schwz

Generated automatically from dealii-ex

Contents

Chapter 1

Main Page

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

Modules

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

- Initialization : Handles the initialization of the problem and the solver.

- Communicate : Handles the communication.

- Solve : Handles the local solution and the convergence detection.

- Schwarz Class : The Classes related to the Schwarz solvers.

- Utils : Provides some basic utilities.

Chapter 2

Installation Instructions

Building

Use the standard cmake build procedure:

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

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

- `-DSCHWARZ_BUILD_BENCHMARKING={ON, OFF}` Builds some example benchmarks. Default is `ON`

- `-DSCHWARZ_BUILD_METIS={ON, OFF}` Builds with support for the `METIS` partitioner. User needs to provide the path to the installation of the `METIS` library in `METIS_DIR`, preferably as an environment variable. Default is `OFF`

- `-DSCHWARZ_BUILD_CHOLMOD={ON, OFF}` Builds with support for the `CHOLMOD` module from the Suitesparse library. User needs to set an environment variable `CHOLMOD_DIR` to the path containing the `CHOLMOD` installation. Default is `OFF`

- `-DSCHWARZ_BUILD_CUDA={ON, OFF}` Builds with CUDA support. Though Ginkgo provides most of the required CUDA support, we do need to link to CUDA for explicit setting of GPU affinities, some custom gather and scatter operations. Default is `OFF`.

- `-DSCHWARZ_BUILD_CLANG_TIDY={ON, OFF}` Builds with support for `clang-tidy` Default is `OFF`

- `-DSCHWARZ_BUILD_DEALII={ON, OFF}` Builds with support for the finite element library `deal.ii` Default is `OFF`

- `-DSCHWARZ_WITH_HWLOC={ON, OFF}` Builds with support for the hardware locality library used for binding hardware. `hwloc` is distributed as a part of the Open-MPI project. Default is `ON`

- `-DSCHWARZ_DEVEL_TOOLS={ON, OFF}` Builds with some developer tools support. Default is `ON`. In particular uses `git-cmake-format` to automatically format the source files with `clang-format`.

Tips

- If you are having CUDA problems and you are not using CUDA, then feel free to switch the CUDA module off with `-DSCHWARZ_BUILD_CUDA=off`.

- Installing CHOLMOD can be a bit annoying. TODO add some details on fixing Suitesparse compilation.

- When doing merge commits it is possible that make format does not work. You can run `cmake -DSCH↩ WARZ_DEVEL_TOOLS=OFF ..` to temporarily switch off the formatting. Please switch it on again when committing normally.

Chapter 3

Testing Instructions

Chapter 4

Benchmarking.

Benchmark example 1.

Poisson solver using Restricted Additive Schwarz with overlap.

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

The executable is run in the following fashion:

"'sh [MPI_COMMAND] [MPI_OPTIONS]

Chapter 5

Module Documentation

5.1 Communicate

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

Namespaces

- schwz::CommHelpers

    The CommHelper namespace .

- ProcessTopology

    The ProcessTopology namespace .

Classes

- class schwz::Communicate< ValueType, IndexType >

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

- struct schwz::Metadata< ValueType, IndexType >

    The solver metadata struct.

5.1.1 Detailed Description

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

5.2 Initialization

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

Namespaces

- schwz::PartitionTools

  The PartitionTools namespace .
- ProcessTopology

  The ProcessTopology namespace .

Classes

- class schwz::device_guard

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

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

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

  The solver metadata struct.

5.2.1 Detailed Description

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

5.3 Schwarz Class

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

Classes

- class schwz::SolverRAS< ValueType, IndexType >

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

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

5.3.1 Detailed Description

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

5.4   Solve

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

Namespaces

- schwz::ConvergenceTools

    The ConvergenceTools namespace .
- schwz::SolverTools

    The SolverTools namespace .

Classes

- struct schwz::Metadata< ValueType, IndexType >

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

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

5.4.1   Detailed Description

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

5.5 Utils

A module dedicated to the utilities in schwarz-lib.

Classes

- struct schwz::Utils< ValueType, IndexType >

  The utilities class which provides some checks and basic utilities.

5.5.1 Detailed Description

A module dedicated to the utilities in schwarz-lib.

Chapter 6

Namespace Documentation

6.1 ProcessTopology Namespace Reference

The ProcessTopology namespace .

6.1.1 Detailed Description

The ProcessTopology namespace .

proc_topo

6.2 schwz Namespace Reference

The Schwarz wrappers namespace.

Namespaces

- CommHelpers

    The CommHelper namespace .
- ConvergenceTools

    The ConvergenceTools 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::ConvergenceTools Namespace Reference

The ConvergenceTools namespace .

6.4.1 Detailed Description

The ConvergenceTools namespace .

conv_tools

6.5 schwz::PartitionTools Namespace Reference

The PartitionTools namespace .

6.5.1 Detailed Description

The PartitionTools namespace .

part_tools

6.6 schwz::SolverTools Namespace Reference

The SolverTools namespace .

6.6.1 Detailed Description

The SolverTools namespace .

solver_tools

Chapter 7

Class Documentation

7.1 BadDimension Class Reference

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

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

Public Member Functions

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

  Initializes a bad dimension error.

7.1.1 Detailed Description

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

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BadDimension()

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

Initializes a bad dimension error.

Parameters

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

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

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

- exception.hpp (87f773a)

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 (87f773a)

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

The communication struct used to store the communication data.

`#include <communicate.hpp>`

Public Attributes

- int num_neighbors_in

  The number of neighbors this subdomain has to receive data from.
- int num_neighbors_out

  The number of neighbors this subdomain has to send data to.
- std::shared_ptr< gko::Array< IndexType > > neighbors_in

  The neighbors this subdomain has to receive data from.
- std::shared_ptr< gko::Array< IndexType > > neighbors_out

  The neighbors this subdomain has to send data to.
- std::vector< bool > is_local_neighbor

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

  The number of neighbors this subdomain has to receive data from.
- int local_num_neighbors_out

  The number of neighbors this subdomain has to send data to.
- std::shared_ptr< gko::Array< IndexType > > local_neighbors_in

  The neighbors this subdomain has to receive data from.
- std::shared_ptr< gko::Array< IndexType > > local_neighbors_out

  The neighbors this subdomain has to send data to.
- std::shared_ptr< gko::Array< IndexType ∗ > > global_put

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

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

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

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

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

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

  The RDMA window ids.

