Ginkgo

Generated from pipelines/508661960 branch based on develop. Ginkgo version 1.5.0

Generated by Doxygen 1.8.16

Chapter 1

Main Page

This is the main page for the Ginkgo 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. The Example programs can help you get started with using Ginkgo.

1.0.0.1 Modules

The Ginkgo library can be grouped into `modules` and these modules form the basic building blocks of Ginkgo. The modules can be summarized as follows:

- Executors : Where do you want your code to be executed ?
- Linear Operators : What kind of operation do you want Ginkgo to perform ?
  - Solvers : Solve a linear system for a given matrix.
  - Preconditioners : Precondition a system for a solve.
  - SpMV employing different Matrix formats : Perform a sparse matrix vector multiplication with a particular matrix format.
- Logging : Monitor your code execution.
- Stopping criteria : Manage your iteration stopping criteria.

Chapter 2

Installation Instructions

2.0.1 Building

Use the standard CMake build procedure:

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

Use `cmake --build .` in some systems like MinGW or Microsoft Visual Studio which do not use `make`.

For Microsoft Visual Studio, use `cmake --build . --config <build_type>` to decide the build type. The possible options are `Debug`, `Release`, `RelWithDebInfo` and `MinSizeRel`.

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

- `-DGINKGO_DEVEL_TOOLS={ON, OFF}` sets up the build system for development (requires clang-format, will also download git-cmake-format), default is `OFF`. The default behavior installs a pre-commit hook, which disables git commits. If it is set to `ON`, a new pre-commit hook for formatting will be installed (enabling commits again). In both cases the hook may overwrite a user defined pre-commit hook when Ginkgo is used as a submodule.
- `-DGINKGO_MIXED_PRECISION={ON, OFF}` compiles true mixed-precision kernels instead of converting data on the fly, default is `OFF`. Enabling this flag increases the library size, but improves performance of mixed-precision kernels.
- `-DGINKGO_BUILD_TESTS={ON, OFF}` builds Ginkgo's tests (will download googletest), default is `ON`.
- `-DGINKGO_FAST_TESTS={ON, OFF}` reduces the input sizes for a few slow tests to speed them up, default is `OFF`.
- `-DGINKGO_BUILD_BENCHMARKS={ON, OFF}` builds Ginkgo's benchmarks (will download gflags and rapidjson), default is `ON`.
- `-DGINKGO_BUILD_EXAMPLES={ON, OFF}` builds Ginkgo's examples, default is `ON`
- `-DGINKGO_BUILD_EXTLIB_EXAMPLE={ON, OFF}` builds the interfacing example with deal.II, default is `OFF`.
- `-DGINKGO_BUILD_REFERENCE={ON, OFF}` build reference implementations of the kernels, useful for testing, default is `ON`
- `-DGINKGO_BUILD_OMP={ON, OFF}` builds optimized OpenMP versions of the kernels, default is `ON` if the selected C++ compiler supports OpenMP, `OFF` otherwise.

- -DGINKGO_BUILD_CUDA={ON, OFF} builds optimized cuda versions of the kernels (requires CUDA), default is ON if a CUDA compiler could be detected, OFF otherwise.
- -DGINKGO_BUILD_DPCPP={ON, OFF} builds optimized DPC++ versions of the kernels (requires CMAKE_CXX_COMPILER to be set to the dpcpp compiler). The default is ON if CMAKE_CXX_COMPILER is a DPC++ compiler, OFF otherwise.
- -DGINKGO_BUILD_HIP={ON, OFF} builds optimized HIP versions of the kernels (requires HIP), default is ON if an installation of HIP could be detected, OFF otherwise.
- -DGINKGO_HIP_AMDGPU="gpuarch1;gpuarch2" the amdgpu_target(s) variable passed to hipcc for the hcc HIP backend. The default is none (auto).
- -DGINKGO_BUILD_HWLOC={ON, OFF} builds Ginkgo with HWLOC. If system HWLOC is not found, Ginkgo will try to build it. Default is ON on Linux. Ginkgo does not support HWLOC on Windows/MacOS, so the default is OFF on Windows/MacOS.
- -DGINKGO_BUILD_DOC={ON, OFF} creates an HTML version of Ginkgo's documentation from inline comments in the code. The default is OFF.
- -DGINKGO_DOC_GENERATE_EXAMPLES={ON, OFF} generates the documentation of examples in Ginkgo. The default is ON.
- -DGINKGO_DOC_GENERATE_PDF={ON, OFF} generates a PDF version of Ginkgo's documentation from inline comments in the code. The default is OFF.
- -DGINKGO_DOC_GENERATE_DEV={ON, OFF} generates the developer version of Ginkgo's documentation. The default is OFF.
- -DGINKGO_EXPORT_BUILD_DIR={ON, OFF} adds the Ginkgo build directory to the CMake package registry. The default is OFF.
- -DGINKGO_WITH_CLANG_TIDY={ON, OFF} makes Ginkgo call clang-tidy to find programming issues. The path can be manually controlled with the CMake variable -DGINKGO_CLANG_TIDY_PATH=<path>. The default is OFF.
- -DGINKGO_WITH_IWYU={ON, OFF} makes Ginkgo call iwyu to find include issues. The path can be manually controlled with the CMake variable -DGINKGO_IWYU_PATH=<path>. The default is OFF.
- -DGINKGO_CHECK_CIRCULAR_DEPS={ON, OFF} enables compile-time checks for circular dependencies between different Ginkgo libraries and self-sufficient headers. Should only be used for development purposes. The default is OFF.
- -DGINKGO_VERBOSE_LEVEL=integer sets the verbosity of Ginkgo.
  - 0 disables all output in the main libraries,
  - 1 enables a few important messages related to unexpected behavior (default).
- GINKGO_INSTALL_RPATH allows setting any RPATH information when installing the Ginkgo libraries. If this is OFF, the behavior is the same as if all other RPATH flags are set to OFF as well. The default is ON.
- GINKGO_INSTALL_RPATH_ORIGIN adds $ORIGIN (Linux) or @loader_path (MacOS) to the installation RPATH. The default is ON.
- GINKGO_INSTALL_RPATH_DEPENDENCIES adds the dependencies to the installation RPATH. The default is OFF.
- -DCMAKE_INSTALL_PREFIX=path sets the installation path for make install. The default value is usually something like /usr/local.
- -DCMAKE_BUILD_TYPE=type specifies which configuration will be used for this build of Ginkgo. The default is RELEASE. Supported values are CMake's standard build types such as DEBUG and RELEASE and the Ginkgo specific COVERAGE, ASAN (AddressSanitizer), LSAN (LeakSanitizer), TSAN (ThreadSanitizer) and UBSAN (undefined behavior sanitizer) types.

- `-DBUILD_SHARED_LIBS={ON, OFF}` builds ginkgo as shared libraries (`OFF`) or as dynamic libraries (`ON`), default is `ON`.
- `-DGINKGO_JACOBI_FULL_OPTIMIZATIONS={ON, OFF}` use all the optimizations for the CUDA Jacobi algorithm. `OFF` by default. Setting this option to `ON` may lead to very slow compile time (>20 minutes) for the `jacobi_generate_kernels.cu` file and high memory usage.
- `-DCMAKE_CUDA_HOST_COMPILER=path` instructs the build system to explicitly set CUDA's host compiler to the path given as argument. By default, CUDA uses its toolchain's host compiler. Setting this option may help if you're experiencing linking errors due to ABI incompatibilities. This option is supported since `CMake 3.8` but `documented starting from 3.10`.
- `-DGINKGO_CUDA_ARCHITECTURES=<list>` where `<list>` is a semicolon (`;`) separated list of architectures. Supported values are:
  - `Auto`
  - `Kepler`, `Maxwell`, `Pascal`, `Volta`, `Turing`, `Ampere`
  - `CODE`, `CODE(COMPUTE)`, `(COMPUTE)`

  `Auto` will automatically detect the present CUDA-enabled GPU architectures in the system. `Kepler`, `Maxwell`, `Pascal`, `Volta` and `Ampere` will add flags for all architectures of that particular NVIDIA GPU generation. `COMPUTE` and `CODE` are placeholders that should be replaced with compute and code numbers (e.g. for `compute_70` and `sm_70` `COMPUTE` and `CODE` should be replaced with `70`. Default is `Auto`. For a more detailed explanation of this option see the `ARCHITECTURES specification list` section in the documentation of the CudaArchitectureSelector CMake module.

For example, to build everything (in debug mode), use:

```
cmake  -G "Unix Makefiles" -H. -BDebug -DCMAKE_BUILD_TYPE=Debug -DGINKGO_DEVEL_TOOLS=ON \
    -DGINKGO_BUILD_TESTS=ON -DGINKGO_BUILD_REFERENCE=ON -DGINKGO_BUILD_OMP=ON \
    -DGINKGO_BUILD_CUDA=ON -DGINKGO_BUILD_HIP=ON
cmake --build Debug
```

NOTE: Ginkgo is known to work with the `Unix Makefiles`, `Ninja`, `MinGW Makefiles` and `Visual Studio 16 2019` based generators. Other CMake generators are untested.

2.0.2 Building Ginkgo in Windows

Depending on the configuration settings, some manual work might be required:

- Build Ginkgo with Debug mode: Some Debug build specific issues can appear depending on the machine and environment: When you encounter the error message `ld: error: export ordinal too large`, add the compilation flag `-O1` by adding `-DCMAKE_CXX_FLAGS=-O1` to the CMake invocation.
- Build Ginkgo in *MinGW*:\ If encountering the issue `cc1plus.exe: out of memory allocating 65536 bytes`, please follow the workaround in `reference`, or trying to compile ginkgo again might work.

2.0.3 Building Ginkgo with HIP support

Ginkgo provides a `HIP` backend. This allows to compile optimized versions of the kernels for either AMD or NV↩IDIA GPUs. The CMake configuration step will try to auto-detect the presence of HIP either at `/opt/rocm/hip` or at the path specified by `HIP_PATH` as a CMake parameter (`-DHIP_PATH=`) or environment variable (`export HIP_PATH=`), unless `-DGINKGO_BUILD_HIP=ON/OFF` is set explicitly.

2.0.3.1 Changing the paths to search for HIP and other packages

All HIP installation paths can be configured through the use of environment variables or CMake variables. This way of configuring the paths is currently imposed by the `HIP` tool suite. The variables are the following:

- CMake `-DROCM_PATH=` or environment `export ROCM_PATH=`: sets the `ROCM` installation path. The default value is `/opt/rocm/`.
- CMake `-DHIP_CLANG_PATH` or environment `export HIP_CLANG_PATH=`: sets the `HIP` compatible `clang` binary path. The default value is `${ROCM_PATH}/llvm/bin`.
- CMake `-DHIP_PATH=` or environment `export HIP_PATH=`: sets the `HIP` installation path. The default value is `${ROCM_PATH}/hip`.
- CMake `-DHIPBLAS_PATH=` or environment `export HIPBLAS_PATH=`: sets the `hipBLAS` installation path. The default value is `${ROCM_PATH}/hipblas`.
- CMake `-DHIPSPARSE_PATH=` or environment `export HIPSPARSE_PATH=`: sets the `hipSPARSE` installation path. The default value is `${ROCM_PATH}/hipsparse`.
- CMake `-DHIPFFT_PATH=` or environment `export HIPFFT_PATH=`: sets the `hipFFT` installation path. The default value is `${ROCM_PATH}/hipfft`.
- CMake `-DROCRAND_PATH=` or environment `export ROCRAND_PATH=`: sets the `rocRAND` installation path. The default value is `${ROCM_PATH}/rocrand`.
- CMake `-DHIPRAND_PATH=` or environment `export HIPRAND_PATH=`: sets the `hipRAND` installation path. The default value is `${ROCM_PATH}/hiprand`.
- environment `export CUDA_PATH=`: where `hipcc` can find `CUDA` if it is not in the default `/usr/local/cuda` path.

2.0.3.2 HIP platform detection of AMD and NVIDIA

By default, Ginkgo uses the output of `/opt/rocm/hip/bin/hipconfig --platform` to select the backend. The accepted values are either `hcc` (`amd` with ROCM >= 4.1) or `nvcc` (`nvidia` with ROCM >= 4.1). When on an AMD or NVIDIA system, this should output the correct platform by default. When on a system without GPUs, this should output `hcc` by default. To change this value, export the environment variable `HIP_PLATFORM` like so:

```
export HIP_PLATFORM=nvcc # or nvidia for ROCM >= 4.1
```

2.0.3.3 Setting platform specific compilation flags

Platform specific compilation flags can be given through the following CMake variables:

- `-DGINKGO_HIP_COMPILER_FLAGS=`: compilation flags given to all platforms.
- `-DGINKGO_HIP_NVCC_COMPILER_FLAGS=`: compilation flags given to NVIDIA platforms.
- `-DGINKGO_HIP_CLANG_COMPILER_FLAGS=`: compilation flags given to AMD clang compiler.

2.0.4 Third party libraries and packages

Ginkgo relies on third party packages in different cases. These third party packages can be turned off by disabling the relevant options.

- GINKGO_BUILD_TESTS=ON: Our tests are implemented with `Google Test`;
- GINKGO_BUILD_BENCHMARKS=ON: For argument management we use `gflags` and for JSON parsing we use `RapidJSON`;
- GINKGO_DEVEL_TOOLS=ON: `git-cmake-format` is our CMake helper for code formatting.
- GINKGO_BUILD_HWLOC=ON: `hwloc` to detect and control cores and devices.

Ginkgo attempts to use pre-installed versions of these package if they match version requirements using `find_package`. Otherwise, the configuration step will download the files for each of the packages `GTest`, `gflags`, `RapidJSON` and `hwloc` and build them internally.

Note that, if the external packages were not installed to the default location, the CMake option `-DCMAKE_PREFIX_PATH=<path-list>` needs to be set to the semicolon (`;`) separated list of install paths of these external packages. For more Information, see the `CMake documentation for CMAKE_PREFIX_PATH` for details.

For convenience, the options `GINKGO_INSTALL_RPATH[_.*]` can be used to bind the installed Ginkgo shared libraries to the path of its dependencies.

2.0.5 Installing Ginkgo

To install Ginkgo into the specified folder, execute the following command in the build folder

```
make install
```

If the installation prefix (see `CMAKE_INSTALL_PREFIX`) is not writable for your user, e.g. when installing Ginkgo system-wide, it might be necessary to prefix the call with `sudo`.

After the installation, CMake can find ginkgo with `find_package(Ginkgo)`. An example can be found in the `test_install`.

Chapter 3

Testing Instructions

3.0.1 Running the unit tests

You need to compile ginkgo with `-DGINKGO_BUILD_TESTS=ON` option to be able to run the tests.

3.0.1.1 Using make test

After configuring Ginkgo, use the following command inside the build folder to run all tests:

```
make test
```

The output should contain several lines of the form:

```
    Start  1: path/to/test
1/13 Test  #1: path/to/test .............................   Passed    0.01 sec
```

To run only a specific test and see more details results (e.g. if a test failed) run the following from the build folder:

```
./path/to/test
```

where `path/to/test` is the path returned by `make test`.

3.0.1.2 Using make quick_test

After compiling Ginkgo, use the following command inside the build folder to run a small subset of tests that should execute quickly:

```
make quick_test
```

These tests do not use GPU features except for a few device property queries, so they may still fail if Ginkgo was compiled with GPU support, but no such GPU is available. The output is equivalent to `make test`.

3.0.1.3 Using CTest

The tests can also be ran through CTest from the command line, for example when in a configured build directory:

```
ctest -T start -T build -T test -T submit
```

Will start a new test campaign (usually in `Experimental` mode), build Ginkgo with the set configuration, run the tests and submit the results to our CDash dashboard.

Another option is to use Ginkgo's CTest script which is configured to build Ginkgo with default settings, runs the tests and submits the test to our CDash dashboard automatically.

To run the script, use the following command:

```
ctest -S cmake/CTestScript.cmake
```

The default settings are for our own CI system. Feel free to configure the script before launching it through variables or by directly changing its values. A documentation can be found in the script itself.

Chapter 4

Running the benchmarks

In addition to the unit tests designed to verify correctness, Ginkgo also includes an extensive benchmark suite for checking its performance on all Ginkgo supported systems. The purpose of Ginkgo's benchmarking suite is to allow easy and complete reproduction of Ginkgo's performance, and to facilitate performance debugging as well. Most results published in Ginkgo papers are generated thanks to this benchmarking suite and are accessible online under the `ginkgo-data repository`. These results can also be used for performance comparison in order to ensure that you get similar performance as what is published on this repository.

To compile the benchmarks, the flag `-GINKGO_BUILD_BENCHMARKS=ON` has to be set during the `cmake` step. In addition, the `ssget command-line utility` has to be installed on the system. The purpose of this file is to explain in detail the capacities of this benchmarking suite as well as how to properly setup everything.

Here is a short description of the content of this file:

1. Ginkgo setup and best practice guidelines
2. Installing and using the `ssget` tool to fetch the `SuiteSparse matrices`.
3. Benchmarking overview and how to run them in a simple way.
4. How to publish the benchmark results online and use the `Ginkgo Performance Explorer (GPE)` for performance analysis (optional).
5. Using the benchmark suite for performance debugging thanks to the loggers.
6. All available benchmark customization options.

4.0.1 1: Ginkgo setup and best practice guidelines

Before benchmarking Ginkgo, make sure that you follow the general guidelines in order to ensure best performance.

1. The code should be compiled in `Release` mode.
2. Make sure the machine has no competing jobs. On a Linux machine multiple commands can be used, `last` shows the currently opened sessions, `top` or `htop` allows to show the current machine load, and if considering using specific GPUs, `nvidia-smi` or `rocm-smi` can be used to check their load.
3. By default, Ginkgo's benchmarks will always do at least one warm-up run. For better accuracy, every benchmark is also averaged over 10 runs, except for the solver benchmark which are usually fairly long. These parameters can be tuned at the command line to either shorten benchmarking time or improve benchmarking accuracy.

In addition, the following specific options can be considered:

1. When specifically using the adaptive block jacobi preconditioner, enable the `GINKGO_JACOBI_FULL_O←PTIMIZATIONS` CMake flag. Be careful that this will use much more memory and time for the compilation due to compiler performance issues with register optimizations, in particular.
2. The current benchmarking setup also allows to benchmark only the overhead by using as either (or for all) preconditioner/spmv/solver, the special `overhead` LinOp. If your purpose is to check Ginkgo's overhead, make sure to try this mode.

4.0.2 2: Using ssget to fetch the matrices

The benchmark suite tests Ginkgo's performance using the `SuiteSparse matrix collection` and artificially generated matrices. The suite sparse collection will be downloaded automatically when the benchmarks are run. This is done thanks to the `ssget command-line utility`.

To install `ssget`, access the repository and copy the file `ssget` into a directory present in your `PATH` variable as per the tool's `README.md` instructions. The tool can be installed either in a global system path or a local directory such as `$HOME/.local/bin`. After installing the tool, it is important to review the `ssget` script and configure as needed the variable `ARCHIVE_LOCATION` on line 39. This is where the matrices will be stored into.

The Ginkgo benchmark can be set to run on only a portion of the SuiteSparse matrix collection as we will see in the following section. Please note that the entire collection requires roughly 100GB of disk storage in its compressed format, and roughly 25GB of additional disk space for intermediate data (such us uncompressing the archive). Additionally, the benchmark runs usually take a long time (SpMV benchmarks on the complete collection take roughly 24h using the K20 GPU), and will stress the system.

Before proceeding, it can be useful in order to save time to download the matrices as preparation. This can be done by using the `ssget -f -i i` command where `i` is the ID of the matrix to be downloaded. The following loop allows to download the full SuiteSparse matrix collection:

```
for i in $(seq 0 $(ssget -n)); do
    ssget -f -i ${i}
done
```

Note that `ssget` can also be used to query properties of the matrix and filter the matrices which are downloaded. For example, the following will download only positive definite matrices with less than 500M non zero elements and 10M columns. Please refer to the `ssget documentation` for more information.

```
for i in $(seq 0 $(ssget -n)); do
    posdef=$(ssget -p posdef -i ${i})
    cols=$(ssget -p cols -i ${i})
    nnz=$(ssget -p nonzeros -i ${i})
    if [ "$posdef" -eq 1 -a "$cols" -lt 10000000 -a "$nnz" -lt 500000000 ]; then
        ssget -f -i ${i}
    fi
done
```

4.0.3 3: Benchmarking overview

The benchmark suite is invoked using the `make benchmark` command in the build directory. Under the hood, this command simply calls the script `benchmark/run_all_benchmarks.sh` so it is possible to manually launch this script as well. The behavior of the suite can be modified using environment variables. Assuming the `bash` shell is used, these can either be specified via the `export` command to persist between multiple runs:

```
export VARIABLE="value"
...
make benchmark
```

or specified on the fly, on the same line as the `make benchmark` command:

```
VARIABLE="value" ... make benchmark
```

Since make sets any variables passed to it as temporary environment variables, the following shorthand can also be used:

```
make benchmark VARIABLE="value" ...
```

A combination of the above approaches is also possible (e.g. it may be useful to export the SYSTEM_NAME variable, and specify the others at every benchmark run).

The benchmark suite can take a number of configuration parameters. Benchmarks can be run only for sparse matrix vector products (spmv), for full solvers (with or without preconditioners), or for preconditioners only when supported. The benchmark suite also allows to target a sub-part of the SuiteSparse matrix collection. For details, see the available benchmark options. Here are the most important options:

- BENCHMARK={spmv, solver, preconditioner} - allows to select the type of benchmark to be ran.
- EXECUTOR={reference,cuda,hip,omp,dpcpp} - select the executor and platform the benchmarks should be ran on.
- SYSTEM_NAME=<name> - a name which will be used to designate this platform (e.g. V100, RadeonVII, ...).
- SEGMENTS=<N> - Split the benchmarked matrix space into <N> segments. If specified, SEGMENT_ID also has to be set.
- SEGMENT_ID=<I> - used in combination with the SEGMENTS variable. <I> should be an integer between 1 and <N>, the number of SEGMENTS. If specified, only the <I>-th segment of the benchmark suite will be run.
- BENCHMARK_PRECISION - defines the precision the benchmarks are run in. Supported values are↩ : "double" (default), "single", "dcomplex" and "scomplex"
- MATRIX_LIST_FILE=/path/to/matrix_list.file - allows to list SuiteSparse matrix id or name to benchmark. As an example, a matrix list file containing the following will ensure that benchmarks are ran for only those three matrices:

  ``` 1903 Freescale/circuit5M thermal2 ```

4.0.3.1 3.1: Benchmarking experimental batched functionality

4.0.3.1.1 Arranging your matrices. To allow for benchmarking of the batched functionality in Ginkgo, you need to arrange the matrix in the following fashion:

Store path to matrices in ${BATCH_MATRIX_FOLDER}

```
matrix_class_name
\
 \---0
      \---A.mtx
      \---b.mtx
      \---S.mtx
 \---1
      \---A.mtx
      \---b.mtx
      \---S.mtx
 \---2
      \---A.mtx
      \---b.mtx
      \---S.mtx
   .
   .
   .
```

In a similar fashion to the above usual Ginkgo benchmarking, you can set the following variables:

- EXECUTOR=<option> - Select the executor to benchmark, Current supported options are: cuda and reference.
```

- `BENCHMARK=batch_solver` - Benchmark the batch solvers.
- `FORMATS=batch_csr` - Benchmark with the `BatchCsr` sparse matrix format.
- `BATCH_SOLVERS=<solvers>` - The names of the batch solvers to benchmark, Options include : `bicgstab`, `gmres`, `direct`(Uses CUBLAS Dense direct). Separated by commas.
- `PRECONDS=<preconditioner>` - The preconditioner to use. Only `jacobi` or `none` are currently supported.
- `NUM_SHARED_VECS=<I>`- The number of vectors in shared memory for the batch BiCGSTAB solver. This can be reduced to allow for solution of larger system matrices and may be requried for them.
- `SOLVERS_RHS=<option>` - The RHS to use for the benchmarking. Options include `file` (Read from file, see folder structure above) and `1` (all ones), `random` (use random values).
- `SOLVERS_PRECISION=<double>` - The tolerance to solve for the batched iterative solvers.
- `SOLVERS_MAX_ITERATIONS=<int>` - The maximum number of iterations to perform for each of the systems. Each system is independent and will iterate for these many iterations if it has not reached the convergence level.
- `PRINT_RES_ITERS=<0 or 1>` - Whether to print the residuals, norms and errors.
- `DETAILED=<0 or 1>` - Whether to run and gather detailed information. This prints the timings for some individual kernel timings.
- `BATCH_SCALING=<option>` - Whether to use a pre-scaling to improve the conditioning of the matrix for the solve. Options include `explicit` (read from file), `implicit` (look for `A_scaled.mtx`) or `none`.
- `BATCH_MATRIX_LIST=<filepath>` - A text file containing the class names to be benchmarked.
- `BATCH_MATRIX_FOLDER=<foldername>` - The folder containing the matrix classes arranged in a structure as shown above.
- `NUM_BATCH_DUP=<int>` - Number of times to duplicate the provided batch systems. For example, if the `BATCH_MATRIX_FOLDER` contained 20 matrices, and `NUM_BATCH_DUP` was set to 3, then the total number of matrices would be equal to 60. This allows for performing scaling tests.
- `USE_SUITE_SPARSE=<0 or 1>` - Whether to use matrices from SuiteSparse instead. In this case, the suite sparse matrix will be downloaded (ssget must be available) and used for individual batches (if duplication is set.)

4.0.4 4: Publishing the results on Github and analyze the results with the GPE (optional)

The previous experiments generated json files for each matrices, each containing timing, iteration count, achieved precision, ... depending on the type of benchmark run. These files are available in the directory `${ginkgo_build_dir}/benchmark/results/`. These files can be analyzed and processed through any tool (e.g. python). In this section, we describe how to generate the plots by using Ginkgo's GPE tool. First, we need to publish the experiments into a Github repository which will be then linked as source input to the GPE. For this, we can simply fork the ginkgo-data repository. To do so, we can go to the github repository and use the forking interface: https://github.com/ginkgo-project/ginkgo-data/

Once it's done, we want to clone the repository locally, put all results online and access the GPE for plotting the results. Here are the detailed steps:

```
git clone https://github.com/<username>/ginkgo-data.git $HOME/ginkgo_benchmark/ginkgo-data
# send the benchmarked data to the ginkgo-data repository
# If needed, remove the old data so that no previous data is left.
# rm -r ${HOME}/ginkgo_benchmark/ginkgo-data/data/${SYSTEM_NAME}
rsync -rtv ${ginkgo_build_dir}/benchmark/results/ $HOME/ginkgo_benchmark/ginkgo-data/data/
cd ${HOME}/ginkgo_benchmark/ginkgo-data/data/
# The following updates the main `.json` files with the list of data.
# Ensure a python 3 installation is available.
```

```
./build-list . > list.json
./agregate < list.json > agregate.json
./represent . > represent.json
git config --local user.name "<Name>"
git config --local user.email "<email>"
git commit -am "Ginkgo benchmark ${BENCHMARK} of ${SYSTEM_NAME}..."
git push
```

Note that depending on what data is of interest, you may need to update the scripts `build-list` or `agregate` to change which files you want to agglomerate and summarize (depending on the system name), or which data you want to select (solver results, spmv results, ...).

For the generating the plots in the GPE, here are the steps to go through:

1. Access the GPE: `https://ginkgo-project.github.io/gpe/`
2. Update data root URL, from `https://raw.githubusercontent.com/ginkgo-project/ginkgo-data/mas` to `https://raw.githubusercontent.com/<username>/ginkgo-data/<branch>/data`
3. Click on the arrow to load the data, select the `Result Summary` entry above.
4. Click on `select an example` to choose a plotting script. Multiple scripts are available by default in different branches. You can use the `jsonata` and `chartjs` languages to develop your own as well.
5. The results should be available in the tab "plot" on the right side. Other tabs allow to access the result of the processed data after invoking the processing script.

4.0.5 5: Detailed performance analysis and debugging

Detailed performance analysis can be ran by passing the environment variable `DETAILED=1` to the benchmarking script. This detailed run is available for solvers and allows to log the internal residual after every iteration as well as log the time taken by all operations. These features are also available in the `performance-debugging` example which can be used instead and modified as needed to analyze Ginkgo's performance.

These features are implemented thanks to the loggers located in the file `${ginkgo_src_dir}/benchmark/utils/loggers` Ginkgo possesses hooks at all important code location points which can be inspected thanks to the logger. In this fashion, it is easy to use these loggers also for tracking memory allocation sizes and other important library aspects.

4.0.6 6: Available benchmark options

There are a set amount of options available for benchmarking. Most important options can be configured through the benchmarking script itself thanks to environment variables. Otherwise, some specific options are not available through the benchmarking scripts but can be directly configured when running the benchmarking program itself. For a list of all options, run for example `${ginkgo_build_dir}/benchmark/solver/solver --help`.

The supported environment variables are described in the following list:

- `BENCHMARK={spmv, solver, preconditioner}` - allows to select the type of benchmark to be ran. Default is `spmv`.
  - `spmv` - Runs the sparse matrix-vector product benchmarks on the SuiteSparse collection.
  - `solver` - Runs the solver benchmarks on the SuiteSparse collection. The matrix format is determined by running the `spmv` benchmarks first, and using the fastest format determined by that benchmark.
  - `preconditioner` - Runs the preconditioner benchmarks on artificially generated block-diagonal matrices.

- EXECUTOR={reference,cuda,hip,omp,dpcpp} - select the executor and platform the benchmarks should be ran on. Default is cuda.
- SYSTEM_NAME=<name> - a name which will be used to designate this platform (e.g. V100, RadeonVII, ...) and not overwrite previous results. Default is unknown.
- SEGMENTS=<N> - Split the benchmarked matrix space into <N> segments. If specified, SEGMENT_ID also has to be set. Default is 1.
- SEGMENT_ID=<I> - used in combination with the SEGMENTS variable. <I> should be an integer between 1 and <N>, the number of SEGMENTS. If specified, only the <I>-th segment of the benchmark suite will be run. Default is 1.
- MATRIX_LIST_FILE=/path/to/matrix_list.file - allows to list SuiteSparse matrix id or name to benchmark. As an example, a matrix list file containing the following will ensure that benchmarks are ran for only those three matrices: ``` 1903 Freescale/circuit5M thermal2 ``∗DEVICE_ID- the accelerator device ID to target for the benchmark. The default is0. ∗DRY_RUN={true, false}- If set totrue`, prepares the system for the benchmark runs (downloads the collections, creates the result structure, etc.) and outputs the list of commands that would normally be run, but does not run the benchmarks themselves. Default is false.
- PRECONDS={jacobi,ic,ilu,paric,parict,parilu,parilut,ic-isai,ilu-isai,paric-isai,pari
the preconditioners to use for either solver or preconditioner benchmarks. Multiple options can be passed to this variable. Default is none.
- FORMATS={csr,coo,ell,hybrid,sellp,hybridxx,cusparse_xx,hipsparse_xx} the matrix formats to benchmark for the spmv phase of the benchmark. Run ${ginkgo_build_↩ dir}/benchmark/spmv/spmv --help for a full list. If needed, multiple options for hybrid with different optimization parameters are available. Depending on the libraries available at build time, vendor library formats (cuSPARSE with cusparse_ prefix or hipSPARSE with hipsparse_ prefix) can be used as well. Multiple options can be passed. The default is csr,coo,ell,hybrid,sellp.
- SOLVERS={bicgstab,bicg,cg,cgs,fcg,gmres,cb_gmres_{keep,reduce1,reduce2,integer,iredu
_trs,upper_trs}
  - the solvers which should be benchmarked. Multiple options can be passed. The default is bicgstab,cg,cgs,fcg,gmres,idr. Note that lower/upper_trs by default don't use a preconditioner, as they are by default exact direct solvers.
- SOLVERS_PRECISION=<precision> - the minimal residual reduction before which the solver should stop. The default is 1e-6.
- SOLVERS_MAX_ITERATION=<number> - the maximum number of iterations with which a solver should be ran. The default is 10000.
- SOLVERS_RHS={1, random, sinus} - whether to use a vector of all ones, random values or b = A ∗ (s / |s|)$ with s(idx) = sin(idx) (for complex numbers, s(idx) = sin(2∗idx) + i ∗ sin(2∗idx+1)) as the right-hand side in solver benchmarks. Default is 1.
- SOLVERS_INITIAL_GUESS={rhs,0,random} - the initial guess generation of the solvers. rhs uses the right-hand side, 0 uses a zero vector and random generates a random vector as the initial guess.
- DETAILED={0,1} - selects whether detailed benchmarks should be ran for the solver benchmarks, can be either 0 (off) or 1 (on). The default is 0.
- GPU_TIMER={true, false} - If set to true, use the gpu timer, which is valid for cuda/hip executor, to measure the timing. Default is false.
- SOLVERS_JACOBI_MAX_BS - sets the maximum block size for the Jacobi preconditioner (if used, otherwise, it does nothing) in the solvers benchmark. The default is '32'.
- SOLVERS_GMRES_RESTART - the maximum dimension of the Krylov space to use in GMRES. The default is 100.

Chapter 5

Contributing guidelines

We are glad that you are interested in contributing to Ginkgo. Please have a look at our coding guidelines before proposing a pull request.

5.1 Table of Contents

- Create a new algorithm
- Converting CUDA code to HIP code

Writing Tests

- Testing know-how
- Some general rules
- Writing tests for kernels

Documentation style

- Developer targeted notes
- Whitespaces
- Documenting examples

Other programming comments

- C++ standard stream objects
- Warnings
- Avoiding circular dependencies

5.2 Most important stuff (A TL;DR)

- `GINKGO_DEVEL_TOOLS` needs to be set to `on` to commit. This requires `clang-format` to be installed. See Automatic code formatting for more details. Once installed, you can run `make format` in your `build/` folder to automatically format your modified files. As `make format` unstages your files post-formatting, you must stage the files again once you have verified that `make format` has done the appropriate formatting, before committing the files.
- See Our git workflow to get a quick overview of our workflow.
- See Creating, Reviewing and Merging Pull Requests on how to create a Pull request.

5.3 Project structure

Ginkgo is divided into a `core` module with common functionalities independent of the architecture, and several kernel modules (`reference`, `omp`, `cuda`, `hip`, `dpcpp`) which contain low-level computational routines for each supported architecture.

5.3.1 Extended header files

Some header files from the core module have to be extended to include special functionality for specific architectures. An example of this is `core/base/math.hpp`, which has a GPU counterpart in `cuda/base/math.hpp`. For such files you should always include the version from the module you are working on, and this file will internally include its `core` counterpart.

5.3.2 Using library classes

You can use and call functions of existing classes inside a kernel (that are defined and not just declared in a header file), however, you are not allowed to create new instances of a polymorphic class inside a kernel (or in general inside any kernel module like cuda/hip/omp/reference) as this creates circular dependencies between the `core` and the backend library. With this in mind, our CI contains a job which checks if such a circular dependency exists. These checks can be run manually using the `-DGINKGO_CHECK_CIRCULAR_DEPS=ON` option in the CMake configuration.

For example, when creating a new matrix class `AB` by combining existing classes `A` and `B`, the `AB::apply()` function composed of invocations to `A::apply()` and `B::apply()` can only be defined in the core module, it is not possible to create instances of `A` and `B` inside the `AB` kernel files. This is to avoid the aforementioned circular dependency issue. An example for such a class is the `Hybrid` matrix format, which uses the `apply()` of the `Ell` and `Coo` matrix formats. Nevertheless, it is possible to call the kernels themselves directly within the same executor. For example, `cuda::dense::add_scaled()` can be called from any other `cuda` kernel.

5.4 Git related

Ginkgo uses git, the distributed version control system to track code changes and coordinate work among its developers. A general guide to git can be found in `its extensive documentation`.

5.4.1 Our git workflow

In Ginkgo, we prioritize keeping a clean history over accurate tracking of commits. `git rebase` is hence our command of choice to make sure that we have a nice and linear history, especially for pulling the latest changes from the `develop` branch. More importantly, rebasing upon develop is **required** before the commits of the PR are merged into the `develop` branch.

5.4.2 Writing good commit messages

With software sustainability and maintainability in mind, it is important to write commit messages that are short, clear and informative. Ideally, this would be the format to prefer:

```
Summary of the changes in a sentence, max 50 chars.
More detailed comments:
+ Changes that have been added.
- Changes that been removed.
Related PR: https://github.com/ginkgo-project/ginkgo/pull/<PR-number>
```

You can refer to `this informative guide` for more details.

5.4.2.1 Attributing credit

Git has a nice feature where it allows you to add a co-author for your commit, if you would like to attribute credits for the changes made in the commit. This can be done by:

```
Commit message.
Co-authored-by: Name <email@domain>
```

In the Ginkgo commit history, this is most common associated with suggested improvements from code reviews.

5.4.3 Creating, Reviewing and Merging Pull Requests

- The `develop` branch is the default branch to submit PR's to. From time to time, we merge the `develop` branch to the `master` branch and create tags on the `master` to create new releases of Ginkgo. Therefore, all pull requests must be merged into `develop`.
- Please have a look at the labels and make sure to add the relevant labels.
- You can mark the PR as a `WIP` if you are still working on it, `Ready for Review` when it is ready for others to review it.
- Assignees to the PR should be the ones responsible for merging that PR. Currently, it is only possible to assign members within the `ginkgo-project`.
- Each pull request requires at least two approvals before merging.
- PR's created from within the repository will automatically trigger two CI pipelines on pushing to the branch from the which the PR has been created. The Github Actions pipeline tests our framework on Mac OSX and on Windows platforms. Another comprehensive Linux based pipeline is run from a `mirror on gitlab` and contains additional checks like static analysis and test coverage.
- Once a PR has been approved and the build has passed, one of the reviewers can mark the PR as `READY TO MERGE`. At this point the creator/assignee of the PR *needs to* verify that the branch is up to date with `develop` and rebase it on `develop` if it is not.

5.5 Code style

5.5.1 Automatic code formatting

Ginkgo uses `ClangFormat` (executable is usually named `clang-format`) and a custom `.clang-format` configuration file (mostly based on ClangFormat's *Google* style) to automatically format your code. **Make sure you have ClangFormat set up and running properly** (you should be able to run `make format` from Ginkgo's build directory) before committing anything that will end up in a pull request against `ginkgo-project/ginkgo` repository. In addition, you should **never** modify the `.clang-format` configuration file shipped with Ginkgo. E.g. if ClangFormat has trouble reading this file on your system, you should install a newer version of ClangFormat, and avoid commenting out parts of the configuration file.

ClangFormat is the primary tool that helps us achieve a uniform look of Ginkgo's codebase, while reducing the learning curve of potential contributors. However, ClangFormat configuration is not expressive enough to incorporate the entire coding style, so there are several additional rules that all contributed code should follow.

Note: To learn more about how ClangFormat will format your code, see existing files in Ginkgo, `.clang-format` configuration file shipped with Ginkgo, and ClangFormat's documentation.

5.5.2 Naming scheme

5.5.2.1 Filenames

Filenames use `snake_case` and use the following extensions:

- C++ source files: `.cpp`
- C++ header files: `.hpp`

- CUDA source files: `.cu`
- CUDA header files: `.cuh`
- HIP source files: `.hip.cpp`
- HIP header files: `.hip.hpp`
- Common source files used by both CUDA and HIP: `.hpp.inc`
- CMake utility files: `.cmake`
- Shell scripts: `.sh`

Note: A C++ source/header file is considered a `CUDA` file if it contains CUDA code that is not guarded with `#if` guards that disable this code in non-CUDA compilers. I.e. if a file can be compiled by a general C++ compiler, it is not considered a CUDA file.

5.5.2.2 Macros

Macros (both object-like and function-like macros) use `CAPITAL_CASE`. They have to start with `GKO_` to avoid name clashes (even if they are `#undef`-ed in the same file!).

5.5.2.3 Variables

Variables use `snake_case`.

5.5.2.4 Constants

Constants use `snake_case`.

5.5.2.5 Functions

Functions use `snake_case`.

5.5.2.6 Structures and classes

Structures and classes which do not experience polymorphic behavior (i.e. do not contain virtual methods, nor members which experience polymorphic behavior) use `snake_case`.

All other structures and classes use `CamelCase`.

5.5.2.7 Members

All structure / class members use the same naming scheme as they would if they were not members:

- methods use the naming scheme for functions
- data members the naming scheme for variables or constants
- type members for classes / structures

Additionally, non-public data members end with an underscore (_).

5.5.2.8 Namespaces

Namespaces use `snake_case`.

5.5.2.9 Template parameters

- Type template parameters use `CamelCase`, for example `ValueType`.
- Non-type template parameters use `snake_case`, for example `subwarp_size`.

5.5.3 Whitespace

Spaces and tabs are handled by ClangFormat, but blank lines are only partially handled (the current configuration doesn't allow for more than 2 blank lines). Thus, contributors should be aware of the following rules for blank lines:

1. Top-level statements and statements directly within namespaces are separated with 2 blank lines. The first / last statement of a namespace is separated by two blank lines from the opening / closing brace of the namespace.

   (a) *exception*: if the first **or** the last statement in the namespace is another namespace, then no blank lines are required *example*: ```c++ namespace foo {

```
    struct x {
    };


    }  // namespace foo


    namespace bar {
    namespace baz {


    void f();


    }  // namespace baz
    }  // namespace bar
    ```

2. _exception_: in header files whose only purpose is to _declare_ a bunch
    of functions (e.g. the `*_kernel.hpp` files) these declarations can be
    separated by only 1 blank line (note: standard rules apply for all other
    statements that might be present in that file)
3. _exception_: "related" statement can have 1 blank line between them.
    "Related" is not a strictly defined adjective in this sense, but is in
    general one of:

    1. overload of a same function,
    2. function / class template and it's specializations,
    3. macro that modifies the meaning or adds functionality to the
        previous / following statement.

    However, simply calling function `f` from function `g` does not imply
    that `f` and `g` are "related".
```

1. Statements within structures / classes are separated with 1 blank line. There are no blank lines betweeen the first / last statement in the structure / class.
```

(a) *exception*: there is no blank line between an access modifier (`private`, `protected`, `public`) and the following statement. *example*: ```c++ class foo { public: int get_x() const noexcept { return x_; }
int &get_x() noexcept { return x_; }
private: int x_; }; ```

2. Function bodies cannot have multiple consecutive blank lines, and a single blank line can only appear between two logical sections of the function.

3. Unit tests should follow the AAA pattern, and a single blank line must appear between consecutive "A" sections. No other blank lines are allowed in unit tests.

4. Enumeration definitions should have no blank lines between consecutive enumerators.

5.5.4 Include statement grouping

In general, all include statements should be present on the top of the file, ordered in the following groups, with two blank lines between each group:

1. Related header file (e.g. `core/foo/bar.hpp` included in `core/foo/bar.cpp`, or in the unit test`core/test/foo/bar.cpp`)
2. Standard library headers (e.g. `vector`)
3. Executor specific library headers (e.g. `omp.h`)
4. System third-party library headers (e.g. `papi.h`)
5. Local third-party library headers
6. Public Ginkgo headers
7. Private Ginkgo headers

Example: A file `core/base/my_file.cpp` might have an include list like this:

```
{c++}
#include <ginkgo/core/base/my_file.hpp>
#include <algorithm>
#include <vector>
#include <tuple>
#include <omp.h>
#include <papi.h>
#include "third_party/blas/cblas.hpp"
#include "third_party/lapack/lapack.hpp"
#include <ginkgo/core/base/executor.hpp>
#include <ginkgo/core/base/lin_op.hpp>
#include <ginkgo/core/base/types.hpp>
#include "core/base/my_file_kernels.hpp"
```

5.5.4.1 Main header

This section presents general rules used to define the main header attributed to the file. In the previous example, this would be `#include <ginkgo/core/base/my_file.hpp>`.

General rules:

1. Some fixed main header.
2. components:
   - with `_kernel` suffix looks for the header in the same folder.

- without `_kernel` suffix looks for the header in `core`.

3. `test/utils`: looks for the header in `core`
4. `core`: looks for the header in `ginkgo`
5. `test` or `base`: looks for the header in `ginkgo/core`
6. others: looks for the header in `core`

Note: Please see the detail in the `dev_tools/scripts/config`.

5.5.4.2 Some general comments.

1. Private headers of Ginkgo should not be included within the public Ginkgo header.
2. It is a good idea to keep the headers self-sufficient, See [Google Style guide for reasoning](). When compiling with `GINKGO_CHECK_CIRCULAR_DEPS` enabled, this property is explicitly checked.
3. The recommendations of the `iwyu` (Include what you use) tool can be used to make sure that the headers are self-sufficient and that the compiled files (`.cu`, `.cpp`, `.hip.cpp`) include only what they use. A [CI pipeline]() is available that runs with the `iwyu` tool. Please be aware that this tool can be incorrect in some cases.

5.5.4.3 Automatic header arrangement

1. `dev_tools/script/format_header.sh` will take care of the group/sorting of headers according to this guideline.
2. `make format_header` arranges the header of the modified files in the branch.
3. `make format_header_all` arranges the header of all files.

5.5.5 Other Code Formatting not handled by ClangFormat

5.5.5.1 Control flow constructs

Single line statements should be avoided in all cases. Use of brackets is mandatory for all control flow constructs (e.g. `if`, `for`, `while`, ...).

5.5.5.2 Variable declarations

C++ supports declaring / defining multiple variables using a single *type-specifier*. However, this is often very confusing as references and pointers exhibit strange behavior:

```
 {c++}
template <typename T> using pointer = T *;
int *         x, y;  // x is a pointer, y is not
pointer<int> x, y;  // both x and y are pointers
```

For this reason, **always** declare each variable on a separate line, with its own *type-specifier*.

5.5.6 CMake coding style

5.5.6.1 Whitespaces

All alignment in CMake files should use four spaces.

5.5.6.2 Use of macros vs functions

Macros in CMake do not have a scope. This means that any variable set in this macro will be available to the whole project. In contrast, functions in CMake have local scope and therefore all set variables are local only. In general, wrap all piece of algorithms using temporary variables in a function and use macros to propagate variables to the whole project.

5.5.6.3 Naming style

All Ginkgo specific variables should be prefixed with a `GINKGO_` and all functions by `ginkgo_`.

5.6 Helper scripts

To facilitate easy development within Ginkgo and to encourage coders and scientists who do not want get bogged down by the details of the Ginkgo library, but rather focus on writing the algorithms and the kernels, Ginkgo provides the developers with a few helper scripts.

5.6.1 Create a new algorithm

A `create_new_algorithm.sh` script is available for developers to facilitate easy addition of new algorithms. The options it provides can be queried with

```
./create_new_algorithm.sh --help
```

The main objective of this script is to add files and boiler plate code for the new algorithm using a model and an instance of that model. For example, models can be any one of `factorization`, `matrix`, `preconditioner` or `solver`. For example to create a new solver named `my_solver` similar to `gmres`, you would set the `ModelType` to `solver` and set the `ModelName` to `gmres`. This would duplicate the core algorithm and kernels of the `gmres` algorithm and replace the naming to `my_solver`. Additionally, all the kernels of the new `my_solver` are marked as `GKO_NOT_IMPLEMENTED`. For easy navigation and `.txt` file is created in the folder where the script is run, which lists all the TODO's. These TODO's can also be found in the corresponding files.

5.6.2 Converting CUDA code to HIP code

We provide a `cuda2hip` script that converts `cuda` kernel code into `hip` kernel code. Internally, this script calls the `hipify script` provided by HIP, converting the CUDA syntax to HIP syntax. Additionally, it also automatically replaces the instances of CUDA with HIP as appropriate. Hence, this script can be called on a Ginkgo CUDA file. You can find this script in the `dev_tools/scripts/` folder.

5.7 Writing Tests

Ginkgo uses the GTest framework for the unit test framework within Ginkgo. Writing good tests are extremely important to verify the functionality of the new code and to make sure that none of the existing code has been broken.

5.7.1 Testing know-how

- GTest provides a comprehensive documentation of the functionality available within Gtest.
- Reduce code duplication with Testing Fixtures, TEST_F
- Write templated tests using TYPED_TEST.

5.7.2 Some general rules.

- Unit tests must follow the KISS principle.
- Unit tests must follow the AAA pattern, and a single blank line must appear between consecutive "A" sections.

5.7.3 Writing tests for kernels

- Reference kernels, kernels on the `ReferenceExecutor`, are meant to be single threaded reference implementations. Therefore, tests for reference kernels need to be performed with data that can be as small as possible. For example, matrices lesser than 5x5 are acceptable. This allows the reviewers to verify the results for exactness with tools such as MATLAB.
- OpenMP, CUDA, HIP and DPC++ kernels have to be tested against the reference kernels. Hence data for the tests of these kernels can be generated in the test files using helper functions or by using external files to be read through the standard input. In particular for CUDA, HIP and DPC++ the data size should be at least bigger than the architecture's warp size to ensure there is no corner case in the kernels.

5.8 Documentation style

Documentation uses standard Doxygen.

5.8.1 Developer targeted notes

Make use of `@internal` doxygen tag. This can be used for any comment which is not intended for users, but is useful to better understand a piece of code.

5.8.2 Whitespaces

5.8.2.1 After named tags such as <tt>@param foo</tt>

The documentation tags which use an additional name should be followed by two spaces in order to better distinguish the text from the doxygen tag. It is also possible to use a line break instead.

5.8.3 Documenting examples

There are two main steps:

1. First, you can just copy over the doc/ folder (you can copy it from the example most relevant to you) and adapt your example names and such, then you can modify the actual documentation.

- In `tooltip`: A short description of the example.
- In `short-intro`: The name of the example.
- In `results.dox`: Run the example and write the output you get.
- In `kind`: The kind of the example. For different kinds see the documentation. Examples can be of `basic`, `techniques`, `logging`, `stopping_criteria` or `preconditioners`. If your example does not fit any of these categories, feel free to create one.
- In `intro.dox`: You write an explanation of your code with some introduction similar to what you see in an existing example most relevant to you.
- In `builds-on`: You write the examples it builds on.

1. You also need to modify the examples.hpp.in file. You add the name of the example in the main section and in the section that you specified in the `doc/kind` file in the example documentation.

5.9 Other programming comments

5.9.1 C++ standard stream objects

These are global objects and are shared inside the same translation unit. Therefore, whenever its state or formatting is changed (e.g. using `std::hex` or floating point formatting) inside library code, make sure to restore the state before returning the control to the user. See this stackoverflow question for examples on how to do it correctly. This is extremely important for header files.

5.9.2 Warnings

By default, the `-DGINKGO_COMPILER_FLAGS` is set to `-Wpedantic` and hence pedantic warnings are emitted by default. Some of these warnings are false positives and a complete list of the resolved warnings and their solutions is listed in Issue 174. Specifically, when macros are being used, we have the issue of having `extra ;` warnings, which is resolved by adding a `static_assert()`. The CI system additionally also has a step where it compiles for pedantic warnings to be errors.

5.9.3 Avoiding circular dependencies

To facilitate finding circular dependencies issues (see Using library classes for more details), a CI step `no-circular-deps` was created. For more details on its usage, see this pipeline, where Ginkgo did not abide to this policy and PR #278 which fixed this. Note that doing so is not enough to guarantee with 100% accuracy that no circular dependency is present. For an example of such a case, take a look at this pipeline where one of the compiler setups detected an incorrect dependency of the `cuda` module (due to jacobi) on the `core` module.

Chapter 6

Citing Ginkgo

The main Ginkgo paper describing Ginkgo's purpose, design and interface is available through the following reference:

```
@article{ginkgo-toms-2022,
title = {{Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing}},
volume = {48},
copyright = {All rights reserved},
issn = {0098-3500},
shorttitle = {Ginkgo},
url = {https://doi.org/10.1145/3480935},
doi = {10.1145/3480935},
number = {1},
urldate = {2022-02-17},
journal = {ACM Transactions on Mathematical Software},
author = {Anzt, Hartwig and Cojean, Terry and Flegar, Goran and Göbel, Fritz and Grützmacher, Thomas and
       Nayak, Pratik and Ribizel, Tobias and Tsai, Yuhsiang Mike and Quintana-Ortí, Enrique S.},
month = feb,
year = {2022},
keywords = {ginkgo, healthy software lifecycle, High performance computing, multi-core and manycore
       architectures},
pages = {2:1--2:33}
}
```

Multiple topical papers exist on Ginkgo and its algorithms. The following papers can be used to cite specific aspects of the Ginkgo project.

6.0.1 The Ginkgo Software

The Ginkgo software itself was reviewed and has a paper published in the Journal of Open Source Software, which can be cited with the following reference:

```
@article{GinkgoJoss2020,
  doi = {10.21105/joss.02260},
  url = {https://doi.org/10.21105/joss.02260},
  year = {2020},
  publisher = {The Open Journal},
  volume = {5},
  number = {52},
  pages = {2260},
  author = {Hartwig Anzt and Terry Cojean and Yen-Chen Chen and Goran Flegar and Fritz G\"{o}bel and Thomas
       Gr\"{u}tzmacher and Pratik Nayak and Tobias Ribizel and Yu-Hsiang Tsai},
  title = {Ginkgo: A high performance numerical linear algebra library},
  journal = {Journal of Open Source Software}
}
```

6.0.2 On Portability

```
@misc{tsai2020amdportability,
    title={Preparing Ginkgo for AMD GPUs -- A Testimonial on Porting CUDA Code to HIP},
    author={Yuhsiang M. Tsai and Terry Cojean and Tobias Ribizel and Hartwig Anzt},
    year={2020},
    eprint={2006.14290},
    archivePrefix={arXiv},
    primaryClass={cs.MS}
}
```

6.0.3 On Software Sustainability

```
@inproceedings{anzt2019pasccb,
author = {Anzt, Hartwig and Chen, Yen-Chen and Cojean, Terry and Dongarra, Jack and Flegar, Goran and Nayak,
       Pratik and Quintana-Ort\'{\i}, Enrique S. and Tsai, Yuhsiang M. and Wang, Weichung},
title = {Towards Continuous Benchmarking: An Automated Performance Evaluation Framework for High Performance
       Software},
year = {2019},
isbn = {9781450367707},
publisher = {Association for Computing Machinery},
address = {New York, NY, USA},
url = {https://doi.org/10.1145/3324989.3325719},
doi = {10.1145/3324989.3325719},
booktitle = {Proceedings of the Platform for Advanced Scientific Computing Conference},
articleno = {9},
numpages = {11},
keywords = {interactive performance visualization, healthy software lifecycle, continuous integration,
       automated performance benchmarking},
location = {Zurich, Switzerland},
series = {PASC '19}
}
```

6.0.4 On SpMV or solvers performance

```
@InProceedings{tsai2020amdspmv,
author="Tsai, Yuhsiang M.
and Cojean, Terry
and Anzt, Hartwig",
editor="Sadayappan, Ponnuswamy
and Chamberlain, Bradford L.
and Juckeland, Guido
and Ltaief, Hatem",
title="Sparse Linear Algebra on AMD and NVIDIA GPUs -- The Race Is On",
booktitle="High Performance Computing",
year="2020",
publisher="Springer International Publishing",
address="Cham",
pages="309--327",
abstract="Efficiently processing sparse matrices is a central and performance-critical part of many
       scientific simulation codes. Recognizing the adoption of manycore accelerators in HPC, we evaluate in
       this paper the performance of the currently best sparse matrix-vector product (SpMV) implementations
       on high-end GPUs from AMD and NVIDIA. Specifically, we optimize SpMV kernels for the CSR, COO, ELL,
       and HYB format taking the hardware characteristics of the latest GPU technologies into account. We
       compare for 2,800 test matrices the performance of our kernels against AMD's hipSPARSE library and
       NVIDIA's cuSPARSE library, and ultimately assess how the GPU technologies from AMD and NVIDIA compare
       in terms of SpMV performance.",
isbn="978-3-030-50743-5"
}
@article{anzt2020spmv,
author = {Anzt, Hartwig and Cojean, Terry and Yen-Chen, Chen and Dongarra, Jack and Flegar, Goran and Nayak,
       Pratik and Tomov, Stanimire and Tsai, Yuhsiang M. and Wang, Weichung},
title = {Load-Balancing Sparse Matrix Vector Product Kernels on GPUs},
year = {2020},
issue_date = {March 2020},
publisher = {Association for Computing Machinery},
address = {New York, NY, USA},
volume = {7},
number = {1},
issn = {2329-4949},
url = {https://doi.org/10.1145/3380930},
doi = {10.1145/3380930},
journal = {ACM Trans. Parallel Comput.},
month = mar,
articleno = {2},
numpages = {26},
keywords = {irregular matrices, GPUs, Sparse Matrix Vector Product (SpMV)}
}
@misc{tsai2020evaluating,
    title={Evaluating the Performance of NVIDIA's A100 Ampere GPU for Sparse Linear Algebra Computations},
    author={Yuhsiang Mike Tsai and Terry Cojean and Hartwig Anzt},
    year={2020},
    eprint={2008.08478},
    archivePrefix={arXiv},
    primaryClass={cs.MS}
}
```

Chapter 7

Example programs

Here you can find example programs that demonstrate the usage of Ginkgo. Some examples are built on one another and some are stand-alone and demonstrate a concept of Ginkgo, which can be used in your own code.

You can browse the available example programs

1. as **a graph** that shows how example programs build upon each other.
2. as **a list** that provides a short synopsis of each program.
3. or **grouped by topic**.

By default, all Ginkgo examples are built using CMake.

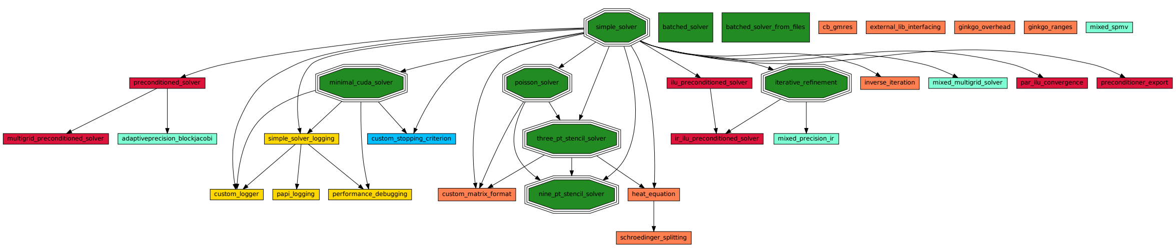
An example for building the examples and using Ginkgo as an external library without CMake can be found in the script provided for each example, which should be called with the form: `./build.sh PATH_TO_GINKGO_B↩UILD_DIR`

By default, Ginkgo is compiled with at least `-DGINKGO_BUILD_REFERENCE=ON`. Ginkgo also tries to detect your environment setup (presence of CUDA, ...) to enable the relevant accelerator modules. If you want to target a specific GPU, make sure that Ginkgo is compiled with the accelerator specific module enabled, such as:

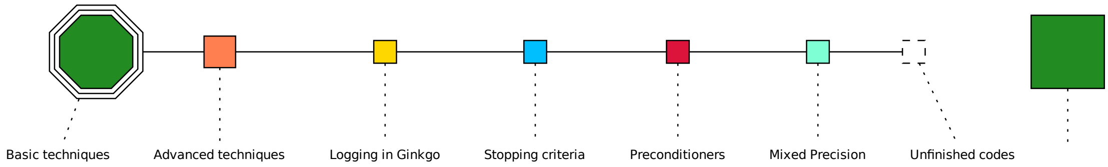
1. `-DGINKGO_BUILD_CUDA=ON` option for NVIDIA GPUs.
2. `-DGINKGO_BUILD_HIP=ON` option for AMD or NVIDIA GPUs.
3. `-DGINKGO_BUILD_DPCPP=ON` option for Intel GPUs (and possibly any other platform). </ol

Connections between example programs

The following graph shows the connections between example programs and how they build on each other. Click on any of the boxes to go to one of the programs. If you hover your mouse pointer over a box, a brief description of the program should appear.



Legend:



Example programs

The simple-solver program	A minimal CG solver in Ginkgo, which reads a matrix from a file.
The minimal-cuda-solver program	A minimal solver on the CUDA executor than can be run on NVIDIA GPU's.
The poisson-solver program	Solve an actual physically relevant problem, the poisson problem. The matrix is generated within Ginkgo.
The preconditioned-solver program	Using a Jacobi preconditioner to solve a linear system.
The ilu-preconditioned-solver program	Using an ILU preconditioner to solve a linear system.
The performance-debugging program	Using Loggers to debug the performance within Ginkgo.
The three-pt-stencil-solver program	Using a three point stencil to solve the poisson equation with array views.
The nine-pt-stencil-solver program	Using a nine point 2D stencil to solve the poisson equation with array views.
The external-lib-interfacing program	Using Ginkgo's solver with the external library deal.II.
The custom-logger program	Creating a custom logger specifically for comparing the recurrent and the real residual norms.
The custom-matrix-format program	Creating a matrix-free stencil solver by using Ginkgo's advanced methods to build your own custom matrix format.
The inverse-iteration program	Using Ginkgo to compute eigenvalues of a matrix with the inverse iteration method.
The simple-solver-logging program	Using the logging functionality in Ginkgo to get solver and other information to diagnose and debug your code.
The papi-logging program	Using the PAPI logging library in Ginkgo to get advanced information about your code and its behaviour.
The ginkgo-overhead program	Measuring the overhead of the Ginkgo library.
The custom-stopping-criterion program	Creating a custom stopping criterion for the iterative solution process.
The ginkgo-ranges program	Using the ranges concept to factorize a matrix with the LU factorization.

The mixed-spmv program	Shows the Ginkgo mixed precision spmv functionality.
The mixed-precision-ir program	Manual implementation of a Mixed Precision Iterative Refinement (MPIR) solver.
The adaptiveprecision-blockjacobi program	Shows how to use the adaptive precision block-Jacobi preconditioner.
The cb-gmres program	Using the Ginkgo CB-GMRES solver (Compressed Basis GMRES).
The heat-equation program	Solving a 2D heat equation and showing matrix assembly, vector initalization and solver setup in a more complex setting with output visualization.
The iterative-refinement program	Using a low accuracy CG solver as an inner solver to an iterative refinement (IR) method which solves a linear system.
The ir-ilu-preconditioned-solver program	Combining iterative refinement with the adaptive precision block-Jacobi preconditioner to approximate triangular systems occurring in ILU preconditioning.
The par-ilu-convergence program	Convergence analysis at the examples of parallel incomplete factorization solver.
The preconditioner-export program	Explicit generation and storage of preconditioners for given matrices.
The multigrid-preconditioned-solver program	Use multigrid as preconditioner to a solver.
The mixed-multigrid-solver program	Use multigrid with different precision multigrid_level as a solver.

Example programs grouped by topics

Solving a simple linear system with choice of executors	The simple-solver program
Debug the performance of a solver or preconditioner	The performance-debugging program The preconditioner-export program
Using the CUDA executor	The minimal-cuda-solver program
Using preconditioners	The preconditioned-solver program, The ilu-preconditioned-solver program, The ir-ilu-preconditioned-solver program, The adaptiveprecision-blockjacobi program, The par-ilu-convergence program, The preconditioner-export program The multigrid-preconditioned-solver program
Iterative refinement	The iterative-refinement program, The mixed-precision-ir program, The ir-ilu-preconditioned-solver program
Solving a physically relevant problem	The poisson-solver program, The three-pt-stencil-solver program, The nine-pt-stencil-solver program, The custom-matrix-format program

Reading in a matrix and right hand side from a file	The simple-solver program, The minimal-cuda-solver program, The preconditioned-solver program, The ilu-preconditioned-solver program, The inverse-iteration program, The simple-solver-logging program, The papi-logging program, The custom-stopping-criterion program, The custom-logger program

Basic techniques

Using Ginkgo with external libraries	The external-lib-interfacing program
Customizing Ginkgo	The custom-logger program, The custom-stopping-criterion program, The custom-matrix-format program
Writing your own matrix format	The custom-matrix-format program
Using Ginkgo to construct more complex linear algebra routines	The inverse-iteration program
Logging within Ginkgo	The simple-solver-logging program, The papi-logging program, The performance-debugging program The custom-logger program
Constructing your own stopping criterion	The custom-stopping-criterion program
Using ranges in Ginkgo	The ginkgo-ranges program
Mixed precision	The mixed-spmv program, The mixed-precision-ir program, The adaptiveprecision-blockjacobi program The mixed-multigrid-solver program
Multigrid	The multigrid-preconditioned-solver program The mixed-multigrid-solver program

Chapter 8

The adaptiveprecision-blockjacobi program

The preconditioned solver example..

This example depends on preconditioned-solver.

This example shows how to use the adaptive precision block-Jacobi preconditioner.

In this example, we first read in a matrix from file, then generate a right-hand side and an initial guess. The preconditioned CG solver is enhanced with a block-Jacobi preconditioner that optimizes the storage format for the distinct inverted diagonal blocks to the numerical requirements. The example features the iteration count and runtime of the CG solver.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
```

```
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create RHS and initial guess as 1

```
gko::size_type size = A->get_size()[0];
auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
for (auto i = 0; i < size; i++) {
    host_x->at(i, 0) = 1.;
}
auto x = gko::clone(exec, host_x);
auto b = gko::clone(exec, host_x);
```

Calculate initial residual by overwriting b

```
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto initres = gko::initialize<real_vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(initres));
```

copy b again

```
b->copy_from(host_x.get());
const RealValueType reduction_factor = 1e-7;
auto iter_stop =
    gko::stop::Iteration::build().with_max_iters(10000u).on(exec);
auto tol_stop = gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec);
std::shared_ptr<const gko::log::Convergence<ValueType» logger =
    gko::log::Convergence<ValueType>::create(exec);
iter_stop->add_logger(logger);
tol_stop->add_logger(logger);
```

Create solver factory

```
auto solver_gen =
    cg::build()
        .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
```

Add preconditioner, these 2 lines are the only difference from the simple solver example

```
.with_preconditioner(bj::build()
                         .with_max_block_size(16u)
                         .with_storage_optimization(
                             gko::precision_reduction::autodetect())
                         .on(exec))
.on(exec);
```

Create solver

```
solver_gen->add_logger(logger);
auto solver = solver_gen->generate(A);
```

Solve system

```
exec->synchronize();
std::chrono::nanoseconds time(0);
auto tic = std::chrono::steady_clock::now();
solver->apply(lend(b), lend(x));
auto toc = std::chrono::steady_clock::now();
time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
```

Calculate residual

```
auto res = gko::initialize<real_vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(res));
auto impl_res = gko::as<real_vec>(logger->get_implicit_sq_resnorm());
std::cout « "Initial residual norm sqrt(r^T r):\n";
write(std::cout, lend(initres));
std::cout « "Final residual norm sqrt(r^T r):\n";
write(std::cout, lend(res));
std::cout « "Implicit residual norm squared (r^2):\n";
write(std::cout, lend(impl_res));
```

Print solver statistics

```
    std::cout « "CG iteration count:     " « logger->get_num_iterations()
              « std::endl;
    std::cout « "CG execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
}
```

Results

This is the expected output:

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
194.679
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
5.69384e-06
Implicit residual norm squared (r^2):
%%MatrixMarket matrix array real general
1 1
1.27043e-15
CG iteration count:     5
CG execution time [ms]: 0.080041
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
```

```
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    gko::size_type size = A->get_size()[0];
    auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
    for (auto i = 0; i < size; i++) {
        host_x->at(i, 0) = 1.;
    }
    auto x = gko::clone(exec, host_x);
    auto b = gko::clone(exec, host_x);
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto initres = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(initres));
    b->copy_from(host_x.get());
    const RealValueType reduction_factor = 1e-7;
    auto iter_stop =
        gko::stop::Iteration::build().with_max_iters(10000u).on(exec);
    auto tol_stop = gko::stop::ResidualNorm<ValueType>::build()
                        .with_reduction_factor(reduction_factor)
                        .on(exec);
    std::shared_ptr<const gko::log::Convergence<ValueType» logger =
        gko::log::Convergence<ValueType>::create(exec);
    iter_stop->add_logger(logger);
    tol_stop->add_logger(logger);
    auto solver_gen =
        cg::build()
            .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
            .with_preconditioner(bj::build()
                                     .with_max_block_size(16u)
                                     .with_storage_optimization(
                                         gko::precision_reduction::autodetect())
                                     .on(exec))
            .on(exec);
    solver_gen->add_logger(logger);
    auto solver = solver_gen->generate(A);
    exec->synchronize();
    std::chrono::nanoseconds time(0);
    auto tic = std::chrono::steady_clock::now();
    solver->apply(lend(b), lend(x));
    auto toc = std::chrono::steady_clock::now();
    time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    auto impl_res = gko::as<real_vec>(logger->get_implicit_sq_resnorm());
    std::cout « "Initial residual norm sqrt(r^T r):\n";
    write(std::cout, lend(initres));
    std::cout « "Final residual norm sqrt(r^T r):\n";
```

```
    write(std::cout, lend(res));
    std::cout « "Implicit residual norm squared (r^2):\n";
    write(std::cout, lend(impl_res));
    std::cout « "CG iteration count:     " « logger->get_num_iterations()
              « std::endl;
    std::cout « "CG execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
}
```

Chapter 9

The batched-solver program

Using and interfacing with a batched solver..

Using batched solvers

This example shows how to use Ginkgo batched solvers with data coming from an application. The "application" in this case is just a function in the example itself; nevertheless, the steps to be taken are shown.

A ‘batch’ here means a set of small linear systems that can be solved independently, but each system is too small to use an entire computing device. For now, a batch of sparse matrices is required to have a common sparsity pattern for all the matrices.

The commented program

Include files

This is the main ginkgo header file.
```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <random>
#include <string>
```

Type aliases for convenience

Use some shortcuts. In Ginkgo, vectors are seen as a gko::matrix::Dense with one column/one row. The advantage of this concept is that using multiple vectors is a now a natural extension of adding columns/rows are necessary.
```
using value_type = double;
using real_type = gko::remove_complex<value_type>;
using index_type = int;
using size_type = gko::size_type;
using vec_type = gko::matrix::BatchDense<value_type>;
using real_vec_type = gko::matrix::BatchDense<real_type>;
using mtx_type = gko::matrix::BatchCsr<value_type, index_type>;
using solver_type = gko::solver::BatchBicgstab<value_type>;
```

'Application' structures and functions

Structure to simulate application data related to the linear systems to be solved.

We use raw pointers below to demonstrate how to handle the situation when the application only gives us raw pointers. Ideally, one should use Ginkgo's gko::Array class here.
```
struct ApplSysData {
```

Number of small systems in the batch.
```
size_type nsystems;
```

Number of rows in each system.
```
int nrows;
```

Number of non-zeros in each system matrix.
```
int nnz;
```

Row pointers for one matrix
```
const index_type* row_ptrs;
```

Column indices of non-zeros for one matrix
```
const index_type* col_idxs;
```

Nonzero values for all matrices in the batch, concatenated
```
const value_type* all_values;
```

RHS vectors for all systems in the batch, concatenated
```
    const value_type* all_rhs;
};
```

Generates a batch of tridiagonal systems.

Parameters

nrows	Number of rows in each system.
nsystems	Number of systems in the batch.
exec	The device executor to use for the solver. Normally, the application may not deal with Ginkgo executors, nor do we need it to. Here, we use the executor for backend-independent device memory allocation. The application, for example, might assume Hip (for AMD GPUs) and use `hipMalloc` directly.

```
ApplSysData appl_generate_system(const int nrows, const size_type nsystems,
                                 std::shared_ptr<gko::Executor> exec);
```

Deallocate application data.
```
void appl_clean_up(ApplSysData& appl_data, std::shared_ptr<gko::Executor> exec);
int main(int argc, char* argv[])
{
```

Print the ginkgo version information.
```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
    std::exit(-1);
}
```

Where do you want to run your solver ?

The gko::Executor class is one of the cornerstones of Ginkgo. Currently, we have support for an gko::OmpExecutor, which uses OpenMP multi-threading in most of its kernels, a gko::ReferenceExecutor, a single threaded specialization of the OpenMP executor and a gko::CudaExecutor which runs the code on a NVIDIA GPU if available.

Note

With the help of C++, you see that you only ever need to change the executor and all the other functions/ routines within Ginkgo should automatically work and run on the executor with any other changes.

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
const size_type num_systems = argc >= 3 ? std::atoi(argv[2]) : 2;
const int num_rows = 35;  // per system
```

Generate data

The "application" generates the batch of linear systems on the device

```
auto appl_sys = appl_generate_system(num_rows, num_systems, exec);
```

Create batch_dim object to describe the dimensions of the batch matrix.

```
auto batch_mat_size =
    gko::batch_dim<>(num_systems, gko::dim<2>(num_rows, num_rows));
auto batch_vec_size =
    gko::batch_dim<>(num_systems, gko::dim<2>(num_rows, 1));
```

Use of application-allocated memory

We can either work on the existing memory allocated in the application, or we can copy it for the linear solve. Note: it is not possible to use data through a const pointer directly. Because our pointers are not const, we can just 'wrap' the given pointers into Ginkgo Array views so that we can create a Ginkgo matrix out of them. Ginkgo expects the nonzero values for all the small matrices to be allocated contiguously, one matrix after the other.

```
auto vals_view = gko::Array<value_type>::const_view(
    exec, num_systems * appl_sys.nnz, appl_sys.all_values);
auto rowptrs_view = gko::Array<index_type>::const_view(exec, num_rows + 1,
                                                       appl_sys.row_ptrs);
auto colidxs_view = gko::Array<index_type>::const_view(exec, appl_sys.nnz,
                                                       appl_sys.col_idxs);
auto A = gko::share(mtx_type::create_const(
    exec, batch_mat_size, std::move(vals_view), std::move(colidxs_view),
    std::move(rowptrs_view)));
```

RHS and solution vectors

batch_stride object specifies the access stride within the individual matrices (vectors) in the batch. In this case, we specify a stride of 1 as the common value for all the matrices.

```
auto batch_vec_stride = gko::batch_stride(num_systems, 1);
```

Create RHS, again reusing application allocation

```
auto b_view = gko::Array<value_type>::const_view(
    exec, num_systems * num_rows, appl_sys.all_rhs);
auto b = vec_type::create_const(exec, batch_vec_size, std::move(b_view),
```

```
                                 batch_vec_stride);
```

Create initial guess as 0 and copy to device

```
auto x = vec_type::create(exec);
auto host_x =
    vec_type::create(exec->get_master(), batch_vec_size, batch_vec_stride);
for (size_type isys = 0; isys < num_systems; isys++) {
    for (int irow = 0; irow < num_rows; irow++) {
        host_x->at(isys, irow, 0) = gko::zero<value_type>();
    }
}
x->copy_from(host_x.get());
```

Create the batch solver factory

```
const real_type reduction_factor{1e-6};
```

Create a batched solver factory with relevant parameters.

```
auto solver_gen =
    solver_type::build()
        .with_max_iterations(500)
        .with_residual_tol(reduction_factor)
        .with_tolerance_type(gko::stop::batch::ToleranceType::relative)
        .with_preconditioner(gko::preconditioner::batch::type::jacobi)
        .on(exec);
```

Batch logger

Create a logger to obtain the iteration counts and "implicit" residual norms for every system after the solve.

```
std::shared_ptr<const gko::log::BatchConvergence<value_type>> logger =
    gko::log::BatchConvergence<value_type>::create(exec);
```

Generate and solve

Generate the batch solver from the batch matrix

```
auto solver = solver_gen->generate(A);
```

add the logger to the solver

```
solver->add_logger(logger);
```

Solve the batch system

```
solver->apply(lend(b), lend(x));
```

This is not necessary, but one might want to remove the logger before the next solve using the same solver object.

```
solver->remove_logger(logger.get());
```

Check result

Compute norm of RHS on the device and automatically copy to host

```
auto b_norm = gko::batch_initialize<real_vec_type>(num_systems, {0.0},
                                                   exec->get_master());
b->compute_norm2(lend(b_norm));
```

we need constants on the device

```
auto one = gko::batch_initialize<vec_type>(num_systems, {1.0}, exec);
auto neg_one = gko::batch_initialize<vec_type>(num_systems, {-1.0}, exec);
```

allocate and compute the residual

```
auto res = vec_type::create(exec, batch_vec_size);
res->copy_from(lend(b));
A->apply(lend(one), lend(x), lend(neg_one), lend(res));
```

allocate and compute residual norm

```
auto res_norm = gko::batch_initialize<real_vec_type>(num_systems, {0.0},
                                                     exec->get_master());
res->compute_norm2(lend(res_norm));
std::cout « "Residual norm sqrt(r^T r):\n";
```

"unbatch" converts a batch object into a vector of objects of the corresponding single type, eg. BatchDense --> vector<Dense>.

```
auto unb_res = res_norm->unbatch();
auto unb_bnorm = b_norm->unbatch();
for (size_type i = 0; i < num_systems; ++i) {
    std::cout « " System no. " « i
              « ": residual norm = " « unb_res[i]->at(0, 0)
              « ", internal residual norm = "
              « logger->get_residual_norm()->at(i, 0, 0)
              « ", iterations = "
              « logger->get_num_iterations().get_const_data()[i]
              « std::endl;
    const real_type relresnorm =
        unb_res[i]->at(0, 0) / unb_bnorm[i]->at(0, 0);
    if (!(relresnorm <= reduction_factor)) {
        std::cout « "System " « i « " converged only to " « relresnorm
                  « " relative residual." « std::endl;
    }
}
```

Ginkgo objects are cleaned up automatically; but the "application" still needs to clean up its data in this case.

```
    appl_clean_up(appl_sys, exec);
    return 0;
}
ApplSysData appl_generate_system(const int nrows, const size_type nsystems,
                                 std::shared_ptr<gko::Executor> exec)
{
    const int nnz = nrows * 3 - 2;
    std::ranlux48 rgen(15);
    std::normal_distribution<real_type> distb(0.5, 0.1);
    std::vector<real_type> spacings(nsystems * nrows);
    std::generate(spacings.begin(), spacings.end(),
                  [&]() { return distb(rgen); });
    std::vector<value_type> allvalues(nnz * nsystems);
    for (size_type isys = 0; isys < nsystems; isys++) {
        allvalues[isys * nnz] = 2.0 / spacings[isys * nrows];
        allvalues[isys * nnz + 1] = -1.0;
        for (int irow = 0; irow < nrows - 2; irow++) {
            allvalues[isys * nnz + 2 + irow * 3] = -1.0;
            allvalues[isys * nnz + 2 + irow * 3 + 1] =
                2.0 / spacings[isys * nrows + irow + 1];
            allvalues[isys * nnz + 2 + irow * 3 + 2] = -1.0;
        }
        allvalues[isys * nnz + 2 + (nrows - 2) * 3] = -1.0;
        allvalues[isys * nnz + 2 + (nrows - 2) * 3 + 1] =
            2.0 / spacings[(isys + 1) * nrows - 1];
        assert(isys * nnz + 2 + (nrows - 2) * 3 + 2 == (isys + 1) * nnz);
    }
    std::vector<index_type> rowptrs(nrows + 1);
    rowptrs[0] = 0;
    rowptrs[1] = 2;
    for (int i = 2; i < nrows; i++) {
        rowptrs[i] = rowptrs[i - 1] + 3;
    }
    rowptrs[nrows] = rowptrs[nrows - 1] + 2;
    assert(rowptrs[nrows] == nnz);
    std::vector<index_type> colidxs(nnz);
    colidxs[0] = 0;
    colidxs[1] = 1;
    const int nnz_per_row = 3;
    for (int irow = 1; irow < nrows - 1; irow++) {
        colidxs[2 + (irow - 1) * nnz_per_row] = irow - 1;
        colidxs[2 + (irow - 1) * nnz_per_row + 1] = irow;
        colidxs[2 + (irow - 1) * nnz_per_row + 2] = irow + 1;
    }
    colidxs[2 + (nrows - 2) * nnz_per_row] = nrows - 2;
    colidxs[2 + (nrows - 2) * nnz_per_row + 1] = nrows - 1;
    assert(2 + (nrows - 2) * nnz_per_row + 1 == nnz - 1);
    std::vector<value_type> allb(nrows * nsystems);
    for (size_type isys = 0; isys < nsystems; isys++) {
        const value_type bval = distb(rgen);
        std::fill(allb.begin() + isys * nrows,
                  allb.begin() + (isys + 1) * nrows, bval);
    }
    index_type* const row_ptrs = exec->alloc<index_type>(nrows + 1);
    exec->copy_from(exec->get_master().get(), static_cast<size_type>(nrows + 1),
                    rowptrs.data(), row_ptrs);
    index_type* const col_idxs = exec->alloc<index_type>(nnz);
    exec->copy_from(exec->get_master().get(), static_cast<size_type>(nnz),
                    colidxs.data(), col_idxs);
```

```
    value_type* const all_values = exec->alloc<value_type>(nsystems * nnz);
    exec->copy_from(exec->get_master().get(), nsystems * nnz, allvalues.data(),
                    all_values);
    value_type* const all_b = exec->alloc<value_type>(nsystems * nrows);
    exec->copy_from(exec->get_master().get(), nsystems * nrows, allb.data(),
                    all_b);
    return {nsystems, nrows, nnz, row_ptrs, col_idxs, all_values, all_b};
}
void appl_clean_up(ApplSysData& appl_data, std::shared_ptr<gko::Executor> exec)
{
```

In general, the application would control non-const pointers; the const casts below would not be needed.

```
    exec->free(const_cast<index_type*>(appl_data.row_ptrs));
    exec->free(const_cast<index_type*>(appl_data.col_idxs));
    exec->free(const_cast<value_type*>(appl_data.all_values));
    exec->free(const_cast<value_type*>(appl_data.all_rhs));
}
```

Results

The following is the expected result on the reference executor:

```
Residual norm sqrt(r^T r):
 System no. 0: residual norm = 2.16283e-06, internal residual norm = 2.16283e-06, iterations = 6
 System no. 1: residual norm = 1.07502e-06, internal residual norm = 1.07502e-06, iterations = 6
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <random>
#include <string>
using value_type = double;
using real_type = gko::remove_complex<value_type>;
using index_type = int;
using size_type = gko::size_type;
using vec_type = gko::matrix::BatchDense<value_type>;
using real_vec_type = gko::matrix::BatchDense<real_type>;
using mtx_type = gko::matrix::BatchCsr<value_type, index_type>;
using solver_type = gko::solver::BatchBicgstab<value_type>;
```

```
struct ApplSysData {
    size_type nsystems;
    int nrows;
    int nnz;
    const index_type* row_ptrs;
    const index_type* col_idxs;
    const value_type* all_values;
    const value_type* all_rhs;
};
ApplSysData appl_generate_system(const int nrows, const size_type nsystems,
                                 std::shared_ptr<gko::Executor> exec);
void appl_clean_up(ApplSysData& appl_data, std::shared_ptr<gko::Executor> exec);
int main(int argc, char* argv[])
{
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    const size_type num_systems = argc >= 3 ? std::atoi(argv[2]) : 2;
    const int num_rows = 35;  // per system
    auto appl_sys = appl_generate_system(num_rows, num_systems, exec);
    auto batch_mat_size =
        gko::batch_dim<>(num_systems, gko::dim<2>(num_rows, num_rows));
    auto batch_vec_size =
        gko::batch_dim<>(num_systems, gko::dim<2>(num_rows, 1));
    auto vals_view = gko::Array<value_type>::const_view(
        exec, num_systems * appl_sys.nnz, appl_sys.all_values);
    auto rowptrs_view = gko::Array<index_type>::const_view(exec, num_rows + 1,
                                                           appl_sys.row_ptrs);
    auto colidxs_view = gko::Array<index_type>::const_view(exec, appl_sys.nnz,
                                                           appl_sys.col_idxs);
    auto A = gko::share(mtx_type::create_const(
        exec, batch_mat_size, std::move(vals_view), std::move(colidxs_view),
        std::move(rowptrs_view)));
    auto batch_vec_stride = gko::batch_stride(num_systems, 1);
    auto b_view = gko::Array<value_type>::const_view(
        exec, num_systems * num_rows, appl_sys.all_rhs);
    auto b = vec_type::create_const(exec, batch_vec_size, std::move(b_view),
                                    batch_vec_stride);
    auto x = vec_type::create(exec);
    auto host_x =
        vec_type::create(exec->get_master(), batch_vec_size, batch_vec_stride);
    for (size_type isys = 0; isys < num_systems; isys++) {
        for (int irow = 0; irow < num_rows; irow++) {
            host_x->at(isys, irow, 0) = gko::zero<value_type>();
        }
    }
    x->copy_from(host_x.get());
    const real_type reduction_factor{1e-6};
    auto solver_gen =
        solver_type::build()
            .with_max_iterations(500)
            .with_residual_tol(reduction_factor)
            .with_tolerance_type(gko::stop::batch::ToleranceType::relative)
            .with_preconditioner(gko::preconditioner::batch::type::jacobi)
            .on(exec);
    std::shared_ptr<const gko::log::BatchConvergence<value_type» logger =
        gko::log::BatchConvergence<value_type>::create(exec);
    auto solver = solver_gen->generate(A);
    solver->add_logger(logger);
    solver->apply(lend(b), lend(x));
    solver->remove_logger(logger.get());
    auto b_norm = gko::batch_initialize<real_vec_type>(num_systems, {0.0},
                                                        exec->get_master());
    b->compute_norm2(lend(b_norm));
```

```
    auto one = gko::batch_initialize<vec_type>(num_systems, {1.0}, exec);
    auto neg_one = gko::batch_initialize<vec_type>(num_systems, {-1.0}, exec);
    auto res = vec_type::create(exec, batch_vec_size);
    res->copy_from(lend(b));
    A->apply(lend(one), lend(x), lend(neg_one), lend(res));
    auto res_norm = gko::batch_initialize<real_vec_type>(num_systems, {0.0},
                                                         exec->get_master());
    res->compute_norm2(lend(res_norm));
    std::cout « "Residual norm sqrt(r^T r):\n";
    auto unb_res = res_norm->unbatch();
    auto unb_bnorm = b_norm->unbatch();
    for (size_type i = 0; i < num_systems; ++i) {
        std::cout « " System no. " « i
                  « ": residual norm = " « unb_res[i]->at(0, 0)
                  « ", internal residual norm = "
                  « logger->get_residual_norm()->at(i, 0, 0)
                  « ", iterations = "
                  « logger->get_num_iterations().get_const_data()[i]
                  « std::endl;
        const real_type relresnorm =
            unb_res[i]->at(0, 0) / unb_bnorm[i]->at(0, 0);
        if (!(relresnorm <= reduction_factor)) {
            std::cout « "System " « i « " converged only to " « relresnorm
                      « " relative residual." « std::endl;
        }
    }
    appl_clean_up(appl_sys, exec);
    return 0;
}
ApplSysData appl_generate_system(const int nrows, const size_type nsystems,
                                 std::shared_ptr<gko::Executor> exec)
{
    const int nnz = nrows * 3 - 2;
    std::ranlux48 rgen(15);
    std::normal_distribution<real_type> distb(0.5, 0.1);
    std::vector<real_type> spacings(nsystems * nrows);
    std::generate(spacings.begin(), spacings.end(),
                  [&]() { return distb(rgen); });
    std::vector<value_type> allvalues(nnz * nsystems);
    for (size_type isys = 0; isys < nsystems; isys++) {
        allvalues[isys * nnz] = 2.0 / spacings[isys * nrows];
        allvalues[isys * nnz + 1] = -1.0;
        for (int irow = 0; irow < nrows - 2; irow++) {
            allvalues[isys * nnz + 2 + irow * 3] = -1.0;
            allvalues[isys * nnz + 2 + irow * 3 + 1] =
                2.0 / spacings[isys * nrows + irow + 1];
            allvalues[isys * nnz + 2 + irow * 3 + 2] = -1.0;
        }
        allvalues[isys * nnz + 2 + (nrows - 2) * 3] = -1.0;
        allvalues[isys * nnz + 2 + (nrows - 2) * 3 + 1] =
            2.0 / spacings[(isys + 1) * nrows - 1];
        assert(isys * nnz + 2 + (nrows - 2) * 3 + 2 == (isys + 1) * nnz);
    }
    std::vector<index_type> rowptrs(nrows + 1);
    rowptrs[0] = 0;
    rowptrs[1] = 2;
    for (int i = 2; i < nrows; i++) {
        rowptrs[i] = rowptrs[i - 1] + 3;
    }
    rowptrs[nrows] = rowptrs[nrows - 1] + 2;
    assert(rowptrs[nrows] == nnz);
    std::vector<index_type> colidxs(nnz);
    colidxs[0] = 0;
    colidxs[1] = 1;
    const int nnz_per_row = 3;
    for (int irow = 1; irow < nrows - 1; irow++) {
        colidxs[2 + (irow - 1) * nnz_per_row] = irow - 1;
        colidxs[2 + (irow - 1) * nnz_per_row + 1] = irow;
        colidxs[2 + (irow - 1) * nnz_per_row + 2] = irow + 1;
    }
    colidxs[2 + (nrows - 2) * nnz_per_row] = nrows - 2;
    colidxs[2 + (nrows - 2) * nnz_per_row + 1] = nrows - 1;
    assert(2 + (nrows - 2) * nnz_per_row + 1 == nnz - 1);
    std::vector<value_type> allb(nrows * nsystems);
    for (size_type isys = 0; isys < nsystems; isys++) {
        const value_type bval = distb(rgen);
        std::fill(allb.begin() + isys * nrows,
                  allb.begin() + (isys + 1) * nrows, bval);
    }
    index_type* const row_ptrs = exec->alloc<index_type>(nrows + 1);
    exec->copy_from(exec->get_master().get(), static_cast<size_type>(nrows + 1),
                    rowptrs.data(), row_ptrs);
    index_type* const col_idxs = exec->alloc<index_type>(nnz);
    exec->copy_from(exec->get_master().get(), static_cast<size_type>(nnz),
                    colidxs.data(), col_idxs);
    value_type* const all_values = exec->alloc<value_type>(nsystems * nnz);
    exec->copy_from(exec->get_master().get(), nsystems * nnz, allvalues.data(),
```

```
                    all_values);
    value_type* const all_b = exec->alloc<value_type>(nsystems * nrows);
    exec->copy_from(exec->get_master().get(), nsystems * nrows, allb.data(),
                    all_b);
    return {nsystems, nrows, nnz, row_ptrs, col_idxs, all_values, all_b};
}
void appl_clean_up(ApplSysData& appl_data, std::shared_ptr<gko::Executor> exec)
{
    exec->free(const_cast<index_type*>(appl_data.row_ptrs));
    exec->free(const_cast<index_type*>(appl_data.col_idxs));
    exec->free(const_cast<value_type*>(appl_data.all_values));
    exec->free(const_cast<value_type*>(appl_data.all_rhs));
}
```

Chapter 10

The batched-solver-from-files program

Using a batched solver with matrices stored in files..

Using batched solvers with data read from files

This example shows how to use Ginkgo batched solvers with data coming from files. A specific directory structure is assumed; it should be general enough for the use cases that Ginkgo batched functionality is intended for.

The sizes and sparsity pattern of all the matrices in the batch are assumed to be the same.

The commented program

Include files

This is the main ginkgo header file.
```
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
```

Type aliases for convenience

Use some shortcuts. In Ginkgo, vectors are seen as a gko::matrix::Dense with one column/one row. The advantage of this concept is that using multiple vectors is a now a natural extension of adding columns/rows are necessary.
```
using value_type = double;
using real_type = gko::remove_complex<value_type>;
using index_type = int;
using size_type = gko::size_type;
using vec_type = gko::matrix::BatchDense<value_type>;
using real_vec_type = gko::matrix::BatchDense<real_type>;
using mtx_type = gko::matrix::BatchCsr<value_type, index_type>;
```

using mtx_type = gko::matrix::BatchEll<value_type, index_type>;
```
using solver_type = gko::solver::BatchBicgstab<value_type>;
int main(int argc, char* argv[])
{
```

Print the ginkgo version information.
```
std::cout << gko::version_info::get() << std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr << "Usage: " << argv[0] << " [executor] " << std::endl;
    std::exit(-1);
}
```

Where do you want to run your solver ?

The gko::Executor class is one of the cornerstones of Ginkgo. Currently, we have support for an gko::OmpExecutor, which uses OpenMP multi-threading in most of its kernels, a gko::ReferenceExecutor, a single threaded specialization of the OpenMP executor and a gko::CudaExecutor which runs the code on a NVIDIA GPU if available.

Note

With the help of C++, you see that you only ever need to change the executor and all the other functions/routines within Ginkgo should automatically work and run on the executor with any other changes.

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read batch from files

Name of the problem, which is also the directory under which all matrices are stored.

```
const std::string problem_descr_str = argc >= 3 ? argv[2] : "pores_1";
```

Number of linear systems to read from files.

```
const size_type num_systems = argc >= 4 ? std::atoi(argv[3]) : 2;
```

Number of times to duplicate whatever systems are read from files.

```
const size_type num_duplications = argc >= 5 ? std::atoi(argv[4]) : 2;
```

Whether to enable diagonal scaling of the matrices before solving. The scaling vectors need to be available in a 'S.mtx' file.

```
const std::string batch_scaling = argc >= 6 ? argv[5] : "none";
auto data = std::vector<gko::matrix_data<value_type»(num_systems);
std::vector<gko::matrix_data<value_type» bdata(num_systems);
auto scale_data = std::vector<gko::matrix_data<value_type»(num_systems);
for (size_type i = 0; i < data.size(); ++i) {
    const std::string mat_str = "A.mtx";
    const std::string fbase =
        "data/" + problem_descr_str + "/" + std::to_string(i) + "/";
    std::string fname = fbase + mat_str;
    std::ifstream mtx_fd(fname);
    std::string b_str = "b.mtx";
    data[i] = gko::read_raw<value_type>(mtx_fd);
    std::string bfname =
        "data/" + problem_descr_str + "/" + std::to_string(i) + "/" + b_str;
    std::ifstream b_fd(bfname);
    bdata[i] = gko::read_raw<value_type>(b_fd);
```

If necessary, 'scaling vectors' can be provided to diagonal-scale a system from the left and the right. For this example, no scaling vectors are provided.

```
    if (batch_scaling == "explicit") {
        std::string scale_fname = fbase + "S.mtx";
        std::ifstream scale_fd(scale_fname);
        scale_data[i] = gko::read_raw<value_type>(scale_fd);
```

```
    }
}
auto single_batch = mtx_type::create(exec);
single_batch->read(data);
```

We can duplicate the batch a few times if we wish.

