init_guess->copy_from(local_rhs.get());
365
366     if (settings.executor_string == "omp") {
367         ValueType sum_rhs = std::accumulate(
368             local_rhs->get_values(),
369             local_rhs->get_values() + local_rhs->get_size()[0], 0.0);
370         std::cout << " Rank " << this->metadata.my_rank << " sum local rhs "
371             << sum_rhs << std::endl;
372     }
373
374     // Initialize all vectors - tbd
375
376     // std::vector<IndexType> local_converged_iter_count;
377
378     // Setup the windows for the onesided communication.
379     this->setup_windows(this->settings, this->metadata, global_solution);
380
381     const auto solver_settings =
382         (Settings::local_solver_settings::direct_solver_cholmod |
383          Settings::local_solver_settings::direct_solver_umfpack |
384          Settings::local_solver_settings::direct_solver_ginkgo |
385          Settings::local_solver_settings::iterative_solver_dealii |
386          Settings::local_solver_settings::iterative_solver_ginkgo) &
387         settings.local_solver;
388     prev_global_solution->copy_from(gko::lend(global_solution));
389
390     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
391         global_residual_norm = 0.0, global_residual_norm0 = -1.0;
392     metadata.iter_count = 0;
393     int num_converged_procs = 0;
394
395     std::string rank_string = std::to_string(metadata.my_rank);
396     if (metadata.my_rank < 10) {
397         rank_string = "0" + std::to_string(metadata.my_rank);
398     }
399
400     std::ofstream fps; // file for sending log
401     std::ofstream fpr; // file for receiving log
402     if (settings.debug_print) {
403         // Opening files for event logs
404         fps.open("send" + rank_string + ".txt");
405         fpr.open("recv" + rank_string + ".txt");
406     }
407
408     if (metadata.my_rank == 0) {
409         std::cout << "Send history - " << metadata.sent_history
410             << ", Recv history - " << metadata.recv_history << std::endl;
411         std::cout << "Thres type - " << settings.thres_type << std::endl;
412         std::cout << "Overlap - " << settings.overlap << std::endl;
413     }
414
415     auto start_time = std::chrono::steady_clock::now();
416
417     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
418         // Exchange the boundary values. The communication part.
419         MEASURE_ELAPSED_FUNC_TIME(

```

```

420         this->exchange_boundary(settings, metadata, prev_global_solution,
421                                global_solution, prev_event_solution, fps,
422                                fpr),
423         0, metadata.my_rank, boundary_exchange, metadata.iter_count);
424     prev_global_solution->copy_from(gko::lend(global_solution));
425
426     // Update the boundary and interior values after the exchanging from
427     // other processes.
428     MEASURE_ELAPSED_FUNC_TIME(
429     this->update_boundary(settings, metadata, this->
local_solution,
430                          this->local_rhs, global_solution,
431                          this->interface_matrix),
432     1, metadata.my_rank, boundary_update, metadata.iter_count);
433
434     // Check for the convergence of the solver.
435     // num_converged_procs = 0;
436     MEASURE_ELAPSED_FUNC_TIME(
437     (Solve<ValueType, IndexType, MixedValueType>::check_convergence(
438     settings, metadata, this->comm_struct, this->convergence_vector,
439     global_solution, this->local_solution, this->
local_matrix,
440     work_vector, local_residual_norm, local_residual_norm0,
441     global_residual_norm, global_residual_norm0,
442     num_converged_procs)),
443     2, metadata.my_rank, convergence_check, metadata.iter_count);
444
445     // break if the solution diverges.
446     if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
447         std::cout << " Rank " << metadata.my_rank << " diverged in "
448         << metadata.iter_count << " iters " << std::endl;
449         std::exit(-1);
450     }
451
452     // break if all processes detect that all other processes have
453     // converged otherwise continue iterations.
454     if (num_converged_procs == metadata.num_subdomains) {
455         break;
456     } else {
457         MEASURE_ELAPSED_FUNC_TIME(
458         (Solve<ValueType, IndexType, MixedValueType>::local_solve(
459         settings, metadata, this->local_matrix,
460         this->triangular_factor_l, this->
triangular_factor_u,
461         this->local_perm, this->local_inv_perm, work_vector,
462         init_guess, this->local_solution)),
463         3, metadata.my_rank, local_solve, metadata.iter_count);
464
465         // Gather the local vector into the locally global vector for
466         // communication.
467         MEASURE_ELAPSED_FUNC_TIME(
468         (Communicate<ValueType, IndexType, MixedValueType>::
469         local_to_global_vector(settings, metadata,
470         this->local_solution,
471         global_solution)),
472         4, metadata.my_rank, expand_local_vec, metadata.iter_count);
473     }
474 }
475 MPI_Barrier(MPI_COMM_WORLD);
476 auto elapsed_time = std::chrono::duration<ValueType>(
477     std::chrono::steady_clock::now() - start_time);
478
479 if (settings.debug_print) {
480     // Closing event log files
481     fps.close();
482     fpr.close();
483 }
484
485 // adding 1 to include the 0-th iteration
486 // metadata.iter_count = metadata.iter_count + 1;
487
488 // number of messages a PE would send without event-based
489 int noevent_msg_count = metadata.iter_count * num_neighbors_out;
490
491 int total_events = 0;
492
493 // Printing msg count
494 if (settings.debug_print) {
495     for (int k = 0; k < num_neighbors_out; k++) {
496         std::cout << " Rank: " << metadata.my_rank << " to "
497         << neighbors_out[k] << " : "
498         << this->comm_struct.msg_count->get_data()[k];
499         total_events += this->comm_struct.msg_count->get_data()[k];
500     }
501     std::cout << std::endl;
502 }
503

```