- std::shared_ptr< gko::Array< IndexType > > windows_from

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

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

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

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

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

    The recv buffer used for the actual communication for both one-sided and two-sided.
- std::shared_ptr< gko::Array< IndexType > > get_displacements

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

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

    The RDMA window for the recv buffer.
- MPI_Win window_send_buffer

    The RDMA window for the send buffer.
- MPI_Win window_x

    The RDMA window for the solution vector.

7.3.1 Detailed Description

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

The communication struct used to store the communication data.

7.3.2 Member Data Documentation

7.3.2.1 global_get

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

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

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

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

7.3.2.2   global_put

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

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

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

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

7.3.2.3   is_local_neighbor

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

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

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

7.3.2.4   local_get

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

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

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

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

7.3.2.5   local_put

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

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

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

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

7.3.2.6 remote_get

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

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

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

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

7.3.2.7 remote_put

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

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

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

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

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

- communicate.hpp (87f773a)

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

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

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

Classes

- struct comm_struct

    The communication struct used to store the communication data.

Public Member Functions

- virtual void setup_comm_buffers ()=0

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

    Sets up the windows needed for the asynchronous communication.
- virtual void exchange_boundary (const Settings &settings, const Metadata< ValueType, IndexType > &meta-data, std::shared_ptr< gko::matrix::Dense< ValueType >> &global_solution)=0

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

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

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

    Clears the data.

7.4.1 Detailed Description

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

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

Template Parameters

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

Communicate

7.4.2 Member Function Documentation

7.4.2.1 exchange_boundary()

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

Exchanges the elements of the solution vector.

Parameters

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

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

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

7.4.2.2 local_to_global_vector()

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

Transforms data from a local vector to a global vector.

Parameters

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

```
69 {
70     using vec = gko::matrix::Dense<ValueType>;
71     auto alpha = gko::initialize<gko::matrix::Dense<ValueType>>(
72         {1.0}, settings.executor);
73     auto temp_vector = vec::create(
74         settings.executor, gko::dim<2>(metadata.local_size, 1),
75         gko::Array<ValueType>::view(
76             settings.executor, metadata.local_size,
77             &global_vector->get_values()[metadata.first_row
78                                          ->get_data()[metadata.my_rank]]),
79         1);
80
81     auto temp_vector2 = vec::create(
82         settings.executor, gko::dim<2>(metadata.local_size, 1),
83         gko::Array<ValueType>::view(settings.executor, metadata.local_size,
84                                     local_vector->get_values()),
85         1);
86     if (settings.convergence_settings.convergence_crit ==
87         Settings::convergence_settings::local_convergence_crit::
88             residual_based) {
89         local_vector->add_scaled(alpha.get(), temp_vector.get());
90         temp_vector->add_scaled(alpha.get(), local_vector.get());
91     } else {
92         temp_vector->copy_from(temp_vector2.get());
93     }
94 }
```

7.4.2.3 setup_windows()

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

Sets up the windows needed for the asynchronous communication.

Parameters

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

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

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

7.4.2.4 update_boundary()

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

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

Parameters

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

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

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

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

- communicate.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp (87f773a)

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 (87f773a)

7.6 CudaError Class Reference

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

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

Public Member Functions

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

7.6.1 Detailed Description

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

7.6.2 Constructor & Destructor Documentation

7.6.2.1 CudaError()

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

Initializes a CUDA error.

Parameters

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

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

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

- exception.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (87f773a)

7.7 CusparseError Class Reference

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

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

Public Member Functions

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

7.7.1 Detailed Description

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 CusparseError()

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

Initializes a cuSPARSE error.

Parameters

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

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

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

- exception.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (87f773a)

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 (87f773a)

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

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

```
#include <initialization.hpp>
```

Public Member Functions

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

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

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

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

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

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

Public Attributes

- std::vector< unsigned int > partition_indices

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

  The cell weights for the partition algorithm.

Additional Inherited Members

7.9.1 Detailed Description

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

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

Template Parameters

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

Initialization

7.9.2 Member Function Documentation

7.9.2.1 generate_rhs()

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

Generates the right hand side vector.

Parameters

rhs	The rhs vector.

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

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

```
81 {
82     std::uniform_real_distribution<double> unif(0.0, 1.0);
83     std::default_random_engine engine;
84     for (gko::size_type i = 0; i < rhs.size(); ++i) {
85         rhs[i] = unif(engine);
86     }
87 }
```

7.9.2.2 partition()

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

The partitioning function.

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

Parameters

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

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

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

```
283 {
284     partition_indices.resize(metadata.global_size);
285     if (metadata.my_rank == 0) {
286         auto partition_settings =
287             (Settings::partition_settings::partition_zoltan |
288              Settings::partition_settings::partition_metis |
289              Settings::partition_settings::partition_regular |
290              Settings::partition_settings::partition_regular2d |
291              Settings::partition_settings::partition_custom) &
292             settings.partition;
293
294         if (partition_settings ==
295             Settings::partition_settings::partition_zoltan) {
296             SCHWARZ_NOT_IMPLEMENTED;
297         } else if (partition_settings ==
298                 Settings::partition_settings::partition_metis) {
299             if (metadata.my_rank == 0) {
300                 std::cout << " METIS partition" << std::endl;
301             }
302             PartitionTools::PartitionMetis(
303                 settings, global_matrix, this->cell_weights,
304                 metadata.num_subdomains, partition_indices);
305         } else if (partition_settings ==
306                 Settings::partition_settings::partition_regular) {
307             if (metadata.my_rank == 0) {
308                 std::cout << " Regular 1D partition" << std::endl;
309             }
310             PartitionTools::PartitionRegular(
311                 global_matrix, metadata.num_subdomains, partition_indices);
312         } else if (partition_settings ==
313                 Settings::partition_settings::partition_regular2d) {
314             if (metadata.my_rank == 0) {
315                 std::cout << " Regular 2D partition" << std::endl;
316             }
317             PartitionTools::PartitionRegular2D(
318                 global_matrix, settings.write_debug_out,
319                 metadata.num_subdomains, partition_indices);
320         } else if (partition_settings ==
321                 Settings::partition_settings::partition_custom) {
322             // User partitions mesh manually
323             SCHWARZ_NOT_IMPLEMENTED;
324         } else {
325             SCHWARZ_NOT_IMPLEMENTED;
326         }
327     }
328 }
```