```
std::shared_ptr<mtx_type> A =
    mtx_type::create(exec, num_duplications, single_batch.get());
```

Create RHS

```
auto temp_b = vec_type::create(exec);
temp_b->read(bdata);
auto b = vec_type::create(exec, num_duplications, temp_b.get());
```

Create initial guess as 0 and copy to device

```
const size_type num_total_systems = num_systems * num_duplications;
auto x = vec_type::create(exec);
auto host_x = vec_type::create(exec->get_master(), b->get_size());
```

The number of rows in each system is taken as the 0th dimension of the size of the 0th system in the batch.

```
const int num_rows = static_cast<int>(b->get_size().at(0)[0]);
for (size_type isys = 0; isys < num_total_systems; isys++) {
    for (int irow = 0; irow < num_rows; irow++) {
        host_x->at(isys, irow, 0) = gko::zero<value_type>();
    }
}
x->copy_from(host_x.get());
```

Create the batch solver factory

```
const real_type reduction_factor{1e-6};
```

Create a batched solver factory with relevant parameters.

```
auto solver_gen =
    solver_type::build()
        .with_max_iterations(500)
        .with_residual_tol(reduction_factor)
        .with_tolerance_type(gko::stop::batch::ToleranceType::relative)
```

.with_preconditioner(gko::preconditioner::batch::type::jacobi)

```
.on(exec);
```

Batch logger

Create a logger to obtain the iteration counts and "implicit" residual norms for every system after the solve.

```
std::shared_ptr<const gko::log::BatchConvergence<value_type>> logger =
    gko::log::BatchConvergence<value_type>::create(exec);
```

Generate and solve

Generate the batch solver from the batch matrix

```
auto solver = solver_gen->generate(A);
```

add the logger to the solver

```
solver->add_logger(logger);
std::chrono::steady_clock::time_point t1 = std::chrono::steady_clock::now();
```

Solve the batch system

```
solver->apply(lend(b), lend(x));
std::chrono::steady_clock::time_point t2 = std::chrono::steady_clock::now();
```

This is not necessary, but one might want to remove the logger before the next solve using the same solver object.

```
solver->remove_logger(logger.get());
```

Check result

Compute norms of RHS and final residual to check the result

```
auto b_norm = gko::batch_initialize<real_vec_type>(num_total_systems, {0.0},
                                                   exec->get_master());
b->compute_norm2(lend(b_norm));
```

we need constants on the device

```
auto one = gko::batch_initialize<vec_type>(num_total_systems, {1.0}, exec);
auto neg_one =
    gko::batch_initialize<vec_type>(num_total_systems, {-1.0}, exec);
```

allocate and compute the residual

```
auto res = vec_type::create(exec);
res->copy_from(lend(b));
A->apply(lend(one), lend(x), lend(neg_one), lend(res));
```

allocate and compute residual norm on the device

```
auto res_norm = gko::batch_initialize<real_vec_type>(
    num_total_systems, {0.0}, exec->get_master());
res->compute_norm2(lend(res_norm));
std::cout « "Residual norm sqrt(r^T r):\n";
```

"unbatch" converts a batch object into a vector of objects of the corresponding single type, eg. BatchDense --> vector<Dense>.

```
    auto unb_res = res_norm->unbatch();
    auto unb_bnorm = b_norm->unbatch();
    for (size_type i = 0; i < num_total_systems; ++i) {
        std::cout « " System no. " « i
                  « ": residual norm = " « unb_res[i]->at(0, 0)
                  « ", internal residual norm = "
                  « logger->get_residual_norm()->at(i, 0, 0)
                  « ", iterations = "
                  « logger->get_num_iterations().get_const_data()[i]
                  « std::endl;
        const real_type relresnorm =
            unb_res[i]->at(0, 0) / unb_bnorm[i]->at(0, 0);
        if (!(relresnorm <= reduction_factor)) {
            std::cout « "System " « i « " converged only to " « relresnorm
                      « " relative residual." « std::endl;
        }
    }
    auto time_span =
        std::chrono::duration_cast<std::chrono::duration<double»(t2 - t1);
    std::cout « "Entire solve took " « time_span.count() « " seconds."
              « std::endl;
    return 0;
}
```

Results

The following is the expected result on the reference executor:

```
Residual norm sqrt(r^T r):
 System no. 0: residual norm = 2.69282e-06, internal residual norm = 2.69281e-06, iterations = 94
 System no. 1: residual norm = 2.35024e-07, internal residual norm = 2.3492e-07, iterations = 91
 System no. 2: residual norm = 2.69282e-06, internal residual norm = 2.69281e-06, iterations = 94
 System no. 3: residual norm = 2.35024e-07, internal residual norm = 2.3492e-07, iterations = 91
Entire solve took 0.00708691 seconds.
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
```

```
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
using value_type = double;
using real_type = gko::remove_complex<value_type>;
using index_type = int;
using size_type = gko::size_type;
using vec_type = gko::matrix::BatchDense<value_type>;
using real_vec_type = gko::matrix::BatchDense<real_type>;
using mtx_type = gko::matrix::BatchCsr<value_type, index_type>;
using solver_type = gko::solver::BatchBicgstab<value_type>;
int main(int argc, char* argv[])
{
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    const std::string problem_descr_str = argc >= 3 ? argv[2] : "pores_1";
    const size_type num_systems = argc >= 4 ? std::atoi(argv[3]) : 2;
    const size_type num_duplications = argc >= 5 ? std::atoi(argv[4]) : 2;
    const std::string batch_scaling = argc >= 6 ? argv[5] : "none";
    auto data = std::vector<gko::matrix_data<value_type»(num_systems);
    std::vector<gko::matrix_data<value_type» bdata(num_systems);
    auto scale_data = std::vector<gko::matrix_data<value_type»(num_systems);
    for (size_type i = 0; i < data.size(); ++i) {
        const std::string mat_str = "A.mtx";
        const std::string fbase =
            "data/" + problem_descr_str + "/" + std::to_string(i) + "/";
        std::string fname = fbase + mat_str;
        std::ifstream mtx_fd(fname);
        std::string b_str = "b.mtx";
        data[i] = gko::read_raw<value_type>(mtx_fd);
        std::string bfname =
            "data/" + problem_descr_str + "/" + std::to_string(i) + "/" + b_str;
        std::ifstream b_fd(bfname);
        bdata[i] = gko::read_raw<value_type>(b_fd);
        if (batch_scaling == "explicit") {
```

```
            std::string scale_fname = fbase + "S.mtx";
            std::ifstream scale_fd(scale_fname);
            scale_data[i] = gko::read_raw<value_type>(scale_fd);
        }
    }
    auto single_batch = mtx_type::create(exec);
    single_batch->read(data);
    std::shared_ptr<mtx_type> A =
        mtx_type::create(exec, num_duplications, single_batch.get());
    auto temp_b = vec_type::create(exec);
    temp_b->read(bdata);
    auto b = vec_type::create(exec, num_duplications, temp_b.get());
    const size_type num_total_systems = num_systems * num_duplications;
    auto x = vec_type::create(exec);
    auto host_x = vec_type::create(exec->get_master(), b->get_size());
    const int num_rows = static_cast<int>(b->get_size().at(0)[0]);
    for (size_type isys = 0; isys < num_total_systems; isys++) {
        for (int irow = 0; irow < num_rows; irow++) {
            host_x->at(isys, irow, 0) = gko::zero<value_type>();
        }
    }
    x->copy_from(host_x.get());
    const real_type reduction_factor{1e-6};
    auto solver_gen =
        solver_type::build()
            .with_max_iterations(500)
            .with_residual_tol(reduction_factor)
            .with_tolerance_type(gko::stop::batch::ToleranceType::relative)
            .on(exec);
    std::shared_ptr<const gko::log::BatchConvergence<value_type>> logger =
        gko::log::BatchConvergence<value_type>::create(exec);
    auto solver = solver_gen->generate(A);
    solver->add_logger(logger);
    std::chrono::steady_clock::time_point t1 = std::chrono::steady_clock::now();
    solver->apply(lend(b), lend(x));
    std::chrono::steady_clock::time_point t2 = std::chrono::steady_clock::now();
    solver->remove_logger(logger.get());
    auto b_norm = gko::batch_initialize<real_vec_type>(num_total_systems, {0.0},
                                                        exec->get_master());
    b->compute_norm2(lend(b_norm));
    auto one = gko::batch_initialize<vec_type>(num_total_systems, {1.0}, exec);
    auto neg_one =
        gko::batch_initialize<vec_type>(num_total_systems, {-1.0}, exec);
    auto res = vec_type::create(exec);
    res->copy_from(lend(b));
    A->apply(lend(one), lend(x), lend(neg_one), lend(res));
    auto res_norm = gko::batch_initialize<real_vec_type>(
        num_total_systems, {0.0}, exec->get_master());
    res->compute_norm2(lend(res_norm));
    std::cout << "Residual norm sqrt(r^T r):\n";
    auto unb_res = res_norm->unbatch();
    auto unb_bnorm = b_norm->unbatch();
    for (size_type i = 0; i < num_total_systems; ++i) {
        std::cout << " System no. " << i
                  << ": residual norm = " << unb_res[i]->at(0, 0)
                  << ", internal residual norm = "
                  << logger->get_residual_norm()->at(i, 0, 0)
                  << ", iterations = "
                  << logger->get_num_iterations().get_const_data()[i]
                  << std::endl;
        const real_type relresnorm =
            unb_res[i]->at(0, 0) / unb_bnorm[i]->at(0, 0);
        if (!(relresnorm <= reduction_factor)) {
            std::cout << "System " << i << " converged only to " << relresnorm
                      << " relative residual." << std::endl;
        }
    }
    auto time_span =
        std::chrono::duration_cast<std::chrono::duration<double>>(t2 - t1);
    std::cout << "Entire solve took " << time_span.count() << " seconds."
              << std::endl;
    return 0;
}
```

Chapter 11

The cb-gmres program

The CB-GMRES solver example..

Introduction

About the example

This example showcases the usage of the Ginkgo solver CB-GMRES (Compressed Basis GMRES). A small system is solved with two un-preconditioned CB-GMRES solvers:

1. without compressing the krylov basis; it uses double precision for both the matrix and the krylov basis, and
2. with a compression of the krylov basis; it uses double precision for the matrix and all arithmetic operations, while using single precision for the storage of the krylov basis

Both solves are timed and the residual norm of each solution is computed to show that both solutions are correct.

The commented program

This is the main ginkgo header file.
```
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <cmath>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
```

Helper function which measures the time of `solver->apply(b, x)` in seconds To get an accurate result, the solve is repeated multiple times (while ensuring the initial guess is always the same). The result of the solve will be written to x.
```
double measure_solve_time_in_s(const gko::Executor* exec, gko::LinOp* solver,
                               const gko::LinOp* b, gko::LinOp* x)
{
    constexpr int repeats{5};
    double duration{0};
```

Make a copy of x, so we can re-use the same initial guess multiple times
```
auto x_copy = clone(x);
for (int i = 0; i < repeats; ++i) {
```

No need to copy it in the first iteration
```
if (i != 0) {
    x_copy->copy_from(x);
}
```

Make sure all previous executor operations have finished before starting the time
```
exec->synchronize();
auto tic = std::chrono::steady_clock::now();
solver->apply(b, lend(x_copy));
```

Make sure all computations are done before stopping the time
```
    exec->synchronize();
    auto tac = std::chrono::steady_clock::now();
    duration += std::chrono::duration<double>(tac - tic).count();
}
```

Copy the solution back to x, so the caller has the result
```
    x->copy_from(lend(x_copy));
    return duration / static_cast<double>(repeats);
}
int main(int argc, char* argv[])
{
```

Use some shortcuts. In Ginkgo, vectors are seen as a gko::matrix::Dense with one column/one row. The advantage of this concept is that using multiple vectors is a now a natural extension of adding columns/rows are necessary.
```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
```

The gko::matrix::Csr class is used here, but any other matrix class such as gko::matrix::Coo, gko::matrix::Hybrid, gko::matrix::Ell or gko::matrix::Sellp could also be used.
```
using mtx = gko::matrix::Csr<ValueType, IndexType>;
```

The gko::solver::CbGmres is used here, but any other solver class can also be used.
```
using cb_gmres = gko::solver::CbGmres<ValueType>;
```

Print the ginkgo version information.
```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
    std::exit(-1);
}
```

Map which generates the appropriate executor
```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation
```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Note: this matrix is copied from "SOURCE_DIR/matrices" instead of from the local directory. For details, see "examples/cb-gmres/CMakeLists.txt"
```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create a uniform right-hand side with a norm2 of 1 on the host (norm2(b) == 1), followed by copying it to the actual executor (to make sure it also works for GPUs)

```
const auto A_size = A->get_size();
auto b_host = vec::create(exec->get_master(), gko::dim<2>{A_size[0], 1});
for (gko::size_type i = 0; i < A_size[0]; ++i) {
    b_host->at(i, 0) =
        ValueType{1} / std::sqrt(static_cast<ValueType>(A_size[0]));
}
auto b_norm = gko::initialize<real_vec>({0.0}, exec);
b_host->compute_norm2(lend(b_norm));
auto b = clone(exec, lend(b_host));
```

As an initial guess, use the right-hand side

```
auto x_keep = clone(lend(b));
auto x_reduce = clone(x_keep);
const RealValueType reduction_factor{1e-6};
```

Generate two solver factories: `_keep` uses the same precision for the krylov basis as the matrix, and `_reduce` uses one precision below it. If `ValueType` is double, then `_reduce` uses float as the krylov basis storage type

```
auto solver_gen_keep =
    cb_gmres::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1000u).on(exec),
            gko::stop::RelativeResidualNorm<ValueType>::build()
                .with_tolerance(reduction_factor)
                .on(exec))
        .with_krylov_dim(100u)
        .with_storage_precision(
            gko::solver::cb_gmres::storage_precision::keep)
        .on(exec);
auto solver_gen_reduce =
    cb_gmres::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1000u).on(exec),
            gko::stop::RelativeResidualNorm<ValueType>::build()
                .with_tolerance(reduction_factor)
                .on(exec))
        .with_krylov_dim(100u)
        .with_storage_precision(
            gko::solver::cb_gmres::storage_precision::reduce1)
        .on(exec);
```

Generate the actual solver from the factory and the matrix.

```
auto solver_keep = solver_gen_keep->generate(A);
auto solver_reduce = solver_gen_reduce->generate(A);
```

Solve both system and measure the time for each.

```
auto time_keep = measure_solve_time_in_s(lend(exec), lend(solver_keep),
                                         lend(b), lend(x_keep));
auto time_reduce = measure_solve_time_in_s(lend(exec), lend(solver_reduce),
                                           lend(b), lend(x_reduce));
```

Make sure the output is in scientific notation for easier comparison

```
std::cout « std::scientific;
```

Note: The time might not be significantly different since the matrix is quite small

```
std::cout « "Solve time without compression: " « time_keep « " s\n"
          « "Solve time with compression:    " « time_reduce « " s\n";
```

To measure if your solution has actually converged, the error of the solution is measured. one, neg_one are objects that represent the numbers which allow for a uniform interface when computing on any device. To compute the residual, the (advanced) apply method is used.

```
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto res_norm_keep = gko::initialize<real_vec>({0.0}, exec);
auto res_norm_reduce = gko::initialize<real_vec>({0.0}, exec);
auto tmp = gko::clone(gko::lend(b));
```

tmp = Ax - tmp

```
    A->apply(lend(one), lend(x_keep), lend(neg_one), lend(tmp));
    tmp->compute_norm2(lend(res_norm_keep));
    std::cout « "\nResidual norm without compression:\n";
    write(std::cout, lend(res_norm_keep));
    tmp->copy_from(lend(b));
    A->apply(lend(one), lend(x_reduce), lend(neg_one), lend(tmp));
    tmp->compute_norm2(lend(res_norm_reduce));
    std::cout « "\nResidual norm with compression:\n";
    write(std::cout, lend(res_norm_reduce));
}
```

Results

The following is the expected result:

```
Solve time without compression: 1.842690e-04 s
Solve time with compression:    1.589936e-04 s
Residual norm without compression:
%%MatrixMarket matrix array real general
1 1
2.430544e-07
Residual norm with compression:
%%MatrixMarket matrix array real general
1 1
3.437257e-07
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <cmath>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
double measure_solve_time_in_s(const gko::Executor* exec, gko::LinOp* solver,
                               const gko::LinOp* b, gko::LinOp* x)
{
    constexpr int repeats{5};
    double duration{0};
    auto x_copy = clone(x);
    for (int i = 0; i < repeats; ++i) {
        if (i != 0) {
            x_copy->copy_from(x);
        }
        exec->synchronize();
        auto tic = std::chrono::steady_clock::now();
        solver->apply(b, lend(x_copy));
        exec->synchronize();
        auto tac = std::chrono::steady_clock::now();
        duration += std::chrono::duration<double>(tac - tic).count();
    }
    x->copy_from(lend(x_copy));
    return duration / static_cast<double>(repeats);
}
int main(int argc, char* argv[])
{
    using ValueType = double;
```

```
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cb_gmres = gko::solver::CbGmres<ValueType>;
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
const auto exec = exec_map.at(executor_string)();  // throws if not valid
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
const auto A_size = A->get_size();
auto b_host = vec::create(exec->get_master(), gko::dim<2>{A_size[0], 1});
for (gko::size_type i = 0; i < A_size[0]; ++i) {
    b_host->at(i, 0) =
        ValueType{1} / std::sqrt(static_cast<ValueType>(A_size[0]));
}
auto b_norm = gko::initialize<real_vec>({0.0}, exec);
b_host->compute_norm2(lend(b_norm));
auto b = clone(exec, lend(b_host));
auto x_keep = clone(lend(b));
auto x_reduce = clone(x_keep);
const RealValueType reduction_factor{1e-6};
auto solver_gen_keep =
    cb_gmres::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1000u).on(exec),
            gko::stop::RelativeResidualNorm<ValueType>::build()
                .with_tolerance(reduction_factor)
                .on(exec))
        .with_krylov_dim(100u)
        .with_storage_precision(
            gko::solver::cb_gmres::storage_precision::keep)
        .on(exec);
auto solver_gen_reduce =
    cb_gmres::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1000u).on(exec),
            gko::stop::RelativeResidualNorm<ValueType>::build()
                .with_tolerance(reduction_factor)
                .on(exec))
        .with_krylov_dim(100u)
        .with_storage_precision(
            gko::solver::cb_gmres::storage_precision::reduce1)
        .on(exec);
auto solver_keep = solver_gen_keep->generate(A);
auto solver_reduce = solver_gen_reduce->generate(A);
auto time_keep = measure_solve_time_in_s(lend(exec), lend(solver_keep),
                                         lend(b), lend(x_keep));
auto time_reduce = measure_solve_time_in_s(lend(exec), lend(solver_reduce),
                                           lend(b), lend(x_reduce));
std::cout « std::scientific;
std::cout « "Solve time without compression: " « time_keep « " s\n"
          « "Solve time with compression:    " « time_reduce « " s\n";
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto res_norm_keep = gko::initialize<real_vec>({0.0}, exec);
auto res_norm_reduce = gko::initialize<real_vec>({0.0}, exec);
auto tmp = gko::clone(gko::lend(b));
A->apply(lend(one), lend(x_keep), lend(neg_one), lend(tmp));
tmp->compute_norm2(lend(res_norm_keep));
std::cout « "\nResidual norm without compression:\n";
write(std::cout, lend(res_norm_keep));
tmp->copy_from(lend(b));
A->apply(lend(one), lend(x_reduce), lend(neg_one), lend(tmp));
```

```
    tmp->compute_norm2(lend(res_norm_reduce));
    std::cout « "\nResidual norm with compression:\n";
    write(std::cout, lend(res_norm_reduce));
}
```

Chapter 12

The custom-logger program

The simple solver with a custom logger example..

This example depends on simple-solver, simple-solver-logging, minimal-cuda-solver.

Introduction

The custom-logger example shows how Ginkgo's API can be leveraged to implement application-specific callbacks for Ginkgo's events. This is the most basic way of extending Ginkgo and a good first step for any application developer who wants to adapt Ginkgo to his specific needs.

Ginkgo's gko::log::Logger abstraction provides hooks to the events that happen during the library execution. These hooks concern any low-level event such as memory allocations, deallocations, copies and kernel launches up to high-level events such as linear operator applications and completion of solver iterations.

In this example, a simple logger is implemented to track the solver's recurrent residual norm and compute the true residual norm. At the end of the solver execution, a comparison table is shown on-screen.

About the example

Each example has the following sections:

1. **Introduction:**This gives an overview of the example and mentions any interesting aspects in the example that might help the reader.
2. **The commented program:** This section is intended for you to understand the details of the example so that you can play with it and understand Ginkgo and its features better.
3. **Results:** This section shows the results of the code when run. Though the results may not be completely the same, you can expect the behaviour to be similar.
4. **The plain program:** This is the complete code without any comments to have an complete overview of the code.

The commented program

Include files

This is the main ginkgo header file.
```
#include <ginkgo/ginkgo.hpp>
```

Add the fstream header to read from data from files.
```
#include <fstream>
```

Add the map header for storing the executor map.
```
#include <map>
```

Add the C++ iomanip header to prettify the output.
```
#include <iomanip>
```

Add formatting flag modification capabilities.
```
#include <ios>
```

Add the C++ iostream header to output information to the console.
```
#include <iostream>
```

Add the string manipulation header to handle strings.
```
#include <string>
```

Add the vector header for storing the logger's data
```
#include <vector>
```

Utility function which returns the first element (position [0, 0]) from a given gko::matrix::Dense matrix / vector.
```
template <typename ValueType>
ValueType get_first_element(const gko::matrix::Dense<ValueType>* mtx)
{
```

Copy the matrix / vector to the host device before accessing the value in case it is stored in a GPU.
```
    return mtx->get_executor()->copy_val_to_host(mtx->get_const_values());
}
```

Utility function which computes the norm of a Ginkgo gko::matrix::Dense vector.
```
template <typename ValueType>
gko::remove_complex<ValueType> compute_norm(
    const gko::matrix::Dense<ValueType>* b)
{
```

Get the executor of the vector
```
auto exec = b->get_executor();
```

Initialize a result scalar containing the value 0.0.
```
auto b_norm =
    gko::initialize<gko::matrix::Dense<gko::remove_complex<ValueType>>(
        {0.0}, exec);
```

Use the dense `compute_norm2` function to compute the norm.
```
b->compute_norm2(gko::lend(b_norm));
```

Use the other utility function to return the norm contained in `b_norm`
```
    return get_first_element(gko::lend(b_norm));
}
```

Custom logger class which intercepts the residual norm scalar and solution vector in order to print a table of real vs recurrent (internal to the solvers) residual norms.
```
template <typename ValueType>
struct ResidualLogger : gko::log::Logger {
    using RealValueType = gko::remove_complex<ValueType>;
```

Output the logger's data in a table format
```
void write() const
{
```

Print a header for the table

```
std::cout « "Recurrent vs true vs implicit residual norm:"
          « std::endl;
std::cout « '|' « std::setw(10) « "Iteration" « '|' « std::setw(25)
          « "Recurrent Residual Norm" « '|' « std::setw(25)
          « "True Residual Norm" « '|' « std::setw(25)
          « "Implicit Residual Norm" « '|' « std::endl;
```

Print a separation line. Note that for creating `10` characters `std::setw()` should be set to `11`.

```
std::cout « '|' « std::setfill('-') « std::setw(11) « '|'
          « std::setw(26) « '|' « std::setw(26) « '|'
          « std::setw(26) « '|' « std::setfill(' ') « std::endl;
```

Print the data one by one in the form

```
std::cout « std::scientific;
for (std::size_t i = 0; i < iterations.size(); i++) {
    std::cout « '|' « std::setw(10) « iterations[i] « '|'
              « std::setw(25) « recurrent_norms[i] « '|'
              « std::setw(25) « real_norms[i] « '|' « std::setw(25)
              « implicit_norms[i] « '|' « std::endl;
}
```

std::defaultfloat could be used here but some compilers do not support it properly, e.g. the Intel compiler

```
std::cout.unsetf(std::ios_base::floatfield);
```

Print a separation line

```
    std::cout « '|' « std::setfill('-') « std::setw(11) « '|'
              « std::setw(26) « '|' « std::setw(26) « '|'
              « std::setw(26) « '|' « std::setfill(' ') « std::endl;
}
using gko_dense = gko::matrix::Dense<ValueType>;
using gko_real_dense = gko::matrix::Dense<RealValueType>;
```

This overload is necessary to avoid interface breaks for Ginkgo 2.0

```
void on_iteration_complete(const gko::LinOp* solver,
                           const gko::size_type& iteration,
                           const gko::LinOp* residual,
                           const gko::LinOp* solution,
                           const gko::LinOp* residual_norm) const override
{
    this->on_iteration_complete(solver, iteration, residual, solution,
                                residual_norm, nullptr);
}
```

Customize the logging hook which is called everytime an iteration is completed

```
void on_iteration_complete(
    const gko::LinOp*, const gko::size_type& iteration,
    const gko::LinOp* residual, const gko::LinOp* solution,
    const gko::LinOp* residual_norm,
    const gko::LinOp* implicit_sq_residual_norm) const override
{
```

If the solver shares a residual norm, log its value

```
if (residual_norm) {
    auto dense_norm = gko::as<gko_real_dense>(residual_norm);
```

Add the norm to the `recurrent_norms` vector

```
recurrent_norms.push_back(get_first_element(gko::lend(dense_norm)));
```

Otherwise, use the recurrent residual vector

```
} else {
    auto dense_residual = gko::as<gko_dense>(residual);
```

Compute the residual vector's norm

```
auto norm = compute_norm(gko::lend(dense_residual));
```

Add the computed norm to the `recurrent_norms` vector

```
    recurrent_norms.push_back(norm);
}
```

If the solver shares the current solution vector

```
if (solution) {
```

Store the matrix's executor

```
auto exec = matrix->get_executor();
```

Create a scalar containing the value 1.0
```
auto one = gko::initialize<gko_dense>({1.0}, exec);
```

Create a scalar containing the value -1.0
```
auto neg_one = gko::initialize<gko_dense>({-1.0}, exec);
```

Instantiate a temporary result variable
```
auto res = gko::clone(b);
```

Compute the real residual vector by calling apply on the system matrix
```
matrix->apply(gko::lend(one), gko::lend(solution),
              gko::lend(neg_one), gko::lend(res));
```

Compute the norm of the residual vector and add it to the `real_norms` vector
```
    real_norms.push_back(compute_norm(gko::lend(res)));
} else {
```

Add to the `real_norms` vector the value -1.0 if it could not be computed
```
    real_norms.push_back(-1.0);
}
if (implicit_sq_residual_norm) {
    auto dense_norm =
        gko::as<gko_real_dense>(implicit_sq_residual_norm);
```

Add the norm to the `implicit_norms` vector
```
    implicit_norms.push_back(
        std::sqrt(get_first_element(gko::lend(dense_norm))));
} else {
```

Add to the `implicit_norms` vector the value -1.0 if it could not be computed
```
    implicit_norms.push_back(-1.0);
}
```

Add the current iteration number to the `iterations` vector
```
    iterations.push_back(iteration);
}
```

Construct the logger and store the system matrix and b vectors
```
    ResidualLogger(std::shared_ptr<const gko::Executor> exec,
                   const gko::LinOp* matrix, const gko_dense* b)
        : gko::log::Logger(exec, gko::log::Logger::iteration_complete_mask),
          matrix{matrix},
          b{b}
    {}
private:
```

Pointer to the system matrix
```
const gko::LinOp* matrix;
```

Pointer to the right hand sides
```
const gko_dense* b;
```

Vector which stores all the recurrent residual norms
```
mutable std::vector<RealValueType> recurrent_norms{};
```

Vector which stores all the real residual norms
```
mutable std::vector<RealValueType> real_norms{};
```

Vector which stores all the implicit residual norms
```
mutable std::vector<RealValueType> implicit_norms{};
```

Vector which stores all the iteration numbers
```
    mutable std::vector<std::size_t> iterations{};
};
int main(int argc, char* argv[])
{
```

Use some shortcuts. In Ginkgo, vectors are seen as a gko::matrix::Dense with one column/one row. The advantage of this concept is that using multiple vectors is a now a natural extension of adding columns/rows are necessary.
```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
```

The gko::matrix::Csr class is used here, but any other matrix class such as gko::matrix::Coo, gko::matrix::Hybrid, gko::matrix::Ell or gko::matrix::Sellp could also be used.
```
using mtx = gko::matrix::Csr<ValueType, IndexType>;
```

The gko::solver::Cg is used here, but any other solver class can also be used.
```
using cg = gko::solver::Cg<ValueType>;
```

Print the ginkgo version information.
```
std::cout « gko::version_info::get() « std::endl;
```

Where do you want to run your solver ?

The gko::Executor class is one of the cornerstones of Ginkgo. Currently, we have support for an gko::OmpExecutor, which uses OpenMP multi-threading in most of its kernels, a gko::ReferenceExecutor, a single threaded specialization of the OpenMP executor and a gko::CudaExecutor which runs the code on a NVIDIA GPU if available.

Note

> With the help of C++, you see that you only ever need to change the executor and all the other functions/ routines within Ginkgo should automatically work and run on the executor with any other changes.

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
```

Figure out where to run the code
```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation
```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Reading your data and transfer to the proper device.

Read the matrix, right hand side and the initial solution using the read function.

Note

> Ginkgo uses C++ smart pointers to automatically manage memory. To this end, we use our own object ownership transfer functions that under the hood call the required smart pointer functions to manage object ownership. The gko::share , gko::give and gko::lend are the functions that you would need to use.

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
const RealValueType reduction_factor = 1e-7;
```

Creating the solver

Generate the gko::solver factory. Ginkgo uses the concept of Factories to build solvers with certain properties. Observe the Fluent interface used here. Here a cg solver is generated with a stopping criteria of maximum iterations of 20 and a residual norm reduction of 1e-15. You also observe that the stopping criteria(gko::stop) are also generated from factories using their build methods. You need to specify the executors which each of the object needs to be built on.
```
auto solver_gen =
    cg::build()
```

```
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(20u).on(exec),
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec))
        .on(exec);
```

Instantiate a ResidualLogger logger.

```
auto logger = std::make_shared<ResidualLogger<ValueType>>(
    exec, gko::lend(A), gko::lend(b));
```

Add the previously created logger to the solver factory. The logger will be automatically propagated to all solvers created from this factory.

```
solver_gen->add_logger(logger);
```

Generate the solver from the matrix. The solver factory built in the previous step takes a "matrix"(a gko::LinOp to be more general) as an input. In this case we provide it with a full matrix that we previously read, but as the solver only effectively uses the apply() method within the provided "matrix" object, you can effectively create a gko::LinOp class with your own apply implementation to accomplish more tasks. We will see an example of how this can be done in the custom-matrix-format example

```
auto solver = solver_gen->generate(A);
```

Finally, solve the system. The solver, being a gko::LinOp, can be applied to a right hand side, b to obtain the solution, x.

```
solver->apply(gko::lend(b), gko::lend(x));
```

Print the solution to the command line.

```
std::cout << "Solution (x):\n";
write(std::cout, gko::lend(x));
```

Print the table of the residuals obtained from the logger

```
logger->write();
```

To measure if your solution has actually converged, you can measure the error of the solution. one, neg_one are objects that represent the numbers which allow for a uniform interface when computing on any device. To compute the residual, all you need to do is call the apply method, which in this case is an spmv and equivalent to the LAPACK z_spmv routine. Finally, you compute the euclidean 2-norm with the compute_norm2 function.

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one), gko::lend(b));
    b->compute_norm2(gko::lend(res));
    std::cout << "Residual norm sqrt(r^T r):\n";
    write(std::cout, gko::lend(res));
}
```

Results

The following is the expected result:

```
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
```

```
0.0123025
Recurrent vs true vs implicit residual norm:
| Iteration|  Recurrent Residual Norm|       True Residual Norm|   Implicit Residual Norm|
|----------|-------------------------|-------------------------|-------------------------|
|         0|             4.358899e+00|             4.358899e+00|             4.358899e+00|
|         1|             2.304548e+00|             2.304548e+00|             2.304548e+00|
|         2|             1.467706e+00|             1.467706e+00|             1.467706e+00|
|         3|             9.848751e-01|             9.848751e-01|             9.848751e-01|
|         4|             7.418330e-01|             7.418330e-01|             7.418330e-01|
|         5|             5.136231e-01|             5.136231e-01|             5.136231e-01|
|         6|             3.841650e-01|             3.841650e-01|             3.841650e-01|
|         7|             3.164394e-01|             3.164394e-01|             3.164394e-01|
|         8|             2.277088e-01|             2.277088e-01|             2.277088e-01|
|         9|             1.703121e-01|             1.703121e-01|             1.703121e-01|
|        10|             9.737220e-02|             9.737220e-02|             9.737220e-02|
|        11|             6.168306e-02|             6.168306e-02|             6.168306e-02|
|        12|             4.541231e-02|             4.541231e-02|             4.541231e-02|
|        13|             3.195304e-02|             3.195304e-02|             3.195304e-02|
|        14|             1.616058e-02|             1.616058e-02|             1.616058e-02|
|        15|             6.570152e-03|             6.570152e-03|             6.570152e-03|
|        16|             2.643669e-03|             2.643669e-03|             2.643669e-03|
|        17|             8.588089e-04|             8.588089e-04|             8.588089e-04|
|        18|             2.864613e-04|             2.864613e-04|             2.864613e-04|
|        19|             1.641952e-15|             2.107881e-15|             1.641952e-15|
|----------|-------------------------|-------------------------|-------------------------|
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
2.10788e-15
```

Comments about programming and debugging

The plain program

```
/*****************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <map>
#include <iomanip>
#include <ios>
#include <iostream>
#include <string>
#include <vector>
template <typename ValueType>
ValueType get_first_element(const gko::matrix::Dense<ValueType>* mtx)
{
    return mtx->get_executor()->copy_val_to_host(mtx->get_const_values());
}
template <typename ValueType>
gko::remove_complex<ValueType> compute_norm(
    const gko::matrix::Dense<ValueType>* b)
```

```
{
    auto exec = b->get_executor();
    auto b_norm =
        gko::initialize<gko::matrix::Dense<gko::remove_complex<ValueType>>>(
            {0.0}, exec);
    b->compute_norm2(gko::lend(b_norm));
    return get_first_element(gko::lend(b_norm));
}
template <typename ValueType>
struct ResidualLogger : gko::log::Logger {
    using RealValueType = gko::remove_complex<ValueType>;
    void write() const
    {
        std::cout << "Recurrent vs true vs implicit residual norm:"
                  << std::endl;
        std::cout << '|' << std::setw(10) << "Iteration" << '|' << std::setw(25)
                  << "Recurrent Residual Norm" << '|' << std::setw(25)
                  << "True Residual Norm" << '|' << std::setw(25)
                  << "Implicit Residual Norm" << '|' << std::endl;
        std::cout << '|' << std::setfill('-') << std::setw(11) << '|'
                  << std::setw(26) << '|' << std::setw(26) << '|'
                  << std::setw(26) << '|' << std::setfill(' ') << std::endl;
        std::cout << std::scientific;
        for (std::size_t i = 0; i < iterations.size(); i++) {
            std::cout << '|' << std::setw(10) << iterations[i] << '|'
                      << std::setw(25) << recurrent_norms[i] << '|'
                      << std::setw(25) << real_norms[i] << '|' << std::setw(25)
                      << implicit_norms[i] << '|' << std::endl;
        }
        std::cout.unsetf(std::ios_base::floatfield);
        std::cout << '|' << std::setfill('-') << std::setw(11) << '|'
                  << std::setw(26) << '|' << std::setw(26) << '|'
                  << std::setw(26) << '|' << std::setfill(' ') << std::endl;
    }
    using gko_dense = gko::matrix::Dense<ValueType>;
    using gko_real_dense = gko::matrix::Dense<RealValueType>;
    void on_iteration_complete(const gko::LinOp* solver,
                               const gko::size_type& iteration,
                               const gko::LinOp* residual,
                               const gko::LinOp* solution,
                               const gko::LinOp* residual_norm) const override
    {
        this->on_iteration_complete(solver, iteration, residual, solution,
                                    residual_norm, nullptr);
    }
    void on_iteration_complete(
        const gko::LinOp*, const gko::size_type& iteration,
        const gko::LinOp* residual, const gko::LinOp* solution,
        const gko::LinOp* residual_norm,
        const gko::LinOp* implicit_sq_residual_norm) const override
    {
        if (residual_norm) {
            auto dense_norm = gko::as<gko_real_dense>(residual_norm);
            recurrent_norms.push_back(get_first_element(gko::lend(dense_norm)));
        } else {
            auto dense_residual = gko::as<gko_dense>(residual);
            auto norm = compute_norm(gko::lend(dense_residual));
            recurrent_norms.push_back(norm);
        }
        if (solution) {
            auto exec = matrix->get_executor();
            auto one = gko::initialize<gko_dense>({1.0}, exec);
            auto neg_one = gko::initialize<gko_dense>({-1.0}, exec);
            auto res = gko::clone(b);
            matrix->apply(gko::lend(one), gko::lend(solution),
                          gko::lend(neg_one), gko::lend(res));
            real_norms.push_back(compute_norm(gko::lend(res)));
        } else {
            real_norms.push_back(-1.0);
        }
        if (implicit_sq_residual_norm) {
            auto dense_norm =
                gko::as<gko_real_dense>(implicit_sq_residual_norm);
            implicit_norms.push_back(
                std::sqrt(get_first_element(gko::lend(dense_norm))));
        } else {
            implicit_norms.push_back(-1.0);
        }
        iterations.push_back(iteration);
    }
    ResidualLogger(std::shared_ptr<const gko::Executor> exec,
                   const gko::LinOp* matrix, const gko_dense* b)
        : gko::log::Logger(exec, gko::log::Logger::iteration_complete_mask),
          matrix{matrix},
          b{b}
    {}
private:
```

```
    const gko::LinOp* matrix;
    const gko_dense* b;
    mutable std::vector<RealValueType> recurrent_norms{};
    mutable std::vector<RealValueType> real_norms{};
    mutable std::vector<RealValueType> implicit_norms{};
    mutable std::vector<std::size_t> iterations{};
};
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    const RealValueType reduction_factor = 1e-7;
    auto solver_gen =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(20u).on(exec),
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec))
            .on(exec);
    auto logger = std::make_shared<ResidualLogger<ValueType»(
        exec, gko::lend(A), gko::lend(b));
    solver_gen->add_logger(logger);
    auto solver = solver_gen->generate(A);
    solver->apply(gko::lend(b), gko::lend(x));
    std::cout « "Solution (x):\n";
    write(std::cout, gko::lend(x));
    logger->write();
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one), gko::lend(b));
    b->compute_norm2(gko::lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, gko::lend(res));
}
```

Chapter 13

The custom-matrix-format program

The custom matrix format example..

This example depends on simple-solver, poisson-solver, three-pt-stencil-solver, .

Introduction

This example solves a 1D Poisson equation:

$u : [0, 1] \to R$
$u" = f$
$u(0) = u0$
$u(1) = u1$

using a finite difference method on an equidistant grid with `K` discretization points (`K` can be controlled with a command line parameter). The discretization is done via the second order Taylor polynomial:

$u(x + h) = u(x) - u'(x)h + 1/2u"(x)h^2 + O(h^3)$
$u(x - h) = u(x) + u'(x)h + 1/2u"(x)h^2 + O(h^3)/+$
$------------------------$
$-u(x - h) + 2u(x) + -u(x + h) = -f(x)h^2 + O(h^3)$

For an equidistant grid with K "inner" discretization points $x1, ..., xk$,and step size $h = 1/(K + 1)$, the formula produces a system of linear equations

$2u_1 - u_2 = -f_1h^2 + u0$
$-u_(k - 1) + 2u_k - u_(k + 1) = -f_kh^2, k = 2, ..., K - 1$
$-u_(K - 1) + 2u_K = -f_Kh^2 + u1$

which is then solved using Ginkgo's implementation of the CG method preconditioned with block-Jacobi. It is also possible to specify on which executor Ginkgo will solve the system via the command line. The function 'f'is set to '$f(x) = 6x$' (making the solution '$u(x) = x^3$'), but that can be changed in the `main` function.

The intention of this example is to show how a custom linear operator can be created and integrated into Ginkgo to achieve performance benefits.

About the example

The commented program

```
#include <iostream>
#include <map>
#include <string>
#include <omp.h>
#include <ginkgo/ginkgo.hpp>
```

A CUDA kernel implementing the stencil, which will be used if running on the CUDA executor. Unfortunately, NVCC has serious problems interpreting some parts of Ginkgo's code, so the kernel has to be compiled separately.

```
template <typename ValueType>
void stencil_kernel(std::size_t size, const ValueType* coefs,
                    const ValueType* b, ValueType* x);
```

A stencil matrix class representing the 3pt stencil linear operator. We include the gko::EnableLinOp mixin which implements the entire LinOp interface, except the two apply_impl methods, which get called inside the default implementation of apply (after argument verification) to perform the actual application of the linear operator. In addition, it includes the implementation of the entire PolymorphicObject interface.

It also includes the gko::EnableCreateMethod mixin which provides a default implementation of the static create method. This method will forward all its arguments to the constructor to create the object, and return an std::unique_ptr to the created object.

```
template <typename ValueType>
class StencilMatrix : public gko::EnableLinOp<StencilMatrix<ValueType>>,
                      public gko::EnableCreateMethod<StencilMatrix<ValueType>> {
public:
```

This constructor will be called by the create method. Here we initialize the coefficients of the stencil.

```
    StencilMatrix(std::shared_ptr<const gko::Executor> exec,
                  gko::size_type size = 0, ValueType left = -1.0,
                  ValueType center = 2.0, ValueType right = -1.0)
        : gko::EnableLinOp<StencilMatrix>(exec, gko::dim<2>{size}),
          coefficients(exec, {left, center, right})
    {}
protected:
    using vec = gko::matrix::Dense<ValueType>;
    using coef_type = gko::Array<ValueType>;
```

Here we implement the application of the linear operator, x = A ∗ b. apply_impl will be called by the apply method, after the arguments have been moved to the correct executor and the operators checked for conforming sizes.

For simplicity, we assume that there is always only one right hand side and the stride of consecutive elements in the vectors is 1 (both of these are always true in this example).

```
void apply_impl(const gko::LinOp* b, gko::LinOp* x) const override
{
```

we only implement the operator for dense RHS. gko::as will throw an exception if its argument is not Dense.

```
auto dense_b = gko::as<vec>(b);
auto dense_x = gko::as<vec>(x);
```

we need separate implementations depending on the executor, so we create an operation which maps the call to the correct implementation

```
struct stencil_operation : gko::Operation {
    stencil_operation(const coef_type& coefficients, const vec* b,
                      vec* x)
        : coefficients{coefficients}, b{b}, x{x}
    {}
```

OpenMP implementation

```
            void run(std::shared_ptr<const gko::OmpExecutor>) const override
            {
                auto b_values = b->get_const_values();
                auto x_values = x->get_values();
#pragma omp parallel for
                for (std::size_t i = 0; i < x->get_size()[0]; ++i) {
                    auto coefs = coefficients.get_const_data();
                    auto result = coefs[1] * b_values[i];
                    if (i > 0) {
                        result += coefs[0] * b_values[i - 1];
```

```
                    }
                    if (i < x->get_size()[0] - 1) {
                        result += coefs[2] * b_values[i + 1];
                    }
                    x_values[i] = result;
                }
            }
```

CUDA implementation

```
void run(std::shared_ptr<const gko::CudaExecutor>) const override
{
    stencil_kernel(x->get_size()[0], coefficients.get_const_data(),
                   b->get_const_values(), x->get_values());
}
```

We do not provide an implementation for reference executor. If not provided, Ginkgo will use the implementation for the OpenMP executor when calling it in the reference executor.

```
        const coef_type& coefficients;
        const vec* b;
        vec* x;
    };
    this->get_executor()->run(
        stencil_operation(coefficients, dense_b, dense_x));
}
```

There is also a version of the apply function which does the operation x = alpha ∗ A ∗ b + beta ∗ x. This function is commonly used and can often be better optimized than implementing it using x = A ∗ b. However, for simplicity, we will implement it exactly like that in this example.

```
    void apply_impl(const gko::LinOp* alpha, const gko::LinOp* b,
                    const gko::LinOp* beta, gko::LinOp* x) const override
    {
        auto dense_b = gko::as<vec>(b);
        auto dense_x = gko::as<vec>(x);
        auto tmp_x = dense_x->clone();
        this->apply_impl(b, lend(tmp_x));
        dense_x->scale(beta);
        dense_x->add_scaled(alpha, lend(tmp_x));
    }
private:
    coef_type coefficients;
};
```

Creates a stencil matrix in CSR format for the given number of discretization points.

```
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(gko::matrix::Csr<ValueType, IndexType>* matrix)
{
    const auto discretization_points = matrix->get_size()[0];
    auto row_ptrs = matrix->get_row_ptrs();
    auto col_idxs = matrix->get_col_idxs();
    auto values = matrix->get_values();
    IndexType pos = 0;
    const ValueType coefs[] = {-1, 2, -1};
    row_ptrs[0] = pos;
    for (int i = 0; i < discretization_points; ++i) {
        for (auto ofs : {-1, 0, 1}) {
            if (0 <= i + ofs && i + ofs < discretization_points) {
                values[pos] = coefs[ofs + 1];
                col_idxs[pos] = i + ofs;
                ++pos;
            }
        }
        row_ptrs[i + 1] = pos;
    }
}
```

Generates the RHS vector given `f` and the boundary conditions.

```
template <typename Closure, typename ValueType>
void generate_rhs(Closure f, ValueType u0, ValueType u1,
                  gko::matrix::Dense<ValueType>* rhs)
{
    const auto discretization_points = rhs->get_size()[0];
    auto values = rhs->get_values();
    const ValueType h = 1.0 / (discretization_points + 1);
    for (int i = 0; i < discretization_points; ++i) {
        const ValueType xi = ValueType(i + 1) * h;
        values[i] = -f(xi) * h * h;
    }
    values[0] += u0;
    values[discretization_points - 1] += u1;
}
```

Prints the solution u.

```
template <typename ValueType>
void print_solution(ValueType u0, ValueType u1,
                    const gko::matrix::Dense<ValueType>* u)
{
    std::cout « u0 « '\n';
    for (int i = 0; i < u->get_size()[0]; ++i) {
        std::cout « u->get_const_values()[i] « '\n';
    }
    std::cout « u1 « std::endl;
}
```

Computes the 1-norm of the error given the computed u and the correct solution function correct_u.

```
template <typename Closure, typename ValueType>
double calculate_error(int discretization_points,
                       const gko::matrix::Dense<ValueType>* u,
                       Closure correct_u)
{
    const auto h = 1.0 / (discretization_points + 1);
    auto error = 0.0;
    for (int i = 0; i < discretization_points; ++i) {
        using std::abs;
        const auto xi = (i + 1) * h;
        error +=
            abs(u->get_const_values()[i] - correct_u(xi)) / abs(correct_u(xi));
    }
    return error;
}
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
const unsigned int discretization_points =
    argc >= 3 ? std::atoi(argv[2]) : 100u;
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

executor used by the application

```
const auto app_exec = exec->get_master();
```

problem:

```
auto correct_u = [](ValueType x) { return x * x * x; };
auto f = [](ValueType x) { return ValueType{6} * x; };
auto u0 = correct_u(0);
auto u1 = correct_u(1);
```

initialize vectors

```
auto rhs = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
```

```
generate_rhs(f, u0, u1, lend(rhs));
auto u = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
for (int i = 0; i < u->get_size()[0]; ++i) {
    u->get_values()[i] = 0.0;
}
const RealValueType reduction_factor{1e-7};
```

Generate solver and solve the system

```
cg::build()
    .with_criteria(gko::stop::Iteration::build()
                       .with_max_iters(discretization_points)
                       .on(exec),
                   gko::stop::ResidualNorm<ValueType>::build()
                       .with_reduction_factor(reduction_factor)
                       .on(exec))
    .on(exec)
```

notice how our custom StencilMatrix can be used in the same way as any built-in type

```
        ->generate(StencilMatrix<ValueType>::create(exec, discretization_points,
                                                    -1, 2, -1))
        ->apply(lend(rhs), lend(u));
    std::cout « "\nSolve complete."
              « "\nThe average relative error is "
              « calculate_error(discretization_points, lend(u), correct_u) /
                     discretization_points
              « std::endl;
}
```

Results

Comments about programming and debugging

The plain program

```
/*****************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <iostream>
#include <map>
#include <string>
#include <omp.h>
#include <ginkgo/ginkgo.hpp>
template <typename ValueType>
void stencil_kernel(std::size_t size, const ValueType* coefs,
                    const ValueType* b, ValueType* x);
template <typename ValueType>
class StencilMatrix : public gko::EnableLinOp<StencilMatrix<ValueType>»,
                      public gko::EnableCreateMethod<StencilMatrix<ValueType>» {
```

```
public:
    StencilMatrix(std::shared_ptr<const gko::Executor> exec,
                  gko::size_type size = 0, ValueType left = -1.0,
                  ValueType center = 2.0, ValueType right = -1.0)
        : gko::EnableLinOp<StencilMatrix>(exec, gko::dim<2>{size}),
          coefficients(exec, {left, center, right})
    {}
protected:
    using vec = gko::matrix::Dense<ValueType>;
    using coef_type = gko::Array<ValueType>;
    void apply_impl(const gko::LinOp* b, gko::LinOp* x) const override
    {
        auto dense_b = gko::as<vec>(b);
        auto dense_x = gko::as<vec>(x);
        struct stencil_operation : gko::Operation {
            stencil_operation(const coef_type& coefficients, const vec* b,
                              vec* x)
                : coefficients{coefficients}, b{b}, x{x}
            {}
            void run(std::shared_ptr<const gko::OmpExecutor>) const override
            {
                auto b_values = b->get_const_values();
                auto x_values = x->get_values();
#pragma omp parallel for
                for (std::size_t i = 0; i < x->get_size()[0]; ++i) {
                    auto coefs = coefficients.get_const_data();
                    auto result = coefs[1] * b_values[i];
                    if (i > 0) {
                        result += coefs[0] * b_values[i - 1];
                    }
                    if (i < x->get_size()[0] - 1) {
                        result += coefs[2] * b_values[i + 1];
                    }
                    x_values[i] = result;
                }
            }
            void run(std::shared_ptr<const gko::CudaExecutor>) const override
            {
                stencil_kernel(x->get_size()[0], coefficients.get_const_data(),
                               b->get_const_values(), x->get_values());
            }
            const coef_type& coefficients;
            const vec* b;
            vec* x;
        };
        this->get_executor()->run(
            stencil_operation(coefficients, dense_b, dense_x));
    }
    void apply_impl(const gko::LinOp* alpha, const gko::LinOp* b,
                    const gko::LinOp* beta, gko::LinOp* x) const override
    {
        auto dense_b = gko::as<vec>(b);
        auto dense_x = gko::as<vec>(x);
        auto tmp_x = dense_x->clone();
        this->apply_impl(b, lend(tmp_x));
        dense_x->scale(beta);
        dense_x->add_scaled(alpha, lend(tmp_x));
    }
private:
    coef_type coefficients;
};
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(gko::matrix::Csr<ValueType, IndexType>* matrix)
{
    const auto discretization_points = matrix->get_size()[0];
    auto row_ptrs = matrix->get_row_ptrs();
    auto col_idxs = matrix->get_col_idxs();
    auto values = matrix->get_values();
    IndexType pos = 0;
    const ValueType coefs[] = {-1, 2, -1};
    row_ptrs[0] = pos;
    for (int i = 0; i < discretization_points; ++i) {
        for (auto ofs : {-1, 0, 1}) {
            if (0 <= i + ofs && i + ofs < discretization_points) {
                values[pos] = coefs[ofs + 1];
                col_idxs[pos] = i + ofs;
                ++pos;
            }
        }
        row_ptrs[i + 1] = pos;
    }
}
template <typename Closure, typename ValueType>
void generate_rhs(Closure f, ValueType u0, ValueType u1,
                  gko::matrix::Dense<ValueType>* rhs)
{
    const auto discretization_points = rhs->get_size()[0];
```

```
    auto values = rhs->get_values();
    const ValueType h = 1.0 / (discretization_points + 1);
    for (int i = 0; i < discretization_points; ++i) {
        const ValueType xi = ValueType(i + 1) * h;
        values[i] = -f(xi) * h * h;
    }
    values[0] += u0;
    values[discretization_points - 1] += u1;
}
template <typename ValueType>
void print_solution(ValueType u0, ValueType u1,
                    const gko::matrix::Dense<ValueType>* u)
{
    std::cout « u0 « '\n';
    for (int i = 0; i < u->get_size()[0]; ++i) {
        std::cout « u->get_const_values()[i] « '\n';
    }
    std::cout « u1 « std::endl;
}
template <typename Closure, typename ValueType>
double calculate_error(int discretization_points,
                       const gko::matrix::Dense<ValueType>* u,
                       Closure correct_u)
{
    const auto h = 1.0 / (discretization_points + 1);
    auto error = 0.0;
    for (int i = 0; i < discretization_points; ++i) {
        using std::abs;
        const auto xi = (i + 1) * h;
        error +=
            abs(u->get_const_values()[i] - correct_u(xi)) / abs(correct_u(xi));
    }
    return error;
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    const unsigned int discretization_points =
        argc >= 3 ? std::atoi(argv[2]) : 100u;
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    const auto app_exec = exec->get_master();
    auto correct_u = [](ValueType x) { return x * x * x; };
    auto f = [](ValueType x) { return ValueType{6} * x; };
    auto u0 = correct_u(0);
    auto u1 = correct_u(1);
    auto rhs = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
    generate_rhs(f, u0, u1, lend(rhs));
    auto u = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
    for (int i = 0; i < u->get_size()[0]; ++i) {
        u->get_values()[i] = 0.0;
    }
    const RealValueType reduction_factor{1e-7};
    cg::build()
        .with_criteria(gko::stop::Iteration::build()
                           .with_max_iters(discretization_points)
                           .on(exec),
                       gko::stop::ResidualNorm<ValueType>::build()
                           .with_reduction_factor(reduction_factor)
```

```
                            .on(exec))
        .on(exec)
        ->generate(StencilMatrix<ValueType>::create(exec, discretization_points,
                                                     -1, 2, -1))
        ->apply(lend(rhs), lend(u));
    std::cout « "\nSolve complete."
              « "\nThe average relative error is "
              « calculate_error(discretization_points, lend(u), correct_u) /
                     discretization_points
              « std::endl;
}
```

Chapter 14

The custom-stopping-criterion program

The custom stopping criterion creation example..

This example depends on simple-solver, minimal-cuda-solver.

Introduction

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <thread>
/ **
 * The ByInteraction class is a criterion which asks for user input to stop
 * the iteration process. Using this criterion is slightly more complex than the
 * other ones, because it is asynchronous therefore requires the use of threads.
 * /
class ByInteraction
    : public gko::EnablePolymorphicObject<ByInteraction, gko::stop::Criterion> {
    friend class gko::EnablePolymorphicObject<ByInteraction,
                                              gko::stop::Criterion>;
    using Criterion = gko::stop::Criterion;
public:
    GKO_CREATE_FACTORY_PARAMETERS(parameters, Factory)
    {
        / **
         * Boolean set by the user to stop the iteration process
         * /
        std::add_pointer<volatile bool>::type GKO_FACTORY_PARAMETER_SCALAR(
            stop_iteration_process, nullptr);
    };
    GKO_ENABLE_CRITERION_FACTORY(ByInteraction, parameters, Factory);
    GKO_ENABLE_BUILD_METHOD(Factory);
protected:
    bool check_impl(gko::uint8 stoppingId, bool setFinalized,
                    gko::Array<gko::stopping_status>* stop_status,
                    bool* one_changed, const Criterion::Updater&) override
    {
        bool result = *(parameters_.stop_iteration_process);
        if (result) {
            this->set_all_statuses(stoppingId, setFinalized, stop_status);
            *one_changed = true;
        }
        return result;
    }
    explicit ByInteraction(std::shared_ptr<const gko::Executor> exec)
        : EnablePolymorphicObject<ByInteraction, Criterion>(std::move(exec))
```

```
    {}
    explicit ByInteraction(const Factory* factory,
                           const gko::stop::CriterionArgs& args)
        : EnablePolymorphicObject<ByInteraction, Criterion>(
              factory->get_executor()),
          parameters_{factory->get_parameters()}
    {}
};
void run_solver(volatile bool* stop_iteration_process,
                std::shared_ptr<gko::Executor> exec)
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using bicg = gko::solver::Bicgstab<ValueType>;
```

Read Data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
```

Create solver factory and solve system

```
auto solver = bicg::build()
                  .with_criteria(ByInteraction::build()
                                     .with_stop_iteration_process(
                                         stop_iteration_process)
                                     .on(exec))
                  .on(exec)
                  ->generate(A);
solver->add_logger(gko::log::Stream<ValueType>::create(
    exec, gko::log::Logger::iteration_complete_mask, std::cout, true));
solver->apply(lend(b), lend(x));
std::cout « "Solver stopped" « std::endl;
```

Print solution

```
std::cout « "Solution (x): \n";
write(std::cout, lend(x));
```

Calculate residual

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r): \n";
    write(std::cout, lend(res));
}
int main(int argc, char* argv[])
{
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
```

Figure out where to run the code

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
```

```
                                                  true);
           }},
          {"dpcpp",
           [] {
               return gko::DpcppExecutor::create(0,
                                                    gko::OmpExecutor::create());
           }},
          {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Declare a user controled boolean for the iteration process

```
volatile bool stop_iteration_process{};
```

Create a new a thread to launch the solver

```
std::thread t(run_solver, &stop_iteration_process, exec);
```

Look for an input command "stop" in the console, which sets the boolean to true

```
    std::cout « "Type 'stop' to stop the iteration process" « std::endl;
    std::string command;
    while (std::cin » command) {
        if (command == "stop") {
            break;
        } else {
            std::cout « "Unknown command" « std::endl;
        }
    }
    std::cout « "User input command 'stop' - The solver will stop!"
              « std::endl;
    stop_iteration_process = true;
    t.join();
}
```

Results

This is the expected output:

```
.
.
.
.
.
.
[LOG] »> iteration 22516 completed with solver LinOp[gko::solver::Bicgstab<double>,0x7fe6a4003710] with
       residual LinOp[gko::matrix::Dense<double>,0x7fe6a40050b0], solution
       LinOp[gko::matrix::Dense<double>,0x7fe6a40048e0] and residual_norm LinOp[gko::LinOp const*,0]
LinOp[gko::matrix::Dense<double>,0x7fe6a40050b0][
    5.17803e-164
    -7.6865e-165
    -2.06149e-164
    -4.84737e-165
    -3.36597e-164
    2.22353e-164
    1.47594e-165
    -1.78592e-165
    -6.17274e-166
    -3.02681e-166
    7.82009e-166
    8.57102e-165
    -1.28879e-164
    -2.62076e-165
    2.55329e-165
    -5.95988e-166
    -5.79273e-166
    -5.20172e-166
    -6.79458e-166
]
// Typing 'stop' stops the solver.
User input command 'stop' - The solver will stop
LinOp[gko::matrix::Dense<double>,0x7fe6a40048e0][
    0.252218
    0.108645
    0.0662811
    0.0630433
    0.0384088
    0.0396536
    0.0402648
```

```
    0.0338935
    0.0193098
    0.0234653
    0.0211499
    0.0196413
    0.0199151
    0.0181674
    0.0162722
    0.0150714
    0.0107016
    0.0121141
    0.0123025
]
Solver stopped
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
6.50306e-16
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
```

```
#include <thread>
class ByInteraction
    : public gko::EnablePolymorphicObject<ByInteraction, gko::stop::Criterion> {
    friend class gko::EnablePolymorphicObject<ByInteraction,
                                              gko::stop::Criterion>;
    using Criterion = gko::stop::Criterion;
public:
    GKO_CREATE_FACTORY_PARAMETERS(parameters, Factory)
    {
        std::add_pointer<volatile bool>::type GKO_FACTORY_PARAMETER_SCALAR(
            stop_iteration_process, nullptr);
    };
    GKO_ENABLE_CRITERION_FACTORY(ByInteraction, parameters, Factory);
    GKO_ENABLE_BUILD_METHOD(Factory);
protected:
    bool check_impl(gko::uint8 stoppingId, bool setFinalized,
                    gko::Array<gko::stopping_status>* stop_status,
                    bool* one_changed, const Criterion::Updater&) override
    {
        bool result = *(parameters_.stop_iteration_process);
        if (result) {
            this->set_all_statuses(stoppingId, setFinalized, stop_status);
            *one_changed = true;
        }
        return result;
    }
    explicit ByInteraction(std::shared_ptr<const gko::Executor> exec)
        : EnablePolymorphicObject<ByInteraction, Criterion>(std::move(exec))
    {}
    explicit ByInteraction(const Factory* factory,
                           const gko::stop::CriterionArgs& args)
        : EnablePolymorphicObject<ByInteraction, Criterion>(
              factory->get_executor()),
          parameters_{factory->get_parameters()}
    {}
};
void run_solver(volatile bool* stop_iteration_process,
                std::shared_ptr<gko::Executor> exec)
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using bicg = gko::solver::Bicgstab<ValueType>;
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    auto solver = bicg::build()
                      .with_criteria(ByInteraction::build()
                                         .with_stop_iteration_process(
                                             stop_iteration_process)
                                         .on(exec))
                      .on(exec)
                      ->generate(A);
    solver->add_logger(gko::log::Stream<ValueType>::create(
        exec, gko::log::Logger::iteration_complete_mask, std::cout, true));
    solver->apply(lend(b), lend(x));
    std::cout « "Solver stopped" « std::endl;
    std::cout « "Solution (x): \n";
    write(std::cout, lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r): \n";
    write(std::cout, lend(res));
}
int main(int argc, char* argv[])
{
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
```

```
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                    gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    volatile bool stop_iteration_process{};
    std::thread t(run_solver, &stop_iteration_process, exec);
    std::cout « "Type 'stop' to stop the iteration process" « std::endl;
    std::string command;
    while (std::cin » command) {
        if (command == "stop") {
            break;
        } else {
            std::cout « "Unknown command" « std::endl;
        }
    }
    std::cout « "User input command 'stop' - The solver will stop!"
              « std::endl;
    stop_iteration_process = true;
    t.join();
}
```

Chapter 15

The external-lib-interfacing program

The external library(deal.II) interfacing example..

Introduction

About the example

The commented program

```
#include <deal.II/base/function.h>
#include <deal.II/base/logstream.h>
#include <deal.II/base/quadrature_lib.h>
#include <deal.II/dofs/dof_accessor.h>
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/dofs/dof_tools.h>
#include <deal.II/fe/fe_q.h>
#include <deal.II/fe/fe_values.h>
#include <deal.II/grid/grid_generator.h>
#include <deal.II/grid/grid_out.h>
#include <deal.II/grid/grid_refinement.h>
#include <deal.II/grid/tria.h>
#include <deal.II/grid/tria_accessor.h>
#include <deal.II/grid/tria_iterator.h>
#include <deal.II/lac/constraint_matrix.h>
#include <deal.II/lac/dynamic_sparsity_pattern.h>
#include <deal.II/lac/full_matrix.h>
#include <deal.II/lac/precondition.h>
#include <deal.II/lac/solver_bicgstab.h>
#include <deal.II/lac/sparse_matrix.h>
#include <deal.II/lac/vector.h>
#include <deal.II/numerics/data_out.h>
#include <deal.II/numerics/matrix_tools.h>
#include <deal.II/numerics/vector_tools.h>
```

The following two files provide classes and information for multithreaded programs. In the first one, the classes and functions are declared which we need to do assembly in parallel (i.e. the `WorkStream` namespace). The second file has a class MultithreadInfo which can be used to query the number of processors in your system, which is often useful when deciding how many threads to start in parallel.

```
#include <deal.II/base/multithread_info.h>
#include <deal.II/base/work_stream.h>
```

The next new include file declares a base class `TensorFunction` not unlike the `Function` class, but with the difference that the return value is tensor-valued rather than scalar of vector-valued.

```
#include <deal.II/base/tensor_function.h>
#include <deal.II/numerics/error_estimator.h>
```

Ginkgo's header file

```
#include <ginkgo/ginkgo.hpp>
```

This is C++, as we want to write some output to disk:

```
#include <fstream>
#include <iostream>
```

The last step is as in previous programs:

```
namespace Step9 {
using namespace dealii;
```

AdvectionProblem class declaration

Following we declare the main class of this program. It is very much like the main classes of previous examples, so we again only comment on the differences.

```
template <int dim>
class AdvectionProblem {
public:
    AdvectionProblem();
     AdvectionProblem();
    void run();
private:
    void setup_system();
```

The next set of functions will be used to assemble the matrix. However, unlike in the previous examples, the `assemble_system()` function will not do the work itself, but rather will delegate the actual assembly to helper functions `assemble_local_system()` and `copy_local_to_global()`. The rationale is that matrix assembly can be parallelized quite well, as the computation of the local contributions on each cell is entirely independent of other cells, and we only have to synchronize when we add the contribution of a cell to the global matrix.

The strategy for parallelization we choose here is one of the possibilities mentioned in detail in the threads module in the documentation. Specifically, we will use the WorkStream approach discussed there. Since there is so much documentation in this module, we will not repeat the rationale for the design choices here (for example, if you read through the module mentioned above, you will understand what the purpose of the `AssemblyScratchData` and `AssemblyCopyData` structures is). Rather, we will only discuss the specific implementation.

If you read the page mentioned above, you will find that in order to parallelize assembly, we need two data structures – one that corresponds to data that we need during local integration ("scratch data", i.e., things we only need as temporary storage), and one that carries information from the local integration to the function that then adds the local contributions to the corresponding elements of the global matrix. The former of these typically contains the FEValues and FEFaceValues objects, whereas the latter has the local matrix, local right hand side, and information about which degrees of freedom live on the cell for which we are assembling a local contribution. With this information, the following should be relatively self-explanatory:

```
struct AssemblyScratchData {
    AssemblyScratchData(const FiniteElement<dim>& fe);
    AssemblyScratchData(const AssemblyScratchData& scratch_data);
    FEValues<dim> fe_values;
    FEFaceValues<dim> fe_face_values;
};
struct AssemblyCopyData {
    FullMatrix<double> cell_matrix;
    Vector<double> cell_rhs;
    std::vector<types::global_dof_index> local_dof_indices;
};
void assemble_system();
void local_assemble_system(
    const typename DoFHandler<dim>::active_cell_iterator& cell,
    AssemblyScratchData& scratch, AssemblyCopyData& copy_data);
void copy_local_to_global(const AssemblyCopyData& copy_data);
```

The following functions again are as in previous examples, as are the subsequent variables.

```
    void solve();
    void refine_grid();
    void output_results(const unsigned int cycle) const;
    Triangulation<dim> triangulation;
    DoFHandler<dim> dof_handler;
    FE_Q<dim> fe;
    ConstraintMatrix hanging_node_constraints;
    SparsityPattern sparsity_pattern;
    SparseMatrix<double> system_matrix;
    Vector<double> solution;
    Vector<double> system_rhs;
};
```

Equation data declaration

Next we declare a class that describes the advection field. This, of course, is a vector field with as many components as there are space dimensions. One could now use a class derived from the `Function` base class, as we have done for boundary values and coefficients in previous examples, but there is another possibility in the library, namely a base class that describes tensor valued functions. In contrast to the usual `Function` objects, we provide the compiler with knowledge on the size of the objects of the return type. This enables the compiler to generate efficient code, which is not so simple for usual vector-valued functions where memory has to be allocated on the heap (thus, the `Function::vector_value` function has to be given the address of an object into which the result is to be written, in order to avoid copying and memory allocation and deallocation on the heap). In addition to the known size, it is possible not only to return vectors, but also tensors of higher rank; however, this is not very often requested by applications, to be honest...

The interface of the `TensorFunction` class is relatively close to that of the `Function` class, so there is probably no need to comment in detail the following declaration:

```
template <int dim>
class AdvectionField : public TensorFunction<1, dim> {
public:
    AdvectionField() : TensorFunction<1, dim>() {}
    virtual Tensor<1, dim> value(const Point<dim>& p) const;
    virtual void value_list(const std::vector<Point<dim>>& points,
                            std::vector<Tensor<1, dim>>& values) const;
```

In previous examples, we have used assertions that throw exceptions in several places. However, we have never seen how such exceptions are declared. This can be done as follows:

```
DeclException2(ExcDimensionMismatch, unsigned int, unsigned int,
               << "The vector has size " << arg1 << " but should have "
               << arg2 << " elements.");
```

The syntax may look a little strange, but is reasonable. The format is basically as follows: use the name of one of the macros `DeclExceptionN`, where `N` denotes the number of additional parameters which the exception object shall take. In this case, as we want to throw the exception when the sizes of two vectors differ, we need two arguments, so we use `DeclException2`. The first parameter then describes the name of the exception, while the following declare the data types of the parameters. The last argument is a sequence of output directives that will be piped into the `std::cerr` object, thus the strange format with the leading $<<$ operator and the like. Note that we can access the parameters which are passed to the exception upon construction (i.e. within the `Assert` call) by using the names `arg1` through `argN`, where `N` is the number of arguments as defined by the use of the respective macro `DeclExceptionN`.

To learn how the preprocessor expands this macro into actual code, please refer to the documentation of the exception classes in the base library. Suffice it to say that by this macro call, the respective exception class is declared, which also has error output functions already implemented.

```
};
```

The following two functions implement the interface described above. The first simply implements the function as described in the introduction, while the second uses the same trick to avoid calling a virtual function as has already been introduced in the previous example program. Note the check for the right sizes of the arguments in the second function, which should always be present in such functions; it is our experience that many if not most programming errors result from incorrectly initialized arrays, incompatible parameters to functions and the like; using assertion as in this case can eliminate many of these problems.

```
template <int dim>
Tensor<1, dim> AdvectionField<dim>::value(const Point<dim>& p) const
{
    Point<dim> value;
    value[0] = 2;
    for (unsigned int i = 1; i < dim; ++i)
        value[i] = 1 + 0.8 * std::sin(8 * numbers::PI * p[0]);
    return value;
}
template <int dim>
void AdvectionField<dim>::value_list(const std::vector<Point<dim>>& points,
                                     std::vector<Tensor<1, dim>>& values) const
{
    Assert(values.size() == points.size(),
           ExcDimensionMismatch(values.size(), points.size()));
    for (unsigned int i = 0; i < points.size(); ++i)
        values[i] = AdvectionField<dim>::value(points[i]);
}
```

Besides the advection field, we need two functions describing the source terms (`right hand side`) and the boundary values. First for the right hand side, which follows the same pattern as in previous examples. As described in the introduction, the source is a constant function in the vicinity of a source point, which we denote by the constant static variable `center_point`. We set the values of this center using the same template tricks as we have shown in the step-7 example program. The rest is simple and has been shown previously, including the way to avoid virtual function calls in the `value_list` function.

```
template <int dim>
class RightHandSide : public Function<dim> {
public:
    RightHandSide() : Function<dim>() {}
    virtual double value(const Point<dim>& p,
                         const unsigned int component = 0) const;
    virtual void value_list(const std::vector<Point<dim>>& points,
                            std::vector<double>& values,
                            const unsigned int component = 0) const;
private:
    static const Point<dim> center_point;
};
template <>
const Point<1> RightHandSide<1>::center_point = Point<1>(-0.75);
template <>
const Point<2> RightHandSide<2>::center_point = Point<2>(-0.75, -0.75);
template <>
const Point<3> RightHandSide<3>::center_point = Point<3>(-0.75, -0.75, -0.75);
```

The only new thing here is that we check for the value of the `component` parameter. As this is a scalar function, it is obvious that it only makes sense if the desired component has the index zero, so we assert that this is indeed the case. `ExcIndexRange` is a global predefined exception (probably the one most often used, we therefore made it global instead of local to some class), that takes three parameters: the index that is outside the allowed range, the first element of the valid range and the one past the last (i.e. again the half-open interval so often used in the C++ standard library):

```
template <int dim>
double RightHandSide<dim>::value(const Point<dim>& p,
                                 const unsigned int component) const
{
    (void)component;
    Assert(component == 0, ExcIndexRange(component, 0, 1));
    const double diameter = 0.1;
    return ((p - center_point).norm_square() < diameter * diameter
                ? .1 / std::pow(diameter, dim)
                : 0);
}
template <int dim>
void RightHandSide<dim>::value_list(const std::vector<Point<dim>>& points,
                                    std::vector<double>& values,
                                    const unsigned int component) const
{
    Assert(values.size() == points.size(),
           ExcDimensionMismatch(values.size(), points.size()));
    for (unsigned int i = 0; i < points.size(); ++i)
        values[i] = RightHandSide<dim>::value(points[i], component);
}
```

Finally for the boundary values, which is just another class derived from the `Function` base class:

```
template <int dim>
class BoundaryValues : public Function<dim> {
public:
    BoundaryValues() : Function<dim>() {}
    virtual double value(const Point<dim>& p,
                         const unsigned int component = 0) const;
    virtual void value_list(const std::vector<Point<dim>>& points,
                            std::vector<double>& values,
                            const unsigned int component = 0) const;
};
template <int dim>
double BoundaryValues<dim>::value(const Point<dim>& p,
                                  const unsigned int component) const
{
    (void)component;
    Assert(component == 0, ExcIndexRange(component, 0, 1));
    const double sine_term =
        std::sin(16 * numbers::PI * std::sqrt(p.norm_square()));
    const double weight = std::exp(-5 * p.norm_square()) / std::exp(-5.);
    return sine_term * weight;
}
template <int dim>
void BoundaryValues<dim>::value_list(const std::vector<Point<dim>>& points,
                                     std::vector<double>& values,
                                     const unsigned int component) const
{
    Assert(values.size() == points.size(),
           ExcDimensionMismatch(values.size(), points.size()));
    for (unsigned int i = 0; i < points.size(); ++i)
        values[i] = BoundaryValues<dim>::value(points[i], component);
}
```

GradientEstimation class declaration

Now, finally, here comes the class that will compute the difference approximation of the gradient on each cell and weighs that with a power of the mesh size, as described in the introduction. This class is a simple version of the `DerivativeApproximation` class in the library, that uses similar techniques to obtain finite difference approximations of the gradient of a finite element field, or of higher derivatives.

The class has one public static function `estimate` that is called to compute a vector of error indicators, and a few private functions that do the actual work on all active cells. As in other parts of the library, we follow an informal convention to use vectors of floats for error indicators rather than the common vectors of doubles, as the additional accuracy is not necessary for estimated values.

In addition to these two functions, the class declares two exceptions which are raised when a cell has no neighbors in each of the space directions (in which case the matrix described in the introduction would be singular and can't be inverted), while the other one is used in the more common case of invalid parameters to a function, namely a vector of wrong size.

Two other comments: first, the class has no non-static member functions or variables, so this is not really a class, but rather serves the purpose of a `namespace` in C++. The reason that we chose a class over a namespace is that this way we can declare functions that are private. This can be done with namespaces as well, if one declares some functions in header files in the namespace and implements these and other functions in the implementation file. The functions not declared in the header file are still in the namespace but are not callable from outside. However, as we have only one file here, it is not possible to hide functions in the present case.

The second comment is that the dimension template parameter is attached to the function rather than to the class itself. This way, you don't have to specify the template parameter yourself as in most other cases, but the compiler can figure its value out itself from the dimension of the DoF handler object that one passes as first argument.

Before jumping into the fray with the implementation, let us also comment on the parallelization strategy. We have already introduced the necessary framework for using the WorkStream concept in the declaration of the main class of this program above. We will use it again here. In the current context, this means that we have to define (i) classes for scratch and copy objects, (ii) a function that does the local computation on one cell, and (iii) a function that copies the local result into a global object. Given this general framework, we will, however, deviate from it a bit. In particular, WorkStream was generally invented for cases where each local computation on a cell *adds* to a global object – for example, when assembling linear systems where we add local contributions into a global matrix and right hand side. WorkStream is designed to handle the potential conflict of multiple threads trying to do this addition at the same time, and consequently has to provide for some way to ensure that only thread gets to do this at a time. Here, however, the situation is slightly different: we compute contributions from every cell individually, but then all we need to do is put them into an element of an output vector that is unique to each cell. Consequently, there is no risk that the write operations from two cells might conflict, and the elaborate machinery of WorkStream to avoid conflicting writes is not necessary. Consequently, what we will do is this: We still need a scratch object that holds, for example, the FEValues object. However, we only create a fake, empty copy data structure. Likewise, we do need the function that computes local contributions, but since it can already put the result into its final location, we do not need a copy-local-to-global function and will instead give the WorkStream::run() function an empty function object – the equivalent to a NULL function pointer.

```
class GradientEstimation {
public:
    template <int dim>
    static void estimate(const DoFHandler<dim>& dof,
                         const Vector<double>& solution,
                         Vector<float>& error_per_cell);
    DeclException2(ExcInvalidVectorLength, int, int,
                   « "Vector has length " « arg1 « ", but should have "
                   « arg2);
    DeclException0(ExcInsufficientDirections);
private:
    template <int dim>
    struct EstimateScratchData {
        EstimateScratchData(const FiniteElement<dim>& fe,
                            const Vector<double>& solution,
                            Vector<float>& error_per_cell);
        EstimateScratchData(const EstimateScratchData& data);
        FEValues<dim> fe_midpoint_value;
        const Vector<double>& solution;
        Vector<float>& error_per_cell;
```

```
    };
    struct EstimateCopyData {};
    template <int dim>
    static void estimate_cell(
        const typename DoFHandler<dim>::active_cell_iterator& cell,
        EstimateScratchData<dim>& scratch_data,
        const EstimateCopyData& copy_data);
};
```

AdvectionProblem class implementation

Now for the implementation of the main class. Constructor, destructor and the function `setup_system` follow the same pattern that was used previously, so we need not comment on these three function:

```
template <int dim>
AdvectionProblem<dim>::AdvectionProblem() : dof_handler(triangulation), fe(1)
{}
template <int dim>
AdvectionProblem<dim>:: AdvectionProblem()
{
    dof_handler.clear();
}
template <int dim>
void AdvectionProblem<dim>::setup_system()
{
    dof_handler.distribute_dofs(fe);
    hanging_node_constraints.clear();
    DoFTools::make_hanging_node_constraints(dof_handler,
                                            hanging_node_constraints);
    hanging_node_constraints.close();
    DynamicSparsityPattern dsp(dof_handler.n_dofs(), dof_handler.n_dofs());
    DoFTools::make_sparsity_pattern(dof_handler, dsp, hanging_node_constraints,
                                    / *keep_constrained_dofs = * / true);
    sparsity_pattern.copy_from(dsp);
    system_matrix.reinit(sparsity_pattern);
    solution.reinit(dof_handler.n_dofs());
    system_rhs.reinit(dof_handler.n_dofs());
}
```

In the following function, the matrix and right hand side are assembled. As stated in the documentation of the main class above, it does not do this itself, but rather delegates to the function following next, utilizing the WorkStream concept discussed in threads .

If you have looked through the threads module, you will have seen that assembling in parallel does not take an incredible amount of extra code as long as you diligently describe what the scratch and copy data objects are, and if you define suitable functions for the local assembly and the copy operation from local contributions to global objects. This done, the following will do all the heavy lifting to get these operations done on multiple threads on as many cores as you have in your system:

```
template <int dim>
void AdvectionProblem<dim>::assemble_system()
{
    WorkStream::run(dof_handler.begin_active(), dof_handler.end(), *this,
                    &AdvectionProblem::local_assemble_system,
                    &AdvectionProblem::copy_local_to_global,
                    AssemblyScratchData(fe), AssemblyCopyData());
```

After the matrix has been assembled in parallel, we still have to eliminate hanging node constraints. This is something that can't be done on each of the threads separately, so we have to do it now. Note also, that unlike in previous examples, there are no boundary conditions to be applied to the system of equations. This, of course, is due to the fact that we have included them into the weak formulation of the problem.

```
    hanging_node_constraints.condense(system_matrix);
    hanging_node_constraints.condense(system_rhs);
}
```

As already mentioned above, we need to have scratch objects for the parallel computation of local contributions. These objects contain FEValues and FEFaceValues objects, and so we will need to have constructors and copy constructors that allow us to create them. In initializing them, note first that we use bilinear elements, soGauss formulae with two points in each space direction are sufficient. For the cell terms we need the values and gradients of the shape functions, the quadrature points in order to determine the source density and the advection field at a given point, and the weights of the quadrature points times the determinant of the Jacobian at these points. In contrast, for the boundary integrals, we don't need the gradients, but rather the normal vectors to the cells. This determines which update flags we will have to pass to the constructors of the members of the class:

```
template <int dim>
AdvectionProblem<dim>::AssemblyScratchData::AssemblyScratchData(
    const FiniteElement<dim>& fe)
    : fe_values(fe, QGauss<dim>(2),
                update_values | update_gradients | update_quadrature_points |
                    update_JxW_values),
      fe_face_values(fe, QGauss<dim - 1>(2),
                     update_values | update_quadrature_points |
                         update_JxW_values | update_normal_vectors)
{}
template <int dim>
AdvectionProblem<dim>::AssemblyScratchData::AssemblyScratchData(
    const AssemblyScratchData& scratch_data)
    : fe_values(scratch_data.fe_values.get_fe(),
                scratch_data.fe_values.get_quadrature(),
                update_values | update_gradients | update_quadrature_points |
                    update_JxW_values),
      fe_face_values(scratch_data.fe_face_values.get_fe(),
                     scratch_data.fe_face_values.get_quadrature(),
                     update_values | update_quadrature_points |
                         update_JxW_values | update_normal_vectors)
{}
```

Now, this is the function that does the actual work. It is not very different from the `assemble_system` functions of previous example programs, so we will again only comment on the differences. The mathematical stuff follows closely what we have said in the introduction.

There are a number of points worth mentioning here, though. The first one is that we have moved the FEValues and FEFaceValues objects into the ScratchData object. We have done so because the alternative would have been to simply create one every time we get into this function – i.e., on every cell. It now turns out that the FEValues classes were written with the explicit goal of moving everything that remains the same from cell to cell into the construction of the object, and only do as little work as possible in FEValues::reinit() whenever we move to a new cell. What this means is that it would be very expensive to create a new object of this kind in this function as we would have to do it for every cell – exactly the thing we wanted to avoid with the FEValues class. Instead, what we do is create it only once (or a small number of times) in the scratch objects and then re-use it as often as we can.

This begs the question of whether there are other objects we create in this function whose creation is expensive compared to its use. Indeed, at the top of the function, we declare all sorts of objects. The `AdvectionField`, `RightHandSide` and `BoundaryValues` do not cost much to create, so there is no harm here. However, allocating memory in creating the `rhs_values` and similar variables below typically costs a significant amount of time, compared to just accessing the (temporary) values we store in them. Consequently, these would be candidates for moving into the `AssemblyScratchData` class. We will leave this as an exercise.

```
template <int dim>
void AdvectionProblem<dim>::local_assemble_system(
    const typename DoFHandler<dim>::active_cell_iterator& cell,
    AssemblyScratchData& scratch_data, AssemblyCopyData& copy_data)
{
```

First of all, we will need some objects that describe boundary values, right hand side function and the advection field. As we will only perform actions on these objects that do not change them, we declare them as constant, which can enable the compiler in some cases to perform additional optimizations.

```
const AdvectionField<dim> advection_field;
const RightHandSide<dim> right_hand_side;
const BoundaryValues<dim> boundary_values;
```

Then we define some abbreviations to avoid unnecessarily long lines:

```
const unsigned int dofs_per_cell = fe.dofs_per_cell;
const unsigned int n_q_points =
    scratch_data.fe_values.get_quadrature().size();
const unsigned int n_face_q_points =
    scratch_data.fe_face_values.get_quadrature().size();
```

We declare cell matrix and cell right hand side...

```
copy_data.cell_matrix.reinit(dofs_per_cell, dofs_per_cell);
copy_data.cell_rhs.reinit(dofs_per_cell);
```

... an array to hold the global indices of the degrees of freedom of the cell on which we are presently working...

```
copy_data.local_dof_indices.resize(dofs_per_cell);
```

... and array in which the values of right hand side, advection direction, and boundary values will be stored, for cell and face integrals respectively:

```
std::vector<double> rhs_values(n_q_points);
std::vector<Tensor<1, dim» advection_directions(n_q_points);
std::vector<double> face_boundary_values(n_face_q_points);
std::vector<Tensor<1, dim» face_advection_directions(n_face_q_points);
```

... then initialize the `FEValues` object...

```
scratch_data.fe_values.reinit(cell);
```

... obtain the values of right hand side and advection directions at the quadrature points...

```
advection_field.value_list(scratch_data.fe_values.get_quadrature_points(),
                           advection_directions);
right_hand_side.value_list(scratch_data.fe_values.get_quadrature_points(),
                           rhs_values);
```

... set the value of the streamline diffusion parameter as described in the introduction...

```
const double delta = 0.1 * cell->diameter();
```

... and assemble the local contributions to the system matrix and right hand side as also discussed above:

```
for (unsigned int q_point = 0; q_point < n_q_points; ++q_point)
    for (unsigned int i = 0; i < dofs_per_cell; ++i) {
        for (unsigned int j = 0; j < dofs_per_cell; ++j)
            copy_data.cell_matrix(i, j) +=
                ((advection_directions[q_point] *
                  scratch_data.fe_values.shape_grad(j, q_point) *
                  (scratch_data.fe_values.shape_value(i, q_point) +
                   delta *
                       (advection_directions[q_point] *
                        scratch_data.fe_values.shape_grad(i, q_point)))) *
                 scratch_data.fe_values.JxW(q_point));
        copy_data.cell_rhs(i) +=
            ((scratch_data.fe_values.shape_value(i, q_point) +
              delta * (advection_directions[q_point] *
                       scratch_data.fe_values.shape_grad(i, q_point))) *
             rhs_values[q_point] * scratch_data.fe_values.JxW(q_point));
    }
```

Besides the cell terms which we have built up now, the bilinear form of the present problem also contains terms on the boundary of the domain. Therefore, we have to check whether any of the faces of this cell are on the boundary of the domain, and if so assemble the contributions of this face as well. Of course, the bilinear form only contains contributions from the `inflow` part of the boundary, but to find out whether a certain part of a face of the present cell is part of the inflow boundary, we have to have information on the exact location of the quadrature points and on the direction of flow at this point; we obtain this information using the FEFaceValues object and only decide within the main loop whether a quadrature point is on the inflow boundary.

```
for (unsigned int face = 0; face < GeometryInfo<dim>::faces_per_cell;
     ++face)
    if (cell->face(face)->at_boundary()) {
```

Ok, this face of the present cell is on the boundary of the domain. Just as for the usual FEValues object which we have used in previous examples and also above, we have to reinitialize the FEFaceValues object for the present face:

```
scratch_data.fe_face_values.reinit(cell, face);
```

For the quadrature points at hand, we ask for the values of the inflow function and for the direction of flow:

```
boundary_values.value_list(
    scratch_data.fe_face_values.get_quadrature_points(),
    face_boundary_values);
advection_field.value_list(
    scratch_data.fe_face_values.get_quadrature_points(),
    face_advection_directions);
```

Now loop over all quadrature points and see whether it is on the inflow or outflow part of the boundary. This is determined by a test whether the advection direction points inwards or outwards of the domain (note that the normal vector points outwards of the cell, and since the cell is at the boundary, the normal vector points outward of the domain, so if the advection direction points into the domain, its scalar product with the normal vector must be negative):

```
for (unsigned int q_point = 0; q_point < n_face_q_points; ++q_point)
    if (scratch_data.fe_face_values.normal_vector(q_point) *
            face_advection_directions[q_point] <
        0)
```

If the is part of the inflow boundary, then compute the contributions of this face to the global matrix and right hand side, using the values obtained from the FEFaceValues object and the formulae discussed in the introduction:

```
            for (unsigned int i = 0; i < dofs_per_cell; ++i) {
                for (unsigned int j = 0; j < dofs_per_cell; ++j)
                    copy_data.cell_matrix(i, j) -=
                        (face_advection_directions[q_point] *
                         scratch_data.fe_face_values.normal_vector(
                             q_point) *
                         scratch_data.fe_face_values.shape_value(
                             i, q_point) *
                         scratch_data.fe_face_values.shape_value(
                             j, q_point) *
                         scratch_data.fe_face_values.JxW(q_point));
                copy_data.cell_rhs(i) -=
                    (face_advection_directions[q_point] *
                     scratch_data.fe_face_values.normal_vector(
                         q_point) *
                     face_boundary_values[q_point] *
                     scratch_data.fe_face_values.shape_value(i,
                                                             q_point) *
                     scratch_data.fe_face_values.JxW(q_point));
            }
}
```

Now go on by transferring the local contributions to the system of equations into the global objects. The first step was to obtain the global indices of the degrees of freedom on this cell.

```
    cell->get_dof_indices(copy_data.local_dof_indices);
}
```

The second function we needed to write was the one that copies the local contributions the previous function has computed and put into the copy data object, into the global matrix and right hand side vector objects. This is essentially what we always had as the last block of code when assembling something on every cell. The following should therefore be pretty obvious:

```
template <int dim>
void AdvectionProblem<dim>::copy_local_to_global(
    const AssemblyCopyData& copy_data)
{
    for (unsigned int i = 0; i < copy_data.local_dof_indices.size(); ++i) {
        for (unsigned int j = 0; j < copy_data.local_dof_indices.size(); ++j)
            system_matrix.add(copy_data.local_dof_indices[i],
                              copy_data.local_dof_indices[j],
                              copy_data.cell_matrix(i, j));
        system_rhs(copy_data.local_dof_indices[i]) += copy_data.cell_rhs(i);
    }
}
```

Following is the function that solves the linear system of equations. As the system is no more symmetric positive definite as in all the previous examples, we can't use the Conjugate Gradients method anymore. Rather, we use a solver that is tailored to nonsymmetric systems like the one at hand, the BiCGStab method. As preconditioner, we use the Block Jacobi method.

```
template <int dim>
void AdvectionProblem<dim>::solve()
{
```

Assert that the system be symmetric.

```
Assert(system_matrix.m() == system_matrix.n(), ExcNotQuadratic());
auto num_rows = system_matrix.m();
```

Make a copy of the rhs to use with Ginkgo.

```
std::vector<double> rhs(num_rows);
std::copy(system_rhs.begin(), system_rhs.begin() + num_rows, rhs.begin());
```

Ginkgo setup Some shortcuts: A vector is a Dense matrix with co-dimension 1. The matrix is setup in CSR. But various formats can be used. Look at Ginkgo's documentation.

```
using vec = gko::matrix::Dense<>;
using mtx = gko::matrix::Csr<>;
using bicgstab = gko::solver::Bicgstab<>;
using bj = gko::preconditioner::Jacobi<>;
using val_array = gko::Array<double>;
```

Where the code is to be executed. Can be changed to `omp` or `cuda` to run on multiple threads or on gpu's

```
std::shared_ptr<gko::Executor> exec = gko::ReferenceExecutor::create();
```

Setup Ginkgo's data structures

```
auto b = vec::create(exec, gko::dim<2>(num_rows, 1),
                     val_array::view(exec, num_rows, rhs.data()), 1);
auto x = vec::create(exec, gko::dim<2>(num_rows, 1));
auto A = mtx::create(exec, gko::dim<2>(num_rows),
```

```
                         system_matrix.n_nonzero_elements());
mtx::value_type* values = A->get_values();
mtx::index_type* row_ptr = A->get_row_ptrs();
mtx::index_type* col_idx = A->get_col_idxs();
```

Convert to standard CSR format As deal.ii does not expose its system matrix pointers, we construct them individually.

```
row_ptr[0] = 0;
for (auto row = 1; row <= num_rows; ++row) {
    row_ptr[row] = row_ptr[row - 1] + system_matrix.get_row_length(row - 1);
}
std::vector<mtx::index_type> ptrs(num_rows + 1);
std::copy(A->get_row_ptrs(), A->get_row_ptrs() + num_rows + 1,
          ptrs.begin());
for (auto row = 0; row < system_matrix.m(); ++row) {
    for (auto p = system_matrix.begin(row); p != system_matrix.end(row);
         ++p) {
```

write entry into the first free one for this row

```
col_idx[ptrs[row]] = p->column();
values[ptrs[row]] = p->value();
```

then move pointer ahead

```
        ++ptrs[row];
    }
}
```

Ginkgo solve The stopping criteria is set at maximum iterations of 1000 and a reduction factor of 1e-12. For other options, refer to Ginkgo's documentation.

```
auto solver_gen =
    bicgstab::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1000).on(exec),
            gko::stop::ResidualNorm<>::build()
                .with_reduction_factor(1e-12)
                .on(exec))
        .with_preconditioner(bj::build().on(exec))
        .on(exec);
auto solver = solver_gen->generate(gko::give(A));
```

Solve system

```
solver->apply(gko::lend(b), gko::lend(x));
```

Copy the solution vector back to deal.ii's data structures.

```
std::copy(x->get_values(), x->get_values() + num_rows, solution.begin());
/ *******************************************************
 * deal.ii internal solver. Here for reference.
 SolverControl           solver_control (1000, 1e-12);
 SolverBicgstab<>        bicgstab (solver_control);
 PreconditionJacobi<> preconditioner;
 preconditioner.initialize(system_matrix, 1.0);
 bicgstab.solve (system_matrix, solution, system_rhs,
                 preconditioner);
******************************************************* /
```

Give the solution back to deall.ii

```
    hanging_node_constraints.distribute(solution);
}
```

The following function refines the grid according to the quantity described in the introduction. The respective computations are made in the class `GradientEstimation`. The only difference to previous examples is that we refine a little more aggressively (0.5 instead of 0.3 of the number of cells).

```
template <int dim>
void AdvectionProblem<dim>::refine_grid()
{
    Vector<float> estimated_error_per_cell(triangulation.n_active_cells());
    GradientEstimation::estimate(dof_handler, solution,
                                 estimated_error_per_cell);
    GridRefinement::refine_and_coarsen_fixed_number(
        triangulation, estimated_error_per_cell, 0.5, 0.03);
    triangulation.execute_coarsening_and_refinement();
}
```

Writing output to disk is done in the same way as in the previous examples. Indeed, the function is identical to the one in step-6.