```

504 // Total no of messages in all PEs
505 MPI_Allreduce(MPI_IN_PLACE, &total_events, 1, MPI_INT, MPI_SUM,
506               MPI_COMM_WORLD);
507 MPI_Allreduce(MPI_IN_PLACE, &noevent_msg_count, 1, MPI_INT, MPI_SUM,
508               MPI_COMM_WORLD);
509
510 if (metadata.my_rank == 0) {
511     std::cout << "Total number of events - " << total_events << std::endl;
512     std::cout << "Total number of msgs without event - "
513               << noevent_msg_count << std::endl;
514 }
515
516 // Write the residuals and iterations to files
517 if (settings.write_iters_and_residuals &&
518     solver_settings ==
519     Settings::local_solver_settings::iterative_solver_ginkgo) {
520     std::string filename = "iter_res_" + rank_string + ".csv";
521     write_iters_and_residuals(
522         metadata.num_subdomains, metadata.my_rank,
523         metadata.post_process_data.local_residual_vector_out.size(),
524         metadata.post_process_data.local_residual_vector_out,
525         metadata.post_process_data.local_converged_iter_count,
526         metadata.post_process_data.local_converged_resnorm,
527         metadata.post_process_data.local_timestamp, filename);
528 }
529 if (num_converged_procs < metadata.num_subdomains) {
530     std::cout << "Rank " << metadata.my_rank << " did not converge in "
531               << metadata.iter_count << " iterations." << std::endl;
532 } else {
533     std::cout << " Rank " << metadata.my_rank << " converged in "
534               << metadata.iter_count << " iterations " << std::endl;
535     ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
536               residual_norm = -1.0;
537
538     // Compute the final residual norm. Also gathers the solution from all
539     // subdomains.
540     Solve<ValueType, IndexType, MixedValueType>::compute_residual_norm(
541         settings, metadata, global_matrix, global_rhs, global_solution,
542         mat_norm, rhs_norm, sol_norm, residual_norm);
543     gather_comm_data<ValueType, IndexType, MixedValueType>(
544         metadata.num_subdomains, this->comm_struct,
545         metadata.comm_data_struct);
546     // clang-format off
547     if (metadata.my_rank == 0)
548     {
549         std::cout
550             << " residual norm " << residual_norm << "\n"
551             << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
552             << " Time taken for solve " << elapsed_time.count()
553             << std::endl;
554     }
555     // clang-format on
556 }
557 if (metadata.my_rank == 0) {
558     solution->copy_from(global_solution.get());
559 }
560
561 // Communicate<ValueType, IndexType>::clear(settings);
562 }

```

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

- schwarz_base.hpp (640e6d0)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz_base.cpp (640e6d0)

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 (640e6d0)

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 (640e6d0)
- /home/runner/work/schwarz-lib/schwarz-lib/source/solve.cpp (92dbd95)

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

An implementation of the solver interface using the RAS solver.