7.9.2.3 setup_global_matrix()

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

Generates the 2D global laplacian matrix.

Parameters

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

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

```
191 {
192     using index_type = IndexType;
```

```
193     using value_type = ValueType;
194     using mtx = gko::matrix::Csr<value_type, index_type>;
195     if (settings.matrix_filename != "null") {
196         auto input_file = std::ifstream(filename, std::ios::in);
197         if (!input_file) {
198             std::cerr << "Could not find the file \"" << filename
199                         << "\", which is required for this test.\n";
200         }
201         global_matrix =
202             gko::read<mtx>(input_file, settings.executor->get_master());
203         global_matrix->sort_by_column_index();
204         std::cout << "Matrix from file " << filename << std::endl;
205     } else if (settings.matrix_filename == "null" &&
206             settings.explicit_laplacian) {
207         std::cout << "Laplacian 2D Matrix (generated in house) " << std::endl;
208         gko::size_type global_size = oned_laplacian_size *
    oned_laplacian_size;
209
210         global_matrix = mtx::create(settings.executor->get_master(),
211                             gko::dim<2>(global_size), 5 * global_size);
212         value_type *values = global_matrix->get_values();
213         index_type *row_ptrs = global_matrix->get_row_ptrs();
214         index_type *col_idxs = global_matrix->get_col_idxs();
215
216         std::vector<gko::size_type> exclusion_set;
217
218         std::map<IndexType, ValueType> stencil_map = {
219             {-oned_laplacian_size, -1}, {-1, -1}, {0, 4}, {1, -1},
220             {oned_laplacian_size, -1},
221         };
222         for (auto i = 2; i < global_size; ++i) {
223             gko::size_type index = (i - 1) * oned_laplacian_size;
224             if (index * index < global_size * global_size) {
225                 exclusion_set.push_back(
226                     linearize_index(index, index - 1, global_size));
227                 exclusion_set.push_back(
228                     linearize_index(index - 1, index, global_size));
229             }
230         }
231
232         std::sort(exclusion_set.begin(),
233                 exclusion_set.begin() + exclusion_set.size());
234
235         IndexType pos = 0;
236         IndexType col_idx = 0;
237         row_ptrs[0] = pos;
238         gko::size_type cur_idx = 0;
239         for (IndexType i = 0; i < global_size; ++i) {
240             for (auto ofs : stencil_map) {
241                 auto in_exclusion_flag =
242                     (exclusion_set[cur_idx] ==
243                     linearize_index(i, i + ofs.first, global_size));
244                 if (0 <= i + ofs.first && i + ofs.first < global_size &&
245                     !in_exclusion_flag) {
246                     values[pos] = ofs.second;
247                     col_idxs[pos] = i + ofs.first;
248                     ++pos;
249                 }
250                 if (in_exclusion_flag) {
251                     cur_idx++;
252                 }
253                 col_idx = row_ptrs[i + 1] - pos;
254             }
255             row_ptrs[i + 1] = pos;
256         }
257     } else {
258         std::cerr << " Need to provide a matrix or enable the default "
259                     "laplacian matrix."
260                 << std::endl;
261         std::exit(-1);
262     }
263 }
```

7.9.2.4 setup_local_matrices()

```
template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::setup_local_matrices (
```

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

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

Parameters

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

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

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

7.9.2.5 setup_vectors()

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

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

Parameters

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

References schwz::Settings::executor, schwz::Metadata< ValueType, IndexType >::first_row, schwz::Metadata<

ValueType, IndexType >::global_size, schwz::Metadata< ValueType, IndexType >::local_size_x, and schwz::↩
Metadata< ValueType, IndexType >::my_rank.

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

```
339 {
340     using vec = gko::matrix::Dense<ValueType>;
341     auto my_rank = metadata.my_rank;
342     auto first_row = metadata.first_row->get_data()[my_rank];
343
344     // Copy the global rhs vector to the required executor.
345     gko::Array<ValueType> temp_rhs{settings.executor->get_master(), rhs.begin(),
346                                    rhs.end()};
347     global_rhs = vec::create(settings.executor,
348                              gko::dim<2>{metadata.global_size, 1}, temp_rhs, 1);
349     global_solution = vec::create(settings.executor->get_master(),
350                                   gko::dim<2>(metadata.global_size, 1));
351
352     local_rhs =
353         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
354     // Extract the local rhs from the global rhs. Also takes into account the
355     // overlap.
356     SolverTools::extract_local_vector(settings, metadata, local_rhs.get(),
357                                       global_rhs.get(), first_row);
358
359     local_solution =
360         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
361 }
```

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

- initialization.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp (87f773a)

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

The solver metadata struct.

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

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 solver.
- unsigned int precond_max_block_size

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

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

  The minimum residual norm of the subdomain.
- std::vector< std::tuple< int, int, int, std::string, std::vector< ValueType > > > time_struct

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

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

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

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

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

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

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

  The inverse permutation used for the re-ordering.

7.10.1 Detailed Description

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

The solver metadata struct.