```
template <int dim>
```

```
void AdvectionProblem<dim>::output_results(const unsigned int cycle) const
{
    {
        GridOut grid_out;
        std::ofstream output("grid-" + std::to_string(cycle) + ".eps");
        grid_out.write_eps(triangulation, output);
    }
    {
        DataOut<dim> data_out;
        data_out.attach_dof_handler(dof_handler);
        data_out.add_data_vector(solution, "solution");
        data_out.build_patches();
        std::ofstream output("solution-" + std::to_string(cycle) + ".vtk");
        data_out.write_vtk(output);
    }
}
```

... as is the main loop (setup – solve – refine)

```
template <int dim>
void AdvectionProblem<dim>::run()
{
    for (unsigned int cycle = 0; cycle < 6; ++cycle) {
        std::cout « "Cycle " « cycle « ':' « std::endl;
        if (cycle == 0) {
            GridGenerator::hyper_cube(triangulation, -1, 1);
            triangulation.refine_global(4);
        } else {
            refine_grid();
        }
        std::cout « "   Number of active cells:       "
                  « triangulation.n_active_cells() « std::endl;
        setup_system();
        std::cout « "   Number of degrees of freedom: " « dof_handler.n_dofs()
                  « std::endl;
        assemble_system();
        solve();
        output_results(cycle);
    }
}
```

GradientEstimation class implementation

Now for the implementation of the `GradientEstimation` class. Let us start by defining constructors for the `EstimateScratchData` class used by the `estimate_cell()` function:

```
template <int dim>
GradientEstimation::EstimateScratchData<dim>::EstimateScratchData(
    const FiniteElement<dim>& fe, const Vector<double>& solution,
    Vector<float>& error_per_cell)
    : fe_midpoint_value(fe, QMidpoint<dim>(),
                        update_values | update_quadrature_points),
      solution(solution),
      error_per_cell(error_per_cell)
{}
template <int dim>
GradientEstimation::EstimateScratchData<dim>::EstimateScratchData(
    const EstimateScratchData& scratch_data)
    : fe_midpoint_value(scratch_data.fe_midpoint_value.get_fe(),
                        scratch_data.fe_midpoint_value.get_quadrature(),
                        update_values | update_quadrature_points),
      solution(scratch_data.solution),
      error_per_cell(scratch_data.error_per_cell)
{}
```

Next for the implementation of the `GradientEstimation` class. The first function does not much except for delegating work to the other function, but there is a bit of setup at the top.

Before starting with the work, we check that the vector into which the results are written has the right size. Programming mistakes in which one forgets to size arguments correctly at the calling site are quite common. Because the resulting damage from not catching such errors is often subtle (e.g., corruption of data somewhere in memory, or non-reproducible results), it is well worth the effort to check for such things.

```
template <int dim>
void GradientEstimation::estimate(const DoFHandler<dim>& dof_handler,
                                  const Vector<double>& solution,
                                  Vector<float>& error_per_cell)
{
    Assert(error_per_cell.size() ==
               dof_handler.get_triangulation().n_active_cells(),
```

```
          ExcInvalidVectorLength(
              error_per_cell.size(),
              dof_handler.get_triangulation().n_active_cells()));
    WorkStream::run(dof_handler.begin_active(), dof_handler.end(),
                    &GradientEstimation::template estimate_cell<dim>,
                    std::function<void(const EstimateCopyData&)>(),
                    EstimateScratchData<dim>(dof_handler.get_fe(), solution,
                                             error_per_cell),
                    EstimateCopyData());
}
```

Following now the function that actually computes the finite difference approximation to the gradient. The general outline of the function is to first compute the list of active neighbors of the present cell and then compute the quantities described in the introduction for each of the neighbors. The reason for this order is that it is not a one-liner to find a given neighbor with locally refined meshes. In principle, an optimized implementation would find neighbors and the quantities depending on them in one step, rather than first building a list of neighbors and in a second step their contributions but we will gladly leave this as an exercise. As discussed before, the worker function passed to WorkStream::run works on "scratch" objects that keep all temporary objects. This way, we do not need to create and initialize objects that are expensive to initialize within the function that does the work, every time it is called for a given cell. Such an argument is passed as the second argument. The third argument would be a "copy-data" object (see threads for more information) but we do not actually use any of these here. Because WorkStream::run() insists on passing three arguments, we declare this function with three arguments, but simply ignore the last one.

(This is unsatisfactory from an esthetic perspective. It can be avoided, at the cost of some other trickery. If you allow, let us here show how. First, assume that we had declared this function to only take two arguments by omitting the unused last one. Now, WorkStream::run still wants to call this function with three arguments, so we need to find a way to "forget" the third argument in the call. Simply passing WorkStream::run the pointer to the function as we do above will not do this – the compiler will complain that a function declared to have two arguments is called with three arguments. However, we can do this by passing the following as the third argument when calling WorkStream::run() above:

```
std::function<void (const typename DoFHandler<dim>::active_cell_iterator
&,
                    EstimateScratchData<dim>                  &,
                    EstimateCopyData                          &)>
  (std::bind (&GradientEstimation::template estimate_cell<dim>,
              std::placeholders::_1,
              std::placeholders::_2))
```

This creates a function object taking three arguments, but when it calls the underlying function object, it simply only uses the first and second argument – we simply "forget" to use the third argument :-) In the end, this isn't completely obvious either, and so we didn't implement it, but hey – it can be done!)

Now for the details:

```
template <int dim>
void GradientEstimation::estimate_cell(
    const typename DoFHandler<dim>::active_cell_iterator& cell,
    EstimateScratchData<dim>& scratch_data, const EstimateCopyData&)
{
```

We need space for the tensor `Y`, which is the sum of outer products of the y-vectors.

```
Tensor<2, dim> Y;
```

Then we allocate a vector to hold iterators to all active neighbors of a cell. We reserve the maximal number of active neighbors in order to avoid later reallocations. Note how this maximal number of active neighbors is computed here.

```
std::vector<typename DoFHandler<dim>::active_cell_iterator>
    active_neighbors;
active_neighbors.reserve(GeometryInfo<dim>::faces_per_cell *
                         GeometryInfo<dim>::max_children_per_face);
```

First initialize the `FEValues` object, as well as the `Y` tensor:

```
scratch_data.fe_midpoint_value.reinit(cell);
```

Then allocate the vector that will be the sum over the y-vectors times the approximate directional derivative:

```
Tensor<1, dim> projected_gradient;
```

Now before going on first compute a list of all active neighbors of the present cell. We do so by first looping over all faces and see whether the neighbor there is active, which would be the case if it is on the same level as the present cell or one level coarser (note that a neighbor can only be once coarser than the present cell, as we only allow a maximal difference of one refinement over a face in deal.II). Alternatively, the neighbor could be on the same level and be further refined; then we have to find which of its children are next to the present cell and select these (note that if a child of a neighbor of an active cell that is next to this active cell, needs necessarily be active itself, due to the one-refinement rule cited above).

Things are slightly different in one space dimension, as there the one-refinement rule does not exist: neighboring active cells may differ in as many refinement levels as they like. In this case, the computation becomes a little more difficult, but we will explain this below.

Before starting the loop over all neighbors of the present cell, we have to clear the array storing the iterators to the active neighbors, of course.

```
active_neighbors.clear();
for (unsigned int face_no = 0; face_no < GeometryInfo<dim>::faces_per_cell;
     ++face_no)
    if (!cell->at_boundary(face_no)) {
```

First define an abbreviation for the iterator to the face and the neighbor

```
const typename DoFHandler<dim>::face_iterator face =
    cell->face(face_no);
const typename DoFHandler<dim>::cell_iterator neighbor =
    cell->neighbor(face_no);
```

Then check whether the neighbor is active. If it is, then it is on the same level or one level coarser (if we are not in 1D), and we are interested in it in any case.

```
if (neighbor->active())
    active_neighbors.push_back(neighbor);
else {
```

If the neighbor is not active, then check its children.

```
if (dim == 1) {
```

To find the child of the neighbor which bounds to the present cell, successively go to its right child if we are left of the present cell (n==0), or go to the left child if we are on the right (n==1), until we find an active cell.

```
typename DoFHandler<dim>::cell_iterator neighbor_child =
    neighbor;
while (neighbor_child->has_children())
    neighbor_child =
        neighbor_child->child(face_no == 0 ? 1 : 0);
```

As this used some non-trivial geometrical intuition, we might want to check whether we did it right, i.e. check whether the neighbor of the cell we found is indeed the cell we are presently working on. Checks like this are often useful and have frequently uncovered errors both in algorithms like the line above (where it is simple to involuntarily exchange `n==1` for `n==0` or the like) and in the library (the assumptions underlying the algorithm above could either be wrong, wrongly documented, or are violated due to an error in the library). One could in principle remove such checks after the program works for some time, but it might be a good things to leave it in anyway to check for changes in the library or in the algorithm above.

Note that if this check fails, then this is certainly an error that is irrecoverable and probably qualifies as an internal error. We therefore use a predefined exception class to throw here.

```
Assert(
    neighbor_child->neighbor(face_no == 0 ? 1 : 0) == cell,
    ExcInternalError());
```

If the check succeeded, we push the active neighbor we just found to the stack we keep:

```
    active_neighbors.push_back(neighbor_child);
} else
```

If we are not in 1d, we collect all neighbor children ‘behind’ the subfaces of the current face

```
            for (unsigned int subface_no = 0;
                 subface_no < face->n_children(); ++subface_no)
                active_neighbors.push_back(
                    cell->neighbor_child_on_subface(face_no,
                                                    subface_no));
    }
}
```

OK, now that we have all the neighbors, lets start the computation on each of them. First we do some preliminaries: find out about the center of the present cell and the solution at this point. The latter is obtained as a vector of function values at the quadrature points, of which there are only one, of course. Likewise, the position of the center is the position of the first (and only) quadrature point in real space.

```
const Point<dim> this_center =
    scratch_data.fe_midpoint_value.quadrature_point(0);
std::vector<double> this_midpoint_value(1);
scratch_data.fe_midpoint_value.get_function_values(scratch_data.solution,
                                                   this_midpoint_value);
```

Now loop over all active neighbors and collect the data we need. Allocate a vector just like `this_midpoint_value` which we will use to store the value of the solution in the midpoint of the neighbor cell. We allocate it here already, since that way we don't have to allocate memory repeatedly in each iteration of this inner loop (memory allocation is a rather expensive operation):

```
std::vector<double> neighbor_midpoint_value(1);
typename std::vector<typename DoFHandler<dim>::active_cell_iterator>::
    const_iterator neighbor_ptr = active_neighbors.begin();
for (; neighbor_ptr != active_neighbors.end(); ++neighbor_ptr) {
```

First define an abbreviation for the iterator to the active neighbor cell:

```
const typename DoFHandler<dim>::active_cell_iterator neighbor =
    *neighbor_ptr;
```

Then get the center of the neighbor cell and the value of the finite element function thereon. Note that for this information we have to reinitialize the `FEValues` object for the neighbor cell.

```
scratch_data.fe_midpoint_value.reinit(neighbor);
const Point<dim> neighbor_center =
    scratch_data.fe_midpoint_value.quadrature_point(0);
scratch_data.fe_midpoint_value.get_function_values(
    scratch_data.solution, neighbor_midpoint_value);
```

Compute the vector `y` connecting the centers of the two cells. Note that as opposed to the introduction, we denote by `y` the normalized difference vector, as this is the quantity used everywhere in the computations.

```
Tensor<1, dim> y = neighbor_center - this_center;
const double distance = y.norm();
y /= distance;
```

Then add up the contribution of this cell to the Y matrix...

```
for (unsigned int i = 0; i < dim; ++i)
    for (unsigned int j = 0; j < dim; ++j) Y[i][j] += y[i] * y[j];
```

... and update the sum of difference quotients:

```
    projected_gradient +=
        (neighbor_midpoint_value[0] - this_midpoint_value[0]) / distance *
        y;
}
```

If now, after collecting all the information from the neighbors, we can determine an approximation of the gradient for the present cell, then we need to have passed over vectors `y` which span the whole space, otherwise we would not have all components of the gradient. This is indicated by the invertibility of the matrix.

If the matrix should not be invertible, this means that the present cell had an insufficient number of active neighbors. In contrast to all previous cases, where we raised exceptions, this is, however, not a programming error: it is a runtime error that can happen in optimized mode even if it ran well in debug mode, so it is reasonable to try to catch this error also in optimized mode. For this case, there is the `AssertThrow` macro: it checks the condition like the `Assert` macro, but not only in debug mode; it then outputs an error message, but instead of terminating the program as in the case of the `Assert` macro, the exception is thrown using the `throw` command of C++. This way, one has the possibility to catch this error and take reasonable counter actions. One such measure would be to refine the grid globally, as the case of insufficient directions can not occur if every cell of the initial grid has been refined at least once.

```
AssertThrow(determinant(Y) != 0, ExcInsufficientDirections());
```

If, on the other hand the matrix is invertible, then invert it, multiply the other quantity with it and compute the estimated error using this quantity and the right powers of the mesh width:

```
const Tensor<2, dim> Y_inverse = invert(Y);
Tensor<1, dim> gradient = Y_inverse * projected_gradient;
```

The last part of this function is the one where we write into the element of the output vector what we have just computed. The address of this vector has been stored in the scratch data object, and all we have to do is know how to get at the correct element inside this vector – but we can ask the cell we're on the how-manyth active cell it is for this:

```
    scratch_data.error_per_cell(cell->active_cell_index()) =
        (std::pow(cell->diameter(), 1 + 1.0 * dim / 2) *
         std::sqrt(gradient.norm_square()));
}
}  // namespace Step9
```

Main function

The `main` function is similar to the previous examples. The main difference is that we use MultithreadInfo to set the maximum number of threads (see Parallel computing with multiple processors accessing shared memory" documentation module for more explanation). The number of threads used is the minimum of the environment variable DEAL_II_NUM_THREADS and the parameter of `set_thread_limit`. If no value is given to `set_thread_limit`, the default value from the Intel Threading Building Blocks (TBB) library is used. If the call to `set_thread_limit` is omitted, the number of threads will be chosen by TBB indepently of DEAL_II_NUM_THREADS.

```
int main()
{
    try {
        dealii::MultithreadInfo::set_thread_limit();
        Step9::AdvectionProblem<2> advection_problem_2d;
        advection_problem_2d.run();
    } catch (std::exception& exc) {
        std::cerr « std::endl
                  « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        std::cerr « "Exception on processing: " « std::endl
                  « exc.what() « std::endl
                  « "Aborting!" « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        return 1;
    } catch (...) {
        std::cerr « std::endl
                  « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        std::cerr « "Unknown exception!" « std::endl
                  « "Aborting!" « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        return 1;
    }
    return 0;
}
```

Results

Comments about programming and debugging

The plain program

```
/* ---------------------------------------------------------------------
 *
 * Copyright (C) 2000 - 2018 by the deal.II authors
 *
 * This file is part of the deal.II library.
 *
 * The deal.II library is free software; you can use it, redistribute
 * it, and/or modify it under the terms of the GNU Lesser General
 * Public License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 * The full text of the license can be found in the file LICENSE at
 * the top level of the deal.II distribution.
 *
 * ---------------------------------------------------------------------

 *
 * Author: Wolfgang Bangerth, University of Heidelberg, 2000
 */
/* ---------------------------------------------------------------------
 *
 * This file has been taken verbatim from the deal.ii (version 9.0)
 * examples directory and modified.
 *
 * This example aims to demonstrate the ease with which Ginkgo can
 * be interfaced with other libraries. The only modification/ addition
 * has been to the AdvectionProblem::solve () function.
 *
```

```
 */
#include <deal.II/base/function.h>
#include <deal.II/base/logstream.h>
#include <deal.II/base/quadrature_lib.h>
#include <deal.II/dofs/dof_accessor.h>
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/dofs/dof_tools.h>
#include <deal.II/fe/fe_q.h>
#include <deal.II/fe/fe_values.h>
#include <deal.II/grid/grid_generator.h>
#include <deal.II/grid/grid_out.h>
#include <deal.II/grid/grid_refinement.h>
#include <deal.II/grid/tria.h>
#include <deal.II/grid/tria_accessor.h>
#include <deal.II/grid/tria_iterator.h>
#include <deal.II/lac/constraint_matrix.h>
#include <deal.II/lac/dynamic_sparsity_pattern.h>
#include <deal.II/lac/full_matrix.h>
#include <deal.II/lac/precondition.h>
#include <deal.II/lac/solver_bicgstab.h>
#include <deal.II/lac/sparse_matrix.h>
#include <deal.II/lac/vector.h>
#include <deal.II/numerics/data_out.h>
#include <deal.II/numerics/matrix_tools.h>
#include <deal.II/numerics/vector_tools.h>
#include <deal.II/base/multithread_info.h>
#include <deal.II/base/work_stream.h>
#include <deal.II/base/tensor_function.h>
#include <deal.II/numerics/error_estimator.h>
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
namespace Step9 {
using namespace dealii;
template <int dim>
class AdvectionProblem {
public:
    AdvectionProblem();
     AdvectionProblem();
    void run();
private:
    void setup_system();
    struct AssemblyScratchData {
        AssemblyScratchData(const FiniteElement<dim>& fe);
        AssemblyScratchData(const AssemblyScratchData& scratch_data);
        FEValues<dim> fe_values;
        FEFaceValues<dim> fe_face_values;
    };
    struct AssemblyCopyData {
        FullMatrix<double> cell_matrix;
        Vector<double> cell_rhs;
        std::vector<types::global_dof_index> local_dof_indices;
    };
    void assemble_system();
    void local_assemble_system(
        const typename DoFHandler<dim>::active_cell_iterator& cell,
        AssemblyScratchData& scratch, AssemblyCopyData& copy_data);
    void copy_local_to_global(const AssemblyCopyData& copy_data);
    void solve();
    void refine_grid();
    void output_results(const unsigned int cycle) const;
    Triangulation<dim> triangulation;
    DoFHandler<dim> dof_handler;
    FE_Q<dim> fe;
    ConstraintMatrix hanging_node_constraints;
    SparsityPattern sparsity_pattern;
    SparseMatrix<double> system_matrix;
    Vector<double> solution;
    Vector<double> system_rhs;
};
template <int dim>
class AdvectionField : public TensorFunction<1, dim> {
public:
    AdvectionField() : TensorFunction<1, dim>() {}
    virtual Tensor<1, dim> value(const Point<dim>& p) const;
    virtual void value_list(const std::vector<Point<dim»& points,
                            std::vector<Tensor<1, dim»& values) const;
    DeclException2(ExcDimensionMismatch, unsigned int, unsigned int,
                   « "The vector has size " « arg1 « " but should have "
                   « arg2 « " elements.");
};
template <int dim>
Tensor<1, dim> AdvectionField<dim>::value(const Point<dim>& p) const
{
    Point<dim> value;
    value[0] = 2;
    for (unsigned int i = 1; i < dim; ++i)
```

```
        value[i] = 1 + 0.8 * std::sin(8 * numbers::PI * p[0]);
    return value;
}
template <int dim>
void AdvectionField<dim>::value_list(const std::vector<Point<dim»& points,
                                     std::vector<Tensor<1, dim»& values) const
{
    Assert(values.size() == points.size(),
           ExcDimensionMismatch(values.size(), points.size()));
    for (unsigned int i = 0; i < points.size(); ++i)
        values[i] = AdvectionField<dim>::value(points[i]);
}
template <int dim>
class RightHandSide : public Function<dim> {
public:
    RightHandSide() : Function<dim>() {}
    virtual double value(const Point<dim>& p,
                         const unsigned int component = 0) const;
    virtual void value_list(const std::vector<Point<dim»& points,
                            std::vector<double>& values,
                            const unsigned int component = 0) const;
private:
    static const Point<dim> center_point;
};
template <>
const Point<1> RightHandSide<1>::center_point = Point<1>(-0.75);
template <>
const Point<2> RightHandSide<2>::center_point = Point<2>(-0.75, -0.75);
template <>
const Point<3> RightHandSide<3>::center_point = Point<3>(-0.75, -0.75, -0.75);
template <int dim>
double RightHandSide<dim>::value(const Point<dim>& p,
                                 const unsigned int component) const
{
    (void)component;
    Assert(component == 0, ExcIndexRange(component, 0, 1));
    const double diameter = 0.1;
    return ((p - center_point).norm_square() < diameter * diameter
                ? .1 / std::pow(diameter, dim)
                : 0);
}
template <int dim>
void RightHandSide<dim>::value_list(const std::vector<Point<dim»& points,
                                    std::vector<double>& values,
                                    const unsigned int component) const
{
    Assert(values.size() == points.size(),
           ExcDimensionMismatch(values.size(), points.size()));
    for (unsigned int i = 0; i < points.size(); ++i)
        values[i] = RightHandSide<dim>::value(points[i], component);
}
template <int dim>
class BoundaryValues : public Function<dim> {
public:
    BoundaryValues() : Function<dim>() {}
    virtual double value(const Point<dim>& p,
                         const unsigned int component = 0) const;
    virtual void value_list(const std::vector<Point<dim»& points,
                            std::vector<double>& values,
                            const unsigned int component = 0) const;
};
template <int dim>
double BoundaryValues<dim>::value(const Point<dim>& p,
                                  const unsigned int component) const
{
    (void)component;
    Assert(component == 0, ExcIndexRange(component, 0, 1));
    const double sine_term =
        std::sin(16 * numbers::PI * std::sqrt(p.norm_square()));
    const double weight = std::exp(-5 * p.norm_square()) / std::exp(-5.);
    return sine_term * weight;
}
template <int dim>
void BoundaryValues<dim>::value_list(const std::vector<Point<dim»& points,
                                     std::vector<double>& values,
                                     const unsigned int component) const
{
    Assert(values.size() == points.size(),
           ExcDimensionMismatch(values.size(), points.size()));
    for (unsigned int i = 0; i < points.size(); ++i)
        values[i] = BoundaryValues<dim>::value(points[i], component);
}
class GradientEstimation {
public:
    template <int dim>
    static void estimate(const DoFHandler<dim>& dof,
                         const Vector<double>& solution,
```

```
                         Vector<float>& error_per_cell);
    DeclException2(ExcInvalidVectorLength, int, int,
                   « "Vector has length " « arg1 « ", but should have "
                   « arg2);
    DeclException0(ExcInsufficientDirections);
private:
    template <int dim>
    struct EstimateScratchData {
        EstimateScratchData(const FiniteElement<dim>& fe,
                            const Vector<double>& solution,
                            Vector<float>& error_per_cell);
        EstimateScratchData(const EstimateScratchData& data);
        FEValues<dim> fe_midpoint_value;
        const Vector<double>& solution;
        Vector<float>& error_per_cell;
    };
    struct EstimateCopyData {};
    template <int dim>
    static void estimate_cell(
        const typename DoFHandler<dim>::active_cell_iterator& cell,
        EstimateScratchData<dim>& scratch_data,
        const EstimateCopyData& copy_data);
};
template <int dim>
AdvectionProblem<dim>::AdvectionProblem() : dof_handler(triangulation), fe(1)
{}
template <int dim>
AdvectionProblem<dim>:: AdvectionProblem()
{
    dof_handler.clear();
}
template <int dim>
void AdvectionProblem<dim>::setup_system()
{
    dof_handler.distribute_dofs(fe);
    hanging_node_constraints.clear();
    DoFTools::make_hanging_node_constraints(dof_handler,
                                            hanging_node_constraints);
    hanging_node_constraints.close();
    DynamicSparsityPattern dsp(dof_handler.n_dofs(), dof_handler.n_dofs());
    DoFTools::make_sparsity_pattern(dof_handler, dsp, hanging_node_constraints,
                                    /*keep_constrained_dofs = */ true);
    sparsity_pattern.copy_from(dsp);
    system_matrix.reinit(sparsity_pattern);
    solution.reinit(dof_handler.n_dofs());
    system_rhs.reinit(dof_handler.n_dofs());
}
template <int dim>
void AdvectionProblem<dim>::assemble_system()
{
    WorkStream::run(dof_handler.begin_active(), dof_handler.end(), *this,
                    &AdvectionProblem::local_assemble_system,
                    &AdvectionProblem::copy_local_to_global,
                    AssemblyScratchData(fe), AssemblyCopyData());
    hanging_node_constraints.condense(system_matrix);
    hanging_node_constraints.condense(system_rhs);
}
template <int dim>
AdvectionProblem<dim>::AssemblyScratchData::AssemblyScratchData(
    const FiniteElement<dim>& fe)
    : fe_values(fe, QGauss<dim>(2),
                update_values | update_gradients | update_quadrature_points |
                    update_JxW_values),
      fe_face_values(fe, QGauss<dim - 1>(2),
                     update_values | update_quadrature_points |
                         update_JxW_values | update_normal_vectors)
{}
template <int dim>
AdvectionProblem<dim>::AssemblyScratchData::AssemblyScratchData(
    const AssemblyScratchData& scratch_data)
    : fe_values(scratch_data.fe_values.get_fe(),
                scratch_data.fe_values.get_quadrature(),
                update_values | update_gradients | update_quadrature_points |
                    update_JxW_values),
      fe_face_values(scratch_data.fe_face_values.get_fe(),
                     scratch_data.fe_face_values.get_quadrature(),
                     update_values | update_quadrature_points |
                         update_JxW_values | update_normal_vectors)
{}
template <int dim>
void AdvectionProblem<dim>::local_assemble_system(
    const typename DoFHandler<dim>::active_cell_iterator& cell,
    AssemblyScratchData& scratch_data, AssemblyCopyData& copy_data)
{
    const AdvectionField<dim> advection_field;
    const RightHandSide<dim> right_hand_side;
    const BoundaryValues<dim> boundary_values;
```

```
    const unsigned int dofs_per_cell = fe.dofs_per_cell;
    const unsigned int n_q_points =
        scratch_data.fe_values.get_quadrature().size();
    const unsigned int n_face_q_points =
        scratch_data.fe_face_values.get_quadrature().size();
    copy_data.cell_matrix.reinit(dofs_per_cell, dofs_per_cell);
    copy_data.cell_rhs.reinit(dofs_per_cell);
    copy_data.local_dof_indices.resize(dofs_per_cell);
    std::vector<double> rhs_values(n_q_points);
    std::vector<Tensor<1, dim» advection_directions(n_q_points);
    std::vector<double> face_boundary_values(n_face_q_points);
    std::vector<Tensor<1, dim» face_advection_directions(n_face_q_points);
    scratch_data.fe_values.reinit(cell);
    advection_field.value_list(scratch_data.fe_values.get_quadrature_points(),
                               advection_directions);
    right_hand_side.value_list(scratch_data.fe_values.get_quadrature_points(),
                               rhs_values);
    const double delta = 0.1 * cell->diameter();
    for (unsigned int q_point = 0; q_point < n_q_points; ++q_point)
        for (unsigned int i = 0; i < dofs_per_cell; ++i) {
            for (unsigned int j = 0; j < dofs_per_cell; ++j)
                copy_data.cell_matrix(i, j) +=
                    ((advection_directions[q_point] *
                      scratch_data.fe_values.shape_grad(j, q_point) *
                      (scratch_data.fe_values.shape_value(i, q_point) +
                       delta *
                           (advection_directions[q_point] *
                            scratch_data.fe_values.shape_grad(i, q_point)))) *
                     scratch_data.fe_values.JxW(q_point));
            copy_data.cell_rhs(i) +=
                ((scratch_data.fe_values.shape_value(i, q_point) +
                  delta * (advection_directions[q_point] *
                           scratch_data.fe_values.shape_grad(i, q_point))) *
                 rhs_values[q_point] * scratch_data.fe_values.JxW(q_point));
        }
    for (unsigned int face = 0; face < GeometryInfo<dim>::faces_per_cell;
         ++face)
        if (cell->face(face)->at_boundary()) {
            scratch_data.fe_face_values.reinit(cell, face);
            boundary_values.value_list(
                scratch_data.fe_face_values.get_quadrature_points(),
                face_boundary_values);
            advection_field.value_list(
                scratch_data.fe_face_values.get_quadrature_points(),
                face_advection_directions);
            for (unsigned int q_point = 0; q_point < n_face_q_points; ++q_point)
                if (scratch_data.fe_face_values.normal_vector(q_point) *
                        face_advection_directions[q_point] <
                    0)
                    for (unsigned int i = 0; i < dofs_per_cell; ++i) {
                        for (unsigned int j = 0; j < dofs_per_cell; ++j)
                            copy_data.cell_matrix(i, j) -=
                                (face_advection_directions[q_point] *
                                 scratch_data.fe_face_values.normal_vector(
                                     q_point) *
                                 scratch_data.fe_face_values.shape_value(
                                     i, q_point) *
                                 scratch_data.fe_face_values.shape_value(
                                     j, q_point) *
                                 scratch_data.fe_face_values.JxW(q_point));
                        copy_data.cell_rhs(i) -=
                            (face_advection_directions[q_point] *
                             scratch_data.fe_face_values.normal_vector(
                                 q_point) *
                             face_boundary_values[q_point] *
                             scratch_data.fe_face_values.shape_value(i,
                                                                     q_point) *
                             scratch_data.fe_face_values.JxW(q_point));
                    }
        }
    cell->get_dof_indices(copy_data.local_dof_indices);
}
template <int dim>
void AdvectionProblem<dim>::copy_local_to_global(
    const AssemblyCopyData& copy_data)
{
    for (unsigned int i = 0; i < copy_data.local_dof_indices.size(); ++i) {
        for (unsigned int j = 0; j < copy_data.local_dof_indices.size(); ++j)
            system_matrix.add(copy_data.local_dof_indices[i],
                              copy_data.local_dof_indices[j],
                              copy_data.cell_matrix(i, j));
        system_rhs(copy_data.local_dof_indices[i]) += copy_data.cell_rhs(i);
    }
}
template <int dim>
void AdvectionProblem<dim>::solve()
{
```

```
    Assert(system_matrix.m() == system_matrix.n(), ExcNotQuadratic());
    auto num_rows = system_matrix.m();
    std::vector<double> rhs(num_rows);
    std::copy(system_rhs.begin(), system_rhs.begin() + num_rows, rhs.begin());
    using vec = gko::matrix::Dense<>;
    using mtx = gko::matrix::Csr<>;
    using bicgstab = gko::solver::Bicgstab<>;
    using bj = gko::preconditioner::Jacobi<>;
    using val_array = gko::Array<double>;
    std::shared_ptr<gko::Executor> exec = gko::ReferenceExecutor::create();
    auto b = vec::create(exec, gko::dim<2>(num_rows, 1),
                         val_array::view(exec, num_rows, rhs.data()), 1);
    auto x = vec::create(exec, gko::dim<2>(num_rows, 1));
    auto A = mtx::create(exec, gko::dim<2>(num_rows),
                         system_matrix.n_nonzero_elements());
    mtx::value_type* values = A->get_values();
    mtx::index_type* row_ptr = A->get_row_ptrs();
    mtx::index_type* col_idx = A->get_col_idxs();
    row_ptr[0] = 0;
    for (auto row = 1; row <= num_rows; ++row) {
        row_ptr[row] = row_ptr[row - 1] + system_matrix.get_row_length(row - 1);
    }
    std::vector<mtx::index_type> ptrs(num_rows + 1);
    std::copy(A->get_row_ptrs(), A->get_row_ptrs() + num_rows + 1,
              ptrs.begin());
    for (auto row = 0; row < system_matrix.m(); ++row) {
        for (auto p = system_matrix.begin(row); p != system_matrix.end(row);
             ++p) {
            col_idx[ptrs[row]] = p->column();
            values[ptrs[row]] = p->value();
            ++ptrs[row];
        }
    }
    auto solver_gen =
        bicgstab::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(1000).on(exec),
                gko::stop::ResidualNorm<>::build()
                    .with_reduction_factor(1e-12)
                    .on(exec))
            .with_preconditioner(bj::build().on(exec))
            .on(exec);
    auto solver = solver_gen->generate(gko::give(A));
    solver->apply(gko::lend(b), gko::lend(x));
    std::copy(x->get_values(), x->get_values() + num_rows, solution.begin());
    /*******************************************************
     * deal.ii internal solver. Here for reference.
     SolverControl           solver_control (1000, 1e-12);
     SolverBicgstab<>        bicgstab (solver_control);

     PreconditionJacobi<> preconditioner;
     preconditioner.initialize(system_matrix, 1.0);

     bicgstab.solve (system_matrix, solution, system_rhs,
                     preconditioner);
    *******************************************************/
    hanging_node_constraints.distribute(solution);
}
template <int dim>
void AdvectionProblem<dim>::refine_grid()
{
    Vector<float> estimated_error_per_cell(triangulation.n_active_cells());
    GradientEstimation::estimate(dof_handler, solution,
                                 estimated_error_per_cell);
    GridRefinement::refine_and_coarsen_fixed_number(
        triangulation, estimated_error_per_cell, 0.5, 0.03);
    triangulation.execute_coarsening_and_refinement();
}
template <int dim>
void AdvectionProblem<dim>::output_results(const unsigned int cycle) const
{
    {
        GridOut grid_out;
        std::ofstream output("grid-" + std::to_string(cycle) + ".eps");
        grid_out.write_eps(triangulation, output);
    }
    {
        DataOut<dim> data_out;
        data_out.attach_dof_handler(dof_handler);
        data_out.add_data_vector(solution, "solution");
        data_out.build_patches();
        std::ofstream output("solution-" + std::to_string(cycle) + ".vtk");
        data_out.write_vtk(output);
    }
}
template <int dim>
void AdvectionProblem<dim>::run()
```

```
{
    for (unsigned int cycle = 0; cycle < 6; ++cycle) {
        std::cout « "Cycle " « cycle « ':' « std::endl;
        if (cycle == 0) {
            GridGenerator::hyper_cube(triangulation, -1, 1);
            triangulation.refine_global(4);
        } else {
            refine_grid();
        }
        std::cout « "   Number of active cells:       "
                  « triangulation.n_active_cells() « std::endl;
        setup_system();
        std::cout « "   Number of degrees of freedom: " « dof_handler.n_dofs()
                  « std::endl;
        assemble_system();
        solve();
        output_results(cycle);
    }
}
template <int dim>
GradientEstimation::EstimateScratchData<dim>::EstimateScratchData(
    const FiniteElement<dim>& fe, const Vector<double>& solution,
    Vector<float>& error_per_cell)
    : fe_midpoint_value(fe, QMidpoint<dim>(),
                        update_values | update_quadrature_points),
      solution(solution),
      error_per_cell(error_per_cell)
{}
template <int dim>
GradientEstimation::EstimateScratchData<dim>::EstimateScratchData(
    const EstimateScratchData& scratch_data)
    : fe_midpoint_value(scratch_data.fe_midpoint_value.get_fe(),
                        scratch_data.fe_midpoint_value.get_quadrature(),
                        update_values | update_quadrature_points),
      solution(scratch_data.solution),
      error_per_cell(scratch_data.error_per_cell)
{}
template <int dim>
void GradientEstimation::estimate(const DoFHandler<dim>& dof_handler,
                                  const Vector<double>& solution,
                                  Vector<float>& error_per_cell)
{
    Assert(error_per_cell.size() ==
               dof_handler.get_triangulation().n_active_cells(),
           ExcInvalidVectorLength(
               error_per_cell.size(),
               dof_handler.get_triangulation().n_active_cells()));
    WorkStream::run(dof_handler.begin_active(), dof_handler.end(),
                    &GradientEstimation::template estimate_cell<dim>,
                    std::function<void(const EstimateCopyData&)>(),
                    EstimateScratchData<dim>(dof_handler.get_fe(), solution,
                                             error_per_cell),
                    EstimateCopyData());
}
template <int dim>
void GradientEstimation::estimate_cell(
    const typename DoFHandler<dim>::active_cell_iterator& cell,
    EstimateScratchData<dim>& scratch_data, const EstimateCopyData&)
{
    Tensor<2, dim> Y;
    std::vector<typename DoFHandler<dim>::active_cell_iterator>
        active_neighbors;
    active_neighbors.reserve(GeometryInfo<dim>::faces_per_cell *
                             GeometryInfo<dim>::max_children_per_face);
    scratch_data.fe_midpoint_value.reinit(cell);
    Tensor<1, dim> projected_gradient;
    active_neighbors.clear();
    for (unsigned int face_no = 0; face_no < GeometryInfo<dim>::faces_per_cell;
         ++face_no)
        if (!cell->at_boundary(face_no)) {
            const typename DoFHandler<dim>::face_iterator face =
                cell->face(face_no);
            const typename DoFHandler<dim>::cell_iterator neighbor =
                cell->neighbor(face_no);
            if (neighbor->active())
                active_neighbors.push_back(neighbor);
            else {
                if (dim == 1) {
                    typename DoFHandler<dim>::cell_iterator neighbor_child =
                        neighbor;
                    while (neighbor_child->has_children())
                        neighbor_child =
                            neighbor_child->child(face_no == 0 ? 1 : 0);
                    Assert(
                        neighbor_child->neighbor(face_no == 0 ? 1 : 0) == cell,
                        ExcInternalError());
                    active_neighbors.push_back(neighbor_child);
```

```
                } else
                    for (unsigned int subface_no = 0;
                         subface_no < face->n_children(); ++subface_no)
                        active_neighbors.push_back(
                            cell->neighbor_child_on_subface(face_no,
                                                            subface_no));
            }
        }
    const Point<dim> this_center =
        scratch_data.fe_midpoint_value.quadrature_point(0);
    std::vector<double> this_midpoint_value(1);
    scratch_data.fe_midpoint_value.get_function_values(scratch_data.solution,
                                                       this_midpoint_value);
    std::vector<double> neighbor_midpoint_value(1);
    typename std::vector<typename DoFHandler<dim>::active_cell_iterator>::
        const_iterator neighbor_ptr = active_neighbors.begin();
    for (; neighbor_ptr != active_neighbors.end(); ++neighbor_ptr) {
        const typename DoFHandler<dim>::active_cell_iterator neighbor =
            *neighbor_ptr;
        scratch_data.fe_midpoint_value.reinit(neighbor);
        const Point<dim> neighbor_center =
            scratch_data.fe_midpoint_value.quadrature_point(0);
        scratch_data.fe_midpoint_value.get_function_values(
            scratch_data.solution, neighbor_midpoint_value);
        Tensor<1, dim> y = neighbor_center - this_center;
        const double distance = y.norm();
        y /= distance;
        for (unsigned int i = 0; i < dim; ++i)
            for (unsigned int j = 0; j < dim; ++j) Y[i][j] += y[i] * y[j];
        projected_gradient +=
            (neighbor_midpoint_value[0] - this_midpoint_value[0]) / distance *
            y;
    }
    AssertThrow(determinant(Y) != 0, ExcInsufficientDirections());
    const Tensor<2, dim> Y_inverse = invert(Y);
    Tensor<1, dim> gradient = Y_inverse * projected_gradient;
    scratch_data.error_per_cell(cell->active_cell_index()) =
        (std::pow(cell->diameter(), 1 + 1.0 * dim / 2) *
         std::sqrt(gradient.norm_square()));
}
}  // namespace Step9
int main()
{
    try {
        dealii::MultithreadInfo::set_thread_limit();
        Step9::AdvectionProblem<2> advection_problem_2d;
        advection_problem_2d.run();
    } catch (std::exception& exc) {
        std::cerr « std::endl
                  « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        std::cerr « "Exception on processing: " « std::endl
                  « exc.what() « std::endl
                  « "Aborting!" « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        return 1;
    } catch (...) {
        std::cerr « std::endl
                  « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        std::cerr « "Unknown exception!" « std::endl
                  « "Aborting!" « std::endl
                  « "----------------------------------------------------"
                  « std::endl;
        return 1;
    }
    return 0;
}
```

Chapter 16

The ginkgo-overhead program

The ginkgo overhead measurement example..

Introduction

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <cmath>
#include <iostream>
[[noreturn]] void print_usage_and_exit(const char* name)
{
    std::cerr « "Usage: " « name « " [NUM_ITERS]" « std::endl;
    std::exit(-1);
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    long unsigned num_iters = 1000000;
    if (argc > 2) {
        print_usage_and_exit(argv[0]);
    }
    if (argc == 2) {
        num_iters = std::atol(argv[1]);
        if (num_iters == 0) {
            print_usage_and_exit(argv[0]);
        }
    }
    std::cout « gko::version_info::get() « std::endl;
    auto exec = gko::ReferenceExecutor::create();
    auto cg_factory =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(num_iters).on(
                    exec))
            .on(exec);
    auto A = gko::initialize<mtx>({1.0}, exec);
    auto b = gko::initialize<vec>({std::nan("")}, exec);
    auto x = gko::initialize<vec>({0.0}, exec);
    auto tic = std::chrono::steady_clock::now();
    auto solver = cg_factory->generate(gko::give(A));
    solver->apply(lend(x), lend(b));
    exec->synchronize();
    auto tac = std::chrono::steady_clock::now();
    auto time = std::chrono::duration_cast<std::chrono::nanoseconds>(tac - tic);
    std::cout « "Running " « num_iters
```

```
              « " iterations of the CG solver took a total of "
              « static_cast<double>(time.count()) /
                    static_cast<double>(std::nano::den)
              « " seconds." « std::endl
              « "\tAverage library overhead:     "
              « static_cast<double>(time.count()) /
                    static_cast<double>(num_iters)
              « " [nanoseconds / iteration]" « std::endl;
}
```

Results

This is the expected output:

```
Running 1000000 iterations of the CG solver took a total of 1.60337 seconds.
    Average library overhead:     1603.37 [nanoseconds / iteration]
```

Comments about programming and debugging

The plain program

```
/*****************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <cmath>
#include <iostream>
[[noreturn]] void print_usage_and_exit(const char* name)
{
    std::cerr « "Usage: " « name « " [NUM_ITERS]" « std::endl;
    std::exit(-1);
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    long unsigned num_iters = 1000000;
    if (argc > 2) {
        print_usage_and_exit(argv[0]);
    }
    if (argc == 2) {
        num_iters = std::atol(argv[1]);
        if (num_iters == 0) {
            print_usage_and_exit(argv[0]);
```

```
        }
    }
    std::cout « gko::version_info::get() « std::endl;
    auto exec = gko::ReferenceExecutor::create();
    auto cg_factory =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(num_iters).on(
                    exec))
            .on(exec);
    auto A = gko::initialize<mtx>({1.0}, exec);
    auto b = gko::initialize<vec>({std::nan("")}, exec);
    auto x = gko::initialize<vec>({0.0}, exec);
    auto tic = std::chrono::steady_clock::now();
    auto solver = cg_factory->generate(gko::give(A));
    solver->apply(lend(x), lend(b));
    exec->synchronize();
    auto tac = std::chrono::steady_clock::now();
    auto time = std::chrono::duration_cast<std::chrono::nanoseconds>(tac - tic);
    std::cout « "Running " « num_iters
              « " iterations of the CG solver took a total of "
              « static_cast<double>(time.count()) /
                     static_cast<double>(std::nano::den)
              « " seconds." « std::endl
              « "\tAverage library overhead:     "
              « static_cast<double>(time.count()) /
                     static_cast<double>(num_iters)
              « " [nanoseconds / iteration]" « std::endl;
}
```

Chapter 17

The ginkgo-ranges program

The ranges and accessor example..

Introduction

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <iomanip>
#include <iostream>
```

LU factorization implementation using Ginkgo ranges For simplicity, we only consider square matrices, and no pivoting.

```
template <typename Accessor>
void factorize(const gko::range<Accessor>& A)
```

note: const means that the range (i.e. the data handler) is constant, not that the underlying data is constant!

```
{
    using gko::span;
    assert(A.length(0) == A.length(1));
    for (gko::size_type i = 0; i < A.length(0) - 1; ++i) {
        const auto trail = span{i + 1, A.length(0)};
```

note: neither of the lines below need additional memory to store intermediate arrays, all computation is done at the point of assignment

```
A(trail, i) = A(trail, i) / A(i, i);
```

caveat: operator $*$ is element-wise multiplication, mmul is matrix multiplication

```
        A(trail, trail) = A(trail, trail) - mmul(A(trail, i), A(i, trail));
    }
}
```

a utility function for printing the factorization on screen

```
template <typename Accessor>
void print_lu(const gko::range<Accessor>& A)
{
    std::cout « std::setprecision(2) « std::fixed;
    std::cout « "L = [";
    for (int i = 0; i < A.length(0); ++i) {
        std::cout « "\n  ";
        for (int j = 0; j < A.length(1); ++j) {
            std::cout « (i > j ? A(i, j) : (i == j) * 1.) « " ";
        }
    }
    std::cout « "\n]\n\nU = [";
```

```
    for (int i = 0; i < A.length(0); ++i) {
        std::cout « "\n  ";
        for (int j = 0; j < A.length(1); ++j) {
            std::cout « (i <= j ? A(i, j) : 0.) « " ";
        }
    }
    std::cout « "\n]" « std::endl;
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
```

Create some test data, add some padding just to demonstrate how to use it with ranges. clang-format off

```
ValueType data[] = {
    2.,  4.,  5., -1.0,
    4., 11., 12., -1.0,
    6., 24., 24., -1.0
};
```

clang-format on

Create a 3-by-3 range, with a 2D row-major accessor using data as the underlying storage. Set the stride (a.k.a. "LDA") to 4.

```
auto A =
    gko::range<gko::accessor::row_major<ValueType, 2>>(data, 3u, 3u, 4u);
```

use the LU factorization routine defined above to factorize the matrix

```
factorize(A);
```

print the factorization on screen

```
    print_lu(A);
}
```

Results

This is the expected output:

```
L = [
  1.00 0.00 0.00
  2.00 1.00 0.00
  3.00 4.00 1.00
]
U = [
  2.00 4.00 5.00
  0.00 3.00 2.00
  0.00 0.00 1.00
]
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
```

```
#include <ginkgo/ginkgo.hpp>
#include <iomanip>
#include <iostream>
template <typename Accessor>
void factorize(const gko::range<Accessor>& A)
{
    using gko::span;
    assert(A.length(0) == A.length(1));
    for (gko::size_type i = 0; i < A.length(0) - 1; ++i) {
        const auto trail = span{i + 1, A.length(0)};
        A(trail, i) = A(trail, i) / A(i, i);
        A(trail, trail) = A(trail, trail) - mmul(A(trail, i), A(i, trail));
    }
}
template <typename Accessor>
void print_lu(const gko::range<Accessor>& A)
{
    std::cout « std::setprecision(2) « std::fixed;
    std::cout « "L = [";
    for (int i = 0; i < A.length(0); ++i) {
        std::cout « "\n  ";
        for (int j = 0; j < A.length(1); ++j) {
            std::cout « (i > j ? A(i, j) : (i == j) * 1.) « " ";
        }
    }
    std::cout « "\n]\n\nU = [";
    for (int i = 0; i < A.length(0); ++i) {
        std::cout « "\n  ";
        for (int j = 0; j < A.length(1); ++j) {
            std::cout « (i <= j ? A(i, j) : 0.) « " ";
        }
    }
    std::cout « "\n]" « std::endl;
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    std::cout « gko::version_info::get() « std::endl;
    ValueType data[] = {
        2.,  4.,  5., -1.0,
        4., 11., 12., -1.0,
        6., 24., 24., -1.0
    };
    auto A =
        gko::range<gko::accessor::row_major<ValueType, 2>>(data, 3u, 3u, 4u);
    factorize(A);
    print_lu(A);
}
```

Chapter 18

The heat-equation program

The heat equation example..

This example depends on simple-solver, three-pt-stencil-solver.

Introduction

This example solves a 2D heat conduction equation

$$u : [0, d]^2 \to R$$
$$\partial_t u = \delta u + f$$

with Dirichlet boundary conditions and given initial condition and constant-in-time source function f.

The partial differential equation (PDE) is solved with a finite difference spatial discretization on an equidistant grid: For n grid points, and grid distance $h = 1/n$ we write

$$u_{i,j}\text{'} = \alpha \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} + f_{i,j}$$

We then build an implicit Euler integrator by discretizing with time step τ

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\tau} = \alpha \frac{u_{i-1,j}^{k+1} + u_{i+1,j}^{k+1} - u_{i,j-1}^{k+1} - u_{i,j+1}^{k+1} + 4u_{i,j}^{k+1}}{h^2} + f_{i,j}$$

and solve the resulting linear system for $u_{\cdot}^{k+1}$ using Ginkgo's CG solver preconditioned with an incomplete Cholesky factorization for each time step, occasionally writing the resulting grid values into a video file using OpenCV and a custom color mapping.

The intention of this example is to provide a mini-app showing matrix assembly, vector initialization, solver setup and the use of Ginkgo in a more complex setting.

About the example

The commented program

```
/ *****************************<DESCRIPTION>**********************************
This example solves a 2D heat conduction equation
    u : [0, d]^2 \rightarrow R\\
    \partial_t u = \delta u + f
with Dirichlet boundary conditions and given initial condition and
constant-in-time source function f.
The partial differential equation (PDE) is solved with a finite difference
spatial discretization on an equidistant grid: For `n` grid points,
and grid distance h = 1/n we write
    u_{i,j}' = \alpha (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}
                   - 4 u_{i,j}) / h^2
               + f_{i,j}

We then build an implicit Euler integrator by discretizing with time step \tau

    (u_{i,j}^{k+1} - u_{i,j}^k) / \tau =
    \alpha (u_{i-1,j}^{k+1} - u_{i+1,j}^{k+1}
          + u_{i,j-1}^{k+1} - u_{i,j+1}^{k+1} - 4 u_{i,j}^{k+1}) / h^2
    + f_{i,j}

and solve the resulting linear system for  u_{\cdot}^{k+1} using Ginkgo's CG
solver preconditioned with an incomplete Cholesky factorization for each time
step, occasionally writing the resulting grid values into a video file using
OpenCV and a custom color mapping.
The intention of this example is to provide a mini-app showing matrix assembly,
vector initialization, solver setup and the use of Ginkgo in a more complex
setting.
*****************************<DESCRIPTION>********************************* /
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <fstream>
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
```

This function implements a simple Ginkgo-themed clamped color mapping for values in the range [0,5].

```
void set_val(unsigned char* data, double value)
{
```

RGB values for the 6 colors used for values 0, 1, ..., 5 We will interpolate linearly between these values.

```
double col_r[] = {255, 221, 129, 201, 249, 255};
double col_g[] = {255, 220, 130, 161, 158, 204};
double col_b[] = {255, 220, 133, 93, 24, 8};
value = std::max(0.0, value);
auto i = std::max(0, std::min(4, int(value)));
auto d = std::max(0.0, std::min(1.0, value - i));
```

OpenCV uses BGR instead of RGB by default, revert indices

```
    data[2] = static_cast<unsigned char>(col_r[i + 1] * d + col_r[i] * (1 - d));
    data[1] = static_cast<unsigned char>(col_g[i + 1] * d + col_g[i] * (1 - d));
    data[0] = static_cast<unsigned char>(col_b[i + 1] * d + col_b[i] * (1 - d));
}
```

Initialize video output with given dimension and FPS (frames per seconds)

```
std::pair<cv::VideoWriter, cv::Mat> build_output(int n, double fps)
{
    cv::Size videosize{n, n};
    auto output =
        std::make_pair(cv::VideoWriter{}, cv::Mat{videosize, CV_8UC3});
    auto fourcc = cv::VideoWriter::fourcc('a', 'v', 'c', '1');
    output.first.open("heat.mp4", fourcc, fps, videosize);
    return output;
}
```

Write the current frame to video output using the above color mapping

```
void output_timestep(std::pair<cv::VideoWriter, cv::Mat>& output, int n,
                     const double* data)
{
    for (int i = 0; i < n; i++) {
        auto row = output.second.ptr(i);
        for (int j = 0; j < n; j++) {
            set_val(&row[3 * j], data[i * n + j]);
        }
    }
    output.first.write(output.second);
}
```

```
int main(int argc, char* argv[])
{
    using mtx = gko::matrix::Csr<>;
    using vec = gko::matrix::Dense<>;
```

Problem parameters: simulation length

```
auto t0 = 5.0;
```

diffusion factor

```
auto diffusion = 0.0005;
```

scaling factor for heat source

```
auto source_scale = 2.5;
```

Simulation parameters: inner grid points per discretization direction

```
auto n = 256;
```

number of simulation steps per second

```
auto steps_per_sec = 500;
```

number of video frames per second

```
auto fps = 25;
```

number of grid points

```
auto n2 = n * n;
```

grid point distance (ignoring boundary points)

```
auto h = 1.0 / (n + 1);
auto h2 = h * h;
```

time step size for the simulation

```
auto tau = 1.0 / steps_per_sec;
```

create a CUDA executor with an associated OpenMP host executor

```
auto exec = gko::CudaExecutor::create(0, gko::OmpExecutor::create());
```

load heat source and initial state vectors

```
std::ifstream initial_stream("data/gko_logo_2d.mtx");
std::ifstream source_stream("data/gko_text_2d.mtx");
auto source = gko::read<vec>(source_stream, exec);
auto in_vector = gko::read<vec>(initial_stream, exec);
```

create output vector with initial guess for

```
auto out_vector = in_vector->clone();
```

create scalar for source update

```
auto tau_source_scalar = gko::initialize<vec>({source_scale * tau}, exec);
```

create stencil matrix as shared_ptr for solver

```
auto stencil_matrix = gko::share(mtx::create(exec));
```

assemble matrix

```
gko::matrix_data<> mtx_data{gko::dim<2>(n2, n2)};
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        auto c = i * n + j;
        auto c_val = diffusion * tau * 4.0 / h2 + 1.0;
        auto off_val = -diffusion * tau / h2;
```

for each grid point: insert 5 stencil points with Dirichlet boundary conditions, i.e. with zero boundary value

```
        if (i > 0) {
            mtx_data.nonzeros.emplace_back(c, c - n, off_val);
        }
        if (j > 0) {
            mtx_data.nonzeros.emplace_back(c, c - 1, off_val);
        }
        mtx_data.nonzeros.emplace_back(c, c, c_val);
        if (j < n - 1) {
            mtx_data.nonzeros.emplace_back(c, c + 1, off_val);
        }
        if (i < n - 1) {
            mtx_data.nonzeros.emplace_back(c, c + n, off_val);
```

```
        }
    }
}
stencil_matrix->read(mtx_data);
```

prepare video output
```
auto output = build_output(n, fps);
```

build CG solver on stencil with incomplete Cholesky preconditioner stopping at 1e-10 relative accuracy
```
auto solver =
    gko::solver::Cg<>::build()
        .with_preconditioner(gko::preconditioner::Ic<>::build().on(exec))
        .with_criteria(gko::stop::RelativeResidualNorm<>::build()
                           .with_tolerance(1e-10)
                           .on(exec))
        .on(exec)
        ->generate(stencil_matrix);
```

time stamp of the last output frame (initialized to a sentinel value)
```
double last_t = -t0;
```

execute implicit Euler method: for each timestep, solve stencil system
```
for (double t = 0; t < t0; t += tau) {
```

if enough time has passed, output the next video frame
```
if (t - last_t > 1.0 / fps) {
    last_t = t;
    std::cout « t « std::endl;
    output_timestep(
        output, n,
        gko::make_temporary_clone(exec->get_master(), in_vector.get())
            ->get_const_values());
}
```

add heat source contribution
```
in_vector->add_scaled(gko::lend(tau_source_scalar), gko::lend(source));
```

execute Euler step
```
solver->apply(gko::lend(in_vector), gko::lend(out_vector));
```

swap input and output
```
        std::swap(in_vector, out_vector);
    }
}
```

Results

The program will generate a video file named heat.mp4 and output the timestamp of each generated frame.

Comments about programming and debugging

The plain program

```
3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
/*****************************<DESCRIPTION>**********************************
This example solves a 2D heat conduction equation

    u : [0, d]^2 \rightarrow R\\
    \partial_t u = \delta u + f

with Dirichlet boundary conditions and given initial condition and
constant-in-time source function f.

The partial differential equation (PDE) is solved with a finite difference
spatial discretization on an equidistant grid: For `n` grid points,
and grid distance @f$h = 1/n@f$ we write

    u_{i,j}' = \alpha (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}
                   - 4 u_{i,j}) / h^2
               + f_{i,j}

We then build an implicit Euler integrator by discretizing with time step @f$\tau@f$

    (u_{i,j}^{k+1} - u_{i,j}^k) / \tau =
    \alpha (u_{i-1,j}^{k+1} - u_{i+1,j}^{k+1}
          + u_{i,j-1}^{k+1} - u_{i,j+1}^{k+1} - 4 u_{i,j}^{k+1}) / h^2
    + f_{i,j}

and solve the resulting linear system for @f$ u_{\cdot}^{k+1}@f$ using Ginkgo's CG
solver preconditioned with an incomplete Cholesky factorization for each time
step, occasionally writing the resulting grid values into a video file using
OpenCV and a custom color mapping.

The intention of this example is to provide a mini-app showing matrix assembly,
vector initialization, solver setup and the use of Ginkgo in a more complex
setting.
*****************************<DESCRIPTION>**********************************/
#include <ginkgo/ginkgo.hpp>
#include <chrono>
#include <fstream>
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
void set_val(unsigned char* data, double value)
{
    double col_r[] = {255, 221, 129, 201, 249, 255};
    double col_g[] = {255, 220, 130, 161, 158, 204};
    double col_b[] = {255, 220, 133, 93, 24, 8};
    value = std::max(0.0, value);
    auto i = std::max(0, std::min(4, int(value)));
    auto d = std::max(0.0, std::min(1.0, value - i));
    data[2] = static_cast<unsigned char>(col_r[i + 1] * d + col_r[i] * (1 - d));
    data[1] = static_cast<unsigned char>(col_g[i + 1] * d + col_g[i] * (1 - d));
    data[0] = static_cast<unsigned char>(col_b[i + 1] * d + col_b[i] * (1 - d));
}
std::pair<cv::VideoWriter, cv::Mat> build_output(int n, double fps)
{
    cv::Size videosize{n, n};
    auto output =
        std::make_pair(cv::VideoWriter{}, cv::Mat{videosize, CV_8UC3});
    auto fourcc = cv::VideoWriter::fourcc('a', 'v', 'c', '1');
    output.first.open("heat.mp4", fourcc, fps, videosize);
    return output;
}
void output_timestep(std::pair<cv::VideoWriter, cv::Mat>& output, int n,
                     const double* data)
{
    for (int i = 0; i < n; i++) {
        auto row = output.second.ptr(i);
        for (int j = 0; j < n; j++) {
            set_val(&row[3 * j], data[i * n + j]);
        }
    }
    output.first.write(output.second);
}
```

```
int main(int argc, char* argv[])
{
    using mtx = gko::matrix::Csr<>;
    using vec = gko::matrix::Dense<>;
    auto t0 = 5.0;
    auto diffusion = 0.0005;
    auto source_scale = 2.5;
    auto n = 256;
    auto steps_per_sec = 500;
    auto fps = 25;
    auto n2 = n * n;
    auto h = 1.0 / (n + 1);
    auto h2 = h * h;
    auto tau = 1.0 / steps_per_sec;
    auto exec = gko::CudaExecutor::create(0, gko::OmpExecutor::create());
    std::ifstream initial_stream("data/gko_logo_2d.mtx");
    std::ifstream source_stream("data/gko_text_2d.mtx");
    auto source = gko::read<vec>(source_stream, exec);
    auto in_vector = gko::read<vec>(initial_stream, exec);
    auto out_vector = in_vector->clone();
    auto tau_source_scalar = gko::initialize<vec>({source_scale * tau}, exec);
    auto stencil_matrix = gko::share(mtx::create(exec));
    gko::matrix_data<> mtx_data{gko::dim<2>(n2, n2)};
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            auto c = i * n + j;
            auto c_val = diffusion * tau * 4.0 / h2 + 1.0;
            auto off_val = -diffusion * tau / h2;
            if (i > 0) {
                mtx_data.nonzeros.emplace_back(c, c - n, off_val);
            }
            if (j > 0) {
                mtx_data.nonzeros.emplace_back(c, c - 1, off_val);
            }
            mtx_data.nonzeros.emplace_back(c, c, c_val);
            if (j < n - 1) {
                mtx_data.nonzeros.emplace_back(c, c + 1, off_val);
            }
            if (i < n - 1) {
                mtx_data.nonzeros.emplace_back(c, c + n, off_val);
            }
        }
    }
    stencil_matrix->read(mtx_data);
    auto output = build_output(n, fps);
    auto solver =
        gko::solver::Cg<>::build()
            .with_preconditioner(gko::preconditioner::Ic<>::build().on(exec))
            .with_criteria(gko::stop::RelativeResidualNorm<>::build()
                               .with_tolerance(1e-10)
                               .on(exec))
            .on(exec)
            ->generate(stencil_matrix);
    double last_t = -t0;
    for (double t = 0; t < t0; t += tau) {
        if (t - last_t > 1.0 / fps) {
            last_t = t;
            std::cout « t « std::endl;
            output_timestep(
                output, n,
                gko::make_temporary_clone(exec->get_master(), in_vector.get())
                    ->get_const_values());
        }
        in_vector->add_scaled(gko::lend(tau_source_scalar), gko::lend(source));
        solver->apply(gko::lend(in_vector), gko::lend(out_vector));
        std::swap(in_vector, out_vector);
    }
}
```

Chapter 19

The ilu-preconditioned-solver program

The ILU-preconditioned solver example..

This example depends on simple-solver.

Introduction

About the example

This example shows how to use incomplete factors generated via the ParILU algorithm to generate an incomplete factorization (ILU) preconditioner, how to specify the sparse triangular solves in the ILU preconditioner application, and how to generate an ILU-preconditioned solver and apply it to a specific problem.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using gmres = gko::solver::Gmres<ValueType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
```

```
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = gko::share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
```

Generate incomplete factors using ParILU

```
auto par_ilu_fact =
    gko::factorization::ParIlu<ValueType, IndexType>::build().on(exec);
```

Generate concrete factorization for input matrix

```
auto par_ilu = par_ilu_fact->generate(A);
```

Generate an ILU preconditioner factory by setting lower and upper triangular solver - in this case the exact triangular solves

```
auto ilu_pre_factory =
    gko::preconditioner::Ilu<gko::solver::LowerTrs<ValueType, IndexType>,
                             gko::solver::UpperTrs<ValueType, IndexType>,
                             false>::build()
        .on(exec);
```

Use incomplete factors to generate ILU preconditioner

```
auto ilu_preconditioner = ilu_pre_factory->generate(gko::share(par_ilu));
```

Use preconditioner inside GMRES solver factory Generating a solver factory tied to a specific preconditioner makes sense if there are several very similar systems to solve, and the same solver+preconditioner combination is expected to be effective.

```
const RealValueType reduction_factor{1e-7};
auto ilu_gmres_factory =
    gmres::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1000u).on(exec),
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec))
        .with_generated_preconditioner(gko::share(ilu_preconditioner))
        .on(exec);
```

Generate preconditioned solver for a specific target system

```
auto ilu_gmres = ilu_gmres_factory->generate(A);
```

Solve system

```
ilu_gmres->apply(gko::lend(b), gko::lend(x));
```

Print solution

```
std::cout « "Solution (x):\n";
write(std::cout, gko::lend(x));
```

Calculate residual

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one), gko::lend(b));
    b->compute_norm2(gko::lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, gko::lend(res));
}
```

Results

This is the expected output:

```
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1.46249e-08
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
```

```
    using gmres = gko::solver::Gmres<ValueType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = gko::share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    auto par_ilu_fact =
        gko::factorization::ParIlu<ValueType, IndexType>::build().on(exec);
    auto par_ilu = par_ilu_fact->generate(A);
    auto ilu_pre_factory =
        gko::preconditioner::Ilu<gko::solver::LowerTrs<ValueType, IndexType>,
                                 gko::solver::UpperTrs<ValueType, IndexType>,
                                 false>::build()
            .on(exec);
    auto ilu_preconditioner = ilu_pre_factory->generate(gko::share(par_ilu));
    const RealValueType reduction_factor{1e-7};
    auto ilu_gmres_factory =
        gmres::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(1000u).on(exec),
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec))
            .with_generated_preconditioner(gko::share(ilu_preconditioner))
            .on(exec);
    auto ilu_gmres = ilu_gmres_factory->generate(A);
    ilu_gmres->apply(gko::lend(b), gko::lend(x));
    std::cout « "Solution (x):\n";
    write(std::cout, gko::lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one), gko::lend(b));
    b->compute_norm2(gko::lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, gko::lend(res));
}
```

Chapter 20

The inverse-iteration program

The inverse iteration example..

This example depends on simple-solver, .

Introduction

This example shows how components available in Ginkgo can be used to implement higher-level numerical methods. The method used here will be the shifted inverse iteration method for eigenvalue computation which find the eigenvalue and eigenvector of A closest to z, for some scalar z. The method requires repeatedly solving the shifted linear system (A - zI)x = b, as well as performing matrix-vector products with the matrix `A`. Here is the complete pseudocode of the method:

```
x_0 = initial guess
for i = 0 .. max_iterations:
    solve (A - zI) y_i = x_i for y_i+1
    x_(i+1) = y_i / || y_i ||      # compute next eigenvector approximation
    g_(i+1) = x_(i+1)^* A x_(i+1)  # approximate eigenvalue (Rayleigh quotient)
    if ||A x_(i+1) - g_(i+1)x_(i+1)|| < tol * g_(i+1):  # check convergence
        break
```

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <cmath>
#include <complex>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using precision = std::complex<double>;
using real_precision = gko::remove_complex<precision>;
using vec = gko::matrix::Dense<precision>;
using real_vec = gko::matrix::Dense<real_precision>;
using mtx = gko::matrix::Csr<precision>;
using solver_type = gko::solver::Bicgstab<precision>;
using std::abs;
using std::sqrt;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
std::cout « std::scientific « std::setprecision(8) « std::showpos;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
auto this_exec = exec->get_master();
```

linear system solver parameters

```
auto system_max_iterations = 100u;
auto system_residual_goal = real_precision{1e-16};
```

eigensolver parameters

```
auto max_iterations = 20u;
auto residual_goal = real_precision{1e-8};
auto z = precision{20.0, 2.0};
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Generate shifted matrix A - zI

- we avoid duplicating memory by not storing both A and A - zI, but compute A - zI on the fly by using Ginkgo's utilities for creating linear combinations of operators

```
auto one = share(gko::initialize<vec>({precision{1.0}}, exec));
auto neg_one = share(gko::initialize<vec>({-precision{1.0}}, exec));
auto neg_z = gko::initialize<vec>({-z}, exec);
auto system_matrix = share(gko::Combination<precision>::create(
    one, A, gko::initialize<vec>({-z}, exec),
    gko::matrix::Identity<precision>::create(exec, A->get_size()[0])));
```

Generate solver operator (A - zI)^-1

```
auto solver =
    solver_type::build()
        .with_criteria(gko::stop::Iteration::build()
                           .with_max_iters(system_max_iterations)
                           .on(exec),
                       gko::stop::ResidualNorm<precision>::build()
                           .with_reduction_factor(system_residual_goal)
                           .on(exec))
        .on(exec)
        ->generate(system_matrix);
```

inverse iterations

start with guess [1, 1, ..., 1]

```
auto x = [&] {
    auto work = vec::create(this_exec, gko::dim<2>{A->get_size()[0], 1});
    const auto n = work->get_size()[0];
```

```
    for (int i = 0; i < n; ++i) {
        work->get_values()[i] = precision{1.0} / sqrt(n);
    }
    return clone(exec, work);
}();
auto y = clone(x);
auto tmp = clone(x);
auto norm = gko::initialize<real_vec>({1.0}, exec);
auto inv_norm = clone(this_exec, one);
auto g = clone(one);
for (auto i = 0u; i < max_iterations; ++i) {
    std::cout « "{ ";
```

(A - zI)y = x

```
solver->apply(lend(x), lend(y));
system_matrix->apply(lend(one), lend(y), lend(neg_one), lend(x));
x->compute_norm2(lend(norm));
std::cout « "\"system_residual\": "
          « clone(this_exec, norm)->get_values()[0] « ", ";
x->copy_from(lend(y));
```

x = y / || y ||

```
x->compute_norm2(lend(norm));
inv_norm->get_values()[0] =
    real_precision{1.0} / clone(this_exec, norm)->get_values()[0];
x->scale(lend(clone(exec, inv_norm)));
```

$g = x^{\wedge} * A\ x$

```
A->apply(lend(x), lend(tmp));
x->compute_dot(lend(tmp), lend(g));
auto g_val = clone(this_exec, g)->get_values()[0];
std::cout « "\"eigenvalue\": " « g_val « ", ";
```

$||Ax - gx|| < tol * g$

```
        auto v = gko::initialize<vec>({-g_val}, exec);
        tmp->add_scaled(lend(v), lend(x));
        tmp->compute_norm2(lend(norm));
        auto res_val = clone(exec->get_master(), norm)->get_values()[0];
        std::cout « "\"residual\": " « res_val / g_val « " }," « std::endl;
        if (abs(res_val) < residual_goal * abs(g_val)) {
            break;
        }
    }
}
```

Results

This is the expected output:

```
{ "system_residual": +1.61736920e-14, "eigenvalue": (+2.03741410e+01,-1.17744356e-16), "residual":
       (+2.92231055e-01,+1.68883476e-18) },
{ "system_residual": +4.98014795e-15, "eigenvalue": (+1.94878474e+01,+1.25948378e-15), "residual":
       (+7.94370276e-02,-5.13395071e-18) },
{ "system_residual": +3.39296916e-15, "eigenvalue": (+1.93282121e+01,-1.19329332e-15), "residual":
       (+4.11149623e-02,+2.53837290e-18) },
{ "system_residual": +3.35953656e-15, "eigenvalue": (+1.92638912e+01,+3.28657016e-16), "residual":
       (+2.34717040e-02,-4.00445585e-19) },
{ "system_residual": +2.91474009e-15, "eigenvalue": (+1.92409166e+01,+3.65597737e-16), "residual":
       (+1.34709547e-02,-2.55962367e-19) },
{ "system_residual": +3.09863953e-15, "eigenvalue": (+1.92331106e+01,-1.07919176e-15), "residual":
       (+7.72060707e-03,+4.33212063e-19) },
{ "system_residual": +2.31198069e-15, "eigenvalue": (+1.92305014e+01,-2.89755360e-16), "residual":
       (+4.42106625e-03,+6.66143651e-20) },
{ "system_residual": +3.02771202e-15, "eigenvalue": (+1.92296339e+01,+8.04259901e-16), "residual":
       (+2.53081312e-03,-1.05848687e-19) },
{ "system_residual": +2.02954523e-15, "eigenvalue": (+1.92293461e+01,+7.81834016e-16), "residual":
       (+1.44862114e-03,-5.88985854e-20) },
{ "system_residual": +2.31762332e-15, "eigenvalue": (+1.92292506e+01,-1.11718775e-16), "residual":
       (+8.29183451e-04,+4.81741912e-21) },
{ "system_residual": +8.12541038e-15, "eigenvalue": (+1.92292190e+01,-6.55606254e-16), "residual":
       (+4.74636702e-04,+1.61823936e-20) },
{ "system_residual": +2.77259926e-15, "eigenvalue": (+1.92292085e+01,+4.30588140e-16), "residual":
       (+2.71701077e-04,-6.08403935e-21) },
{ "system_residual": +8.87888675e-14, "eigenvalue": (+1.92292051e+01,+9.67936313e-18), "residual":
       (+1.55539937e-04,-7.82937998e-23) },
{ "system_residual": +2.85077117e-15, "eigenvalue": (+1.92292039e+01,-4.52923128e-16), "residual":
       (+8.90457139e-05,+2.09737561e-21) },
```

```
{ "system_residual": +6.46865302e-14, "eigenvalue": (+1.92292035e+01,+1.58710681e-17), "residual":
      (+5.09805252e-05,-4.20774259e-23) },
{ "system_residual": +4.18913713e-15, "eigenvalue": (+1.92292034e+01,+1.06839590e-15), "residual":
      (+2.91887365e-05,-1.62175862e-21) },
{ "system_residual": +1.06421578e-11, "eigenvalue": (+1.92292034e+01,+3.26089685e-17), "residual":
      (+1.67126561e-05,-2.83413965e-23) },
{ "system_residual": +2.97434420e-13, "eigenvalue": (+1.92292034e+01,-7.85427712e-16), "residual":
      (+9.56961199e-06,+3.90876227e-22) },
{ "system_residual": +1.63230281e-11, "eigenvalue": (+1.92292033e+01,+3.69307000e-16), "residual":
      (+5.47975753e-06,-1.05241636e-22) },
{ "system_residual": +6.14939758e-14, "eigenvalue": (+1.92292033e+01,+1.36057865e-15), "residual":
      (+3.13794996e-06,-2.22028320e-22) },
```


Comments about programming and debugging

The plain program


```
#include <ginkgo/ginkgo.hpp>
#include <cmath>
#include <complex>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using precision = std::complex<double>;
    using real_precision = gko::remove_complex<precision>;
    using vec = gko::matrix::Dense<precision>;
    using real_vec = gko::matrix::Dense<real_precision>;
    using mtx = gko::matrix::Csr<precision>;
    using solver_type = gko::solver::Bicgstab<precision>;
    using std::abs;
    using std::sqrt;
    std::cout « gko::version_info::get() « std::endl;
    std::cout « std::scientific « std::setprecision(8) « std::showpos;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
```

```
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto this_exec = exec->get_master();
    auto system_max_iterations = 100u;
    auto system_residual_goal = real_precision{1e-16};
    auto max_iterations = 20u;
    auto residual_goal = real_precision{1e-8};
    auto z = precision{20.0, 2.0};
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto one = share(gko::initialize<vec>({precision{1.0}}, exec));
    auto neg_one = share(gko::initialize<vec>({-precision{1.0}}, exec));
    auto neg_z = gko::initialize<vec>({-z}, exec);
    auto system_matrix = share(gko::Combination<precision>::create(
        one, A, gko::initialize<vec>({-z}, exec),
        gko::matrix::Identity<precision>::create(exec, A->get_size()[0])));
    auto solver =
        solver_type::build()
            .with_criteria(gko::stop::Iteration::build()
                               .with_max_iters(system_max_iterations)
                               .on(exec),
                           gko::stop::ResidualNorm<precision>::build()
                               .with_reduction_factor(system_residual_goal)
                               .on(exec))
            .on(exec)
            ->generate(system_matrix);
    auto x = [&] {
        auto work = vec::create(this_exec, gko::dim<2>{A->get_size()[0], 1});
        const auto n = work->get_size()[0];
        for (int i = 0; i < n; ++i) {
            work->get_values()[i] = precision{1.0} / sqrt(n);
        }
        return clone(exec, work);
    }();
    auto y = clone(x);
    auto tmp = clone(x);
    auto norm = gko::initialize<real_vec>({1.0}, exec);
    auto inv_norm = clone(this_exec, one);
    auto g = clone(one);
    for (auto i = 0u; i < max_iterations; ++i) {
        std::cout « "{ ";
        solver->apply(lend(x), lend(y));
        system_matrix->apply(lend(one), lend(y), lend(neg_one), lend(x));
        x->compute_norm2(lend(norm));
        std::cout « "\"system_residual\": "
                  « clone(this_exec, norm)->get_values()[0] « ", ";
        x->copy_from(lend(y));
        x->compute_norm2(lend(norm));
        inv_norm->get_values()[0] =
            real_precision{1.0} / clone(this_exec, norm)->get_values()[0];
        x->scale(lend(clone(exec, inv_norm)));
        A->apply(lend(x), lend(tmp));
        x->compute_dot(lend(tmp), lend(g));
        auto g_val = clone(this_exec, g)->get_values()[0];
        std::cout « "\"eigenvalue\": " « g_val « ", ";
        auto v = gko::initialize<vec>({-g_val}, exec);
        tmp->add_scaled(lend(v), lend(x));
        tmp->compute_norm2(lend(norm));
        auto res_val = clone(exec->get_master(), norm)->get_values()[0];
        std::cout « "\"residual\": " « res_val / g_val « " }," « std::endl;
        if (abs(res_val) < residual_goal * abs(g_val)) {
            break;
        }
    }
}
```

Chapter 21

The ir-ilu-preconditioned-solver program

The IR-ILU preconditioned solver example..

This example depends on ilu-preconditioned-solver, iterative-refinement.

Introduction

About the example

This example shows how to combine iterative refinement with the adaptive precision block-Jacobi preconditioner in order to approximately solve the triangular systems occurring in ILU preconditioning. Using an adaptive precision block-Jacobi preconditioner matrix as inner solver for the iterative refinement method is equivalent to doing adaptive precision block-Jacobi relaxation in the triangular solves. This example roughly approximates the triangular solves with five adaptive precision block-Jacobi sweeps with a maximum block size of 16.

This example is motivated by "Multiprecision block-Jacobi for Iterative Triangular Solves" (Göbel, Anzt, Cojean, Flegar, Quintana-Ortí, Euro-Par 2020). The theory and a detailed analysis can be found there.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using gmres = gko::solver::Gmres<ValueType>;
using ir = gko::solver::Ir<ValueType>;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
```

Figure out where to run the code
```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
const unsigned int sweeps = argc == 3 ? std::atoi(argv[2]) : 5u;
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation
```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data
```
auto A = gko::share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create RHS and initial guess as 1
```
gko::size_type num_rows = A->get_size()[0];
auto host_x = vec::create(exec->get_master(), gko::dim<2>(num_rows, 1));
for (gko::size_type i = 0; i < num_rows; i++) {
    host_x->at(i, 0) = 1.;
}
auto x = gko::clone(exec, host_x);
auto b = gko::clone(exec, host_x);
auto clone_x = gko::clone(exec, x);
```

Generate incomplete factors using ParILU
```
auto par_ilu_fact =
    gko::factorization::ParIlu<ValueType, IndexType>::build().on(exec);
```

Generate concrete factorization for input matrix
```
auto par_ilu = par_ilu_fact->generate(A);
```

Generate an iterative refinement factory to be used as a triangular solver in the preconditioner application. The generated method is equivalent to doing five block-Jacobi sweeps with a maximum block size of 16.
```
auto bj_factory =
    bj::build()
        .with_max_block_size(16u)
        .with_storage_optimization(gko::precision_reduction::autodetect())
        .on(exec);
auto trisolve_factory =
    ir::build()
        .with_solver(share(bj_factory))
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(sweeps).on(exec))
        .on(exec);
```

Generate an ILU preconditioner factory by setting lower and upper triangular solver - in this case the previously defined iterative refinement method.
```
auto ilu_pre_factory =
    gko::preconditioner::Ilu<ir, ir>::build()
        .with_l_solver_factory(gko::clone(trisolve_factory))
        .with_u_solver_factory(gko::clone(trisolve_factory))
        .on(exec);
```

Use incomplete factors to generate ILU preconditioner
```
auto ilu_preconditioner = ilu_pre_factory->generate(gko::share(par_ilu));
```

Create stopping criteria for Gmres
```
const RealValueType reduction_factor{1e-12};
auto iter_stop =
```

```
    gko::stop::Iteration::build().with_max_iters(1000u).on(exec);
auto tol_stop = gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec);
std::shared_ptr<const gko::log::Convergence<ValueType>> logger =
    gko::log::Convergence<ValueType>::create(exec);
iter_stop->add_logger(logger);
tol_stop->add_logger(logger);
```

Use preconditioner inside GMRES solver factory Generating a solver factory tied to a specific preconditioner makes sense if there are several very similar systems to solve, and the same solver+preconditioner combination is expected to be effective.

```
auto ilu_gmres_factory =
    gmres::build()
        .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
        .with_generated_preconditioner(gko::share(ilu_preconditioner))
        .on(exec);
```

Generate preconditioned solver for a specific target system

```
auto ilu_gmres = ilu_gmres_factory->generate(A);
```

Warmup run

```
ilu_gmres->apply(lend(b), lend(x));
```

Solve system 100 times and take the average time.

```
std::chrono::nanoseconds time(0);
for (int i = 0; i < 100; i++) {
    x->copy_from(lend(clone_x));
    auto tic = std::chrono::high_resolution_clock::now();
    ilu_gmres->apply(lend(b), lend(x));
    auto toc = std::chrono::high_resolution_clock::now();
    time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
}
std::cout << "Using " << sweeps << " block-Jacobi sweeps.\n";
```

Print solution

```
std::cout << "Solution (x):\n";
write(std::cout, gko::lend(x));
```

Calculate residual

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one), gko::lend(b));
    b->compute_norm2(gko::lend(res));
    std::cout << "GMRES iteration count:     " << logger->get_num_iterations()
              << "\n";
    std::cout << "GMRES execution time [ms]: "
              << static_cast<double>(time.count()) / 100000000.0 << "\n";
    std::cout << "Residual norm sqrt(r^T r):\n";
    write(std::cout, gko::lend(res));
}
```

Results

This is the expected output:

```
Using 5 block-Jacobi sweeps.
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
```

```
0.0150714
0.0107016
0.0121141
0.0123025
GMRES iteration count:     8
GMRES execution time [ms]: 0.377673
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1.65303e-12
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using gmres = gko::solver::Gmres<ValueType>;
    using ir = gko::solver::Ir<ValueType>;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    const unsigned int sweeps = argc == 3 ? std::atoi(argv[2]) : 5u;
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
```

```
                                                    true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = gko::share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    gko::size_type num_rows = A->get_size()[0];
    auto host_x = vec::create(exec->get_master(), gko::dim<2>(num_rows, 1));
    for (gko::size_type i = 0; i < num_rows; i++) {
        host_x->at(i, 0) = 1.;
    }
    auto x = gko::clone(exec, host_x);
    auto b = gko::clone(exec, host_x);
    auto clone_x = gko::clone(exec, x);
    auto par_ilu_fact =
        gko::factorization::ParIlu<ValueType, IndexType>::build().on(exec);
    auto par_ilu = par_ilu_fact->generate(A);
    auto bj_factory =
        bj::build()
            .with_max_block_size(16u)
            .with_storage_optimization(gko::precision_reduction::autodetect())
            .on(exec);
    auto trisolve_factory =
        ir::build()
            .with_solver(share(bj_factory))
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(sweeps).on(exec))
            .on(exec);
    auto ilu_pre_factory =
        gko::preconditioner::Ilu<ir, ir>::build()
            .with_l_solver_factory(gko::clone(trisolve_factory))
            .with_u_solver_factory(gko::clone(trisolve_factory))
            .on(exec);
    auto ilu_preconditioner = ilu_pre_factory->generate(gko::share(par_ilu));
    const RealValueType reduction_factor{1e-12};
    auto iter_stop =
        gko::stop::Iteration::build().with_max_iters(1000u).on(exec);
    auto tol_stop = gko::stop::ResidualNorm<ValueType>::build()
                        .with_reduction_factor(reduction_factor)
                        .on(exec);
    std::shared_ptr<const gko::log::Convergence<ValueType» logger =
        gko::log::Convergence<ValueType>::create(exec);
    iter_stop->add_logger(logger);
    tol_stop->add_logger(logger);
    auto ilu_gmres_factory =
        gmres::build()
            .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
            .with_generated_preconditioner(gko::share(ilu_preconditioner))
            .on(exec);
    auto ilu_gmres = ilu_gmres_factory->generate(A);
    ilu_gmres->apply(lend(b), lend(x));
    std::chrono::nanoseconds time(0);
    for (int i = 0; i < 100; i++) {
        x->copy_from(lend(clone_x));
        auto tic = std::chrono::high_resolution_clock::now();
        ilu_gmres->apply(lend(b), lend(x));
        auto toc = std::chrono::high_resolution_clock::now();
        time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
    }
    std::cout « "Using " « sweeps « " block-Jacobi sweeps.\n";
    std::cout « "Solution (x):\n";
    write(std::cout, gko::lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one), gko::lend(b));
    b->compute_norm2(gko::lend(res));
    std::cout « "GMRES iteration count:     " « logger->get_num_iterations()
              « "\n";
    std::cout « "GMRES execution time [ms]: "
              « static_cast<double>(time.count()) / 100000000.0 « "\n";
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, gko::lend(res));
}
```

Chapter 22

The iterative-refinement program

The iterative refinement solver example..

This example depends on simple-solver.

This example shows how to use the iterative refinement solver.

In this example, we first read in a matrix from file, then generate a right-hand side and an initial guess. An inaccurate CG solver is used as the inner solver to an iterative refinement (IR) method which solves a linear system. The example features the iteration count and runtime of the IR solver.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
using ir = gko::solver::Ir<ValueType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
```

```
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create RHS and initial guess as 1

```
gko::size_type size = A->get_size()[0];
auto host_x = gko::matrix::Dense<ValueType>::create(exec->get_master(),
                                                    gko::dim<2>(size, 1));
for (auto i = 0; i < size; i++) {
    host_x->at(i, 0) = 1.;
}
auto x = gko::clone(exec, host_x);
auto b = gko::clone(exec, host_x);
```

Calculate initial residual by overwriting b

```
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto initres = gko::initialize<real_vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(initres));
```

copy b again

```
b->copy_from(host_x.get());
gko::size_type max_iters = 10000u;
RealValueType outer_reduction_factor{1e-12};
auto iter_stop =
    gko::stop::Iteration::build().with_max_iters(max_iters).on(exec);
auto tol_stop = gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(outer_reduction_factor)
                    .on(exec);
std::shared_ptr<const gko::log::Convergence<ValueType» logger =
    gko::log::Convergence<ValueType>::create(exec);
iter_stop->add_logger(logger);
tol_stop->add_logger(logger);
```

Create solver factory

```
RealValueType inner_reduction_factor{1e-2};
auto solver_gen =
    ir::build()
        .with_solver(
            cg::build()
                .with_criteria(
                    gko::stop::ResidualNorm<ValueType>::build()
                        .with_reduction_factor(inner_reduction_factor)
                        .on(exec))
                .on(exec))
        .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
        .on(exec);
```

Create solver

```
auto solver = solver_gen->generate(A);
```

Solve system

```
exec->synchronize();
std::chrono::nanoseconds time(0);
auto tic = std::chrono::steady_clock::now();
solver->apply(lend(b), lend(x));
auto toc = std::chrono::steady_clock::now();
time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
```

Calculate residual

```
auto res = gko::initialize<real_vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(res));
```

```
std::cout « "Initial residual norm sqrt(r^T r):\n";
write(std::cout, lend(initres));
std::cout « "Final residual norm sqrt(r^T r):\n";
write(std::cout, lend(res));
```

Print solver statistics

```
    std::cout « "IR iteration count:     " « logger->get_num_iterations()
              « std::endl;
    std::cout « "IR execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
}
```

Results

This is the expected output:

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
194.679
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
4.23821e-11
IR iteration count:     24
IR execution time [ms]: 0.794962
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
```

```
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    using ir = gko::solver::Ir<ValueType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    gko::size_type size = A->get_size()[0];
    auto host_x = gko::matrix::Dense<ValueType>::create(exec->get_master(),
                                                        gko::dim<2>(size, 1));
    for (auto i = 0; i < size; i++) {
        host_x->at(i, 0) = 1.;
    }
    auto x = gko::clone(exec, host_x);
    auto b = gko::clone(exec, host_x);
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto initres = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(initres));
    b->copy_from(host_x.get());
    gko::size_type max_iters = 10000u;
    RealValueType outer_reduction_factor{1e-12};
    auto iter_stop =
        gko::stop::Iteration::build().with_max_iters(max_iters).on(exec);
    auto tol_stop = gko::stop::ResidualNorm<ValueType>::build()
                        .with_reduction_factor(outer_reduction_factor)
                        .on(exec);
    std::shared_ptr<const gko::log::Convergence<ValueType» logger =
        gko::log::Convergence<ValueType>::create(exec);
    iter_stop->add_logger(logger);
    tol_stop->add_logger(logger);
    RealValueType inner_reduction_factor{1e-2};
    auto solver_gen =
        ir::build()
            .with_solver(
                cg::build()
                    .with_criteria(
                        gko::stop::ResidualNorm<ValueType>::build()
                            .with_reduction_factor(inner_reduction_factor)
                            .on(exec))
                    .on(exec))
            .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
            .on(exec);
    auto solver = solver_gen->generate(A);
    exec->synchronize();
    std::chrono::nanoseconds time(0);
    auto tic = std::chrono::steady_clock::now();
    solver->apply(lend(b), lend(x));
    auto toc = std::chrono::steady_clock::now();
    time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Initial residual norm sqrt(r^T r):\n";
    write(std::cout, lend(initres));
    std::cout « "Final residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
    std::cout « "IR iteration count:     " « logger->get_num_iterations()
              « std::endl;
    std::cout « "IR execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
}
```

Chapter 23

The minimal-cuda-solver program

The minimal CUDA solver example..

This example depends on simple-solver.

Introduction

This is a minimal example that solves a system with Ginkgo. The matrix, right hand side and initial guess are read from standard input, and the result is written to standard output. The system matrix is stored in CSR format, and the system solved using the CG method, preconditioned with the block-Jacobi preconditioner. All computations are done on the GPU.

The easiest way to use the example data from the data/ folder is to concatenate the matrix, the right hand side and the initial solution (in that exact order), and pipe the result to the minimal_solver_cuda executable:

```
cat data/A.mtx data/b.mtx data/x0.mtx | ./minimal-cuda-solver
```

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <iostream>
int main()
{
```

Instantiate a CUDA executor

```
auto gpu = gko::CudaExecutor::create(0, gko::OmpExecutor::create(), true);
```

Read data

```
auto A = gko::read<gko::matrix::Csr<»(std::cin, gpu);
auto b = gko::read<gko::matrix::Dense<»(std::cin, gpu);
auto x = gko::read<gko::matrix::Dense<»(std::cin, gpu);
```

Create the solver

```
auto solver =
    gko::solver::Cg<>::build()
        .with_preconditioner(gko::preconditioner::Jacobi<>::build().on(gpu))
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(20u).on(gpu),
            gko::stop::ResidualNorm<>::build()
                .with_reduction_factor(1e-15)
                .on(gpu))
        .on(gpu);
```

Solve system

```
solver->generate(give(A))->apply(lend(b), lend(x));
```

Write result

```
    write(std::cout, lend(x));
}
```

Results

The following is the expected result when using the data contained in the folder `data` as input:

```
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <iostream>
int main()
{
    auto gpu = gko::CudaExecutor::create(0, gko::OmpExecutor::create(), true);
    auto A = gko::read<gko::matrix::Csr<»(std::cin, gpu);
    auto b = gko::read<gko::matrix::Dense<»(std::cin, gpu);
    auto x = gko::read<gko::matrix::Dense<»(std::cin, gpu);
    auto solver =
        gko::solver::Cg<>::build()
            .with_preconditioner(gko::preconditioner::Jacobi<>::build().on(gpu))
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(20u).on(gpu),
                gko::stop::ResidualNorm<>::build()
                    .with_reduction_factor(1e-15)
                    .on(gpu))
            .on(gpu);
    solver->generate(give(A))->apply(lend(b), lend(x));
    write(std::cout, lend(x));
}
```

Chapter 24

The mixed-multigrid-solver program

The mixed multigrid solver example..

This example depends on simple-solver.

This example shows how to use the mixed-precision multigrid solver.