```
#include <restricted_schwarz.hpp>
```

Public Member Functions

- [SolverRAS](#) ([Settings](#) &settings, [Metadata](#)< ValueType, IndexType > &metadata)
The constructor that takes in the user settings and a metadata struct containing the solver metadata.
- void [setup_local_matrices](#) ([Settings](#) &settings, [Metadata](#)< ValueType, IndexType > &metadata, std::vector< unsigned int > &[partition_indices](#), std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &[global_matrix](#), std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &[local_matrix](#), std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &[interface_matrix](#)) override
Sets up the local and the interface matrices from the global matrix and the partition indices.
- void [setup_comm_buffers](#) () override
Sets up the communication buffers needed for the boundary exchange.
- void [setup_windows](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &[main_buffer](#)) override
Sets up the windows needed for the asynchronous communication.
- void [exchange_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, const std::shared_ptr< gko::matrix::Dense< ValueType >> &[prev_global_solution](#), std::shared_ptr< gko::matrix::Dense< ValueType >> &[global_solution](#), 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,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & prev_global_solution,
    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](#).

```

1261 {
1262     if (settings.comm_settings.enable_onesided) {
1263         exchange_boundary_onesided<ValueType, IndexType, MixedValueType>(
1264             settings, metadata, this->comm_struct, prev_global_solution,
1265             global_solution, prev_event_solution, fps, fpr);
1266     } else {
1267         exchange_boundary_twosided<ValueType, IndexType, MixedValueType>(
1268             settings, metadata, this->comm_struct, prev_global_solution,
1269             global_solution);
1270     }
1271 }
```

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_itype = 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_idxes = global_matrix->get_col_idxes();
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
106     if (partition_settings == Settings::partition_settings::partitionmetis ||
107         partition_settings ==
108         Settings::partition_settings::partition_regular2d) {
109         if (num_subdomains > 1) {
110             for (auto p = 0; p < num_subdomains; p++) {
111                 local_p_size[p] = 0;
112             }
113             for (auto i = 0; i < global_size; i++) {
114                 local_p_size[partition_indices[i]]++;
115             }
116             first_row[0] = 0;
117             for (auto p = 0; p < num_subdomains; ++p) {
118                 first_row[p + 1] = first_row[p] + local_p_size[p];
119             }
120             // permutation
121             for (auto i = 0; i < global_size; i++) {
122                 permutation[first_row[partition_indices[i]]] = i;
123                 first_row[partition_indices[i]]++;
124             }
125             for (auto p = num_subdomains; p > 0; p--) {
126                 first_row[p] = first_row[p - 1];
127             }
128             first_row[0] = 0;
129
130             // iperm
131             for (auto i = 0; i < global_size; i++) {
132                 i_permutation[permutation[i]] = i;
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 metadata.overlap_row = std::shared_ptr<vec_itype>(
190     new vec_itype(gko::Array<IndexType>::view(
191         settings.executor, metadata.overlap_size,
192         &(metadata.local_to_global->get_data()[metadata.local_size]))),
193     std::default_delete<vec_itype>());
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 = metadata.overlap_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_idxes = local_matrix_compute->get_col_idxes();
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_idxes = interface_matrix_compute->get_col_idxes();
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_idxes[j]] != 0) {
250                 lmat_col_idxes[nnz_local] =
251                     global_to_local[gmat_col_idxes[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 = metadata.overlap_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_idxes[j]] != 0) {
271                     lmat_col_idxes[nnz_local] =
272                         global_to_local[gmat_col_idxes[j]] - 1;
273                     lmat_values[nnz_local] = gmat_values[j];
274                     nnz_local++;
275                 } else {
276                     imat_col_idxes[nnz_interface] = gmat_col_idxes[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_idxes[j]] == 0) {
291                 local_to_global[num] = gmat_col_idxes[j];
292                 global_to_local[gmat_col_idxes[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::recv_buffer, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::remote_get, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::remote_put, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::send_buffer, schwz::Settings::thres_type, schwz::Settings::use_mixed_precision, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::window_recv_buffer, schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::window_send_buffer, and schwz::Communicate< ValueType, IndexType, MixedValueType >::comm_struct::window_x.