Template Parameters

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

7.10.2 Member Data Documentation

7.10.2.1 local_solver_tolerance

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

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

The residual norm reduction required.

7.10.2.2 tolerance

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

The tolerance of the complete solver.

The residual norm reduction required.

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

- settings.hpp (87f773a)

7.11 MetisError Class Reference

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

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

Public Member Functions

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

  Initializes a METIS error.

7.11.1 Detailed Description

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

7.11.2 Constructor & Destructor Documentation

7.11.2.1 MetisError()

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

Initializes a METIS error.

Parameters

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

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

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

- exception.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (87f773a)

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

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

```
#include <schwarz_base.hpp>
```

Public Member Functions

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

    The constructor that takes in the user settings and a metadata struct containing the solver metadata.
- void initialize ()

    Initialize the matrix and vectors.
- void run (std::shared_ptr< gko::matrix::Dense< ValueType >> &solution)

    The function that runs the actual solver and obtains the final solution.
- void print_vector (const std::shared_ptr< gko::matrix::Dense< ValueType >> &vector, int subd, std::string name)

    The auxiliary function that prints a passed in vector.
- void print_matrix (const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &matrix, int rank, std::string name)

    The auxiliary function that prints a passed in CSR matrix.

Public Attributes

- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > local_matrix

  The local subdomain matrix.
- std::shared_ptr< gko::matrix::Permutation< IndexType > > local_perm

  The local subdomain permutation matrix/array.
- std::shared_ptr< gko::matrix::Permutation< IndexType > > local_inv_perm

  The local subdomain inverse permutation matrix/array.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > triangular_factor_l

  The local lower triangular factor used for the triangular solves.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > triangular_factor_u

  The local upper triangular factor used for the triangular solves.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > interface_matrix

  The local interface matrix.
- std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > > global_matrix

  The global matrix.
- std::shared_ptr< gko::matrix::Dense< ValueType > > local_rhs

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

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

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

  The global solution vector.

Additional Inherited Members

7.12.1 Detailed Description

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

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

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

Template Parameters

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

7.12.2 Constructor & Destructor Documentation

7.12.2.1 SchwarzBase()

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

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

Parameters

settings	The settings struct.
metadata	The metadata struct.

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

```
50       : Initialize<ValueType, IndexType>(settings, metadata),
51         settings(settings),
52         metadata(metadata)
53 {
54     using vec_itype = gko::Array<IndexType>;
55     using vec_vecshared = gko::Array<IndexType *>;
56     metadata.my_local_rank =
57         Utils<ValueType, IndexType>::get_local_rank(metadata.mpi_communicator);
58     metadata.local_num_procs = Utils<ValueType, IndexType>::get_local_num_procs(
59         metadata.mpi_communicator);
60     auto my_local_rank = metadata.my_local_rank;
61     if (settings.executor_string == "omp") {
62         settings.executor = gko::OmpExecutor::create();
63         auto exec_info =
64             static_cast<gko::OmpExecutor *>(settings.executor.get())
65                 ->get_exec_info();
66         exec_info->bind_to_core(metadata.my_local_rank);
67
68     } else if (settings.executor_string == "cuda") {
69         int num_devices = 0;
70 #if SCHW_HAVE_CUDA
71         SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
72 #else
73         SCHWARZ_NOT_IMPLEMENTED;
74 #endif
75         Utils<ValueType, IndexType>::assert_correct_cuda_devices(
76             num_devices, metadata.my_rank);
77         settings.executor = gko::CudaExecutor::create(
78             my_local_rank, gko::OmpExecutor::create());
79         auto exec_info = static_cast<gko::OmpExecutor *>(
80                             settings.executor->get_master().get())
81                             ->get_exec_info();
82         exec_info->bind_to_core(my_local_rank);
83         settings.cuda_device_guard =
84             std::make_shared<schwz::device_guard>(my_local_rank);
85
86         std::cout << " Rank " << metadata.my_rank << " with local rank "
87                 << my_local_rank << " has "
88                 << (static_cast<gko::CudaExecutor *>(settings.executor.get()))
89                         ->get_device_id()
90                 << " id of gpu" << std::endl;
91         MPI_Barrier(metadata.mpi_communicator);
92     } else if (settings.executor_string == "reference") {
93         settings.executor = gko::ReferenceExecutor::create();
94         auto exec_info =
95             static_cast<gko::ReferenceExecutor *>(settings.executor.get())
96                 ->get_exec_info();
97         exec_info->bind_to_core(my_local_rank);
98     }
99 }
```

7.12.3 Member Function Documentation

7.12.3.1 print_matrix()

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void schwz::SchwarzBase< ValueType, IndexType >::print_matrix (
            const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & matrix,
            int rank,
            std::string name )
```

The auxiliary function that prints a passed in CSR matrix.

Parameters

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

7.12.3.2 print_vector()

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

The auxiliary function that prints a passed in vector.

Parameters

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

7.12.3.3 run()

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

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

Parameters

solution	The solution vector.

References schwz::Communicate< ValueType, IndexType >::exchange_boundary(), schwz::Settings::executor,

schwz::SchwarzBase< ValueType, IndexType >::global_matrix, schwz::SchwarzBase< ValueType, IndexType >↩
::global_rhs, schwz::SchwarzBase< ValueType, IndexType >::global_solution, schwz::SchwarzBase< Value↩
Type, IndexType >::interface_matrix, schwz::SchwarzBase< ValueType, IndexType >::local_inv_perm, schwz↩
::SchwarzBase< ValueType, IndexType >::local_matrix, schwz::SchwarzBase< ValueType, IndexType >::local↩
_perm, schwz::SchwarzBase< ValueType, IndexType >::local_rhs, schwz::SchwarzBase< ValueType, Index↩
Type >::local_solution, schwz::Communicate< ValueType, IndexType >::setup_windows(), schwz::SchwarzBase<
ValueType, IndexType >::triangular_factor_l, schwz::SchwarzBase< ValueType, IndexType >::triangular_factor_u,
and schwz::Communicate< ValueType, IndexType >::update_boundary().