In this example, we first read in a matrix from a file, then generate a right-hand side and an initial guess. The multigrid solver can mix different precision of MultigridLevel. The example features the generating time and runtime of the multigrid solver.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using ValueType2 = float;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using fcg = gko::solver::Fcg<ValueType>;
using cg = gko::solver::Cg<ValueType2>;
using ir = gko::solver::Ir<ValueType>;
using ir2 = gko::solver::Ir<ValueType2>;
using mg = gko::solver::Multigrid;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
using bj2 = gko::preconditioner::Jacobi<ValueType2, IndexType>;
using amgx_pgm = gko::multigrid::AmgxPgm<ValueType, IndexType>;
using amgx_pgm2 = gko::multigrid::AmgxPgm<ValueType2, IndexType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
const auto executor_string = argc >= 2 ? argv[1] : "reference";
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
```

```
            return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
        }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create RHS as 1 and initial guess as 0

```
gko::size_type size = A->get_size()[0];
auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
auto host_b = vec::create(exec->get_master(), gko::dim<2>(size, 1));
for (auto i = 0; i < size; i++) {
    host_x->at(i, 0) = 0.;
    host_b->at(i, 0) = 1.;
}
auto x = vec::create(exec);
auto b = vec::create(exec);
x->copy_from(host_x.get());
b->copy_from(host_b.get());
```

Calculate initial residual by overwriting b

```
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto initres = gko::initialize<vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(initres));
```

copy b again

```
b->copy_from(host_b.get());
```

Prepare the stopping criteria

```
const gko::remove_complex<ValueType> tolerance = 1e-12;
auto iter_stop =
    gko::stop::Iteration::build().with_max_iters(100u).on(exec);
auto tol_stop = gko::stop::AbsoluteResidualNorm<ValueType>::build()
                    .with_tolerance(tolerance)
                    .on(exec);
std::shared_ptr<const gko::log::Convergence<ValueType» logger =
    gko::log::Convergence<ValueType>::create(exec);
iter_stop->add_logger(logger);
tol_stop->add_logger(logger);
```

Create smoother factory (ir with bj)

```
auto smoother_gen = gko::share(
    ir::build()
        .with_solver(bj::build().with_max_block_size(1u).on(exec))
        .with_relaxation_factor(0.9)
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(2u).on(exec))
        .on(exec));
auto smoother_gen2 = gko::share(
    ir2::build()
        .with_solver(bj2::build().with_max_block_size(1u).on(exec))
        .with_relaxation_factor(0.9f)
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(2u).on(exec))
        .on(exec));
```

Create RestrictProlong factory

```
auto mg_level_gen = amgx_pgm::build().with_deterministic(true).on(exec);
auto mg_level_gen2 = amgx_pgm2::build().with_deterministic(true).on(exec);
```

Create CoarsesSolver factory

```
auto coarsest_solver_gen =
    cg::build()
```

```
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(4u).on(exec))
        .on(exec);
```

Create multigrid factory
```
auto multigrid_gen =
    mg::build()
        .with_max_levels(2u)
        .with_min_coarse_rows(5u)
        .with_pre_smoother(gko::share(smoother_gen),
                           gko::share(smoother_gen2))
        .with_post_uses_pre(true)
        .with_mg_level(gko::share(mg_level_gen), gko::share(mg_level_gen2))
        .with_coarsest_solver(
            gko::share(bj2::build().with_max_block_size(1u).on(exec)))
        .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
        .on(exec);
std::chrono::nanoseconds gen_time(0);
auto gen_tic = std::chrono::steady_clock::now();
```

auto solver = solver_gen->generate(A);
```
auto solver = multigrid_gen->generate(A);
exec->synchronize();
auto gen_toc = std::chrono::steady_clock::now();
gen_time +=
    std::chrono::duration_cast<std::chrono::nanoseconds>(gen_toc - gen_tic);
```

Solve system
```
exec->synchronize();
std::chrono::nanoseconds time(0);
auto tic = std::chrono::steady_clock::now();
solver->apply(lend(b), lend(x));
exec->synchronize();
auto toc = std::chrono::steady_clock::now();
time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
```

Calculate residual
```
auto res = gko::initialize<vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(res));
std::cout « "Initial residual norm sqrt(r^T r): \n";
write(std::cout, lend(initres));
std::cout « "Final residual norm sqrt(r^T r): \n";
write(std::cout, lend(res));
auto mg_level_list = solver->get_mg_level_list();
auto smoother_list = solver->get_pre_smoother_list();
```

Check the MultigridLevel and smoother. throw error if there is mismatch
```
auto level0 = gko::as<amgx_pgm>(mg_level_list.at(0));
auto level1 = gko::as<amgx_pgm2>(mg_level_list.at(1));
auto smoother0 = gko::as<ir>(smoother_list.at(0));
auto smoother1 = gko::as<ir2>(smoother_list.at(1));
```

Print solver statistics
```
    std::cout « "Multigrid iteration count:     "
              « logger->get_num_iterations() « std::endl;
    std::cout « "Multigrid generation time [ms]: "
              « static_cast<double>(gen_time.count()) / 1000000.0 « std::endl;
    std::cout « "Multigrid execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
    std::cout « "Multigrid execution time per iteraion[ms]: "
              « static_cast<double>(time.count()) / 1000000.0 /
                     logger->get_num_iterations()
              « std::endl;
}
```

Results

This is the expected output:
```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
4.3589
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
6.31088e-14
Multigrid iteration count:     9
Multigrid generation time [ms]: 3.35361
Multigrid execution time [ms]: 10.048
Multigrid execution time per iteraion[ms]: 1.11644
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using ValueType2 = float;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using fcg = gko::solver::Fcg<ValueType>;
    using cg = gko::solver::Cg<ValueType2>;
    using ir = gko::solver::Ir<ValueType>;
    using ir2 = gko::solver::Ir<ValueType2>;
    using mg = gko::solver::Multigrid;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    using bj2 = gko::preconditioner::Jacobi<ValueType2, IndexType>;
    using amgx_pgm = gko::multigrid::AmgxPgm<ValueType, IndexType>;
    using amgx_pgm2 = gko::multigrid::AmgxPgm<ValueType2, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    gko::size_type size = A->get_size()[0];
    auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
    auto host_b = vec::create(exec->get_master(), gko::dim<2>(size, 1));
    for (auto i = 0; i < size; i++) {
```

```
        host_x->at(i, 0) = 0.;
        host_b->at(i, 0) = 1.;
    }
    auto x = vec::create(exec);
    auto b = vec::create(exec);
    x->copy_from(host_x.get());
    b->copy_from(host_b.get());
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto initres = gko::initialize<vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(initres));
    b->copy_from(host_b.get());
    const gko::remove_complex<ValueType> tolerance = 1e-12;
    auto iter_stop =
        gko::stop::Iteration::build().with_max_iters(100u).on(exec);
    auto tol_stop = gko::stop::AbsoluteResidualNorm<ValueType>::build()
                        .with_tolerance(tolerance)
                        .on(exec);
    std::shared_ptr<const gko::log::Convergence<ValueType» logger =
        gko::log::Convergence<ValueType>::create(exec);
    iter_stop->add_logger(logger);
    tol_stop->add_logger(logger);
    auto smoother_gen = gko::share(
        ir::build()
            .with_solver(bj::build().with_max_block_size(1u).on(exec))
            .with_relaxation_factor(0.9)
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(2u).on(exec))
            .on(exec));
    auto smoother_gen2 = gko::share(
        ir2::build()
            .with_solver(bj2::build().with_max_block_size(1u).on(exec))
            .with_relaxation_factor(0.9f)
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(2u).on(exec))
            .on(exec));
    auto mg_level_gen = amgx_pgm::build().with_deterministic(true).on(exec);
    auto mg_level_gen2 = amgx_pgm2::build().with_deterministic(true).on(exec);
    auto coarsest_solver_gen =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(4u).on(exec))
            .on(exec);
    auto multigrid_gen =
        mg::build()
            .with_max_levels(2u)
            .with_min_coarse_rows(5u)
            .with_pre_smoother(gko::share(smoother_gen),
                               gko::share(smoother_gen2))
            .with_post_uses_pre(true)
            .with_mg_level(gko::share(mg_level_gen), gko::share(mg_level_gen2))
            .with_coarsest_solver(
                gko::share(bj2::build().with_max_block_size(1u).on(exec)))
            .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
            .on(exec);
    std::chrono::nanoseconds gen_time(0);
    auto gen_tic = std::chrono::steady_clock::now();
    auto solver = multigrid_gen->generate(A);
    exec->synchronize();
    auto gen_toc = std::chrono::steady_clock::now();
    gen_time +=
        std::chrono::duration_cast<std::chrono::nanoseconds>(gen_toc - gen_tic);
    exec->synchronize();
    std::chrono::nanoseconds time(0);
    auto tic = std::chrono::steady_clock::now();
    solver->apply(lend(b), lend(x));
    exec->synchronize();
    auto toc = std::chrono::steady_clock::now();
    time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
    auto res = gko::initialize<vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Initial residual norm sqrt(r^T r): \n";
    write(std::cout, lend(initres));
    std::cout « "Final residual norm sqrt(r^T r): \n";
    write(std::cout, lend(res));
    auto mg_level_list = solver->get_mg_level_list();
    auto smoother_list = solver->get_pre_smoother_list();
    auto level0 = gko::as<amgx_pgm>(mg_level_list.at(0));
    auto level1 = gko::as<amgx_pgm2>(mg_level_list.at(1));
    auto smoother0 = gko::as<ir>(smoother_list.at(0));
    auto smoother1 = gko::as<ir2>(smoother_list.at(1));
    std::cout « "Multigrid iteration count:     "
              « logger->get_num_iterations() « std::endl;
    std::cout « "Multigrid generation time [ms]: "
              « static_cast<double>(gen_time.count()) / 1000000.0 « std::endl;
```

```
    std::cout « "Multigrid execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
    std::cout « "Multigrid execution time per iteraion[ms]: "
              « static_cast<double>(time.count()) / 1000000.0 /
                    logger->get_num_iterations()
              « std::endl;
}
```

Chapter 25

The mixed-precision-ir program

The Mixed Precision Iterative Refinement (MPIR) solver example..

This example depends on iterative-refinement.

This example manually implements a Mixed Precision Iterative Refinement (MPIR) solver.

In this example, we first read in a matrix from file, then generate a right-hand side and an initial guess. An inaccurate CG solver in single precision is used as the inner solver to an iterative refinement (IR) in double precision method which solves a linear system.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using SolverType = float;
using RealSolverType = gko::remove_complex<SolverType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using solver_vec = gko::matrix::Dense<SolverType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using solver_mtx = gko::matrix::Csr<SolverType, IndexType>;
using cg = gko::solver::Cg<SolverType>;
gko::size_type max_outer_iters = 100u;
gko::size_type max_inner_iters = 100u;
RealValueType outer_reduction_factor{1e-12};
RealSolverType inner_reduction_factor{1e-2};
```

Print version information

```
std::cout << gko::version_info::get() << std::endl;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr << "Usage: " << argv[0] << " [executor]" << std::endl;
    std::exit(-1);
```

```
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()>»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create RHS and initial guess as 1

```
gko::size_type size = A->get_size()[0];
auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
for (auto i = 0; i < size; i++) {
    host_x->at(i, 0) = 1.;
}
auto x = gko::clone(exec, host_x);
auto b = gko::clone(exec, host_x);
```

Calculate initial residual by overwriting b

```
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto initres_vec = gko::initialize<real_vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(initres_vec));
```

Build lower-precision system matrix and residual

```
auto solver_A = solver_mtx::create(exec);
auto inner_residual = solver_vec::create(exec);
auto outer_residual = vec::create(exec);
A->convert_to(lend(solver_A));
b->convert_to(lend(outer_residual));
```

restore b

```
b->copy_from(host_x.get());
```

Create inner solver

```
auto inner_solver =
    cg::build()
        .with_criteria(gko::stop::ResidualNorm<SolverType>::build()
                           .with_reduction_factor(inner_reduction_factor)
                           .on(exec),
                       gko::stop::Iteration::build()
                           .with_max_iters(max_inner_iters)
                           .on(exec))
        .on(exec)
        ->generate(give(solver_A));
```

Solve system

```
exec->synchronize();
std::chrono::nanoseconds time(0);
auto res_vec = gko::initialize<real_vec>({0.0}, exec);
auto initres = exec->copy_val_to_host(initres_vec->get_const_values());
auto inner_solution = solver_vec::create(exec);
auto outer_delta = vec::create(exec);
auto tic = std::chrono::steady_clock::now();
int iter = -1;
while (true) {
    ++iter;
```

convert residual to inner precision

```
outer_residual->convert_to(lend(inner_residual));
outer_residual->compute_norm2(lend(res_vec));
auto res = exec->copy_val_to_host(res_vec->get_const_values());
```

break if we exceed the number of iterations or have converged

```
if (iter > max_outer_iters || res / initres < outer_reduction_factor) {
    break;
}
```

Use the inner solver to solve A ∗ inner_solution = inner_residual with residual as initial guess.

```
inner_solution->copy_from(lend(inner_residual));
inner_solver->apply(lend(inner_residual), lend(inner_solution));
```

convert inner solution to outer precision

```
inner_solution->convert_to(lend(outer_delta));
```

x = x + inner_solution

```
x->add_scaled(lend(one), lend(outer_delta));
```

residual = b - A ∗ x

```
    outer_residual->copy_from(lend(b));
    A->apply(lend(neg_one), lend(x), lend(one), lend(outer_residual));
}
auto toc = std::chrono::steady_clock::now();
time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
```

Calculate residual

```
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(res_vec));
std::cout « "Initial residual norm sqrt(r^T r):\n";
write(std::cout, lend(initres_vec));
std::cout « "Final residual norm sqrt(r^T r):\n";
write(std::cout, lend(res_vec));
```

Print solver statistics

```
    std::cout « "MPIR iteration count:     " « iter « std::endl;
    std::cout « "MPIR execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
}
```

Results

This is the expected output:

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
194.679
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1.22728e-10
MPIR iteration count:     25
MPIR execution time [ms]: 0.846559
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
```

```
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using SolverType = float;
    using RealSolverType = gko::remove_complex<SolverType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using solver_vec = gko::matrix::Dense<SolverType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using solver_mtx = gko::matrix::Csr<SolverType, IndexType>;
    using cg = gko::solver::Cg<SolverType>;
    gko::size_type max_outer_iters = 100u;
    gko::size_type max_inner_iters = 100u;
    RealValueType outer_reduction_factor{1e-12};
    RealSolverType inner_reduction_factor{1e-2};
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    gko::size_type size = A->get_size()[0];
    auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
    for (auto i = 0; i < size; i++) {
        host_x->at(i, 0) = 1.;
    }
    auto x = gko::clone(exec, host_x);
    auto b = gko::clone(exec, host_x);
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto initres_vec = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(initres_vec));
    auto solver_A = solver_mtx::create(exec);
    auto inner_residual = solver_vec::create(exec);
    auto outer_residual = vec::create(exec);
    A->convert_to(lend(solver_A));
    b->convert_to(lend(outer_residual));
```

```
    b->copy_from(host_x.get());
    auto inner_solver =
        cg::build()
            .with_criteria(gko::stop::ResidualNorm<SolverType>::build()
                               .with_reduction_factor(inner_reduction_factor)
                               .on(exec),
                           gko::stop::Iteration::build()
                               .with_max_iters(max_inner_iters)
                               .on(exec))
            .on(exec)
            ->generate(give(solver_A));
    exec->synchronize();
    std::chrono::nanoseconds time(0);
    auto res_vec = gko::initialize<real_vec>({0.0}, exec);
    auto initres = exec->copy_val_to_host(initres_vec->get_const_values());
    auto inner_solution = solver_vec::create(exec);
    auto outer_delta = vec::create(exec);
    auto tic = std::chrono::steady_clock::now();
    int iter = -1;
    while (true) {
        ++iter;
        outer_residual->convert_to(lend(inner_residual));
        outer_residual->compute_norm2(lend(res_vec));
        auto res = exec->copy_val_to_host(res_vec->get_const_values());
        if (iter > max_outer_iters || res / initres < outer_reduction_factor) {
            break;
        }
        inner_solution->copy_from(lend(inner_residual));
        inner_solver->apply(lend(inner_residual), lend(inner_solution));
        inner_solution->convert_to(lend(outer_delta));
        x->add_scaled(lend(one), lend(outer_delta));
        outer_residual->copy_from(lend(b));
        A->apply(lend(neg_one), lend(x), lend(one), lend(outer_residual));
    }
    auto toc = std::chrono::steady_clock::now();
    time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res_vec));
    std::cout « "Initial residual norm sqrt(r^T r):\n";
    write(std::cout, lend(initres_vec));
    std::cout « "Final residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res_vec));
    std::cout « "MPIR iteration count:     " « iter « std::endl;
    std::cout « "MPIR execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
}
```

Chapter 26

The mixed-spmv program

The mixed spmv example..

Introduction

This mixed spmv example should give the usage of Ginkgo mixed precision. This example is meant for you to understand how Ginkgo works with different precision of data. We encourage you to play with the code, change the parameters and see what is best suited for your purposes.

About the example

Each example has the following sections:

1. **Introduction:**This gives an overview of the example and mentions any interesting aspects in the example that might help the reader.
2. **The commented program:** This section is intended for you to understand the details of the example so that you can play with it and understand Ginkgo and its features better.
3. **Results:** This section shows the results of the code when run. Though the results may not be completely the same, you can expect the behaviour to be similar.
4. **The plain program:** This is the complete code without any comments to have an complete overview of the code.

The commented program

Include files

This is the main ginkgo header file.
```
#include <ginkgo/ginkgo.hpp>
```

Add the fstream header to read from data from files.
```
#include <fstream>
```

Add the C++ iostream header to output information to the console.

```
#include <iostream>
```

Add the STL map header for the executor selection.

```
#include <map>
```

Add the string manipulation header to handle strings.

```
#include <string>
```

Add the timing header for timing.

```
#include <chrono>
```

Add the random header to generate random vectors.

```
#include <random>
namespace {
/ **
 * Generate a random value.
 *
 * @tparam ValueType  valuetype of the value
 * @tparam ValueDistribution  type of value distribution
 * @tparam Engine  type of random engine
 *
 * @param value_dist  distribution of array values
 * @param engine  a random engine
 *
 * @return ValueType
 * /
template <typename ValueType, typename ValueDistribution, typename Engine>
typename std::enable_if<!gko::is_complex_s<ValueType>::value, ValueType>::type
get_rand_value(ValueDistribution&& value_dist, Engine&& gen)
{
    return value_dist(gen);
}
/ **
 * Specialization for complex types.
 *
 * @copydoc get_rand_value
 * /
template <typename ValueType, typename ValueDistribution, typename Engine>
typename std::enable_if<gko::is_complex_s<ValueType>::value, ValueType>::type
get_rand_value(ValueDistribution&& value_dist, Engine&& gen)
{
    return ValueType(value_dist(gen), value_dist(gen));
}
/ **
 * timing the apply operation A->apply(b, x). It will runs 2 warmup and get
 * average time among 10 times.
 *
 * @return seconds
 * /
double timing(std::shared_ptr<const gko::Executor> exec,
              std::shared_ptr<const gko::LinOp> A,
              std::shared_ptr<const gko::LinOp> b,
              std::shared_ptr<gko::LinOp> x)
{
    int warmup = 2;
    int rep = 10;
    for (int i = 0; i < warmup; i++) {
        A->apply(lend(b), lend(x));
    }
    double total_sec = 0;
    for (int i = 0; i < rep; i++) {
```

always clone the x in each apply

```
auto xx = x->clone();
```

synchronize to make sure data is already on device

```
exec->synchronize();
auto start = std::chrono::steady_clock::now();
A->apply(lend(b), lend(xx));
```

synchronize to make sure the operation is done

```
exec->synchronize();
auto stop = std::chrono::steady_clock::now();
```

get the duration in seconds

```
std::chrono::duration<double> duration_time = stop - start;
total_sec += duration_time.count();
if (i + 1 == rep) {
```

copy the result back to x
```
            x->copy_from(lend(xx));
        }
    }
    return total_sec / rep;
}
}  // namespace
int main(int argc, char* argv[])
{
```

Use some shortcuts. In Ginkgo, vectors are seen as a gko::matrix::Dense with one column/one row. The advantage of this concept is that using multiple vectors is a now a natural extension of adding columns/rows are necessary.
```
using HighPrecision = double;
using RealValueType = gko::remove_complex<HighPrecision>;
using LowPrecision = float;
using IndexType = int;
using hp_vec = gko::matrix::Dense<HighPrecision>;
using lp_vec = gko::matrix::Dense<LowPrecision>;
using real_vec = gko::matrix::Dense<RealValueType>;
```

The gko::matrix::Ell class is used here, but any other matrix class such as gko::matrix::Coo, gko::matrix::Hybrid, gko::matrix::Csr or gko::matrix::Sellp could also be used. Note. the behavior will depends GINKGO_MIXED_PR←ECISION flags and the actual implementation from different matrices.
```
using hp_mtx = gko::matrix::Ell<HighPrecision, IndexType>;
using lp_mtx = gko::matrix::Ell<LowPrecision, IndexType>;
```

Print the ginkgo version information.
```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
    std::exit(-1);
}
```

Where do you want to run your operation?

The gko::Executor class is one of the cornerstones of Ginkgo. Currently, we have support for an gko::OmpExecutor, which uses OpenMP multi-threading in most of its kernels, a gko::ReferenceExecutor, a single threaded specialization of the OpenMP executor and a gko::CudaExecutor which runs the code on a NVIDIA GPU if available.

Note

> With the help of C++, you see that you only ever need to change the executor and all the other functions/ routines within Ginkgo should automatically work and run on the executor with any other changes.

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation
```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Preparing your data and transfer to the proper device.

Read the matrix using the read function and set the right hand side randomly.

Note

Ginkgo uses C++ smart pointers to automatically manage memory. To this end, we use our own object ownership transfer functions that under the hood call the required smart pointer functions to manage object ownership. The gko::share , gko::give and gko::lend are the functions that you would need to use.

read the matrix into HighPrecision and LowPrecision.

```
auto hp_A = share(gko::read<hp_mtx>(std::ifstream("data/A.mtx"), exec));
auto lp_A = share(gko::read<lp_mtx>(std::ifstream("data/A.mtx"), exec));
```

Set the shortcut for each dimension

```
auto A_dim = hp_A->get_size();
auto b_dim = gko::dim<2>{A_dim[1], 1};
auto x_dim = gko::dim<2>{A_dim[0], b_dim[1]};
auto host_b = hp_vec::create(exec->get_master(), b_dim);
```

fill the b vector with some random data

```
std::default_random_engine rand_engine(32);
auto dist = std::uniform_real_distribution<RealValueType>(0.0, 1.0);
for (int i = 0; i < host_b->get_size()[0]; i++) {
    host_b->at(i, 0) = get_rand_value<HighPrecision>(dist, rand_engine);
}
```

copy the data from host to device

```
auto hp_b = share(gko::clone(exec, host_b));
auto lp_b = share(lp_vec::create(exec));
lp_b->copy_from(lend(hp_b));
```

create several result x vector in different precision

```
auto hp_x = share(hp_vec::create(exec, x_dim));
auto lp_x = share(lp_vec::create(exec, x_dim));
auto hplp_x = share(hp_x->clone());
auto lplp_x = share(hp_x->clone());
auto lphp_x = share(hp_x->clone());
```

Measure the time of apply

We measure the time among different combination of apply operation.

Hp $*$ Hp -> Hp

```
auto hp_sec = timing(exec, hp_A, hp_b, hp_x);
```

Lp $*$ Lp -> Lp

```
auto lp_sec = timing(exec, lp_A, lp_b, lp_x);
```

Hp $*$ Lp -> Hp

```
auto hplp_sec = timing(exec, hp_A, lp_b, hplp_x);
```

Lp $*$ Lp -> Hp

```
auto lplp_sec = timing(exec, lp_A, lp_b, lplp_x);
```

Lp $*$ Hp -> Hp

```
auto lphp_sec = timing(exec, lp_A, hp_b, lphp_x);
```

To measure error of result. neg_one is an object that represent the number -1.0 which allows for a uniform interface when computing on any device. To compute the residual, all you need to do is call the add_scaled method, which in this case is an axpy and equivalent to the LAPACK axpy routine. Finally, you compute the euclidean 2-norm with the compute_norm2 function.

```
auto neg_one = gko::initialize<hp_vec>({-1.0}, exec);
auto hp_x_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
auto lp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
```

```
    auto hplp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lplp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lphp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lp_diff = hp_x->clone();
    auto hplp_diff = hp_x->clone();
    auto lplp_diff = hp_x->clone();
    auto lphp_diff = hp_x->clone();
    hp_x->compute_norm2(lend(hp_x_norm));
    lp_diff->add_scaled(lend(neg_one), lend(lp_x));
    lp_diff->compute_norm2(lend(lp_diff_norm));
    hplp_diff->add_scaled(lend(neg_one), lend(hplp_x));
    hplp_diff->compute_norm2(lend(hplp_diff_norm));
    lplp_diff->add_scaled(lend(neg_one), lend(lplp_x));
    lplp_diff->compute_norm2(lend(lplp_diff_norm));
    lphp_diff->add_scaled(lend(neg_one), lend(lphp_x));
    lphp_diff->compute_norm2(lend(lphp_diff_norm));
    exec->synchronize();
    std::cout.precision(10);
    std::cout « std::scientific;
    std::cout « "High Precision time(s): " « hp_sec « std::endl;
    std::cout « "High Precision result norm: " « hp_x_norm->at(0)
              « std::endl;
    std::cout « "Low Precision time(s): " « lp_sec « std::endl;
    std::cout « "Low Precision relative error: "
              « lp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
    std::cout « "Hp * Lp -> Hp time(s): " « hplp_sec « std::endl;
    std::cout « "Hp * Lp -> Hp relative error: "
              « hplp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
    std::cout « "Lp * Lp -> Hp time(s): " « lplp_sec « std::endl;
    std::cout « "Lp * Lp -> Hp relative error: "
              « lplp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
    std::cout « "Lp * Hp -> Hp time(s): " « lplp_sec « std::endl;
    std::cout « "Lp * Hp -> Hp relative error: "
              « lphp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
}
```

Results

The following is the expected result (omp):

```
High Precision time(s): 2.0568800000e-05
High Precision result norm: 1.7725534898e+05
Low Precision time(s): 2.0955600000e-05
Low Precision relative error: 9.1052887738e-08
Hp * Lp -> Hp time(s): 2.1186100000e-05
Hp * Lp -> Hp relative error: 3.7799774251e-08
Lp * Lp -> Hp time(s): 2.0312300000e-05
Lp * Lp -> Hp relative error: 5.7910008031e-08
Lp * Hp -> Hp time(s): 2.0312300000e-05
Lp * Hp -> Hp relative error: 3.7173133506e-08
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
```

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <chrono>
#include <random>
namespace {
template <typename ValueType, typename ValueDistribution, typename Engine>
typename std::enable_if<!gko::is_complex_s<ValueType>::value, ValueType>::type
get_rand_value(ValueDistribution&& value_dist, Engine&& gen)
{
    return value_dist(gen);
}
template <typename ValueType, typename ValueDistribution, typename Engine>
typename std::enable_if<gko::is_complex_s<ValueType>::value, ValueType>::type
get_rand_value(ValueDistribution&& value_dist, Engine&& gen)
{
    return ValueType(value_dist(gen), value_dist(gen));
}
double timing(std::shared_ptr<const gko::Executor> exec,
              std::shared_ptr<const gko::LinOp> A,
              std::shared_ptr<const gko::LinOp> b,
              std::shared_ptr<gko::LinOp> x)
{
    int warmup = 2;
    int rep = 10;
    for (int i = 0; i < warmup; i++) {
        A->apply(lend(b), lend(x));
    }
    double total_sec = 0;
    for (int i = 0; i < rep; i++) {
        auto xx = x->clone();
        exec->synchronize();
        auto start = std::chrono::steady_clock::now();
        A->apply(lend(b), lend(xx));
        exec->synchronize();
        auto stop = std::chrono::steady_clock::now();
        std::chrono::duration<double> duration_time = stop - start;
        total_sec += duration_time.count();
        if (i + 1 == rep) {
            x->copy_from(lend(xx));
        }
    }
    return total_sec / rep;
}
}  // namespace
int main(int argc, char* argv[])
{
    using HighPrecision = double;
    using RealValueType = gko::remove_complex<HighPrecision>;
    using LowPrecision = float;
    using IndexType = int;
    using hp_vec = gko::matrix::Dense<HighPrecision>;
    using lp_vec = gko::matrix::Dense<LowPrecision>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using hp_mtx = gko::matrix::Ell<HighPrecision, IndexType>;
    using lp_mtx = gko::matrix::Ell<LowPrecision, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
```

```
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto hp_A = share(gko::read<hp_mtx>(std::ifstream("data/A.mtx"), exec));
    auto lp_A = share(gko::read<lp_mtx>(std::ifstream("data/A.mtx"), exec));
    auto A_dim = hp_A->get_size();
    auto b_dim = gko::dim<2>{A_dim[1], 1};
    auto x_dim = gko::dim<2>{A_dim[0], b_dim[1]};
    auto host_b = hp_vec::create(exec->get_master(), b_dim);
    std::default_random_engine rand_engine(32);
    auto dist = std::uniform_real_distribution<RealValueType>(0.0, 1.0);
    for (int i = 0; i < host_b->get_size()[0]; i++) {
        host_b->at(i, 0) = get_rand_value<HighPrecision>(dist, rand_engine);
    }
    auto hp_b = share(gko::clone(exec, host_b));
    auto lp_b = share(lp_vec::create(exec));
    lp_b->copy_from(lend(hp_b));
    auto hp_x = share(hp_vec::create(exec, x_dim));
    auto lp_x = share(lp_vec::create(exec, x_dim));
    auto hplp_x = share(hp_x->clone());
    auto lplp_x = share(hp_x->clone());
    auto lphp_x = share(hp_x->clone());
    auto hp_sec = timing(exec, hp_A, hp_b, hp_x);
    auto lp_sec = timing(exec, lp_A, lp_b, lp_x);
    auto hplp_sec = timing(exec, hp_A, lp_b, hplp_x);
    auto lplp_sec = timing(exec, lp_A, lp_b, lplp_x);
    auto lphp_sec = timing(exec, lp_A, hp_b, lphp_x);
    auto neg_one = gko::initialize<hp_vec>({-1.0}, exec);
    auto hp_x_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto hplp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lplp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lphp_diff_norm = gko::initialize<real_vec>({0.0}, exec->get_master());
    auto lp_diff = hp_x->clone();
    auto hplp_diff = hp_x->clone();
    auto lplp_diff = hp_x->clone();
    auto lphp_diff = hp_x->clone();
    hp_x->compute_norm2(lend(hp_x_norm));
    lp_diff->add_scaled(lend(neg_one), lend(lp_x));
    lp_diff->compute_norm2(lend(lp_diff_norm));
    hplp_diff->add_scaled(lend(neg_one), lend(hplp_x));
    hplp_diff->compute_norm2(lend(hplp_diff_norm));
    lplp_diff->add_scaled(lend(neg_one), lend(lplp_x));
    lplp_diff->compute_norm2(lend(lplp_diff_norm));
    lphp_diff->add_scaled(lend(neg_one), lend(lphp_x));
    lphp_diff->compute_norm2(lend(lphp_diff_norm));
    exec->synchronize();
    std::cout.precision(10);
    std::cout « std::scientific;
    std::cout « "High Precision time(s): " « hp_sec « std::endl;
    std::cout « "High Precision result norm: " « hp_x_norm->at(0)
              « std::endl;
    std::cout « "Low Precision time(s): " « lp_sec « std::endl;
    std::cout « "Low Precision relative error: "
              « lp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
    std::cout « "Hp * Lp -> Hp time(s): " « hplp_sec « std::endl;
    std::cout « "Hp * Lp -> Hp relative error: "
              « hplp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
    std::cout « "Lp * Lp -> Hp time(s): " « lplp_sec « std::endl;
    std::cout « "Lp * Lp -> Hp relative error: "
              « lplp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
    std::cout « "Lp * Hp -> Hp time(s): " « lplp_sec « std::endl;
    std::cout « "Lp * Hp -> Hp relative error: "
              « lphp_diff_norm->at(0) / hp_x_norm->at(0) « "\n";
}
```

Chapter 27

The multigrid-preconditioned-solver program

The preconditioned solver example..

This example depends on preconditioned-solver.

This example shows how to use the multigrid preconditioner.

In this example, we first read in a matrix from a file. The preconditioned CG solver is enhanced with a multigrid preconditioner. The example features the generating time and runtime of the CG solver.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using fcg = gko::solver::Fcg<ValueType>;
using cg = gko::solver::Cg<ValueType>;
using ir = gko::solver::Ir<ValueType>;
using mg = gko::solver::Multigrid;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
using amgx_pgm = gko::multigrid::AmgxPgm<ValueType, IndexType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
const auto executor_string = argc >= 2 ? argv[1] : "reference";
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
```

```
                                                     true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
```

Create RHS as 1 and initial guess as 0

```
gko::size_type size = A->get_size()[0];
auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
auto host_b = vec::create(exec->get_master(), gko::dim<2>(size, 1));
for (auto i = 0; i < size; i++) {
    host_x->at(i, 0) = 0.;
    host_b->at(i, 0) = 1.;
}
auto x = vec::create(exec);
auto b = vec::create(exec);
x->copy_from(host_x.get());
b->copy_from(host_b.get());
```

Calculate initial residual by overwriting b

```
auto one = gko::initialize<vec>({1.0}, exec);
auto neg_one = gko::initialize<vec>({-1.0}, exec);
auto initres = gko::initialize<vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(initres));
```

copy b again

```
b->copy_from(host_b.get());
```

Prepare the stopping criteria

```
const gko::remove_complex<ValueType> tolerance = 1e-8;
auto iter_stop =
    gko::stop::Iteration::build().with_max_iters(100u).on(exec);
auto tol_stop = gko::stop::AbsoluteResidualNorm<ValueType>::build()
                    .with_tolerance(tolerance)
                    .on(exec);
std::shared_ptr<const gko::log::Convergence<ValueType>> logger =
    gko::log::Convergence<ValueType>::create(exec);
iter_stop->add_logger(logger);
tol_stop->add_logger(logger);
```

Create smoother factory (ir with bj)

```
auto inner_solver_gen =
    gko::share(bj::build().with_max_block_size(1u).on(exec));
auto smoother_gen = gko::share(
    ir::build()
        .with_solver(inner_solver_gen)
        .with_relaxation_factor(0.9)
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(2u).on(exec))
        .on(exec));
```

Create MultigridLevel factory

```
auto mg_level_gen = amgx_pgm::build().with_deterministic(true).on(exec);
```

Create CoarsestSolver factory

```
auto coarsest_gen = gko::share(
    ir::build()
        .with_solver(inner_solver_gen)
        .with_relaxation_factor(0.9)
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(4u).on(exec))
        .on(exec));
```

Create multigrid factory

```
auto multigrid_gen =
    mg::build()
        .with_max_levels(9u)
        .with_min_coarse_rows(10u)
```

```
        .with_pre_smoother(smoother_gen)
        .with_post_uses_pre(true)
        .with_mg_level(gko::share(mg_level_gen))
        .with_coarsest_solver(coarsest_gen)
        .with_zero_guess(true)
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(1u).on(exec))
        .on(exec);
```

Create solver factory

```
auto solver_gen =
    cg::build()
        .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
        .with_preconditioner(gko::share(multigrid_gen))
        .on(exec);
```

Create solver

```
std::chrono::nanoseconds gen_time(0);
auto gen_tic = std::chrono::steady_clock::now();
auto solver = solver_gen->generate(A);
exec->synchronize();
auto gen_toc = std::chrono::steady_clock::now();
gen_time +=
    std::chrono::duration_cast<std::chrono::nanoseconds>(gen_toc - gen_tic);
```

Solve system

```
exec->synchronize();
std::chrono::nanoseconds time(0);
auto tic = std::chrono::steady_clock::now();
solver->apply(lend(b), lend(x));
exec->synchronize();
auto toc = std::chrono::steady_clock::now();
time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
```

Calculate residual

```
auto res = gko::initialize<vec>({0.0}, exec);
A->apply(lend(one), lend(x), lend(neg_one), lend(b));
b->compute_norm2(lend(res));
std::cout « "Initial residual norm sqrt(r^T r): \n";
write(std::cout, lend(initres));
std::cout « "Final residual norm sqrt(r^T r): \n";
write(std::cout, lend(res));
```

Print solver statistics

```
    std::cout « "CG iteration count:     " « logger->get_num_iterations()
              « std::endl;
    std::cout « "CG generation time [ms]: "
              « static_cast<double>(gen_time.count()) / 1000000.0 « std::endl;
    std::cout « "CG execution time [ms]: "
              « static_cast<double>(time.count()) / 1000000.0 « std::endl;
    std::cout « "CG execution time per iteraion[ms]: "
              « static_cast<double>(time.count()) / 1000000.0 /
                    logger->get_num_iterations()
              « std::endl;
}
```

Results

This is the expected output:

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
4.3589
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1.69858e-09
CG iteration count:     39
CG generation time [ms]: 2.04293
CG execution time [ms]: 22.3874
CG execution time per iteraion[ms]: 0.574036
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using fcg = gko::solver::Fcg<ValueType>;
    using cg = gko::solver::Cg<ValueType>;
    using ir = gko::solver::Ir<ValueType>;
    using mg = gko::solver::Multigrid;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    using amgx_pgm = gko::multigrid::AmgxPgm<ValueType, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    gko::size_type size = A->get_size()[0];
    auto host_x = vec::create(exec->get_master(), gko::dim<2>(size, 1));
    auto host_b = vec::create(exec->get_master(), gko::dim<2>(size, 1));
    for (auto i = 0; i < size; i++) {
        host_x->at(i, 0) = 0.;
        host_b->at(i, 0) = 1.;
    }
    auto x = vec::create(exec);
```

```
    auto b = vec::create(exec);
    x->copy_from(host_x.get());
    b->copy_from(host_b.get());
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto initres = gko::initialize<vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(initres));
    b->copy_from(host_b.get());
    const gko::remove_complex<ValueType> tolerance = 1e-8;
    auto iter_stop =
        gko::stop::Iteration::build().with_max_iters(100u).on(exec);
    auto tol_stop = gko::stop::AbsoluteResidualNorm<ValueType>::build()
                        .with_tolerance(tolerance)
                        .on(exec);
    std::shared_ptr<const gko::log::Convergence<ValueType>> logger =
        gko::log::Convergence<ValueType>::create(exec);
    iter_stop->add_logger(logger);
    tol_stop->add_logger(logger);
    auto inner_solver_gen =
        gko::share(bj::build().with_max_block_size(1u).on(exec));
    auto smoother_gen = gko::share(
        ir::build()
            .with_solver(inner_solver_gen)
            .with_relaxation_factor(0.9)
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(2u).on(exec))
            .on(exec));
    auto mg_level_gen = amgx_pgm::build().with_deterministic(true).on(exec);
    auto coarsest_gen = gko::share(
        ir::build()
            .with_solver(inner_solver_gen)
            .with_relaxation_factor(0.9)
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(4u).on(exec))
            .on(exec));
    auto multigrid_gen =
        mg::build()
            .with_max_levels(9u)
            .with_min_coarse_rows(10u)
            .with_pre_smoother(smoother_gen)
            .with_post_uses_pre(true)
            .with_mg_level(gko::share(mg_level_gen))
            .with_coarsest_solver(coarsest_gen)
            .with_zero_guess(true)
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(1u).on(exec))
            .on(exec);
    auto solver_gen =
        cg::build()
            .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
            .with_preconditioner(gko::share(multigrid_gen))
            .on(exec);
    std::chrono::nanoseconds gen_time(0);
    auto gen_tic = std::chrono::steady_clock::now();
    auto solver = solver_gen->generate(A);
    exec->synchronize();
    auto gen_toc = std::chrono::steady_clock::now();
    gen_time +=
        std::chrono::duration_cast<std::chrono::nanoseconds>(gen_toc - gen_tic);
    exec->synchronize();
    std::chrono::nanoseconds time(0);
    auto tic = std::chrono::steady_clock::now();
    solver->apply(lend(b), lend(x));
    exec->synchronize();
    auto toc = std::chrono::steady_clock::now();
    time += std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic);
    auto res = gko::initialize<vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout << "Initial residual norm sqrt(r^T r): \n";
    write(std::cout, lend(initres));
    std::cout << "Final residual norm sqrt(r^T r): \n";
    write(std::cout, lend(res));
    std::cout << "CG iteration count:     " << logger->get_num_iterations()
              << std::endl;
    std::cout << "CG generation time [ms]: "
              << static_cast<double>(gen_time.count()) / 1000000.0 << std::endl;
    std::cout << "CG execution time [ms]: "
              << static_cast<double>(time.count()) / 1000000.0 << std::endl;
    std::cout << "CG execution time per iteraion[ms]: "
              << static_cast<double>(time.count()) / 1000000.0 /
                     logger->get_num_iterations()
              << std::endl;
}
```

Chapter 28

The nine-pt-stencil-solver program

The 9-point stencil example..

This example depends on simple-solver, three-pt-stencil-solver, poisson-solver.

Introduction

This example solves a 2D Poisson equation:

[\Omega = (0,1)^2 \ \Omega_b = [0,1]^2 \text{ (with boundary)} \ \partial\Omega = \Omega_b \backslash \Omega \ u : \Omega_b -> R \ u'' = f \in \Omega \ u = u_D \in \partial\Omega \ ]

using a finite difference method on an equidistant grid with `K` discretization points (`K` can be controlled with a command line parameter). The discretization may be done by any order Taylor polynomial. For an equidistant grid with K "inner" discretization points ((x1,y1), \ldots, (xk,y1),(x1,y2), \ldots, (xk,yk,z1)) step size (h = 1 / (K + 1)) and a stencil (\in \mathbb{R}^{3 \times 3}), the formula produces a system of linear equations

(\sum_{a,b=-1}^1 stencil(a,b) ∗ u_{(i+a,j+b} = -f_k h^2), on any inner node with a neighborhood of inner nodes

On any node, where neighbor is on the border, the neighbor is replaced with a (-stencil(a,b) ∗ u_{i+a,j+b}) and added to the right hand side vector. For example a node with a neighborhood of only edge nodes may look like this

[\sum_{a,b=-1}^(1,0) stencil(a,b) ∗ u_{(i+a,j+b} = -f_k h^2 - \sum_{a=-1}^1 stencil(a,1) ∗ u_{(i+a,j+1}]

which is then solved using Ginkgo's implementation of the CG method preconditioned with block-Jacobi. It is also possible to specify on which executor Ginkgo will solve the system via the command line. The function `f` is set to (f(x,y) = 6x + 6y) (making the solution (u(x,y) = x^3

- y^3)), but that can be changed in the `main` function. Also the stencil values for the core, the faces, the edge and the corners can be changed when passing additional parameters.

The intention of this is to show how generation of stencil values and the right hand side vector changes when increasing the dimension.

About the example

The commented program

```
/ *****************************<DESCRIPTION>***********************************
This example solves a 2D Poisson equation:
    \Omega = (0,1)^2
    \Omega_b = [0,1]^2   (with boundary)
    \partial\Omega = \Omega_b \backslash \Omega
    u : \Omega_b -> R
    u" = f in \Omega
    u = u_D on \partial\Omega
using a finite difference method on an equidistant grid with `K` discretization
points (`K` can be controlled with a command line parameter). The discretization
may be done by any order Taylor polynomial.
For an equidistant grid with K "inner" discretization points (x1,y1), ...,
(xk,y1),(x1,y2), ..., (xk,yk) step size h = 1 / (K + 1) and a stencil \in
\R^{3 x 3}, the formula produces a system of linear equations
\sum_{a,b=-1}^1 stencil(a,b) * u_{(i+a,j+b} = -f_k h^2,  on any inner node with
a neighborhood of inner nodes
On any node, where neighbor is on the border, the neighbor is replaced with a
'-stencil(a,b) * u_{i+a,j+b}' and added to the right hand side vector. For
example a node with a neighborhood of only edge nodes may look like this
\sum_{a,b=-1}^(1,0) stencil(a,b) * u_{(i+a,j+b} = -f_k h^2 - \sum_{a=-1}^1
stencil(a,1) * u_{(i+a,j+1}
which is then solved using Ginkgo's implementation of the CG method
preconditioned with block-Jacobi. It is also possible to specify on which
executor Ginkgo will solve the system via the command line.
The function `f` is set to `f(x,y) = 6x + 6y` (making the solution `u(x,y) = x^3
+ y^3`), but that can be changed in the `main` function. Also the stencil values
for the core, the faces, the edge and the corners can be changed when passing
additional parameters.

The intention of this is to show how generation of stencil values and the right
hand side vector changes when increasing the dimension.
*****************************<DESCRIPTION>********************************** /

#include <array>
#include <chrono>
#include <ginkgo/ginkgo.hpp>
#include <iostream>
#include <map>
#include <string>
#include <vector>
```

Stencil values. Ordering can be seen in the main function Can also be changed by passing additional parameter when executing

```
constexpr double default_alpha = 10.0 / 3.0;
constexpr double default_beta = -2.0 / 3.0;
constexpr double default_gamma = -1.0 / 6.0;
/ * Possible alternative default values are
 * default_alpha = 8.0;
 * default_beta = -1.0;
 * default_gamma = -1.0;
 * /
```

Creates a stencil matrix in CSR format for the given number of discretization points.

```
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(IndexType dp, IndexType* row_ptrs,
                             IndexType* col_idxs, ValueType* values,
                             ValueType* coefs)
{
    IndexType pos = 0;
    const size_t dp_2 = dp * dp;
    row_ptrs[0] = pos;
    for (IndexType k = 0; k < dp; ++k) {
        for (IndexType i = 0; i < dp; ++i) {
            const size_t index = i + k * dp;
            for (IndexType j = -1; j <= 1; ++j) {
                for (IndexType l = -1; l <= 1; ++l) {
                    const IndexType offset = l + 1 + 3 * (j + 1);
                    if ((k + j) >= 0 && (k + j) < dp && (i + l) >= 0 &&
                        (i + l) < dp) {
                        values[pos] = coefs[offset];
                        col_idxs[pos] = index + l + dp * j;
                        ++pos;
                    }
                }
            }
            row_ptrs[index + 1] = pos;
        }
    }
```

```
}
```

Generates the RHS vector given `f` and the boundary conditions.

```
template <typename Closure, typename ClosureT, typename ValueType,
          typename IndexType>
void generate_rhs(IndexType dp, Closure f, ClosureT u, ValueType* rhs,
                  ValueType* coefs)
{
    const size_t dp_2 = dp * dp;
    const ValueType h = 1.0 / (dp + 1.0);
    for (IndexType i = 0; i < dp; ++i) {
        const auto yi = ValueType(i + 1) * h;
        for (IndexType j = 0; j < dp; ++j) {
            const auto xi = ValueType(j + 1) * h;
            const auto index = i * dp + j;
            rhs[index] = -f(xi, yi) * h * h;
        }
    }
```

Iterating over the edges to add boundary values and adding the overlapping 3x1 to the rhs

```
for (size_t i = 0; i < dp; ++i) {
    const auto xi = ValueType(i + 1) * h;
    const auto index_top = i;
    const auto index_bot = i + dp * (dp - 1);
    rhs[index_top] -= u(xi - h, 0.0) * coefs[0];
    rhs[index_top] -= u(xi, 0.0) * coefs[1];
    rhs[index_top] -= u(xi + h, 0.0) * coefs[2];
    rhs[index_bot] -= u(xi - h, 1.0) * coefs[6];
    rhs[index_bot] -= u(xi, 1.0) * coefs[7];
    rhs[index_bot] -= u(xi + h, 1.0) * coefs[8];
}
for (size_t i = 0; i < dp; ++i) {
    const auto yi = ValueType(i + 1) * h;
    const auto index_left = i * dp;
    const auto index_right = i * dp + (dp - 1);
    rhs[index_left] -= u(0.0, yi - h) * coefs[0];
    rhs[index_left] -= u(0.0, yi) * coefs[3];
    rhs[index_left] -= u(0.0, yi + h) * coefs[6];
    rhs[index_right] -= u(1.0, yi - h) * coefs[2];
    rhs[index_right] -= u(1.0, yi) * coefs[5];
    rhs[index_right] -= u(1.0, yi + h) * coefs[8];
}
```

remove the double corner values

```
    rhs[0] += u(0.0, 0.0) * coefs[0];
    rhs[(dp - 1)] += u(1.0, 0.0) * coefs[2];
    rhs[(dp - 1) * dp] += u(0.0, 1.0) * coefs[6];
    rhs[dp * dp - 1] += u(1.0, 1.0) * coefs[8];
}
```

Prints the solution `u`.

```
template <typename ValueType, typename IndexType>
void print_solution(IndexType dp, const ValueType* u)
{
    for (IndexType i = 0; i < dp; ++i) {
        for (IndexType j = 0; j < dp; ++j) {
            std::cout << u[i * dp + j] << ' ';
        }
        std::cout << '\n';
    }
    std::cout << std::endl;
}
```

Computes the 1-norm of the error given the computed `u` and the correct solution function `correct_u`.

```
template <typename Closure, typename ValueType, typename IndexType>
gko::remove_complex<ValueType> calculate_error(IndexType dp, const ValueType* u,
                                               Closure correct_u)
{
    const ValueType h = 1.0 / (dp + 1);
    gko::remove_complex<ValueType> error = 0.0;
    for (IndexType j = 0; j < dp; ++j) {
        const auto xi = ValueType(j + 1) * h;
        for (IndexType i = 0; i < dp; ++i) {
            using std::abs;
            const auto yi = ValueType(i + 1) * h;
            error +=
                abs(u[i * dp + j] - correct_u(xi, yi)) / abs(correct_u(xi, yi));
        }
    }
    return error;
}
template <typename ValueType, typename IndexType>
```

```
void solve_system(const std::string& executor_string,
                  unsigned int discretization_points, IndexType* row_ptrs,
                  IndexType* col_idxs, ValueType* values, ValueType* rhs,
                  ValueType* u, gko::remove_complex<ValueType> reduction_factor)
{
```

Some shortcuts

```
using vec = gko::matrix::Dense<ValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
using val_array = gko::Array<ValueType>;
using idx_array = gko::Array<IndexType>;
const auto& dp = discretization_points;
const gko::size_type dp_2 = dp * dp;
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()>>
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

executor where the application initialized the data

```
const auto app_exec = exec->get_master();
```

Tell Ginkgo to use the data in our application

Matrix: we have to set the executor of the matrix to the one where we want SpMVs to run (in this case `exec`). When creating array views, we have to specify the executor where the data is (in this case `app_exec`).

If the two do not match, Ginkgo will automatically create a copy of the data on `exec` (however, it will not copy the data back once it is done

- here this is not important since we are not modifying the matrix).

```
auto matrix = mtx::create(
    exec, gko::dim<2>(dp_2),
    val_array::view(app_exec, (3 * dp - 2) * (3 * dp - 2), values),
    idx_array::view(app_exec, (3 * dp - 2) * (3 * dp - 2), col_idxs),
    idx_array::view(app_exec, dp_2 + 1, row_ptrs));
```

RHS: similar to matrix

```
auto b = vec::create(exec, gko::dim<2>(dp_2, 1),
                     val_array::view(app_exec, dp_2, rhs), 1);
```

Solution: we have to be careful here - if the executors are different, once we compute the solution the array will not be automatically copied back to the original memory locations. Fortunately, whenever `apply` is called on a linear operator (e.g. matrix, solver) the arguments automatically get copied to the executor where the operator is, and copied back once the operation is completed. Thus, in this case, we can just define the solution on `app_exec`, and it will be automatically transferred to/from `exec` if needed.

```
auto x = vec::create(app_exec, gko::dim<2>(dp_2, 1),
                     val_array::view(app_exec, dp_2, u), 1);
```

Generate solver

```
auto solver_gen =
```

```
    cg::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(dp_2).on(exec),
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec))
        .with_preconditioner(bj::build().on(exec))
        .on(exec);
auto solver = solver_gen->generate(gko::give(matrix));
```

Solve system

```
    solver->apply(gko::lend(b), gko::lend(x));
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && std::string(argv[1]) == "--help") {
    std::cerr
        « "Usage: " « argv[0]
        « " [executor] [DISCRETIZATION_POINTS] [alpha] [beta] [gamma]"
        « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
const IndexType discretization_points =
    argc >= 3 ? std::atoi(argv[2]) : 100;
const ValueType alpha_c = argc >= 4 ? std::atof(argv[3]) : default_alpha;
const ValueType beta_c = argc >= 5 ? std::atof(argv[4]) : default_beta;
const ValueType gamma_c = argc >= 6 ? std::atof(argv[5]) : default_gamma;
```

clang-format off

```
std::array<ValueType, 9> coefs{
    gamma_c, beta_c, gamma_c,
        beta_c, alpha_c, beta_c,
    gamma_c, beta_c, gamma_c};
```

clang-format on

```
const auto dp = discretization_points;
const size_t dp_2 = dp * dp;
```

problem:

```
auto correct_u = [](ValueType x, ValueType y) {
    return x * x * x + y * y * y;
};
auto f = [](ValueType x, ValueType y) {
    return ValueType(6) * x + ValueType(6) * y;
};
```

matrix

```
std::vector<IndexType> row_ptrs(dp_2 + 1);
std::vector<IndexType> col_idxs((3 * dp - 2) * (3 * dp - 2));
std::vector<ValueType> values((3 * dp - 2) * (3 * dp - 2));
```

right hand side

```
std::vector<ValueType> rhs(dp_2);
```

solution

```
std::vector<ValueType> u(dp_2, 0.0);
generate_stencil_matrix(dp, row_ptrs.data(), col_idxs.data(), values.data(),
                        coefs.data());
```

looking for solution u = x^3: f = 6x, u(0) = 0, u(1) = 1

```
generate_rhs(dp, f, correct_u, rhs.data(), coefs.data());
const gko::remove_complex<ValueType> reduction_factor = 1e-7;
auto start_time = std::chrono::steady_clock::now();
solve_system(executor_string, dp, row_ptrs.data(), col_idxs.data(),
             values.data(), rhs.data(), u.data(), reduction_factor);
auto stop_time = std::chrono::steady_clock::now();
auto runtime_duration =
    static_cast<double>(
        std::chrono::duration_cast<std::chrono::nanoseconds>(stop_time -
                                                             start_time)
            .count()) *
    1e-6;
```

Uncomment to print the solution print_solution(dp, u.data());

```
    std::cout « "The average relative error is "
              « calculate_error(dp, u.data(), correct_u) /
                    static_cast<gko::remove_complex<ValueType>>(dp_2)
              « std::endl;
    std::cout « "The runtime is " « std::to_string(runtime_duration) « " ms"
              « std::endl;
}
```

Results

The expected output should be

```
The average relative error is 6.35715e-06
The runtime is 167.320520 ms
```

Comments about programming and debugging

The plain program

```
/*****************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
/******************************<DESCRIPTION>*********************************
This example solves a 2D Poisson equation:

    \Omega = (0,1)^2
    \Omega_b = [0,1]^2   (with boundary)
    \partial\Omega = \Omega_b \backslash \Omega
    u : \Omega_b -> R
    u" = f in \Omega
    u = u_D on \partial\Omega

using a finite difference method on an equidistant grid with `K` discretization
points (`K` can be controlled with a command line parameter). The discretization
may be done by any order Taylor polynomial.
For an equidistant grid with K "inner" discretization points (x1,y1), ...,
(xk,y1),(x1,y2), ..., (xk,yk) step size h = 1 / (K + 1) and a stencil \in
\R^{3 x 3}, the formula produces a system of linear equations

\sum_{a,b=-1}^1 stencil(a,b) * u_{(i+a,j+b} = -f_k h^2,  on any inner node with
a neighborhood of inner nodes

On any node, where neighbor is on the border, the neighbor is replaced with a
'-stencil(a,b) * u_{i+a,j+b}' and added to the right hand side vector. For
example a node with a neighborhood of only edge nodes may look like this

\sum_{a,b=-1}^(1,0) stencil(a,b) * u_{(i+a,j+b} = -f_k h^2 - \sum_{a=-1}^1
stencil(a,1) * u_{(i+a,j+1}

which is then solved using Ginkgo's implementation of the CG method
preconditioned with block-Jacobi. It is also possible to specify on which
executor Ginkgo will solve the system via the command line.
The function `f` is set to `f(x,y) = 6x + 6y` (making the solution `u(x,y) = x^3
+ y^3`), but that can be changed in the `main` function. Also the stencil values
for the core, the faces, the edge and the corners can be changed when passing
additional parameters.

The intention of this is to show how generation of stencil values and the right
hand side vector changes when increasing the dimension.
*****************************<DESCRIPTION>**********************************/
```

```
#include <array>
#include <chrono>
#include <ginkgo/ginkgo.hpp>
#include <iostream>
#include <map>
#include <string>
#include <vector>
constexpr double default_alpha = 10.0 / 3.0;
constexpr double default_beta = -2.0 / 3.0;
constexpr double default_gamma = -1.0 / 6.0;
/* Possible alternative default values are
 * default_alpha = 8.0;
 * default_beta = -1.0;
 * default_gamma = -1.0;
 */
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(IndexType dp, IndexType* row_ptrs,
                             IndexType* col_idxs, ValueType* values,
                             ValueType* coefs)
{
    IndexType pos = 0;
    const size_t dp_2 = dp * dp;
    row_ptrs[0] = pos;
    for (IndexType k = 0; k < dp; ++k) {
        for (IndexType i = 0; i < dp; ++i) {
            const size_t index = i + k * dp;
            for (IndexType j = -1; j <= 1; ++j) {
                for (IndexType l = -1; l <= 1; ++l) {
                    const IndexType offset = l + 1 + 3 * (j + 1);
                    if ((k + j) >= 0 && (k + j) < dp && (i + l) >= 0 &&
                        (i + l) < dp) {
                        values[pos] = coefs[offset];
                        col_idxs[pos] = index + l + dp * j;
                        ++pos;
                    }
                }
            }
            row_ptrs[index + 1] = pos;
        }
    }
}
template <typename Closure, typename ClosureT, typename ValueType,
          typename IndexType>
void generate_rhs(IndexType dp, Closure f, ClosureT u, ValueType* rhs,
                  ValueType* coefs)
{
    const size_t dp_2 = dp * dp;
    const ValueType h = 1.0 / (dp + 1.0);
    for (IndexType i = 0; i < dp; ++i) {
        const auto yi = ValueType(i + 1) * h;
        for (IndexType j = 0; j < dp; ++j) {
            const auto xi = ValueType(j + 1) * h;
            const auto index = i * dp + j;
            rhs[index] = -f(xi, yi) * h * h;
        }
    }
    for (size_t i = 0; i < dp; ++i) {
        const auto xi = ValueType(i + 1) * h;
        const auto index_top = i;
        const auto index_bot = i + dp * (dp - 1);
        rhs[index_top] -= u(xi - h, 0.0) * coefs[0];
        rhs[index_top] -= u(xi, 0.0) * coefs[1];
        rhs[index_top] -= u(xi + h, 0.0) * coefs[2];
        rhs[index_bot] -= u(xi - h, 1.0) * coefs[6];
        rhs[index_bot] -= u(xi, 1.0) * coefs[7];
        rhs[index_bot] -= u(xi + h, 1.0) * coefs[8];
    }
    for (size_t i = 0; i < dp; ++i) {
        const auto yi = ValueType(i + 1) * h;
        const auto index_left = i * dp;
        const auto index_right = i * dp + (dp - 1);
        rhs[index_left] -= u(0.0, yi - h) * coefs[0];
        rhs[index_left] -= u(0.0, yi) * coefs[3];
        rhs[index_left] -= u(0.0, yi + h) * coefs[6];
        rhs[index_right] -= u(1.0, yi - h) * coefs[2];
        rhs[index_right] -= u(1.0, yi) * coefs[5];
        rhs[index_right] -= u(1.0, yi + h) * coefs[8];
    }
    rhs[0] += u(0.0, 0.0) * coefs[0];
    rhs[(dp - 1)] += u(1.0, 0.0) * coefs[2];
    rhs[(dp - 1) * dp] += u(0.0, 1.0) * coefs[6];
    rhs[dp * dp - 1] += u(1.0, 1.0) * coefs[8];
}
template <typename ValueType, typename IndexType>
void print_solution(IndexType dp, const ValueType* u)
{
    for (IndexType i = 0; i < dp; ++i) {
```

```
        for (IndexType j = 0; j < dp; ++j) {
            std::cout « u[i * dp + j] « ' ';
        }
        std::cout « '\n';
    }
    std::cout « std::endl;
}
template <typename Closure, typename ValueType, typename IndexType>
gko::remove_complex<ValueType> calculate_error(IndexType dp, const ValueType* u,
                                               Closure correct_u)
{
    const ValueType h = 1.0 / (dp + 1);
    gko::remove_complex<ValueType> error = 0.0;
    for (IndexType j = 0; j < dp; ++j) {
        const auto xi = ValueType(j + 1) * h;
        for (IndexType i = 0; i < dp; ++i) {
            using std::abs;
            const auto yi = ValueType(i + 1) * h;
            error +=
                abs(u[i * dp + j] - correct_u(xi, yi)) / abs(correct_u(xi, yi));
        }
    }
    return error;
}
template <typename ValueType, typename IndexType>
void solve_system(const std::string& executor_string,
                  unsigned int discretization_points, IndexType* row_ptrs,
                  IndexType* col_idxs, ValueType* values, ValueType* rhs,
                  ValueType* u, gko::remove_complex<ValueType> reduction_factor)
{
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    using val_array = gko::Array<ValueType>;
    using idx_array = gko::Array<IndexType>;
    const auto& dp = discretization_points;
    const gko::size_type dp_2 = dp * dp;
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    const auto app_exec = exec->get_master();
    auto matrix = mtx::create(
        exec, gko::dim<2>(dp_2),
        val_array::view(app_exec, (3 * dp - 2) * (3 * dp - 2), values),
        idx_array::view(app_exec, (3 * dp - 2) * (3 * dp - 2), col_idxs),
        idx_array::view(app_exec, dp_2 + 1, row_ptrs));
    auto b = vec::create(exec, gko::dim<2>(dp_2, 1),
                         val_array::view(app_exec, dp_2, rhs), 1);
    auto x = vec::create(app_exec, gko::dim<2>(dp_2, 1),
                         val_array::view(app_exec, dp_2, u), 1);
    auto solver_gen =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(dp_2).on(exec),
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec))
            .with_preconditioner(bj::build().on(exec))
            .on(exec);
    auto solver = solver_gen->generate(gko::give(matrix));
    solver->apply(gko::lend(b), gko::lend(x));
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && std::string(argv[1]) == "--help") {
        std::cerr
```

```
            « "Usage: " « argv[0]
            « " [executor] [DISCRETIZATION_POINTS] [alpha] [beta] [gamma]"
            « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    const IndexType discretization_points =
        argc >= 3 ? std::atoi(argv[2]) : 100;
    const ValueType alpha_c = argc >= 4 ? std::atof(argv[3]) : default_alpha;
    const ValueType beta_c = argc >= 5 ? std::atof(argv[4]) : default_beta;
    const ValueType gamma_c = argc >= 6 ? std::atof(argv[5]) : default_gamma;
    std::array<ValueType, 9> coefs{
        gamma_c, beta_c, gamma_c,
        beta_c, alpha_c, beta_c,
        gamma_c, beta_c, gamma_c};
    const auto dp = discretization_points;
    const size_t dp_2 = dp * dp;
    auto correct_u = [](ValueType x, ValueType y) {
        return x * x * x + y * y * y;
    };
    auto f = [](ValueType x, ValueType y) {
        return ValueType(6) * x + ValueType(6) * y;
    };
    std::vector<IndexType> row_ptrs(dp_2 + 1);
    std::vector<IndexType> col_idxs((3 * dp - 2) * (3 * dp - 2));
    std::vector<ValueType> values((3 * dp - 2) * (3 * dp - 2));
    std::vector<ValueType> rhs(dp_2);
    std::vector<ValueType> u(dp_2, 0.0);
    generate_stencil_matrix(dp, row_ptrs.data(), col_idxs.data(), values.data(),
                            coefs.data());
    generate_rhs(dp, f, correct_u, rhs.data(), coefs.data());
    const gko::remove_complex<ValueType> reduction_factor = 1e-7;
    auto start_time = std::chrono::steady_clock::now();
    solve_system(executor_string, dp, row_ptrs.data(), col_idxs.data(),
                 values.data(), rhs.data(), u.data(), reduction_factor);
    auto stop_time = std::chrono::steady_clock::now();
    auto runtime_duration =
        static_cast<double>(
            std::chrono::duration_cast<std::chrono::nanoseconds>(stop_time -
                                                                 start_time)
                .count()) *
        1e-6;
    std::cout « "The average relative error is "
              « calculate_error(dp, u.data(), correct_u) /
                    static_cast<gko::remove_complex<ValueType>>(dp_2)
              « std::endl;
    std::cout « "The runtime is " « std::to_string(runtime_duration) « " ms"
              « std::endl;
}
```

Chapter 29

The papi-logging program

The papi logging example..

This example depends on simple-solver-logging.

Introduction

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <papi.h>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <thread>
namespace {
void papi_add_event(const std::string& event_name, int& eventset)
{
    int code;
    int ret_val = PAPI_event_name_to_code(event_name.c_str(), &code);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_name_to_code()" « std::endl;
        std::exit(-1);
    }
    ret_val = PAPI_add_event(eventset, code);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_name_to_code()" « std::endl;
        std::exit(-1);
    }
}
template <typename T>
std::string to_string(T* ptr)
{
    std::ostringstream os;
    os « reinterpret_cast<gko::uintptr>(ptr);
    return os.str();
}
}  // namespace
int init_papi_counters(std::string solver_name, std::string A_name)
{
```

Initialize PAPI, add events and start it up

```
    int eventset = PAPI_NULL;
    int ret_val = PAPI_library_init(PAPI_VER_CURRENT);
    if (ret_val != PAPI_VER_CURRENT) {
        std::cerr « "Error at PAPI_library_init()" « std::endl;
        std::exit(-1);
    }
```

```
    ret_val = PAPI_create_eventset(&eventset);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_create_eventset()" « std::endl;
        std::exit(-1);
    }
    std::string simple_apply_string("sde:::ginkgo0::linop_apply_completed::");
    std::string advanced_apply_string(
        "sde:::ginkgo0::linop_advanced_apply_completed::");
    papi_add_event(simple_apply_string + solver_name, eventset);
    papi_add_event(simple_apply_string + A_name, eventset);
    papi_add_event(advanced_apply_string + A_name, eventset);
    ret_val = PAPI_start(eventset);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_start()" « std::endl;
        std::exit(-1);
    }
    return eventset;
}
void print_papi_counters(int eventset)
{
```

Stop PAPI and read the linop_apply_completed event for all of them

```
long long int values[3];
int ret_val = PAPI_stop(eventset, values);
if (PAPI_OK != ret_val) {
    std::cerr « "Error at PAPI_stop()" « std::endl;
    std::exit(-1);
}
PAPI_shutdown();
```

Print all values returned from PAPI

```
    std::cout « "PAPI SDE counters:" « std::endl;
    std::cout « "solver did " « values[0] « " applies." « std::endl;
    std::cout « "A did " « values[1] « " simple applies." « std::endl;
    std::cout « "A did " « values[2] « " advanced applies." « std::endl;
}
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
```

Figure out where to run the code

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
```

Generate solver

```
const RealValueType reduction_factor{1e-7};
auto solver_gen =
    cg::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(20u).on(exec),
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec))
        .on(exec);
auto solver = solver_gen->generate(A);
```

In this example, we split as much as possible the Ginkgo solver/logger and the PAPI interface. Note that the PAPI ginkgo namespaces are of the form sde:::ginkgo<x> where <x> starts from 0 and is incremented with every new PAPI logger.

```
int eventset =
    init_papi_counters(to_string(solver.get()), to_string(A.get()));
```

Create a PAPI logger and add it to relevant LinOps

```
auto logger = gko::log::Papi<ValueType>::create(
    exec, gko::log::Logger::linop_apply_completed_mask |
              gko::log::Logger::linop_advanced_apply_completed_mask);
solver->add_logger(logger);
A->add_logger(logger);
```

Solve system

```
solver->apply(lend(b), lend(x));
```

Stop PAPI event gathering and print the counters

```
print_papi_counters(eventset);
```

Print solution

```
std::cout « "Solution (x): \n";
write(std::cout, lend(x));
```

Calculate residual

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r): \n";
    write(std::cout, lend(res));
}
```

Results

The following is the expected result:

```
PAPI SDE counters:
solver did 1 applies.
A did 20 simple applies.
A did 1 advanced applies.
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
8.87107e-16
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <papi.h>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <thread>
namespace {
void papi_add_event(const std::string& event_name, int& eventset)
{
    int code;
    int ret_val = PAPI_event_name_to_code(event_name.c_str(), &code);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_name_to_code()" « std::endl;
        std::exit(-1);
    }
    ret_val = PAPI_add_event(eventset, code);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_name_to_code()" « std::endl;
        std::exit(-1);
    }
}
template <typename T>
std::string to_string(T* ptr)
{
    std::ostringstream os;
    os « reinterpret_cast<gko::uintptr>(ptr);
    return os.str();
}
}  // namespace
int init_papi_counters(std::string solver_name, std::string A_name)
{
    int eventset = PAPI_NULL;
    int ret_val = PAPI_library_init(PAPI_VER_CURRENT);
    if (ret_val != PAPI_VER_CURRENT) {
        std::cerr « "Error at PAPI_library_init()" « std::endl;
        std::exit(-1);
    }
    ret_val = PAPI_create_eventset(&eventset);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_create_eventset()" « std::endl;
        std::exit(-1);
    }
    std::string simple_apply_string("sde:::ginkgo0::linop_apply_completed::");
    std::string advanced_apply_string(
        "sde:::ginkgo0::linop_advanced_apply_completed::");
    papi_add_event(simple_apply_string + solver_name, eventset);
    papi_add_event(simple_apply_string + A_name, eventset);
    papi_add_event(advanced_apply_string + A_name, eventset);
```

```
    ret_val = PAPI_start(eventset);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_start()" « std::endl;
        std::exit(-1);
    }
    return eventset;
}
void print_papi_counters(int eventset)
{
    long long int values[3];
    int ret_val = PAPI_stop(eventset, values);
    if (PAPI_OK != ret_val) {
        std::cerr « "Error at PAPI_stop()" « std::endl;
        std::exit(-1);
    }
    PAPI_shutdown();
    std::cout « "PAPI SDE counters:" « std::endl;
    std::cout « "solver did " « values[0] « " applies." « std::endl;
    std::cout « "A did " « values[1] « " simple applies." « std::endl;
    std::cout « "A did " « values[2] « " advanced applies." « std::endl;
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    const RealValueType reduction_factor{1e-7};
    auto solver_gen =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(20u).on(exec),
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec))
            .on(exec);
    auto solver = solver_gen->generate(A);
    int eventset =
        init_papi_counters(to_string(solver.get()), to_string(A.get()));
    auto logger = gko::log::Papi<ValueType>::create(
        exec, gko::log::Logger::linop_apply_completed_mask |
                  gko::log::Logger::linop_advanced_apply_completed_mask);
    solver->add_logger(logger);
    A->add_logger(logger);
    solver->apply(lend(b), lend(x));
    print_papi_counters(eventset);
    std::cout « "Solution (x): \n";
    write(std::cout, lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r): \n";
    write(std::cout, lend(res));
```

```
}
```

Chapter 30

The par-ilu-convergence program

The ParILU convergence example..

This example depends on simple-solver.

Introduction

About the example

This example can be used to inspect the convergence behavior of parallel incomplete factorizations. *

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <functional>
#include <iostream>
#include <map>
#include <memory>
#include <string>
const std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    executors{
        {"reference", [] { return gko::ReferenceExecutor::create(); }},
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create());
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create());
         }},
        {"dpcpp", [] {
             return gko::DpcppExecutor::create(0, gko::OmpExecutor::create());
         }}};
template <typename Function>
auto try_generate(Function fun) -> decltype(fun())
{
    decltype(fun()) result;
    try {
        result = fun();
    } catch (const gko::Error& err) {
        std::cerr « "Error: " « err.what() « '\n';
        std::exit(-1);
    }
    return result;
}
template <typename ValueType, typename IndexType>
double compute_ilu_residual_norm(
```

```
    const gko::matrix::Csr<ValueType, IndexType>* residual,
    const gko::matrix::Csr<ValueType, IndexType>* mtx)
{
    gko::matrix_data<ValueType, IndexType> residual_data;
    gko::matrix_data<ValueType, IndexType> mtx_data;
    residual->write(residual_data);
    mtx->write(mtx_data);
    residual_data.ensure_row_major_order();
    mtx_data.ensure_row_major_order();
    auto it = mtx_data.nonzeros.begin();
    double residual_norm{};
    for (auto entry : residual_data.nonzeros) {
        auto ref_row = it->row;
        auto ref_col = it->column;
        if (entry.row == ref_row && entry.column == ref_col) {
            residual_norm += gko::squared_norm(entry.value);
            ++it;
        }
    }
    return std::sqrt(residual_norm);
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
```

print usage message

```
if (argc < 2 || executors.find(argv[1]) == executors.end()) {
    std::cerr « "Usage: executable"
              « " <reference|omp|cuda|hip|dpcpp> [<matrix-file>] "
                 "[<parilu|parilut|paric|parict] [<max-iterations>] "
                 "[<num-repetitions>] [<fill-in-limit>]\n";
    return -1;
}
```

generate executor based on first argument

```
auto exec = try_generate([&] { return executors.at(argv[1])(); });
```

set matrix and preconditioner name with default values

```
std::string matrix = argc < 3 ? "data/A.mtx" : argv[2];
std::string precond = argc < 4 ? "parilu" : argv[3];
int max_iterations = argc < 5 ? 10 : std::stoi(argv[4]);
int num_repetitions = argc < 6 ? 10 : std::stoi(argv[5]);
double limit = argc < 7 ? 2 : std::stod(argv[6]);
```

load matrix file into Csr format

```
    auto mtx = gko::share(try_generate([&] {
        std::ifstream mtx_stream{matrix};
        if (!mtx_stream) {
            throw GKO_STREAM_ERROR("Unable to open matrix file");
        }
        std::cerr « "Reading " « matrix « std::endl;
        return gko::read<gko::matrix::Csr<ValueType, IndexType»(mtx_stream,
                                                                 exec);
    }));
    std::shared_ptr<gko::LinOpFactory> factory;
    std::function<void(int)> set_iterations;
    if (precond == "parilu") {
        factory =
            gko::factorization::ParIlu<ValueType, IndexType>::build().on(exec);
        set_iterations = [&](int it) {
            gko::as<gko::factorization::ParIlu<ValueType, IndexType>::Factory>(
                factory)
                ->get_parameters()
                .iterations = it;
        };
    } else if (precond == "paric") {
        factory =
            gko::factorization::ParIc<ValueType, IndexType>::build().on(exec);
        set_iterations = [&](int it) {
            gko::as<gko::factorization::ParIc<ValueType, IndexType>::Factory>(
                factory)
                ->get_parameters()
                .iterations = it;
        };
    } else if (precond == "parilut") {
        factory = gko::factorization::ParIlut<ValueType, IndexType>::build()
                      .with_fill_in_limit(limit)
                      .on(exec);
        set_iterations = [&](int it) {
            gko::as<gko::factorization::ParIlut<ValueType, IndexType>::Factory>(
                factory)
                ->get_parameters()
```

```
                .iterations = it;
        };
    } else if (precond == "parict") {
        factory = gko::factorization::ParIct<ValueType, IndexType>::build()
                      .with_fill_in_limit(limit)
                      .on(exec);
        set_iterations = [&](int it) {
            gko::as<gko::factorization::ParIct<ValueType, IndexType>::Factory>(
                factory)
                ->get_parameters()
                .iterations = it;
        };
    }
    auto one = gko::initialize<gko::matrix::Dense<ValueType»({1.0}, exec);
    auto minus_one =
        gko::initialize<gko::matrix::Dense<ValueType»({-1.0}, exec);
    for (int it = 1; it <= max_iterations; ++it) {
        set_iterations(it);
        std::cout « it « ';';
        std::vector<long> times;
        std::vector<double> residuals;
        for (int rep = 0; rep < num_repetitions; ++rep) {
            auto tic = std::chrono::high_resolution_clock::now();
            auto result =
                gko::as<gko::Composition<ValueType»(factory->generate(mtx));
            exec->synchronize();
            auto toc = std::chrono::high_resolution_clock::now();
            auto residual = gko::clone(exec, mtx);
            result->get_operators()[0]->apply(lend(one),
                                              lend(result->get_operators()[1]),
                                              lend(minus_one), lend(residual));
            times.push_back(
                std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic)
                    .count());
            residuals.push_back(
                compute_ilu_residual_norm(lend(residual), lend(mtx)));
        }
        for (auto el : times) {
            std::cout « el « ';';
        }
        for (auto el : residuals) {
            std::cout « el « ';';
        }
        std::cout « '\n';
    }
}
```

Results

This is the expected output:

```
Usage: executable <reference|omp|cuda|hip|dpcpp> [<matrix-file>] [<parilu|parilut|paric|parict]
       [<max-iterations>] [<num-repetitions>] [fill-in-limit]
```

When specifying an executor:

```
Reading data/A.mtx
1;71800;10300;8800;8200;8000;7700;7500;7500;7500;7400;1.0331e-14;1.0331e-14;1.0331e-14;1.0331e-14;1.0331e-14;1.0331e-14;1.033
2;15500;9100;13500;9000;8600;8800;8700;8600;8600;8500;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16
3;16500;10200;10100;10100;9900;10000;9800;9800;9900;9900;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e
4;17500;11500;11200;15600;11300;11200;11400;11200;11200;11100;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.1
5;18800;12800;12700;12600;12500;12400;12400;12400;12400;14100;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.1
6;19200;13400;23100;15400;13200;13000;13000;13000;13100;13000;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.1
7;20500;14500;14400;14200;14200;14300;14200;14100;14300;14200;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.1
8;21600;15700;86200;16300;15700;15600;15500;15400;15500;15600;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.1
9;22700;17000;16700;16600;16700;16800;20400;17400;17500;17400;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.1
10;25500;19000;18800;18700;18700;18800;18600;18700;18600;18700;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.15407e-16;4.
```

Comments about programming and debugging

The plain program

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <functional>
#include <iostream>
#include <map>
#include <memory>
#include <string>
const std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    executors{
        {"reference", [] { return gko::ReferenceExecutor::create(); }},
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create());
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create());
         }},
        {"dpcpp", [] {
             return gko::DpcppExecutor::create(0, gko::OmpExecutor::create());
         }}};
template <typename Function>
auto try_generate(Function fun) -> decltype(fun())
{
    decltype(fun()) result;
    try {
        result = fun();
    } catch (const gko::Error& err) {
        std::cerr « "Error: " « err.what() « '\n';
        std::exit(-1);
    }
    return result;
}
template <typename ValueType, typename IndexType>
double compute_ilu_residual_norm(
    const gko::matrix::Csr<ValueType, IndexType>* residual,
    const gko::matrix::Csr<ValueType, IndexType>* mtx)
{
    gko::matrix_data<ValueType, IndexType> residual_data;
    gko::matrix_data<ValueType, IndexType> mtx_data;
    residual->write(residual_data);
    mtx->write(mtx_data);
    residual_data.ensure_row_major_order();
    mtx_data.ensure_row_major_order();
    auto it = mtx_data.nonzeros.begin();
    double residual_norm{};
    for (auto entry : residual_data.nonzeros) {
        auto ref_row = it->row;
        auto ref_col = it->column;
        if (entry.row == ref_row && entry.column == ref_col) {
            residual_norm += gko::squared_norm(entry.value);
            ++it;
        }
    }
    return std::sqrt(residual_norm);
}
int main(int argc, char* argv[])
{
    using ValueType = double;
```

```
using IndexType = int;
if (argc < 2 || executors.find(argv[1]) == executors.end()) {
    std::cerr « "Usage: executable"
              « " <reference|omp|cuda|hip|dpcpp> [<matrix-file>] "
                 "[<parilu|parilut|paric|parict] [<max-iterations>] "
                 "[<num-repetitions>] [<fill-in-limit>]\n";
    return -1;
}
auto exec = try_generate([&] { return executors.at(argv[1])(); });
std::string matrix = argc < 3 ? "data/A.mtx" : argv[2];
std::string precond = argc < 4 ? "parilu" : argv[3];
int max_iterations = argc < 5 ? 10 : std::stoi(argv[4]);
int num_repetitions = argc < 6 ? 10 : std::stoi(argv[5]);
double limit = argc < 7 ? 2 : std::stod(argv[6]);
auto mtx = gko::share(try_generate([&] {
    std::ifstream mtx_stream{matrix};
    if (!mtx_stream) {
        throw GKO_STREAM_ERROR("Unable to open matrix file");
    }
    std::cerr « "Reading " « matrix « std::endl;
    return gko::read<gko::matrix::Csr<ValueType, IndexType»(mtx_stream,
                                                             exec);
}));
std::shared_ptr<gko::LinOpFactory> factory;
std::function<void(int)> set_iterations;
if (precond == "parilu") {
    factory =
        gko::factorization::ParIlu<ValueType, IndexType>::build().on(exec);
    set_iterations = [&](int it) {
        gko::as<gko::factorization::ParIlu<ValueType, IndexType>::Factory>(
            factory)
            ->get_parameters()
            .iterations = it;
    };
} else if (precond == "paric") {
    factory =
        gko::factorization::ParIc<ValueType, IndexType>::build().on(exec);
    set_iterations = [&](int it) {
        gko::as<gko::factorization::ParIc<ValueType, IndexType>::Factory>(
            factory)
            ->get_parameters()
            .iterations = it;
    };
} else if (precond == "parilut") {
    factory = gko::factorization::ParIlut<ValueType, IndexType>::build()
                  .with_fill_in_limit(limit)
                  .on(exec);
    set_iterations = [&](int it) {
        gko::as<gko::factorization::ParIlut<ValueType, IndexType>::Factory>(
            factory)
            ->get_parameters()
            .iterations = it;
    };
} else if (precond == "parict") {
    factory = gko::factorization::ParIct<ValueType, IndexType>::build()
                  .with_fill_in_limit(limit)
                  .on(exec);
    set_iterations = [&](int it) {
        gko::as<gko::factorization::ParIct<ValueType, IndexType>::Factory>(
            factory)
            ->get_parameters()
            .iterations = it;
    };
}
auto one = gko::initialize<gko::matrix::Dense<ValueType»({1.0}, exec);
auto minus_one =
    gko::initialize<gko::matrix::Dense<ValueType»({-1.0}, exec);
for (int it = 1; it <= max_iterations; ++it) {
    set_iterations(it);
    std::cout « it « ';';
    std::vector<long> times;
    std::vector<double> residuals;
    for (int rep = 0; rep < num_repetitions; ++rep) {
        auto tic = std::chrono::high_resolution_clock::now();
        auto result =
            gko::as<gko::Composition<ValueType»(factory->generate(mtx));
        exec->synchronize();
        auto toc = std::chrono::high_resolution_clock::now();
        auto residual = gko::clone(exec, mtx);
        result->get_operators()[0]->apply(lend(one),
                                          lend(result->get_operators()[1]),
                                          lend(minus_one), lend(residual));
        times.push_back(
            std::chrono::duration_cast<std::chrono::nanoseconds>(toc - tic)
                .count());
        residuals.push_back(
            compute_ilu_residual_norm(lend(residual), lend(mtx)));
```

```
        }
        for (auto el : times) {
            std::cout « el « ';';
        }
        for (auto el : residuals) {
            std::cout « el « ';';
        }
        std::cout « '\n';
    }
}
```

Chapter 31

The performance-debugging program

The simple solver with performance debugging example..

This example depends on simple-solver-logging, minimal-cuda-solver.

Introduction

About the example

This example runs a solver on a test problem and shows how to use loggers to debug performance and convergence rate.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <algorithm>
#include <array>
#include <chrono>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <ostream>
#include <sstream>
#include <string>
#include <unordered_map>
#include <utility>
#include <vector>
template <typename ValueType>
using vec = gko::matrix::Dense<ValueType>;
template <typename ValueType>
using real_vec = gko::matrix::Dense<gko::remove_complex<ValueType>>;
namespace utils {
```

creates a zero vector

```
template <typename ValueType>
std::unique_ptr<vec<ValueType>> create_vector(
    std::shared_ptr<const gko::Executor> exec, gko::size_type size,
    ValueType value)
{
    auto res = vec<ValueType>::create(exec);
    res->read(gko::matrix_data<ValueType>(gko::dim<2>{size, 1}, value));
    return res;
}
```

utilities for computing norms and residuals

```
template <typename ValueType>
ValueType get_first_element(const vec<ValueType>* norm)
{
    return norm->get_executor()->copy_val_to_host(norm->get_const_values());
}
template <typename ValueType>
gko::remove_complex<ValueType> compute_norm(const vec<ValueType>* b)
{
    auto exec = b->get_executor();
    auto b_norm = gko::initialize<real_vec<ValueType>>({0.0}, exec);
    b->compute_norm2(gko::lend(b_norm));
    return get_first_element(gko::lend(b_norm));
}
template <typename ValueType>
gko::remove_complex<ValueType> compute_residual_norm(
    const gko::LinOp* system_matrix, const vec<ValueType>* b,
    const vec<ValueType>* x)
{
    auto exec = system_matrix->get_executor();
    auto one = gko::initialize<vec<ValueType>>({1.0}, exec);
    auto neg_one = gko::initialize<vec<ValueType>>({-1.0}, exec);
    auto res = gko::clone(b);
    system_matrix->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one),
                         gko::lend(res));
    return compute_norm(gko::lend(res));
}
}  // namespace utils
namespace loggers {
```

A logger that accumulates the time of all operations. For each operation type (allocations, free, copy, internal operations i.e. kernels), the timing is taken before and after. This can create significant overhead since to ensure proper timings, calls to `synchronize` are required.

```
struct OperationLogger : gko::log::Logger {
    void on_allocation_started(const gko::Executor* exec,
                               const gko::size_type&) const override
    {
        this->start_operation(exec, "allocate");
    }
    void on_allocation_completed(const gko::Executor* exec,
                                 const gko::size_type&,
                                 const gko::uintptr&) const override
    {
        this->end_operation(exec, "allocate");
    }
    void on_free_started(const gko::Executor* exec,
                         const gko::uintptr&) const override
    {
        this->start_operation(exec, "free");
    }
    void on_free_completed(const gko::Executor* exec,
                           const gko::uintptr&) const override
    {
        this->end_operation(exec, "free");
    }
    void on_copy_started(const gko::Executor* from, const gko::Executor* to,
                         const gko::uintptr&, const gko::uintptr&,
                         const gko::size_type&) const override
    {
        from->synchronize();
        this->start_operation(to, "copy");
    }
    void on_copy_completed(const gko::Executor* from, const gko::Executor* to,
                           const gko::uintptr&, const gko::uintptr&,
                           const gko::size_type&) const override
    {
        from->synchronize();
        this->end_operation(to, "copy");
    }
    void on_operation_launched(const gko::Executor* exec,
                               const gko::Operation* op) const override
    {
        this->start_operation(exec, op->get_name());
    }
    void on_operation_completed(const gko::Executor* exec,
                                const gko::Operation* op) const override
    {
        this->end_operation(exec, op->get_name());
    }
    void write_data(std::ostream& ostream)
    {
        for (const auto& entry : total) {
            ostream << "\t" << entry.first.c_str() << ": "
                    << std::chrono::duration_cast<std::chrono::nanoseconds>(
                           entry.second)
                           .count()
```

```
                    « std::endl;
        }
    }
    OperationLogger(std::shared_ptr<const gko::Executor> exec)
        : gko::log::Logger(exec)
    {}
private:
```

Helper which synchronizes and starts the time before every operation.

```
void start_operation(const gko::Executor* exec,
                     const std::string& name) const
{
    nested.emplace_back(0);
    exec->synchronize();
    start[name] = std::chrono::steady_clock::now();
}
```

Helper to compute the end time and store the operation's time at its end. Also time nested operations.

```
void end_operation(const gko::Executor* exec, const std::string& name) const
{
    exec->synchronize();
    const auto end = std::chrono::steady_clock::now();
    const auto diff = end - start[name];
```

make sure timings for nested operations are not counted twice

```
    total[name] += diff - nested.back();
    nested.pop_back();
    if (nested.size() > 0) {
        nested.back() += diff;
    }
}
mutable std::map<std::string, std::chrono::steady_clock::time_point> start;
mutable std::map<std::string, std::chrono::steady_clock::duration> total;
```

the position i of this vector holds the total time spend on child operations on nesting level i

```
    mutable std::vector<std::chrono::steady_clock::duration> nested;
};
```

This logger tracks the persistently allocated data

```
struct StorageLogger : gko::log::Logger {
```

Store amount of bytes allocated on every allocation

```
void on_allocation_completed(const gko::Executor*,
                             const gko::size_type& num_bytes,
                             const gko::uintptr& location) const override
{
    storage[location] = num_bytes;
}
```

Reset the amount of bytes on every free

```
void on_free_completed(const gko::Executor*,
                       const gko::uintptr& location) const override
{
    storage[location] = 0;
}
```

Write the data after summing the total from all allocations

```
    void write_data(std::ostream& ostream)
    {
        gko::size_type total{};
        for (const auto& e : storage) {
            total += e.second;
        }
        ostream « "Storage: " « total « std::endl;
    }
    StorageLogger(std::shared_ptr<const gko::Executor> exec)
        : gko::log::Logger(exec)
    {}
private:
    mutable std::unordered_map<gko::uintptr, gko::size_type> storage;
};
```

Logs true and recurrent residuals of the solver

```
template <typename ValueType>
struct ResidualLogger : gko::log::Logger {
```

Depending on the available information, store the norm or compute it from the residual. If the true residual norm could not be computed, store the value `-1.0`.

```
    void on_iteration_complete(const gko::LinOp*, const gko::size_type&,
                               const gko::LinOp* residual,
                               const gko::LinOp* solution,
                               const gko::LinOp* residual_norm) const override
    {
        if (residual_norm) {
            rec_res_norms.push_back(utils::get_first_element(
                gko::as<real_vec<ValueType>>(residual_norm)));
        } else {
            rec_res_norms.push_back(
                utils::compute_norm(gko::as<vec<ValueType>>(residual)));
        }
        if (solution) {
            true_res_norms.push_back(utils::compute_residual_norm(
                matrix, b, gko::as<vec<ValueType>>(solution)));
        } else {
            true_res_norms.push_back(-1.0);
        }
    }
    ResidualLogger(std::shared_ptr<const gko::Executor> exec,
                   const gko::LinOp* matrix, const vec<ValueType>* b)
        : gko::log::Logger(exec, gko::log::Logger::iteration_complete_mask),
          matrix{matrix},
          b{b}
    {}
    void write_data(std::ostream& ostream)
    {
        ostream << "Recurrent Residual Norms: " << std::endl;
        ostream << "[" << std::endl;
        for (const auto& entry : rec_res_norms) {
            ostream << "\t" << entry << std::endl;
        }
        ostream << "];" << std::endl;
        ostream << "True Residual Norms: " << std::endl;
        ostream << "[" << std::endl;
        for (const auto& entry : true_res_norms) {
            ostream << "\t" << entry << std::endl;
        }
        ostream << "];" << std::endl;
    }
private:
    const gko::LinOp* matrix;
    const vec<ValueType>* b;
    mutable std::vector<gko::remove_complex<ValueType>> rec_res_norms;
    mutable std::vector<gko::remove_complex<ValueType>> true_res_norms;
};
}  // namespace loggers
namespace {
```

Print usage help

```
void print_usage(const char* filename)
{
    std::cerr << "Usage: " << filename << " [executor] [matrix file]"
              << std::endl;
    std::cerr << "matrix file should be a file in matrix market format. "
                 "The file data/A.mtx is provided as an example."
              << std::endl;
    std::exit(-1);
}
template <typename ValueType>
void print_vector(const gko::matrix::Dense<ValueType>* vec)
{
    auto elements_to_print = std::min(gko::size_type(10), vec->get_size()[0]);
    std::cout << "[" << std::endl;
    for (int i = 0; i < elements_to_print; ++i) {
        std::cout << "\t" << vec->at(i) << std::endl;
    }
    std::cout << "];" << std::endl;
}
}  // namespace
int main(int argc, char* argv[])
{
```

Parametrize the benchmark here Pick a value type

```
using ValueType = double;
using IndexType = int;
```

Pick a matrix format

```
using mtx = gko::matrix::Csr<ValueType, IndexType>;
```

Pick a solver

```
using solver = gko::solver::Cg<ValueType>;
```

Pick a preconditioner type

```
using preconditioner = gko::matrix::IdentityFactory<ValueType>;
```

Pick a residual norm reduction value

```
const gko::remove_complex<ValueType> reduction_factor = 1e-12;
```

Pick an output file name

```
const auto of_name = "log.txt";
```

Simple shortcut

```
using vec = gko::matrix::Dense<ValueType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
```

Figure out where to run the code

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read the input matrix file directory

```
std::string input_mtx = "data/A.mtx";
if (argc == 3) {
    input_mtx = std::string(argv[2]);
}
```

Read data: A is read from disk Create a StorageLogger to track the size of A

```
auto storage_logger = std::make_shared<loggers::StorageLogger>(exec);
```

Add the logger to the executor

```
exec->add_logger(storage_logger);
```

Read the matrix A from file

```
auto A = gko::share(gko::read<mtx>(std::ifstream(input_mtx), exec));
```

Remove the storage logger

```
exec->remove_logger(gko::lend(storage_logger));
```

Pick a maximum iteration count

```
const auto max_iters = A->get_size()[0];
```

Generate b and x vectors

```
auto b = utils::create_vector<ValueType>(exec, A->get_size()[0], 1.0);
auto x = utils::create_vector<ValueType>(exec, A->get_size()[0], 0.0);
```

Declare the solver factory. The preconditioner's arguments should be adapted if needed.

```
auto solver_factory =
```

```
    solver::build()
        .with_criteria(
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec),
            gko::stop::Iteration::build().with_max_iters(max_iters).on(
                exec))
        .with_preconditioner(preconditioner::create(exec))
        .on(exec);
```

Declare the output file for all our loggers

```
std::ofstream output_file(of_name);
```

Do a warmup run

```
{
```

Clone x to not overwrite the original one

```
auto x_clone = gko::clone(x);
```

Generate and call apply on a solver

```
    solver_factory->generate(A)->apply(gko::lend(b), gko::lend(x_clone));
    exec->synchronize();
}
```

Do a timed run

```
{
```

Clone x to not overwrite the original one

```
auto x_clone = gko::clone(x);
```

Synchronize ensures no operation are ongoing

```
exec->synchronize();
```

Time before generate

```
auto g_tic = std::chrono::steady_clock::now();
```

Generate a solver

```
auto generated_solver = solver_factory->generate(A);
exec->synchronize();
```

Time after generate

```
auto g_tac = std::chrono::steady_clock::now();
```

Compute the generation time

```
auto generate_time =
    std::chrono::duration_cast<std::chrono::nanoseconds>(g_tac - g_tic);
```

Write the generate time to the output file

```
output_file « "Generate time (ns): " « generate_time.count()
            « std::endl;
```

Similarly time the apply

```
exec->synchronize();
auto a_tic = std::chrono::steady_clock::now();
generated_solver->apply(gko::lend(b), gko::lend(x_clone));
exec->synchronize();
auto a_tac = std::chrono::steady_clock::now();
auto apply_time =
    std::chrono::duration_cast<std::chrono::nanoseconds>(a_tac - a_tic);
output_file « "Apply time (ns): " « apply_time.count() « std::endl;
```

Compute the residual norm

```
    auto residual = utils::compute_residual_norm(gko::lend(A), gko::lend(b),
                                                 gko::lend(x_clone));
    output_file « "Residual_norm: " « residual « std::endl;
}
```

Log the internal operations using the OperationLogger without timing

```
{
```

Create an OperationLogger to analyze the generate step

```
auto gen_logger = std::make_shared<loggers::OperationLogger>(exec);
```

Add the generate logger to the executor
```
exec->add_logger(gen_logger);
```

Generate a solver
```
auto generated_solver = solver_factory->generate(A);
```

Remove the generate logger from the executor
```
exec->remove_logger(gko::lend(gen_logger));
```

Write the data to the output file
```
output_file « "Generate operations times (ns):" « std::endl;
gen_logger->write_data(output_file);
```

Create an OperationLogger to analyze the apply step
```
auto apply_logger = std::make_shared<loggers::OperationLogger>(exec);
exec->add_logger(apply_logger);
```

Create a ResidualLogger to log the recurent residual
```
auto res_logger = std::make_shared<loggers::ResidualLogger<ValueType»(
    exec, gko::lend(A), gko::lend(b));
generated_solver->add_logger(res_logger);
```

Solve the system
```
generated_solver->apply(gko::lend(b), gko::lend(x));
exec->remove_logger(gko::lend(apply_logger));
```

Write the data to the output file
```
    output_file « "Apply operations times (ns):" « std::endl;
    apply_logger->write_data(output_file);
    res_logger->write_data(output_file);
}
```

Print solution
```
std::cout « "Solution, first ten entries: \n";
print_vector(gko::lend(x));
```

Print output file location
```
    std::cout « "The performance and residual data can be found in " « of_name
              « std::endl;
}
```

Results

This is the expected standard output:
```
Solution, first ten entries:
[
    0.252218
    0.108645
    0.0662811
    0.0630433
    0.0384088
    0.0396536
    0.0402648
    0.0338935
    0.0193098
    0.0234653
];
The performance and residual data can be found in log.txt
```

Here is a sample output in the file log.txt:
```
Generate time (ns): 861
Apply time (ns): 108144
Residual_norm: 2.10788e-15
Generate operations times (ns):
Apply operations times (ns):
    allocate: 14991
    cg::initialize#8: 872
    cg::step_1#5: 7683
    cg::step_2#7: 7756
    copy: 7751
    csr::advanced_spmv#5: 21819
    csr::spmv#3: 20429
```

```
    dense::compute_dot#3: 18043
    dense::compute_norm2#2: 16726
    free: 8857
    residual_norm::residual_norm#9: 3614
Recurrent Residual Norms:
[
    4.3589
    2.30455
    1.46771
    0.984875
    0.741833
    0.513623
    0.384165
    0.316439
    0.227709
    0.170312
    0.0973722
    0.0616831
    0.0454123
    0.031953
    0.0161606
    0.00657015
    0.00264367
    0.000858809
    0.000286461
    1.64195e-15
];
True Residual Norms:
[
    4.3589
    2.30455
    1.46771
    0.984875
    0.741833
    0.513623
    0.384165
    0.316439
    0.227709
    0.170312
    0.0973722
    0.0616831
    0.0454123
    0.031953
    0.0161606
    0.00657015
    0.00264367
    0.000858809
    0.000286461
    2.10788e-15
];
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