```

667 {
668     using vec_itype = gko::Array<IndexType>;
669     using vec_vtype = gko::matrix::Dense<ValueType>;
670     auto num_subdomains = metadata.num_subdomains;
671     auto local_size_o = metadata.local_size_o;
672     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
673     auto global_get = this->comm_struct.global_get->get_data();
674     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
675     auto global_put = this->comm_struct.global_put->get_data();
676
677     // set displacement for the MPI buffer
678     auto get_displacements = this->comm_struct.get_displacements->get_data();
679     auto put_displacements = this->comm_struct.put_displacements->get_data();
680     {
681         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
682         tmp_num_comm_elems[0] = 0;
683         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
684             if ((global_get[j])[0] > 0) {
685                 int p = neighbors_in[j];
686                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
687             }
688         }
689         for (auto j = 0; j < num_subdomains; j++) {
690             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
691         }
692
693         auto mpi_itype = schwz::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
694         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
695                     1, mpi_itype, MPI_COMM_WORLD);
696     }
697
698     {
699         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
700         tmp_num_comm_elems[0] = 0;
701         for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
702             if ((global_put[j])[0] > 0) {
703                 int p = neighbors_out[j];
704                 tmp_num_comm_elems[p + 1] = (global_put[j])[0];
705             }
706         }
707         for (auto j = 0; j < num_subdomains; j++) {
708             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
709         }
710
711         auto mpi_itype = schwz::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
712         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
713                     1, mpi_itype, MPI_COMM_WORLD);
714     }
715
716     // setup windows
717     if (settings.comm_settings.enable_onesided) {
718         // Onesided
719
720         for (int i = 0; i < main_buffer->get_size()[0]; i++) {
721             main_buffer->get_values()[i] = 0.0;
722         }
723
724         MPI_Win_create(main_buffer->get_values(),
725                       main_buffer->get_size()[0] * sizeof(ValueType),
726                       sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
727                       &(this->comm_struct.window_x));
728     }
729
730
731     if (settings.comm_settings.enable_onesided) {
732         // MPI_Alloc_mem ? Custom allocator ? TODO
733
734         for (int i = 0; i < num_subdomains; i++) {
735             this->local_residual_vector->get_values()[i] = 0.0;
736         }
737
738         MPI_Win_create(this->local_residual_vector->get_values(),
739                       (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
740                       MPI_INFO_NULL, MPI_COMM_WORLD,
741                       &(this->window_residual_vector));

```

```

742     std::vector<IndexType> zero_vec(num_subdomains, 0);
743     gko::Array<IndexType> temp_array(settings.executor->get_master(),
744                                     zero_vec.begin(), zero_vec.end());
745     this->convergence_vector = std::shared_ptr<vec_itype>(
746         new vec_itype(settings.executor->get_master(), temp_array),
747         std::default_delete<vec_itype>());
748     this->convergence_sent = std::shared_ptr<vec_itype>(
749         new vec_itype(settings.executor->get_master(), num_subdomains),
750         std::default_delete<vec_itype>());
751     this->convergence_local = std::shared_ptr<vec_itype>(
752         new vec_itype(settings.executor->get_master(), num_subdomains),
753         std::default_delete<vec_itype>());
754
755     for (int i = 0; i < num_subdomains; i++) {
756         this->convergence_vector->get_data()[i] = 0;
757         this->convergence_sent->get_data()[i] = 0;
758         this->convergence_local->get_data()[i] = 0;
759     }
760
761     MPI_Win_create(this->convergence_vector->get_data(),
762                   (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
763                   MPI_INFO_NULL, MPI_COMM_WORLD,
764                   &(this->window_convergence));
765 }
766
767 if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
768     // Lock all windows.
769     if (settings.comm_settings.enable_get &&
770         settings.comm_settings.enable_lock_all) {
771         MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
772     }
773     if (settings.comm_settings.enable_put &&
774         settings.comm_settings.enable_lock_all) {
775         MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
776     }
777     if (settings.comm_settings.enable_one_by_one &&
778         settings.comm_settings.enable_lock_all) {
779         MPI_Win_lock_all(0, this->comm_struct.window_x);
780     }
781     MPI_Win_lock_all(0, this->window_residual_vector);
782     MPI_Win_lock_all(0, this->window_convergence);
783 }
784 }

```

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](#).