```
289 {
290     using vec_vtype = gko::matrix::Dense<ValueType>;
291
292     solution = vec_vtype::create(settings.executor->get_master(),
293                                  gko::dim<2>(this->metadata.global_size, 1));
294     // The main solution vector
295     std::shared_ptr<vec_vtype> global_solution = vec_vtype::create(
296         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
297     // A work vector.
298     std::shared_ptr<vec_vtype> work_vector = vec_vtype::create(
299         settings.executor, gko::dim<2>(2 * this->metadata.local_size_x, 1));
300     // Setup the windows for the onesided communication.
301     this->setup_windows(this->settings, this->metadata, global_solution);
302
303     const auto solver_settings =
304         (Settings::local_solver_settings::direct_solver_cholmod |
305          Settings::local_solver_settings::direct_solver_umfpack |
306          Settings::local_solver_settings::direct_solver_ginkgo |
307          Settings::local_solver_settings::iterative_solver_dealii |
308          Settings::local_solver_settings::iterative_solver_ginkgo) &
309         settings.local_solver;
310
311     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
312               global_residual_norm = 0.0, global_residual_norm0 = -1.0;
313     metadata.iter_count = 0;
314     auto start_time = std::chrono::steady_clock::now();
315     int num_converged_procs = 0;
316
317     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
318         // Exchange the boundary values. The communication part.
319         MEASURE_ELAPSED_FUNC_TIME(
320             this->exchange_boundary(settings, metadata, global_solution), 0,
321             metadata.my_rank, boundary_exchange, metadata.iter_count);
322
323         // Update the boundary and interior values after the exchanging from
324         // other processes.
325         MEASURE_ELAPSED_FUNC_TIME(
326             this->update_boundary(settings, metadata, this->
    local_solution,
327                                   this->local_rhs, global_solution,
328                                   this->interface_matrix),
329             1, metadata.my_rank, boundary_update, metadata.iter_count);
330
331         // Check for the convergence of the solver.
332         // num_converged_procs = 0;
333         MEASURE_ELAPSED_FUNC_TIME(
334             (Solve<ValueType, IndexType>::check_convergence(
335                 settings, metadata, this->comm_struct, this->convergence_vector,
336                 global_solution, this->local_solution, this->
    local_matrix,
337                 work_vector, local_residual_norm, local_residual_norm0,
338                 global_residual_norm, global_residual_norm0,
339                 num_converged_procs)),
340             2, metadata.my_rank, convergence_check, metadata.iter_count);
341
342         // break if the solution diverges.
343         if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
344             std::cout << " Rank " << metadata.my_rank << " diverged in "
345                       << metadata.iter_count << " iters " << std::endl;
346             std::exit(-1);
347         }
348
349         // break if all processes detect that all other processes have
350         // converged otherwise continue iterations.
351         if (num_converged_procs == metadata.num_subdomains) {
352             break;
353         } else {
354             MEASURE_ELAPSED_FUNC_TIME(
355                 (Solve<ValueType, IndexType>::local_solve(
356                     settings, metadata, this->local_matrix,
357                     this->triangular_factor_l, this->
    triangular_factor_u,
358                     this->local_perm, this->local_inv_perm, work_vector,
```

```
359                          this->local_solution)),
360                      3, metadata.my_rank, local_solve, metadata.iter_count);
361              // init_guess->copy_from(this->local_solution.get());
362              // Gather the local vector into the locally global vector for
363              // communication.
364              MEASURE_ELAPSED_FUNC_TIME(
365                  (Communicate<ValueType, IndexType>::local_to_global_vector
     (
366                      settings, metadata, this->local_solution, global_solution)),
367                  4, metadata.my_rank, expand_local_vec, metadata.iter_count);
368          }
369      }
370      MPI_Barrier(MPI_COMM_WORLD);
371      auto elapsed_time = std::chrono::duration<ValueType>(
372          std::chrono::steady_clock::now() - start_time);
373      std::cout << " Rank " << metadata.my_rank << " converged in "
374                << metadata.iter_count << " iters " << std::endl;
375      ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
376                residual_norm = -1.0;
377      // Compute the final residual norm. Also gathers the solution from all
378      // subdomains.
379      Solve<ValueType, IndexType>::compute_residual_norm(
380          settings, metadata, global_matrix, global_rhs, global_solution,
381          mat_norm, rhs_norm, sol_norm, residual_norm);
382      gather_comm_data<ValueType, IndexType>(
383          metadata.num_subdomains, this->comm_struct, metadata.comm_data_struct);
384      // clang-format off
385      if (metadata.my_rank == 0)
386        {
387          std::cout
388              << " residual norm " << residual_norm << "\n"
389              << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
390              << " Time taken for solve " << elapsed_time.count()
391              << std::endl;
392          if (num_converged_procs < metadata.num_subdomains)
393            {
394              std::cout << " Did not converge in " << metadata.iter_count
395                        << " iterations."
396                        << std::endl;
397            }
398        }
399      // clang-format on
400      if (metadata.my_rank == 0) {
401          solution->copy_from(global_solution.get());
402      }
403
404      // Communicate<ValueType, IndexType>::clear(settings);
405 }
```

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

- schwarz_base.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz_base.cpp (87f773a)

7.13 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 enable_random_rhs = false

    Flag to enable a random rhs.
- bool print_matrices = false

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

    Flag to enable some debug printing.
- bool 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_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.

7.13.1 Detailed Description

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

settings

7.13.2 Member Data Documentation

7.13.2.1 explicit_laplacian

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

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

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

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

7.13.2.2 naturally_ordered_factor

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

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

Note

This is mainly to allow compatibility with GPU solution.

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

- settings.hpp (87f773a)

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

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