```
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <algorithm>
#include <array>
#include <chrono>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <ostream>
#include <sstream>
#include <string>
#include <unordered_map>
#include <utility>
#include <vector>
template <typename ValueType>
using vec = gko::matrix::Dense<ValueType>;
template <typename ValueType>
using real_vec = gko::matrix::Dense<gko::remove_complex<ValueType>>;
namespace utils {
template <typename ValueType>
std::unique_ptr<vec<ValueType>> create_vector(
    std::shared_ptr<const gko::Executor> exec, gko::size_type size,
    ValueType value)
{
    auto res = vec<ValueType>::create(exec);
    res->read(gko::matrix_data<ValueType>(gko::dim<2>{size, 1}, value));
    return res;
}
template <typename ValueType>
ValueType get_first_element(const vec<ValueType>* norm)
{
    return norm->get_executor()->copy_val_to_host(norm->get_const_values());
}
template <typename ValueType>
gko::remove_complex<ValueType> compute_norm(const vec<ValueType>* b)
{
    auto exec = b->get_executor();
    auto b_norm = gko::initialize<real_vec<ValueType>>({0.0}, exec);
    b->compute_norm2(gko::lend(b_norm));
    return get_first_element(gko::lend(b_norm));
}
template <typename ValueType>
gko::remove_complex<ValueType> compute_residual_norm(
    const gko::LinOp* system_matrix, const vec<ValueType>* b,
    const vec<ValueType>* x)
{
    auto exec = system_matrix->get_executor();
    auto one = gko::initialize<vec<ValueType>>({1.0}, exec);
    auto neg_one = gko::initialize<vec<ValueType>>({-1.0}, exec);
    auto res = gko::clone(b);
    system_matrix->apply(gko::lend(one), gko::lend(x), gko::lend(neg_one),
                         gko::lend(res));
    return compute_norm(gko::lend(res));
}
}  // namespace utils
namespace loggers {
struct OperationLogger : gko::log::Logger {
    void on_allocation_started(const gko::Executor* exec,
                               const gko::size_type&) const override
    {
        this->start_operation(exec, "allocate");
    }
    void on_allocation_completed(const gko::Executor* exec,
                                 const gko::size_type&,
                                 const gko::uintptr&) const override
    {
        this->end_operation(exec, "allocate");
    }
    void on_free_started(const gko::Executor* exec,
                         const gko::uintptr&) const override
    {
        this->start_operation(exec, "free");
    }
    void on_free_completed(const gko::Executor* exec,
                           const gko::uintptr&) const override
    {
        this->end_operation(exec, "free");
    }
    void on_copy_started(const gko::Executor* from, const gko::Executor* to,
                         const gko::uintptr&, const gko::uintptr&,
                         const gko::size_type&) const override
```

```
    {
        from->synchronize();
        this->start_operation(to, "copy");
    }
    void on_copy_completed(const gko::Executor* from, const gko::Executor* to,
                           const gko::uintptr&, const gko::uintptr&,
                           const gko::size_type&) const override
    {
        from->synchronize();
        this->end_operation(to, "copy");
    }
    void on_operation_launched(const gko::Executor* exec,
                               const gko::Operation* op) const override
    {
        this->start_operation(exec, op->get_name());
    }
    void on_operation_completed(const gko::Executor* exec,
                                const gko::Operation* op) const override
    {
        this->end_operation(exec, op->get_name());
    }
    void write_data(std::ostream& ostream)
    {
        for (const auto& entry : total) {
            ostream « "\t" « entry.first.c_str() « ": "
                    « std::chrono::duration_cast<std::chrono::nanoseconds>(
                           entry.second)
                           .count()
                    « std::endl;
        }
    }
    OperationLogger(std::shared_ptr<const gko::Executor> exec)
        : gko::log::Logger(exec)
    {}
private:
    void start_operation(const gko::Executor* exec,
                         const std::string& name) const
    {
        nested.emplace_back(0);
        exec->synchronize();
        start[name] = std::chrono::steady_clock::now();
    }
    void end_operation(const gko::Executor* exec, const std::string& name) const
    {
        exec->synchronize();
        const auto end = std::chrono::steady_clock::now();
        const auto diff = end - start[name];
        total[name] += diff - nested.back();
        nested.pop_back();
        if (nested.size() > 0) {
            nested.back() += diff;
        }
    }
    mutable std::map<std::string, std::chrono::steady_clock::time_point> start;
    mutable std::map<std::string, std::chrono::steady_clock::duration> total;
    mutable std::vector<std::chrono::steady_clock::duration> nested;
};
struct StorageLogger : gko::log::Logger {
    void on_allocation_completed(const gko::Executor*,
                                 const gko::size_type& num_bytes,
                                 const gko::uintptr& location) const override
    {
        storage[location] = num_bytes;
    }
    void on_free_completed(const gko::Executor*,
                           const gko::uintptr& location) const override
    {
        storage[location] = 0;
    }
    void write_data(std::ostream& ostream)
    {
        gko::size_type total{};
        for (const auto& e : storage) {
            total += e.second;
        }
        ostream « "Storage: " « total « std::endl;
    }
    StorageLogger(std::shared_ptr<const gko::Executor> exec)
        : gko::log::Logger(exec)
    {}
private:
    mutable std::unordered_map<gko::uintptr, gko::size_type> storage;
};
template <typename ValueType>
struct ResidualLogger : gko::log::Logger {
    void on_iteration_complete(const gko::LinOp*, const gko::size_type&,
                               const gko::LinOp* residual,
```

```
                                const gko::LinOp* solution,
                                const gko::LinOp* residual_norm) const override
    {
        if (residual_norm) {
            rec_res_norms.push_back(utils::get_first_element(
                gko::as<real_vec<ValueType»(residual_norm)));
        } else {
            rec_res_norms.push_back(
                utils::compute_norm(gko::as<vec<ValueType»(residual)));
        }
        if (solution) {
            true_res_norms.push_back(utils::compute_residual_norm(
                matrix, b, gko::as<vec<ValueType»(solution)));
        } else {
            true_res_norms.push_back(-1.0);
        }
    }
    ResidualLogger(std::shared_ptr<const gko::Executor> exec,
                   const gko::LinOp* matrix, const vec<ValueType>* b)
        : gko::log::Logger(exec, gko::log::Logger::iteration_complete_mask),
          matrix{matrix},
          b{b}
    {}
    void write_data(std::ostream& ostream)
    {
        ostream « "Recurrent Residual Norms: " « std::endl;
        ostream « "[" « std::endl;
        for (const auto& entry : rec_res_norms) {
            ostream « "\t" « entry « std::endl;
        }
        ostream « "];" « std::endl;
        ostream « "True Residual Norms: " « std::endl;
        ostream « "[" « std::endl;
        for (const auto& entry : true_res_norms) {
            ostream « "\t" « entry « std::endl;
        }
        ostream « "];" « std::endl;
    }
private:
    const gko::LinOp* matrix;
    const vec<ValueType>* b;
    mutable std::vector<gko::remove_complex<ValueType» rec_res_norms;
    mutable std::vector<gko::remove_complex<ValueType» true_res_norms;
};
}  // namespace loggers
namespace {
void print_usage(const char* filename)
{
    std::cerr « "Usage: " « filename « " [executor] [matrix file]"
              « std::endl;
    std::cerr « "matrix file should be a file in matrix market format. "
                 "The file data/A.mtx is provided as an example."
              « std::endl;
    std::exit(-1);
}
template <typename ValueType>
void print_vector(const gko::matrix::Dense<ValueType>* vec)
{
    auto elements_to_print = std::min(gko::size_type(10), vec->get_size()[0]);
    std::cout « "[" « std::endl;
    for (int i = 0; i < elements_to_print; ++i) {
        std::cout « "\t" « vec->at(i) « std::endl;
    }
    std::cout « "];" « std::endl;
}
}  // namespace
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using solver = gko::solver::Cg<ValueType>;
    using preconditioner = gko::matrix::IdentityFactory<ValueType>;
    const gko::remove_complex<ValueType> reduction_factor = 1e-12;
    const auto of_name = "log.txt";
    using vec = gko::matrix::Dense<ValueType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
```

```
                return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
            }},
           {"hip",
            [] {
                return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                true);
            }},
           {"dpcpp",
            [] {
                return gko::DpcppExecutor::create(0,
                                                  gko::OmpExecutor::create());
            }},
           {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    std::string input_mtx = "data/A.mtx";
    if (argc == 3) {
        input_mtx = std::string(argv[2]);
    }
    auto storage_logger = std::make_shared<loggers::StorageLogger>(exec);
    exec->add_logger(storage_logger);
    auto A = gko::share(gko::read<mtx>(std::ifstream(input_mtx), exec));
    exec->remove_logger(gko::lend(storage_logger));
    const auto max_iters = A->get_size()[0];
    auto b = utils::create_vector<ValueType>(exec, A->get_size()[0], 1.0);
    auto x = utils::create_vector<ValueType>(exec, A->get_size()[0], 0.0);
    auto solver_factory =
        solver::build()
            .with_criteria(
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec),
                gko::stop::Iteration::build().with_max_iters(max_iters).on(
                    exec))
            .with_preconditioner(preconditioner::create(exec))
            .on(exec);
    std::ofstream output_file(of_name);
    {
        auto x_clone = gko::clone(x);
        solver_factory->generate(A)->apply(gko::lend(b), gko::lend(x_clone));
        exec->synchronize();
    }
    {
        auto x_clone = gko::clone(x);
        exec->synchronize();
        auto g_tic = std::chrono::steady_clock::now();
        auto generated_solver = solver_factory->generate(A);
        exec->synchronize();
        auto g_tac = std::chrono::steady_clock::now();
        auto generate_time =
            std::chrono::duration_cast<std::chrono::nanoseconds>(g_tac - g_tic);
        output_file « "Generate time (ns): " « generate_time.count()
                    « std::endl;
        exec->synchronize();
        auto a_tic = std::chrono::steady_clock::now();
        generated_solver->apply(gko::lend(b), gko::lend(x_clone));
        exec->synchronize();
        auto a_tac = std::chrono::steady_clock::now();
        auto apply_time =
            std::chrono::duration_cast<std::chrono::nanoseconds>(a_tac - a_tic);
        output_file « "Apply time (ns): " « apply_time.count() « std::endl;
        auto residual = utils::compute_residual_norm(gko::lend(A), gko::lend(b),
                                                     gko::lend(x_clone));
        output_file « "Residual_norm: " « residual « std::endl;
    }
    {
        auto gen_logger = std::make_shared<loggers::OperationLogger>(exec);
        exec->add_logger(gen_logger);
        auto generated_solver = solver_factory->generate(A);
        exec->remove_logger(gko::lend(gen_logger));
        output_file « "Generate operations times (ns):" « std::endl;
        gen_logger->write_data(output_file);
        auto apply_logger = std::make_shared<loggers::OperationLogger>(exec);
        exec->add_logger(apply_logger);
        auto res_logger = std::make_shared<loggers::ResidualLogger<ValueType»(
            exec, gko::lend(A), gko::lend(b));
        generated_solver->add_logger(res_logger);
        generated_solver->apply(gko::lend(b), gko::lend(x));
        exec->remove_logger(gko::lend(apply_logger));
        output_file « "Apply operations times (ns):" « std::endl;
        apply_logger->write_data(output_file);
        res_logger->write_data(output_file);
    }
    std::cout « "Solution, first ten entries: \n";
    print_vector(gko::lend(x));
    std::cout « "The performance and residual data can be found in " « of_name
              « std::endl;
```

```
}
```

Chapter 32

The poisson-solver program

The poisson solver example..

This example depends on simple-solver.

Introduction

This example solves a 1D Poisson equation:

$u : [0, 1] \to R$
$u" = f$
$u(0) = u0$
$u(1) = u1$

using a finite difference method on an equidistant grid with `K` discretization points (`K` can be controlled with a command line parameter). The discretization is done via the second order Taylor polynomial:

$u(x + h) = u(x) - u'(x)h + 1/2u"(x)h^2 + O(h^3)$
$u(x - h) = u(x) + u'(x)h + 1/2u"(x)h^2 + O(h^3)/ +$
$- -$
$-u(x - h) + 2u(x) + -u(x + h) = -f(x)h^2 + O(h^3)$

For an equidistant grid with K "inner" discretization points $x1, ..., xk,$and step size $h = 1/(K + 1)$, the formula produces a system of linear equations

$2u_1 - u_2 = -f_1h^2 + u0$
$-u_(k - 1) + 2u_k - u_(k + 1) = -f_kh^2, k = 2, ..., K - 1$
$-u_(K - 1) + 2u_K = -f_Kh^2 + u1$

which is then solved using Ginkgo's implementation of the CG method preconditioned with block-Jacobi. It is also possible to specify on which executor Ginkgo will solve the system via the command line. The function 'f'is set to '$f(x) = 6x$' (making the solution '$u(x) = x^3$'), but that can be changed in the `main` function.

The intention of the example is to show how Ginkgo can be used to build an application solving a real-world problem, which includes a solution of a large, sparse linear system as a component.

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <iostream>
#include <map>
#include <string>
#include <vector>
```

Creates a stencil matrix in CSR format for the given number of discretization points.

```
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(gko::matrix::Csr<ValueType, IndexType>* matrix)
{
    const auto discretization_points = matrix->get_size()[0];
    auto row_ptrs = matrix->get_row_ptrs();
    auto col_idxs = matrix->get_col_idxs();
    auto values = matrix->get_values();
    int pos = 0;
    const ValueType coefs[] = {-1, 2, -1};
    row_ptrs[0] = pos;
    for (int i = 0; i < discretization_points; ++i) {
        for (auto ofs : {-1, 0, 1}) {
            if (0 <= i + ofs && i + ofs < discretization_points) {
                values[pos] = coefs[ofs + 1];
                col_idxs[pos] = i + ofs;
                ++pos;
            }
        }
        row_ptrs[i + 1] = pos;
    }
}
```

Generates the RHS vector given `f` and the boundary conditions.

```
template <typename Closure, typename ValueType>
void generate_rhs(Closure f, ValueType u0, ValueType u1,
                  gko::matrix::Dense<ValueType>* rhs)
{
    const auto discretization_points = rhs->get_size()[0];
    auto values = rhs->get_values();
    const ValueType h = 1.0 / static_cast<ValueType>(discretization_points + 1);
    for (gko::size_type i = 0; i < discretization_points; ++i) {
        const auto xi = static_cast<ValueType>(i + 1) * h;
        values[i] = -f(xi) * h * h;
    }
    values[0] += u0;
    values[discretization_points - 1] += u1;
}
```

Prints the solution `u`.

```
template <typename Closure, typename ValueType>
void print_solution(ValueType u0, ValueType u1,
                    const gko::matrix::Dense<ValueType>* u)
{
    std::cout « u0 « '\n';
    for (int i = 0; i < u->get_size()[0]; ++i) {
        std::cout « u->get_const_values()[i] « '\n';
    }
    std::cout « u1 « std::endl;
}
```

Computes the 1-norm of the error given the computed `u` and the correct solution function `correct_u`.

```
template <typename Closure, typename ValueType>
gko::remove_complex<ValueType> calculate_error(
    int discretization_points, const gko::matrix::Dense<ValueType>* u,
    Closure correct_u)
{
    const ValueType h = 1.0 / static_cast<ValueType>(discretization_points + 1);
    gko::remove_complex<ValueType> error = 0.0;
    for (int i = 0; i < discretization_points; ++i) {
        using std::abs;
        const auto xi = static_cast<ValueType>(i + 1) * h;
        error +=
            abs(u->get_const_values()[i] - correct_u(xi)) / abs(correct_u(xi));
    }
    return error;
}
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0]
              « " [executor] [DISCRETIZATION_POINTS]" « std::endl;
    std::exit(-1);
}
```

Get number of discretization points

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
const unsigned int discretization_points =
    argc >= 3 ? std::atoi(argv[2]) : 100;
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

executor used by the application

```
const auto app_exec = exec->get_master();
```

problem:

```
auto correct_u = [](ValueType x) { return x * x * x; };
auto f = [](ValueType x) { return ValueType(6) * x; };
auto u0 = correct_u(0);
auto u1 = correct_u(1);
```

initialize matrix and vectors

```
auto matrix = mtx::create(app_exec, gko::dim<2>(discretization_points),
                          3 * discretization_points - 2);
generate_stencil_matrix(lend(matrix));
auto rhs = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
generate_rhs(f, u0, u1, lend(rhs));
auto u = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
for (int i = 0; i < u->get_size()[0]; ++i) {
    u->get_values()[i] = 0.0;
}
const gko::remove_complex<ValueType> reduction_factor = 1e-7;
```

Generate solver and solve the system

```
cg::build()
    .with_criteria(gko::stop::Iteration::build()
                       .with_max_iters(discretization_points)
                       .on(exec),
                   gko::stop::ResidualNorm<ValueType>::build()
                       .with_reduction_factor(reduction_factor)
                       .on(exec))
    .with_preconditioner(bj::build().on(exec))
    .on(exec)
    ->generate(clone(exec, matrix))  // copy the matrix to the executor
    ->apply(lend(rhs), lend(u));
```

Uncomment to print the solution print_solution<ValueType>(u0, u1, lend(u));

```
    std::cout « "Solve complete.\nThe average relative error is "
              « calculate_error(discretization_points, lend(u), correct_u) /
                    static_cast<gko::remove_complex<ValueType>>(
                        discretization_points)
              « std::endl;
}
```

Results

This is the expected output:

```
Solve complete.
The average relative error is 2.52236e-11
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <iostream>
#include <map>
#include <string>
#include <vector>
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(gko::matrix::Csr<ValueType, IndexType>* matrix)
{
    const auto discretization_points = matrix->get_size()[0];
    auto row_ptrs = matrix->get_row_ptrs();
    auto col_idxs = matrix->get_col_idxs();
    auto values = matrix->get_values();
    int pos = 0;
    const ValueType coefs[] = {-1, 2, -1};
    row_ptrs[0] = pos;
    for (int i = 0; i < discretization_points; ++i) {
        for (auto ofs : {-1, 0, 1}) {
            if (0 <= i + ofs && i + ofs < discretization_points) {
                values[pos] = coefs[ofs + 1];
                col_idxs[pos] = i + ofs;
                ++pos;
            }
        }
        row_ptrs[i + 1] = pos;
    }
}
template <typename Closure, typename ValueType>
void generate_rhs(Closure f, ValueType u0, ValueType u1,
                  gko::matrix::Dense<ValueType>* rhs)
{
    const auto discretization_points = rhs->get_size()[0];
    auto values = rhs->get_values();
    const ValueType h = 1.0 / static_cast<ValueType>(discretization_points + 1);
    for (gko::size_type i = 0; i < discretization_points; ++i) {
        const auto xi = static_cast<ValueType>(i + 1) * h;
        values[i] = -f(xi) * h * h;
    }
    values[0] += u0;
```

```
    values[discretization_points - 1] += u1;
}
template <typename Closure, typename ValueType>
void print_solution(ValueType u0, ValueType u1,
                    const gko::matrix::Dense<ValueType>* u)
{
    std::cout « u0 « '\n';
    for (int i = 0; i < u->get_size()[0]; ++i) {
        std::cout « u->get_const_values()[i] « '\n';
    }
    std::cout « u1 « std::endl;
}
template <typename Closure, typename ValueType>
gko::remove_complex<ValueType> calculate_error(
    int discretization_points, const gko::matrix::Dense<ValueType>* u,
    Closure correct_u)
{
    const ValueType h = 1.0 / static_cast<ValueType>(discretization_points + 1);
    gko::remove_complex<ValueType> error = 0.0;
    for (int i = 0; i < discretization_points; ++i) {
        using std::abs;
        const auto xi = static_cast<ValueType>(i + 1) * h;
        error +=
            abs(u->get_const_values()[i] - correct_u(xi)) / abs(correct_u(xi));
    }
    return error;
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0]
                  « " [executor] [DISCRETIZATION_POINTS]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    const unsigned int discretization_points =
        argc >= 3 ? std::atoi(argv[2]) : 100;
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    const auto app_exec = exec->get_master();
    auto correct_u = [](ValueType x) { return x * x * x; };
    auto f = [](ValueType x) { return ValueType(6) * x; };
    auto u0 = correct_u(0);
    auto u1 = correct_u(1);
    auto matrix = mtx::create(app_exec, gko::dim<2>(discretization_points),
                              3 * discretization_points - 2);
    generate_stencil_matrix(lend(matrix));
    auto rhs = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
    generate_rhs(f, u0, u1, lend(rhs));
    auto u = vec::create(app_exec, gko::dim<2>(discretization_points, 1));
    for (int i = 0; i < u->get_size()[0]; ++i) {
        u->get_values()[i] = 0.0;
    }
    const gko::remove_complex<ValueType> reduction_factor = 1e-7;
    cg::build()
        .with_criteria(gko::stop::Iteration::build()
                           .with_max_iters(discretization_points)
                           .on(exec),
                       gko::stop::ResidualNorm<ValueType>::build()
                           .with_reduction_factor(reduction_factor)
                           .on(exec))
        .with_preconditioner(bj::build().on(exec))
```

```
        .on(exec)
        ->generate(clone(exec, matrix))  // copy the matrix to the executor
        ->apply(lend(rhs), lend(u));
    std::cout « "Solve complete.\nThe average relative error is "
              « calculate_error(discretization_points, lend(u), correct_u) /
                     static_cast<gko::remove_complex<ValueType>>(
                         discretization_points)
              « std::endl;
}
```

Chapter 33

The preconditioned-solver program

The preconditioned solver example..

This example depends on simple-solver.

Introduction

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
```

Figure out where to run the code

```
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
```

Figure out where to run the code

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
```

```
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
const RealValueType reduction_factor{1e-7};
```

Create solver factory

```
auto solver_gen =
    cg::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(20u).on(exec),
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec))
```

Add preconditioner, these 2 lines are the only difference from the simple solver example

```
.with_preconditioner(bj::build().with_max_block_size(8u).on(exec))
.on(exec);
```

Create solver

```
auto solver = solver_gen->generate(A);
```

Solve system

```
solver->apply(lend(b), lend(x));
```

Print solution

```
std::cout « "Solution (x):\n";
write(std::cout, lend(x));
```

Calculate residual

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
}
```

Results

This is the expected output:

```
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
4.82005e-08
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    const RealValueType reduction_factor{1e-7};
    auto solver_gen =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(20u).on(exec),
```

```
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec))
            .with_preconditioner(bj::build().with_max_block_size(8u).on(exec))
            .on(exec);
    auto solver = solver_gen->generate(A);
    solver->apply(lend(b), lend(x));
    std::cout « "Solution (x):\n";
    write(std::cout, lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
}
```

Chapter 34

The preconditioner-export program

The preconditioner export example..

This example depends on simple-solver.

Introduction

About the example

This example shows how to explicitly generate and store preconditioners for a given matrix. It can also be used to inspect and debug the preconditioner generation.

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <functional>
#include <iostream>
#include <map>
#include <memory>
#include <string>
const std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    executors{{"reference", [] { return gko::ReferenceExecutor::create(); }},
              {"omp", [] { return gko::OmpExecutor::create(); }},
              {"cuda",
               [] {
                   return gko::CudaExecutor::create(
                       0, gko::ReferenceExecutor::create());
               }},
              {"hip",
               [] {
                   return gko::HipExecutor::create(
                       0, gko::ReferenceExecutor::create());
               }},
              {"dpcpp", [] {
                   return gko::DpcppExecutor::create(
                       0, gko::ReferenceExecutor::create());
               }}};
void output(const gko::WritableToMatrixData<double, int>* mtx, std::string name)
{
    std::ofstream stream{name};
    std::cerr « "Writing " « name « std::endl;
    gko::write(stream, mtx, gko::layout_type::coordinate);
}
template <typename Function>
auto try_generate(Function fun) -> decltype(fun())
{
    decltype(fun()) result;
```

```
    try {
        result = fun();
    } catch (const gko::Error& err) {
        std::cerr « "Error: " « err.what() « '\n';
        std::exit(-1);
    }
    return result;
}
int main(int argc, char* argv[])
{
```

print usage message

```
if (argc < 2 || executors.find(argv[1]) == executors.end()) {
    std::cerr « "Usage: executable"
              « " <reference|omp|cuda|hip|dpcpp> [<matrix-file>] "
                 "[<jacobi|ilu|parilu|parilut|ilu-isai|parilu-isai|parilut-"
                 "isai] [<preconditioner args>]\n";
    std::cerr « "Jacobi parameters: [<max-block-size>] [<accuracy>] "
                 "[<storage-optimization:auto|0|1|2>]\n";
    std::cerr « "ParILU parameters: [<iteration-count>]\n";
    std::cerr
        « "ParILUT parameters: [<iteration-count>] [<fill-in-limit>]\n";
    std::cerr « "ILU-ISAI parameters: [<sparsity-power>]\n";
    std::cerr « "ParILU-ISAI parameters: [<iteration-count>] "
                 "[<sparsity-power>]\n";
    std::cerr « "ParILUT-ISAI parameters: [<iteration-count>] "
                 "[<fill-in-limit>] [<sparsity-power>]\n";
    return -1;
}
```

generate executor based on first argument

```
auto exec = try_generate([&] { return executors.at(argv[1])(); });
```

set matrix and preconditioner name with default values

```
std::string matrix = argc < 3 ? "data/A.mtx" : argv[2];
std::string precond = argc < 4 ? "jacobi" : argv[3];
```

load matrix file into Csr format

```
auto mtx = gko::share(try_generate([&] {
    std::ifstream mtx_stream{matrix};
    if (!mtx_stream) {
        throw GKO_STREAM_ERROR("Unable to open matrix file");
    }
    std::cerr « "Reading " « matrix « std::endl;
    return gko::read<gko::matrix::Csr<»(mtx_stream, exec);
}));
```

concatenate remaining arguments for filename

```
std::string output_suffix;
for (auto i = 4; i < argc; ++i) {
    output_suffix = output_suffix + "-" + argv[i];
}
```

handle different preconditioners

```
if (precond == "jacobi") {
```

jacobi: max_block_size, accuracy, storage_optimization

```
    auto factory = gko::preconditioner::Jacobi<>::build().on(exec);
    if (argc >= 5) {
        factory->get_parameters().max_block_size = std::stoi(argv[4]);
    }
    if (argc >= 6) {
        factory->get_parameters().accuracy = std::stod(argv[5]);
    }
    if (argc >= 7) {
        factory->get_parameters().storage_optimization =
            std::string{argv[6]} == "auto"
                ? gko::precision_reduction::autodetect()
                : gko::precision_reduction(0, std::stoi(argv[6]));
    }
    auto jacobi = try_generate([&] { return factory->generate(mtx); });
    output(jacobi.get(), matrix + ".jacobi" + output_suffix);
} else if (precond == "ilu") {
```

ilu: no parameters

```
    auto ilu = gko::as<gko::Composition<»(try_generate([&] {
        return gko::factorization::Ilu<>::build().on(exec)->generate(mtx);
    }));
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[0].get()),
           matrix + ".ilu-l");
```

```
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[1].get()),
           matrix + ".ilu-u");
} else if (precond == "parilu") {
```

parilu: iterations

```
    auto factory = gko::factorization::ParIlu<>::build().on(exec);
    if (argc >= 5) {
        factory->get_parameters().iterations = std::stoi(argv[4]);
    }
    auto ilu = gko::as<gko::Composition<»(
        try_generate([&] { return factory->generate(mtx); }));
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[0].get()),
           matrix + ".parilu" + output_suffix + "-l");
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[1].get()),
           matrix + ".parilu" + output_suffix + "-u");
} else if (precond == "parilut") {
```

parilut: iterations, fill-in limit

```
    auto factory = gko::factorization::ParIlut<>::build().on(exec);
    if (argc >= 5) {
        factory->get_parameters().iterations = std::stoi(argv[4]);
    }
    if (argc >= 6) {
        factory->get_parameters().fill_in_limit = std::stod(argv[5]);
    }
    auto ilut = gko::as<gko::Composition<»(
        try_generate([&] { return factory->generate(mtx); }));
    output(gko::as<gko::matrix::Csr<>>(ilut->get_operators()[0].get()),
           matrix + ".parilut" + output_suffix + "-l");
    output(gko::as<gko::matrix::Csr<>>(ilut->get_operators()[1].get()),
           matrix + ".parilut" + output_suffix + "-u");
} else if (precond == "ilu-isai") {
```

ilu-isai: sparsity power

```
    auto fact_factory =
        gko::share(gko::factorization::Ilu<>::build().on(exec));
    int sparsity_power = 1;
    if (argc >= 5) {
        sparsity_power = std::stoi(argv[4]);
    }
    auto factory =
        gko::preconditioner::Ilu<gko::preconditioner::LowerIsai<>,
                                 gko::preconditioner::UpperIsai<>>::build()
            .with_factorization_factory(fact_factory)
            .with_l_solver_factory(gko::preconditioner::LowerIsai<>::build()
                                       .with_sparsity_power(sparsity_power)
                                       .on(exec))
            .with_u_solver_factory(gko::preconditioner::UpperIsai<>::build()
                                       .with_sparsity_power(sparsity_power)
                                       .on(exec))
            .on(exec);
    auto ilu_isai = try_generate([&] { return factory->generate(mtx); });
    output(ilu_isai->get_l_solver()->get_approximate_inverse().get(),
           matrix + ".ilu-isai" + output_suffix + "-l");
    output(ilu_isai->get_u_solver()->get_approximate_inverse().get(),
           matrix + ".ilu-isai" + output_suffix + "-u");
} else if (precond == "parilu-isai") {
```

parilu-isai: iterations, sparsity power

```
    auto fact_factory =
        gko::share(gko::factorization::ParIlu<>::build().on(exec));
    int sparsity_power = 1;
    if (argc >= 5) {
        fact_factory->get_parameters().iterations = std::stoi(argv[4]);
    }
    if (argc >= 6) {
        sparsity_power = std::stoi(argv[5]);
    }
    auto factory =
        gko::preconditioner::Ilu<gko::preconditioner::LowerIsai<>,
                                 gko::preconditioner::UpperIsai<>>::build()
            .with_factorization_factory(fact_factory)
            .with_l_solver_factory(gko::preconditioner::LowerIsai<>::build()
                                       .with_sparsity_power(sparsity_power)
                                       .on(exec))
            .with_u_solver_factory(gko::preconditioner::UpperIsai<>::build()
                                       .with_sparsity_power(sparsity_power)
                                       .on(exec))
            .on(exec);
    auto ilu_isai = try_generate([&] { return factory->generate(mtx); });
    output(ilu_isai->get_l_solver()->get_approximate_inverse().get(),
           matrix + ".parilu-isai" + output_suffix + "-l");
    output(ilu_isai->get_u_solver()->get_approximate_inverse().get(),
```

```
            matrix + ".parilu-isai" + output_suffix + "-u");
} else if (precond == "parilut-isai") {
```

parilut-isai: iterations, fill-in limit, sparsity power

```
        auto fact_factory =
            gko::share(gko::factorization::ParIlut<>::build().on(exec));
        int sparsity_power = 1;
        if (argc >= 5) {
            fact_factory->get_parameters().iterations = std::stoi(argv[4]);
        }
        if (argc >= 6) {
            fact_factory->get_parameters().fill_in_limit = std::stod(argv[5]);
        }
        if (argc >= 7) {
            sparsity_power = std::stoi(argv[6]);
        }
        auto factory =
            gko::preconditioner::Ilu<gko::preconditioner::LowerIsai<>,
                                     gko::preconditioner::UpperIsai<>>::build()
                .with_factorization_factory(fact_factory)
                .with_l_solver_factory(gko::preconditioner::LowerIsai<>::build()
                                           .with_sparsity_power(sparsity_power)
                                           .on(exec))
                .with_u_solver_factory(gko::preconditioner::UpperIsai<>::build()
                                           .with_sparsity_power(sparsity_power)
                                           .on(exec))
                .on(exec);
        auto ilu_isai = try_generate([&] { return factory->generate(mtx); });
        output(ilu_isai->get_l_solver()->get_approximate_inverse().get(),
               matrix + ".parilut-isai" + output_suffix + "-l");
        output(ilu_isai->get_u_solver()->get_approximate_inverse().get(),
               matrix + ".parilut-isai" + output_suffix + "-u");
    }
}
```

Results

This is the expected output:

```
Usage: ./preconditioner-export <reference|omp|cuda|hip|dpcpp> [<matrix-file>]
       [<jacobi|ilu|parilu|parilut|ilu-isai|parilu-isai|parilut-isai] [<preconditioner args>]
Jacobi parameters: [<max-block-size>] [<accuracy>] [<storage-optimization:auto|0|1|2>]
ParILU parameters: [<iteration-count>]
ParILUT parameters: [<iteration-count>] [<fill-in-limit>]
ILU-ISAI parameters: [<sparsity-power>]
ParILU-ISAI parameters: [<iteration-count>] [<sparsity-power>]
ParILUT-ISAI parameters: [<iteration-count>] [<fill-in-limit>] [<sparsity-power>]
```

When specifying an executor:

```
Reading data/A.mtx
Writing data/A.mtx.jacobi
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <functional>
#include <iostream>
#include <map>
#include <memory>
#include <string>
const std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    executors{{"reference", [] { return gko::ReferenceExecutor::create(); }},
              {"omp", [] { return gko::OmpExecutor::create(); }},
              {"cuda",
               [] {
                   return gko::CudaExecutor::create(
                       0, gko::ReferenceExecutor::create());
               }},
              {"hip",
               [] {
                   return gko::HipExecutor::create(
                       0, gko::ReferenceExecutor::create());
               }},
              {"dpcpp", [] {
                   return gko::DpcppExecutor::create(
                       0, gko::ReferenceExecutor::create());
               }}};
void output(const gko::WritableToMatrixData<double, int>* mtx, std::string name)
{
    std::ofstream stream{name};
    std::cerr « "Writing " « name « std::endl;
    gko::write(stream, mtx, gko::layout_type::coordinate);
}
template <typename Function>
auto try_generate(Function fun) -> decltype(fun())
{
    decltype(fun()) result;
    try {
        result = fun();
    } catch (const gko::Error& err) {
        std::cerr « "Error: " « err.what() « '\n';
        std::exit(-1);
    }
    return result;
}
int main(int argc, char* argv[])
{
    if (argc < 2 || executors.find(argv[1]) == executors.end()) {
        std::cerr « "Usage: executable"
                  « " <reference|omp|cuda|hip|dpcpp> [<matrix-file>] "
                    "[<jacobi|ilu|parilu|parilut|ilu-isai|parilu-isai|parilut-"
                    "isai] [<preconditioner args>]\n";
        std::cerr « "Jacobi parameters: [<max-block-size>] [<accuracy>] "
                    "[<storage-optimization:auto|0|1|2>]\n";
        std::cerr « "ParILU parameters: [<iteration-count>]\n";
        std::cerr
            « "ParILUT parameters: [<iteration-count>] [<fill-in-limit>]\n";
        std::cerr « "ILU-ISAI parameters: [<sparsity-power>]\n";
        std::cerr « "ParILU-ISAI parameters: [<iteration-count>] "
                    "[<sparsity-power>]\n";
        std::cerr « "ParILUT-ISAI parameters: [<iteration-count>] "
                    "[<fill-in-limit>] [<sparsity-power>]\n";
        return -1;
    }
    auto exec = try_generate([&] { return executors.at(argv[1])(); });
    std::string matrix = argc < 3 ? "data/A.mtx" : argv[2];
    std::string precond = argc < 4 ? "jacobi" : argv[3];
    auto mtx = gko::share(try_generate([&] {
        std::ifstream mtx_stream{matrix};
        if (!mtx_stream) {
            throw GKO_STREAM_ERROR("Unable to open matrix file");
        }
        std::cerr « "Reading " « matrix « std::endl;
        return gko::read<gko::matrix::Csr<»(mtx_stream, exec);
    }));
    std::string output_suffix;
    for (auto i = 4; i < argc; ++i) {
        output_suffix = output_suffix + "-" + argv[i];
```

```
}
if (precond == "jacobi") {
    auto factory = gko::preconditioner::Jacobi<>::build().on(exec);
    if (argc >= 5) {
        factory->get_parameters().max_block_size = std::stoi(argv[4]);
    }
    if (argc >= 6) {
        factory->get_parameters().accuracy = std::stod(argv[5]);
    }
    if (argc >= 7) {
        factory->get_parameters().storage_optimization =
            std::string{argv[6]} == "auto"
                ? gko::precision_reduction::autodetect()
                : gko::precision_reduction(0, std::stoi(argv[6]));
    }
    auto jacobi = try_generate([&] { return factory->generate(mtx); });
    output(jacobi.get(), matrix + ".jacobi" + output_suffix);
} else if (precond == "ilu") {
    auto ilu = gko::as<gko::Composition<»(try_generate([&] {
        return gko::factorization::Ilu<>::build().on(exec)->generate(mtx);
    }));
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[0].get()),
           matrix + ".ilu-l");
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[1].get()),
           matrix + ".ilu-u");
} else if (precond == "parilu") {
    auto factory = gko::factorization::ParIlu<>::build().on(exec);
    if (argc >= 5) {
        factory->get_parameters().iterations = std::stoi(argv[4]);
    }
    auto ilu = gko::as<gko::Composition<»(
        try_generate([&] { return factory->generate(mtx); }));
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[0].get()),
           matrix + ".parilu" + output_suffix + "-l");
    output(gko::as<gko::matrix::Csr<>>(ilu->get_operators()[1].get()),
           matrix + ".parilu" + output_suffix + "-u");
} else if (precond == "parilut") {
    auto factory = gko::factorization::ParIlut<>::build().on(exec);
    if (argc >= 5) {
        factory->get_parameters().iterations = std::stoi(argv[4]);
    }
    if (argc >= 6) {
        factory->get_parameters().fill_in_limit = std::stod(argv[5]);
    }
    auto ilut = gko::as<gko::Composition<»(
        try_generate([&] { return factory->generate(mtx); }));
    output(gko::as<gko::matrix::Csr<>>(ilut->get_operators()[0].get()),
           matrix + ".parilut" + output_suffix + "-l");
    output(gko::as<gko::matrix::Csr<>>(ilut->get_operators()[1].get()),
           matrix + ".parilut" + output_suffix + "-u");
} else if (precond == "ilu-isai") {
    auto fact_factory =
        gko::share(gko::factorization::Ilu<>::build().on(exec));
    int sparsity_power = 1;
    if (argc >= 5) {
        sparsity_power = std::stoi(argv[4]);
    }
    auto factory =
        gko::preconditioner::Ilu<gko::preconditioner::LowerIsai<>,
                                 gko::preconditioner::UpperIsai<>>::build()
            .with_factorization_factory(fact_factory)
            .with_l_solver_factory(gko::preconditioner::LowerIsai<>::build()
                                       .with_sparsity_power(sparsity_power)
                                       .on(exec))
            .with_u_solver_factory(gko::preconditioner::UpperIsai<>::build()
                                       .with_sparsity_power(sparsity_power)
                                       .on(exec))
            .on(exec);
    auto ilu_isai = try_generate([&] { return factory->generate(mtx); });
    output(ilu_isai->get_l_solver()->get_approximate_inverse().get(),
           matrix + ".ilu-isai" + output_suffix + "-l");
    output(ilu_isai->get_u_solver()->get_approximate_inverse().get(),
           matrix + ".ilu-isai" + output_suffix + "-u");
} else if (precond == "parilu-isai") {
    auto fact_factory =
        gko::share(gko::factorization::ParIlu<>::build().on(exec));
    int sparsity_power = 1;
    if (argc >= 5) {
        fact_factory->get_parameters().iterations = std::stoi(argv[4]);
    }
    if (argc >= 6) {
        sparsity_power = std::stoi(argv[5]);
    }
    auto factory =
        gko::preconditioner::Ilu<gko::preconditioner::LowerIsai<>,
                                 gko::preconditioner::UpperIsai<>>::build()
            .with_factorization_factory(fact_factory)
```

```
                .with_l_solver_factory(gko::preconditioner::LowerIsai<>::build()
                                           .with_sparsity_power(sparsity_power)
                                           .on(exec))
                .with_u_solver_factory(gko::preconditioner::UpperIsai<>::build()
                                           .with_sparsity_power(sparsity_power)
                                           .on(exec))
                .on(exec);
        auto ilu_isai = try_generate([&] { return factory->generate(mtx); });
        output(ilu_isai->get_l_solver()->get_approximate_inverse().get(),
               matrix + ".parilu-isai" + output_suffix + "-l");
        output(ilu_isai->get_u_solver()->get_approximate_inverse().get(),
               matrix + ".parilu-isai" + output_suffix + "-u");
    } else if (precond == "parilut-isai") {
        auto fact_factory =
            gko::share(gko::factorization::ParIlut<>::build().on(exec));
        int sparsity_power = 1;
        if (argc >= 5) {
            fact_factory->get_parameters().iterations = std::stoi(argv[4]);
        }
        if (argc >= 6) {
            fact_factory->get_parameters().fill_in_limit = std::stod(argv[5]);
        }
        if (argc >= 7) {
            sparsity_power = std::stoi(argv[6]);
        }
        auto factory =
            gko::preconditioner::Ilu<gko::preconditioner::LowerIsai<>,
                                     gko::preconditioner::UpperIsai<>>::build()
                .with_factorization_factory(fact_factory)
                .with_l_solver_factory(gko::preconditioner::LowerIsai<>::build()
                                           .with_sparsity_power(sparsity_power)
                                           .on(exec))
                .with_u_solver_factory(gko::preconditioner::UpperIsai<>::build()
                                           .with_sparsity_power(sparsity_power)
                                           .on(exec))
                .on(exec);
        auto ilu_isai = try_generate([&] { return factory->generate(mtx); });
        output(ilu_isai->get_l_solver()->get_approximate_inverse().get(),
               matrix + ".parilut-isai" + output_suffix + "-l");
        output(ilu_isai->get_u_solver()->get_approximate_inverse().get(),
               matrix + ".parilut-isai" + output_suffix + "-u");
    }
}
```

Chapter 35

The schroedinger-splitting program

The Schroedinger equation example..

This example depends on heat-equation.

Introduction

This example shows how to use the FFT and iFFT implementations in Ginkgo to solve the non-linear Schrödinger equation with a splitting method.

The non-linear Schrödinger equation (NLS) is given by

$i\partial_t \theta = -\delta\theta + |\theta|^2\theta$

Here θ is the wave function of a single particle in two dimensions. Its magnitude $|\theta|^2$ describes the probability distribution of the particle's position.

This equation can be split in to its linear (1) and non-linear (2) part

$$\begin{array}{rl} (1) & i\partial_t\theta = -\delta\theta \\ (2) & i\partial_t\theta = |\theta|^2\theta \end{array}$$

For both of these equations, we can compute exact solutions, assuming periodic boundary conditions and using the Fourier series expansion for (1) and using the fact that $|\theta|^2$ is constant in (2):

$$\begin{array}{rl} (\hat{1}) & \partial_t\hat{\theta}_k = -i|k|^2\theta \\ (2') & \partial_t|\theta|^2 = i|\theta|^2(\theta - \theta) = 0 \end{array}$$

The exact solutions are then given by

$$\begin{array}{rl} (\hat{1}) & \hat{\theta}(t) = e^{-i|k|^2 t}\hat{\theta}(0) \\ (2') & \theta(t) = e^{-i|\theta|^2 t}\theta(0) \end{array}$$

These partial solutions can be used to approximate a solution to the full NLS by alternating between small time steps for (1) and (2).

For nicer visual results, we add another constant potential term V(x) \theta to the non-linear part, which turns it into the Gross–Pitaevskii equation.

About the example

The commented program

```
/ *****************************<DESCRIPTION>**********************************
This example shows how to use the FFT and iFFT implementations in Ginkgo
to solve the non-linear Schrödinger equation with a splitting method.
The non-linear Schrödinger equation (NLS) is given by
    i \partial_t \theta = -\delta \theta + |\theta|^2 \theta
Here \theta is the wave function of a single particle in two dimensions.
Its magnitude |\theta|^2 describes the probability distribution of the
particle's position.

This equation can be split in to its linear (1) and non-linear (2) part

\f{align*}{
    (1) \quad i \partial_t \theta &= -\delta \theta\\
    (2) \quad i \partial_t \theta &= |\theta|^2 \theta
\f}

For both of these equations, we can compute exact solutions, assuming periodic
boundary conditions and using the Fourier series expansion for (1) and using the
fact that | \theta |^2 is constant in (2):

\f{align*}{
    (\hat 1) \quad \quad \partial_t \hat\theta_k &= -i |k|^2 \theta \\
    (2') \quad \partial_t |\theta|^2 &= i |\theta|^2 (\theta - \theta) = 0
\f}
The exact solutions are then given by
\f{align*}{
    (\hat 1) \quad \hat\theta(t) &= e^{-i |k|^2 t} \hat\theta(0)\\
    (2') \quad \theta(t) &= e^{-i |\theta|^2 t} \theta(0)
\f}

These partial solutions can be used to approximate a solution to the full NLS
by alternating between small time steps for (1) and (2).

For nicer visual results, we add another constant potential term V(x) \theta
to the non-linear part, which turns it into the Gross-Pitaevskii equation.

****************************<DESCRIPTION>********************************** /

#include <ginkgo/ginkgo.hpp>

#include <algorithm>
#include <chrono>
#include <fstream>
#include <iostream>
#include <utility>

#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
```

This function implements a simple Ginkgo-themed clamped color mapping for values in the range [0,5].

```
void set_val(unsigned char* data, double value)
{
```

RGB values for the 6 colors used for values 0, 1, ..., 5 We will interpolate linearly between these values.

```
double col_r[] = {255, 221, 129, 201, 249, 255};
double col_g[] = {255, 220, 130, 161, 158, 204};
double col_b[] = {255, 220, 133, 93, 24, 8};
value = std::max(0.0, value);
auto i = std::max(0, std::min(4, int(value)));
auto d = std::max(0.0, std::min(1.0, value - i));
```

OpenCV uses BGR instead of RGB by default, revert indices

```
    data[2] = static_cast<unsigned char>(col_r[i + 1] * d + col_r[i] * (1 - d));
    data[1] = static_cast<unsigned char>(col_g[i + 1] * d + col_g[i] * (1 - d));
    data[0] = static_cast<unsigned char>(col_b[i + 1] * d + col_b[i] * (1 - d));
}
```

Initialize video output with given dimension and FPS (frames per seconds)

```
std::pair<cv::VideoWriter, cv::Mat> build_output(int n, double fps)
{
    cv::Size videosize{n, n};
    auto output =
        std::make_pair(cv::VideoWriter{}, cv::Mat{videosize, CV_8UC3});
    auto fourcc = cv::VideoWriter::fourcc('a', 'v', 'c', '1');
    output.first.open("nls.mp4", fourcc, fps, videosize);
    return output;
```

```
}
```

Write the current frame to video output using the above color mapping

```
void output_timestep(std::pair<cv::VideoWriter, cv::Mat>& output, int n,
                     const std::complex<double>* data)
{
    for (int i = 0; i < n; i++) {
        auto row = output.second.ptr(i);
        for (int j = 0; j < n; j++) {
            set_val(&row[3 * j], abs(data[i * n + j]));
        }
    }
    output.first.write(output.second);
}
int main(int argc, char* argv[])
{
    using vec = gko::matrix::Dense<std::complex<double>>;
    using real_vec = gko::matrix::Dense<double>;
    using fft2 = gko::matrix::Fft2;
```

Problem parameters: simulation length

```
const auto t0 = 15.0;
```

scaling factor for non-linearity

```
const auto nonlinear_scale = 1.0;
```

scaling factor for potential

```
const auto potential_scale = 3.0;
```

Simulation parameters: time scaling factor

```
const auto time_scale = 0.25;
```

number of grid points in each dimension

```
const auto n = 256;
```

number of simulation steps per second

```
const auto steps_per_sec = 1000;
```

number of video frames per second

```
const auto fps = 25;
```

number of grid points

```
const auto n2 = n * n;
```

phase difference between neighboring grid points

```
const auto h = 2.0 * gko::pi<double>() / n;
const auto h2 = h * h;
```

time step size for the simulation

```
const auto tau = 1.0 / steps_per_sec;
const auto idx = [&](int i, int j) { return i * n + j; };
```

create an OpenMP executor

```
auto exec = gko::OmpExecutor::create();
```

load initial state vector

```
std::ifstream initial_stream("data/gko_logo_2d.mtx");
std::ifstream potential_stream("data/gko_text_2d.mtx");
auto amplitude = gko::read<vec>(initial_stream, exec);
auto potential = gko::read<real_vec>(potential_stream, exec);
```

create vector for frequency space representation

```
auto frequency = vec::create(exec, amplitude->get_size());
```

create Fourier matrix

```
auto fft = fft2::create(exec, n, n);
auto ifft = fft->conj_transpose();
```

prepare video output

```
auto output = build_output(n, fps);
```

time stamp of the last output frame (sentinel value)
```
double last_t = -t0;
```

execute splitting method: time step in linear part, then non-linear part
```
for (double t = 0; t < t0; t += tau) {
```

if enough time has passed, output the next frame
```
if (t - last_t > 1.0 / fps) {
    last_t = t;
    std::cout « t « std::endl;
    output_timestep(output, n, amplitude->get_const_values());
}
```

time step in linear part
```
fft->apply(lend(amplitude), lend(frequency));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        frequency->at(idx(i, j)) *=
            std::polar(1.0, -h2 * (i * i + j * j) * tau * time_scale);
```

scale by FFT∗iFFT normalization factor
```
        frequency->at(idx(i, j)) *= 1.0 / n2;
    }
}
ifft->apply(lend(frequency), lend(amplitude));
```

time step in non-linear part
```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                amplitude->at(idx(i, j)) *= std::polar(
                    1.0, -(nonlinear_scale *
                               gko::squared_norm(amplitude->at(idx(i, j))) +
                           potential_scale * potential->at(idx(i, j))) *
                             tau * time_scale);
            }
        }
    }
}
```

Results

The program will generate a video file named nls.mp4 and output the timestamp of each generated frame.

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```

```
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
/*****************************<DESCRIPTION>***********************************
This example shows how to use the FFT and iFFT implementations in Ginkgo
to solve the non-linear Schrödinger equation with a splitting method.

The non-linear Schrödinger equation (NLS) is given by

@f$
    i \partial_t \theta = -\delta \theta + |\theta|^2 \theta
@f$

Here @f$\theta@f$ is the wave function of a single particle in two dimensions.
Its magnitude @f$|\theta|^2@f$ describes the probability distribution of the
particle's position.

This equation can be split in to its linear (1) and non-linear (2) part

\f{align*}{
    (1) \quad i \partial_t \theta &= -\delta \theta\\
    (2) \quad i \partial_t \theta &= |\theta|^2 \theta
\f}

For both of these equations, we can compute exact solutions, assuming periodic
boundary conditions and using the Fourier series expansion for (1) and using the
fact that @f$| \theta |^2@f$ is constant in (2):

\f{align*}{
    (\hat 1) \quad \quad \partial_t \hat\theta_k &= -i |k|^2 \theta \\
    (2') \quad \partial_t |\theta|^2 &= i |\theta|^2 (\theta - \theta) = 0
\f}

The exact solutions are then given by

\f{align*}{
    (\hat 1) \quad \hat\theta(t) &= e^{-i |k|^2 t} \hat\theta(0)\\
    (2') \quad \theta(t) &= e^{-i |\theta|^2 t} \theta(0)
\f}

These partial solutions can be used to approximate a solution to the full NLS
by alternating between small time steps for (1) and (2).

For nicer visual results, we add another constant potential term V(x) \theta
to the non-linear part, which turns it into the Gross-Pitaevskii equation.

*****************************<DESCRIPTION>**********************************/
#include <ginkgo/ginkgo.hpp>
#include <algorithm>
#include <chrono>
#include <fstream>
#include <iostream>
#include <utility>
#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
void set_val(unsigned char* data, double value)
{
    double col_r[] = {255, 221, 129, 201, 249, 255};
    double col_g[] = {255, 220, 130, 161, 158, 204};
    double col_b[] = {255, 220, 133, 93, 24, 8};
    value = std::max(0.0, value);
    auto i = std::max(0, std::min(4, int(value)));
    auto d = std::max(0.0, std::min(1.0, value - i));
    data[2] = static_cast<unsigned char>(col_r[i + 1] * d + col_r[i] * (1 - d));
    data[1] = static_cast<unsigned char>(col_g[i + 1] * d + col_g[i] * (1 - d));
    data[0] = static_cast<unsigned char>(col_b[i + 1] * d + col_b[i] * (1 - d));
}
std::pair<cv::VideoWriter, cv::Mat> build_output(int n, double fps)
{
    cv::Size videosize{n, n};
    auto output =
        std::make_pair(cv::VideoWriter{}, cv::Mat{videosize, CV_8UC3});
    auto fourcc = cv::VideoWriter::fourcc('a', 'v', 'c', '1');
    output.first.open("nls.mp4", fourcc, fps, videosize);
    return output;
}
void output_timestep(std::pair<cv::VideoWriter, cv::Mat>& output, int n,
                     const std::complex<double>* data)
{
    for (int i = 0; i < n; i++) {
        auto row = output.second.ptr(i);
        for (int j = 0; j < n; j++) {
            set_val(&row[3 * j], abs(data[i * n + j]));
```

```
        }
    }
    output.first.write(output.second);
}
int main(int argc, char* argv[])
{
    using vec = gko::matrix::Dense<std::complex<double>>;
    using real_vec = gko::matrix::Dense<double>;
    using fft2 = gko::matrix::Fft2;
    const auto t0 = 15.0;
    const auto nonlinear_scale = 1.0;
    const auto potential_scale = 3.0;
    const auto time_scale = 0.25;
    const auto n = 256;
    const auto steps_per_sec = 1000;
    const auto fps = 25;
    const auto n2 = n * n;
    const auto h = 2.0 * gko::pi<double>() / n;
    const auto h2 = h * h;
    const auto tau = 1.0 / steps_per_sec;
    const auto idx = [&](int i, int j) { return i * n + j; };
    auto exec = gko::OmpExecutor::create();
    std::ifstream initial_stream("data/gko_logo_2d.mtx");
    std::ifstream potential_stream("data/gko_text_2d.mtx");
    auto amplitude = gko::read<vec>(initial_stream, exec);
    auto potential = gko::read<real_vec>(potential_stream, exec);
    auto frequency = vec::create(exec, amplitude->get_size());
    auto fft = fft2::create(exec, n, n);
    auto ifft = fft->conj_transpose();
    auto output = build_output(n, fps);
    double last_t = -t0;
    for (double t = 0; t < t0; t += tau) {
        if (t - last_t > 1.0 / fps) {
            last_t = t;
            std::cout « t « std::endl;
            output_timestep(output, n, amplitude->get_const_values());
        }
        fft->apply(lend(amplitude), lend(frequency));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                frequency->at(idx(i, j)) *=
                    std::polar(1.0, -h2 * (i * i + j * j) * tau * time_scale);
                frequency->at(idx(i, j)) *= 1.0 / n2;
            }
        }
        ifft->apply(lend(frequency), lend(amplitude));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                amplitude->at(idx(i, j)) *= std::polar(
                    1.0, -(nonlinear_scale *
                               gko::squared_norm(amplitude->at(idx(i, j))) +
                           potential_scale * potential->at(idx(i, j))) *
                             tau * time_scale);
            }
        }
    }
}
```

Chapter 36

The simple-solver program

The simple solver example..

Introduction

This simple solver example should help you get started with Ginkgo. This example is meant for you to understand how Ginkgo works and how you can solve a simple linear system with Ginkgo. We encourage you to play with the code, change the parameters and see what is best suited for your purposes.

About the example

Each example has the following sections:

1. **Introduction:**This gives an overview of the example and mentions any interesting aspects in the example that might help the reader.
2. **The commented program:** This section is intended for you to understand the details of the example so that you can play with it and understand Ginkgo and its features better.
3. **Results:** This section shows the results of the code when run. Though the results may not be completely the same, you can expect the behaviour to be similar.
4. **The plain program:** This is the complete code without any comments to have an complete overview of the code.

The commented program

Include files

This is the main ginkgo header file.
```
#include <ginkgo/ginkgo.hpp>
```

Add the fstream header to read from data from files.
```
#include <fstream>
```

Add the C++ iostream header to output information to the console.

```
#include <iostream>
```

Add the STL map header for the executor selection

```
#include <map>
```

Add the string manipulation header to handle strings.

```
#include <string>
int main(int argc, char* argv[])
{
```

Use some shortcuts. In Ginkgo, vectors are seen as a gko::matrix::Dense with one column/one row. The advantage of this concept is that using multiple vectors is a now a natural extension of adding columns/rows are necessary.

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
```

The gko::matrix::Csr class is used here, but any other matrix class such as gko::matrix::Coo, gko::matrix::Hybrid, gko::matrix::Ell or gko::matrix::Sellp could also be used.

```
using mtx = gko::matrix::Csr<ValueType, IndexType>;
```

The gko::solver::Cg is used here, but any other solver class can also be used.

```
using cg = gko::solver::Cg<ValueType>;
```

Print the ginkgo version information.

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
    std::exit(-1);
}
```

Where do you want to run your solver ?

The gko::Executor class is one of the cornerstones of Ginkgo. Currently, we have support for an gko::OmpExecutor, which uses OpenMP multi-threading in most of its kernels, a gko::ReferenceExecutor, a single threaded specialization of the OpenMP executor and a gko::CudaExecutor which runs the code on a NVIDIA GPU if available.

Note

> With the help of C++, you see that you only ever need to change the executor and all the other functions/routines within Ginkgo should automatically work and run on the executor with any other changes.

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)(); // throws if not valid
```

Reading your data and transfer to the proper device.

Read the matrix, right hand side and the initial solution using the read function.

Note

> Ginkgo uses C++ smart pointers to automatically manage memory. To this end, we use our own object ownership transfer functions that under the hood call the required smart pointer functions to manage object ownership. The gko::share , gko::give and gko::lend are the functions that you would need to use.

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
```

Creating the solver

Generate the gko::solver factory. Ginkgo uses the concept of Factories to build solvers with certain properties. Observe the Fluent interface used here. Here a cg solver is generated with a stopping criteria of maximum iterations of 20 and a residual norm reduction of 1e-7. You also observe that the stopping criteria(gko::stop) are also generated from factories using their build methods. You need to specify the executors which each of the object needs to be built on.

```
const RealValueType reduction_factor{1e-7};
auto solver_gen =
    cg::build()
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(20u).on(exec),
            gko::stop::ResidualNorm<ValueType>::build()
                .with_reduction_factor(reduction_factor)
                .on(exec))
        .on(exec);
```

Generate the solver from the matrix. The solver factory built in the previous step takes a "matrix"(a gko::LinOp to be more general) as an input. In this case we provide it with a full matrix that we previously read, but as the solver only effectively uses the apply() method within the provided "matrix" object, you can effectively create a gko::LinOp class with your own apply implementation to accomplish more tasks. We will see an example of how this can be done in the custom-matrix-format example

```
auto solver = solver_gen->generate(A);
```

Finally, solve the system. The solver, being a gko::LinOp, can be applied to a right hand side, b to obtain the solution, x.

```
solver->apply(lend(b), lend(x));
```

Print the solution to the command line.

```
std::cout « "Solution (x):\n";
write(std::cout, lend(x));
```

To measure if your solution has actually converged, you can measure the error of the solution. one, neg_one are objects that represent the numbers which allow for a uniform interface when computing on any device. To compute the residual, all you need to do is call the apply method, which in this case is an spmv and equivalent to the LAPACK z_spmv routine. Finally, you compute the euclidean 2-norm with the compute_norm2 function.

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
}
```

Results

The following is the expected result:
```
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
2.10788e-15
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iostream>
#include <map>
#include <string>
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
```

```
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor] " « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    const RealValueType reduction_factor{1e-7};
    auto solver_gen =
        cg::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(20u).on(exec),
                gko::stop::ResidualNorm<ValueType>::build()
                    .with_reduction_factor(reduction_factor)
                    .on(exec))
            .on(exec);
    auto solver = solver_gen->generate(A);
    solver->apply(lend(b), lend(x));
    std::cout « "Solution (x):\n";
    write(std::cout, lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
}
```

Chapter 37

The simple-solver-logging program

The simple solver with logging example..

This example depends on simple-solver, minimal-cuda-solver.

Introduction

About the example

The commented program

```
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
namespace {
template <typename ValueType>
void print_vector(const std::string& name,
                  const gko::matrix::Dense<ValueType>* vec)
{
    std::cout « name « " = [" « std::endl;
    for (int i = 0; i < vec->get_size()[0]; ++i) {
        std::cout « "    " « vec->at(i, 0) « std::endl;
    }
    std::cout « "];" « std::endl;
}
}  // namespace
int main(int argc, char* argv[])
{
```

Some shortcuts

```
using ValueType = double;
using RealValueType = gko::remove_complex<ValueType>;
using IndexType = int;
using vec = gko::matrix::Dense<ValueType>;
using real_vec = gko::matrix::Dense<RealValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && (std::string(argv[1]) == "--help")) {
    std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
    std::exit(-1);
}
```

Figure out where to run the code

```
const auto executor_string = argc >= 2 ? argv[1] : "reference";
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

Read data

```
auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
```

Let's declare a logger which prints to std::cout instead of printing to a file. We log all events except for all linop factory and polymorphic object events. Events masks are group of events which are provided for convenience.

```
std::shared_ptr<gko::log::Stream<ValueType» stream_logger =
    gko::log::Stream<ValueType>::create(
        exec,
        gko::log::Logger::all_events_mask ^
            gko::log::Logger::linop_factory_events_mask ^
            gko::log::Logger::polymorphic_object_events_mask,
        std::cout);
```

Add stream_logger to the executor

```
exec->add_logger(stream_logger);
```

Add stream_logger only to the ResidualNorm criterion Factory Note that the logger will get automatically propagated to every criterion generated from this factory.

```
const RealValueType reduction_factor{1e-7};
using ResidualCriterionFactory =
    gko::stop::ResidualNorm<ValueType>::Factory;
std::shared_ptr<ResidualCriterionFactory> residual_criterion =
    ResidualCriterionFactory::create()
        .with_reduction_factor(reduction_factor)
        .on(exec);
residual_criterion->add_logger(stream_logger);
```

Generate solver

```
auto solver_gen =
    cg::build()
        .with_criteria(
            residual_criterion,
            gko::stop::Iteration::build().with_max_iters(20u).on(exec))
        .on(exec);
auto solver = solver_gen->generate(A);
```

First we add facilities to only print to a file. It's possible to select events, using masks, e.g. only iterations mask: gko::log::Logger::iteration_complete_mask. See the documentation of Logger class for more information.

```
std::ofstream filestream("my_file.txt");
solver->add_logger(gko::log::Stream<ValueType>::create(
    exec, gko::log::Logger::all_events_mask, filestream));
solver->add_logger(stream_logger);
```

Add another logger which puts all the data in an object, we can later retrieve this object in our code. Here we only have want Executor and criterion check completed events.

```
std::shared_ptr<gko::log::Record> record_logger = gko::log::Record::create(
    exec, gko::log::Logger::executor_events_mask |
              gko::log::Logger::criterion_check_completed_mask);
exec->add_logger(record_logger);
residual_criterion->add_logger(record_logger);
```

Solve system

```
solver->apply(lend(b), lend(x));
```

Finally, get some data from `record_logger` and print the last memory location copied

```
auto& last_copy = record_logger->get().copy_completed.back();
std::cout « "Last memory copied was of size " « std::hex
          « std::get<0>(*last_copy).num_bytes « " FROM executor "
          « std::get<0>(*last_copy).exec « " pointer "
          « std::get<0>(*last_copy).location « " TO executor "
          « std::get<1>(*last_copy).exec « " pointer "
          « std::get<1>(*last_copy).location « std::dec « std::endl;
```

Also print the residual of the last criterion check event (where convergence happened)

```
auto residual =
    record_logger->get().criterion_check_completed.back()->residual.get();
auto residual_d = gko::as<vec>(residual);
print_vector("Residual", residual_d);
```

Print solution

```
std::cout « "Solution (x):\n";
write(std::cout, lend(x));
```

Calculate residual

```
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
}
```

Results

This is the expected output:

```
[LOG] »> apply started on A LinOp[gko::solver::Cg<double>,0x2142d60] with b
       LinOp[gko::matrix::Dense<double>,0x2142140] and x LinOp[gko::matrix::Dense<double>,0x2143450]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2142280] with
       Bytes[8]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143410] with
       Bytes[8]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21480a0] with
       Bytes[152]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21482f0] with
       Bytes[152]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21484d0] with
       Bytes[152]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21486b0] with
       Bytes[152]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148010] with
       Bytes[8]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148a60] with
       Bytes[8]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21482b0] with
       Bytes[8]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148a40] with
       Bytes[8]
[LOG] »> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[1]
[LOG] »> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2147c90] with
       Bytes[1]
[LOG] »> Operation[gko::solver::cg::initialize_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
```

```
[LOG] >> Operation[gko::solver::cg::initialize_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::advanced_spmv_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Csr<double, int> const*, gko::matrix::Dense<double> const*, gko::matrix::Dense<double>
      const*, gko::matrix::Dense<double>*>,0x7ffd93d14aa0] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::advanced_spmv_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Csr<double, int> const*, gko::matrix::Dense<double> const*, gko::matrix::Dense<double>
      const*, gko::matrix::Dense<double>*>,0x7ffd93d14aa0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[2]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148ee0] with
      Bytes[2]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148e50] with
      Bytes[8]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2147ce0] with
      Bytes[8]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14a20] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14a20] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
      Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 0 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
      LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
      residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 0 with ID
      1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
      gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
      gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 0 with
      ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149550] with
      Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149550] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149550] with
      Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149730] with
      Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149730] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149730] with
      Bytes[152]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
```

```
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 1 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 1 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 1 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149980] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149980] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149980] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b80] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149b80] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149b80] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149730]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149730]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149550]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149550]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
```

```
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 2 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 2 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 2 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149290] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149290] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149290] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149690] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149690] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149690] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b80]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b80]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149980]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149980]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
```

```
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 3 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
       LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
       residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 3 with ID
       1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 3 with
       ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149890] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149890] with
       Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149ae0] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149ae0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149ae0] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149690]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149690]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149290]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149290]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
```

```
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 4 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 4 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 4 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149200] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149200] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149200] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149310] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149310] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149310] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149ae0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149ae0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
```

```
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 5 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 5 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 5 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149890] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149890] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149cc0] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149cc0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149cc0] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149310]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149310]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149200]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149200]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
```

```
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 6 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 6 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 6 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149450] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149450] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149450] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21494f0] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21494f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21494f0] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149cc0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149cc0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
```

```
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 7 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
       LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
       residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 7 with ID
       1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 7 with
       ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149730] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149730] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149730] with
       Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21497d0] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21497d0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21497d0] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21494f0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21494f0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149450]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149450]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
```

```
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 8 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 8 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 8 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149200] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149200] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149200] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21492a0] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21492a0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21492a0] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21497d0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21497d0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149730]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149730]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 9 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
```

```
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 9 with ID
        1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 9 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149620] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149620] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149620] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21496c0] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21496c0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21496c0] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21492a0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21492a0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149200]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149200]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 10 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 10 with
        ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
```

```
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 10 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149450] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149450] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149450] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149760] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149760] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149760] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21496c0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21496c0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149620]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149620]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 11 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 11 with
        ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
```

```
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 11 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149860] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149860] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149860] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149900] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149900] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149900] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149760]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149760]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149450]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149450]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 12 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 12 with
        ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
```

```
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 12 with
       ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21499a0] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21499a0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21499a0] with
       Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21493d0] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21493d0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21493d0] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149900]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149900]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149860]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149860]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 13 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
       LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
       residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 13 with
       ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 13 with
```

```
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149490] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149490] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149490] with
        Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149580] with
        Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149580] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149580] with
        Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21493d0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21493d0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21499a0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21499a0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
        gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
        Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
        Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 14 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
        LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
        residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 14 with
        ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
        gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
        gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
        gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
        Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 14 with
        ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b50] with
        Bytes[152]
```

```
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149b50] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149b50] with
      Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21499c0] with
      Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21499c0] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x21499c0] with
      Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149580]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149580]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149490]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149490]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
      Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 15 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
      LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
      residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 15 with
      ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
      gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
      gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 15 with
      ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149a70] with
      Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149a70] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
```

```
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149a70] with
      Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149340] with
      Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149340] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149340] with
      Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21499c0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21499c0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b50]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b50]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
      gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
      Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 16 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
      LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
      residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 16 with
      ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
      gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
      gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
      gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
      gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
      Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 16 with
      ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149970] with
      Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149970] with
      Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
      Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149970] with
      Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b10] with
```

```
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149b10] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149b10] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149340]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149340]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149a70]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149a70]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 17 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
       LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
       residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 17 with
       ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 17 with
       ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149780] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149780] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149780] with
       Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149890] with
       Bytes[152]
```

```
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149890] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b10]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149b10]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149970]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149970]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 18 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
       LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
       residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 18 with
       ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 18 with
       ID 1 and finalized set to 1. It changed one RHS 0, stopped the iteration process 0
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149620] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149620] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149620] with
       Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149cf0] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149cf0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149cf0] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890]
```

```
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149780]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149780]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_1_operation<gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::csr::spmv_operation<gko::matrix::Csr<double, int> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14b80] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::solver::cg::step_2_operation<gko::matrix::Dense<double>*&,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::matrix::Dense<double>*, gko::matrix::Dense<double>*,
       gko::Array<gko::stopping_status>*>,0x7ffd93d14ef0] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x21482f0] with
       Bytes[152]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_dot_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double> const*, gko::matrix::Dense<double>*>,0x7ffd93d14c50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> iteration 19 completed with solver LinOp[gko::solver::Cg<double>,0x2142d60] with residual
       LinOp[gko::matrix::Dense<double>,0x2147b30], solution LinOp[gko::matrix::Dense<double>,0x2143450] and
       residual_norm LinOp[gko::LinOp const*,0]
[LOG] >> check started for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 19 with
       ID 1 and finalized set to 1
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14ad0] completed on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> Operation[gko::stop::residual_norm::residual_norm_operation<gko::matrix::Dense<double> const*&,
       gko::matrix::Dense<double>*, double&, unsigned char&, bool&, gko::Array<gko::stopping_status>*&,
       gko::Array<bool>*, bool*, bool*&>,0x7ffd93d14b90] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] >> check completed for stop::Criterion[gko::stop::ResidualNorm<double>,0x2148db0] at iteration 19 with
       ID 1 and finalized set to 1. It changed one RHS 1, stopped the iteration process 1
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149890] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149890] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x21480a0] to Location[0x2149890] with
       Bytes[152]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[152]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149340] with
       Bytes[152]
[LOG] >> copy started from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149340] with
       Bytes[152]
[LOG] >> copy completed from Executor[gko::ReferenceExecutor,0x21400d0] to
       Executor[gko::ReferenceExecutor,0x21400d0] from Location[0x2143e90] to Location[0x2149340] with
       Bytes[152]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149cf0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149cf0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149620]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149620]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148ee0]
```

```
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148ee0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2147ce0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2147ce0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148e50]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148e50]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2147c90]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2147c90]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21482b0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21482b0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148a40]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148a40]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148a60]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148a60]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148010]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2148010]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21486b0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21486b0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21484d0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21484d0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21482f0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21482f0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21480a0]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x21480a0]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143410]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143410]
[LOG] >> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2142280]
[LOG] >> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2142280]
[LOG] >> apply completed on A LinOp[gko::solver::Cg<double>,0x2142d60] with b
       LinOp[gko::matrix::Dense<double>,0x2142140] and x LinOp[gko::matrix::Dense<double>,0x2143450]
Last memory copied was of size 98 FROM executor 0x21400d0 pointer 2143e90 TO executor 0x21400d0 pointer
       2149340
Residual = [
    8.1654e-19
    -1.51449e-17
    2.23854e-17
    -1.0842e-19
    6.09864e-20
    -1.92446e-18
    1.97867e-18
    -4.58075e-18
    -1.55854e-18
    -2.64274e-17
    4.20128e-17
    -8.71427e-18
    -2.62919e-18
    -5.49947e-17
    5.51893e-17
    -1.57022e-16
    -4.2034e-17
    -8.71951e-16
    1.37837e-15
];
Solution (x):
%%MatrixMarket matrix array real general
19 1
0.252218
0.108645
0.0662811
0.0630433
0.0384088
0.0396536
0.0402648
0.0338935
0.0193098
0.0234653
0.0211499
0.0196413
0.0199151
0.0181674
0.0162722
0.0150714
0.0107016
0.0121141
0.0123025
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149bb0] with
       Bytes[8]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149870] with
       Bytes[8]
[LOG] >> allocation started on Executor[gko::ReferenceExecutor,0x21400d0] with Bytes[8]
[LOG] >> allocation completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149500] with
       Bytes[8]
[LOG] >> Operation[gko::matrix::csr::advanced_spmv_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Csr<double, int> const*, gko::matrix::Dense<double> const*, gko::matrix::Dense<double>
       const*, gko::matrix::Dense<double>*>,0x7ffd93d14e50] started on
       Executor[gko::ReferenceExecutor,0x21400d0]
```

```
[LOG] »> Operation[gko::matrix::csr::advanced_spmv_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Csr<double, int> const*, gko::matrix::Dense<double> const*, gko::matrix::Dense<double>
       const*, gko::matrix::Dense<double>*>,0x7ffd93d14e50] completed on
       Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] »> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14f70] started on Executor[gko::ReferenceExecutor,0x21400d0]
[LOG] »> Operation[gko::matrix::dense::compute_norm2_operation<gko::matrix::Dense<double> const*,
       gko::matrix::Dense<double>*>,0x7ffd93d14f70] completed on Executor[gko::ReferenceExecutor,0x21400d0]
Residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
2.10788e-15
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149500]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149500]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149870]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149870]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149bb0]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2149bb0]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143e90]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143e90]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143590]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143590]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2142b10]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2142b10]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143c30]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143c30]
[LOG] »> free started on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143790]
[LOG] »> free completed on Executor[gko::ReferenceExecutor,0x21400d0] at Location[0x2143790]
```

Comments about programming and debugging

The plain program

```
/*******************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
#include <ginkgo/ginkgo.hpp>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
namespace {
template <typename ValueType>
void print_vector(const std::string& name,
                  const gko::matrix::Dense<ValueType>* vec)
{
    std::cout « name « " = [" « std::endl;
    for (int i = 0; i < vec->get_size()[0]; ++i) {
        std::cout « "    " « vec->at(i, 0) « std::endl;
    }
    std::cout « "];" « std::endl;
}
```

```
}  // namespace
int main(int argc, char* argv[])
{
    using ValueType = double;
    using RealValueType = gko::remove_complex<ValueType>;
    using IndexType = int;
    using vec = gko::matrix::Dense<ValueType>;
    using real_vec = gko::matrix::Dense<RealValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && (std::string(argv[1]) == "--help")) {
        std::cerr « "Usage: " « argv[0] « " [executor]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
                 return gko::DpcppExecutor::create(0,
                                                   gko::OmpExecutor::create());
             }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    auto A = share(gko::read<mtx>(std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);
    std::shared_ptr<gko::log::Stream<ValueType» stream_logger =
        gko::log::Stream<ValueType>::create(
            exec,
            gko::log::Logger::all_events_mask ^
                gko::log::Logger::linop_factory_events_mask ^
                gko::log::Logger::polymorphic_object_events_mask,
            std::cout);
    exec->add_logger(stream_logger);
    const RealValueType reduction_factor{1e-7};
    using ResidualCriterionFactory =
        gko::stop::ResidualNorm<ValueType>::Factory;
    std::shared_ptr<ResidualCriterionFactory> residual_criterion =
        ResidualCriterionFactory::create()
            .with_reduction_factor(reduction_factor)
            .on(exec);
    residual_criterion->add_logger(stream_logger);
    auto solver_gen =
        cg::build()
            .with_criteria(
                residual_criterion,
                gko::stop::Iteration::build().with_max_iters(20u).on(exec))
            .on(exec);
    auto solver = solver_gen->generate(A);
    std::ofstream filestream("my_file.txt");
    solver->add_logger(gko::log::Stream<ValueType>::create(
        exec, gko::log::Logger::all_events_mask, filestream));
    solver->add_logger(stream_logger);
    std::shared_ptr<gko::log::Record> record_logger = gko::log::Record::create(
        exec, gko::log::Logger::executor_events_mask |
                  gko::log::Logger::criterion_check_completed_mask);
    exec->add_logger(record_logger);
    residual_criterion->add_logger(record_logger);
    solver->apply(lend(b), lend(x));
    auto& last_copy = record_logger->get().copy_completed.back();
    std::cout « "Last memory copied was of size " « std::hex
              « std::get<0>(*last_copy).num_bytes « " FROM executor "
              « std::get<0>(*last_copy).exec « " pointer "
              « std::get<0>(*last_copy).location « " TO executor "
              « std::get<1>(*last_copy).exec « " pointer "
              « std::get<1>(*last_copy).location « std::dec « std::endl;
    auto residual =
        record_logger->get().criterion_check_completed.back()->residual.get();
    auto residual_d = gko::as<vec>(residual);
    print_vector("Residual", residual_d);
    std::cout « "Solution (x):\n";
    write(std::cout, lend(x));
    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
```

```
    auto res = gko::initialize<real_vec>({0.0}, exec);
    A->apply(lend(one), lend(x), lend(neg_one), lend(b));
    b->compute_norm2(lend(res));
    std::cout « "Residual norm sqrt(r^T r):\n";
    write(std::cout, lend(res));
}
```

Chapter 38

The three-pt-stencil-solver program

The 3-point stencil example..

This example depends on simple-solver, poisson-solver.

Introduction

This example solves a 1D Poisson equation:

$u : [0, 1] \rightarrow R$
$u" = f$
$u(0) = u0$
$u(1) = u1$

using a finite difference method on an equidistant grid with `K` discretization points (`K` can be controlled with a command line parameter). The discretization is done via the second order Taylor polynomial:

$u(x + h) = u(x) - u'(x)h + 1/2u"(x)h^2 + O(h^3)$
$u(x - h) = u(x) + u'(x)h + 1/2u"(x)h^2 + O(h^3)/+$
$- -$
$-u(x - h) + 2u(x) + -u(x + h) = -f(x)h^2 + O(h^3)$

For an equidistant grid with K "inner" discretization points $x1, ..., xk,$and step size $h = 1/(K + 1)$, the formula produces a system of linear equations

$2u_1 - u_2 = -f_1h^2 + u0$
$-u_(k - 1) + 2u_k - u_(k + 1) = -f_kh^2, k = 2, ..., K - 1$
$-u_(K - 1) + 2u_K = -f_Kh^2 + u1$

which is then solved using Ginkgo's implementation of the CG method preconditioned with block-Jacobi. It is also possible to specify on which executor Ginkgo will solve the system via the command line. The function 'f'is set to '$f(x) = 6x$' (making the solution '$u(x) = x^3$'), but that can be changed in the `main` function.

The intention of the example is to show how Ginkgo can be integrated into existing software - the `generate_stencil_matrix`, `generate_rhs`, `print_solution`, `compute_error` and `main` function do not reference Ginkgo at all (i.e. they could have been there before the application developer decided to use Ginkgo, and the only part where Ginkgo is introduced is inside the `solve_system` function.

About the example

The commented program

```
/ *****************************<DESCRIPTION>**********************************
This example solves a 1D Poisson equation:
    u : [0, 1] -> R
    u'' = f
    u(0) = u0
    u(1) = u1
using a finite difference method on an equidistant grid with `K` discretization
points (`K` can be controlled with a command line parameter). The discretization
is done via the second order Taylor polynomial:
u(x + h) = u(x) - u'(x)h + 1/2 u''(x)h^2 + O(h^3)
u(x - h) = u(x) + u'(x)h + 1/2 u''(x)h^2 + O(h^3)  / +
---------------------------------------------
-u(x - h) + 2u(x) + -u(x + h) = -f(x)h^2 + O(h^3)
For an equidistant grid with K "inner" discretization points x1, ..., xk, and
step size h = 1 / (K + 1), the formula produces a system of linear equations
           2u_1 - u_2     = -f_1 h^2 + u0
-u_(k-1) + 2u_k - u_(k+1) = -f_k h^2,       k = 2, ..., K - 1
-u_(K-1) + 2u_K           = -f_K h^2 + u1
which is then solved using Ginkgo's implementation of the CG method
preconditioned with block-Jacobi. It is also possible to specify on which
executor Ginkgo will solve the system via the command line.
The function `f` is set to `f(x) = 6x` (making the solution `u(x) = x^3`), but
that can be changed in the `main` function.

The intention of the example is to show how Ginkgo can be integrated into
existing software - the `generate_stencil_matrix`, `generate_rhs`,
`print_solution`, `compute_error` and `main` function do not reference Ginkgo at
all (i.e. they could have been there before the application developer decided to
use Ginkgo, and the only part where Ginkgo is introduced is inside the
`solve_system` function.
****************************<DESCRIPTION>********************************* /

#include <ginkgo/ginkgo.hpp>
#include <iostream>
#include <map>
#include <string>
#include <vector>
```

Creates a stencil matrix in CSR format for the given number of discretization points.

```
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(IndexType discretization_points,
                             IndexType* row_ptrs, IndexType* col_idxs,
                             ValueType* values)
{
    IndexType pos = 0;
    const ValueType coefs[] = {-1, 2, -1};
    row_ptrs[0] = pos;
    for (IndexType i = 0; i < discretization_points; ++i) {
        for (auto ofs : {-1, 0, 1}) {
            if (0 <= i + ofs && i + ofs < discretization_points) {
                values[pos] = coefs[ofs + 1];
                col_idxs[pos] = i + ofs;
                ++pos;
            }
        }
        row_ptrs[i + 1] = pos;
    }
}
```

Generates the RHS vector given `f` and the boundary conditions.

```
template <typename Closure, typename ValueType, typename IndexType>
void generate_rhs(IndexType discretization_points, Closure f, ValueType u0,
                  ValueType u1, ValueType* rhs)
{
    const ValueType h = 1.0 / (discretization_points + 1);
    for (IndexType i = 0; i < discretization_points; ++i) {
        const ValueType xi = ValueType(i + 1) * h;
        rhs[i] = -f(xi) * h * h;
    }
    rhs[0] += u0;
    rhs[discretization_points - 1] += u1;
}
```

Prints the solution `u`.

```
template <typename ValueType, typename IndexType>
void print_solution(IndexType discretization_points, ValueType u0, ValueType u1,
                    const ValueType* u)
```

```
{
    std::cout « u0 « '\n';
    for (IndexType i = 0; i < discretization_points; ++i) {
        std::cout « u[i] « '\n';
    }
    std::cout « u1 « std::endl;
}
```

Computes the 1-norm of the error given the computed `u` and the correct solution function `correct_u`.

```
template <typename Closure, typename ValueType, typename IndexType>
gko::remove_complex<ValueType> calculate_error(IndexType discretization_points,
                                               const ValueType* u,
                                               Closure correct_u)
{
    const ValueType h = 1.0 / (discretization_points + 1);
    gko::remove_complex<ValueType> error = 0.0;
    for (IndexType i = 0; i < discretization_points; ++i) {
        using std::abs;
        const ValueType xi = ValueType(i + 1) * h;
        error += abs(u[i] - correct_u(xi)) / abs(correct_u(xi));
    }
    return error;
}
template <typename ValueType, typename IndexType>
void solve_system(const std::string& executor_string,
                  IndexType discretization_points, IndexType* row_ptrs,
                  IndexType* col_idxs, ValueType* values, ValueType* rhs,
                  ValueType* u, gko::remove_complex<ValueType> reduction_factor)
{
```

Some shortcuts

```
using vec = gko::matrix::Dense<ValueType>;
using mtx = gko::matrix::Csr<ValueType, IndexType>;
using cg = gko::solver::Cg<ValueType>;
using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
using val_array = gko::Array<ValueType>;
using idx_array = gko::Array<IndexType>;
const auto& dp = discretization_points;
```

Figure out where to run the code

```
std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
    exec_map{
        {"omp", [] { return gko::OmpExecutor::create(); }},
        {"cuda",
         [] {
             return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                              true);
         }},
        {"hip",
         [] {
             return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                             true);
         }},
        {"dpcpp",
         [] {
             return gko::DpcppExecutor::create(0,
                                               gko::OmpExecutor::create());
         }},
        {"reference", [] { return gko::ReferenceExecutor::create(); }}};
```

executor where Ginkgo will perform the computation

```
const auto exec = exec_map.at(executor_string)();  // throws if not valid
```

executor where the application initialized the data

```
const auto app_exec = exec->get_master();
```

Tell Ginkgo to use the data in our application

Matrix: we have to set the executor of the matrix to the one where we want SpMVs to run (in this case `exec`). When creating array views, we have to specify the executor where the data is (in this case `app_exec`).

If the two do not match, Ginkgo will automatically create a copy of the data on `exec` (however, it will not copy the data back once it is done

- here this is not important since we are not modifying the matrix).

```
auto matrix = mtx::create(exec, gko::dim<2>(dp),
                          val_array::view(app_exec, 3 * dp - 2, values),
                          idx_array::view(app_exec, 3 * dp - 2, col_idxs),
                          idx_array::view(app_exec, dp + 1, row_ptrs));
```

RHS: similar to matrix

```
auto b = vec::create(exec, gko::dim<2>(dp, 1),
                     val_array::view(app_exec, dp, rhs), 1);
```

Solution: we have to be careful here - if the executors are different, once we compute the solution the array will not be automatically copied back to the original memory locations. Fortunately, whenever `apply` is called on a linear operator (e.g. matrix, solver) the arguments automatically get copied to the executor where the operator is, and copied back once the operation is completed. Thus, in this case, we can just define the solution on `app_exec`, and it will be automatically transferred to/from `exec` if needed.

```
auto x = vec::create(app_exec, gko::dim<2>(dp, 1),
                     val_array::view(app_exec, dp, u), 1);
```

Generate solver

```
auto solver_gen =
    cg::build()
        .with_criteria(gko::stop::Iteration::build()
                           .with_max_iters(gko::size_type(dp))
                           .on(exec),
                       gko::stop::ResidualNorm<ValueType>::build()
                           .with_reduction_factor(reduction_factor)
                           .on(exec))
        .with_preconditioner(bj::build().on(exec))
        .on(exec);
auto solver = solver_gen->generate(gko::give(matrix));
```

Solve system

```
    solver->apply(gko::lend(b), gko::lend(x));
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
```

Print version information

```
std::cout « gko::version_info::get() « std::endl;
if (argc == 2 && std::string(argv[1]) == "--help") {
    std::cerr « "Usage: " « argv[0]
              « " [executor] [DISCRETIZATION_POINTS]" « std::endl;
    std::exit(-1);
}
const auto executor_string = argc >= 2 ? argv[1] : "reference";
const IndexType discretization_points =
    argc >= 3 ? std::atoi(argv[2]) : 100;
```

problem:

```
auto correct_u = [](ValueType x) { return x * x * x; };
auto f = [](ValueType x) { return ValueType(6) * x; };
auto u0 = correct_u(0);
auto u1 = correct_u(1);
```

matrix

```
std::vector<IndexType> row_ptrs(discretization_points + 1);
std::vector<IndexType> col_idxs(3 * discretization_points - 2);
std::vector<ValueType> values(3 * discretization_points - 2);
```

right hand side

```
std::vector<ValueType> rhs(discretization_points);
```

solution

```
std::vector<ValueType> u(discretization_points, 0.0);
const gko::remove_complex<ValueType> reduction_factor = 1e-7;
generate_stencil_matrix(discretization_points, row_ptrs.data(),
                        col_idxs.data(), values.data());
```

looking for solution u = x^3: f = 6x, u(0) = 0, u(1) = 1

```
generate_rhs(discretization_points, f, u0, u1, rhs.data());
solve_system(executor_string, discretization_points, row_ptrs.data(),
             col_idxs.data(), values.data(), rhs.data(), u.data(),
             reduction_factor);
```

Uncomment to print the solution print_solution<ValueType, IndexType>(discretization_points, 0, 1, u.data());

```
    std::cout « "The average relative error is "
              « calculate_error(discretization_points, u.data(), correct_u) /
                    discretization_points
              « std::endl;
}
```

Results

This is the expected output:
```
The average relative error is 2.52236e-11
```

Comments about programming and debugging

The plain program

```
/*****************************<GINKGO LICENSE>******************************
Copyright (c) 2017-2022, the Ginkgo authors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
******************************<GINKGO LICENSE>*******************************/
/******************************<DESCRIPTION>*********************************
This example solves a 1D Poisson equation:

    u : [0, 1] -> R
    u" = f
    u(0) = u0
    u(1) = u1

using a finite difference method on an equidistant grid with `K` discretization
points (`K` can be controlled with a command line parameter). The discretization
is done via the second order Taylor polynomial:

u(x + h) = u(x) - u'(x)h + 1/2 u"(x)h^2 + O(h^3)
u(x - h) = u(x) + u'(x)h + 1/2 u"(x)h^2 + O(h^3)  / +
---------------------------------------------
-u(x - h) + 2u(x) + -u(x + h) = -f(x)h^2 + O(h^3)

For an equidistant grid with K "inner" discretization points x1, ..., xk, and
step size h = 1 / (K + 1), the formula produces a system of linear equations

           2u_1 - u_2     = -f_1 h^2 + u0
-u_(k-1) + 2u_k - u_(k+1) = -f_k h^2,       k = 2, ..., K - 1
-u_(K-1) + 2u_K           = -f_K h^2 + u1


which is then solved using Ginkgo's implementation of the CG method
preconditioned with block-Jacobi. It is also possible to specify on which
executor Ginkgo will solve the system via the command line.
The function `f` is set to `f(x) = 6x` (making the solution `u(x) = x^3`), but
that can be changed in the `main` function.

The intention of the example is to show how Ginkgo can be integrated into
existing software - the `generate_stencil_matrix`, `generate_rhs`,
`print_solution`, `compute_error` and `main` function do not reference Ginkgo at
all (i.e. they could have been there before the application developer decided to
use Ginkgo, and the only part where Ginkgo is introduced is inside the
`solve_system` function.
*****************************<DESCRIPTION>**********************************/
#include <ginkgo/ginkgo.hpp>
```

```
#include <iostream>
#include <map>
#include <string>
#include <vector>
template <typename ValueType, typename IndexType>
void generate_stencil_matrix(IndexType discretization_points,
                             IndexType* row_ptrs, IndexType* col_idxs,
                             ValueType* values)
{
    IndexType pos = 0;
    const ValueType coefs[] = {-1, 2, -1};
    row_ptrs[0] = pos;
    for (IndexType i = 0; i < discretization_points; ++i) {
        for (auto ofs : {-1, 0, 1}) {
            if (0 <= i + ofs && i + ofs < discretization_points) {
                values[pos] = coefs[ofs + 1];
                col_idxs[pos] = i + ofs;
                ++pos;
            }
        }
        row_ptrs[i + 1] = pos;
    }
}
template <typename Closure, typename ValueType, typename IndexType>
void generate_rhs(IndexType discretization_points, Closure f, ValueType u0,
                  ValueType u1, ValueType* rhs)
{
    const ValueType h = 1.0 / (discretization_points + 1);
    for (IndexType i = 0; i < discretization_points; ++i) {
        const ValueType xi = ValueType(i + 1) * h;
        rhs[i] = -f(xi) * h * h;
    }
    rhs[0] += u0;
    rhs[discretization_points - 1] += u1;
}
template <typename ValueType, typename IndexType>
void print_solution(IndexType discretization_points, ValueType u0, ValueType u1,
                    const ValueType* u)
{
    std::cout « u0 « '\n';
    for (IndexType i = 0; i < discretization_points; ++i) {
        std::cout « u[i] « '\n';
    }
    std::cout « u1 « std::endl;
}
template <typename Closure, typename ValueType, typename IndexType>
gko::remove_complex<ValueType> calculate_error(IndexType discretization_points,
                                               const ValueType* u,
                                               Closure correct_u)
{
    const ValueType h = 1.0 / (discretization_points + 1);
    gko::remove_complex<ValueType> error = 0.0;
    for (IndexType i = 0; i < discretization_points; ++i) {
        using std::abs;
        const ValueType xi = ValueType(i + 1) * h;
        error += abs(u[i] - correct_u(xi)) / abs(correct_u(xi));
    }
    return error;
}
template <typename ValueType, typename IndexType>
void solve_system(const std::string& executor_string,
                  IndexType discretization_points, IndexType* row_ptrs,
                  IndexType* col_idxs, ValueType* values, ValueType* rhs,
                  ValueType* u, gko::remove_complex<ValueType> reduction_factor)
{
    using vec = gko::matrix::Dense<ValueType>;
    using mtx = gko::matrix::Csr<ValueType, IndexType>;
    using cg = gko::solver::Cg<ValueType>;
    using bj = gko::preconditioner::Jacobi<ValueType, IndexType>;
    using val_array = gko::Array<ValueType>;
    using idx_array = gko::Array<IndexType>;
    const auto& dp = discretization_points;
    std::map<std::string, std::function<std::shared_ptr<gko::Executor>()»
        exec_map{
            {"omp", [] { return gko::OmpExecutor::create(); }},
            {"cuda",
             [] {
                 return gko::CudaExecutor::create(0, gko::OmpExecutor::create(),
                                                  true);
             }},
            {"hip",
             [] {
                 return gko::HipExecutor::create(0, gko::OmpExecutor::create(),
                                                 true);
             }},
            {"dpcpp",
             [] {
```

```
                return gko::DpcppExecutor::create(0,
                                                  gko::OmpExecutor::create());
            }},
            {"reference", [] { return gko::ReferenceExecutor::create(); }}};
    const auto exec = exec_map.at(executor_string)();  // throws if not valid
    const auto app_exec = exec->get_master();
    auto matrix = mtx::create(exec, gko::dim<2>(dp),
                              val_array::view(app_exec, 3 * dp - 2, values),
                              idx_array::view(app_exec, 3 * dp - 2, col_idxs),
                              idx_array::view(app_exec, dp + 1, row_ptrs));
    auto b = vec::create(exec, gko::dim<2>(dp, 1),
                         val_array::view(app_exec, dp, rhs), 1);
    auto x = vec::create(app_exec, gko::dim<2>(dp, 1),
                         val_array::view(app_exec, dp, u), 1);
    auto solver_gen =
        cg::build()
            .with_criteria(gko::stop::Iteration::build()
                               .with_max_iters(gko::size_type(dp))
                               .on(exec),
                           gko::stop::ResidualNorm<ValueType>::build()
                               .with_reduction_factor(reduction_factor)
                               .on(exec))
            .with_preconditioner(bj::build().on(exec))
            .on(exec);
    auto solver = solver_gen->generate(gko::give(matrix));
    solver->apply(gko::lend(b), gko::lend(x));
}
int main(int argc, char* argv[])
{
    using ValueType = double;
    using IndexType = int;
    std::cout « gko::version_info::get() « std::endl;
    if (argc == 2 && std::string(argv[1]) == "--help") {
        std::cerr « "Usage: " « argv[0]
                  « " [executor] [DISCRETIZATION_POINTS]" « std::endl;
        std::exit(-1);
    }
    const auto executor_string = argc >= 2 ? argv[1] : "reference";
    const IndexType discretization_points =
        argc >= 3 ? std::atoi(argv[2]) : 100;
    auto correct_u = [](ValueType x) { return x * x * x; };
    auto f = [](ValueType x) { return ValueType(6) * x; };
    auto u0 = correct_u(0);
    auto u1 = correct_u(1);
    std::vector<IndexType> row_ptrs(discretization_points + 1);
    std::vector<IndexType> col_idxs(3 * discretization_points - 2);
    std::vector<ValueType> values(3 * discretization_points - 2);
    std::vector<ValueType> rhs(discretization_points);
    std::vector<ValueType> u(discretization_points, 0.0);
    const gko::remove_complex<ValueType> reduction_factor = 1e-7;
    generate_stencil_matrix(discretization_points, row_ptrs.data(),
                            col_idxs.data(), values.data());
    generate_rhs(discretization_points, f, u0, u1, rhs.data());
    solve_system(executor_string, discretization_points, row_ptrs.data(),
                 col_idxs.data(), values.data(), rhs.data(), u.data(),
                 reduction_factor);
    std::cout « "The average relative error is "
              « calculate_error(discretization_points, u.data(), correct_u) /
                     discretization_points
              « std::endl;
}
```

Chapter 39

Module Documentation

39.1 CUDA Executor

A module dedicated to the implementation and usage of the CUDA executor in Ginkgo.