```

1282 {
1283     using vec_vtype = gko::matrix::Dense<ValueType>;
1284     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
1285         {1.0}, settings.executor);
1286     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
1287         {-1.0}, settings.executor);
1288     auto local_size_x = metadata.local_size_x;
1289     local_solution->copy_from(local_rhs.get());
1290     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
1291         auto temp_solution = vec_vtype::create(
1292             settings.executor, local_solution->get_size(),
1293             gko::Array<ValueType>::view(settings.executor,
1294                                     local_solution->get_size()[0],
1295                                     global_solution->get_values()),
1296             1);
1297         interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
1298                               local_solution.get());
1299     }
1300 }
```

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

- [restricted_schwarz.hpp \(640e6d0\)](#)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/restricted_schwarz.cpp \(640e6d0\)](#)

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, [31](#)
 - CudaError, [31](#)
- CusparsError, [32](#)
 - 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, [35](#)
- 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, [43](#)
- local_to_global_vector
 - schwz::Communicate, [28](#)
- MetisError, [43](#)
 - MetisError, [44](#)
- naturally_ordered_factor
 - schwz::Settings, [53](#)
- norm_type
 - schwz::Settings, [54](#)
- partition
 - schwz::Initialize, [36](#)
- print_matrix
 - schwz::SchwarzBase, [47](#)
- print_vector
 - schwz::SchwarzBase, [47](#)
- ProcessTopology, [17](#)
- remote_get
 - schwz::Communicate::comm_struct, [26](#)
- remote_put
 - schwz::Communicate::comm_struct, [26](#)
- run
 - schwz::SchwarzBase, [48](#)
- 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, [30](#)
- 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](#)
 - remote_get, [26](#)
 - remote_put, [26](#)
- schwz::EventHelpers, [19](#)
- schwz::Initialize
 - generate_dipole_rhs, [34](#)
 - generate_rhs, [35](#)
 - generate_sin_rhs, [35](#)
 - partition, [36](#)
 - setup_global_matrix, [37](#)
 - setup_local_matrices, [38](#)
 - setup_vectors, [39](#)
- schwz::Initialize< ValueType, IndexType >, [33](#)
- schwz::Metadata
 - local_solver_tolerance, [43](#)
 - tolerance, [43](#)
- 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, 48
 - SchwarzBase, 46
- schwz::SchwarzBase< ValueType, IndexType, Mixed↵
 - ValueType >, 45
- schwz::Settings, 51
 - enable_logging, 53
 - explicit_laplacian, 53
 - naturally_ordered_factor, 53
 - norm_type, 54
 - thres_type, 54
- 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, 61
 - SolverRAS, 56
 - update_boundary, 63
- schwz::SolverTools, 19
- schwz::Utils< ValueType, IndexType >, 65
- schwz::conv_tools, 18
- schwz::device_guard, 33
- setup_global_matrix
 - schwz::Initialize, 37
- setup_local_matrices
 - schwz::Initialize, 38
 - schwz::SolverRAS, 57
- setup_vectors
 - schwz::Initialize, 39
- setup_windows
 - schwz::Communicate, 29
 - schwz::SolverRAS, 61
- Solve, 14
- SolverRAS
 - schwz::SolverRAS, 56
- thres_type
 - schwz::Settings, 54
- tolerance
 - schwz::Metadata, 43
- UmfpackError, 64
 - UmfpackError, 64
- update_boundary
 - schwz::Communicate, 30
 - schwz::SolverRAS, 63
- Utils, 15