```
#include <solve.hpp>
```

Additional Inherited Members

7.14.1 Detailed Description

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

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

Template Parameters

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

[Solve](#)

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

- solve.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/solve.cpp (87f773a)

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

An implementation of the solver interface using the RAS solver.

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

Public Member Functions

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

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

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

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

    Sets up the windows needed for the asynchronous communication.
- void [exchange_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &[global_solution](#)) override

    Exchanges the elements of the solution vector.
- void [update_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared_ptr< gko::matrix::Dense< ValueType >> &[local_solution](#), const std::shared_ptr< gko::matrix ↩ ::Dense< ValueType >> &[local_rhs](#), const std::shared_ptr< gko::matrix::Dense< ValueType >> &[global↩ _solution](#), const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> &[interface_matrix](#)) override

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

Additional Inherited Members

7.15.1   Detailed Description

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

An implementation of the solver interface using the RAS solver.

Template Parameters

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

7.15.2 Constructor & Destructor Documentation

7.15.2.1 SolverRAS()

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

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

Parameters

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

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

7.15.3 Member Function Documentation

7.15.3.1 exchange_boundary()

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

Exchanges the elements of the solution vector.

Parameters

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

Implements schwz::Communicate$<$ ValueType, IndexType $>$.

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

```
798 {
799     if (settings.comm_settings.enable_onesided) {
800         exchange_boundary_onesided<ValueType, IndexType>(
801             settings, metadata, this->comm_struct, global_solution);
802     } else {
803         exchange_boundary_twosided<ValueType, IndexType>(
804             settings, metadata, this->comm_struct, global_solution);
805     }
806 }
```

7.15.3.2 setup_local_matrices()

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

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

Parameters

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

Implements schwz::Initialize$<$ ValueType, IndexType $>$.

References schwz::Metadata$<$ ValueType, IndexType $>$::comm_size, schwz::Settings::executor, schwz::$\leftarrow$
Metadata$<$ ValueType, IndexType $>$::first_row, schwz::SchwarzBase$<$ ValueType, IndexType $>$::global_matrix,
schwz::Metadata$<$ ValueType, IndexType $>$::global_size, schwz::Metadata$<$ ValueType, IndexType $>$::global_to$\leftarrow$
_local, schwz::Metadata$<$ ValueType, IndexType $>$::i_permutation, schwz::SchwarzBase$<$ ValueType, IndexType
$>$::interface_matrix, schwz::SchwarzBase$<$ ValueType, IndexType $>$::local_matrix, schwz::Metadata$<$ Value$\leftarrow$
Type, 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::$\leftarrow$
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.

```
61  {
62      using mtx = gko::matrix::Csr<ValueType, IndexType>;
63      using vec_itype = gko::Array<IndexType>;
64      using perm_type = gko::matrix::Permutation<IndexType>;
65      using arr = gko::Array<IndexType>;
66      auto my_rank = metadata.my_rank;
67      auto comm_size = metadata.comm_size;
68      auto num_subdomains = metadata.num_subdomains;
69      auto global_size = metadata.global_size;
70      auto mpi_itype = boost::mpi::get_mpi_datatype(*partition_indices.data());
71
72      MPI_Bcast(partition_indices.data(), global_size, mpi_itype, 0,
73               MPI_COMM_WORLD);
74
75      std::vector<IndexType> local_p_size(num_subdomains);
76      auto global_to_local = metadata.global_to_local->get_data();
77      auto local_to_global = metadata.local_to_global->get_data();
78
79      auto first_row = metadata.first_row->get_data();
80      auto permutation = metadata.permutation->get_data();
81      auto i_permutation = metadata.i_permutation->get_data();
82
83      auto nb = (global_size + num_subdomains - 1) /
    num_subdomains;
84      auto partition_settings =
85          (Settings::partition_settings::partition_zoltan |
86           Settings::partition_settings::partition_metis |
87           Settings::partition_settings::partition_regular |
88           Settings::partition_settings::partition_regular2d |
89           Settings::partition_settings::partition_custom) &
90          settings.partition;
91
92      IndexType *gmat_row_ptrs = global_matrix->get_row_ptrs();
93      IndexType *gmat_col_idxs = global_matrix->get_col_idxs();
94      ValueType *gmat_values = global_matrix->get_values();
95
96      // default local p size set for 1 subdomain.
97      first_row[0] = 0;
98      for (auto p = 0; p < num_subdomains; ++p) {
99          local_p_size[p] = std::min(global_size - first_row[p], nb);
100         first_row[p + 1] = first_row[p] + local_p_size[p];
101     }
102
103
104      if (partition_settings == Settings::partition_settings::partition_metis ||
105          partition_settings ==
106              Settings::partition_settings::partition_regular2d) {
107          if (num_subdomains > 1) {
108              for (auto p = 0; p < num_subdomains; p++) {
109                  local_p_size[p] = 0;
110              }
111              for (auto i = 0; i < global_size; i++) {
112                  local_p_size[partition_indices[i]]++;
113              }
114              first_row[0] = 0;
115              for (auto p = 0; p < num_subdomains; ++p) {
116                  first_row[p + 1] = first_row[p] + local_p_size[p];
117              }
118              // permutation
119              for (auto i = 0; i < global_size; i++) {
120                  permutation[first_row[partition_indices[i]]] = i;
121                  first_row[partition_indices[i]]++;
122              }
123              for (auto p = num_subdomains; p > 0; p--) {
124                  first_row[p] = first_row[p - 1];
125              }
126              first_row[0] = 0;
127
128              // iperm
129              for (auto i = 0; i < global_size; i++) {
130                  i_permutation[permutation[i]] = i;
131              }
132          }
133
134          auto gmat_temp = mtx::create(settings.executor->get_master(),
135                                       global_matrix->get_size(),
136                                       global_matrix->get_num_stored_elements());
137
138          auto nnz = 0;
139          gmat_temp->get_row_ptrs()[0] = 0;
140          for (auto row = 0; row < metadata.global_size; ++row) {
141              for (auto col = gmat_row_ptrs[permutation[row]];
142                   col < gmat_row_ptrs[permutation[row] + 1]; ++col) {
143                  gmat_temp->get_col_idxs()[nnz] =
144                      i_permutation[gmat_col_idxs[col]];
145                  gmat_temp->get_values()[nnz] = gmat_values[col];
146                  nnz++;
```