Classes

- class gko::CudaExecutor

  This is the Executor subclass which represents the CUDA device.

39.1.1 Detailed Description

A module dedicated to the implementation and usage of the CUDA executor in Ginkgo.

39.2 DPC++ Executor

A module dedicated to the implementation and usage of the DPC++ executor in Ginkgo.

Classes

- class gko::DpcppExecutor

    This is the Executor subclass which represents a DPC++ enhanced device.

39.2.1 Detailed Description

A module dedicated to the implementation and usage of the DPC++ executor in Ginkgo.

39.3 Executors

A module dedicated to the implementation and usage of the executors in Ginkgo.

Modules

- CUDA Executor

  A module dedicated to the implementation and usage of the CUDA executor in Ginkgo.
- DPC++ Executor

  A module dedicated to the implementation and usage of the DPC++ executor in Ginkgo.
- HIP Executor

  A module dedicated to the implementation and usage of the HIP executor in Ginkgo.
- OpenMP Executor

  A module dedicated to the implementation and usage of the OpenMP executor in Ginkgo.
- Reference Executor

  A module dedicated to the implementation and usage of the Reference executor in Ginkgo.

Classes

- class gko::Operation

  Operations can be used to define functionalities whose implementations differ among devices.
- class gko::Executor

  The first step in using the Ginkgo library consists of creating an executor.
- class gko::executor_deleter< T >

  This is a deleter that uses an executor's `free` method to deallocate the data.
- class gko::OmpExecutor

  This is the Executor subclass which represents the OpenMP device (typically CPU).
- class gko::ReferenceExecutor

  This is a specialization of the OmpExecutor, which runs the reference implementations of the kernels used for debugging purposes.
- class gko::CudaExecutor

  This is the Executor subclass which represents the CUDA device.
- class gko::HipExecutor

  This is the Executor subclass which represents the HIP enhanced device.
- class gko::DpcppExecutor

  This is the Executor subclass which represents a DPC++ enhanced device.

Macros

- #define GKO_REGISTER_OPERATION(_name, _kernel)

  Binds a set of device-specific kernels to an Operation.

39.3.1 Detailed Description

A module dedicated to the implementation and usage of the executors in Ginkgo.

Below, we provide a brief introduction to executors in Ginkgo, how they have been implemented, how to best make use of them and how to add new executors.

39.3.2 Executors in Ginkgo.

The first step in using the Ginkgo library consists of creating an executor. Executors are used to specify the location for the data of linear algebra objects, and to determine where the operations will be executed. Ginkgo currently supports three different executor types:

- OpenMP Executor specifies that the data should be stored and the associated operations executed on an OpenMP-supporting device (e.g. host CPU);
- CUDA Executor specifies that the data should be stored and the operations executed on the NVIDIA GPU accelerator;
- HIP Executor uses the HIP library to compile code for either NVIDIA or AMD GPU accelerator;
- DPC++ Executor uses the DPC++ compiler for any DPC++ supported hardware (e.g. Intel CPUs, GPU, FPGAs, ...);
- Reference Executor executes a non-optimized reference implementation, which can be used to debug the library.

39.3.3 Macro Definition Documentation

39.3.3.1 GKO_REGISTER_OPERATION

```
#define GKO_REGISTER_OPERATION(
                _name,
                _kernel )
```

Binds a set of device-specific kernels to an Operation.

It also defines a helper function which creates the associated operation. Any input arguments passed to the helper function are forwarded to the kernel when the operation is executed.

The kernels used to bind the operation are searched in `kernels::DEV_TYPE` namespace, where `DEV_TYPE` is replaced by `omp`, `cuda`, `hip`, `dpcpp` and `reference`.

Parameters

_name	operation name
_kernel	kernel which will be bound to the operation

39.3.3.2 Example

```
 {c++}
// define the omp, cuda, hip and reference kernels which will be bound to the
// operation
namespace kernels {
namespace omp {
void my_kernel(int x) {
     // omp code
}
}
namespace cuda {
```

```
void my_kernel(int x) {
     // cuda code
}
}
namespace hip {
void my_kernel(int x) {
     // hip code
}
}
namespace dpcpp {
void my_kernel(int x) {
     // dpcpp code
}
}
namespace reference {
void my_kernel(int x) {
    // reference code
}
}
// Bind the kernels to the operation
GKO_REGISTER_OPERATION(my_op, my_kernel);
int main() {
    // create executors
    auto omp = OmpExecutor::create();
    auto cuda = CudaExecutor::create(0, omp);
    auto hip = HipExecutor::create(0, omp);
    auto dpcpp = DpcppExecutor::create(0, omp);
    auto ref = ReferenceExecutor::create();
    // create the operation
    auto op = make_my_op(5); // x = 5
    omp->run(op);  // run omp kernel
    cuda->run(op);  // run cuda kernel
    hip->run(op);  // run hip kernel
    dpcpp->run(op);  // run DPC++ kernel
    ref->run(op);  // run reference kernel
}
```

39.4 Factorizations

A module dedicated to the implementation and usage of the Factorizations in Ginkgo.

Namespaces

- gko::factorization

    The Factorization namespace.

Classes

- class gko::factorization::Ic< ValueType, IndexType >

    Represents an incomplete Cholesky factorization (IC(0)) of a sparse matrix.
- class gko::factorization::Ilu< ValueType, IndexType >

    Represents an incomplete LU factorization – ILU(0) – of a sparse matrix.
- class gko::factorization::ParIc< ValueType, IndexType >

    ParIC is an incomplete Cholesky factorization which is computed in parallel.
- class gko::factorization::ParIct< ValueType, IndexType >

    ParICT is an incomplete threshold-based Cholesky factorization which is computed in parallel.
- class gko::factorization::ParIlu< ValueType, IndexType >

    ParILU is an incomplete LU factorization which is computed in parallel.
- class gko::factorization::ParIlut< ValueType, IndexType >

    ParILUT is an incomplete threshold-based LU factorization which is computed in parallel.

39.4.1 Detailed Description

A module dedicated to the implementation and usage of the Factorizations in Ginkgo.

39.5 HIP Executor

A module dedicated to the implementation and usage of the HIP executor in Ginkgo.

Classes

- class gko::HipExecutor

  This is the Executor subclass which represents the HIP enhanced device.

39.5.1 Detailed Description

A module dedicated to the implementation and usage of the HIP executor in Ginkgo.

39.6 Jacobi Preconditioner

A module dedicated to the implementation and usage of the Jacobi Preconditioner in Ginkgo.

Classes

- struct gko::preconditioner::block_interleaved_storage_scheme< IndexType >

  Defines the parameters of the interleaved block storage scheme used by block-Jacobi blocks.
- class gko::preconditioner::Jacobi< ValueType, IndexType >

  A block-Jacobi preconditioner is a block-diagonal linear operator, obtained by inverting the diagonal blocks of the source operator.

39.6.1 Detailed Description

A module dedicated to the implementation and usage of the Jacobi Preconditioner in Ginkgo.

39.7 Linear Operators

A module dedicated to the implementation and usage of the Linear operators in Ginkgo.

Modules

- Factorizations

  A module dedicated to the implementation and usage of the Factorizations in Ginkgo.
- SpMV employing different Matrix formats

  A module dedicated to the implementation and usage of the various Matrix Formats in Ginkgo.
- Preconditioners

  A module dedicated to the implementation and usage of the Preconditioners in Ginkgo.
- Solvers

  A module dedicated to the implementation and usage of the Solvers in Ginkgo.

Classes

- class gko::Combination< ValueType >

  The Combination class can be used to construct a linear combination of multiple linear operators `c1 * op1 + c2 * op2 + ...`
- class gko::Composition< ValueType >

  The Composition class can be used to compose linear operators `op1, op2, ..., opn` and obtain the operator `op1 * op2 * ...`
- class gko::LinOpFactory

  A LinOpFactory represents a higher order mapping which transforms one linear operator into another.
- class gko::ReadableFromMatrixData< ValueType, IndexType >

  A LinOp implementing this interface can read its data from a matrix_data structure.
- class gko::WritableToMatrixData< ValueType, IndexType >

  A LinOp implementing this interface can write its data to a matrix_data structure.
- class gko::Preconditionable

  A LinOp implementing this interface can be preconditioned.
- class gko::DiagonalLinOpExtractable

  The diagonal of a LinOp can be extracted.
- class gko::DiagonalExtractable< ValueType >

  The diagonal of a LinOp implementing this interface can be extracted.
- class gko::EnableAbsoluteComputation< AbsoluteLinOp >

  The EnableAbsoluteComputation mixin provides the default implementations of `compute_absolute_linop` and the absolute interface.
- class gko::EnableLinOp< ConcreteLinOp, PolymorphicBase >

  The EnableLinOp mixin can be used to provide sensible default implementations of the majority of the LinOp and PolymorphicObject interface.
- class gko::Perturbation< ValueType >

  The Perturbation class can be used to construct a LinOp to represent the operation `(identity + scalar * basis * projector)`.
- class gko::factorization::Ic< ValueType, IndexType >

  Represents an incomplete Cholesky factorization (IC(0)) of a sparse matrix.
- class gko::factorization::Ilu< ValueType, IndexType >

  Represents an incomplete LU factorization – ILU(0) – of a sparse matrix.
- class gko::factorization::ParIc< ValueType, IndexType >

  ParIC is an incomplete Cholesky factorization which is computed in parallel.

- class gko::factorization::ParIct< ValueType, IndexType >

  ParICT is an incomplete threshold-based Cholesky factorization which is computed in parallel.
- class gko::factorization::ParIlu< ValueType, IndexType >

  ParILU is an incomplete LU factorization which is computed in parallel.
- class gko::factorization::ParIlut< ValueType, IndexType >

  ParILUT is an incomplete threshold-based LU factorization which is computed in parallel.
- class gko::matrix::Coo< ValueType, IndexType >

  COO stores a matrix in the coordinate matrix format.
- class gko::matrix::Csr< ValueType, IndexType >

  CSR is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class gko::matrix::Dense< ValueType >

  Dense is a matrix format which explicitly stores all values of the matrix.
- class gko::matrix::Diagonal< ValueType >

  This class is a utility which efficiently implements the diagonal matrix (a linear operator which scales a vector row wise).
- class gko::matrix::Ell< ValueType, IndexType >

  ELL is a matrix format where stride with explicit zeros is used such that all rows have the same number of stored elements.
- class gko::matrix::Fbcsr< ValueType, IndexType >

  Fixed-block compressed sparse row storage matrix format.
- class gko::matrix::Fft

  This LinOp implements a 1D Fourier matrix using the FFT algorithm.
- class gko::matrix::Fft2

  This LinOp implements a 2D Fourier matrix using the FFT algorithm.
- class gko::matrix::Fft3

  This LinOp implements a 3D Fourier matrix using the FFT algorithm.
- class gko::matrix::Hybrid< ValueType, IndexType >

  HYBRID is a matrix format which splits the matrix into ELLPACK and COO format.
- class gko::matrix::Identity< ValueType >

  This class is a utility which efficiently implements the identity matrix (a linear operator which maps each vector to itself).
- class gko::matrix::IdentityFactory< ValueType >

  This factory is a utility which can be used to generate Identity operators.
- class gko::matrix::Permutation< IndexType >

  Permutation is a matrix "format" which stores the row and column permutation arrays which can be used for re-ordering the rows and columns a matrix.
- class gko::matrix::RowGatherer< IndexType >

  RowGatherer is a matrix "format" which stores the gather indices arrays which can be used to gather rows to another matrix.
- class gko::matrix::Sellp< ValueType, IndexType >

  SELL-P is a matrix format similar to ELL format.
- class gko::matrix::SparsityCsr< ValueType, IndexType >

  SparsityCsr is a matrix format which stores only the sparsity pattern of a sparse matrix by compressing each row of the matrix (compressed sparse row format).
- class gko::multigrid::AmgxPgm< ValueType, IndexType >

  Amgx parallel graph match (AmgxPgm) is the aggregate method introduced in the paper M.
- class gko::preconditioner::Ic< LSolverType, IndexType >

  The Incomplete Cholesky (IC) preconditioner solves the equation $LL^H * x = b$ for a given lower triangular matrix L and the right hand side b (can contain multiple right hand sides).
- class gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >

  The Incomplete LU (ILU) preconditioner solves the equation $LUx = b$ for a given lower triangular matrix L, an upper triangular matrix U and the right hand side b (can contain multiple right hand sides).

- class gko::preconditioner::Isai< IsaiType, ValueType, IndexType >

  The Incomplete Sparse Approximate Inverse (ISAI) Preconditioner generates an approximate inverse matrix for a given square matrix A, lower triangular matrix L, upper triangular matrix U or symmetric positive (spd) matrix B.
- class gko::preconditioner::Jacobi< ValueType, IndexType >

  A block-Jacobi preconditioner is a block-diagonal linear operator, obtained by inverting the diagonal blocks of the source operator.
- class gko::solver::Bicg< ValueType >

  BICG or the Biconjugate gradient method is a Krylov subspace solver.
- class gko::solver::Bicgstab< ValueType >

  BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.
- class gko::solver::CbGmres< ValueType >

  CB-GMRES or the compressed basis generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.
- class gko::solver::Cg< ValueType >

  CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.
- class gko::solver::Cgs< ValueType >

  CGS or the conjugate gradient square method is an iterative type Krylov subspace method which is suitable for general systems.
- class gko::solver::Fcg< ValueType >

  FCG or the flexible conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.
- class gko::solver::Gmres< ValueType >

  GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.
- class gko::solver::Idr< ValueType >

  IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.
- class gko::solver::Ir< ValueType >

  Iterative refinement (IR) is an iterative method that uses another coarse method to approximate the error of the current solution via the current residual.
- class gko::solver::LowerTrs< ValueType, IndexType >

  LowerTrs is the triangular solver which solves the system L x = b, when L is a lower triangular matrix.
- class gko::solver::Multigrid

  Multigrid methods have a hierarchy of many levels, whose corase level is a subset of the fine level, of the problem.
- class gko::solver::UpperTrs< ValueType, IndexType >

  UpperTrs is the triangular solver which solves the system U x = b, when U is an upper triangular matrix.

Macros

- #define GKO_CREATE_FACTORY_PARAMETERS(_parameters_name, _factory_name)

  This Macro will generate a new type containing the parameters for the factory `_factory_name`.
- #define GKO_ENABLE_LIN_OP_FACTORY(_lin_op, _parameters_name, _factory_name)

  This macro will generate a default implementation of a LinOpFactory for the LinOp subclass it is defined in.

Typedefs

- template<typename ConcreteFactory , typename ConcreteLinOp , typename ParametersType , typename PolymorphicBase = Lin↩ OpFactory>
  using gko::EnableDefaultLinOpFactory = EnableDefaultFactory< ConcreteFactory, ConcreteLinOp, ParametersType, PolymorphicBase >

  This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of LinOpFactory.

39.7.1 Detailed Description

A module dedicated to the implementation and usage of the Linear operators in Ginkgo.

Below we elaborate on one of the most important concepts of Ginkgo, the linear operator. The linear operator (LinOp) is a base class for all linear algebra objects in Ginkgo. The main benefit of having a single base class for the entire collection of linear algebra objects (as opposed to having separate hierarchies for matrices, solvers and preconditioners) is the generality it provides.

39.7.2 Advantages of this approach and usage

A common interface often allows for writing more generic code. If a user's routine requires only operations provided by the LinOp interface, the same code can be used for any kind of linear operators, independent of whether these are matrices, solvers or preconditioners. This feature is also extensively used in Ginkgo itself. For example, a preconditioner used inside a Krylov solver is a LinOp. This allows the user to supply a wide variety of preconditioners: either the ones which were designed to be used in this scenario (like ILU or block-Jacobi), a user-supplied matrix which is known to be a good preconditioner for the specific problem, or even another solver (e.g., if constructing a flexible GMRES solver).

For example, a matrix free implementation would require the user to provide an apply implementation and instead of passing the generated matrix to the solver, they would have to provide their apply implementation for all the executors needed and no other code needs to be changed. See The custom-matrix-format program example for more details.

39.7.3 Linear operator as a concept

The linear operator (LinOp) is a base class for all linear algebra objects in Ginkgo. The main benefit of having a single base class for the entire collection of linear algebra objects (as opposed to having separate hierarchies for matrices, solvers and preconditioners) is the generality it provides.

First, since all subclasses provide a common interface, the library users are exposed to a smaller set of routines. For example, a matrix-vector product, a preconditioner application, or even a system solve are just different terms given to the operation of applying a certain linear operator to a vector. As such, Ginkgo uses the same routine name, LinOp::apply() for each of these operations, where the actual operation performed depends on the type of linear operator involved in the operation.

Second, a common interface often allows for writing more generic code. If a user's routine requires only operations provided by the LinOp interface, the same code can be used for any kind of linear operators, independent of whether these are matrices, solvers or preconditioners. This feature is also extensively used in Ginkgo itself. For example, a preconditioner used inside a Krylov solver is a LinOp. This allows the user to supply a wide variety of preconditioners: either the ones which were designed to be used in this scenario (like ILU or block-Jacobi), a user-supplied matrix which is known to be a good preconditioner for the specific problem, or even another solver (e.g., if constructing a flexible GMRES solver).

A key observation for providing a unified interface for matrices, solvers, and preconditioners is that the most common operation performed on all of them can be expressed as an application of a linear operator to a vector:

- the sparse matrix-vector product with a matrix A is a linear operator application $y = Ax$;
- the application of a preconditioner is a linear operator application $y = M^{-1}x$, where M is an approximation of the original system matrix A (thus a preconditioner represents an "approximate inverse" operator M^{-1}).
- the system solve $Ax = b$ can be viewed as linear operator application $x = A^{-1}b$ (it goes without saying that the implementation of linear system solves does not follow this conceptual idea), so a linear system solver can be viewed as a representation of the operator A^{-1}.

Finally, direct manipulation of LinOp objects is rarely required in simple scenarios. As an illustrative example, one could construct a fixed-point iteration routine $x_{k+1} = Lx_k + b$ as follows:

```
std::unique_ptr<matrix::Dense<» calculate_fixed_point(
        int iters, const LinOp *L, const matrix::Dense<> *x0
        const matrix::Dense<> *b)
{
    auto x = gko::clone(x0);
    auto tmp = gko::clone(x0);
    auto one = Dense<>::create(L->get_executor(), {1.0,});
    for (int i = 0; i < iters; ++i) {
        L->apply(gko::lend(tmp), gko::lend(x));
        x->add_scaled(gko::lend(one), gko::lend(b));
        tmp->copy_from(gko::lend(x));
    }
    return x;
}
```

Here, if L is a matrix, LinOp::apply() refers to the matrix vector product, and `L->apply(a, b)` computes $b = L \cdot a$. `x->add_scaled(one.get(), b.get())` is the `axpy` vector update $x := x + b$.

The interesting part of this example is the apply() routine at line 4 of the function body. Since this routine is part of the LinOp base class, the fixed-point iteration routine can calculate a fixed point not only for matrices, but for any type of linear operator.

Linear Operators

39.7.4 Macro Definition Documentation

39.7.4.1 GKO_CREATE_FACTORY_PARAMETERS

```
#define GKO_CREATE_FACTORY_PARAMETERS(
                _parameters_name,
                _factory_name )
```

Value:

```
public:                                                            \
    class _factory_name;                                           \
    struct _parameters_name##_type                                 \
        : public ::gko::enable_parameters_type<_parameters_name##_type, \
                                               _factory_name>
```

This Macro will generate a new type containing the parameters for the factory `_factory_name`.

For more details, see GKO_ENABLE_LIN_OP_FACTORY(). It is required to use this macro **before** calling the macro GKO_ENABLE_LIN_OP_FACTORY(). It is also required to use the same names for all parameters between both macros.

Parameters

_parameters_name	name of the parameters member in the class
_factory_name	name of the generated factory type

39.7.4.2 GKO_ENABLE_LIN_OP_FACTORY

```
#define GKO_ENABLE_LIN_OP_FACTORY(
```

_lin_op,
_parameters_name,
_factory_name)

This macro will generate a default implementation of a LinOpFactory for the LinOp subclass it is defined in.

It is required to first call the macro GKO_CREATE_FACTORY_PARAMETERS() before this one in order to instantiate the parameters type first.

The list of parameters for the factory should be defined in a code block after the macro definition, and should contain a list of GKO_FACTORY_PARAMETER_* declarations. The class should provide a constructor with signature ↩ _lin_op(const _factory_name *, std::shared_ptr<const LinOp>) which the factory will use a callback to construct the object.

A minimal example of a linear operator is the following:

```c++ struct MyLinOp : public EnableLinOp<MyLinOp> { GKO_ENABLE_LIN_OP_FACTORY(MyLinOp, my_parameters, Factory) { // a factory parameter named "my_value", of type int and default // value of 5 int GKO_FACTORY_PARAMETER_SCALAR(my_value, // a factory parameter named `my_pair` of type `std::pair<int,int>` // and default value {5, 5} std::pair<int, int> GKO_FACTORY_PARAMETER_VECTOR(my_pair, 5, 5); }; // constructor needed by EnableLinOp explicit MyLinOp(std::shared_ptr<const Executor> exec) { : EnableLinOp<MyLinOp>(exec) {} // constructor needed by the factory explicit MyLinOp(const Factory *factory, std::shared_ptr<const LinOp> matrix) : EnableLinOp<↩ MyLinOp>(factory->get_executor()), matrix->get_size()), // store factory's parameters locally my_parameters_↩ {factory->get_parameters()}, { int value = my_parameters_.my_value; // do something with value }

```
MyLinOp can then be created as follows:
"'c++
auto exec = gko::ReferenceExecutor::create();
// create a factory with default 'my_value' parameter
auto fact = MyLinOp::build().on(exec);
// create a operator using the factory:
auto my_op = fact->generate(gko::matrix::Identity::create(exec, 2));
std::cout « my_op->get_my_parameters().my_value;  // prints 5
// create a factory with custom 'my_value' parameter
auto fact = MyLinOp::build().with_my_value(0).on(exec);
// create a operator using the factory:
auto my_op = fact->generate(gko::matrix::Identity::create(exec, 2));
std::cout « my_op->get_my_parameters().my_value;  // prints 0
```

Note

It is possible to combine both the #GKO_CREATE_FACTORY_PARAMETER_*() macros with this one in a unique macro for class **templates** (not with regular classes). Splitting this into two distinct macros allows to use them in all contexts. See `https://stackoverflow.com/q/50202718/9385966` for more details.

Parameters

_lin_op	concrete operator for which the factory is to be created [CRTP parameter]
_parameters_name	name of the parameters member in the class (its type is `<_parameters_name>_type`, the protected member's name is `<_parameters_name>_`, and the public getter's name is `get_<_parameters_name>()`)
_factory_name	name of the generated factory type

39.7.5 Typedef Documentation
```

39.7.5.1 EnableDefaultLinOpFactory

```
template<typename ConcreteFactory , typename ConcreteLinOp , typename ParametersType , typename
PolymorphicBase = LinOpFactory>
using gko::EnableDefaultLinOpFactory = typedef EnableDefaultFactory<ConcreteFactory, Concrete←
LinOp, ParametersType, PolymorphicBase>
```

This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of LinOpFactory.

Template Parameters

ConcreteFactory	the concrete factory which is being implemented [CRTP parmeter]
ConcreteLinOp	the concrete LinOp type which this factory produces, needs to have a constructor which takes a const ConcreteFactory $*$, and an std::shared_ptr$<$const LinOp$>$ as parameters.
ParametersType	a subclass of enable_parameters_type template which defines all of the parameters of the factory
PolymorphicBase	parent of ConcreteFactory in the polymorphic hierarchy, has to be a subclass of LinOpFactory

39.8 Logging

A module dedicated to the implementation and usage of the Logging in Ginkgo.

Namespaces

- gko::log

  The logger namespace .

Classes

- class gko::log::BatchConvergence< ValueType >

  Logs the final residuals and iteration counts for a batch solver.
- class gko::log::Convergence< ValueType >

  Convergence is a Logger which logs data strictly from the `criterion_check_completed` event.
- class gko::log::Stream< ValueType >

  Stream is a Logger which logs every event to a stream.

39.8.1 Detailed Description

A module dedicated to the implementation and usage of the Logging in Ginkgo.

The Logger class represents a simple Logger object. It comprises all masks and events internally. Every new logging event addition should be done here. The Logger class also provides a default implementation for most events which do nothing, therefore it is not an obligation to change all classes which derive from Logger, although it is good practice. The logger class is built using event masks to control which events should be logged, and which should not.

39.9 SpMV employing different Matrix formats

A module dedicated to the implementation and usage of the various Matrix Formats in Ginkgo.

Classes

- class gko::matrix::BatchCsr< ValueType, IndexType >

    BatchCsr is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class gko::matrix::BatchDense< ValueType >

    BatchDense is a batch matrix format which explicitly stores all values of the matrix in each of the batches.
- class gko::matrix::BatchDiagonal< ValueType >

    BatchDiagonal is a batch matrix format which explicitly stores all values of the matrix in each of the batches.
- class gko::matrix::BatchEll< ValueType, IndexType >

    BatchEll is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class gko::matrix::Coo< ValueType, IndexType >

    COO stores a matrix in the coordinate matrix format.
- class gko::matrix::Csr< ValueType, IndexType >

    CSR is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class gko::matrix::Dense< ValueType >

    Dense is a matrix format which explicitly stores all values of the matrix.
- class gko::matrix::Diagonal< ValueType >

    This class is a utility which efficiently implements the diagonal matrix (a linear operator which scales a vector row wise).
- class gko::matrix::Ell< ValueType, IndexType >

    ELL is a matrix format where stride with explicit zeros is used such that all rows have the same number of stored elements.
- class gko::matrix::Fbcsr< ValueType, IndexType >

    Fixed-block compressed sparse row storage matrix format.
- class gko::matrix::Fft

    This LinOp implements a 1D Fourier matrix using the FFT algorithm.
- class gko::matrix::Fft2

    This LinOp implements a 2D Fourier matrix using the FFT algorithm.
- class gko::matrix::Fft3

    This LinOp implements a 3D Fourier matrix using the FFT algorithm.
- class gko::matrix::Hybrid< ValueType, IndexType >

    HYBRID is a matrix format which splits the matrix into ELLPACK and COO format.
- class gko::matrix::Identity< ValueType >

    This class is a utility which efficiently implements the identity matrix (a linear operator which maps each vector to itself).
- class gko::matrix::IdentityFactory< ValueType >

    This factory is a utility which can be used to generate Identity operators.
- class gko::matrix::Permutation< IndexType >

    Permutation is a matrix "format" which stores the row and column permutation arrays which can be used for re-ordering the rows and columns a matrix.
- class gko::matrix::Sellp< ValueType, IndexType >

    SELL-P is a matrix format similar to ELL format.
- class gko::matrix::SparsityCsr< ValueType, IndexType >

    SparsityCsr is a matrix format which stores only the sparsity pattern of a sparse matrix by compressing each row of the matrix (compressed sparse row format).

Functions

- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (std::vector< size_type > stride, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of column-vectors.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of column-vectors.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (std::vector< size_type > stride, std::initializer_list< std::initializer_list< std::initializer_list< typename Matrix::value_type >>> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of matrices.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (std::initializer_list< std::initializer_list< std::initializer_list< typename Matrix::value_type >>> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of matrices.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (std::vector< size_type > stride, const size_type num_vectors, std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch column-vector by making copies of the single input column vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (const size_type num_vectors, std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a column-vector from copies of a given vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (std::vector< size_type > stride, const size_type num_matrices, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix from copies of a given matrix.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::batch_initialize (const size_type num_matrices, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix from copies of a given matrix.
- template<typename ValueType >
  std::unique_ptr< matrix::BatchDiagonal< ValueType > > gko::batch_diagonal_initialize (std::initializer_list< std::initializer_list< ValueType >> vals, std::shared_ptr< const Executor > exec)

  Creates and initializes a batch of diagonal matrices.
- template<typename ValueType >
  std::unique_ptr< matrix::BatchDiagonal< ValueType > > gko::batch_diagonal_initialize (const size_type num_matrices, std::initializer_list< ValueType > vals, std::shared_ptr< const Executor > exec)

  Creates and initializes a batch diagonal matrix by making copies of the single input diagonal matrix.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::initialize (size_type stride, std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a column-vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::initialize (std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a column-vector.

- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::initialize (size_type stride, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > gko::initialize (std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix.

39.9.1 Detailed Description

A module dedicated to the implementation and usage of the various Matrix Formats in Ginkgo.

39.9.2 Function Documentation

39.9.2.1 batch_diagonal_initialize() [1/2]

```
template<typename ValueType >
std::unique_ptr<matrix::BatchDiagonal<ValueType> > gko::batch_diagonal_initialize (
            const size_type num_matrices,
            std::initializer_list< ValueType > vals,
            std::shared_ptr< const Executor > exec )
```

Creates and initializes a batch diagonal matrix by making copies of the single input diagonal matrix.

This function first creates a temporary batch matrix, fills it with passed in values, and then copies the matrix to the requested backend.

Template Parameters

Matrix	matrix type to initialize

Parameters

num_matrices	The number of times the input vector is copied into the final output
vals	values used to initialize each vector in the temp. batch
exec	Executor associated to the vector

```
{
    using batch_diag = matrix::BatchDiagonal<ValueType>;
    std::vector<size_type> num_rows(num_matrices);
    std::vector<dim<2>> sizes(num_matrices);
    for (size_type b = 0; b < num_matrices; ++b) {
        num_rows[b] = vals.size();
        sizes[b] = dim<2>(vals.size(), vals.size());
    }
    auto b_size = batch_dim<2>(sizes);
    auto tmp = batch_diag::create(exec->get_master(), b_size);
    for (size_type batch = 0; batch < num_matrices; batch++) {
        size_type idx = 0;
        for (const auto& elem : vals) {
            tmp->at(batch, idx) = elem;
```

```
            ++idx;
        }
    }
    auto mtx = batch_diag::create(exec);
    tmp->move_to(mtx.get());
    return mtx;
}
```

39.9.2.2 batch_diagonal_initialize() [2/2]

```
template<typename ValueType >
std::unique_ptr<matrix::BatchDiagonal<ValueType> > gko::batch_diagonal_initialize (
            std::initializer_list< std::initializer_list< ValueType >> vals,
            std::shared_ptr< const Executor > exec )
```

Creates and initializes a batch of diagonal matrices.

This function first creates a temporary matrix, fills it with passed in values, and then copies the matrix to the requested backend.

Template Parameters

Matrix	matrix type to initialize

Parameters

vals	values used to initialize the batch vector
exec	Executor associated to the vector

39.9.2.3 batch_initialize() [1/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            const size_type num_matrices,
            std::initializer_list< std::initializer_list< typename Matrix::value_type >>
vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a matrix from copies of a given matrix.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type. The stride of the intermediate Dense matrix is set to 1.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

num_vectors	The number of times the input vector is copied into the final output
vals	values used to initialize the vector
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

```
{
    auto strides = std::vector<size_type>(num_matrices, begin(vals)->size());
    return batch_initialize<Matrix>(strides, num_matrices, vals,
                                    std::move(exec),
                                    std::forward<TArgs>(create_args)...);
}
```

39.9.2.4 batch_initialize() [2/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            const size_type num_vectors,
            std::initializer_list< typename Matrix::value_type > vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a column-vector from copies of a given vector.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type. The stride of the intermediate Dense matrix is set to 1.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

num_vectors	The number of times the input vector is copied into the final output
vals	values used to initialize the vector
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.5 batch_initialize() [3/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            std::initializer_list< std::initializer_list< std::initializer_list< typename
Matrix::value_type >>> vals,
```

```
std::shared_ptr< const Executor > exec,
TArgs &&... create_args )
```

Creates and initializes a batch of matrices.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type. The stride of the intermediate Dense matrix is set to the number of columns of the initializer list.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

vals	values used to initialize the matrix
exec	Executor associated to the matrix
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.6 batch_initialize() [4/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            std::initializer_list< std::initializer_list< typename Matrix::value_type >>
vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a batch of column-vectors.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type. The stride of the intermediate Dense matrix is set to 1.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

vals	values used to initialize the vector
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.7 batch_initialize() [5/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            std::vector< size_type > stride,
            const size_type num_matrices,
            std::initializer_list< std::initializer_list< typename Matrix::value_type >>
vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a matrix from copies of a given matrix.

This function first creates a temporary batch dense matrix, fills it with passed in values, and then converts the matrix to the requested type.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

stride	row strides for the temporary batch dense matrix
num_matrices	The number of times the input matrix is copied into the final output
vals	values used to initialize each vector in the temp. batch
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.8 batch_initialize() [6/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            std::vector< size_type > stride,
            const size_type num_vectors,
            std::initializer_list< typename Matrix::value_type > vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a batch column-vector by making copies of the single input column vector.

This function first creates a temporary batch dense matrix, fills it with passed in values, and then converts the matrix to the requested type.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

stride	row strides for the temporary batch dense matrix
num_vectors	The number of times the input vector is copied into the final output
vals	values used to initialize each vector in the temp. batch
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.9 batch_initialize() [7/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            std::vector< size_type > stride,
            std::initializer_list< std::initializer_list< std::initializer_list< typename
Matrix::value_type >>> vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a batch of matrices.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

stride	row stride for the temporary Dense matrix
vals	values used to initialize the matrix
exec	Executor associated to the matrix
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.10 batch_initialize() [8/8]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::batch_initialize (
            std::vector< size_type > stride,
            std::initializer_list< std::initializer_list< typename Matrix::value_type >>
vals,
```

```
            std::shared_ptr< const Executor > exec,
            TArgs &&...   create_args )
```

Creates and initializes a batch of column-vectors.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

stride	row stride for the temporary Dense matrix
vals	values used to initialize the batch vector
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.11 initialize() [1/4]

```
template<typename Matrix , typename...  TArgs>
std::unique_ptr<Matrix> gko::initialize (
            size_type stride,
            std::initializer_list< std::initializer_list< typename Matrix::value_type >>
vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&...   create_args )
```

Creates and initializes a matrix.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

stride	row stride for the temporary Dense matrix
vals	values used to initialize the matrix
exec	Executor associated to the matrix
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

```
{
    using dense = matrix::Dense<typename Matrix::value_type>;
    size_type num_rows = vals.size();
    size_type num_cols = num_rows > 0 ? begin(vals)->size() : 1;
    auto tmp =
        dense::create(exec->get_master(), dim<2>{num_rows, num_cols}, stride);
    size_type ridx = 0;
    for (const auto& row : vals) {
        size_type cidx = 0;
        for (const auto& elem : row) {
            tmp->at(ridx, cidx) = elem;
            ++cidx;
        }
        ++ridx;
    }
    auto mtx = Matrix::create(exec, std::forward<TArgs>(create_args)...);
    tmp->move_to(mtx.get());
    return mtx;
}
```

References gko::matrix::Dense< ValueType >::at().

39.9.2.12 initialize() [2/4]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::initialize (
            size_type stride,
            std::initializer_list< typename Matrix::value_type > vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&... create_args )
```

Creates and initializes a column-vector.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

stride	row stride for the temporary Dense matrix
vals	values used to initialize the vector
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

References gko::matrix::Dense< ValueType >::at().

39.9.2.13 initialize() [3/4]

```
template<typename Matrix , typename... TArgs>
std::unique_ptr<Matrix> gko::initialize (
```

```
            std::initializer_list< std::initializer_list< typename Matrix::value_type >>
vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&...  create_args )
```

Creates and initializes a matrix.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type. The stride of the intermediate Dense matrix is set to the number of columns of the initializer list.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

vals	values used to initialize the matrix
exec	Executor associated to the matrix
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.9.2.14 initialize() [4/4]

```
template<typename Matrix , typename...  TArgs>
std::unique_ptr<Matrix> gko::initialize (
            std::initializer_list< typename Matrix::value_type > vals,
            std::shared_ptr< const Executor > exec,
            TArgs &&...  create_args )
```

Creates and initializes a column-vector.

This function first creates a temporary Dense matrix, fills it with passed in values, and then converts the matrix to the requested type. The stride of the intermediate Dense matrix is set to 1.

Template Parameters

Matrix	matrix type to initialize (Dense has to implement the ConvertibleTo<Matrix> interface)
TArgs	argument types for Matrix::create method (not including the implied Executor as the first argument)

Parameters

vals	values used to initialize the vector
exec	Executor associated to the vector
create_args	additional arguments passed to Matrix::create, not including the Executor, which is passed as the first argument

39.10 OpenMP Executor

A module dedicated to the implementation and usage of the OpenMP executor in Ginkgo.

Classes

- class gko::OmpExecutor

  This is the Executor subclass which represents the OpenMP device (typically CPU).

39.10.1 Detailed Description

A module dedicated to the implementation and usage of the OpenMP executor in Ginkgo.

39.11 Preconditioners

A module dedicated to the implementation and usage of the Preconditioners in Ginkgo.

Modules

- Jacobi Preconditioner

  A module dedicated to the implementation and usage of the Jacobi Preconditioner in Ginkgo.

Namespaces

- gko::preconditioner

  The Preconditioner namespace.

Classes

- class gko::Preconditionable

  A LinOp implementing this interface can be preconditioned.
- class gko::preconditioner::Ic< LSolverType, IndexType >

  The Incomplete Cholesky (IC) preconditioner solves the equation $LL^H * x = b$ for a given lower triangular matrix L and the right hand side b (can contain multiple right hand sides).
- class gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >

  The Incomplete LU (ILU) preconditioner solves the equation $LUx = b$ for a given lower triangular matrix L, an upper triangular matrix U and the right hand side b (can contain multiple right hand sides).
- class gko::preconditioner::Isai< IsaiType, ValueType, IndexType >

  The Incomplete Sparse Approximate Inverse (ISAI) Preconditioner generates an approximate inverse matrix for a given square matrix A, lower triangular matrix L, upper triangular matrix U or symmetric positive (spd) matrix B.
- class gko::preconditioner::Jacobi< ValueType, IndexType >

  A block-Jacobi preconditioner is a block-diagonal linear operator, obtained by inverting the diagonal blocks of the source operator.

39.11.1 Detailed Description

A module dedicated to the implementation and usage of the Preconditioners in Ginkgo.

39.12 Reference Executor

A module dedicated to the implementation and usage of the Reference executor in Ginkgo.

Classes

- class gko::ReferenceExecutor

    This is a specialization of the OmpExecutor, which runs the reference implementations of the kernels used for debugging purposes.

39.12.1 Detailed Description

A module dedicated to the implementation and usage of the Reference executor in Ginkgo.

39.13 Solvers

A module dedicated to the implementation and usage of the Solvers in Ginkgo.

Namespaces

- gko::solver

  The ginkgo Solve namespace.

Classes

- class gko::solver::BatchBicgstab< ValueType >

  BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.
- class gko::solver::BatchCg< ValueType >

  CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.
- class gko::solver::BatchDirect< ValueType >

  Solves a batch of linear systems using a vendor-provided dense batched direct solver.
- class gko::solver::BatchGmres< ValueType >

  GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.
- class gko::solver::BatchIdr< ValueType >

  IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.
- class gko::solver::BatchRichardson< ValueType >

  The (preconditioned) Richardson solver is an iterative method that uses a preconditioner to approximate the error of the current solution via the current (preconditioned) residual.
- class gko::solver::Bicg< ValueType >

  BICG or the Biconjugate gradient method is a Krylov subspace solver.
- class gko::solver::Bicgstab< ValueType >

  BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.
- class gko::solver::CbGmres< ValueType >

  CB-GMRES or the compressed basis generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.
- class gko::solver::Cg< ValueType >

  CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.
- class gko::solver::Cgs< ValueType >

  CGS or the conjugate gradient square method is an iterative type Krylov subspace method which is suitable for general systems.
- class gko::solver::Fcg< ValueType >

  FCG or the flexible conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.
- class gko::solver::Gmres< ValueType >

  GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.
- class gko::solver::Idr< ValueType >

  IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.
- class gko::solver::Ir< ValueType >

  Iterative refinement (IR) is an iterative method that uses another coarse method to approximate the error of the current solution via the current residual.
- class gko::solver::LowerTrs< ValueType, IndexType >

LowerTrs is the triangular solver which solves the system L x = b, when L is a lower triangular matrix.

- class gko::solver::Multigrid

  Multigrid methods have a hierarchy of many levels, whose corase level is a subset of the fine level, of the problem.

- class gko::solver::UpperTrs< ValueType, IndexType >

  UpperTrs is the triangular solver which solves the system U x = b, when U is an upper triangular matrix.

39.13.1 Detailed Description

A module dedicated to the implementation and usage of the Solvers in Ginkgo.

39.14 Stopping criteria

A module dedicated to the implementation and usage of the Stopping Criteria in Ginkgo.

Namespaces

- gko::stop
  The Stopping criterion namespace.

Classes

- class gko::stop::Combined
  The Combined class is used to combine multiple criterions together through an OR operation.
- class gko::stop::Iteration
  The Iteration class is a stopping criterion which stops the iteration process after a preset number of iterations.
- class gko::stop::ResidualNormBase< ValueType >
  The ResidualNormBase class provides a framework for stopping criteria related to the residual norm.
- class gko::stop::ResidualNorm< ValueType >
  The ResidualNorm class is a stopping criterion which stops the iteration process when the actual residual norm is below a certain threshold relative to.
- class gko::stop::ImplicitResidualNorm< ValueType >
  The ImplicitResidualNorm class is a stopping criterion which stops the iteration process when the implicit residual norm is below a certain threshold relative to.
- class gko::stop::ResidualNormReduction< ValueType >
  The ResidualNormReduction class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the initial residual, i.e.
- class gko::stop::RelativeResidualNorm< ValueType >
  The RelativeResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the right-hand side, i.e.
- class gko::stop::AbsoluteResidualNorm< ValueType >
  The AbsoluteResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold, i.e.
- class gko::stopping_status
  This class is used to keep track of the stopping status of one vector.
- class gko::stop::Time
  The Time class is a stopping criterion which stops the iteration process after a certain amout of time has passed.

Enumerations

- enum gko::stop::mode
  The mode for the residual norm criterion.

Functions

- template< typename FactoryContainer >
  std::shared_ptr< const CriterionFactory > gko::stop::combine (FactoryContainer &&factories)
  Combines multiple criterion factories into a single combined criterion factory.

39.14.1 Detailed Description

A module dedicated to the implementation and usage of the Stopping Criteria in Ginkgo.

39.14.2 Enumeration Type Documentation

39.14.2.1 mode

```
enum gko::stop::mode  [strong]
```

The mode for the residual norm criterion.

- absolute: Check for tolerance against residual norm. $||r|| < \tau$
- initial_resnorm: Check for tolerance relative to the initial residual norm. $\frac{||r||}{||r_0||} < \tau$
- rhs_resnorm: Check for tolerance relative to the rhs norm. $\frac{||r||}{||b||} < \tau$

```
65 { absolute, initial_resnorm, rhs_norm };
```

39.14.3 Function Documentation

39.14.3.1 combine()

```
template<typename FactoryContainer >
std::shared_ptr<const CriterionFactory> gko::stop::combine (
            FactoryContainer && factories )
```

Combines multiple criterion factories into a single combined criterion factory.

This function treats a singleton container as a special case and avoids creating an additional object and just returns the input factory.

Template Parameters

FactoryContainer	a random access container type

Parameters

factories	a list of factories to combined

Returns

a combined criterion factory if the input contains multiple factories or the input factory if the input contains only one factory

```
{
    switch (factories.size()) {
    case 0:
        GKO_NOT_SUPPORTED(nullptr);
        return nullptr;
    case 1:
        if (factories[0] == nullptr) {
            GKO_NOT_SUPPORTED(nullptr);
        }
        return factories[0];
    default:
        if (factories[0] == nullptr) {
            // first factory must be valid to capture executor
            GKO_NOT_SUPPORTED(nullptr);
            return nullptr;
        } else {
            auto exec = factories[0]->get_executor();
            return Combined::build()
                .with_criteria(std::forward<FactoryContainer>(factories))
                .on(exec);
        }
    }
}
```

39.15 BatchLinOp

Classes

- class gko::BatchLinOpFactory
  A BatchLinOpFactory represents a higher order mapping which transforms one batch linear operator into another.
- class gko::EnableBatchLinOp< ConcreteBatchLinOp, PolymorphicBase >
  The EnableBatchLinOp mixin can be used to provide sensible default implementations of the majority of the BatchLinOp and PolymorphicObject interface.
- class gko::BatchReadableFromMatrixData< ValueType, IndexType >
  A BatchLinOp implementing this interface can read its data from a matrix_data structure.
- class gko::BatchWritableToMatrixData< ValueType, IndexType >
  A BatchLinOp implementing this interface can write its data to a std::vector of matrix_data objects.
- class gko::matrix::BatchCsr< ValueType, IndexType >
  BatchCsr is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class gko::matrix::BatchDense< ValueType >
  BatchDense is a batch matrix format which explicitly stores all values of the matrix in each of the batches.
- class gko::matrix::BatchDiagonal< ValueType >
  BatchDiagonal is a batch matrix format which explicitly stores all values of the matrix in each of the batches.
- class gko::matrix::BatchEll< ValueType, IndexType >
  BatchEll is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class gko::solver::BatchBicgstab< ValueType >
  BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.
- class gko::solver::BatchCg< ValueType >
  CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.
- class gko::solver::BatchDirect< ValueType >
  Solves a batch of linear systems using a vendor-provided dense batched direct solver.
- class gko::solver::BatchGmres< ValueType >
  GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.
- class gko::solver::BatchIdr< ValueType >
  IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.
- class gko::solver::BatchRichardson< ValueType >
  The (preconditioned) Richardson solver is an iterative method that uses a preconditioner to approximate the error of the current solution via the current (preconditioned) residual.

Macros

- #define GKO_ENABLE_BATCH_LIN_OP_FACTORY(_batch_lin_op, _parameters_name, _factory_name)
  This macro will generate a default implementation of a BatchLinOpFactory for the BatchLinOp subclass it is defined in.
- #define GKO_ENABLE_BUILD_METHOD(_factory_name)
  Defines a build method for the factory, simplifying its construction by removing the repetitive typing of factory's name.
- #define GKO_FACTORY_PARAMETER(_name, ...)
  Creates a factory parameter in the factory parameters structure.
- #define GKO_FACTORY_PARAMETER_SCALAR(_name, _default) GKO_FACTORY_PARAMETER(_name, _default)
  Creates a scalar factory parameter in the factory parameters structure.
- #define GKO_FACTORY_PARAMETER_VECTOR(_name, ...) GKO_FACTORY_PARAMETER(_name, __VA_ARGS__)
  Creates a vector factory parameter in the factory parameters structure.

Typedefs

- template<typename ConcreteFactory , typename ConcreteBatchLinOp , typename ParametersType , typename PolymorphicBase = BatchLinOpFactory>
  using gko::EnableDefaultBatchLinOpFactory = EnableDefaultFactory< ConcreteFactory, ConcreteBatch↩ LinOp, ParametersType, PolymorphicBase >

  This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of BatchLinOpFactory.

39.15.1 Detailed Description

39.15.2 Batched Linear operator as a concept

A batch linear operator (BatchLinOp) forms the base class for all batched linear algebra objects. In general, it follows the same structure as the LinOp class, but has some crucial differences which make it not strictly representable through or with the LinOp class.

A batched operator is defined as a set of independent linear operators which have no communication/information exchange between them. Therefore, any collective operations between the batches is not possible and not implemented. This allows for each batch to be computed and operated on in an embarrasingly parallel fashion.

Similar to the LinOp class, the BatchLinOp also implements BatchLinOp::apply() methods which call the internal apply_impl() methods which the concrete BatchLinOp's have to implement.

A key difference between the LinOp and the BatchLinOp classes is the storing of dimensions. BatchLinOp allows for storing non-equal objects in the batches and hence stores a batch_dim object instead of a dim object. The batch_dim object is optimized to store less amount of data when storing uniform batches.

All size validation functions again verify first that the number of batches are conformant and that the dimensions in the corresponding batches themselves are also valid/conformant. Here too, optimizations for uniform batches have been added.

BatchLinOp

39.15.3 Macro Definition Documentation

39.15.3.1 GKO_ENABLE_BATCH_LIN_OP_FACTORY

```
#define GKO_ENABLE_BATCH_LIN_OP_FACTORY(
              _batch_lin_op,
              _parameters_name,
              _factory_name )
```

This macro will generate a default implementation of a BatchLinOpFactory for the BatchLinOp subclass it is defined in.

It is required to first call the macro GKO_CREATE_FACTORY_PARAMETERS() before this one in order to instantiate the parameters type first.

The list of parameters for the factory should be defined in a code block after the macro definition, and should contain a list of GKO_FACTORY_PARAMETER_* declarations. The class should provide a constructor with signature _↩ batch_lin_op(const _factory_name *, std::shared_ptr<const BatchLinOp>) which the factory will use a callback to construct the object.

A minimal example of a batch linear operator is the following:

```c++ struct MyBatchLinOp : public EnableBatchLinOp<MyBatchLinOp> { GKO_ENABLE_BATCH_LIN_OP_FACTORY(MyBatchLin { // a factory parameter named "my_value", of type int and default // value of 5 int GKO_FACTORY_PARAMETER_SCALAR(my_value, // a factory parameter named `my_pair` of type `std::pair<int,int>` // and default value {5, 5} std::pair<int, int> GKO_FACTORY_PARAMETER_VECTOR(my_pair, 5, 5); }; // constructor needed by EnableBatchLinOp explicit MyBatchLinOp(std::shared_ptr<const Executor> exec) { : EnableBatchLinOp<MyBatchLinOp>(exec) {} // constructor needed by the factory explicit MyBatchLinOp(const Factory *factory, std::shared_ptr<const Batch↩ LinOp> matrix) : EnableBatchLinOp<MyBatchLinOp>(factory->get_executor()), matrix->get_size()), // store factory's parameters locally my_parameters_{factory->get_parameters()} { int value = my_parameters_.my_value; // do something with value }

```
MyBatchLinOp can then be created as follows:
``c++
auto exec = gko::ReferenceExecutor::create();
// create a factory with default 'my_value' parameter
auto fact = MyBatchLinOp::build().on(exec);
// create a operator using the factory:
auto my_op = fact->generate(gko::matrix::BatchIdentity::create(exec, 2));
std::cout « my_op->get_my_parameters().my_value;  // prints 5
// create a factory with custom 'my_value' parameter
auto fact = MyLinOp::build().with_my_value(0).on(exec);
// create a operator using the factory:
auto my_op = fact->generate(gko::matrix::BatchIdentity::create(exec, 2));
std::cout « my_op->get_my_parameters().my_value;  // prints 0
```

Note

It is possible to combine both the #GKO_CREATE_FACTORY_PARAMETER_*() macros with this one in a unique macro for class **templates** (not with regular classes). Splitting this into two distinct macros allows to use them in all contexts. See `https://stackoverflow.com/q/50202718/9385966` for more details.

Parameters

_lin_op	concrete operator for which the factory is to be created [CRTP parameter]
_parameters_name	name of the parameters member in the class (its type is `<_parameters_name>_type`, the protected member's name is `<_parameters_name>_`, and the public getter's name is `get_<_parameters_name>()`)
_factory_name	name of the generated factory type

39.15.3.2 GKO_ENABLE_BUILD_METHOD

```
#define GKO_ENABLE_BUILD_METHOD(
              _factory_name )
```

Value:
```
static auto build()->decltype(_factory_name::create())                \
    {                                                                    \
        return _factory_name::create();                                  \
    }                                                                    \
    static_assert(true,                                                  \
                  "This assert is used to counter the false positive extra " \
                  "semi-colon warnings")
```

Defines a build method for the factory, simplifying its construction by removing the repetitive typing of factory's name.
```

Parameters

_factory_name	the factory for which to define the method

39.15.3.3 GKO_FACTORY_PARAMETER

```
#define GKO_FACTORY_PARAMETER(
            _name,
            ... )
```

Value:

```
mutable _name{__VA_ARGS__};                                              \
                                                                         \
    template <typename... Args>                                          \
    auto with_##_name(Args&&... _value)                                  \
        const->const std::decay_t<decltype(*this)>&                      \
    {                                                                    \
        using type = decltype(this->_name);                              \
        this->_name = type{std::forward<Args>(_value)...};               \
        return *this;                                                    \
    }                                                                    \
    static_assert(true,                                                  \
                  "This assert is used to counter the false positive extra " \
                  "semi-colon warnings")
```

Creates a factory parameter in the factory parameters structure.

Parameters

_name	name of the parameter
VA_ARGS	default value of the parameter

See also

GKO_ENABLE_LIN_OP_FACTORY for more details, and usage example

39.15.3.4 GKO_FACTORY_PARAMETER_SCALAR

```
#define GKO_FACTORY_PARAMETER_SCALAR(
            _name,
            _default ) GKO_FACTORY_PARAMETER(_name, _default)
```

Creates a scalar factory parameter in the factory parameters structure.

Scalar in this context means that the constructor for this type only takes a single parameter.

Parameters

_name	name of the parameter
_default	default value of the parameter

See also

GKO_ENABLE_LIN_OP_FACTORY for more details, and usage example

39.15.3.5 GKO_FACTORY_PARAMETER_VECTOR

```
#define GKO_FACTORY_PARAMETER_VECTOR(
              _name,
              ... ) GKO_FACTORY_PARAMETER(_name, __VA_ARGS__)
```

Creates a vector factory parameter in the factory parameters structure.

Vector in this context means that the constructor for this type takes multiple parameters.

Parameters

_name	name of the parameter
_default	default value of the parameter

See also

GKO_ENABLE_LIN_OP_FACTORY for more details, and usage example

39.15.4 Typedef Documentation

39.15.4.1 EnableDefaultBatchLinOpFactory

```
template<typename ConcreteFactory , typename ConcreteBatchLinOp , typename ParametersType ,
typename PolymorphicBase = BatchLinOpFactory>
using gko::EnableDefaultBatchLinOpFactory = typedef EnableDefaultFactory<ConcreteFactory,
ConcreteBatchLinOp, ParametersType, PolymorphicBase>
```

This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of BatchLinOpFactory.

Template Parameters

ConcreteFactory	the concrete factory which is being implemented [CRTP parmeter]
ConcreteLinOp	the concrete BatchLinOp type which this factory produces, needs to have a constructor which takes a const ConcreteFactory $*$, and an std::shared_ptr$<$const BatchLinOp$>$ as parameters.
ParametersType	a subclass of enable_parameters_type template which defines all of the parameters of the factory
PolymorphicBase	parent of ConcreteFactory in the polymorphic hierarchy, has to be a subclass of LinOpFactory

Chapter 40

Namespace Documentation

40.1 gko Namespace Reference

The Ginkgo namespace.

Namespaces

- accessor

  The accessor namespace.
- factorization

  The Factorization namespace.
- log

  The logger namespace .
- matrix

  The matrix namespace.
- multigrid

  The multigrid components namespace.
- name_demangling

  The name demangling namespace.
- preconditioner

  The Preconditioner namespace.
- reorder

  The Reorder namespace.
- solver

  The ginkgo Solve namespace.
- stop

  The Stopping criterion namespace.
- syn

  The Synthesizer namespace.
- xstd

  The namespace for functionalities after C++14 standard.

Classes

- class AbsoluteComputable

  The AbsoluteComputable is an interface that allows to get the component wise absolute of a LinOp.
- class AbstractFactory

  The AbstractFactory is a generic interface template that enables easy implementation of the abstract factory design pattern.
- class AllocationError

  AllocationError is thrown if a memory allocation fails.
- class amd_device

  amd_device handles the number of executor on Amd devices and have the corresponding recursive_mutex.
- struct are_all_integral

  Evaluates if all template arguments Args fulfill std::is_integral.
- class Array

  An Array is a container which encapsulates fixed-sized arrays, stored on the Executor tied to the Array.
- class BadDimension

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

  A type representing the dimensions of a multidimensional batch object.
- class BatchLinOpFactory

  A BatchLinOpFactory represents a higher order mapping which transforms one batch linear operator into another.
- class BatchReadableFromMatrixData

  A BatchLinOp implementing this interface can read its data from a matrix_data structure.
- class BatchScalable

  A batch made up of entities that can scaled from the left and right by scaling vectors.
- class BatchScaledIdentityAddable

  Adds the operation M <- a I + b M for batch matrix M, batch identity operator I and batch scalars a and b.
- class BatchTransposable

  Batch Linear operators which support transposition of the distinct batch entries should implement the BatchTransposable interface.
- class BatchWritableToMatrixData

  A BatchLinOp implementing this interface can write its data to a std::vector of matrix_data objects.
- class BlockSizeError

  Error that denotes issues between block sizes and matrix dimensions.
- class Combination

  The Combination class can be used to construct a linear combination of multiple linear operators `c1 * op1 + c2 * op2 + ...`
- class Composition

  The Composition class can be used to compose linear operators `op1, op2, ..., opn` *and obtain the operator* `op1 * op2 * ...`
- class ConvertibleTo

  ConvertibleTo interface is used to mark that the implementer can be converted to the object of ResultType.
- struct cpx_real_type

  Access the underlying real type of a complex number.
- class CublasError

  CublasError is thrown when a cuBLAS routine throws a non-zero error code.
- class CudaError

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

  This is the Executor subclass which represents the CUDA device.
- class CufftError

  CufftError is thrown when a cuFFT routine throws a non-zero error code.

- class CurandError

    CurandError is thrown when a cuRAND routine throws a non-zero error code.
- class CusparseError

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

    Used to convert objects of type `S` to objects of type `R` using static_cast.
- class device_matrix_data

    This type is a device-side equivalent to matrix_data.
- class DiagonalExtractable

    The diagonal of a LinOp implementing this interface can be extracted.
- class DiagonalLinOpExtractable

    The diagonal of a LinOp can be extracted.
- struct dim

    A type representing the dimensions of a multidimensional object.
- class DimensionMismatch

    DimensionMismatch is thrown if an operation is being applied to LinOps of incompatible size.
- class DpcppExecutor

    This is the Executor subclass which represents a DPC++ enhanced device.
- class enable_parameters_type

    The enable_parameters_type mixin is used to create a base implementation of the factory parameters structure.
- class EnableAbsoluteComputation

    The EnableAbsoluteComputation mixin provides the default implementations of `compute_absolute_linop` and the absolute interface.
- class EnableAbstractPolymorphicObject

    This mixin inherits from (a subclass of) PolymorphicObject and provides a base implementation of a new abstract object.
- class EnableBatchLinOp

    The EnableBatchLinOp mixin can be used to provide sensible default implementations of the majority of the Batch←↩ LinOp and PolymorphicObject interface.
- class EnableBatchScaling

    Default batch scalable interface with some type-checking.
- class EnableCreateMethod

    This mixin implements a static `create()` method on `ConcreteType` that dynamically allocates the memory, uses the passed-in arguments to construct the object, and returns an std::unique_ptr to such an object.
- class EnableDefaultFactory

    This mixin provides a default implementation of a concrete factory.
- class EnableLinOp

    The EnableLinOp mixin can be used to provide sensible default implementations of the majority of the LinOp and PolymorphicObject interface.
- class EnablePolymorphicAssignment

    This mixin is used to enable a default PolymorphicObject::copy_from() implementation for objects that have implemented conversions between them.
- class EnablePolymorphicObject

    This mixin inherits from (a subclass of) PolymorphicObject and provides a base implementation of a new concrete polymorphic object.
- class Error

    The Error class is used to report exceptional behaviour in library functions.
- class Executor

    The first step in using the Ginkgo library consists of creating an executor.
- class executor_deleter

    This is a deleter that uses an executor's `free` method to deallocate the data.
- class HipblasError

HipblasError is thrown when a hipBLAS routine throws a non-zero error code.
- class HipError

  HipError is thrown when a HIP routine throws a non-zero error code.
- class HipExecutor

  This is the Executor subclass which represents the HIP enhanced device.
- class HipfftError

  HipfftError is thrown when a hipFFT routine throws a non-zero error code.
- class HiprandError

  HiprandError is thrown when a hipRAND routine throws a non-zero error code.
- class HipsparseError

  HipsparseError is thrown when a hipSPARSE routine throws a non-zero error code.
- class index_set

  An index set class represents an ordered set of intervals.
- class KernelNotFound

  KernelNotFound is thrown if Ginkgo cannot find a kernel which satisfies the criteria imposed by the input arguments.
- class LinOpFactory

  A LinOpFactory represents a higher order mapping which transforms one linear operator into another.
- class MachineTopology

  The machine topology class represents the hierarchical topology of a machine, including NUMA nodes, cores and PCI Devices.
- class matrix_assembly_data

  This structure is used as an intermediate type to assemble a sparse matrix.
- struct matrix_data

  This structure is used as an intermediate data type to store a sparse matrix.
- struct matrix_data_entry

  Type used to store nonzeros.
- class MpiError

  MpiError is thrown when a MPI routine throws a non-zero error code.
- class NotCompiled

  NotCompiled is thrown when attempting to call an operation which is a part of a module that was not compiled on the system.
- class NotImplemented

  NotImplemented is thrown in case an operation has not yet been implemented (but will be implemented in the future).
- class NotSupported

  NotSupported is thrown in case it is not possible to perform the requested operation on the given object type.
- class null_deleter

  This is a deleter that does not delete the object.
- class nvidia_device

  nvidia_device handles the number of executor on Nvidia devices and have the corresponding recursive_mutex.
- class OmpExecutor

  This is the Executor subclass which represents the OpenMP device (typically CPU).
- class Operation

  Operations can be used to define functionalities whose implementations differ among devices.
- class OutOfBoundsError

  OutOfBoundsError is thrown if a memory access is detected to be out-of-bounds.
- class Permutable

  Linear operators which support permutation should implement the Permutable interface.
- class Perturbation

  The Perturbation class can be used to construct a LinOp to represent the operation `(identity + scalar * basis * projector)`.
- class PolymorphicObject

A PolymorphicObject is the abstract base for all "heavy" objects in Ginkgo that behave polymorphically.
- class precision_reduction

  This class is used to encode storage precisions of low precision algorithms.
- class Preconditionable

  A LinOp implementing this interface can be preconditioned.
- class range

  A range is a multidimensional view of the memory.
- class ReadableFromMatrixData

  A LinOp implementing this interface can read its data from a matrix_data structure.
- class ReferenceExecutor

  This is a specialization of the OmpExecutor, which runs the reference implementations of the kernels used for debugging purposes.
- class ScaledIdentityAddable

  Adds the operation M <- a I + b M for matrix M, identity operator I and scalars a and b, where M is the calling object.
- struct span

  A span is a lightweight structure used to create sub-ranges from other ranges.
- class stopping_status

  This class is used to keep track of the stopping status of one vector.
- class StreamError

  StreamError is thrown if accessing a stream failed.
- class Transposable

  Linear operators which support transposition should implement the Transposable interface.
- class UnsupportedMatrixProperty

  Exception throws if a matrix does not have a property required by a numerical method.
- class UseComposition

  The UseComposition class can be used to store the composition information in LinOp.
- class ValueMismatch

  ValueMismatch is thrown if two values are not equal.
- struct version

  This structure is used to represent versions of various Ginkgo modules.
- class version_info

  Ginkgo uses version numbers to label new features and to communicate backward compatibility guarantees:
- class WritableToMatrixData

  A LinOp implementing this interface can write its data to a matrix_data structure.

Typedefs

- template< typename ConcreteFactory , typename ConcreteBatchLinOp , typename ParametersType , typename PolymorphicBase = BatchLinOpFactory>
  using EnableDefaultBatchLinOpFactory = EnableDefaultFactory< ConcreteFactory, ConcreteBatchLinOp, ParametersType, PolymorphicBase >

  This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of BatchLinOpFactory.
- template< typename ConcreteFactory , typename ConcreteLinOp , typename ParametersType , typename PolymorphicBase = LinOp↩Factory>
  using EnableDefaultLinOpFactory = EnableDefaultFactory< ConcreteFactory, ConcreteLinOp, Parameters↩Type, PolymorphicBase >

  This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of LinOpFactory.
- template< typename T >
  using is_complex_s = detail::is_complex_impl< T >

Allows to check if T is a complex value during compile time by accessing the `value` attribute of this struct.

- template<typename T >
  using is_complex_or_scalar_s = detail::is_complex_or_scalar_impl< T >

  Allows to check if T is a complex or scalar value during compile time by accessing the `value` attribute of this struct.
- template<typename T >
  using remove_complex = typename detail::remove_complex_s< T >::type

  Obtain the type which removed the complex of complex/scalar type or the template parameter of class by accessing the `type` attribute of this struct.
- template<typename T >
  using to_complex = typename detail::to_complex_s< T >::type

  Obtain the type which adds the complex of complex/scalar type or the template parameter of class by accessing the `type` attribute of this struct.
- template<typename T >
  using to_real = remove_complex< T >

  to_real is alias of remove_complex
- template<typename T >
  using next_precision = typename detail::next_precision_impl< T >::type

  Obtains the next type in the singly-linked precision list.
- template<typename T >
  using reduce_precision = typename detail::reduce_precision_impl< T >::type

  Obtains the next type in the hierarchy with lower precision than T.
- template<typename T >
  using increase_precision = typename detail::increase_precision_impl< T >::type

  Obtains the next type in the hierarchy with higher precision than T.
- template<typename... Ts>
  using highest_precision = typename detail::highest_precision_variadic< Ts... >::type

  Obtains the smallest arithmetic type that is able to store elements of all template parameter types exactly.
- template<typename T , size_type Limit = sizeof(uint16) * byte_size>
  using truncate_type = std::conditional_t< detail::type_size_impl< T >::value >=2 *Limit, typename detail↩
  ::truncate_type_impl< T >::type, T >

  Truncates the type by half (by dropping bits), but ensures that it is at least `Limit` bits wide.
- using size_type = std::size_t

  Integral type used for allocation quantities.
- using int8 = std::int8_t

  8-bit signed integral type.
- using int16 = std::int16_t

  16-bit signed integral type.
- using int32 = std::int32_t

  32-bit signed integral type.
- using int64 = std::int64_t

  64-bit signed integral type.
- using uint8 = std::uint8_t

  8-bit unsigned integral type.
- using uint16 = std::uint16_t

  16-bit unsigned integral type.
- using uint32 = std::uint32_t

  32-bit unsigned integral type.
- using uint64 = std::uint64_t

  64-bit unsigned integral type.
- using float16 = half

  Half precision floating point type.
- using float32 = float

Single precision floating point type.
- using float64 = double

  Double precision floating point type.
- using full_precision = double

  The most precise floating-point type.
- using default_precision = double

  Precision used if no precision is explicitly specified.

Enumerations

- enum allocation_mode

  Specify the mode of allocation for CUDA/HIP GPUs.
- enum layout_type { layout_type::array, layout_type::coordinate }

  Specifies the layout type when writing data in matrix market format.

Functions

- template< typename ValueType >
  ValueType reduce_add (const Array< ValueType > &input_arr, const ValueType init_val=0)

  Reduce (sum) the values in the array.
- template< typename ValueType >
  void reduce_add (const Array< ValueType > &input_arr, Array< ValueType > &result)

  Reduce (sum) the values in the array.
- template< typename ValueType >
  Array< ValueType > make_array_view (std::shared_ptr< const Executor > exec, size_type size, ValueType ∗data)

  Helper function to create an array view deducing the value type.
- template< size_type Dimensionality, typename DimensionType >
  constexpr bool operator!= (const dim< Dimensionality, DimensionType > &x, const dim< Dimensionality, DimensionType > &y)

  Checks if two dim objects are different.
- template< typename DimensionType >
  constexpr dim< 2, DimensionType > transpose (const dim< 2, DimensionType > &dimensions) noexcept

  Returns a dim<2> object with its dimensions swapped.
- template< size_type Dimensionality, typename DimensionType >
  bool operator!= (const batch_dim< Dimensionality, DimensionType > &x, const batch_dim< Dimensionality, DimensionType > &y)

  Checks if two batch dim objects are different.
- template< typename DimensionType >
  batch_dim< 2, DimensionType > transpose (const batch_dim< 2, DimensionType > &input)

  Returns a batch_dim object with its dimensions swapped for batched operators.
- template< typename T >
  constexpr bool is_complex ()

  Checks if T is a complex type.
- template< typename T >
  constexpr bool is_complex_or_scalar ()

  Checks if T is a complex/scalar type.
- template< typename T >
  constexpr reduce_precision< T > round_down (T val)

  Reduces the precision of the input parameter.

- template< typename T >
  constexpr increase_precision< T > round_up (T val)

  Increases the precision of the input parameter.
- constexpr int64 ceildiv (int64 num, int64 den)

  Performs integer division with rounding up.
- template< typename T >
  constexpr T zero ()

  Returns the additive identity for T.
- template< typename T >
  constexpr T zero (const T &)

  Returns the additive identity for T.
- template< typename T >
  constexpr T one ()

  Returns the multiplicative identity for T.
- template< typename T >
  constexpr T one (const T &)

  Returns the multiplicative identity for T.
- template< typename T >
  constexpr bool is_zero (T value)

  Returns true if and only if the given value is zero.
- template< typename T >
  constexpr bool is_nonzero (T value)

  Returns true if and only if the given value is not zero.
- template< typename T >
  constexpr T max (const T &x, const T &y)

  Returns the larger of the arguments.
- template< typename T >
  constexpr T min (const T &x, const T &y)

  Returns the smaller of the arguments.
- template< typename T >
  constexpr auto real (const T &x)

  Returns the real part of the object.
- template< typename T >
  constexpr auto imag (const T &x)

  Returns the imaginary part of the object.
- template< typename T >
  constexpr auto conj (const T &x)

  Returns the conjugate of an object.
- template< typename T >
  constexpr auto squared_norm (const T &x) -> decltype(real(conj(x) *x))

  Returns the squared norm of the object.
- template< typename T >
  constexpr xstd::enable_if_t<!is_complex_s< T >::value, T > abs (const T &x)

  Returns the absolute value of the object.
- template< typename T >
  constexpr T pi ()

  Returns the value of pi.
- template< typename T >
  constexpr std::complex< remove_complex< T > > unit_root (int64 n, int64 k=1)

  *Returns the value of exp(2 * pi * i * k / n), i.e.*
- template< typename T >
  constexpr uint32 get_significant_bit (const T &n, uint32 hint=0u) noexcept

  Returns the position of the most significant bit of the number.

- template<typename T >
  constexpr T get_superior_power (const T &base, const T &limit, const T &hint=T{1}) noexcept

  Returns the smallest power of `base` not smaller than `limit`.
- template<typename T >
  std::enable_if_t<!is_complex_s< T >::value, bool > is_finite (const T &value)

  Checks if a floating point number is finite, meaning it is neither +/- infinity nor NaN.
- template<typename T >
  std::enable_if_t< is_complex_s< T >::value, bool > is_finite (const T &value)

  Checks if all components of a complex value are finite, meaning they are neither +/- infinity nor NaN.
- template<typename T >
  T safe_divide (T a, T b)

  Computes the quotient of the given parameters, guarding against division by zero.
- template<typename T >
  std::enable_if_t<!is_complex_s< T >::value, bool > is_nan (const T &value)

  Checks if a floating point number is NaN.
- template<typename T >
  std::enable_if_t< is_complex_s< T >::value, bool > is_nan (const T &value)

  Checks if any component of a complex value is NaN.
- template<typename T >
  constexpr std::enable_if_t<!is_complex_s< T >::value, T > nan ()

  Returns a quiet NaN of the given type.
- template<typename T >
  constexpr std::enable_if_t< is_complex_s< T >::value, T > nan ()

  Returns a complex with both components quiet NaN.
- template<typename ValueType = default_precision, typename IndexType = int32>
  matrix_data< ValueType, IndexType > read_raw (std::istream &is)

  Reads a matrix stored in matrix market format from an input stream.
- template<typename ValueType = default_precision, typename IndexType = int32>
  matrix_data< ValueType, IndexType > read_binary_raw (std::istream &is)

  Reads a matrix stored in Ginkgo's binary matrix format from an input stream.
- template<typename ValueType = default_precision, typename IndexType = int32>
  matrix_data< ValueType, IndexType > read_generic_raw (std::istream &is)

  Reads a matrix stored in either binary or matrix market format from an input stream.
- template<typename ValueType , typename IndexType >
  void write_raw (std::ostream &os, const matrix_data< ValueType, IndexType > &data, layout_type layout=layout_type::array)

  Writes a matrix_data structure to a stream in matrix market format.
- template<typename ValueType , typename IndexType >
  void write_binary_raw (std::ostream &os, const matrix_data< ValueType, IndexType > &data)

  Writes a matrix_data structure to a stream in binary format.
- template<typename MatrixType , typename StreamType , typename... MatrixArgs>
  std::unique_ptr< MatrixType > read (StreamType &&is, MatrixArgs &&... args)

  Reads a matrix stored in matrix market format from an input stream.
- template<typename MatrixType , typename StreamType , typename... MatrixArgs>
  std::unique_ptr< MatrixType > read_binary (StreamType &&is, MatrixArgs &&... args)

  Reads a matrix stored in binary format from an input stream.
- template<typename MatrixType , typename StreamType , typename... MatrixArgs>
  std::unique_ptr< MatrixType > read_generic (StreamType &&is, MatrixArgs &&... args)

  Reads a matrix stored either in binary or matrix market format from an input stream.
- template<typename MatrixType , typename StreamType >
  void write (StreamType &&os, MatrixType *matrix, layout_type layout=detail::mtx_io_traits< std::remove_↩
  const_t< MatrixType >>::default_layout)

  Writes a matrix into an output stream in matrix market format.

- template<typename MatrixType , typename StreamType >
  void write_binary (StreamType &&os, MatrixType ∗matrix)

  Writes a matrix into an output stream in binary format.
- template<typename R , typename T >
  std::unique_ptr< R, std::function< void(R ∗)> > copy_and_convert_to (std::shared_ptr< const Executor > exec, T ∗obj)

  Converts the object to R and places it on Executor exec.
- template<typename R , typename T >
  std::unique_ptr< const R, std::function< void(const R ∗)> > copy_and_convert_to (std::shared_ptr< const Executor > exec, const T ∗obj)

  Converts the object to R and places it on Executor exec.
- template<typename R , typename T >
  std::shared_ptr< R > copy_and_convert_to (std::shared_ptr< const Executor > exec, std::shared_ptr< T > obj)

  Converts the object to R and places it on Executor exec.
- template<typename R , typename T >
  std::shared_ptr< const R > copy_and_convert_to (std::shared_ptr< const Executor > exec, std::shared_ptr< const T > obj)
- template<typename ValueType >
  detail::temporary_conversion< matrix::Dense< ValueType > > make_temporary_conversion (LinOp ∗matrix)

  Convert the given LinOp from matrix::Dense<...> to matrix::Dense< ValueType>.
- template<typename ValueType >
  detail::temporary_conversion< const matrix::Dense< ValueType > > make_temporary_conversion (const LinOp ∗matrix)

  Convert the given LinOp from matrix::Dense<...> to matrix::Dense< ValueType>.
- template<typename ValueType , typename Function , typename... Args>
  void precision_dispatch (Function fn, Args ∗... linops)

  Calls the given function with each given argument LinOp temporarily converted into matrix::Dense< ValueType> as parameters.
- template<typename ValueType , typename Function >
  void precision_dispatch_real_complex (Function fn, const LinOp ∗in, LinOp ∗out)

  Calls the given function with the given LinOps temporarily converted to matrix::Dense< ValueType>∗ as parameters.
- template<typename ValueType , typename Function >
  void precision_dispatch_real_complex (Function fn, const LinOp ∗alpha, const LinOp ∗in, LinOp ∗out)

  Calls the given function with the given LinOps temporarily converted to matrix::Dense< ValueType>∗ as parameters.
- template<typename ValueType , typename Function >
  void precision_dispatch_real_complex (Function fn, const LinOp ∗alpha, const LinOp ∗in, const LinOp ∗beta, LinOp ∗out)

  Calls the given function with the given LinOps temporarily converted to matrix::Dense< ValueType>∗ as parameters.
- template<typename ValueType , typename Function >
  void mixed_precision_dispatch (Function fn, const LinOp ∗in, LinOp ∗out)

  Calls the given function with each given argument LinOp converted into matrix::Dense< ValueType> as parameters.
- template<typename ValueType , typename Function , std::enable_if_t< is_complex< ValueType >()> ∗ = nullptr>
  void mixed_precision_dispatch_real_complex (Function fn, const LinOp ∗in, LinOp ∗out)

  Calls the given function with the given LinOps cast to their dynamic type matrix::Dense< ValueType>∗ as parameters.
- template<typename T >
  detail::temporary_clone< T > make_temporary_clone (std::shared_ptr< const Executor > exec, T ∗ptr)

  Creates a temporary_clone.
- template<typename T >
  detail::temporary_clone< T > make_temporary_output_clone (std::shared_ptr< const Executor > exec, T ∗ptr)

  Creates a uninitialized temporary_clone that will be copied back to the input afterwards.
- constexpr bool operator== (precision_reduction x, precision_reduction y) noexcept

Checks if two precision_reduction encodings are equal.

- constexpr bool operator!= (precision_reduction x, precision_reduction y) noexcept

  Checks if two precision_reduction encodings are different.

- template<typename IndexType >
  constexpr IndexType invalid_index ()

  Value for an invalid signed index type.

- template<typename Pointer >
  detail::cloned_type< Pointer > clone (const Pointer &p)

  Creates a unique clone of the object pointed to by `p`.

- template<typename Pointer >
  detail::cloned_type< Pointer > clone (std::shared_ptr< const Executor > exec, const Pointer &p)

  Creates a unique clone of the object pointed to by `p` *on Executor* `exec`.

- template<typename OwningPointer >
  detail::shared_type< OwningPointer > share (OwningPointer &&p)

  Marks the object pointed to by `p` *as shared.*

- template<typename OwningPointer >
  std::remove_reference< OwningPointer >::type && give (OwningPointer &&p)

  Marks that the object pointed to by `p` *can be given to the callee.*

- template<typename Pointer >
  std::enable_if< detail::have_ownership_s< Pointer >::value, detail::pointee< Pointer > * >::type lend (const Pointer &p)

  Returns a non-owning (plain) pointer to the object pointed to by `p`.

- template<typename Pointer >
  std::enable_if<!detail::have_ownership_s< Pointer >::value, detail::pointee< Pointer > * >::type lend (const Pointer &p)

  Returns a non-owning (plain) pointer to the object pointed to by `p`.

- template<typename T , typename U >
  std::decay< T >::type * as (U *obj)

  Performs polymorphic type conversion.

- template<typename T , typename U >
  const std::decay< T >::type * as (const U *obj)

  Performs polymorphic type conversion.

- template<typename T , typename U >
  std::unique_ptr< typename std::decay< T >::type > as (std::unique_ptr< U > &&obj)

  Performs polymorphic type conversion of a unique_ptr.

- template<typename T , typename U >
  std::shared_ptr< typename std::decay< T >::type > as (std::shared_ptr< U > obj)

  Performs polymorphic type conversion of a shared_ptr.

- template<typename T , typename U >
  std::shared_ptr< const typename std::decay< T >::type > as (std::shared_ptr< const U > obj)

  Performs polymorphic type conversion of a shared_ptr.

- std::ostream & operator<< (std::ostream &os, const version &ver)

  Prints version information to a stream.

- std::ostream & operator<< (std::ostream &os, const version_info &ver_info)

  Prints library version information in human-readable format to a stream.

- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (std::vector< size_type > stride, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of column-vectors.

- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

Creates and initializes a batch of column-vectors.

- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (std::vector< size_type > stride, std::initializer_list< std::initializer_list< std::initializer_list< typename Matrix::value_type >>> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of matrices.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (std::initializer_list< std::initializer_list< std::initializer_list< typename Matrix::value_type >>> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch of matrices.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (std::vector< size_type > stride, const size_type num_vectors, std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a batch column-vector by making copies of the single input column vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (const size_type num_vectors, std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a column-vector from copies of a given vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (std::vector< size_type > stride, const size_type num_matrices, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix from copies of a given matrix.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > batch_initialize (const size_type num_matrices, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix from copies of a given matrix.
- template<typename ValueType >
  std::unique_ptr< matrix::BatchDiagonal< ValueType > > batch_diagonal_initialize (std::initializer_list< std::initializer_list< ValueType >> vals, std::shared_ptr< const Executor > exec)

  Creates and initializes a batch of diagonal matrices.
- template<typename ValueType >
  std::unique_ptr< matrix::BatchDiagonal< ValueType > > batch_diagonal_initialize (const size_type num_matrices, std::initializer_list< ValueType > vals, std::shared_ptr< const Executor > exec)

  Creates and initializes a batch diagonal matrix by making copies of the single input diagonal matrix.
- template<typename ValueType , typename IndexType >
  std::unique_ptr< matrix::Csr< ValueType, IndexType > > create_block_diagonal_matrix (std::shared_ptr< const Executor > exec, const std::vector< std::unique_ptr< matrix::Csr< ValueType, IndexType >>> &matrices)

  Generates a single large block-diagonal CSR matrix from the given CSR matrices.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > initialize (size_type stride, std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a column-vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > initialize (std::initializer_list< typename Matrix::value_type > vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a column-vector.
- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > initialize (size_type stride, std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix.

- template<typename Matrix , typename... TArgs>
  std::unique_ptr< Matrix > initialize (std::initializer_list< std::initializer_list< typename Matrix::value_type >> vals, std::shared_ptr< const Executor > exec, TArgs &&... create_args)

  Creates and initializes a matrix.
- template<typename ValueType , typename IndexType >
  std::unique_ptr< matrix::Ell< ValueType, IndexType > > create_block_diagonal_matrix (std::shared_ptr< const Executor > exec, const std::vector< std::unique_ptr< matrix::Ell< ValueType, IndexType >>> &matrices)

  Generates a single large block-diagonal ELL matrix from the given ELL matrices.
- bool operator== (const stopping_status &x, const stopping_status &y) noexcept

  Checks if two stopping statuses are equivalent.
- bool operator!= (const stopping_status &x, const stopping_status &y) noexcept

  Checks if two stopping statuses are different.

Variables

- constexpr size_type byte_size = CHAR_BIT

  Number of bits in a byte.

40.1.1 Detailed Description

The Ginkgo namespace.

40.1.2 Typedef Documentation

40.1.2.1 highest_precision

```
template<typename... Ts>
using gko::highest_precision = typedef typename detail::highest_precision_variadic<Ts...>↩
::type
```

Obtains the smallest arithmetic type that is able to store elements of all template parameter types exactly.

All template type parameters need to be either real or complex types, mixing them is not possible.

Formally, it computes a right-fold over the type list, with the highest precision of a pair of real arithmetic types T1, T2 computed as `decltype(T1{} + T2{})`, or `std::complex<highest_precision<remove_complex<T1>, remove_complex<T2>>>` for complex types.

40.1.2.2 is_complex_or_scalar_s

```
template<typename T >
using gko::is_complex_or_scalar_s = typedef detail::is_complex_or_scalar_impl<T>
```

Allows to check if T is a complex or scalar value during compile time by accessing the `value` attribute of this struct.

If `value` is `true`, T is a complex/scalar type, if it is `false`, T is not a complex/scalar type.

Template Parameters

T	type to check

40.1.2.3 is_complex_s

```
template<typename T >
using gko::is_complex_s = typedef detail::is_complex_impl<T>
```

Allows to check if T is a complex value during compile time by accessing the `value` attribute of this struct.

If `value` is `true`, T is a complex type, if it is `false`, T is not a complex type.

Template Parameters

T	type to check

40.1.2.4 remove_complex

```
template<typename T >
using gko::remove_complex = typedef typename detail::remove_complex_s<T>::type
```

Obtain the type which removed the complex of complex/scalar type or the template parameter of class by accessing the `type` attribute of this struct.

Template Parameters

T	type to remove complex

Note

remove_complex<class> can not be used in friend class declaration.

40.1.2.5 to_complex

```
template<typename T >
using gko::to_complex = typedef typename detail::to_complex_s<T>::type
```

Obtain the type which adds the complex of complex/scalar type or the template parameter of class by accessing the `type` attribute of this struct.

Template Parameters

T	type to complex_type

Note

to_complex<class> can not be used in friend class declaration. the followings are the error message from different combination. friend to_complex<Csr>; error: can not recognize it is class correctly. friend class to_complex<Csr>; error: using alias template specialization friend class to_complex_s<Csr<ValueType,↩ IndexType>>::type; error: can not recognize it is class correctly.

40.1.2.6 to_real

```
template<typename T >
using gko::to_real = typedef remove_complex<T>
```

to_real is alias of remove_complex

Template Parameters

T	type to real

40.1.3 Enumeration Type Documentation

40.1.3.1 allocation_mode

```
enum gko::allocation_mode  [strong]
```

Specify the mode of allocation for CUDA/HIP GPUs.

`device` allocates memory on the device and Unified Memory model is not used.

`unified_global` allocates memory on the device, but is accessible by the host through the Unified memory model.

`unified_host` allocates memory on the host and it is not available on devices which do not have concurrent acesses switched on, but this access can be explictly switched on, when necessary.

```
70 { device, unified_global, unified_host };
```

40.1.3.2 layout_type

```
enum gko::layout_type  [strong]
```

Specifies the layout type when writing data in matrix market format.

Enumerator

array	The matrix should be written as dense matrix in column-major order.
coordinate	The matrix should be written as a sparse matrix in coordinate format.

```
                    {
    array,
    coordinate
};
```

40.1.4 Function Documentation

40.1.4.1 abs()

```
template<typename T >
constexpr xstd::enable_if_t<!is_complex_s<T>::value, T> gko::abs (
            const T & x )  [inline], [constexpr]
```

Returns the absolute value of the object.

Template Parameters

T	the type of the object

Parameters

x	the object

Returns

x >= zero<T>() ? x : -x;

```
{
    return x >= zero<T>() ? x : -x;
}
```

Referenced by is_finite().

40.1.4.2 as() [1/5]

```
template<typename T , typename U >
const std::decay<T>::type* gko::as (
            const U * obj )  [inline]
```

Performs polymorphic type conversion.

This is the constant version of the function.

Template Parameters

T	requested result type
U	static type of the passed object

Parameters

obj	the object which should be converted

Returns

If successful, returns a pointer to the subtype, otherwise throws NotSupported.

```
{
    if (auto p = dynamic_cast<const typename std::decay<T>::type*>(obj)) {
        return p;
    } else {
        throw NotSupported(__FILE__, __LINE__,
                           std::string{"gko::as<"} +
                               name_demangling::get_type_name(typeid(T)) + ">",
                           name_demangling::get_type_name(typeid(*obj)));
    }
}
```

40.1.4.3 as() [2/5]

```
template<typename T , typename U >
std::shared_ptr<const typename std::decay<T>::type> gko::as (
            std::shared_ptr< const U > obj )  [inline]
```

Performs polymorphic type conversion of a shared_ptr.

This is the constant version of the function.

Template Parameters

T	requested result type
U	static type of the passed object

Parameters

obj	the shared_ptr to the object which should be converted.

Returns

If successful, returns a shared_ptr to the subtype, otherwise throws NotSupported. This pointer shares ownership with the input pointer.

40.1.4.4 as() [3/5]

```
template<typename T , typename U >
std::shared_ptr<typename std::decay<T>::type> gko::as (
            std::shared_ptr< U > obj ) [inline]
```

Performs polymorphic type conversion of a shared_ptr.

Template Parameters

T	requested result type
U	static type of the passed object

Parameters

obj	the shared_ptr to the object which should be converted.

Returns

If successful, returns a shared_ptr to the subtype, otherwise throws NotSupported. This pointer shares ownership with the input pointer.

40.1.4.5 as() [4/5]

```
template<typename T , typename U >
std::unique_ptr<typename std::decay<T>::type> gko::as (
            std::unique_ptr< U > && obj ) [inline]
```

Performs polymorphic type conversion of a unique_ptr.

Template Parameters

T	requested result type
U	static type of the passed object

Parameters

obj	the unique_ptr to the object which should be converted. If successful, it will be reset to a nullptr.

Returns

If successful, returns a unique_ptr to the subtype, otherwise throws NotSupported.

40.1.4.6 as() [5/5]

```
template<typename T , typename U >
std::decay<T>::type* gko::as (
            U * obj ) [inline]
```

Performs polymorphic type conversion.

Template Parameters

T	requested result type
U	static type of the passed object

Parameters

obj	the object which should be converted

Returns

If successful, returns a pointer to the subtype, otherwise throws NotSupported.

Referenced by gko::preconditioner::Isai< IsaiType, ValueType, IndexType >::get_approximate_inverse().

40.1.4.7 ceildiv()

```
constexpr int64 gko::ceildiv (
            int64 num,
            int64 den ) [inline], [constexpr]
```

Performs integer division with rounding up.

Parameters

num	numerator
den	denominator

Returns

returns the ceiled quotient.

Referenced by gko::matrix::Csr< ValueType, IndexType >::load_balance::clac_size(), gko::preconditioner::block↩_interleaved_storage_scheme< index_type >::compute_storage_space(), and gko::matrix::Csr< ValueType, IndexType >::load_balance::process().

40.1.4.8 clone() [1/2]

```
template<typename Pointer >
detail::cloned_type<Pointer> gko::clone (
            const Pointer & p ) [inline]
```

Creates a unique clone of the object pointed to by `p`.

The pointee (i.e. `*p`) needs to have a clone method that returns a std::unique_ptr in order for this method to work.

Template Parameters

Pointer	type of pointer to the object (plain or smart pointer)

Parameters

p	a pointer to the object

Note

The difference between this function and directly calling LinOp::clone() is that this one preserves the static type of the object.

40.1.4.9 clone() [2/2]

```
template<typename Pointer >
detail::cloned_type<Pointer> gko::clone (
            std::shared_ptr< const Executor > exec,
            const Pointer & p ) [inline]
```

Creates a unique clone of the object pointed to by `p` on Executor `exec`.

The pointee (i.e. `*p`) needs to have a clone method that takes an executor and returns a std::unique_ptr in order for this method to work.

Template Parameters

Pointer	type of pointer to the object (plain or smart pointer)

Parameters

exec	the executor where the cloned object should be stored
p	a pointer to the object

Note

The difference between this function and directly calling LinOp::clone() is that this one preserves the static type of the object.

40.1.4.10 conj()

```
template<typename T >
constexpr auto gko::conj (
            const T & x )  [inline], [constexpr]
```

Returns the conjugate of an object.

Parameters

x	the number to conjugate

Returns

conjugate of the object (by default, the object itself)

Referenced by squared_norm().

40.1.4.11 copy_and_convert_to() [1/4]

```
template<typename R , typename T >
std::unique_ptr<const R, std::function<void(const R*)> > gko::copy_and_convert_to (
            std::shared_ptr< const Executor > exec,
            const T * obj )
```

Converts the object to R and places it on Executor exec.

If the object is already of the requested type and on the requested executor, the copy and conversion is avoided and a reference to the original object is returned instead.

Template Parameters

R	the type to which the object should be converted
T	the type of the input object

Parameters

exec	the executor where the result should be placed
obj	the object that should be converted

Returns

a unique pointer (with dynamically bound deleter) to the converted object

Note

This is a version of the function which adds the const qualifier to the result if the input had the same qualifier.