```
147                 }
148                 gmat_temp->get_row_ptrs()[row + 1] = nnz;
149             }
150         global_matrix->copy_from(gmat_temp.get());
151     }
152
153
154     for (auto i = 0; i < global_size; i++) {
155         global_to_local[i] = 0;
156         local_to_global[i] = 0;
157     }
158     auto num = 0;
159     for (auto i = first_row[my_rank]; i < first_row[
    my_rank + 1]; i++) {
160         global_to_local[i] = 1 + num;
161         local_to_global[num] = i;
162         num++;
163     }
164
165     IndexType old = 0;
166     for (auto k = 1; k < settings.overlap; k++) {
167         auto now = num;
168         for (auto i = old; i < now; i++) {
169             for (auto j = gmat_row_ptrs[local_to_global[i]];
170                  j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
171                 if (global_to_local[gmat_col_idxs[j]] == 0) {
172                     local_to_global[num] = gmat_col_idxs[j];
173                     global_to_local[gmat_col_idxs[j]] = 1 + num;
174                     num++;
175                 }
176             }
177         }
178         old = now;
179     }
180     metadata.local_size = local_p_size[my_rank];
181     metadata.local_size_x = num;
182     metadata.local_size_o = global_size;
183     auto local_size = metadata.local_size;
184     auto local_size_x = metadata.local_size_x;
185
186     metadata.overlap_size = num - metadata.local_size;
187     metadata.overlap_row = std::shared_ptr<vec_itype>(
188         new vec_itype(gko::Array<IndexType>::view(
189             settings.executor, metadata.overlap_size,
190             &(metadata.local_to_global->get_data()[metadata.local_size]))),
191         std::default_delete<vec_itype>());
192
193     auto nnz_local = 0;
194     auto nnz_interface = 0;
195
196     for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
197         for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; j++) {
198             if (global_to_local[gmat_col_idxs[j]] != 0) {
199                 nnz_local++;
200             } else {
201                 std::cout << " debug: invalid edge?" << std::endl;
202             }
203         }
204     }
205     auto temp = 0;
206     for (auto k = 0; k < metadata.overlap_size; k++) {
207         temp = metadata.overlap_row->get_data()[k];
208         for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1]; j++) {
209             if (global_to_local[gmat_col_idxs[j]] != 0) {
210                 nnz_local++;
211             } else {
212                 nnz_interface++;
213             }
214         }
215     }
216
217     std::shared_ptr<mtx> local_matrix_compute;
218     local_matrix_compute = mtx::create(settings.executor->get_master(),
219                                 gko::dim<2>(local_size_x), nnz_local);
220     IndexType *lmat_row_ptrs = local_matrix_compute->get_row_ptrs();
221     IndexType *lmat_col_idxs = local_matrix_compute->get_col_idxs();
222     ValueType *lmat_values = local_matrix_compute->get_values();
223
224     std::shared_ptr<mtx> interface_matrix_compute;
225     if (nnz_interface > 0) {
226         interface_matrix_compute =
227             mtx::create(settings.executor->get_master(),
228                         gko::dim<2>(local_size_x), nnz_interface);
229     } else {
230         interface_matrix_compute = mtx::create(settings.executor->get_master());
231     }
232
```

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

7.15.3.3 setup_windows()

```
template<typename ValueType , typename IndexType >
void schwz::SolverRAS< ValueType, IndexType >::setup_windows (
            const Settings & settings,
```

```
                    const Metadata< ValueType, IndexType > & metadata,
                    std::shared_ptr< gko::matrix::Dense< ValueType >> & main_buffer ) [override],
[virtual]
```

Sets up the windows needed for the asynchronous communication.

Parameters

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

Implements schwz::Communicate< ValueType, IndexType >.

References schwz::Settings::comm_settings::enable_get, schwz::Settings::comm_settings::enable_lock_←
all, schwz::Settings::comm_settings::enable_one_by_one, schwz::Settings::comm_settings::enable_onesided,
schwz::Settings::comm_settings::enable_overlap, schwz::Settings::comm_settings::enable_put, schwz::Settings←
::executor, schwz::Communicate< ValueType, IndexType >::comm_struct::get_displacements, schwz::←
Communicate< ValueType, IndexType >::comm_struct::get_request, schwz::Communicate< ValueType, Index←
Type >::comm_struct::global_get, schwz::Communicate< ValueType, IndexType >::comm_struct::global_put,
schwz::SchwarzBase< ValueType, IndexType >::global_solution, schwz::Communicate< ValueType, Index←
Type >::comm_struct::is_local_neighbor, schwz::Metadata< ValueType, IndexType >::iter_count, schwz::←
Communicate< ValueType, IndexType >::comm_struct::local_get, schwz::Communicate< ValueType, Index←
Type >::comm_struct::local_neighbors_in, schwz::Communicate< ValueType, IndexType >::comm_struct←
::local_neighbors_out, schwz::Communicate< ValueType, IndexType >::comm_struct::local_num_neighbors_in,
schwz::Communicate< ValueType, IndexType >::comm_struct::local_num_neighbors_out, schwz::Communicate<
ValueType, IndexType >::comm_struct::local_put, schwz::Metadata< ValueType, IndexType >::local_size_←
o, schwz::SchwarzBase< ValueType, IndexType >::local_solution, schwz::Communicate< ValueType, IndexType
>::comm_struct::neighbors_in, schwz::Communicate< ValueType, IndexType >::comm_struct::neighbors_←
out, schwz::Communicate< ValueType, IndexType >::comm_struct::num_neighbors_in, schwz::Communicate<
ValueType, IndexType >::comm_struct::num_neighbors_out, schwz::Metadata< ValueType, IndexType >::num←
_subdomains, schwz::Communicate< ValueType, IndexType >::comm_struct::put_displacements, schwz::←
Communicate< ValueType, IndexType >::comm_struct::put_request, schwz::Communicate< ValueType, Index←
Type >::comm_struct::recv_buffer, schwz::Communicate< ValueType, IndexType >::comm_struct::remote_get,
schwz::Communicate< ValueType, IndexType >::comm_struct::remote_put, schwz::Communicate< ValueType,
IndexType >::comm_struct::send_buffer, schwz::Communicate< ValueType, IndexType >::comm_struct::window←
_recv_buffer, schwz::Communicate< ValueType, IndexType >::comm_struct::window_send_buffer, and schwz::←
Communicate< ValueType, IndexType >::comm_struct::window_x.

```
507 {
508     using vec_itype = gko::Array<IndexType>;
509     using vec_vtype = gko::matrix::Dense<ValueType>;
510     auto num_subdomains = metadata.num_subdomains;
511     auto local_size_o = metadata.local_size_o;
512     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
513     auto global_get = this->comm_struct.global_get->get_data();
514     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
515     auto global_put = this->comm_struct.global_put->get_data();
516
517     // set displacement for the MPI buffer
518     auto get_displacements = this->comm_struct.get_displacements->get_data();
519     auto put_displacements = this->comm_struct.put_displacements->get_data();
520     {
521         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
522         tmp_num_comm_elems[0] = 0;
523         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
524             if ((global_get[j])[0] > 0) {
525                 int p = neighbors_in[j];
526                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
527             }
528         }
529         for (auto j = 0; j < num_subdomains; j++) {
530             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
531         }
532
```

```
533          auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
534          MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
535                       1, mpi_itype, MPI_COMM_WORLD);
536      }
537
538      {
539          std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
540          tmp_num_comm_elems[0] = 0;
541          for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
542              if ((global_put[j])[0] > 0) {
543                  int p = neighbors_out[j];
544                  tmp_num_comm_elems[p + 1] = (global_put[j])[0];
545              }
546          }
547          for (auto j = 0; j < num_subdomains; j++) {
548              tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
549          }
550
551          auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
552          MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
553                       1, mpi_itype, MPI_COMM_WORLD);
554      }
555
556      // setup windows
557      if (settings.comm_settings.enable_onesided) {
558          // Onesided
559          MPI_Win_create(main_buffer->get_values(),
560                         main_buffer->get_size()[0] * sizeof(ValueType),
561                         sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
562                         &(this->comm_struct.window_x));
563      }
564
565
566      if (settings.comm_settings.enable_onesided) {
567          // MPI_Alloc_mem ? Custom allocator ?  TODO
568          MPI_Win_create(this->local_residual_vector->get_values(),
569                         (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
570                         MPI_INFO_NULL, MPI_COMM_WORLD,
571                         &(this->window_residual_vector));
572          std::vector<IndexType> zero_vec(num_subdomains, 0);
573          gko::Array<IndexType> temp_array{settings.executor->get_master(),
574                                           zero_vec.begin(), zero_vec.end()};
575          this->convergence_vector = std::shared_ptr<vec_itype>(
576              new vec_itype(settings.executor->get_master(), temp_array),
577              std::default_delete<vec_itype>());
578          this->convergence_sent = std::shared_ptr<vec_itype>(
579              new vec_itype(settings.executor->get_master(), num_subdomains),
580              std::default_delete<vec_itype>());
581          this->convergence_local = std::shared_ptr<vec_itype>(
582              new vec_itype(settings.executor->get_master(), num_subdomains),
583              std::default_delete<vec_itype>());
584          MPI_Win_create(this->convergence_vector->get_data(),
585                         (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
586                         MPI_INFO_NULL, MPI_COMM_WORLD,
587                         &(this->window_convergence));
588      }
589
590      if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
591          // Lock all windows.
592          if (settings.comm_settings.enable_get &&
593              settings.comm_settings.enable_lock_all) {
594              MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
595          }
596          if (settings.comm_settings.enable_put &&
597              settings.comm_settings.enable_lock_all) {
598              MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
599          }
600          if (settings.comm_settings.enable_one_by_one &&
601              settings.comm_settings.enable_lock_all) {
602              MPI_Win_lock_all(0, this->comm_struct.window_x);
603          }
604          MPI_Win_lock_all(0, this->window_residual_vector);
605          MPI_Win_lock_all(0, this->window_convergence);
606      }
607 }
```

7.15.3.4 update_boundary()

```
template<typename ValueType , typename IndexType >
void schwz::SolverRAS< ValueType, IndexType >::update_boundary (
```

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

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

Parameters

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

Implements schwz::Communicate< ValueType, IndexType >.

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

```
817 {
818     using vec_vtype = gko::matrix::Dense<ValueType>;
819     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
820         {1.0}, settings.executor);
821     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
822         {-1.0}, settings.executor);
823     auto local_size_x = metadata.local_size_x;
824     local_solution->copy_from(local_rhs.get());
825     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
826         auto temp_solution = vec_vtype::create(
827             settings.executor, local_solution->get_size(),
828             gko::Array<ValueType>::view(settings.executor,
829                                         local_solution->get_size()[0],
830                                         global_solution->get_values()),
831             1);
832         interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
833                                 local_solution.get());
834     }
835 }
```

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

- restricted_schwarz.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/restricted_schwarz.cpp (87f773a)

7.16 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.

7.16.1   Detailed Description

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

7.16.2   Constructor & Destructor Documentation

7.16.2.1   UmfpackError()

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

Initializes a METIS error.

Parameters

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

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

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

- exception.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (87f773a)

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

The utilities class which provides some checks and basic utilities.

```
#include <utils.hpp>
```

7.17.1   Detailed Description

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

The utilities class which provides some checks and basic utilities.

Template Parameters

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

Utils

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

- utils.hpp (87f773a)
- /home/runner/work/schwarz-lib/schwarz-lib/source/utils.cpp (87f773a)

Index