```
{
    return detail::copy_and_convert_to_impl<const R>(std::move(exec), obj);
}
```

40.1.4.12 copy_and_convert_to() [2/4]

```
template<typename R , typename T >
std::shared_ptr<const R> gko::copy_and_convert_to (
            std::shared_ptr< const Executor > exec,
            std::shared_ptr< const T > obj )
```

This is the version that takes in the std::shared_ptr and returns a std::shared_ptr

If the object is already of the requested type and on the requested executor, the copy and conversion is avoided and a reference to the original object is returned instead.

Template Parameters

R	the type to which the object should be converted
T	the type of the input object

Parameters

exec	the executor where the result should be placed
obj	the object that should be converted

Returns

a shared pointer to the converted object

Note

This is a version of the function which adds the const qualifier to the result if the input had the same qualifier.

40.1.4.13 copy_and_convert_to() [3/4]

```
template<typename R , typename T >
std::shared_ptr<R> gko::copy_and_convert_to (
            std::shared_ptr< const Executor > exec,
            std::shared_ptr< T > obj )
```

Converts the object to R and places it on Executor exec.

This is the version that takes in the std::shared_ptr and returns a std::shared_ptr

If the object is already of the requested type and on the requested executor, the copy and conversion is avoided and a reference to the original object is returned instead.

Template Parameters

R	the type to which the object should be converted
T	the type of the input object

Parameters

exec	the executor where the result should be placed
obj	the object that should be converted

Returns

a shared pointer to the converted object

40.1.4.14 copy_and_convert_to() [4/4]

```
template<typename R , typename T >
std::unique_ptr<R, std::function<void(R*)> > gko::copy_and_convert_to (
            std::shared_ptr< const Executor > exec,
            T * obj )
```

Converts the object to R and places it on Executor exec.

If the object is already of the requested type and on the requested executor, the copy and conversion is avoided and a reference to the original object is returned instead.

Template Parameters

R	the type to which the object should be converted
T	the type of the input object

Parameters

exec	the executor where the result should be placed
obj	the object that should be converted

Returns

a unique pointer (with dynamically bound deleter) to the converted object

40.1.4.15 create_block_diagonal_matrix() [1/2]

```
template<typename ValueType , typename IndexType >
std::unique_ptr<matrix::Csr<ValueType, IndexType> > gko::create_block_diagonal_matrix (
```

```
            std::shared_ptr< const Executor > exec,
            const std::vector< std::unique_ptr< matrix::Csr< ValueType, IndexType >>> &
matrices )
```

Generates a single large block-diagonal CSR matrix from the given CSR matrices.

Parameters

exec	Executor on which both the input and output reside
matrices	List of matrices to be concatenated in to one block-diagonal matrix.

Warning

Not for use in performance-critical code! The operation currently happens on the host and copies are performed.

40.1.4.16 create_block_diagonal_matrix() [2/2]

```
template<typename ValueType , typename IndexType >
std::unique_ptr<matrix::Ell<ValueType, IndexType> > gko::create_block_diagonal_matrix (
            std::shared_ptr< const Executor > exec,
            const std::vector< std::unique_ptr< matrix::Ell< ValueType, IndexType >>> &
matrices )
```

Generates a single large block-diagonal ELL matrix from the given ELL matrices.

Parameters

exec	Executor on which both the input and output reside
matrices	List of matrices to be concatenated in to one block-diagonal matrix.

Warning

Not for use in performance-critical code! The operation currently happens on the host and copies are performed.

40.1.4.17 get_significant_bit()

```
template<typename T >
constexpr uint32 gko::get_significant_bit (
            const T & n,
            uint32 hint = 0u ) [constexpr], [noexcept]
```

Returns the position of the most significant bit of the number.

This is the same as the rounded down base-2 logarithm of the number.

Template Parameters

T	a numeric type supporting bit shift and comparison

Parameters

n	a number
hint	a lower bound for the position o the significant bit

Returns

maximum of `hint` and the significant bit position of `n`

40.1.4.18 get_superior_power()

```
template<typename T >
constexpr T gko::get_superior_power (
            const T & base,
            const T & limit,
            const T & hint = T{1} )  [constexpr], [noexcept]
```

Returns the smallest power of `base` not smaller than `limit`.

Template Parameters

T	a numeric type supporting multiplication and comparison

Parameters

base	the base of the power to be returned
limit	the lower limit on the size of the power returned
hint	a lower bound on the result, has to be a power of base

Returns

the smallest power of `base` not smaller than `limit`

40.1.4.19 give()

```
template<typename OwningPointer >
std::remove_reference<OwningPointer>::type&& gko::give (
            OwningPointer && p )  [inline]
```

Marks that the object pointed to by `p` can be given to the callee.

Effectively calls `std::move(p)`.

Template Parameters

OwningPointer	type of pointer with ownership to the object (has to be a smart pointer)

Parameters

p	a pointer to the object

Note

The original pointer `p` becomes invalid after this call.

40.1.4.20 imag()

```
template<typename T >
constexpr auto gko::imag (
            const T & x )  [inline], [constexpr]
```

Returns the imaginary part of the object.

Template Parameters

T	type of the object

Parameters

x	the object

Returns

imaginary part of the object (by default, zero<T>())

40.1.4.21 is_complex()

```
template<typename T >
constexpr bool gko::is_complex ( )  [inline], [constexpr]
```

Checks if T is a complex type.

Template Parameters

T	type to check

Returns

`true` if T is a complex type, `false` otherwise

40.1.4.22 is_complex_or_scalar()

```
template<typename T >
constexpr bool gko::is_complex_or_scalar ( )  [inline], [constexpr]
```

Checks if T is a complex/scalar type.

Template Parameters

T	type to check

Returns

`true` if T is a complex/scalar type, `false` otherwise

40.1.4.23 is_finite() [1/2]

```
template<typename T >
std::enable_if_t<!is_complex_s<T>::value, bool> gko::is_finite (
            const T & value )  [inline]
```

Checks if a floating point number is finite, meaning it is neither +/- infinity nor NaN.

Template Parameters

T	type of the value to check

Parameters

value	value to check

Returns

`true` if the value is finite, meaning it are neither +/- infinity nor NaN.

References abs().

Referenced by is_finite().

40.1.4.24 is_finite() [2/2]

```
template<typename T >
std::enable_if_t<is_complex_s<T>::value, bool> gko::is_finite (
            const T & value )  [inline]
```

Checks if all components of a complex value are finite, meaning they are neither +/- infinity nor NaN.

Template Parameters

T	complex type of the value to check

Parameters

value	complex value to check

Returns

`true` if both components of the given value are finite, meaning they are neither +/- infinity nor NaN.

References is_finite().

40.1.4.25 is_nan() [1/2]

```
template<typename T >
std::enable_if_t<!is_complex_s<T>::value, bool> gko::is_nan (
            const T & value )  [inline]
```

Checks if a floating point number is NaN.

Template Parameters

T	type of the value to check

Parameters

value	value to check

Returns

`true` if the value is NaN.

40.1.4.26 is_nan() [2/2]

```
template<typename T >
std::enable_if_t<is_complex_s<T>::value, bool> gko::is_nan (
            const T & value )  [inline]
```

Checks if any component of a complex value is NaN.

Template Parameters

T	complex type of the value to check

Parameters

value	complex value to check

Returns

`true` if any component of the given value is NaN.

40.1.4.27 is_nonzero()

```
template<typename T >
constexpr bool gko::is_nonzero (
            T value )  [inline], [constexpr]
```

Returns true if and only if the given value is not zero.

Template Parameters

T	the type of the value

Parameters

value	the given value

Returns

true iff the given value is not zero, i.e. `value != zero<T>()`

Referenced by gko::matrix_data< ValueType, IndexType >::diag().

40.1.4.28 is_zero()

```
template<typename T >
constexpr bool gko::is_zero (
            T value ) [inline], [constexpr]
```

Returns true if and only if the given value is zero.

Template Parameters

T	the type of the value

Parameters

value	the given value

Returns

true iff the given value is zero, i.e. `value == zero<T>()`

Referenced by gko::matrix_data< ValueType, IndexType >::remove_zeros().

40.1.4.29 lend() [1/2]

```
template<typename Pointer >
std::enable_if<detail::have_ownership_s<Pointer>::value, detail::pointee<Pointer>*>::type
gko::lend (
            const Pointer & p ) [inline]
```

Returns a non-owning (plain) pointer to the object pointed to by `p`.

Template Parameters

Pointer	type of pointer to the object (plain or smart pointer)

Parameters

p	a pointer to the object

Note

This is the overload for owning (smart) pointers, that behaves the same as calling .get() on the smart pointer.

40.1.4.30 lend() [2/2]

```
template<typename Pointer >
std::enable_if<!detail::have_ownership_s<Pointer>::value, detail::pointee<Pointer>*>::type
gko::lend (
            const Pointer & p )  [inline]
```

Returns a non-owning (plain) pointer to the object pointed to by `p`.

Template Parameters

Pointer	type of pointer to the object (plain or smart pointer)

Parameters

p	a pointer to the object

Note

This is the overload for non-owning (plain) pointers, that just returns `p`.

40.1.4.31 make_array_view()

```
template<typename ValueType >
Array<ValueType> gko::make_array_view (
            std::shared_ptr< const Executor > exec,
            size_type size,
            ValueType * data )
```

Helper function to create an array view deducing the value type.

Parameters

exec	the executor on which the array resides
size	the number of elements for the array
data	the pointer to the array we create a view on.

Template Parameters

ValueType	the type of the array elements

Returns

`Array<ValueType>::view(exec, size, data)`

```
753 {
754     return Array<ValueType>::view(exec, size, data);
755 }
```

References gko::Array< ValueType >::view().

40.1.4.32 make_temporary_clone()

```
template<typename T >
detail::temporary_clone<T> gko::make_temporary_clone (
            std::shared_ptr< const Executor > exec,
            T * ptr )
```

Creates a temporary_clone.

This is a helper function which avoids the need to explicitly specify the type of the object, as would be the case if using the constructor of temporary_clone.

Parameters

exec	the executor where the clone will be created
ptr	a pointer to the object of which the clone will be created

```
{
    return detail::temporary_clone<T>(std::move(exec), ptr);
}
```

Referenced by gko::matrix::BatchDense< ValueType >::add_scaled(), gko::matrix::Dense< ValueType >::add_scaled(), gko::ScaledIdentityAddable::add_scaled_identity(), gko::BatchScaledIdentityAddable::add_scaled_identity(), gko::matrix::Coo< ValueType, IndexType >::apply2(), gko::matrix::Dense< ValueType >::compute_conj_dot(), gko::matrix::BatchDense< ValueType >::compute_dot(), gko::matrix::Dense< ValueType >::compute_dot(), gko::matrix::BatchDense< ValueType >::compute_norm2(), gko::matrix::Dense< ValueType >::inv_scale(), gko::matrix::Csr< ValueType, IndexType >::inv_scale(), gko::matrix::BatchDense< ValueType >::scale(), gko::matrix::Dense< ValueType >::scale(), gko::matrix::Csr< ValueType, IndexType >::scale(), and gko::matrix::Dense< ValueType >::sub_scaled().

40.1.4.33 make_temporary_conversion() [1/2]

```
template<typename ValueType >
detail::temporary_conversion<const matrix::Dense<ValueType> > gko::make_temporary_conversion
(
            const LinOp * matrix )
```

Convert the given LinOp from matrix::Dense<...> to matrix::Dense<ValueType>.

The conversion tries to convert the input LinOp to all Dense types with value type recursively reachable by next_precision<...> starting from the ValueType template parameter. This means that all real-to-real and complex-to-complex conversions for default precisions are being considered. If the input matrix is non-const, the contents of the modified converted object will be converted back to the input matrix when the returned object is destroyed. This may lead to a loss of precision!

Parameters

matrix	the input matrix which is supposed to be converted. It is wrapped unchanged if it is already of type matrix::Dense<ValueType>, otherwise it will be converted to this type if possible.

Returns

a detail::temporary_conversion pointing to the (potentially converted) object.

Exceptions

NotSupported	if the input matrix cannot be converted to matrix::Dense<ValueType>

Template Parameters

ValueType	the value type into whose associated matrix::Dense type to convert the input LinOp.

```
{
    auto result = detail::temporary_conversion<const matrix::Dense<ValueType>>::
        template create<matrix::Dense<next_precision<ValueType>>>(matrix);
    if (!result) {
        GKO_NOT_SUPPORTED(matrix);
    }
    return result;
}
```

40.1.4.34 make_temporary_conversion() [2/2]

```
template<typename ValueType >
detail::temporary_conversion<matrix::Dense<ValueType> > gko::make_temporary_conversion (
            LinOp * matrix )
```

Convert the given LinOp from matrix::Dense<...> to matrix::Dense<ValueType>.

The conversion tries to convert the input LinOp to all Dense types with value type recursively reachable by next_precision<...> starting from the ValueType template parameter. This means that all real-to-real and complex-to-complex conversions for default precisions are being considered. If the input matrix is non-const, the contents of the modified converted object will be converted back to the input matrix when the returned object is destroyed. This may lead to a loss of precision!

Parameters

matrix	the input matrix which is supposed to be converted. It is wrapped unchanged if it is already of type matrix::Dense<ValueType>, otherwise it will be converted to this type if possible.

Returns

a detail::temporary_conversion pointing to the (potentially converted) object.

Exceptions

NotSupported	if the input matrix cannot be converted to matrix::Dense<ValueType>

Template Parameters

ValueType	the value type into whose associated matrix::Dense type to convert the input LinOp.

40.1.4.35 make_temporary_output_clone()

```
template<typename T >
detail::temporary_clone<T> gko::make_temporary_output_clone (
            std::shared_ptr< const Executor > exec,
            T * ptr )
```

Creates a uninitialized temporary_clone that will be copied back to the input afterwards.

It can be used for output parameters to avoid an unnecessary copy in make_temporary_clone.

This is a helper function which avoids the need to explicitly specify the type of the object, as would be the case if using the constructor of temporary_clone.

Parameters

exec	the executor where the uninitialized clone will be created
ptr	a pointer to the object of which the clone will be created

Referenced by gko::matrix::Dense< ValueType >::compute_conj_dot(), gko::matrix::Dense< ValueType >::compute_dot(), gko::matrix::Dense< ValueType >::compute_norm1(), and gko::matrix::Dense< ValueType >::compute_norm2().

40.1.4.36 max()

```
template<typename T >
constexpr T gko::max (
            const T & x,
            const T & y ) [inline], [constexpr]
```

Returns the larger of the arguments.

Template Parameters

T	type of the arguments

Parameters

x	first argument
y	second argument

Returns

x >= y ? x : y

40.1.4.37 min()

```
template<typename T >
constexpr T gko::min (
            const T & x,
            const T & y ) [inline], [constexpr]
```

Returns the smaller of the arguments.

Template Parameters

T	type of the arguments

Parameters

x	first argument
y	second argument

Returns

x <= y ? x : y

Referenced by gko::matrix::Csr< ValueType, IndexType >::load_balance::clac_size().

40.1.4.38 mixed_precision_dispatch()

```
template<typename ValueType , typename Function >
void gko::mixed_precision_dispatch (
            Function fn,
            const LinOp * in,
            LinOp * out )
```

Calls the given function with each given argument LinOp converted into matrix::Dense<ValueType> as parameters.

If GINKGO_MIXED_PRECISION is defined, this means that the function will be called with its dynamic type as a static type, so the (templated/generic) function will be instantiated with all pairs of Dense<ValueType> and Dense<next_precision<ValueType>> parameter types, and the appropriate overload will be called based on the dynamic type of the parameter.

If GINKGO_MIXED_PRECISION is not defined, it will behave exactly like precision_dispatch.

Parameters

fn	the given function. It will be called with one const and one non-const matrix::Dense<...> parameter based on the dynamic type of the inputs (GINKGO_MIXED_PRECISION) or of type matrix::Dense<ValueType> (no GINKGO_MIXED_PRECISION).
in	The first parameter to be cast (GINKGO_MIXED_PRECISION) or converted (no GINKGO_MIXED_PRECISION) and used to call `fn`.
out	The second parameter to be cast (GINKGO_MIXED_PRECISION) or converted (no GINKGO_MIXED_PRECISION) and used to call `fn`.

Template Parameters

ValueType	the value type to use for the parameters of `fn` (no GINKGO_MIXED_PRECISION). With GINKGO_MIXED_PRECISION enabled, it only matters whether this type is complex or real.
Function	the function pointer, lambda or other functor type to call with the converted arguments.

40.1.4.39 mixed_precision_dispatch_real_complex()

```
template<typename ValueType , typename Function , std::enable_if_t< is_complex< ValueType
>()> * = nullptr>
void gko::mixed_precision_dispatch_real_complex (
            Function fn,
            const LinOp * in,
            LinOp * out )
```

Calls the given function with the given LinOps cast to their dynamic type matrix::Dense<ValueType>∗ as parameters.

If ValueType is real and both `in` and `out` are complex, uses matrix::Dense::get_real_view() to convert them into real matrices after precision conversion.

See also

mixed_precision_dispatch()

40.1.4.40 nan() [1/2]

```
template<typename T >
constexpr std::enable_if_t<!is_complex_s<T>::value, T> gko::nan ( )  [inline], [constexpr]
```

Returns a quiet NaN of the given type.

Template Parameters

T	the type of the object

Returns

NaN.

Referenced by nan().

40.1.4.41 nan() [2/2]

```
template<typename T >
constexpr std::enable_if_t<is_complex_s<T>::value, T> gko::nan ( )  [inline], [constexpr]
```

Returns a complex with both components quiet NaN.

Template Parameters

T	the type of the object

Returns

complex{NaN, NaN}.

References nan().

40.1.4.42 one() [1/2]

```
template<typename T >
constexpr T gko::one ( )  [inline], [constexpr]
```

Returns the multiplicative identity for T.

Returns

the multiplicative identity for T

Referenced by unit_root().

40.1.4.43 one() [2/2]

```
template<typename T >
constexpr T gko::one (
             const T & )  [inline], [constexpr]
```

Returns the multiplicative identity for T.

Returns

the multiplicative identity for T

Note

This version takes an unused reference argument to avoid complicated calls like `one<decltype(x)>()`. Instead, it allows `one(x)`.

40.1.4.44 operator"!=() [1/4]

```
template<size_type Dimensionality, typename DimensionType >
bool gko::operator!= (
             const batch_dim< Dimensionality, DimensionType > & x,
             const batch_dim< Dimensionality, DimensionType > & y )  [inline]
```

Checks if two batch dim objects are different.

Template Parameters

Dimensionality	number of dimensions of the dim objects
DimensionType	datatype used to represent each dimension

Parameters

x	first object
y	second object

Returns

`!(x == y)`

```
{
    return !(x == y);
}
```

40.1.4.45 operator"!=() [2/4]

```
template<size_type Dimensionality, typename DimensionType >
constexpr bool gko::operator!= (
            const dim< Dimensionality, DimensionType > & x,
            const dim< Dimensionality, DimensionType > & y )  [inline], [constexpr]
```

Checks if two dim objects are different.

Template Parameters

Dimensionality	number of dimensions of the dim objects
DimensionType	datatype used to represent each dimension

Parameters

x	first object
y	second object

Returns

`!(x == y)`

40.1.4.46 operator"!=() [3/4]

```
bool gko::operator!= (
            const stopping_status & x,
            const stopping_status & y )  [inline], [noexcept]
```

Checks if two stopping statuses are different.

Parameters

x	a stopping status
y	a stopping status

Returns

true if and only if `!(x == y)`

```
{
    return x.data_ != y.data_;
}
```

40.1.4.47 operator"!=() [4/4]

```
constexpr bool gko::operator!= (
            precision_reduction x,
            precision_reduction y )  [constexpr], [noexcept]
```

Checks if two precision_reduction encodings are different.

Parameters

x	an encoding
y	an encoding

Returns

true if and only if `x` and `y` are different encodings.

```
{
    using st = precision_reduction::storage_type;
    return static_cast<st>(x) != static_cast<st>(y);
}
```

40.1.4.48 operator<<() [1/2]

```
std::ostream& gko::operator<< (
            std::ostream & os,
            const version & ver )  [inline]
```

Prints version information to a stream.

Parameters

os	output stream
ver	version structure

Returns

os

```
{
    os « ver.major « "." « ver.minor « "." « ver.patch;
    if (ver.tag) {
        os « " (" « ver.tag « ")";
    }
    return os;
}
```

References gko::version::major, gko::version::minor, gko::version::patch, and gko::version::tag.

40.1.4.49 operator<<() [2/2]

```
std::ostream& gko::operator<< (
        std::ostream & os,
        const version_info & ver_info )
```

Prints library version information in human-readable format to a stream.

Parameters

os	output stream
ver_info	version information

Returns

os

40.1.4.50 operator==() [1/2]

```
bool gko::operator== (
        const stopping_status & x,
        const stopping_status & y ) [inline], [noexcept]
```

Checks if two stopping statuses are equivalent.

Parameters

x	a stopping status
y	a stopping status

Returns

true if and only if both `x` and `y` have the same mask and converged and finalized state

40.1.4.51 operator==() [2/2]

```
constexpr bool gko::operator== (
            precision_reduction x,
            precision_reduction y )  [constexpr], [noexcept]
```

Checks if two precision_reduction encodings are equal.

Parameters

x	an encoding
y	an encoding

Returns

true if and only if `x` and `y` are the same encodings

40.1.4.52 pi()

```
template<typename T >
constexpr T gko::pi ( )  [inline], [constexpr]
```

Returns the value of pi.

Template Parameters

T	the value type to return

40.1.4.53 precision_dispatch()

```
template<typename ValueType , typename Function , typename... Args>
void gko::precision_dispatch (
            Function fn,
            Args *... linops )
```

Calls the given function with each given argument LinOp temporarily converted into matrix::Dense<ValueType> as parameters.

Parameters

fn	the given function. It will be passed one (potentially const) matrix::Dense<ValueType>∗ parameter per parameter in the parameter pack `linops`.
linops	the given arguments to be converted and passed on to fn.

Template Parameters

ValueType	the value type to use for the parameters of `fn`.
Function	the function pointer, lambda or other functor type to call with the converted arguments.
Args	the argument type list.

40.1.4.54 precision_dispatch_real_complex() [1/3]

```
template<typename ValueType , typename Function >
void gko::precision_dispatch_real_complex (
            Function fn,
            const LinOp * alpha,
            const LinOp * in,
            const LinOp * beta,
            LinOp * out )
```

Calls the given function with the given LinOps temporarily converted to matrix::Dense<ValueType>* as parameters.

If ValueType is real and both `in` and `out` are complex, uses matrix::Dense::get_real_view() to convert them into real matrices after precision conversion.

See also

precision_dispatch()

40.1.4.55 precision_dispatch_real_complex() [2/3]

```
template<typename ValueType , typename Function >
void gko::precision_dispatch_real_complex (
            Function fn,
            const LinOp * alpha,
            const LinOp * in,
            LinOp * out )
```

Calls the given function with the given LinOps temporarily converted to matrix::Dense<ValueType>* as parameters.

If ValueType is real and both `in` and `out` are complex, uses matrix::Dense::get_real_view() to convert them into real matrices after precision conversion.

See also

precision_dispatch()

40.1.4.56 precision_dispatch_real_complex() [3/3]

```
template<typename ValueType , typename Function >
void gko::precision_dispatch_real_complex (
            Function fn,
            const LinOp * in,
            LinOp * out )
```

Calls the given function with the given LinOps temporarily converted to matrix::Dense<ValueType>* as parameters.

If ValueType is real and both input vectors are complex, uses matrix::Dense::get_real_view() to convert them into real matrices after precision conversion.

See also

precision_dispatch()

40.1.4.57 read()

```
template<typename MatrixType , typename StreamType , typename... MatrixArgs>
std::unique_ptr<MatrixType> gko::read (
            StreamType && is,
            MatrixArgs &&... args ) [inline]
```

Reads a matrix stored in matrix market format from an input stream.

Template Parameters

MatrixType	a ReadableFromMatrixData LinOp type used to store the matrix once it's been read from disk.
StreamType	type of stream used to write the data to
MatrixArgs	additional argument types passed to MatrixType constructor

Parameters

is	input stream from which to read the data
args	additional arguments passed to MatrixType constructor

Returns

A MatrixType LinOp filled with data from filename

References read_raw().

Referenced by gko::ReadableFromMatrixData< ValueType, int32 >::read(), and gko::BatchReadableFromMatrix↩
Data< ValueType, int32 >::read().

40.1.4.58 read_binary()

```
template<typename MatrixType , typename StreamType , typename...  MatrixArgs>
std::unique_ptr<MatrixType> gko::read_binary (
            StreamType && is,
            MatrixArgs &&...  args )  [inline]
```

Reads a matrix stored in binary format from an input stream.

Template Parameters

MatrixType	a ReadableFromMatrixData LinOp type used to store the matrix once it's been read from disk.
StreamType	type of stream used to write the data to
MatrixArgs	additional argument types passed to MatrixType constructor

Parameters

is	input stream from which to read the data
args	additional arguments passed to MatrixType constructor

Returns

A MatrixType LinOp filled with data from filename

References read_binary_raw().

40.1.4.59 read_binary_raw()

```
template<typename ValueType = default_precision, typename IndexType = int32>
matrix_data<ValueType, IndexType> gko::read_binary_raw (
            std::istream & is )
```

Reads a matrix stored in Ginkgo's binary matrix format from an input stream.

Note that this format depends on the processor's endianness, so files from a big endian processor can't be read from a little endian processor and vice-versa.

The binary format has the following structure (in system endianness):

1. A 32 byte header consisting of 4 uint64_t values: magic = GINKGO__: The highest two bytes stand for value and index type. value type: S (float), D (double), C (complex<float>), Z(complex<double>) index type: I (int32), L (int64) num_rows: Number of rows num_cols: Number of columns num_entries: Number of (row, column, value) tuples to follow
2. Following are num_entries blocks of size sizeof(IndexType) $*$ 2 + sizeof(ValueType). Each consists of a row index stored as IndexType, followed by a column index stored as IndexType and a value stored as ValueType.

Template Parameters

ValueType	type of matrix values
IndexType	type of matrix indexes

Parameters

is	input stream from which to read the data

Returns

A matrix_data structure containing the matrix. The nonzero elements are sorted in lexicographic order of their (row, column) indexes.

Note

This is an advanced routine that will return the raw matrix data structure. Consider using gko::read_binary instead.

Referenced by read_binary().

40.1.4.60 read_generic()

```
template<typename MatrixType , typename StreamType , typename...  MatrixArgs>
std::unique_ptr<MatrixType> gko::read_generic (
            StreamType && is,
            MatrixArgs &&...  args )  [inline]
```

Reads a matrix stored either in binary or matrix market format from an input stream.

Template Parameters

MatrixType	a ReadableFromMatrixData LinOp type used to store the matrix once it's been read from disk.
StreamType	type of stream used to write the data to
MatrixArgs	additional argument types passed to MatrixType constructor

Parameters

is	input stream from which to read the data
args	additional arguments passed to MatrixType constructor

Returns

A MatrixType LinOp filled with data from filename

References read_generic_raw().

40.1.4.61 read_generic_raw()

```
template<typename ValueType = default_precision, typename IndexType = int32>
matrix_data<ValueType, IndexType> gko::read_generic_raw (
            std::istream & is )
```

Reads a matrix stored in either binary or matrix market format from an input stream.

Template Parameters

ValueType	type of matrix values
IndexType	type of matrix indexes

Parameters

is	input stream from which to read the data

Returns

A matrix_data structure containing the matrix. The nonzero elements are sorted in lexicographic order of their (row, column) indexes.

Note

This is an advanced routine that will return the raw matrix data structure. Consider using gko::read_generic instead.

Referenced by read_generic().

40.1.4.62 read_raw()

```
template<typename ValueType = default_precision, typename IndexType = int32>
matrix_data<ValueType, IndexType> gko::read_raw (
            std::istream & is )
```

Reads a matrix stored in matrix market format from an input stream.

Template Parameters

ValueType	type of matrix values
IndexType	type of matrix indexes

Parameters

is	input stream from which to read the data

Returns

A matrix_data structure containing the matrix. The nonzero elements are sorted in lexicographic order of their (row, column) indexes.

Note

This is an advanced routine that will return the raw matrix data structure. Consider using gko::read instead.

Referenced by read().

40.1.4.63 real()

```
template<typename T >
constexpr auto gko::real (
            const T & x )  [inline], [constexpr]
```

Returns the real part of the object.

Template Parameters

T	type of the object

Parameters

x	the object

Returns

real part of the object (by default, the object itself)

Referenced by squared_norm().

40.1.4.64 reduce_add() [1/2]

```
template<typename ValueType >
void gko::reduce_add (
            const Array< ValueType > & input_arr,
            Array< ValueType > & result )
```

Reduce (sum) the values in the array.

Template Parameters

The	type of the input data

Parameters

in	*input_arr*	the input array to be reduced
in,out	*result*	the reduced value. The result is written into the first entry and the value in the first entry is used as the initial value for the reduce.

40.1.4.65 reduce_add() [2/2]

```
template<typename ValueType >
ValueType gko::reduce_add (
            const Array< ValueType > & input_arr,
            const ValueType init_val = 0 )
```

Reduce (sum) the values in the array.

Template Parameters

The	type of the input data

Parameters

in	*input_arr*	the input array to be reduced
in	*init_val*	the initial value

Returns

the reduced value

40.1.4.66 round_down()

```
template<typename T >
constexpr reduce_precision<T> gko::round_down (
            T val ) [inline], [constexpr]
```

Reduces the precision of the input parameter.

Template Parameters

T	the original precision

Parameters

val	the value to round down

Returns

the rounded down value

40.1.4.67 round_up()

```
template<typename T >
constexpr increase_precision<T> gko::round_up (
            T val ) [inline], [constexpr]
```

Increases the precision of the input parameter.

Template Parameters

T	the original precision

Parameters

val	the value to round up

Returns

the rounded up value

40.1.4.68 safe_divide()

```
template<typename T >
T gko::safe_divide (
            T a,
            T b ) [inline]
```

Computes the quotient of the given parameters, guarding against division by zero.

Template Parameters

T	value type of the parameters

Parameters

a	the dividend
b	the divisor

Returns

the value of `a / b` if b is non-zero, zero otherwise.

40.1.4.69 share()

```
template<typename OwningPointer >
detail::shared_type<OwningPointer> gko::share (
            OwningPointer && p )  [inline]
```

Marks the object pointed to by `p` as shared.

Effectively converts a pointer with ownership to std::shared_ptr.

Template Parameters

OwningPointer	type of pointer with ownership to the object (has to be a smart pointer)

Parameters

p	a pointer to the object

Note

The original pointer `p` becomes invalid after this call.

Referenced by gko::preconditioner::Ic< LSolverType, IndexType >::conj_transpose(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::conj_transpose(), gko::preconditioner::Ic< LSolverType, IndexType >::transpose(), and gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::transpose().

40.1.4.70 squared_norm()

```
template<typename T >
constexpr auto gko::squared_norm (
            const T & x ) -> decltype(real(conj(x) * x))   [inline], [constexpr]
```

Returns the squared norm of the object.

Template Parameters

T	type of the object.

Returns

The squared norm of the object.

References conj(), and real().

40.1.4.71 transpose() [1/2]

```
template<typename DimensionType >
batch_dim<2, DimensionType> gko::transpose (
            const batch_dim< 2, DimensionType > & input )  [inline]
```

Returns a batch_dim object with its dimensions swapped for batched operators.

Template Parameters

DimensionType	datatype used to represent each dimension

Parameters

dimensions	original object

Returns

a batch_dim object with the individual batches having their dimensions swapped

References gko::batch_dim< Dimensionality, DimensionType >::at(), gko::batch_dim< Dimensionality, DimensionType >::get_num_batch_entries(), gko::batch_dim< Dimensionality, DimensionType >::stores_equal←_sizes(), and transpose().

40.1.4.72 transpose() [2/2]

```
template<typename DimensionType >
constexpr dim<2, DimensionType> gko::transpose (
            const dim< 2, DimensionType > & dimensions )  [inline], [constexpr], [noexcept]
```

Returns a dim<2> object with its dimensions swapped.

Template Parameters

DimensionType	datatype used to represent each dimension

Parameters

dimensions	original object

Returns

a dim< 2 > object with its dimensions swapped

Referenced by gko::preconditioner::Ic< LSolverType, IndexType >::conj_transpose(), gko::preconditioner::Ilu< L↩ SolverType, USolverType, ReverseApply, IndexType >::conj_transpose(), gko::preconditioner::Ic< LSolverType, IndexType >::transpose(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >↩ ::transpose(), and transpose().

40.1.4.73 unit_root()

```
template<typename T >
constexpr std::complex<remove_complex<T> > gko::unit_root (
            int64 n,
            int64 k = 1 )  [inline], [constexpr]
```

Returns the value of exp(2 ∗ pi ∗ i ∗ k / n), i.e.

an nth root of unity.

Parameters

n	the denominator of the argument
k	the numerator of the argument. Defaults to 1.

Template Parameters

T	the corresponding real value type.

References one().

40.1.4.74 write()

```
template<typename MatrixType , typename StreamType >
void gko::write (
            StreamType && os,
            MatrixType * matrix,
            layout_type layout = detail::mtx_io_traits<std::remove_const_t<MatrixType>>:↩
:default_layout )  [inline]
```

Writes a matrix into an output stream in matrix market format.

Template Parameters

MatrixType	a WritableToMatrixData object providing data to be written.
StreamType	type of stream used to write the data to

Parameters

os	output stream where the data is to be written
matrix	the matrix to write
layout	the layout used in the output

References write_raw().

40.1.4.75 write_binary()

```
template<typename MatrixType , typename StreamType >
void gko::write_binary (
            StreamType && os,
            MatrixType * matrix )  [inline]
```

Writes a matrix into an output stream in binary format.

Note that this format depends on the processor's endianness, so files from a big endian processor can't be read from a little endian processor and vice-versa.

Template Parameters

MatrixType	a WritableToMatrixData object providing data to be written.
StreamType	type of stream used to write the data to

Parameters

os	output stream where the data is to be written
matrix	the matrix to write

References write_binary_raw().

40.1.4.76 write_binary_raw()

```
template<typename ValueType , typename IndexType >
void gko::write_binary_raw (
            std::ostream & os,
            const matrix_data< ValueType, IndexType > & data )
```

Writes a matrix_data structure to a stream in binary format.

Note that this format depends on the processor's endianness, so files from a big endian processor can't be read from a little endian processor and vice-versa.

Template Parameters

ValueType	type of matrix values
IndexType	type of matrix indexes

Parameters

os	output stream where the data is to be written
data	the matrix data to write

Note

This is an advanced routine that writes the raw matrix data structure. If you are trying to write an existing matrix, consider using gko::write_binary instead.

Referenced by write_binary().

40.1.4.77 write_raw()

```
template<typename ValueType , typename IndexType >
void gko::write_raw (
            std::ostream & os,
            const matrix_data< ValueType, IndexType > & data,
            layout_type layout = layout_type::array )
```

Writes a matrix_data structure to a stream in matrix market format.

Template Parameters

ValueType	type of matrix values
IndexType	type of matrix indexes

Parameters

os	output stream where the data is to be written
data	the matrix data to write
layout	the layout used in the output

Note

This is an advanced routine that writes the raw matrix data structure. If you are trying to write an existing matrix, consider using gko::write instead.

Referenced by write().

40.1.4.78 zero() [1/2]

```
template<typename T >
constexpr T gko::zero ( ) [inline], [constexpr]
```

Returns the additive identity for T.

Returns

additive identity for T

40.1.4.79 zero() [2/2]

```
template<typename T >
constexpr T gko::zero (
            const T & ) [inline], [constexpr]
```

Returns the additive identity for T.

Returns

additive identity for T

Note

This version takes an unused reference argument to avoid complicated calls like `zero<decltype(x)>()`. Instead, it allows `zero(x)`.

40.2 gko::accessor Namespace Reference

The accessor namespace.

Classes

- class row_major

  A row_major accessor is a bridge between a range and the row-major memory layout.

40.2.1 Detailed Description

The accessor namespace.

40.3 gko::factorization Namespace Reference

The Factorization namespace.

Classes

- class Ic

  Represents an incomplete Cholesky factorization (IC(0)) of a sparse matrix.
- class Ilu

  Represents an incomplete LU factorization – ILU(0) – of a sparse matrix.
- class ParIc

  ParIC is an incomplete Cholesky factorization which is computed in parallel.
- class ParIct

  ParICT is an incomplete threshold-based Cholesky factorization which is computed in parallel.
- class ParIlu

  ParILU is an incomplete LU factorization which is computed in parallel.
- class ParIlut

  ParILUT is an incomplete threshold-based LU factorization which is computed in parallel.

40.3.1 Detailed Description

The Factorization namespace.

40.4 gko::log Namespace Reference

The logger namespace .

Classes

- class BatchConvergence

  Logs the final residuals and iteration counts for a batch solver.
- class Convergence

  Convergence is a Logger which logs data strictly from the `criterion_check_completed` event.
- struct criterion_data

  Struct representing Criterion related data.
- class EnableLogging

  EnableLogging is a mixin which should be inherited by any class which wants to enable logging.
- struct executor_data

  Struct representing Executor related data.
- struct iteration_complete_data

  Struct representing iteration complete related data.
- struct linop_data

  Struct representing LinOp related data.
- struct linop_factory_data

  Struct representing LinOp factory related data.
- class Loggable

  Loggable class is an interface which should be implemented by classes wanting to support logging.
- struct operation_data

  Struct representing Operator related data.
- struct polymorphic_object_data

  Struct representing PolymorphicObject related data.
- class Record

  Record is a Logger which logs every event to an object.
- class Stream

  Stream is a Logger which logs every event to a stream.

40.4.1 Detailed Description

The logger namespace .

The Logging namespace.

Logging

40.5 gko::matrix Namespace Reference

The matrix namespace.

Classes

- class BatchCsr

  BatchCsr is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class BatchDense

  BatchDense is a batch matrix format which explicitly stores all values of the matrix in each of the batches.
- class BatchDiagonal

  BatchDiagonal is a batch matrix format which explicitly stores all values of the matrix in each of the batches.
- class BatchEll

  BatchEll is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class Coo

  COO stores a matrix in the coordinate matrix format.
- class Csr

  CSR is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).
- class Dense

  Dense is a matrix format which explicitly stores all values of the matrix.
- class Diagonal

  This class is a utility which efficiently implements the diagonal matrix (a linear operator which scales a vector row wise).
- class Ell

  ELL is a matrix format where stride with explicit zeros is used such that all rows have the same number of stored elements.
- class Fbcsr

  Fixed-block compressed sparse row storage matrix format.
- class Fft

  This LinOp implements a 1D Fourier matrix using the FFT algorithm.
- class Fft2

  This LinOp implements a 2D Fourier matrix using the FFT algorithm.
- class Fft3

  This LinOp implements a 3D Fourier matrix using the FFT algorithm.
- class Hybrid

  HYBRID is a matrix format which splits the matrix into ELLPACK and COO format.
- class Identity

  This class is a utility which efficiently implements the identity matrix (a linear operator which maps each vector to itself).

- class IdentityFactory

  This factory is a utility which can be used to generate Identity operators.
- class Permutation

  Permutation is a matrix "format" which stores the row and column permutation arrays which can be used for re-ordering the rows and columns a matrix.
- class RowGatherer

  RowGatherer is a matrix "format" which stores the gather indices arrays which can be used to gather rows to another matrix.
- class Sellp

  SELL-P is a matrix format similar to ELL format.
- class SparsityCsr

  SparsityCsr is a matrix format which stores only the sparsity pattern of a sparse matrix by compressing each row of the matrix (compressed sparse row format).

Functions

- template< typename ValueType >
  void two_sided_batch_system_transform (std::shared_ptr< const Executor > exec, const BatchDiagonal< ValueType > *left, const BatchDiagonal< ValueType > *right, BatchLinOp *mtx, BatchDense< ValueType > *rhs)

  *Transforms the input matrix A and vector b according to S_L*A*S_R and S_L*b where '*' denotes matrix multiplication, and S_L and S_R are the left and right transormation matrices.*

40.5.1 Detailed Description

The matrix namespace.

40.5.2 Function Documentation

40.5.2.1 two_sided_batch_system_transform()

```
template<typename ValueType >
void gko::matrix::two_sided_batch_system_transform (
            std::shared_ptr< const Executor > exec,
            const BatchDiagonal< ValueType > * left,
            const BatchDiagonal< ValueType > * right,
            BatchLinOp * mtx,
            BatchDense< ValueType > * rhs )
```

Transforms the input matrix A and vector b according to S_L*A*S_R and S_L*b where '*' denotes matrix multiplication, and S_L and S_R are the left and right transormation matrices.

Parameters

exec	Exector to run the operation on.
left	Left transformation matrix.
right	Right transformation matrix.
mtx	System matrix to be transformed.
rhs	Right-hand side of the sytem to be transformed.

40.6 gko::multigrid Namespace Reference

The multigrid components namespace.

Classes

- class AmgxPgm

    Amgx parallel graph match (AmgxPgm) is the aggregate method introduced in the paper M.
- class EnableMultigridLevel

    The EnableMultigridLevel gives the default implementation of MultigridLevel with composition and provides `set_↩ multigrid_level` *function.*
- class MultigridLevel

    This class represents two levels in a multigrid hierarchy.

40.6.1 Detailed Description

The multigrid components namespace.

40.7 gko::name_demangling Namespace Reference

The name demangling namespace.

Functions

- template<typename T >
  std::string get_static_type (const T &)

    This function uses name demangling facilities to get the name of the static type (`T`) of the object passed in arguments.
- template<typename T >
  std::string get_dynamic_type (const T &t)

    This function uses name demangling facilities to get the name of the dynamic type of the object passed in arguments.

40.7.1 Detailed Description

The name demangling namespace.

40.7.2 Function Documentation

40.7.2.1 get_dynamic_type()

```
template<typename T >
std::string gko::name_demangling::get_dynamic_type (
            const T & t )
```

This function uses name demangling facilities to get the name of the dynamic type of the object passed in arguments.

Template Parameters

T	the type of the object to demangle

Parameters

t	the object we get the dynamic type of

```
{
    return get_type_name(typeid(t));
}
```

40.7.2.2 get_static_type()

```
template<typename T >
std::string gko::name_demangling::get_static_type (
            const T & )
```

This function uses name demangling facilities to get the name of the static type (`T`) of the object passed in arguments.

Template Parameters

T	the type of the object to demangle

Parameters

unused	

40.8 gko::preconditioner Namespace Reference

The Preconditioner namespace.

Classes

- struct block_interleaved_storage_scheme

  Defines the parameters of the interleaved block storage scheme used by block-Jacobi blocks.
- class Ic

  The Incomplete Cholesky (IC) preconditioner solves the equation $LL^H * x = b$ for a given lower triangular matrix L and the right hand side b (can contain multiple right hand sides).
- class Ilu

  The Incomplete LU (ILU) preconditioner solves the equation $LUx = b$ for a given lower triangular matrix L, an upper triangular matrix U and the right hand side b (can contain multiple right hand sides).
- class Isai

  The Incomplete Sparse Approximate Inverse (ISAI) Preconditioner generates an approximate inverse matrix for a given square matrix A, lower triangular matrix L, upper triangular matrix U or symmetric positive (spd) matrix B.
- class Jacobi

  A block-Jacobi preconditioner is a block-diagonal linear operator, obtained by inverting the diagonal blocks of the source operator.

Enumerations

- enum isai_type

  This enum lists the types of the ISAI preconditioner.

40.8.1 Detailed Description

The Preconditioner namespace.

40.8.2 Enumeration Type Documentation

40.8.2.1 isai_type

```
enum gko::preconditioner::isai_type  [strong]
```

This enum lists the types of the ISAI preconditioner.

ISAI can either be generated for a general square matrix, a lower triangular matrix, an upper triangular matrix or an spd matrix.

```
63 { lower, upper, general, spd };
```

40.9 gko::reorder Namespace Reference

The Reorder namespace.

Classes

- class Rcm

  Rcm is a reordering algorithm minimizing the bandwidth of a matrix.
- class ReorderingBase

  The ReorderingBase class is a base class for all the reordering algorithms.
- struct ReorderingBaseArgs

  This struct is used to pass parameters to the EnableDefaultReorderingBaseFactory::generate() method.

Typedefs

- using ReorderingBaseFactory = AbstractFactory< ReorderingBase, ReorderingBaseArgs >

  Declares an Abstract Factory specialized for ReorderingBases.
- template<typename ConcreteFactory , typename ConcreteReorderingBase , typename ParametersType , typename PolymorphicBase = ReorderingBaseFactory>

  using EnableDefaultReorderingBaseFactory = EnableDefaultFactory< ConcreteFactory, ConcreteReorderingBase, ParametersType, PolymorphicBase >

  This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of ReorderingBaseFactory.

40.9.1 Detailed Description

The Reorder namespace.

The reordering namespace.

40.9.2 Typedef Documentation

40.9.2.1 EnableDefaultReorderingBaseFactory

```
template<typename ConcreteFactory , typename ConcreteReorderingBase , typename ParametersType
, typename PolymorphicBase = ReorderingBaseFactory>
using gko::reorder::EnableDefaultReorderingBaseFactory = typedef EnableDefaultFactory<Concrete↩
Factory, ConcreteReorderingBase, ParametersType, PolymorphicBase>
```

This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of ReorderingBaseFactory.

Template Parameters

ConcreteFactory	the concrete factory which is being implemented [CRTP parmeter]
ConcreteReorderingBase	the concrete ReorderingBase type which this factory produces, needs to have a constructor which takes a const ConcreteFactory $*$, and a const ReorderingBaseArgs $*$ as parameters.
ParametersType	a subclass of enable_parameters_type template which defines all of the parameters of the factory
PolymorphicBase	parent of ConcreteFactory in the polymorphic hierarchy, has to be a subclass of ReorderingBaseFactory

40.10 gko::solver Namespace Reference

The ginkgo Solve namespace.

Namespaces

- multigrid

    The solver multigrid namespace.

Classes

- class BatchBicgstab

    BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.
- class BatchCg

CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

- class BatchDirect

  Solves a batch of linear systems using a vendor-provided dense batched direct solver.

- class BatchGmres

  GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

- class BatchIdr

  IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.

- class BatchRichardson

  The (preconditioned) Richardson solver is an iterative method that uses a preconditioner to approximate the error of the current solution via the current (preconditioned) residual.

- class Bicg

  BICG or the Biconjugate gradient method is a Krylov subspace solver.

- class Bicgstab

  BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.

- class CbGmres

  CB-GMRES or the compressed basis generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

- class Cg

  CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

- class Cgs

  CGS or the conjugate gradient square method is an iterative type Krylov subspace method which is suitable for general systems.

- class EnableBatchSolver
- class Fcg

  FCG or the flexible conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

- class Gmres

  GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

- struct has_with_criteria

  Helper structure to test if the Factory of SolverType has a function `with_criteria`*.*

- struct has_with_criteria< SolverType, xstd::void_t< decltype(SolverType::build().with_criteria(std::shared_ptr< const stop::Crit

  Helper structure to test if the Factory of SolverType has a function `with_criteria`*.*

- class Idr

  IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.

- class Ir

  Iterative refinement (IR) is an iterative method that uses another coarse method to approximate the error of the current solution via the current residual.

- class LowerTrs

  LowerTrs is the triangular solver which solves the system L x = b, when L is a lower triangular matrix.

- class Multigrid

  Multigrid methods have a hierarchy of many levels, whose corase level is a subset of the fine level, of the problem.

- class UpperTrs

  UpperTrs is the triangular solver which solves the system U x = b, when U is an upper triangular matrix.

Functions

- template< typename ValueType >
  auto build_smoother (std::shared_ptr< const LinOpFactory > factory, size_type iteration=1, ValueType relaxation_factor=0.9)

  build_smoother gives a shortcut to build a smoother by IR(Richardson) with limited stop criterion(iterations and relacation_factor).
- template< typename ValueType >
  auto build_smoother (std::shared_ptr< const LinOp > solver, size_type iteration=1, ValueType relaxation_↩ factor=0.9)

  build_smoother gives a shortcut to build a smoother by IR(Richardson) with limited stop criterion(iterations and relacation_factor).

40.10.1 Detailed Description

The ginkgo Solve namespace.

The ginkgo Solver namespace.

40.10.2 Function Documentation

40.10.2.1 build_smoother() [1/2]

```
template<typename ValueType >
auto gko::solver::build_smoother (
            std::shared_ptr< const LinOp > solver,
            size_type iteration = 1,
            ValueType relaxation_factor = 0.9 )
```

build_smoother gives a shortcut to build a smoother by IR(Richardson) with limited stop criterion(iterations and relacation_factor).

Parameters

solver	the shared pointer of solver
iteration	the maximum number of iteraion, which default is 1
relaxation_factor	the relaxation factor for Richardson

Returns

the pointer of Ir(Richardson)

Note

this is the overload function for LinOp.

```
{
    auto exec = solver->get_executor();
    return Ir<ValueType>::build()
        .with_generated_solver(solver)
        .with_relaxation_factor(relaxation_factor)
        .with_criteria(
            gko::stop::Iteration::build().with_max_iters(iteration).on(exec))
        .on(exec);
}
```

40.10.2.2 build_smoother() [2/2]

```
template<typename ValueType >
auto gko::solver::build_smoother (
        std::shared_ptr< const LinOpFactory > factory,
        size_type iteration = 1,
        ValueType relaxation_factor = 0.9 )
```

build_smoother gives a shortcut to build a smoother by IR(Richardson) with limited stop criterion(iterations and relacation_factor).

Parameters

factory	the shared pointer of factory
iteration	the maximum number of iteraion, which default is 1
relaxation_factor	the relaxation factor for Richardson

Returns

the pointer of Ir(Richardson)

40.11 gko::solver::multigrid Namespace Reference

The solver multigrid namespace.

Enumerations

- enum cycle

  cycle defines which kind of multigrid cycle can be used.
- enum mid_smooth_type

  mid_smooth_type gives the options to handle the middle smoother behavior between the two cycles in the same level.

40.11.1 Detailed Description

The solver multigrid namespace.

40.11.2 Enumeration Type Documentation

40.11.2.1 cycle

```
enum gko::solver::multigrid::cycle [strong]
```

cycle defines which kind of multigrid cycle can be used.

It contains V, W, F, and K (KFCG/KGCR) cycle.

- V, W cycle uses the algorithm according to Briggs, Henson, and McCormick: A multigrid tutorial 2nd Edition.
- F cycle uses the algorithm according to Trottenberg, Oosterlee, and Schuller: Multigrid 1st Edition. F cycle first uses the recursive call but second uses the V-cycle call such that F-cycle is between V and W cycle.
- K(KFCG/KGCR) cycle uses the algorithm with up to 2 steps FCG/GCR from Yvan: An aggregation-based algebraic multigrid method

40.11.2.2 mid_smooth_type

```
enum gko::solver::multigrid::mid_smooth_type [strong]
```

mid_smooth_type gives the options to handle the middle smoother behavior between the two cycles in the same level.

It only affects the behavior when there's no operation between the post smoother of previous cycle and the pre smoother of next cycle. Thus, it only affects W cycle and F cycle.

- both: gives the same behavior as the original algorithm, which use posts smoother from previous cycle and pre smoother from next cycle.
- post_smoother: only uses the post smoother of previous cycle in the mid smoother
- pre_smoother: only uses the pre smoother of next cycle in the mid smoother
- standalone: uses the defined smoother in the mid smoother

40.12 gko::stop Namespace Reference

The Stopping criterion namespace.

Classes

- class AbsoluteResidualNorm

  The AbsoluteResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold, i.e.
- class Combined

  The Combined class is used to combine multiple criterions together through an OR operation.
- class Criterion

  The Criterion class is a base class for all stopping criteria.
- struct CriterionArgs

  This struct is used to pass parameters to the EnableDefaultCriterionFactoryCriterionFactory::generate() method.
- class ImplicitResidualNorm

  The ImplicitResidualNorm class is a stopping criterion which stops the iteration process when the implicit residual norm is below a certain threshold relative to.
- class Iteration

  The Iteration class is a stopping criterion which stops the iteration process after a preset number of iterations.
- class RelativeResidualNorm

  The RelativeResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the right-hand side, i.e.
- class ResidualNorm

  The ResidualNorm class is a stopping criterion which stops the iteration process when the actual residual norm is below a certain threshold relative to.
- class ResidualNormBase

  The ResidualNormBase class provides a framework for stopping criteria related to the residual norm.
- class ResidualNormReduction

  The ResidualNormReduction class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the initial residual, i.e.
- class Time

  The Time class is a stopping criterion which stops the iteration process after a certain amout of time has passed.

Typedefs

- using CriterionFactory = AbstractFactory< Criterion, CriterionArgs >

  Declares an Abstract Factory specialized for Criterions.
- template<typename ConcreteFactory , typename ConcreteCriterion , typename ParametersType , typename PolymorphicBase = CriterionFactory>
  using EnableDefaultCriterionFactory = EnableDefaultFactory< ConcreteFactory, ConcreteCriterion, ParametersType, PolymorphicBase >

  This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of CriterionFactory.

Enumerations

- enum mode

  The mode for the residual norm criterion.

Functions

- template<typename FactoryContainer >
  std::shared_ptr< const CriterionFactory > combine (FactoryContainer &&factories)

  Combines multiple criterion factories into a single combined criterion factory.

40.12.1 Detailed Description

The Stopping criterion namespace.

Stopping criteria

40.12.2 Typedef Documentation

40.12.2.1 EnableDefaultCriterionFactory

```
template<typename ConcreteFactory , typename ConcreteCriterion , typename ParametersType ,
typename PolymorphicBase = CriterionFactory>
using gko::stop::EnableDefaultCriterionFactory = typedef EnableDefaultFactory<ConcreteFactory,
ConcreteCriterion, ParametersType, PolymorphicBase>
```

This is an alias for the EnableDefaultFactory mixin, which correctly sets the template parameters to enable a subclass of CriterionFactory.

Template Parameters

ConcreteFactory	the concrete factory which is being implemented [CRTP parmeter]
ConcreteCriterion	the concrete Criterion type which this factory produces, needs to have a constructor which takes a const ConcreteFactory $*$, and a const CriterionArgs $*$ as parameters.
ParametersType	a subclass of enable_parameters_type template which defines all of the parameters of the factory
PolymorphicBase	parent of ConcreteFactory in the polymorphic hierarchy, has to be a subclass of CriterionFactory

40.13 gko::syn Namespace Reference

The Synthesizer namespace.

Classes

- struct range

    range records start, end, step in template
- struct type_list

    type_list records several types in template
- struct value_list

    value_list records several values with the same type in template.

Typedefs

- template< typename List1 , typename List2 >
  using concatenate = typename detail::concatenate_impl< List1, List2 >::type

  concatenate combines two value_list into one value_list.
- template< typename T >
  using as_list = typename detail::as_list_impl< T >::type

  as_list<T> gives the alias type of as_list_impl<T>::type.

Functions

- template< typename T , T... Value>
  constexpr std::array< T, sizeof...(Value)> as_array (value_list< T, Value... > vl)

  as_array<T> returns the array from value_list.

40.13.1 Detailed Description

The Synthesizer namespace.

40.13.2 Typedef Documentation

40.13.2.1 as_list

```
template<typename T >
using gko::syn::as_list = typedef typename detail::as_list_impl<T>::type
```

as_list<T> gives the alias type of as_list_impl<T>::type.

It gives a list (itself) if input is already a list, or generates list type from range input.

Template Parameters

T	list or range

40.13.2.2 concatenate

```
template<typename List1 , typename List2 >
using gko::syn::concatenate = typedef typename detail::concatenate_impl<List1, List2>::type
```

concatenate combines two value_list into one value_list.

Template Parameters

List1	the first list
List2	the second list

40.13.3 Function Documentation

40.13.3.1 as_array()

```
template<typename T , T... Value>
constexpr std::array<T, sizeof...(Value)> gko::syn::as_array (
            value_list< T, Value... > vl ) [constexpr]
```

as_array<T> returns the array from value_list.

It will be helpful if using for in runtime on the array.

Template Parameters

T	the type of value_list
Value	the values of value_list

Parameters

value_list	the input value_list

Returns

std::array the std::array contains the values of value_list

```
204 {
205     return std::array<T, sizeof...(Value)>{Value...};
206 }
```

References gko::array.

40.14 gko::xstd Namespace Reference

The namespace for functionalities after C++14 standard.

40.14.1 Detailed Description

The namespace for functionalities after C++14 standard.

Chapter 41

Class Documentation

41.1 gko::AbsoluteComputable Class Reference

The AbsoluteComputable is an interface that allows to get the component wise absolute of a LinOp.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Public Member Functions

- virtual std::unique_ptr< LinOp > compute_absolute_linop () const =0

  Gets the absolute LinOp.
- virtual void compute_absolute_inplace ()=0

  Compute absolute inplace on each element.

41.1.1 Detailed Description

The AbsoluteComputable is an interface that allows to get the component wise absolute of a LinOp.

Use EnableAbsoluteComputation<AbsoluteLinOp> to implement this interface.

41.1.2 Member Function Documentation

41.1.2.1 compute_absolute_linop()

```
virtual std::unique_ptr<LinOp> gko::AbsoluteComputable::compute_absolute_linop ( ) const
[pure virtual]
```

Gets the absolute LinOp.

Returns

a pointer to the new absolute LinOp

Implemented in gko::EnableAbsoluteComputation< AbsoluteLinOp >, gko::EnableAbsoluteComputation< remove_complex< Hybric
gko::EnableAbsoluteComputation< remove_complex< Sellp< ValueType, IndexType > > >, gko::EnableAbsoluteComputation< re
gko::EnableAbsoluteComputation< remove_complex< Coo< ValueType, IndexType > > >, gko::EnableAbsoluteComputation< ren
gko::EnableAbsoluteComputation< remove_complex< Fbcsr< ValueType, IndexType > > >, gko::EnableAbsoluteComputation< re
and gko::EnableAbsoluteComputation< remove_complex< Ell< ValueType, IndexType > > >.

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

- ginkgo/core/base/lin_op.hpp

41.2 gko::stop::AbsoluteResidualNorm< ValueType > Class Template Reference

The AbsoluteResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold, i.e.

`#include <ginkgo/core/stop/residual_norm.hpp>`

41.2.1 Detailed Description

template<typename ValueType = default_precision>
class gko::stop::AbsoluteResidualNorm< ValueType >

The AbsoluteResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold, i.e.

when norm(residual) / threshold. For better performance, the checks are run thanks to kernels on the executor where the algorithm is executed.

Note

> To use this stopping criterion there are some dependencies. The constructor depends on `b` in order to get the number of right-hand sides. If this is not correctly provided, an exception ::gko::NotSupported() is thrown.

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

- ginkgo/core/stop/residual_norm.hpp

41.3 gko::AbstractFactory< AbstractProductType, ComponentsType > Class Template Reference

The AbstractFactory is a generic interface template that enables easy implementation of the abstract factory design pattern.

`#include <ginkgo/core/base/abstract_factory.hpp>`

Public Member Functions

- template<typename... Args>
  std::unique_ptr< AbstractProductType > generate (Args &&... args) const

  Creates a new product from the given components.

41.3.1 Detailed Description

template<typename AbstractProductType, typename ComponentsType>
class gko::AbstractFactory< AbstractProductType, ComponentsType >

The AbstractFactory is a generic interface template that enables easy implementation of the abstract factory design pattern.

The interface provides the AbstractFactory::generate() method that can produce products of type `AbstractProductType` using an object of `ComponentsType` (which can be constructed on the fly from parameters to its constructors). The generate() method is not declared as virtual, as this allows subclasses to hide the method with a variant that preserves the compile-time type of the objects. Instead, implementers should override the generate_impl() method, which is declared virtual.

Implementers of concrete factories should consider using the EnableDefaultFactory mixin to obtain default implementations of utility methods of PolymorphicObject and AbstractFactory.

Template Parameters

AbstractProductType	the type of products the factory produces
ComponentsType	the type of components the factory needs to produce the product

41.3.2 Member Function Documentation

41.3.2.1 generate()

```
template<typename AbstractProductType, typename ComponentsType>
template<typename... Args>
std::unique_ptr<AbstractProductType> gko::AbstractFactory< AbstractProductType, Components←
Type >::generate (
            Args &&... args ) const [inline]
```

Creates a new product from the given components.

The method will create an ComponentsType object from the arguments of this method, and pass it to the generate←_impl() function which will create a new AbstractProductType.

Template Parameters

Args	types of arguments passed to the constructor of ComponentsType

Parameters

args	arguments passed to the constructor of ComponentsType

Returns

an instance of AbstractProductType

```
{
    auto product = this->generate_impl({std::forward<Args>(args)...});
    for (auto logger : this->loggers_) {
        product->add_logger(logger);
    }
    return product;
}
```

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

- ginkgo/core/base/abstract_factory.hpp

41.4 gko::AllocationError Class Reference

AllocationError is thrown if a memory allocation fails.

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- AllocationError (const std::string &file, int line, const std::string &device, size_type bytes)
  Initializes an allocation error.

41.4.1 Detailed Description

AllocationError is thrown if a memory allocation fails.

41.4.2 Constructor & Destructor Documentation

41.4.2.1 AllocationError()

```
gko::AllocationError::AllocationError (
            const std::string & file,
            int line,
            const std::string & device,
            size_type bytes ) [inline]
```

Initializes an allocation error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
device	The device on which the error occurred
bytes	The size of the memory block whose allocation failed.

```
        : Error(file, line,
                device + ": failed to allocate memory block of " +
                    std::to_string(bytes) + "B")
    {}
```

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

- ginkgo/core/base/exception.hpp

41.5 gko::amd_device Class Reference

amd_device handles the number of executor on Amd devices and have the corresponding recursive_mutex.

`#include <ginkgo/core/base/device.hpp>`

41.5.1 Detailed Description

amd_device handles the number of executor on Amd devices and have the corresponding recursive_mutex.

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

- ginkgo/core/base/device.hpp

41.6 gko::multigrid::AmgxPgm< ValueType, IndexType > Class Template Reference

Amgx parallel graph match (AmgxPgm) is the aggregate method introduced in the paper M.

```
#include <ginkgo/core/multigrid/amgx_pgm.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const
  Returns the system operator (matrix) of the linear system.
- IndexType * get_agg () noexcept
  Returns the aggregate group.
- const IndexType * get_const_agg () const noexcept
  Returns the aggregate group.

41.6.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::multigrid::AmgxPgm< ValueType, IndexType >

Amgx parallel graph match (AmgxPgm) is the aggregate method introduced in the paper M.

Naumov et al., "AmgX: A Library for GPU Accelerated Algebraic Multigrid and Preconditioned Iterative Methods". Current implementation only contains size = 2 version.

AmgxPgm creates the aggregate group according to the matrix value not the structure. AmgxPgm gives two steps (one-phase handshaking) to group the elements. 1: get the strongest neighbor of each unaggregated element. 2: group the elements whose strongest neighbor is each other. repeating until reaching the given conditions. After that, the un-aggregated elements are assigned to an aggregated group or are left alone.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.6.2 Member Function Documentation

41.6.2.1 get_agg()

```
template<typename ValueType = default_precision, typename IndexType = int32>
IndexType* gko::multigrid::AmgxPgm< ValueType, IndexType >::get_agg ( )  [inline], [noexcept]
```

Returns the aggregate group.

Aggregate group whose size is same as the number of rows. Stores the mapping information from row index to coarse row index. i.e., agg[row_idx] = coarse_row_idx.

Returns

the aggregate group.

```
103 { return agg_.get_data(); }
```

References gko::Array< ValueType >::get_data().

41.6.2.2 get_const_agg()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const IndexType* gko::multigrid::AmgxPgm< ValueType, IndexType >::get_const_agg ( ) const
[inline], [noexcept]
```

Returns the aggregate group.

Aggregate group whose size is same as the number of rows. Stores the mapping information from row index to coarse row index. i.e., agg[row_idx] = coarse_row_idx.

Returns

the aggregate group.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.6.2.3 get_system_matrix()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<const LinOp> gko::multigrid::AmgxPgm< ValueType, IndexType >::get_system_↩
matrix ( ) const [inline]
```

Returns the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

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

- ginkgo/core/multigrid/amgx_pgm.hpp

41.7 gko::are_all_integral< Args > Struct Template Reference

Evaluates if all template arguments Args fulfill std::is_integral.

`#include <ginkgo/core/base/types.hpp>`

41.7.1 Detailed Description

template<typename... Args>
struct gko::are_all_integral< Args >

Evaluates if all template arguments Args fulfill std::is_integral.

If that is the case, this class inherits from `std::true_type`, otherwise, it inherits from `std::false_type`. If no values are passed in, `std::true_type` is inherited from.

Template Parameters

Args...	Arguments to test for std::is_integral

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

- ginkgo/core/base/types.hpp

41.8 gko::Array< ValueType > Class Template Reference

An Array is a container which encapsulates fixed-sized arrays, stored on the Executor tied to the Array.

```
#include <ginkgo/core/base/array.hpp>
```

Public Types

- using value_type = ValueType

  The type of elements stored in the array.
- using default_deleter = executor_deleter< value_type[ ]>

  The default deleter type used by Array.
- using view_deleter = null_deleter< value_type[ ]>

  The deleter type used for views.

Public Member Functions

- Array () noexcept

  Creates an empty Array not tied to any executor.
- Array (std::shared_ptr< const Executor > exec) noexcept

  Creates an empty Array tied to the specified Executor.
- Array (std::shared_ptr< const Executor > exec, size_type num_elems)

  Creates an Array on the specified Executor.
- template< typename DeleterType >
  Array (std::shared_ptr< const Executor > exec, size_type num_elems, value_type ∗data, DeleterType deleter)

  Creates an Array from existing memory.
- Array (std::shared_ptr< const Executor > exec, size_type num_elems, value_type ∗data)

  Creates an Array from existing memory.
- template< typename RandomAccessIterator >
  Array (std::shared_ptr< const Executor > exec, RandomAccessIterator begin, RandomAccessIterator end)

  Creates an array on the specified Executor and initializes it with values.
- template< typename T >
  Array (std::shared_ptr< const Executor > exec, std::initializer_list< T > init_list)

  Creates an array on the specified Executor and initializes it with values.
- Array (std::shared_ptr< const Executor > exec, const Array &other)

  Creates a copy of another array on a different executor.
- Array (const Array &other)

  Creates a copy of another array.

- Array (std::shared_ptr< const Executor > exec, Array &&other)

  Moves another array to a different executor.
- Array (Array &&other)

  Moves another array.
- Array< ValueType > as_view ()

  Returns a non-owning view of the memory owned by this array.
- detail::ConstArrayView< ValueType > as_const_view () const

  Returns a non-owning constant view of the memory owned by this array.
- Array & operator= (const Array &other)

  Copies data from another array or view.
- Array & operator= (Array &&other)

  Moves data from another array or view.
- template<typename OtherValueType >
  std::enable_if_t<!std::is_same< ValueType, OtherValueType >::value, Array > & operator= (const Array< OtherValueType > &other)

  Copies and converts data from another array with another data type.
- void clear () noexcept

  Deallocates all data used by the Array.
- void resize_and_reset (size_type num_elems)

  Resizes the array so it is able to hold the specified number of elements.
- void fill (const value_type value)

  Fill the array with the given value.
- size_type get_num_elems () const noexcept

  Returns the number of elements in the Array.
- value_type ∗ get_data () noexcept

  Returns a pointer to the block of memory used to store the elements of the Array.
- const value_type ∗ get_const_data () const noexcept

  Returns a constant pointer to the block of memory used to store the elements of the Array.
- std::shared_ptr< const Executor > get_executor () const noexcept

  Returns the Executor associated with the array.
- void set_executor (std::shared_ptr< const Executor > exec)

  Changes the Executor of the Array, moving the allocated data to the new Executor.
- bool is_owning ()

  Tells whether this Array owns its data or not.

Static Public Member Functions

- static Array view (std::shared_ptr< const Executor > exec, size_type num_elems, value_type ∗data)

  Creates an Array from existing memory.
- static detail::ConstArrayView< ValueType > const_view (std::shared_ptr< const Executor > exec, size_type num_elems, const value_type ∗data)

  Creates a constant (immutable) Array from existing memory.

41.8.1 Detailed Description

template<typename ValueType>
class gko::Array< ValueType >

An Array is a container which encapsulates fixed-sized arrays, stored on the Executor tied to the Array.

The array stores and transfers its data as **raw** memory, which means that the constructors of its elements are not called when constructing, copying or moving the Array. Thus, the Array class is most suitable for storing POD types.

Template Parameters

ValueType	the type of elements stored in the array

41.8.2 Constructor & Destructor Documentation

41.8.2.1 Array() [1/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array ( ) [inline], [noexcept]
```

Creates an empty Array not tied to any executor.

An array without an assigned executor can only be empty. Attempts to change its size (e.g. via the resize_and_reset method) will result in an exception. If such an array is used as the right hand side of an assignment or move assignment expression, the data of the target array will be cleared, but its executor will not be modified.

The executor can later be set by using the set_executor method. If an Array with no assigned executor is assigned or moved to, it will inherit the executor of the source Array.

41.8.2.2 Array() [2/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec ) [inline], [explicit], [noexcept]
```

Creates an empty Array tied to the specified Executor.

Parameters

exec	the Executor where the array data is allocated

41.8.2.3 Array() [3/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            size_type num_elems ) [inline]
```

Creates an Array on the specified Executor.

Parameters

exec	the Executor where the array data will be allocated
num_elems	the amount of memory (expressed as the number of `value_type` elements) allocated on the Executor

41.8.2.4 Array() [4/11]

```
template<typename ValueType>
template<typename DeleterType >
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            size_type num_elems,
            value_type * data,
            DeleterType deleter ) [inline]
```

Creates an Array from existing memory.

The memory will be managed by the array, and deallocated using the specified deleter (e.g. use std::default_delete for data allocated with new).

Template Parameters

DeleterType	type of the deleter

Parameters

exec	executor where `data` is located
num_elems	number of elements in `data`
data	chunk of memory used to create the array
deleter	the deleter used to free the memory

See also

Array::view() to create an array that does not deallocate memory

Array(std::shared_ptr<cont Executor>, size_type, value_type*) to deallocate the memory using Executor::free() method

41.8.2.5 Array() [5/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            size_type num_elems,
            value_type * data ) [inline]
```

Creates an Array from existing memory.

The memory will be managed by the array, and deallocated using the Executor::free method.

Parameters

exec	executor where `data` is located
num_elems	number of elements in `data`
data	chunk of memory used to create the array

41.8.2.6 Array() [6/11]

```
template<typename ValueType>
template<typename RandomAccessIterator >
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            RandomAccessIterator begin,
            RandomAccessIterator end ) [inline]
```

Creates an array on the specified Executor and initializes it with values.

Template Parameters

RandomAccessIterator	type of the iterators

Parameters

exec	the Executor where the array data will be allocated
begin	start of range of values
end	end of range of values

41.8.2.7 Array() [7/11]

```
template<typename ValueType>
template<typename T >
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            std::initializer_list< T > init_list ) [inline]
```

Creates an array on the specified Executor and initializes it with values.

Template Parameters

T	type of values used to initialize the array (T has to be implicitly convertible to value_type)

Parameters

exec	the Executor where the array data will be allocated
init_list	list of values used to initialize the Array

41.8.2.8 Array() [8/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            const Array< ValueType > & other ) [inline]
```

Creates a copy of another array on a different executor.

This does not invoke the constructors of the elements, instead they are copied as POD types.

Parameters

exec	the executor where the new array will be created
other	the Array to copy from

41.8.2.9 Array() [9/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            const Array< ValueType > & other ) [inline]
```

Creates a copy of another array.

This does not invoke the constructors of the elements, instead they are copied as POD types.

Parameters

other	the Array to copy from

41.8.2.10 Array() [10/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            std::shared_ptr< const Executor > exec,
            Array< ValueType > && other ) [inline]
```

Moves another array to a different executor.

This does not invoke the constructors of the elements, instead they are copied as POD types.

Parameters

exec	the executor where the new array will be moved
other	the Array to move

41.8.2.11 Array() [11/11]

```
template<typename ValueType>
gko::Array< ValueType >::Array (
            Array< ValueType > && other ) [inline]
```

Moves another array.

This does not invoke the constructors of the elements, instead they are copied as POD types.

Parameters

other	the Array to move

41.8.3 Member Function Documentation

41.8.3.1 as_const_view()

```
template<typename ValueType>
detail::ConstArrayView<ValueType> gko::Array< ValueType >::as_const_view ( ) const [inline]
```

Returns a non-owning constant view of the memory owned by this array.

It can only be used until this array gets deleted, cleared or resized.

41.8.3.2 as_view()

```
template<typename ValueType>
Array<ValueType> gko::Array< ValueType >::as_view ( ) [inline]
```

Returns a non-owning view of the memory owned by this array.

It can only be used until this array gets deleted, cleared or resized.

41.8.3.3 clear()

```
template<typename ValueType>
void gko::Array< ValueType >::clear ( ) [inline], [noexcept]
```

Deallocates all data used by the Array.

The array is left in a valid, but empty state, so the same array can be used to allocate new memory. Calls to Array::get_data() will return a `nullptr`.

Referenced by gko::index_set< IndexType >::clear(), gko::Array< index_type >::operator=(), and gko::Array< index_type >::resize_and_reset().

41.8.3.4 const_view()

```
template<typename ValueType>
static detail::ConstArrayView<ValueType> gko::Array< ValueType >::const_view (
            std::shared_ptr< const Executor > exec,
            size_type num_elems,
            const value_type * data ) [inline], [static]
```

Creates a constant (immutable) Array from existing memory.

The Array does not take ownership of the memory, and will not deallocate it once it goes out of scope. This array type cannot use the function `resize_and_reset` since it does not own the data it should resize.

Parameters

exec	executor where `data` is located
num_elems	number of elements in `data`
data	chunk of memory used to create the array

Returns

an Array constructed from `data`

Referenced by gko::Array< index_type >::as_const_view().

41.8.3.5 fill()

```
template<typename ValueType>
void gko::Array< ValueType >::fill (
            const value_type value )
```

Fill the array with the given value.

Parameters

value	the value to be filled

41.8.3.6 get_const_data()

```
template<typename ValueType>
const value_type* gko::Array< ValueType >::get_const_data ( ) const [inline], [noexcept]
```

Returns a constant pointer to the block of memory used to store the elements of the Array.

Returns

a constant pointer to the block of memory used to store the elements of the Array

Referenced by gko::Array< index_type >::as_const_view(), gko::matrix::BatchDense< ValueType >::at(), gko↩ ::matrix::BatchDiagonal< ValueType >::at(), gko::matrix::Dense< ValueType >::at(), gko::preconditioner::Jacobi< ValueType, IndexType >::get_blocks(), gko::preconditioner::Jacobi< ValueType, IndexType >::get_conditioning(), gko::multigrid::AmgxPgm< ValueType, IndexType >::get_const_agg(), gko::matrix::Sellp< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::SparsityCsr< ValueType, IndexType >::get_const_col_idxs(), gko::matrix↩ ::Ell< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::Coo< ValueType, IndexType >::get_const↩ _col_idxs(), gko::matrix::BatchCsr< ValueType, IndexType >::get_const_col_idxs(), gko::device_matrix_data< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::BatchEll< ValueType, IndexType >::get_const_col_↩ idxs(), gko::matrix::Fbcsr< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::Csr< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::Permutation< IndexType >::get_const_permutation(), gko::matrix::Row↩ Gatherer< IndexType >::get_const_row_idxs(), gko::matrix::Coo< ValueType, IndexType >::get_const_row_idxs(), gko::device_matrix_data< ValueType, IndexType >::get_const_row_idxs(), gko::matrix::SparsityCsr< ValueType, IndexType >::get_const_row_ptrs(), gko::matrix::BatchCsr< ValueType, IndexType >::get_const_row_ptrs(), gko↩ ::matrix::Fbcsr< ValueType, IndexType >::get_const_row_ptrs(), gko::matrix::Csr< ValueType, IndexType >::get↩ _const_row_ptrs(), gko::matrix::Sellp< ValueType, IndexType >::get_const_slice_lengths(), gko::matrix::Sellp< ValueType, IndexType >::get_const_slice_sets(), gko::matrix::Csr< ValueType, IndexType >::get_const_srow(), gko::matrix::SparsityCsr< ValueType, IndexType >::get_const_value(), gko::matrix::Diagonal< ValueType >↩ ::get_const_values(), gko::matrix::Sellp< ValueType, IndexType >::get_const_values(), gko::matrix::Ell< Value↩ Type, IndexType >::get_const_values(), gko::matrix::Coo< ValueType, IndexType >::get_const_values(), gko↩ ::matrix::BatchCsr< ValueType, IndexType >::get_const_values(), gko::matrix::BatchEll< ValueType, IndexType >::get_const_values(), gko::matrix::BatchDense< ValueType >::get_const_values(), gko::matrix::BatchDiagonal< ValueType >::get_const_values(), gko::device_matrix_data< ValueType, IndexType >::get_const_values(), gko↩ ::matrix::Fbcsr< ValueType, IndexType >::get_const_values(), gko::matrix::Dense< ValueType >::get_const_↩ values(), gko::matrix::Csr< ValueType, IndexType >::get_const_values(), gko::matrix::BatchDense< ValueType >::get_num_stored_elements(), gko::matrix::BatchDiagonal< ValueType >::get_num_stored_elements(), gko↩ ::index_set< IndexType >::get_subsets_begin(), gko::index_set< IndexType >::get_subsets_end(), gko::index↩ _set< IndexType >::get_superset_indices(), gko::matrix::BatchDense< ValueType >::get_values(), gko::matrix↩ ::BatchDiagonal< ValueType >::get_values(), gko::matrix::Csr< ValueType, IndexType >::classical::process(), gko::matrix::Csr< ValueType, IndexType >::load_balance::process(), gko::matrix::BatchDiagonal< ValueType >↩ ::unbatch(), gko::matrix::BatchDense< ValueType >::unbatch(), gko::matrix::Ell< ValueType, IndexType >::val_↩ at(), gko::matrix::Sellp< ValueType, IndexType >::val_at(), and gko::matrix::BatchEll< ValueType, IndexType >↩ ::val_at().

41.8.3.7 get_data()

```
template<typename ValueType>
value_type* gko::Array< ValueType >::get_data ( ) [inline], [noexcept]
```

Returns a pointer to the block of memory used to store the elements of the Array.

Returns

a pointer to the block of memory used to store the elements of the Array

Referenced by gko::Array< index_type >::as_view(), gko::matrix::BatchDense< ValueType >::at(), gko::matrix↩ ::BatchDiagonal< ValueType >::at(), gko::matrix::Dense< ValueType >::at(), gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::compute_ell_num_stored_elements_per_row(), gko::multigrid::AmgxPgm< Value↩ Type, IndexType >::get_agg(), gko::matrix::Sellp< ValueType, IndexType >::get_col_idxs(), gko::matrix::Sparsity↩ Csr< ValueType, IndexType >::get_col_idxs(), gko::matrix::Ell< ValueType, IndexType >::get_col_idxs(), gko↩ ::matrix::Coo< ValueType, IndexType >::get_col_idxs(), gko::matrix::BatchCsr< ValueType, IndexType >::get↩ _col_idxs(), gko::matrix::BatchEll< ValueType, IndexType >::get_col_idxs(), gko::device_matrix_data< Value↩ Type, IndexType >::get_col_idxs(), gko::matrix::Fbcsr< ValueType, IndexType >::get_col_idxs(), gko::matrix::Csr<

ValueType, IndexType >::get_col_idxs(), gko::matrix::Permutation< IndexType >::get_permutation(), gko::matrix::RowGatherer< IndexType >::get_row_idxs(), gko::matrix::Coo< ValueType, IndexType >::get_row_idxs(), gko::device_matrix_data< ValueType, IndexType >::get_row_idxs(), gko::matrix::SparsityCsr< ValueType, IndexType >::get_row_ptrs(), gko::matrix::BatchCsr< ValueType, IndexType >::get_row_ptrs(), gko::matrix::Fbcsr< ValueType, IndexType >::get_row_ptrs(), gko::matrix::Csr< ValueType, IndexType >::get_row_ptrs(), gko::matrix::Sellp< ValueType, IndexType >::get_slice_lengths(), gko::matrix::Sellp< ValueType, IndexType >::get_slice_sets(), gko::matrix::Csr< ValueType, IndexType >::get_srow(), gko::matrix::SparsityCsr< ValueType, IndexType >::get_value(), gko::matrix::Diagonal< ValueType >::get_values(), gko::matrix::Sellp< ValueType, IndexType >::get_values(), gko::matrix::Ell< ValueType, IndexType >::get_values(), gko::matrix::Coo< ValueType, IndexType >::get_values(), gko::matrix::BatchCsr< ValueType, IndexType >::get_values(), gko::matrix::BatchDense< ValueType >::get_values(), gko::matrix::BatchEll< ValueType, IndexType >::get_values(), gko::matrix::BatchDiagonal< ValueType >::get_values(), gko::device_matrix_data< ValueType, IndexType >::get_values(), gko::matrix::Fbcsr< ValueType, IndexType >::get_values(), gko::matrix::Dense< ValueType >::get_values(), gko::matrix::Csr< ValueType, IndexType >::get_values(), gko::Array< index_type >::operator=(), gko::matrix::Csr< ValueType, IndexType >::load_balance::process(), gko::matrix::Ell< ValueType, IndexType >::val_at(), gko::matrix::Sellp< ValueType, IndexType >::val_at(), and gko::matrix::BatchEll< ValueType, IndexType >::val_at().

41.8.3.8 get_executor()

```
template<typename ValueType>
std::shared_ptr<const Executor> gko::Array< ValueType >::get_executor ( ) const  [inline],
[noexcept]
```

Returns the Executor associated with the array.

Returns

the Executor associated with the array

Referenced by gko::Array< index_type >::as_const_view(), gko::Array< index_type >::as_view(), gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::compute_hybrid_config(), gko::device_matrix_data< ValueType, IndexType >::get_executor(), gko::matrix::Csr< ValueType, IndexType >::classical::process(), and gko::matrix::Csr< ValueType, IndexType >::load_balance::process().

41.8.3.9 get_num_elems()

```
template<typename ValueType>
size_type gko::Array< ValueType >::get_num_elems ( ) const  [inline], [noexcept]
```

Returns the number of elements in the Array.

Returns

the number of elements in the Array

Referenced by gko::Array< index_type >::as_const_view(), gko::Array< index_type >::as_view(), gko::matrix::↩
Hybrid< ValueType, IndexType >::imbalance_limit::compute_ell_num_stored_elements_per_row(), gko::matrix↩
::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::compute_ell_num_stored_elements_per_row(), gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::compute_hybrid_config(), gko::device_matrix_↩
data< ValueType, IndexType >::get_num_elems(), gko::matrix::SparsityCsr< ValueType, IndexType >::get↩
_num_nonzeros(), gko::matrix::Csr< ValueType, IndexType >::get_num_srow_elements(), gko::matrix::Fbcsr< ValueType, IndexType >::get_num_stored_blocks(), gko::matrix::Ell< ValueType, IndexType >::get_num_↩
stored_elements(), gko::matrix::Coo< ValueType, IndexType >::get_num_stored_elements(), gko::matrix::Sellp< ValueType, IndexType >::get_num_stored_elements(), gko::matrix::BatchCsr< ValueType, IndexType >::get↩
_num_stored_elements(), gko::matrix::BatchDense< ValueType >::get_num_stored_elements(), gko::matrix↩
::BatchEll< ValueType, IndexType >::get_num_stored_elements(), gko::matrix::BatchDiagonal< ValueType >↩
::get_num_stored_elements(), gko::preconditioner::Jacobi< ValueType, IndexType >::get_num_stored_elements(), gko::matrix::Fbcsr< ValueType, IndexType >::get_num_stored_elements(), gko::matrix::Dense< ValueType >↩
::get_num_stored_elements(), gko::matrix::Csr< ValueType, IndexType >::get_num_stored_elements(), gko↩
::index_set< IndexType >::get_num_subsets(), gko::matrix::Permutation< IndexType >::get_permutation_size(), gko::matrix::Sellp< ValueType, IndexType >::get_total_cols(), gko::index_set< IndexType >::index_set(), gko↩
::matrix::Csr< ValueType, IndexType >::classical::process(), and gko::matrix::Csr< ValueType, IndexType >↩
::load_balance::process().

41.8.3.10 is_owning()

```
template<typename ValueType>
bool gko::Array< ValueType >::is_owning ( ) [inline]
```

Tells whether this Array owns its data or not.

Views do not own their data and this has multiple implications. They cannot be resized since the data is not owned by the Array which stores a view. It is also unclear whether custom deleter types are owning types as they could be a user-created view-type, therefore only proper Array which use the `default_deleter` are considered owning types.

Returns

whether this Array can be resized or not.

Referenced by gko::Array< index_type >::operator=(), and gko::Array< index_type >::resize_and_reset().

41.8.3.11 operator=() [1/3]

```
template<typename ValueType>
Array& gko::Array< ValueType >::operator= (
            Array< ValueType > && other ) [inline]
```

Moves data from another array or view.

Only the pointer and deleter type change, a copy only happens when targeting another executor's data. This means that in the following situation:

```
gko::Array<int> a; // an existing array or view
gko::Array<int> b; // an existing array or view
b = std::move(a);
```

Depending on whether `a` and `b` are array or view, this happens:

- a and b are views, b becomes the only valid view of a;
- a and b are arrays, b becomes the only valid array of a;
- a is a view and b is an array, b frees its data and becomes the only valid view of a ();
- a is an array and b is a view, b becomes the only valid array of a.

In all the previous cases, a becomes invalid (e.g., a `nullptr`).

This does not invoke the constructors of the elements, instead they are copied as POD types.

The executor of this is preserved. In case this does not have an assigned executor, it will inherit the executor of other.

Parameters

other	the Array to move data from

Returns

this

41.8.3.12 operator=() [2/3]

```
template<typename ValueType>
Array& gko::Array< ValueType >::operator= (
            const Array< ValueType > & other )  [inline]
```

Copies data from another array or view.

In the case of an array target, the array is resized to match the source's size. In the case of a view target, if the dimensions are not compatible a gko::OutOfBoundsError is thrown.

This does not invoke the constructors of the elements, instead they are copied as POD types.

The executor of this is preserved. In case this does not have an assigned executor, it will inherit the executor of other.

Parameters

other	the Array to copy from

Returns

this

41.8.3.13 operator=() [3/3]

```
template<typename ValueType>
template<typename OtherValueType >
std::enable_if_t<!std::is_same<ValueType, OtherValueType>::value, Array>& gko::Array< Value←
Type >::operator= (
            const Array< OtherValueType > & other ) [inline]
```

Copies and converts data from another array with another data type.

In the case of an array target, the array is resized to match the source's size. In the case of a view target, if the dimensions are not compatible a gko::OutOfBoundsError is thrown.

This does not invoke the constructors of the elements, instead they are copied as POD types.

The executor of this is preserved. In case this does not have an assigned executor, it will inherit the executor of other.

Parameters

other	the Array to copy from

Template Parameters

OtherValueType	the value type of `other`

Returns

this

41.8.3.14 resize_and_reset()

```
template<typename ValueType>
void gko::Array< ValueType >::resize_and_reset (
            size_type num_elems ) [inline]
```

Resizes the array so it is able to hold the specified number of elements.

For a view and other non-owning Array types, this throws an exception since these types cannot be resized.

All data stored in the array will be lost.

If the Array is not assigned an executor, an exception will be thrown.

Parameters

num_elems	the amount of memory (expressed as the number of `value_type` elements) allocated on the Executor

Referenced by gko::Array< index_type >::operator=().

41.8.3.15 set_executor()

```
template<typename ValueType>
void gko::Array< ValueType >::set_executor (
            std::shared_ptr< const Executor > exec )  [inline]
```

Changes the Executor of the Array, moving the allocated data to the new Executor.

Parameters

exec	the Executor where the data will be moved to

41.8.3.16 view()

```
template<typename ValueType>
static Array gko::Array< ValueType >::view (
            std::shared_ptr< const Executor > exec,
            size_type num_elems,
            value_type * data )  [inline], [static]
```

Creates an Array from existing memory.

The Array does not take ownership of the memory, and will not deallocate it once it goes out of scope. This array type cannot use the function `resize_and_reset` since it does not own the data it should resize.

Parameters

exec	executor where `data` is located
num_elems	number of elements in `data`
data	chunk of memory used to create the array

Returns

an Array constructed from `data`

Referenced by gko::Array< index_type >::as_view(), and gko::make_array_view().

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

- ginkgo/core/base/array.hpp

41.9 gko::device_matrix_data< ValueType, IndexType >::arrays Struct Reference

Stores the internal arrays of a device_matrix_data object.

```
#include <ginkgo/core/base/device_matrix_data.hpp>
```

41.9.1 Detailed Description

template<typename ValueType, typename IndexType>
struct gko::device_matrix_data< ValueType, IndexType >::arrays

Stores the internal arrays of a device_matrix_data object.

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

- ginkgo/core/base/device_matrix_data.hpp

41.10 gko::matrix::Hybrid< ValueType, IndexType >::automatic Class Reference

automatic is a strategy_type which decides the number of stored elements per row of the ell part automatically.

```
#include <ginkgo/core/matrix/hybrid.hpp>
```

Public Member Functions

- automatic ()
  Creates an automatic strategy.
- size_type compute_ell_num_stored_elements_per_row (Array< size_type > *row_nnz) const override
  Computes the number of stored elements per row of the ell part.

41.10.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >::automatic

automatic is a strategy_type which decides the number of stored elements per row of the ell part automatically.

41.10.2 Member Function Documentation

41.10.2.1 compute_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::automatic::compute_ell_num_stored_←
elements_per_row (
            Array< size_type > * row_nnz ) const  [inline], [override], [virtual]
```

Computes the number of stored elements per row of the ell part.

Parameters

row_nnz	the number of nonzeros of each row

Returns

the number of stored elements per row of the ell part

Implements gko::matrix::Hybrid< ValueType, IndexType >::strategy_type.

References gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::compute_ell_num_stored←_elements_per_row().

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

- ginkgo/core/matrix/hybrid.hpp

41.11 gko::BadDimension Class Reference

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

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- BadDimension (const std::string &file, int line, const std::string &func, const std::string &op_name, size_type op_num_rows, size_type op_num_cols, const std::string &clarification)
  Initializes a bad dimension error.

41.11.1 Detailed Description

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

41.11.2 Constructor & Destructor Documentation

41.11.2.1 BadDimension()

```
gko::BadDimension::BadDimension (
            const std::string & file,
            int line,
            const std::string & func,
            const std::string & op_name,
            size_type op_num_rows,
            size_type 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

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

- ginkgo/core/base/exception.hpp

41.12 gko::batch_dim< Dimensionality, DimensionType > Struct Template Reference

A type representing the dimensions of a multidimensional batch object.

`#include <ginkgo/core/base/dim.hpp>`

Public Member Functions

- bool stores_equal_sizes () const

    Checks if the batch_dim object stores equal sizes.
- size_type get_num_batch_entries () const

    Get the number of batch entries stored.
- std::vector< dim< dimensionality, dimension_type > > get_batch_sizes () const

    Get the sizes of all entries as a std::vector.
- const dim< dimensionality, dimension_type > & at (const size_type batch_entry=0) const

    Get the batch size at a particular index.
- batch_dim (const size_type num_batch_entries=0, const dim< dimensionality, dimension_type > &common_size=dim< dimensionality, dimension_type >{})

    Creates a batch_dim object which stores a uniform size for all batch entries.
- batch_dim (const std::vector< dim< dimensionality, dimension_type >> &batch_sizes)

    Creates a batch_dim object which stores possibly non-uniform sizes for the different batch entries.

Friends

- bool operator== (const batch_dim &x, const batch_dim &y)

    Checks if two batch_dim objects are equal.

41.12.1 Detailed Description

template<size_type Dimensionality = 2, typename DimensionType = size_type>
struct gko::batch_dim< Dimensionality, DimensionType >

A type representing the dimensions of a multidimensional batch object.

Template Parameters

Dimensionality	number of dimensions of the object
DimensionType	datatype used to represent each dimension

41.12.2 Constructor & Destructor Documentation

41.12.2.1 batch_dim() [1/2]

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
gko::batch_dim< Dimensionality, DimensionType >::batch_dim (
            const size_type num_batch_entries = 0,
            const dim< dimensionality, dimension_type > & common_size = dim<dimensionality, dimension↩
_type>{} )  [inline], [explicit]
```

Creates a batch_dim object which stores a uniform size for all batch entries.

Parameters

num_batch_entries	number of batch entries to be stored
common_size	the common size of all the batch entries stored

Note

Use this constructor when uniform batches need to be stored.

41.12.2.2 batch_dim() [2/2]

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
gko::batch_dim< Dimensionality, DimensionType >::batch_dim (
            const std::vector< dim< dimensionality, dimension_type >> & batch_sizes )  [inline]
```

Creates a batch_dim object which stores possibly non-uniform sizes for the different batch entries.

Parameters

batch_sizes	the std::vector object that stores the batch_sizes

Note

Use this constructor when non-uniform batches need to be stored.

41.12.3 Member Function Documentation

41.12.3.1 at()

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
const dim<dimensionality, dimension_type>& gko::batch_dim< Dimensionality, DimensionType >↩
::at (
            const size_type batch_entry = 0 ) const  [inline]
```

Get the batch size at a particular index.

Parameters

batch_entry	the index of the entry whose size is needed

Returns

the size of the batch entry at the requested batch-index

Referenced by gko::transpose().

41.12.3.2 get_batch_sizes()

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
std::vector<dim<dimensionality, dimension_type> > gko::batch_dim< Dimensionality, Dimension↩
Type >::get_batch_sizes ( ) const  [inline]
```

Get the sizes of all entries as a std::vector.

Returns

the std::vector of batch sizes

41.12.3.3 get_num_batch_entries()

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
size_type gko::batch_dim< Dimensionality, DimensionType >::get_num_batch_entries ( ) const
[inline]
```

Get the number of batch entries stored.

Returns

num_batch_entries

Referenced by gko::transpose().

41.12.3.4 stores_equal_sizes()

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
bool gko::batch_dim< Dimensionality, DimensionType >::stores_equal_sizes ( ) const  [inline]
```

Checks if the batch_dim object stores equal sizes.

Returns

bool representing whether equal sizes are being stored

Referenced by gko::transpose().

41.12.4 Friends And Related Function Documentation

41.12.4.1 operator==

```
template<size_type Dimensionality = 2, typename DimensionType = size_type>
bool operator== (
            const batch_dim< Dimensionality, DimensionType > & x,
            const batch_dim< Dimensionality, DimensionType > & y )  [friend]
```

Checks if two batch_dim objects are equal.

Parameters

x	first object
y	second object

Returns

true if and only if all dimensions of both objects are equal.

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

- ginkgo/core/base/dim.hpp

41.13 gko::solver::BatchBicgstab< ValueType > Class Template Reference

BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.

```
#include <ginkgo/core/solver/batch_bicgstab.hpp>
```

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override

  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override

  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- bool apply_uses_initial_guess () const override

  Return true as iterative solvers use the data in x as an initial guess.

41.13.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::BatchBicgstab< ValueType >

BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.

Being a generic solver, it is capable of solving general matrices, including non-s.p.d matrices.

This solver solves a batch of linear systems using Bicgstab algorithm.

Unless otherwise specified via the `preconditioner` factory parameter, this implementation does not use any preconditioner by default. The type of tolerance(absolute or relative) and the maximum number of iterations to be used in the stopping criterion can be set via the factory parameters.

Template Parameters

ValueType	precision of matrix elements

41.13.2 Member Function Documentation

41.13.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchBicgstab< ValueType >::apply_uses_initial_guess ( ) const  [inline],
[override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
90 { return true; }
```

41.13.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchBicgstab< ValueType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.13.2.3 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchBicgstab< ValueType >::transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

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

- ginkgo/core/solver/batch_bicgstab.hpp

41.14 gko::solver::BatchCg< ValueType > Class Template Reference

CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

```
#include <ginkgo/core/solver/batch_cg.hpp>
```

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.

41.14.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::BatchCg< ValueType >

CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

Though this method performs very well for symmetric positive definite matrices, it is in general not suitable for general matrices.

This solver solves a batch of linear systems using Cg algorithm.

Unless otherwise specified via the `preconditioner` factory parameter, this implementation does not use any preconditioner by default. The type of tolerance(absolute or relative) and the maximum number of iterations to be used in the stopping criterion can be set via the factory parameters.

Template Parameters

ValueType	precision of matrix elements

41.14.2 Member Function Documentation

41.14.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchCg< ValueType >::apply_uses_initial_guess ( ) const  [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
93 { return true; }
```

41.14.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchCg< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.14.2.3 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchCg< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

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

- ginkgo/core/solver/batch_cg.hpp

41.15 gko::log::BatchConvergence< ValueType > Class Template Reference

Logs the final residuals and iteration counts for a batch solver.

```
#include <ginkgo/core/log/batch_convergence.hpp>
```

Public Member Functions

- void on_batch_solver_completed (const Array< int > &num_iterations, const BatchLinOp *residual_norm) const override

  Copies arrays of iterations and residual norms into this (on the host).
- const Array< int > & get_num_iterations () const noexcept
- const matrix::BatchDense< real_type > * get_residual_norm () const noexcept

Static Public Member Functions

- static std::unique_ptr< BatchConvergence > create (std::shared_ptr< const Executor > exec, const mask↩_type &enabled_events=Logger::all_events_mask)

  Creates a convergence logger.

41.15.1 Detailed Description

template<typename ValueType = default_precision>
class gko::log::BatchConvergence< ValueType >

Logs the final residuals and iteration counts for a batch solver.

The purpose of this logger is to give simple access to standard data generated by the solver once it has converged.

41.15.2 Member Function Documentation

41.15.2.1 create()

```
template<typename ValueType = default_precision>
static std::unique_ptr<BatchConvergence> gko::log::BatchConvergence< ValueType >::create (
            std::shared_ptr< const Executor > exec,
            const mask_type & enabled_events = Logger::all_events_mask ) [inline], [static]
```

Creates a convergence logger.

This dynamically allocates the memory, constructs the object and returns an std::unique_ptr to this object.

Parameters

exec	the executor
enabled_events	the events enabled for this logger. By default all events.

Returns

an std::unique_ptr to the the constructed object

```
{
    return std::unique_ptr<BatchConvergence>(
        new BatchConvergence(exec, enabled_events));
}
```

41.15.2.2 get_num_iterations()

```
template<typename ValueType = default_precision>
const Array<int>& gko::log::BatchConvergence< ValueType >::get_num_iterations ( ) const [inline],
[noexcept]
```

Returns

The number of iterations for entire batch

41.15.2.3 get_residual_norm()

```
template<typename ValueType = default_precision>
const matrix::BatchDense<real_type>* gko::log::BatchConvergence< ValueType >::get_residual_↩
norm ( ) const [inline], [noexcept]
```

Returns

The residual norms for the entire batch.

41.15.2.4 on_batch_solver_completed()

```
template<typename ValueType = default_precision>
void gko::log::BatchConvergence< ValueType >::on_batch_solver_completed (
            const Array< int > & num_iterations,
            const BatchLinOp * residual_norm ) const  [override]
```

Copies arrays of iterations and residual norms into this (on the host).

The arguments can be on any executor and the data is copied to the host executor.

Parameters

num_iterations	Array (size number of matrices x number of right-hand sides) which stores the iteration count at which each RHS of each linear system converged. The convergence iteration count for the different RHS are stored contiguously.
residual_norm	A BatchDense matrix of size num_matrices x 1 x num_RHS, which stores the final residual norms.

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

- ginkgo/core/log/batch_convergence.hpp

41.16 gko::matrix::BatchCsr< ValueType, IndexType > Class Template Reference

BatchCsr is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).

```
#include <ginkgo/core/matrix/batch_csr.hpp>
```

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- std::vector< std::unique_ptr< unbatch_type > > unbatch () const
  Unbatches the BatchCsr matrix into distinct matrices of Csr type.
- void sort_by_column_index ()
  Sorts all (value, col_idx) pairs in each row by column index.
- bool is_sorted_by_column_index () const
  Tests if all row entry pairs (value, col_idx) are sorted by column index.
- value_type * get_values () noexcept
  Returns the values of the matrix.
- const value_type * get_const_values () const noexcept
  Returns the values of the matrix.
- index_type * get_col_idxs () noexcept
  Returns the column indexes of the matrix.

- const index_type ∗ get_const_col_idxs () const noexcept

  Returns the column indexes of the matrix.
- index_type ∗ get_row_ptrs () noexcept

  Returns the row pointers of the matrix.
- const index_type ∗ get_const_row_ptrs () const noexcept

  Returns the row pointers of the matrix.
- size_type get_num_stored_elements () const noexcept

  Returns the number of elements explicitly stored in the matrix, cumulative over all the batches.

Static Public Member Functions

- static std::unique_ptr< const BatchCsr > create_const (std::shared_ptr< const Executor > exec, const batch_dim< 2 > &size, gko::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs, gko::detail::ConstArrayView< IndexType > &&row_ptrs)

  Creates a constant BatchCsr matrix by wrapping a set of constant arrays.

41.16.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::BatchCsr< ValueType, IndexType >

BatchCsr is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).

Each of the individual batches are stored in a CSR matrix.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

Note

Currently, BatchCsr can store matrices with batch entries that have the same sparsity pattern, but different values.

41.16.2 Member Function Documentation

41.16.2.1 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<BatchLinOp> gko::matrix::BatchCsr< ValueType, IndexType >::conj_transpose ( )
const  [override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.16.2.2 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const BatchCsr> gko::matrix::BatchCsr< ValueType, IndexType >::create↩
_const (
            std::shared_ptr< const Executor > exec,
            const batch_dim< 2 > & size,
            gko::detail::ConstArrayView< ValueType > && values,
            gko::detail::ConstArrayView< IndexType > && col_idxs,
            gko::detail::ConstArrayView< IndexType > && row_ptrs )  [inline], [static]
```

Creates a constant BatchCsr matrix by wrapping a set of constant arrays.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
col_idxs	the column index array of the matrix
row_ptrs	the row pointer array of the matrix

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of these arrays on the correct executor.

```
    {
        // cast const-ness away, but return a const object afterwards,
        // so we can ensure that no modifications take place.
        return std::unique_ptr<const BatchCsr>(new BatchCsr{
            exec, size, gko::detail::array_const_cast(std::move(values)),
            gko::detail::array_const_cast(std::move(col_idxs)),
            gko::detail::array_const_cast(std::move(row_ptrs))});
    }
```

41.16.2.3 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::BatchCsr< ValueType, IndexType >::get_col_idxs ( )  [inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

References gko::Array< ValueType >::get_data().

41.16.2.4 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::BatchCsr< ValueType, IndexType >::get_const_col_idxs ( ) const
[inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchCsr< ValueType, IndexType >::unbatch().

41.16.2.5 get_const_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::BatchCsr< ValueType, IndexType >::get_const_row_ptrs ( ) const
[inline], [noexcept]
```

Returns the row pointers of the matrix.

Returns

the row pointers of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchCsr< ValueType, IndexType >::unbatch().

41.16.2.6 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::BatchCsr< ValueType, IndexType >::get_const_values ( ) const
[inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchCsr< ValueType, IndexType >::unbatch().

41.16.2.7 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::BatchCsr< ValueType, IndexType >::get_num_stored_elements ( ) const
[inline], [noexcept]
```

Returns the number of elements explicitly stored in the matrix, cumulative over all the batches.

Returns

the number of elements explicitly stored in the matrix, cumulative over all the batches

References gko::Array< ValueType >::get_num_elems().

Referenced by gko::matrix::BatchCsr< ValueType, IndexType >::unbatch().

41.16.2.8 get_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::BatchCsr< ValueType, IndexType >::get_row_ptrs ( ) [inline], [noexcept]
```

Returns the row pointers of the matrix.

Returns

the row pointers of the matrix.

References gko::Array< ValueType >::get_data().

41.16.2.9 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::BatchCsr< ValueType, IndexType >::get_values ( )  [inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

References gko::Array< ValueType >::get_data().

41.16.2.10 is_sorted_by_column_index()

```
template<typename ValueType = default_precision, typename IndexType = int32>
bool gko::matrix::BatchCsr< ValueType, IndexType >::is_sorted_by_column_index ( ) const
```

Tests if all row entry pairs (value, col_idx) are sorted by column index.

Returns

True if all row entry pairs (value, col_idx) are sorted by column index

41.16.2.11 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<BatchLinOp> gko::matrix::BatchCsr< ValueType, IndexType >::transpose ( )
const  [override], [virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

41.16.2.12 unbatch()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::vector<std::unique_ptr<unbatch_type> > gko::matrix::BatchCsr< ValueType, IndexType >↩
::unbatch ( ) const [inline]
```

Unbatches the BatchCsr matrix into distinct matrices of Csr type.

Returns

a std::vector containing the distinct Csr matrices.

References gko::matrix::BatchCsr< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::BatchCsr< ValueType, IndexType >::get_const_row_ptrs(), gko::matrix::BatchCsr< ValueType, IndexType >::get_const↩ _values(), gko::PolymorphicObject::get_executor(), and gko::matrix::BatchCsr< ValueType, IndexType >::get_↩ num_stored_elements().

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

- ginkgo/core/matrix/batch_csr.hpp

41.17 gko::matrix::BatchDense< ValueType > Class Template Reference

BatchDense is a batch matrix format which explicitly stores all values of the matrix in each of the batches.

```
#include <ginkgo/core/matrix/batch_dense.hpp>
```

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- std::vector< std::unique_ptr< unbatch_type > > unbatch () const
  Unbatches the batched dense and creates a std::vector of Dense matrices.
- value_type ∗ get_values () noexcept
  Returns a pointer to the array of values of the matrix.
- value_type ∗ get_values (size_type batch) noexcept
  Returns a pointer to the array of values of the matrix.
- const value_type ∗ get_const_values () const noexcept
  Returns a pointer to the array of values of the matrix.
- const value_type ∗ get_const_values (size_type batch) const noexcept
  Returns a pointer to the array of values of the matrix.
- const batch_stride & get_stride () const noexcept
  Returns the batch_stride of the matrix.
- size_type get_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the batch matrix, cumulative across all the batches.
- size_type get_num_stored_elements (size_type batch) const noexcept
  Returns the number of elements explicitly stored at a specific batch index.
- value_type & at (size_type batch, size_type row, size_type col) noexcept

Returns a single element for a particular batch.
- value_type at (size_type batch, size_type row, size_type col) const noexcept

  Returns a single element for a particular batch.
- ValueType & at (size_type batch, size_type idx) noexcept

  Returns a single element for a particular batch entry.
- ValueType at (size_type batch, size_type idx) const noexcept

  Returns a single element for a particular batch.
- void scale (const BatchLinOp ∗alpha)

  Scales the matrix with a scalar (aka: BLAS scal).
- void add_scaled (const BatchLinOp ∗alpha, const BatchLinOp ∗b)

  Adds `b` scaled by `alpha` to the matrix (aka: BLAS axpy).
- void compute_dot (const BatchLinOp ∗b, BatchLinOp ∗result) const

  Computes the column-wise dot product of each matrix in this batch and its corresponding entry in `b`.
- void compute_norm2 (BatchLinOp ∗result) const

  Computes the Euclidean (L^2) norm of each matrix in this batch.

Static Public Member Functions

- static std::unique_ptr< BatchDense > create_with_config_of (const BatchDense ∗other)

  Creates a BatchDense matrix with the configuration of another BatchDense matrix.
- static std::unique_ptr< const BatchDense > create_const (std::shared_ptr< const Executor > exec, const batch_dim< 2 > &sizes, gko::detail::ConstArrayView< ValueType > &&values, const batch_stride &strides)

  Creates a constant (immutable) batch dense matrix from a constant array.

41.17.1 Detailed Description

template<typename ValueType = default_precision>
class gko::matrix::BatchDense< ValueType >

BatchDense is a batch matrix format which explicitly stores all values of the matrix in each of the batches.

The values in each of the batches are stored in row-major format (values belonging to the same row appear consecutive in the memory). Optionally, rows can be padded for better memory access.

Template Parameters

ValueType	precision of matrix elements

Note

While this format is not very useful for storing sparse matrices, it is often suitable to store vectors, and sets of vectors.

41.17.2 Member Function Documentation

41.17.2.1 add_scaled()

```
template<typename ValueType = default_precision>
void gko::matrix::BatchDense< ValueType >::add_scaled (
            const BatchLinOp * alpha,
            const BatchLinOp * b )  [inline]
```

Adds `b` scaled by `alpha` to the matrix (aka: BLAS axpy).

Parameters

alpha	If alpha is 1x1 BatchDense matrix, the entire matrix is scaled by alpha. If it is a BatchDense row vector of values, then i-th column of the matrix is scaled with the i-th element of alpha (the number of columns of alpha has to match the number of columns of the matrix).
b	a matrix of the same dimension as this

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.17.2.2 at() [1/4]

```
template<typename ValueType = default_precision>
ValueType gko::matrix::BatchDense< ValueType >::at (
            size_type batch,
            size_type idx ) const  [inline], [noexcept]
```

Returns a single element for a particular batch.

Parameters

batch	the batch index to be queried
row	the row of the requested element
col	the column of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.17.2.3 at() [2/4]

```
template<typename ValueType = default_precision>
ValueType& gko::matrix::BatchDense< ValueType >::at (
            size_type batch,
            size_type idx )  [inline], [noexcept]
```

Returns a single element for a particular batch entry.

Useful for iterating across all elements of the matrix. However, it is less efficient than the two-parameter variant of this method.

Parameters

batch	the batch index to be queried
idx	a linear index of the requested element (ignoring the stride)

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_data().

41.17.2.4 at() [3/4]

```
template<typename ValueType = default_precision>
value_type gko::matrix::BatchDense< ValueType >::at (
            size_type batch,
            size_type row,
            size_type col ) const  [inline], [noexcept]
```

Returns a single element for a particular batch.

Parameters

batch	the batch index to be queried
row	the row of the requested element
col	the column of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.17.2.5 at() [4/4]

```
template<typename ValueType = default_precision>
value_type& gko::matrix::BatchDense< ValueType >::at (
            size_type batch,
            size_type row,
            size_type col )  [inline], [noexcept]
```

Returns a single element for a particular batch.

Parameters

batch	the batch index to be queried
row	the row of the requested element
col	the column of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_data().

Referenced by gko::matrix::BatchDense< ValueType >::unbatch().

41.17.2.6 compute_dot()

```
template<typename ValueType = default_precision>
void gko::matrix::BatchDense< ValueType >::compute_dot (
            const BatchLinOp * b,
            BatchLinOp * result ) const  [inline]
```

Computes the column-wise dot product of each matrix in this batch and its corresponding entry in `b`.

If the matrix has complex value_type, then the conjugate of this is taken.

Parameters

b	a BatchDense matrix of same dimension as this
result	a BatchDense row vector, used to store the dot product (the number of column in the vector must match the number of columns of this)

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.17.2.7 compute_norm2()

```
template<typename ValueType = default_precision>
void gko::matrix::BatchDense< ValueType >::compute_norm2 (
            BatchLinOp * result ) const  [inline]
```

Computes the Euclidean (L^2) norm of each matrix in this batch.

Parameters

result	a BatchDense row vector, used to store the norm (the number of columns in the vector must match the number of columns of this)

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.17.2.8 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::matrix::BatchDense< ValueType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.17.2.9 create_const()

```
template<typename ValueType = default_precision>
static std::unique_ptr<const BatchDense> gko::matrix::BatchDense< ValueType >::create_const (
            std::shared_ptr< const Executor > exec,
            const batch_dim< 2 > & sizes,
            gko::detail::ConstArrayView< ValueType > && values,
            const batch_stride & strides )  [inline], [static]
```

Creates a constant (immutable) batch dense matrix from a constant array.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
stride	the row-stride of the matrix

Returns

A smart pointer to the constant matrix wrapping the input array (if it resides on the same executor as the matrix) or a copy of the array on the correct executor.

41.17.2.10 create_with_config_of()

```
template<typename ValueType = default_precision>
static std::unique_ptr<BatchDense> gko::matrix::BatchDense< ValueType >::create_with_config↩
_of (
            const BatchDense< ValueType > * other )  [inline], [static]
```

Creates a BatchDense matrix with the configuration of another BatchDense matrix.

Parameters

other	The other matrix whose configuration needs to copied.

41.17.2.11 get_const_values() [1/2]

```
template<typename ValueType = default_precision>
const value_type* gko::matrix::BatchDense< ValueType >::get_const_values ( ) const  [inline],
[noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchDense< ValueType >::unbatch().

41.17.2.12 get_const_values() [2/2]

```
template<typename ValueType = default_precision>
const value_type* gko::matrix::BatchDense< ValueType >::get_const_values (
            size_type batch ) const  [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.17.2.13 get_num_stored_elements() [1/2]

```
template<typename ValueType = default_precision>
size_type gko::matrix::BatchDense< ValueType >::get_num_stored_elements ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the batch matrix, cumulative across all the batches.

Returns

the number of elements explicitly stored in the matrix, cumulative across all the batches

References gko::Array< ValueType >::get_num_elems().

41.17.2.14 get_num_stored_elements() [2/2]

```
template<typename ValueType = default_precision>
size_type gko::matrix::BatchDense< ValueType >::get_num_stored_elements (
            size_type batch ) const [inline], [noexcept]
```

Returns the number of elements explicitly stored at a specific batch index.

Parameters

batch	the batch index to be queried

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_const_data().

41.17.2.15 get_stride()

```
template<typename ValueType = default_precision>
const batch_stride& gko::matrix::BatchDense< ValueType >::get_stride ( ) const [inline],
[noexcept]
```

Returns the batch_stride of the matrix.

Returns

the batch_stride of the matrix.

Referenced by gko::matrix::BatchDense< ValueType >::unbatch().

41.17.2.16 get_values() [1/2]

```
template<typename ValueType = default_precision>
value_type* gko::matrix::BatchDense< ValueType >::get_values ( ) [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

References gko::Array< ValueType >::get_data().

41.17.2.17 get_values() [2/2]

```
template<typename ValueType = default_precision>
value_type* gko::matrix::BatchDense< ValueType >::get_values (
            size_type batch ) [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

References gko::Array< ValueType >::get_const_data(), and gko::Array< ValueType >::get_data().

41.17.2.18 scale()

```
template<typename ValueType = default_precision>
void gko::matrix::BatchDense< ValueType >::scale (
            const BatchLinOp * alpha ) [inline]
```

Scales the matrix with a scalar (aka: BLAS scal).

Parameters

alpha	If alpha is 1x1 BatchDense matrix, the entire matrix (all batches) is scaled by alpha. If it is a BatchDense row vector of values, then i-th column of the matrix is scaled with the i-th element of alpha (the number of columns of alpha has to match the number of columns of the matrix).

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.17.2.19 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::matrix::BatchDense< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

41.17.2.20 unbatch()

```
template<typename ValueType = default_precision>
std::vector<std::unique_ptr<unbatch_type> > gko::matrix::BatchDense< ValueType >::unbatch (
) const [inline]
```

Unbatches the batched dense and creates a std::vector of Dense matrices.

Returns

a std::vector containing the Dense matrices.

References gko::matrix::BatchDense< ValueType >::at(), gko::Array< ValueType >::get_const_data(), gko::matrix::BatchDense< ValueType >::get_const_values(), gko::PolymorphicObject::get_executor(), and gko::matrix::BatchDense< ValueType >::get_stride().

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

- ginkgo/core/matrix/batch_csr.hpp
- ginkgo/core/matrix/batch_dense.hpp

41.18 gko::matrix::BatchDiagonal< ValueType > Class Template Reference

BatchDiagonal is a batch matrix format which explicitly stores all values of the matrix in each of the batches.

```
#include <ginkgo/core/matrix/batch_diagonal.hpp>
```

Public Member Functions

- void read (const std::vector< mat_data > &data) override
  Read from a COO-type matrix data object into this batch diagonal matrix.
- void read (const std::vector< mat_data32 > &data) override
  Read from a COO-type matrix data object into this batch diagonal matrix.
- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- std::vector< std::unique_ptr< unbatch_type > > unbatch () const
  Unbatches the batched dense and creates a std::vector of Diagonal matrices.
- value_type ∗ get_values () noexcept
  Returns a pointer to the array of values of the matrix.
- value_type ∗ get_values (size_type batch) noexcept
  Returns a pointer to the array of values of the matrix.
- const value_type ∗ get_const_values () const noexcept
  Returns a pointer to the array of values of the matrix.
- const value_type ∗ get_const_values (size_type batch) const noexcept
  Returns a pointer to the array of values of the matrix.
- size_type get_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the batch matrix, cumulative across all the batches.
- size_type get_num_stored_elements (size_type batch) const noexcept
  Returns the number of elements explicitly stored at a specific batch index.
- ValueType & at (size_type batch, size_type idx) noexcept
  Returns a single element for a particular batch entry.
- ValueType at (size_type batch, size_type idx) const noexcept
  Returns a single element for a particular batch entry.

Static Public Member Functions

- static std::unique_ptr< BatchDiagonal > create_with_config_of (const BatchDiagonal ∗other)
  Creates a BatchDiagonal matrix with the configuration of another BatchDiagonal matrix.

41.18.1 Detailed Description

template<typename ValueType = default_precision>
class gko::matrix::BatchDiagonal< ValueType >

BatchDiagonal is a batch matrix format which explicitly stores all values of the matrix in each of the batches.

The values in each of the batches are stored in row-major format (values belonging to the same row appear consecutive in the memory). Optionally, rows can be padded for better memory access.

Template Parameters

ValueType	precision of matrix elements

Note

While this format is not very useful for storing sparse matrices, it is often suitable to store vectors, and sets of vectors.

41.18.2 Member Function Documentation

41.18.2.1 at() [1/2]

```
template<typename ValueType = default_precision>
ValueType gko::matrix::BatchDiagonal< ValueType >::at (
            size_type batch,
            size_type idx ) const [inline], [noexcept]
```

Returns a single element for a particular batch entry.

Parameters

batch	the batch index to be queried
idx	a linear index of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the host results in a runtime error)

This method may not be the fastest way to access matrix entries.

See also

get_values, get_const_values

References gko::Array< ValueType >::get_const_data().

41.18.2.2 at() [2/2]

```
template<typename ValueType = default_precision>
ValueType& gko::matrix::BatchDiagonal< ValueType >::at (
            size_type batch,
            size_type idx ) [inline], [noexcept]
```

Returns a single element for a particular batch entry.

Parameters

batch	the batch index to be queried
idx	a linear index of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the host results in a runtime error)

This method may not be the fastest way to access matrix entries.

See also

get_values, get_const_values

References gko::Array< ValueType >::get_data().

Referenced by gko::matrix::BatchDiagonal< ValueType >::get_const_values(), gko::matrix::BatchDiagonal< ValueType >::get_num_stored_elements(), gko::matrix::BatchDiagonal< ValueType >::get_values(), and gko::matrix::BatchDiagonal< ValueType >::unbatch().

41.18.2.3 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::matrix::BatchDiagonal< ValueType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.18.2.4 create_with_config_of()

```
template<typename ValueType = default_precision>
static std::unique_ptr<BatchDiagonal> gko::matrix::BatchDiagonal< ValueType >::create_with_config_of (
            const BatchDiagonal< ValueType > * other )  [inline], [static]
```

Creates a BatchDiagonal matrix with the configuration of another BatchDiagonal matrix.

Parameters

other	The other matrix whose configuration needs to copied.

41.18.2.5 get_const_values() [1/2]

```
template<typename ValueType = default_precision>
const value_type* gko::matrix::BatchDiagonal< ValueType >::get_const_values ( ) const [inline],
[noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchDiagonal< ValueType >::unbatch().

41.18.2.6 get_const_values() [2/2]

```
template<typename ValueType = default_precision>
const value_type* gko::matrix::BatchDiagonal< ValueType >::get_const_values (
            size_type batch ) const [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::matrix::BatchDiagonal< ValueType >::at(), and gko::Array< ValueType >::get_const_data().

41.18.2.7 get_num_stored_elements() [1/2]

```
template<typename ValueType = default_precision>
size_type gko::matrix::BatchDiagonal< ValueType >::get_num_stored_elements ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the batch matrix, cumulative across all the batches.

Returns

the number of elements explicitly stored in the matrix, cumulative across all the batches

References gko::Array< ValueType >::get_num_elems().

41.18.2.8 get_num_stored_elements() [2/2]

```
template<typename ValueType = default_precision>
size_type gko::matrix::BatchDiagonal< ValueType >::get_num_stored_elements (
            size_type batch ) const  [inline], [noexcept]
```

Returns the number of elements explicitly stored at a specific batch index.

Parameters

batch	the batch index to be queried

Returns

the number of elements explicitly stored in the matrix

References gko::matrix::BatchDiagonal< ValueType >::at(), and gko::Array< ValueType >::get_const_data().

41.18.2.9 get_values() [1/2]

```
template<typename ValueType = default_precision>
value_type* gko::matrix::BatchDiagonal< ValueType >::get_values ( )  [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

References gko::Array< ValueType >::get_data().

41.18.2.10 get_values() [2/2]

```
template<typename ValueType = default_precision>
value_type* gko::matrix::BatchDiagonal< ValueType >::get_values (
            size_type batch )  [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

References gko::matrix::BatchDiagonal< ValueType >::at(), gko::Array< ValueType >::get_const_data(), and gko::Array< ValueType >::get_data().

41.18.2.11 read() [1/2]

```
template<typename ValueType = default_precision>
void gko::matrix::BatchDiagonal< ValueType >::read (
            const std::vector< mat_data > & data ) [override]
```

Read from a COO-type matrix data object into this batch diagonal matrix.

Any off-diagonal entries in the input are ignored.

41.18.2.12 read() [2/2]

```
template<typename ValueType = default_precision>
void gko::matrix::BatchDiagonal< ValueType >::read (
            const std::vector< mat_data32 > & data ) [override]
```

Read from a COO-type matrix data object into this batch diagonal matrix.

Any off-diagonal entries in the input are ignored.

41.18.2.13 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::matrix::BatchDiagonal< ValueType >::transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

41.18.2.14 unbatch()

```
template<typename ValueType = default_precision>
std::vector<std::unique_ptr<unbatch_type> > gko::matrix::BatchDiagonal< ValueType >::unbatch
( ) const [inline]
```

Unbatches the batched dense and creates a std::vector of Diagonal matrices.

Returns

a std::vector containing the Diagonal matrices.

References gko::matrix::BatchDiagonal< ValueType >::at(), gko::Array< ValueType >::get_const_data(), gko::matrix::BatchDiagonal< ValueType >::get_const_values(), and gko::PolymorphicObject::get_executor().

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

- ginkgo/core/matrix/batch_dense.hpp
- ginkgo/core/matrix/batch_diagonal.hpp

41.19 gko::solver::BatchDirect< ValueType > Class Template Reference

Solves a batch of linear systems using a vendor-provided dense batched direct solver.

```
#include <ginkgo/core/solver/batch_direct.hpp>
```

Public Member Functions

- std::shared_ptr< const BatchLinOp > get_system_matrix () const
  Returns the system operator (matrix) of the linear system.
- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.

41.19.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::BatchDirect< ValueType >

Solves a batch of linear systems using a vendor-provided dense batched direct solver.

Template Parameters

ValueType	precision of matrix elements

41.19.2 Member Function Documentation

41.19.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchDirect< ValueType >::apply_uses_initial_guess ( ) const [inline],
[override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
88 { return false; }
```

41.19.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchDirect< ValueType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.19.2.3 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const BatchLinOp> gko::solver::BatchDirect< ValueType >::get_system_matrix (
) const [inline]
```

Returns the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.19.2.4 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchDirect< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

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

- ginkgo/core/solver/batch_direct.hpp

41.20 gko::matrix::BatchEll< ValueType, IndexType > Class Template Reference

BatchEll is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).

`#include <ginkgo/core/matrix/batch_ell.hpp>`

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- std::vector< std::unique_ptr< unbatch_type > > unbatch () const
  Unbatches the BatchEll matrix into distinct matrices of Ell type.
- value_type * get_values () noexcept
  Returns the values of the matrix.
- const value_type * get_const_values () const noexcept
  Returns the values of the matrix.
- index_type * get_col_idxs () noexcept
  Returns the column indexes of the matrix.
- const index_type * get_const_col_idxs () const noexcept
  Returns the column indexes of the matrix.
- const batch_num_stored_elems_per_row & get_num_stored_elements_per_row () const noexcept
  Returns the number of stored elements per row.
- const batch_stride & get_stride () const noexcept
  Returns the number of elements explicitly stored in the matrix, cumulative over all the batches.
- size_type get_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the matrix, cumulative over all the batches.
- value_type & val_at (size_type id, size_type row, size_type idx) noexcept
  Returns the `idx`-th non-zero element of the `row`-th row .
- value_type val_at (size_type id, size_type row, size_type idx) const noexcept
  Returns the `idx`-th non-zero element of the `row`-th row .
- index_type & col_at (size_type row, size_type idx) noexcept
  Returns the `idx`-th column index of the `row`-th row .
- index_type col_at (size_type row, size_type idx) const noexcept
  Returns the `idx`-th column index of the `row`-th row .

Static Public Member Functions

- static std::unique_ptr< const BatchEll > create_const (std::shared_ptr< const Executor > exec, const batch_dim< 2 > &size, const batch_stride &num_stored_elems_per_row, const batch_stride &stride, gko↩ ::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs)
  Creates an immutable BatchEll matrix from a set of constant arrays.

41.20.1 Detailed Description

**template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::BatchEll< ValueType, IndexType >**

BatchEll is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).

Each of the individual batches are stored in a CSR matrix.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

Note

Currently, BatchEll can store matrices with batch entries that have the same sparsity pattern, but different values.

41.20.2 Member Function Documentation

41.20.2.1 col_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type gko::matrix::BatchEll< ValueType, IndexType >::col_at (
            size_type row,
            size_type idx ) const  [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

```
{
    return this->get_const_col_idxs()[this->linearize_index(0, row, idx)];
}
```

References gko::matrix::BatchEll< ValueType, IndexType >::get_const_col_idxs().

41.20.2.2 col_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type& gko::matrix::BatchEll< ValueType, IndexType >::col_at (
            size_type row,
            size_type idx )  [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::matrix::BatchEll< ValueType, IndexType >::get_col_idxs().

41.20.2.3 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<BatchLinOp> gko::matrix::BatchEll< ValueType, IndexType >::conj_transpose ( )
const  [override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.20.2.4 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const BatchEll> gko::matrix::BatchEll< ValueType, IndexType >::create↩
_const (
            std::shared_ptr< const Executor > exec,
            const batch_dim< 2 > & size,
            const batch_stride & num_stored_elems_per_row,
            const batch_stride & stride,
            gko::detail::ConstArrayView< ValueType > && values,
            gko::detail::ConstArrayView< IndexType > && col_idxs )  [inline], [static]
```

Creates an immutable BatchEll matrix from a set of constant arrays.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
col_idxs	the column index array of the matrix
num_stored_elems_per_row	the number of stored nonzeros per row
stride	the column-stride of the value and column index array

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of the arrays on the correct executor.

41.20.2.5 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::BatchEll< ValueType, IndexType >::get_col_idxs ( )  [inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

References gko::Array< ValueType >::get_data().

Referenced by gko::matrix::BatchEll< ValueType, IndexType >::col_at().

41.20.2.6 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::BatchEll< ValueType, IndexType >::get_const_col_idxs ( ) const
[inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchEll< ValueType, IndexType >::col_at(), and gko::matrix::BatchEll< ValueType, IndexType >::unbatch().

41.20.2.7 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::BatchEll< ValueType, IndexType >::get_const_values ( ) const
[inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::BatchEll< ValueType, IndexType >::unbatch().

41.20.2.8 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::BatchEll< ValueType, IndexType >::get_num_stored_elements ( ) const
[inline], [noexcept]
```

Returns the number of elements explicitly stored in the matrix, cumulative over all the batches.

Returns

the number of elements explicitly stored in the matrix, cumulative over all the batches

References gko::Array< ValueType >::get_num_elems().

Referenced by gko::matrix::BatchEll< ValueType, IndexType >::unbatch().

41.20.2.9 get_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const batch_num_stored_elems_per_row& gko::matrix::BatchEll< ValueType, IndexType >::get_num←
_stored_elements_per_row ( ) const [inline], [noexcept]
```

Returns the number of stored elements per row.

Returns

the number of stored elements per row.

Referenced by gko::matrix::BatchEll< ValueType, IndexType >::unbatch().

41.20.2.10 get_stride()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const batch_stride& gko::matrix::BatchEll< ValueType, IndexType >::get_stride ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the matrix, cumulative over all the batches.

Returns

the number of elements explicitly stored in the matrix, cumulative over all the batches

Referenced by gko::matrix::BatchEll< ValueType, IndexType >::unbatch().

41.20.2.11 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::BatchEll< ValueType, IndexType >::get_values ( ) [inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

References gko::Array< ValueType >::get_data().

41.20.2.12 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<BatchLinOp> gko::matrix::BatchEll< ValueType, IndexType >::transpose ( )
const [override], [virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

41.20.2.13 unbatch()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::vector<std::unique_ptr<unbatch_type> > gko::matrix::BatchEll< ValueType, IndexType >↩
::unbatch ( ) const [inline]
```

Unbatches the BatchEll matrix into distinct matrices of Ell type.

Returns

a std::vector containing the distinct Ell matrices.

References gko::matrix::BatchEll< ValueType, IndexType >::get_const_col_idxs(), gko::matrix::BatchEll< Value↩Type, IndexType >::get_const_values(), gko::PolymorphicObject::get_executor(), gko::matrix::BatchEll< Value↩Type, IndexType >::get_num_stored_elements(), gko::matrix::BatchEll< ValueType, IndexType >::get_num_↩stored_elements_per_row(), and gko::matrix::BatchEll< ValueType, IndexType >::get_stride().

41.20.2.14 val_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type gko::matrix::BatchEll< ValueType, IndexType >::val_at (
            size_type id,
            size_type row,
            size_type idx ) const [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.20.2.15 val_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type& gko::matrix::BatchEll< ValueType, IndexType >::val_at (
            size_type id,
            size_type row,
            size_type idx ) [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_data().

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

- ginkgo/core/matrix/batch_ell.hpp

41.21 gko::solver::BatchGmres< ValueType > Class Template Reference

GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

```
#include <ginkgo/core/solver/batch_gmres.hpp>
```

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override

  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override

  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- bool apply_uses_initial_guess () const override

  Return true as iterative solvers use the data in x as an initial guess.
- int get_restart_number () const

  Gets the restart parameter of the solver.
- void set_restart_number (const int &other)

  Sets the restart paramter.

41.21.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::BatchGmres< ValueType >

GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

This solver solves a batch of linear systems using GMRES algorithm.

Unless otherwise specified via the `preconditioner` factory parameter, this implementation does not use any preconditioner by default. The type of tolerance(absolute or relative) and the maximum number of iterations to be used in the stopping criterion can be set via the factory parameters.

Template Parameters

ValueType	precision of matrix elements

41.21.2 Member Function Documentation

41.21.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchGmres< ValueType >::apply_uses_initial_guess ( ) const  [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
90 { return true; }
```

41.21.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchGmres< ValueType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.21.2.3 get_restart_number()

```
template<typename ValueType = default_precision>
int gko::solver::BatchGmres< ValueType >::get_restart_number ( ) const  [inline]
```

Gets the restart parameter of the solver.

Returns

the restart number

41.21.2.4 set_restart_number()

```
template<typename ValueType = default_precision>
void gko::solver::BatchGmres< ValueType >::set_restart_number (
            const int & other )  [inline]
```

Sets the restart paramter.

Parameters

other	the new restart number

41.21.2.5 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchGmres< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

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

- ginkgo/core/solver/batch_gmres.hpp

41.22 gko::solver::BatchIdr< ValueType > Class Template Reference

IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.

```
#include <ginkgo/core/solver/batch_idr.hpp>
```

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override
  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.
- size_type get_subspace_dim () const
  Gets the subspace dimension of the solver.
- void set_subspace_dim (const size_type other)
  Sets the subspace dimension of the solver.
- real_type get_kappa () const
  Gets the kappa parameter of the solver.
- void set_kappa (const real_type other)
  Sets the kappa parameter of the solver.
- bool get_complex_subspace () const
  Gets the complex_subspace parameter of the solver.

- void set_complex_subspace (const bool other)

  Sets the complex_subspace parameter of the solver.
- bool get_deterministic () const

  Gets the deterministic parameter of the solver.
- void set_deterministic (const bool other)

  Sets the deterministic parameter of the solver.
- bool get_smoothing () const

  Gets the smoothing paramter of solver.
- void set_smoothing (const bool other)

  Sets the smoothing paramter of solver.

41.22.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::BatchIdr< ValueType >

IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.

The implemented version is the one presented in the paper "Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties" by M. B. Van Gijzen and P. Sonneveld.

The method is based on the induced dimension reduction theorem which provides a way to construct subsequent residuals that lie in a sequence of shrinking subspaces. These subspaces are spanned by s vectors which are first generated randomly and then orthonormalized. They are stored in a dense matrix.

This solver solves a batch of linear systems using the IDR(s) algorithm.

Unless otherwise specified via the `preconditioner` factory parameter, this implementation does not use any preconditioner by default. The type of tolerance(absolute or relative) and the maximum number of iterations to be used in the stopping criterion can be set via the factory parameters.

Template Parameters

ValueType	precision of matrix elements

41.22.2 Member Function Documentation

41.22.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchIdr< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
96 { return true; }
```

41.22.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchIdr< ValueType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.22.2.3 get_complex_subspace()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchIdr< ValueType >::get_complex_subspace ( ) const [inline]
```

Gets the complex_subspace parameter of the solver.

Returns

the complex_subspace parameter

41.22.2.4 get_deterministic()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchIdr< ValueType >::get_deterministic ( ) const [inline]
```

Gets the deterministic parameter of the solver.

Returns

the deterministic parameter

41.22.2.5 get_kappa()

```
template<typename ValueType = default_precision>
real_type gko::solver::BatchIdr< ValueType >::get_kappa ( ) const [inline]
```

Gets the kappa parameter of the solver.

Returns

the kappa parameter

41.22.2.6 get_smoothing()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchIdr< ValueType >::get_smoothing ( ) const  [inline]
```

Gets the smoothing paramter of solver.

Returns

the smoothing paramter

41.22.2.7 get_subspace_dim()

```
template<typename ValueType = default_precision>
size_type gko::solver::BatchIdr< ValueType >::get_subspace_dim ( ) const  [inline]
```

Gets the subspace dimension of the solver.

Returns

the subspace Dimension

41.22.2.8 set_complex_subspace()

```
template<typename ValueType = default_precision>
void gko::solver::BatchIdr< ValueType >::set_complex_subspace (
            const bool other )  [inline]
```

Sets the complex_subspace parameter of the solver.

Parameters

other	the new complex_subspace parameter

41.22.2.9 set_deterministic()

```
template<typename ValueType = default_precision>
void gko::solver::BatchIdr< ValueType >::set_deterministic (
            const bool other )  [inline]
```

Sets the deterministic parameter of the solver.

Parameters

other	the new deterministic parameter

41.22.2.10 set_kappa()

```
template<typename ValueType = default_precision>
void gko::solver::BatchIdr< ValueType >::set_kappa (
            const real_type other ) [inline]
```

Sets the kappa parameter of the solver.

Parameters

other	the new kappa parameter

41.22.2.11 set_smoothing()

```
template<typename ValueType = default_precision>
void gko::solver::BatchIdr< ValueType >::set_smoothing (
            const bool other ) [inline]
```

Sets the smoothing paramter of solver.

Parameters

other	the new smoothing paramter

41.22.2.12 set_subspace_dim()

```
template<typename ValueType = default_precision>
void gko::solver::BatchIdr< ValueType >::set_subspace_dim (
            const size_type other ) [inline]
```

Sets the subspace dimension of the solver.

Parameters

other	the new subspace Dimension

41.22.2.13 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchIdr< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

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

- ginkgo/core/solver/batch_idr.hpp

41.23 gko::BatchLinOpFactory Class Reference

A BatchLinOpFactory represents a higher order mapping which transforms one batch linear operator into another.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Additional Inherited Members

41.23.1 Detailed Description

A BatchLinOpFactory represents a higher order mapping which transforms one batch linear operator into another.

In a similar fashion to LinOps, BatchLinOps are also "generated" from the BatchLinOpFactory. A function of this class is to provide a generate method, which internally cals the generate_impl(), which the concrete BatchLinOps have to implement.

41.23.1.1 Example: using BatchCG in Ginkgo

```
{c++}
// Suppose A is a batch matrix, batch_b a batch rhs vector, and batch_x an
// initial guess
// Create a BatchCG which runs for at most 1000 iterations, and stops after
// reducing the residual norm by 6 orders of magnitude
auto batch_cg_factory = solver::BatchCg<>::build()
    .with_max_iters(1000)
    .with_rel_residual_goal(1e-6)
    .on(cuda);
// create a batch linear operator which represents the solver
auto batch_cg = batch_cg_factory->generate(A);
// solve the system
batch_cg->apply(gko::lend(batch_b), gko::lend(batch_x));
```

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

- ginkgo/core/base/batch_lin_op.hpp

41.24 gko::BatchReadableFromMatrixData< ValueType, IndexType > Class Template Reference

A BatchLinOp implementing this interface can read its data from a matrix_data structure.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Public Member Functions

- virtual void read (const std::vector< matrix_data< ValueType, IndexType >> &data)=0

  Reads a batch matrix from a std::vector of matrix_data objects.
- void read (const std::vector< matrix_assembly_data< ValueType, IndexType >> &assembly_data)

  Reads a matrix from a std::vector of matrix_assembly_data objects.

41.24.1 Detailed Description

template<typename ValueType, typename IndexType>
class gko::BatchReadableFromMatrixData< ValueType, IndexType >

A BatchLinOp implementing this interface can read its data from a matrix_data structure.

41.24.2 Member Function Documentation

41.24.2.1 read() [1/2]

```
template<typename ValueType, typename IndexType>
void gko::BatchReadableFromMatrixData< ValueType, IndexType >::read (
            const std::vector< matrix_assembly_data< ValueType, IndexType >> & assembly_data
) [inline]
```

Reads a matrix from a std::vector of matrix_assembly_data objects.

Parameters

data	the std::vector of matrix_assembly_data objects

```
{
    auto mat_data = std::vector<matrix_data<ValueType, IndexType»(
        assembly_data.size());
    size_type ind = 0;
    for (const auto& i : assembly_data) {
        mat_data[ind] = i.get_ordered_data();
        ++ind;
    }
    this->read(mat_data);
}
```

41.24.2.2 read() [2/2]

```
template<typename ValueType, typename IndexType>
virtual void gko::BatchReadableFromMatrixData< ValueType, IndexType >::read (
            const std::vector< matrix_data< ValueType, IndexType >> & data ) [pure virtual]
```

Reads a batch matrix from a std::vector of matrix_data objects.

Parameters

data	the std::vector of matrix_data objects

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

- ginkgo/core/base/batch_lin_op.hpp

41.25 gko::solver::BatchRichardson< ValueType > Class Template Reference

The (preconditioned) Richardson solver is an iterative method that uses a preconditioner to approximate the error of the current solution via the current (preconditioned) residual.

`#include <ginkgo/core/solver/batch_richardson.hpp>`

Public Member Functions

- std::unique_ptr< BatchLinOp > transpose () const override
    Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- std::unique_ptr< BatchLinOp > conj_transpose () const override
    Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.
- bool apply_uses_initial_guess () const override
    Return true as iterative solvers use the data in x as an initial guess.

41.25.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::BatchRichardson< ValueType >

The (preconditioned) Richardson solver is an iterative method that uses a preconditioner to approximate the error of the current solution via the current (preconditioned) residual.

This solver applies the Richardson iteration to a batch of linear systems.

```
solution = initial_guess
while not converged:
    residual = b - A solution
    error = preconditioner(A, residual)
    solution = solution + relaxation_factor * error
```

Unless otherwise specified via the `preconditioner` factory parameter, this implementation uses the Jacobi preconditioner as the default preconditioner. The only stopping criterion currently available is controlled by the `max_iterations` and `rel_residual_tol` factory parameters. The solver is stopped whrn the maximum iterations are reached, or the relative residual is smaller than the specified tolerance.

See also

Ir

Template Parameters

ValueType	precision of matrix elements

41.25.2 Member Function Documentation

41.25.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::BatchRichardson< ValueType >::apply_uses_initial_guess ( ) const  [inline],
[override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
102 { return true; }
```

41.25.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchRichardson< ValueType >::conj_transpose ( )
const  [override], [virtual]
```

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::BatchTransposable.

41.25.2.3 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<BatchLinOp> gko::solver::BatchRichardson< ValueType >::transpose ( ) const
[override], [virtual]
```

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implements gko::BatchTransposable.

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

- ginkgo/core/solver/batch_richardson.hpp

41.26 gko::BatchScalable Class Reference

A batch made up of entities that can scaled from the left and right by scaling vectors.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Public Member Functions

- virtual void batch_scale (const BatchLinOp ∗left_scale, const BatchLinOp ∗right_scale)=0
  Scales each matrix in a batch from the left and right.

41.26.1 Detailed Description

A batch made up of entities that can scaled from the left and right by scaling vectors.

41.26.2 Member Function Documentation

41.26.2.1 batch_scale()

```
virtual void gko::BatchScalable::batch_scale (
            const BatchLinOp * left_scale,
            const BatchLinOp * right_scale )  [pure virtual]
```

Scales each matrix in a batch from the left and right.

Parameters

left_scale	The left scaling batch vector. In case of matrices, this scales the rows.
right_scale	The right scaling batch vector. In case of matrices, this scales the columns.

Implemented in gko::EnableBatchScaling.

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

- ginkgo/core/base/batch_lin_op.hpp

41.27 gko::BatchScaledIdentityAddable Class Reference

Adds the operation M <- a I + b M for batch matrix M, batch identity operator I and batch scalars a and b.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Public Member Functions

- virtual void add_scaled_identity (const BatchLinOp ∗a, const BatchLinOp ∗b)

  Scales this and adds another scalar times the identity to it.

41.27.1 Detailed Description

Adds the operation M <- a I + b M for batch matrix M, batch identity operator I and batch scalars a and b.

M is the calling object.

41.27.2 Member Function Documentation

41.27.2.1 add_scaled_identity()

```
virtual void gko::BatchScaledIdentityAddable::add_scaled_identity (
            const BatchLinOp * a,
            const BatchLinOp * b ) [inline], [virtual]
```

Scales this and adds another scalar times the identity to it.

Parameters

a	Scalar to multiply the identity operator by before adding.
b	Scalar to multiply this before adding the scaled identity to it.

References gko::make_temporary_clone().

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

- ginkgo/core/base/batch_lin_op.hpp

41.28 gko::BatchTransposable Class Reference

Batch Linear operators which support transposition of the distinct batch entries should implement the BatchTransposable interface.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Public Member Functions

- virtual std::unique_ptr< BatchLinOp > transpose () const =0

  Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.
- virtual std::unique_ptr< BatchLinOp > conj_transpose () const =0

  Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

41.28.1 Detailed Description

Batch Linear operators which support transposition of the distinct batch entries should implement the BatchTransposable interface.

It provides two functionalities, the normal transpose and the conjugate transpose, both transposing the invidual batch entries.

The normal transpose returns the transpose of the linear operator without changing any of its elements representing the operation, $B = A^T$.

The conjugate transpose returns the conjugate of each of the elements and additionally transposes the linear operator representing the operation, $B = A^H$.

41.28.1.1 Example: Transposing a BatchCsr matrix:

```
 {c++}
//Transposing an object of BatchLinOp type.
//The object you want to transpose.
auto op = matrix::BatchCsr::create(exec);
//Transpose the object by first converting it to a transposable type.
auto trans = op->transpose();
```

41.28.2 Member Function Documentation

41.28.2.1 conj_transpose()

`virtual std::unique_ptr<BatchLinOp> gko::BatchTransposable::conj_transpose ( ) const [pure virtual]`

Returns a BatchLinOp containing the conjugate transpose of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new conjugate transposed object

Implemented in gko::matrix::BatchDense< ValueType >, gko::matrix::BatchDiagonal< ValueType >, gko::matrix::BatchEll< ValueTyp gko::matrix::BatchCsr< ValueType, IndexType >, gko::solver::BatchRichardson< ValueType >, gko::solver::BatchIdr< ValueType >, gko::solver::BatchCg< ValueType >, gko::solver::BatchBicgstab< ValueType >, gko::solver::BatchGmres< ValueType >, and gko::solver::BatchDirect< ValueType >.

41.28.2.2 transpose()

`virtual std::unique_ptr<BatchLinOp> gko::BatchTransposable::transpose ( ) const [pure virtual]`

Returns a BatchLinOp containing the transposes of the distinct entries of the BatchTransposable object.

Returns

a pointer to the new transposed object

Implemented in gko::matrix::BatchDense< ValueType >, gko::matrix::BatchDiagonal< ValueType >, gko::matrix::BatchEll< ValueTyp gko::matrix::BatchCsr< ValueType, IndexType >, gko::solver::BatchRichardson< ValueType >, gko::solver::BatchIdr< ValueType >, gko::solver::BatchCg< ValueType >, gko::solver::BatchBicgstab< ValueType >, gko::solver::BatchGmres< ValueType >, and gko::solver::BatchDirect< ValueType >.

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

- ginkgo/core/base/batch_lin_op.hpp

41.29 gko::BatchWritableToMatrixData< ValueType, IndexType > Class Template Reference

A BatchLinOp implementing this interface can write its data to a std::vector of matrix_data objects.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Public Member Functions

- virtual void write (std::vector< matrix_data< ValueType, IndexType >> &data) const =0

  Writes a matrix to a matrix_data structure.

41.29.1 Detailed Description

template<typename ValueType, typename IndexType>
class gko::BatchWritableToMatrixData< ValueType, IndexType >

A BatchLinOp implementing this interface can write its data to a std::vector of matrix_data objects.

41.29.2 Member Function Documentation

41.29.2.1 write()

```
template<typename ValueType, typename IndexType>
virtual void gko::BatchWritableToMatrixData< ValueType, IndexType >::write (
            std::vector< matrix_data< ValueType, IndexType >> & data ) const  [pure virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

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

- ginkgo/core/base/batch_lin_op.hpp

41.30 gko::solver::Bicg< ValueType > Class Template Reference

BICG or the Biconjugate gradient method is a Krylov subspace solver.

```
#include <ginkgo/core/solver/bicg.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const

    Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override

    Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

    Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override

    Return true as iterative solvers use the data in x as an initial guess.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const

    Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)

    Sets the stopping criterion of the solver.

41.30.1 Detailed Description

template< typename ValueType = default_precision>
class gko::solver::Bicg< ValueType >

BICG or the Biconjugate gradient method is a Krylov subspace solver.

Being a generic solver, it is capable of solving general matrices, including non-s.p.d matrices. Though, the memory and the computational requirement of the BiCG solver are higher than of its s.p.d solver counterpart, it has the capability to solve generic systems.

BiCG is based on the bi-Lanczos tridiagonalization method and in exact arithmetic should terminate in at most N iterations (2N MV's, with A and A^H). It forms the basis of many of the cheaper methods such as BiCGSTAB and CGS.

Reference: R.Fletcher, Conjugate gradient methods for indefinite systems, doi: 10.1007/BFb0080116

Template Parameters

ValueType	precision of matrix elements

41.30.2 Member Function Documentation

41.30.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Bicg< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
108 { return true; }
```

41.30.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Bicg< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.30.2.3 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Bicg< ValueType >::get_stop_↩
criterion_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.30.2.4 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Bicg< ValueType >::get_system_matrix ( ) const
[inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.30.2.5 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Bicg< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other ) [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.30.2.6 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Bicg< ValueType >::transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/bicg.hpp

41.31 gko::solver::Bicgstab< ValueType > Class Template Reference

BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.

```
#include <ginkgo/core/solver/bicgstab.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const
  Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const
  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)
  Sets the stopping criterion of the solver.

41.31.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Bicgstab< ValueType >

BiCGSTAB or the Bi-Conjugate Gradient-Stabilized is a Krylov subspace solver.

Being a generic solver, it is capable of solving general matrices, including non-s.p.d matrices. Though, the memory and the computational requirement of the BiCGSTAB solver are higher than of its s.p.d solver counterpart, it has the capability to solve generic systems. It was developed by stabilizing the BiCG method.

Template Parameters

ValueType	precision of the elements of the system matrix.

41.31.2 Member Function Documentation

41.31.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Bicgstab< ValueType >::apply_uses_initial_guess ( ) const  [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
106 { return true; }
```

41.31.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Bicgstab< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.31.2.3 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Bicgstab< ValueType >::get_stop_↩
criterion_factory ( ) const  [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.31.2.4 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Bicgstab< ValueType >::get_system_matrix ( ) const
[inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.31.2.5 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Bicgstab< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other ) [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.31.2.6 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Bicgstab< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/bicgstab.hpp

41.32 gko::preconditioner::block_interleaved_storage_scheme< IndexType > Struct Template Reference

Defines the parameters of the interleaved block storage scheme used by block-Jacobi blocks.

```
#include <ginkgo/core/preconditioner/jacobi.hpp>
```

Public Member Functions

- IndexType get_group_size () const noexcept
  Returns the number of elements in the group.
- size_type compute_storage_space (size_type num_blocks) const noexcept
  Computes the storage space required for the requested number of blocks.
- IndexType get_group_offset (IndexType block_id) const noexcept
  Returns the offset of the group belonging to the block with the given ID.
- IndexType get_block_offset (IndexType block_id) const noexcept
  Returns the offset of the block with the given ID within its group.
- IndexType get_global_block_offset (IndexType block_id) const noexcept
  Returns the offset of the block with the given ID.
- IndexType get_stride () const noexcept
  Returns the stride between columns of the block.

Public Attributes

- IndexType block_offset
  The offset between consecutive blocks within the group.
- IndexType group_offset
  The offset between two block groups.
- uint32 group_power
  Then base 2 power of the group.

41.32.1 Detailed Description

template<typename IndexType>
struct gko::preconditioner::block_interleaved_storage_scheme< IndexType >

Defines the parameters of the interleaved block storage scheme used by block-Jacobi blocks.

Template Parameters

IndexType	type used for storing indices of the matrix

41.32.2 Member Function Documentation

41.32.2.1 compute_storage_space()

```
template<typename IndexType>
size_type gko::preconditioner::block_interleaved_storage_scheme< IndexType >::compute_storage↩
_space (
            size_type num_blocks ) const  [inline], [noexcept]
```

Computes the storage space required for the requested number of blocks.

Parameters

num_blocks	the total number of blocks that needs to be stored

Returns

the total memory (as the number of elements) that need to be allocated for the scheme

Note

To simplify using the method in situations where the number of blocks is not known, for a special input `size↩ _type{} - 1` the method returns `0` to avoid overallocation of memory.

```
{
    return (num_blocks + 1 == size_type{0})
               ? size_type{0}
               : ceildiv(num_blocks, this->get_group_size()) * group_offset;
}
```

41.32.2.2 get_block_offset()

```
template<typename IndexType>
IndexType gko::preconditioner::block_interleaved_storage_scheme< IndexType >::get_block_offset
(
            IndexType block_id ) const  [inline], [noexcept]
```

Returns the offset of the block with the given ID within its group.

Parameters

block↩ _id	the ID of the block

Returns

the offset of the block with ID `block_id` within its group

Referenced by gko::preconditioner::block_interleaved_storage_scheme< index_type >::get_global_block_offset().

41.32.2.3 get_global_block_offset()

```
template<typename IndexType>
IndexType gko::preconditioner::block_interleaved_storage_scheme< IndexType >::get_global_↩
block_offset (
            IndexType block_id ) const  [inline], [noexcept]
```

Returns the offset of the block with the given ID.

Parameters

block↩ _id	the ID of the block

Returns

the offset of the block with ID `block_id`

41.32.2.4 get_group_offset()

```
template<typename IndexType>
IndexType gko::preconditioner::block_interleaved_storage_scheme< IndexType >::get_group_offset
(
            IndexType block_id ) const  [inline], [noexcept]
```

Returns the offset of the group belonging to the block with the given ID.

Parameters

block↩ _id	the ID of the block

Returns

the offset of the group belonging to block with ID `block_id`

Referenced by gko::preconditioner::block_interleaved_storage_scheme< index_type >::get_global_block_offset().

41.32.2.5 get_group_size()

```
template<typename IndexType>
IndexType gko::preconditioner::block_interleaved_storage_scheme< IndexType >::get_group_size (
) const  [inline], [noexcept]
```

Returns the number of elements in the group.

Returns

the number of elements in the group

Referenced by gko::preconditioner::block_interleaved_storage_scheme< index_type >::compute_storage_↩ space(), and gko::preconditioner::block_interleaved_storage_scheme< index_type >::get_block_offset().

41.32.2.6 get_stride()

```
template<typename IndexType>
IndexType gko::preconditioner::block_interleaved_storage_scheme< IndexType >::get_stride ( )
const [inline], [noexcept]
```

Returns the stride between columns of the block.

Returns

stride between columns of the block

41.32.3 Member Data Documentation

41.32.3.1 group_power

```
template<typename IndexType>
uint32 gko::preconditioner::block_interleaved_storage_scheme< IndexType >::group_power
```

Then base 2 power of the group.

I.e. the group contains `1 << group_power` elements.

Referenced by gko::preconditioner::block_interleaved_storage_scheme< index_type >::get_group_offset(), gko↩ ::preconditioner::block_interleaved_storage_scheme< index_type >::get_group_size(), and gko::preconditioner↩ ::block_interleaved_storage_scheme< index_type >::get_stride().

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

- ginkgo/core/preconditioner/jacobi.hpp

41.33 gko::BlockSizeError< IndexType > Class Template Reference

Error that denotes issues between block sizes and matrix dimensions.

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- BlockSizeError (const std::string &file, const int line, const int block_size, const IndexType size)

41.33.1 Detailed Description

template<typename IndexType>
class gko::BlockSizeError< IndexType >

Error that denotes issues between block sizes and matrix dimensions.

Template Parameters

IndexType	Type of index used by the linear algebra object that is incompatible with the requried block size.

41.33.2 Constructor & Destructor Documentation

41.33.2.1 BlockSizeError()

```
template<typename IndexType >
gko::BlockSizeError< IndexType >::BlockSizeError (
            const std::string & file,
            const int line,
            const int block_size,
            const IndexType size )  [inline]
```

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
block_size	Size of small dense blocks in a matrix
size	The size that is not exactly divided by the block size

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

- ginkgo/core/base/exception.hpp

41.34 gko::solver::CbGmres< ValueType > Class Template Reference

CB-GMRES or the compressed basis generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

```
#include <ginkgo/core/solver/cb_gmres.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const

    Gets the system operator (matrix) of the linear system.
- size_type get_krylov_dim () const

    Returns the Krylov dimension.
- void set_krylov_dim (size_type other)

    Sets the Krylov dimension.
- cb_gmres::storage_precision get_storage_precision () const

    Returns the storage precision used internally.

41.34.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::CbGmres< ValueType >

CB-GMRES or the compressed basis generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

The implementation in Ginkgo makes use of the merged kernel to make the best use of data locality. The inner operations in one iteration of CB-GMRES are merged into 2 separate steps. Classical Gram-Schmidt with reorthogonalization is used.

The Krylov basis can be stored in reduced precision (compressed) to reduce memory accesses, while all computations (including Krylov basis operations) are performed in the same arithmetic precision ValueType. By default, the Krylov basis are stored in one precision lower than ValueType.

Template Parameters

ValueType	the arithmetic precision and the precision of matrix elements

41.34.2 Member Function Documentation

41.34.2.1 get_krylov_dim()

```
template<typename ValueType = default_precision>
size_type gko::solver::CbGmres< ValueType >::get_krylov_dim ( ) const  [inline]
```

Returns the Krylov dimension.

Returns

the Krylov dimension

```
145 { return krylov_dim_; }
```

41.34.2.2 get_storage_precision()

```
template<typename ValueType = default_precision>
cb_gmres::storage_precision gko::solver::CbGmres< ValueType >::get_storage_precision ( ) const
[inline]
```

Returns the storage precision used internally.

Returns

the storage precision used internally

41.34.2.3 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::CbGmres< ValueType >::get_system_matrix ( ) const
[inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.34.2.4 set_krylov_dim()

```
template<typename ValueType = default_precision>
void gko::solver::CbGmres< ValueType >::set_krylov_dim (
            size_type other )  [inline]
```

Sets the Krylov dimension.

Parameters

other	the new Krylov dimension

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

- ginkgo/core/solver/cb_gmres.hpp

41.35 gko::solver::Cg< ValueType > Class Template Reference

CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

`#include <ginkgo/core/solver/cg.hpp>`

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const
  Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const
  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)
  Sets the stopping criterion of the solver.

41.35.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Cg< ValueType >

CG or the conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

Though this method performs very well for symmetric positive definite matrices, it is in general not suitable for general matrices.

The implementation in Ginkgo makes use of the merged kernel to make the best use of data locality. The inner operations in one iteration of CG are merged into 2 separate steps.

Template Parameters

ValueType	precision of matrix elements

41.35.2 Member Function Documentation

41.35.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Cg< ValueType >::apply_uses_initial_guess ( ) const  [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
102 { return true; }
```

41.35.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Cg< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.35.2.3 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Cg< ValueType >::get_stop_criterion↩
_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.35.2.4 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Cg< ValueType >::get_system_matrix ( ) const [inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.35.2.5 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Cg< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other ) [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.35.2.6 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Cg< ValueType >::transpose ( ) const [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/cg.hpp

41.36 gko::solver::Cgs< ValueType > Class Template Reference

CGS or the conjugate gradient square method is an iterative type Krylov subspace method which is suitable for general systems.

```
#include <ginkgo/core/solver/cgs.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const

  Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override

  Return true as iterative solvers use the data in x as an initial guess.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const

  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)

  Sets the stopping criterion of the solver.

41.36.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Cgs< ValueType >

CGS or the conjugate gradient square method is an iterative type Krylov subspace method which is suitable for general systems.

The implementation in Ginkgo makes use of the merged kernel to make the best use of data locality. The inner operations in one iteration of CGS are merged into 3 separate steps.

Template Parameters

ValueType	precision of matrix elements

41.36.2 Member Function Documentation

41.36.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Cgs< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
99 { return true; }
```

41.36.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Cgs< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.36.2.3 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Cgs< ValueType >::get_stop_↩
criterion_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.36.2.4 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Cgs< ValueType >::get_system_matrix ( ) const [inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.36.2.5 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Cgs< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other ) [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.36.2.6 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Cgs< ValueType >::transpose ( ) const [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/cgs.hpp

41.37 gko::matrix::Csr< ValueType, IndexType >::classical Class Reference

classical is a strategy_type which uses the same number of threads on each row.

`#include <ginkgo/core/matrix/csr.hpp>`

Public Member Functions

- classical ()
  Creates a classical strategy.
- void process (const Array< index_type > &mtx_row_ptrs, Array< index_type > *mtx_srow) override
  Computes srow according to row pointers.
- int64_t clac_size (const int64_t nnz) override
  Computes the srow size according to the number of nonzeros.
- std::shared_ptr< strategy_type > copy () override
  Copy a strategy.

41.37.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >::classical

classical is a strategy_type which uses the same number of threads on each row.

Classical strategy uses multithreads to calculate on parts of rows and then do a reduction of these threads results. The number of threads per row depends on the max number of stored elements per row.

41.37.2 Member Function Documentation

41.37.2.1 clac_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
int64_t gko::matrix::Csr< ValueType, IndexType >::classical::clac_size (
            const int64_t nnz )  [inline], [override], [virtual]
```

Computes the srow size according to the number of nonzeros.

Parameters

nnz	the number of nonzeros

Returns

the size of srow

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.37.2.2 copy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::classical::copy ( )
[inline], [override], [virtual]
```

Copy a strategy.

This is a workaround until strategies are revamped, since strategies like `automatical` do not work when actually shared.

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.37.2.3 process()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::classical::process (
            const Array< index_type > & mtx_row_ptrs,
            Array< index_type > * mtx_srow ) [inline], [override], [virtual]
```

Computes srow according to row pointers.

Parameters

mtx_row_ptrs	the row pointers of the matrix
mtx_srow	the srow of the matrix

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

References gko::Array< ValueType >::get_const_data(), gko::Array< ValueType >::get_executor(), and gko::↩
Array< ValueType >::get_num_elems().

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

- ginkgo/core/matrix/csr.hpp

41.38 gko::matrix::Hybrid< ValueType, IndexType >::column_limit Class Reference

column_limit is a strategy_type which decides the number of stored elements per row of the ell part by specifying the number of columns.

`#include <ginkgo/core/matrix/hybrid.hpp>`

Public Member Functions

- column_limit (size_type num_column=0)

    Creates a column_limit strategy.
- size_type compute_ell_num_stored_elements_per_row (Array< size_type > *row_nnz) const override

    Computes the number of stored elements per row of the ell part.
- auto get_num_columns () const

    Get the number of columns limit.

41.38.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >::column_limit

column_limit is a strategy_type which decides the number of stored elements per row of the ell part by specifying the number of columns.

41.38.2 Constructor & Destructor Documentation

41.38.2.1 column_limit()

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Hybrid< ValueType, IndexType >::column_limit::column_limit (
            size_type num_column = 0 )  [inline], [explicit]
```

Creates a column_limit strategy.

Parameters

num_column	the specified number of columns of the ell part

41.38.3 Member Function Documentation

41.38.3.1 compute_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::column_limit::compute_ell_num_stored_↩
elements_per_row (
            Array< size_type > * row_nnz ) const  [inline], [override], [virtual]
```

Computes the number of stored elements per row of the ell part.

Parameters

row_nnz	the number of nonzeros of each row

Returns

the number of stored elements per row of the ell part

Implements gko::matrix::Hybrid< ValueType, IndexType >::strategy_type.

41.38.3.2 get_num_columns()

```
template<typename ValueType = default_precision, typename IndexType = int32>
auto gko::matrix::Hybrid< ValueType, IndexType >::column_limit::get_num_columns ( ) const
[inline]
```

Get the number of columns limit.

Returns

the number of columns limit

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

- ginkgo/core/matrix/hybrid.hpp

41.39 gko::Combination< ValueType > Class Template Reference

The Combination class can be used to construct a linear combination of multiple linear operators `c1 * op1 + c2 * op2 + ...`

`#include <ginkgo/core/base/combination.hpp>`

Public Member Functions

- const std::vector< std::shared_ptr< const LinOp > > & get_coefficients () const noexcept
  Returns a list of coefficients of the combination.
- const std::vector< std::shared_ptr< const LinOp > > & get_operators () const noexcept
  Returns a list of operators of the combination.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.39.1 Detailed Description

template<typename ValueType = default_precision>
class gko::Combination< ValueType >

The Combination class can be used to construct a linear combination of multiple linear operators `c1 * op1 + c2 * op2 + ...`

- ck ∗ opk.

Template Parameters

ValueType	precision of input and result vectors

41.39.2 Member Function Documentation

41.39.2.1 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::Combination< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.39.2.2 get_coefficients()

```
template<typename ValueType = default_precision>
const std::vector<std::shared_ptr<const LinOp> >& gko::Combination< ValueType >::get_↩
coefficients ( ) const [inline], [noexcept]
```

Returns a list of coefficients of the combination.

Returns

a list of coefficients

```
{
    return coefficients_;
}
```

41.39.2.3 get_operators()

```
template<typename ValueType = default_precision>
const std::vector<std::shared_ptr<const LinOp> >& gko::Combination< ValueType >::get_↩
operators ( ) const [inline], [noexcept]
```

Returns a list of operators of the combination.

Returns

a list of operators

41.39.2.4 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::Combination< ValueType >::transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/base/combination.hpp

41.40 gko::stop::Combined Class Reference

The Combined class is used to combine multiple criterions together through an OR operation.

`#include <ginkgo/core/stop/combined.hpp>`

41.40.1 Detailed Description

The Combined class is used to combine multiple criterions together through an OR operation.

The typical use case is to stop the iteration process if any of the criteria is fulfilled, e.g. a number of iterations, the relative residual norm has reached a threshold, etc.

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

- ginkgo/core/stop/combined.hpp

41.41 gko::Composition< ValueType > Class Template Reference

The Composition class can be used to compose linear operators `op1, op2, ..., opn` and obtain the operator `op1 * op2 * ...`

`#include <ginkgo/core/base/composition.hpp>`

Public Member Functions

- const std::vector< std::shared_ptr< const LinOp > > & get_operators () const noexcept

  Returns a list of operators of the composition.
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.41.1 Detailed Description

template<typename ValueType = default_precision>
class gko::Composition< ValueType >

The Composition class can be used to compose linear operators `op1, op2, ..., opn` and obtain the operator `op1 * op2 * ...`

- opn.

All LinOps of the Composition must operate on Dense inputs. For an operator `op_k` that require an initial guess for their `apply`, Composition provides either

- the output of the previous `op_{k+1}->apply` if `op_k` has square dimension
- zero if `op_k` is rectangular as an initial guess.

Template Parameters

ValueType	precision of input and result vectors

41.41.2 Member Function Documentation

41.41.2.1 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::Composition< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.41.2.2 get_operators()

```
template<typename ValueType = default_precision>
const std::vector<std::shared_ptr<const LinOp> >& gko::Composition< ValueType >::get_←
operators ( ) const [inline], [noexcept]
```

Returns a list of operators of the composition.

Returns

a list of operators

```
{
    return operators_;
}
```

41.41.2.3 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::Composition< ValueType >::transpose ( ) const [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/base/composition.hpp

41.42 gko::log::Convergence< ValueType > Class Template Reference

Convergence is a Logger which logs data strictly from the `criterion_check_completed` event.

`#include <ginkgo/core/log/convergence.hpp>`

Public Member Functions

- bool has_converged () const noexcept
  Returns true if the solver has converged.
- void reset_convergence_status ()
  Resets the convergence status to false.
- const size_type & get_num_iterations () const noexcept
  Returns the number of iterations.
- const LinOp ∗ get_residual () const noexcept
  Returns the residual.
- const LinOp ∗ get_residual_norm () const noexcept
  Returns the residual norm.
- const LinOp ∗ get_implicit_sq_resnorm () const noexcept
  Returns the implicit squared residual norm.

Static Public Member Functions

- static std::unique_ptr< Convergence > create (std::shared_ptr< const Executor > exec, const mask_type &enabled_events=Logger::all_events_mask)

    Creates a convergence logger.

41.42.1 Detailed Description

template<typename ValueType = default_precision>
class gko::log::Convergence< ValueType >

Convergence is a Logger which logs data strictly from the `criterion_check_completed` event.

The purpose of this logger is to give a simple access to standard data generated by the solver once it has stopped with minimal overhead.

This logger also computes the residual norm from the residual when the residual norm was not available. This can add some slight overhead.

41.42.2 Member Function Documentation

41.42.2.1 create()

```
template<typename ValueType = default_precision>
static std::unique_ptr<Convergence> gko::log::Convergence< ValueType >::create (
            std::shared_ptr< const Executor > exec,
            const mask_type & enabled_events = Logger::all_events_mask )  [inline], [static]
```

Creates a convergence logger.

This dynamically allocates the memory, constructs the object and returns an std::unique_ptr to this object.

Parameters

exec	the executor
enabled_events	the events enabled for this logger. By default all events.

Returns

an std::unique_ptr to the the constructed object

```
    {
        return std::unique_ptr<Convergence>(
            new Convergence(exec, enabled_events));
    }
```

41.42.2.2 get_implicit_sq_resnorm()

```
template<typename ValueType = default_precision>
const LinOp* gko::log::Convergence< ValueType >::get_implicit_sq_resnorm ( ) const [inline],
[noexcept]
```

Returns the implicit squared residual norm.

Returns

the implicit squared residual norm

41.42.2.3 get_num_iterations()

```
template<typename ValueType = default_precision>
const size_type& gko::log::Convergence< ValueType >::get_num_iterations ( ) const [inline],
[noexcept]
```

Returns the number of iterations.

Returns

the number of iterations

41.42.2.4 get_residual()

```
template<typename ValueType = default_precision>
const LinOp* gko::log::Convergence< ValueType >::get_residual ( ) const [inline], [noexcept]
```

Returns the residual.

Returns

the residual

41.42.2.5 get_residual_norm()

```
template<typename ValueType = default_precision>
const LinOp* gko::log::Convergence< ValueType >::get_residual_norm ( ) const [inline], [noexcept]
```

Returns the residual norm.

Returns

the residual norm

41.42.2.6 has_converged()

```
template<typename ValueType = default_precision>
bool gko::log::Convergence< ValueType >::has_converged ( ) const  [inline], [noexcept]
```

Returns true if the solver has converged.

Returns

the bool flag for convergence status

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

- ginkgo/core/log/convergence.hpp

41.43 gko::ConvertibleTo< ResultType > Class Template Reference

ConvertibleTo interface is used to mark that the implementer can be converted to the object of ResultType.

`#include <ginkgo/core/base/polymorphic_object.hpp>`

Public Member Functions

- virtual void convert_to (result_type *result) const =0
  Converts the implementer to an object of type result_type.
- virtual void move_to (result_type *result)=0
  Converts the implementer to an object of type result_type by moving data from this object.

41.43.1 Detailed Description

template<typename ResultType>
class gko::ConvertibleTo< ResultType >

ConvertibleTo interface is used to mark that the implementer can be converted to the object of ResultType.

This interface is used to enable conversions between polymorphic objects. To mark that an object of type `U` can be converted to an object of type `V`, `U` should implement ConvertibleTo<V>. Then, the implementation of `PolymorphicObject::copy_from` automatically generated by `EnablePolymorphicObject` mixin will use RTTI to figure out that `U` implements the interface and convert it using the convert_to / move_to methods of the interface.

As an example, the following function:

```
{c++}
void my_function(const U *u, V *v) {
    v->copy_from(u);
}
```

will convert object `u` to object `v` by checking that `u` can be dynamically casted to `ConvertibleTo\<V\>`, and calling ConvertibleTo<V>::convert_to(V*)` to do the actual conversion.

In case `u` is passed as a unique_ptr, call to `convert_to` will be replaced by a call to `move_to` and trigger move semantics.

Template Parameters

ResultType	the type to which the implementer can be converted to, has to be a subclass of PolymorphicObject

41.43.2 Member Function Documentation

41.43.2.1 convert_to()

```
template<typename ResultType>
virtual void gko::ConvertibleTo< ResultType >::convert_to (
            result_type * result ) const  [pure virtual]
```

Converts the implementer to an object of type result_type.

Parameters

result	the object used to store the result of the conversion

Implemented in gko::EnablePolymorphicAssignment< ConcreteType, ResultType >, gko::EnablePolymorphicAssignment< BatchDia
gko::EnablePolymorphicAssignment< Isai< IsaiType, ValueType, IndexType > >, gko::EnablePolymorphicAssignment< SparsityCs
gko::EnablePolymorphicAssignment< Diagonal< ValueType > >, gko::EnablePolymorphicAssignment< Dense< ValueType > >,
gko::EnablePolymorphicAssignment< UpperTrs< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Hybrid< Value
gko::EnablePolymorphicAssignment< Identity< ValueType > >, gko::EnablePolymorphicAssignment< ConcreteLinOp >,
gko::EnablePolymorphicAssignment< BatchDirect< ValueType > >, gko::EnablePolymorphicAssignment< BatchBicgstab< ValueT
gko::EnablePolymorphicAssignment< Fft3 >, gko::EnablePolymorphicAssignment< ConcreteBatchLinOp >,
gko::EnablePolymorphicAssignment< Composition< ValueType > >, gko::EnablePolymorphicAssignment< RowGatherer< IndexT
gko::EnablePolymorphicAssignment< Fft2 >, gko::EnablePolymorphicAssignment< BatchCg< ValueType > >,
gko::EnablePolymorphicAssignment< Fbcsr< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Bicgstab< ValueT
gko::EnablePolymorphicAssignment< LowerTrs< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Combination<
gko::EnablePolymorphicAssignment< Multigrid >, gko::EnablePolymorphicAssignment< Gmres< ValueType > >,
gko::EnablePolymorphicAssignment< BatchCsr< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< CbGmres< Va
gko::EnablePolymorphicAssignment< Csr< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< BatchIdr< ValueType
gko::EnablePolymorphicAssignment< BatchDense< ValueType > >, gko::EnablePolymorphicAssignment< Ir< ValueType > >,
gko::EnablePolymorphicAssignment< ConcreteSolver >, gko::EnablePolymorphicAssignment< BatchEll< ValueType, IndexType >
gko::EnablePolymorphicAssignment< Coo< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Fcg< ValueType >
gko::EnablePolymorphicAssignment< AmgxPgm< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Rcm< ValueT
gko::EnablePolymorphicAssignment< Fft >, gko::EnablePolymorphicAssignment< Idr< ValueType > >, gko::EnablePolymorphicAs
gko::EnablePolymorphicAssignment< Cgs< ValueType > >, gko::EnablePolymorphicAssignment< Ell< ValueType, IndexType > >
gko::EnablePolymorphicAssignment< Ilu< LSolverType, USolverType, ReverseApply, IndexType > >, gko::EnablePolymorphicAssig
gko::EnablePolymorphicAssignment< Permutation< IndexType > >, gko::EnablePolymorphicAssignment< BatchGmres< ValueTyp
gko::EnablePolymorphicAssignment< Jacobi< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< BatchRichardson<
gko::EnablePolymorphicAssignment< Cg< ValueType > >, gko::EnablePolymorphicAssignment< Sellp< ValueType, IndexType >
gko::EnablePolymorphicAssignment< Bicg< ValueType > >, gko::EnablePolymorphicAssignment< Perturbation< ValueType > >,
and gko::preconditioner::Jacobi< ValueType, IndexType >.

41.43.2.2 move_to()

```
template<typename ResultType>
```

```
virtual void gko::ConvertibleTo< ResultType >::move_to (
            result_type * result ) [pure virtual]
```

Converts the implementer to an object of type result_type by moving data from this object.

This method is used when the implementer is a temporary object, and move semantics can be used.

Parameters

result	the object used to emplace the result of the conversion

Note

ConvertibleTo::move_to can be implemented by simply calling ConvertibleTo::convert_to. However, this operation can often be optimized by exploiting the fact that implementer's data can be moved to the result.

Implemented in gko::EnablePolymorphicAssignment< ConcreteType, ResultType >, gko::EnablePolymorphicAssignment< BatchDia
gko::EnablePolymorphicAssignment< Isai< IsaiType, ValueType, IndexType > >, gko::EnablePolymorphicAssignment< SparsityCsr
gko::EnablePolymorphicAssignment< Diagonal< ValueType > >, gko::EnablePolymorphicAssignment< Dense< ValueType > >,
gko::EnablePolymorphicAssignment< UpperTrs< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Hybrid< Value
gko::EnablePolymorphicAssignment< Identity< ValueType > >, gko::EnablePolymorphicAssignment< ConcreteLinOp >,
gko::EnablePolymorphicAssignment< BatchDirect< ValueType > >, gko::EnablePolymorphicAssignment< BatchBicgstab< ValueTy
gko::EnablePolymorphicAssignment< Fft3 >, gko::EnablePolymorphicAssignment< ConcreteBatchLinOp >,
gko::EnablePolymorphicAssignment< Composition< ValueType > >, gko::EnablePolymorphicAssignment< RowGatherer< IndexTy
gko::EnablePolymorphicAssignment< Fft2 >, gko::EnablePolymorphicAssignment< BatchCg< ValueType > >,
gko::EnablePolymorphicAssignment< Fbcsr< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Bicgstab< ValueTy
gko::EnablePolymorphicAssignment< LowerTrs< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Combination<
gko::EnablePolymorphicAssignment< Multigrid >, gko::EnablePolymorphicAssignment< Gmres< ValueType > >,
gko::EnablePolymorphicAssignment< BatchCsr< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< CbGmres< Va
gko::EnablePolymorphicAssignment< Csr< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< BatchIdr< ValueType
gko::EnablePolymorphicAssignment< BatchDense< ValueType > >, gko::EnablePolymorphicAssignment< Ir< ValueType > >,
gko::EnablePolymorphicAssignment< ConcreteSolver >, gko::EnablePolymorphicAssignment< BatchEll< ValueType, IndexType >
gko::EnablePolymorphicAssignment< Coo< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Fcg< ValueType >
gko::EnablePolymorphicAssignment< AmgxPgm< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< Rcm< ValueT
gko::EnablePolymorphicAssignment< Fft >, gko::EnablePolymorphicAssignment< Idr< ValueType > >, gko::EnablePolymorphicAs
gko::EnablePolymorphicAssignment< Cgs< ValueType > >, gko::EnablePolymorphicAssignment< Ell< ValueType, IndexType > >
gko::EnablePolymorphicAssignment< Ilu< LSolverType, USolverType, ReverseApply, IndexType > >, gko::EnablePolymorphicAssig
gko::EnablePolymorphicAssignment< Permutation< IndexType > >, gko::EnablePolymorphicAssignment< BatchGmres< ValueTyp
gko::EnablePolymorphicAssignment< Jacobi< ValueType, IndexType > >, gko::EnablePolymorphicAssignment< BatchRichardson<
gko::EnablePolymorphicAssignment< Cg< ValueType > >, gko::EnablePolymorphicAssignment< Sellp< ValueType, IndexType >
gko::EnablePolymorphicAssignment< Bicg< ValueType > >, gko::EnablePolymorphicAssignment< Perturbation< ValueType > >,
and gko::preconditioner::Jacobi< ValueType, IndexType >.

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

- ginkgo/core/base/polymorphic_object.hpp

41.44 gko::matrix::Coo< ValueType, IndexType > Class Template Reference

COO stores a matrix in the coordinate matrix format.

```
#include <ginkgo/core/matrix/coo.hpp>
```

Public Member Functions

- void read (const mat_data &data) override

  Reads a matrix from a matrix_data structure.
- void read (const device_mat_data &data) override

  Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override

  Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override

  Writes a matrix to a matrix_data structure.
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override

  Extracts the diagonal entries of the matrix into a vector.
- std::unique_ptr< absolute_type > compute_absolute () const override

  Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override

  Compute absolute inplace on each element.
- value_type * get_values () noexcept

  Returns the values of the matrix.
- const value_type * get_const_values () const noexcept

  Returns the values of the matrix.
- index_type * get_col_idxs () noexcept

  Returns the column indexes of the matrix.
- const index_type * get_const_col_idxs () const noexcept

  Returns the column indexes of the matrix.
- index_type * get_row_idxs () noexcept

  Returns the row indexes of the matrix.
- const index_type * get_const_row_idxs () const noexcept
- size_type get_num_stored_elements () const noexcept

  Returns the number of elements explicitly stored in the matrix.
- LinOp * apply2 (const LinOp *b, LinOp *x)

  Applies Coo matrix axpy to a vector (or a sequence of vectors).
- const LinOp * apply2 (const LinOp *b, LinOp *x) const
- LinOp * apply2 (const LinOp *alpha, const LinOp *b, LinOp *x)

  *Performs the operation x = alpha * Coo * b + x.*
- const LinOp * apply2 (const LinOp *alpha, const LinOp *b, LinOp *x) const

  *Performs the operation x = alpha * Coo * b + x.*

Static Public Member Functions

- static std::unique_ptr< const Coo > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs, gko::detail::ConstArrayView< IndexType > &&row_idxs)

  Creates a constant (immutable) Coo matrix from a set of constant arrays.

41.44.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Coo< ValueType, IndexType >

COO stores a matrix in the coordinate matrix format.

The nonzero elements are stored in an array row-wise (but not neccessarily sorted by column index within a row). Two extra arrays contain the row and column indexes of each nonzero element of the matrix.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.44.2 Member Function Documentation

41.44.2.1 apply2() [1/4]

```
template<typename ValueType = default_precision, typename IndexType = int32>
LinOp* gko::matrix::Coo< ValueType, IndexType >::apply2 (
            const LinOp * alpha,
            const LinOp * b,
            LinOp * x )  [inline]
```

Performs the operation x = alpha ∗ Coo ∗ b + x.

Parameters

alpha	scaling of the result of Coo ∗ b
b	vector(s) on which the operator is applied
x	output vector(s)

Returns

this

```
{
    this->validate_application_parameters(b, x);
    GKO_ASSERT_EQUAL_DIMENSIONS(alpha, dim<2>(1, 1));
    auto exec = this->get_executor();
    this->apply2_impl(make_temporary_clone(exec, alpha).get(),
                      make_temporary_clone(exec, b).get(),
                      make_temporary_clone(exec, x).get());
    return this;
}
```

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.44.2.2 apply2() [2/4]

```
template<typename ValueType = default_precision, typename IndexType = int32>
const LinOp* gko::matrix::Coo< ValueType, IndexType >::apply2 (
            const LinOp * alpha,
            const LinOp * b,
            LinOp * x ) const  [inline]
```

Performs the operation x = alpha ∗ Coo ∗ b + x.

Parameters

alpha	scaling of the result of Coo ∗ b
b	vector(s) on which the operator is applied
x	output vector(s)

Returns

this

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.44.2.3 apply2() [3/4]

```
template<typename ValueType = default_precision, typename IndexType = int32>
LinOp* gko::matrix::Coo< ValueType, IndexType >::apply2 (
            const LinOp * b,
            LinOp * x ) [inline]
```

Applies Coo matrix axpy to a vector (or a sequence of vectors).

Performs the operation x = Coo ∗ b + x

Parameters

b	the input vector(s) on which the operator is applied
x	the output vector(s) where the result is stored

Returns

this

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.44.2.4 apply2() [4/4]

```
template<typename ValueType = default_precision, typename IndexType = int32>
const LinOp* gko::matrix::Coo< ValueType, IndexType >::apply2 (
            const LinOp * b,
            LinOp * x ) const [inline]
```

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.44.2.5 compute_absolute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<absolute_type> gko::matrix::Coo< ValueType, IndexType >::compute_absolute ( )
const  [override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Coo< ValueType, IndexType > > >.

41.44.2.6 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const Coo> gko::matrix::Coo< ValueType, IndexType >::create_const (
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            gko::detail::ConstArrayView< ValueType > && values,
            gko::detail::ConstArrayView< IndexType > && col_idxs,
            gko::detail::ConstArrayView< IndexType > && row_idxs )  [inline], [static]
```

Creates a constant (immutable) Coo matrix from a set of constant arrays.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
col_idxs	the column index array of the matrix
row_ptrs	the row index array of the matrix

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of these arrays on the correct executor.

41.44.2.7 extract_diagonal()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Coo< ValueType, IndexType >::extract_↩
diagonal ( ) const  [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.44.2.8 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Coo< ValueType, IndexType >::get_col_idxs ( )  [inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

References gko::Array< ValueType >::get_data().

41.44.2.9 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Coo< ValueType, IndexType >::get_const_col_idxs ( ) const  [inline],
[noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.44.2.10 get_const_row_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Coo< ValueType, IndexType >::get_const_row_idxs ( ) const [inline],
[noexcept]
```

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.44.2.11 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Coo< ValueType, IndexType >::get_const_values ( ) const [inline],
[noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.44.2.12 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Coo< ValueType, IndexType >::get_num_stored_elements ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.44.2.13 get_row_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Coo< ValueType, IndexType >::get_row_idxs ( ) [inline], [noexcept]
```

Returns the row indexes of the matrix.

Returns

the row indexes of the matrix.

References gko::Array< ValueType >::get_data().

41.44.2.14 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Coo< ValueType, IndexType >::get_values ( ) [inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

References gko::Array< ValueType >::get_data().

41.44.2.15 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Coo< ValueType, IndexType >::read (
            const device_mat_data & data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.44.2.16 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Coo< ValueType, IndexType >::read (
            const mat_data & data ) [override], [virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.44.2.17 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Coo< ValueType, IndexType >::read (
            device_mat_data && data )  [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.44.2.18 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Coo< ValueType, IndexType >::write (
            mat_data & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/coo.hpp

41.45 gko::cpx_real_type< T > Struct Template Reference

Access the underlying real type of a complex number.

```
#include <ginkgo/core/base/math.hpp>
```

Public Types

- using type = T
  The type.

41.45.1 Detailed Description

template<typename T>
struct gko::cpx_real_type< T >

Access the underlying real type of a complex number.

Template Parameters

T	the type being checked.

41.45.2 Member Typedef Documentation

41.45.2.1 type

```
template<typename T >
using gko::cpx_real_type< T >::type = T
```

The type.

When the type is not complex, return the type itself.

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

- ginkgo/core/base/math.hpp

41.46 gko::stop::Criterion Class Reference

The Criterion class is a base class for all stopping criteria.

```
#include <ginkgo/core/stop/criterion.hpp>
```

Classes

- class Updater
  The Updater class serves for convenient argument passing to the Criterion's check function.

Public Member Functions

- Updater update ()

  Returns the updater object.
- bool check (uint8 stopping_id, bool set_finalized, Array< stopping_status > ∗stop_status, bool ∗one_↩ changed, const Updater &updater)

  This checks whether convergence was reached for a certain criterion.

41.46.1 Detailed Description

The Criterion class is a base class for all stopping criteria.

It contains a factory to instantiate criteria. It is up to each specific stopping criterion to decide what to do with the data that is passed to it.

Note that depending on the criterion, convergence may not have happened after stopping.

41.46.2 Member Function Documentation

41.46.2.1 check()

```
bool gko::stop::Criterion::check (
            uint8 stopping_id,
            bool set_finalized,
            Array< stopping_status > * stop_status,
            bool * one_changed,
            const Updater & updater )  [inline]
```

This checks whether convergence was reached for a certain criterion.

The actual implantation of the criterion goes here.

Parameters

stopping_id	id of the stopping criterion
set_finalized	Controls if the current version should count as finalized or not
stop_status	status of the stopping criterion
one_changed	indicates if one vector's status changed
updater	the Updater object containing all the information

Returns

whether convergence was completely reached

```
{
    this->template log<log::Logger::criterion_check_started>(
        this, updater.num_iterations_, updater.residual_,
        updater.residual_norm_, updater.solution_, stopping_id,
        set_finalized);
```

```
        auto all_converged = this->check_impl(
            stopping_id, set_finalized, stop_status, one_changed, updater);
        this->template log<log::Logger::criterion_check_completed>(
            this, updater.num_iterations_, updater.residual_,
            updater.residual_norm_, updater.implicit_sq_residual_norm_,
            updater.solution_, stopping_id, set_finalized, stop_status,
            *one_changed, all_converged);
        return all_converged;
    }
```

Referenced by gko::stop::Criterion::Updater::check().

41.46.2.2 update()

`Updater gko::stop::Criterion::update ( ) [inline]`

Returns the updater object.

Returns

the updater object

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

- ginkgo/core/stop/criterion.hpp

41.47 gko::log::criterion_data Struct Reference

Struct representing Criterion related data.

`#include <ginkgo/core/log/record.hpp>`

41.47.1 Detailed Description

Struct representing Criterion related data.

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

- ginkgo/core/log/record.hpp

41.48 gko::stop::CriterionArgs Struct Reference

This struct is used to pass parameters to the EnableDefaultCriterionFactoryCriterionFactory::generate() method.

`#include <ginkgo/core/stop/criterion.hpp>`

41.48.1 Detailed Description

This struct is used to pass parameters to the EnableDefaultCriterionFactoryCriterionFactory::generate() method.

It is the ComponentsType of CriterionFactory.

Note

Dependly on the use case, some of these parameters can be `nullptr` as only some stopping criterion require them to be set. An example is the `ResidualNormReduction` which really requires the `initial_residual` to be set.

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

- ginkgo/core/stop/criterion.hpp

41.49 gko::matrix::Csr< ValueType, IndexType > Class Template Reference

CSR is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).

```
#include <ginkgo/core/matrix/csr.hpp>
```

Classes

- class classical

    classical is a strategy_type which uses the same number of threads on each row.
- class cusparse

    cusparse is a strategy_type which uses the sparselib csr.
- class load_balance

    load_balance is a strategy_type which uses the load balance algorithm.
- class merge_path

    merge_path is a strategy_type which uses the merge_path algorithm.
- class sparselib

    sparselib is a strategy_type which uses the sparselib csr.
- class strategy_type

    strategy_type is to decide how to set the csr algorithm.

Public Member Functions

- void read (const mat_data &data) override

  Reads a matrix from a matrix_data structure.
- void read (const device_mat_data &data) override

  Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override

  Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override

  Writes a matrix to a matrix_data structure.
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.
- std::unique_ptr< LinOp > permute (const Array< IndexType > *permutation_indices) const override

  Returns a LinOp representing the symmetric row and column permutation of the Permutable object.
- std::unique_ptr< LinOp > inverse_permute (const Array< IndexType > *inverse_permutation_indices) const override

  Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.
- std::unique_ptr< LinOp > row_permute (const Array< IndexType > *permutation_indices) const override

  Returns a LinOp representing the row permutation of the Permutable object.
- std::unique_ptr< LinOp > column_permute (const Array< IndexType > *permutation_indices) const override

  Returns a LinOp representing the column permutation of the Permutable object.
- std::unique_ptr< LinOp > inverse_row_permute (const Array< IndexType > *inverse_permutation_indices) const override

  Returns a LinOp representing the row permutation of the inverse permuted object.
- std::unique_ptr< LinOp > inverse_column_permute (const Array< IndexType > *inverse_permutation_↩ indices) const override

  Returns a LinOp representing the row permutation of the inverse permuted object.
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override

  Extracts the diagonal entries of the matrix into a vector.
- std::unique_ptr< absolute_type > compute_absolute () const override

  Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override

  Compute absolute inplace on each element.
- void sort_by_column_index ()

  Sorts all (value, col_idx) pairs in each row by column index.
- value_type * get_values () noexcept

  Returns the values of the matrix.
- const value_type * get_const_values () const noexcept

  Returns the values of the matrix.
- index_type * get_col_idxs () noexcept

  Returns the column indexes of the matrix.
- const index_type * get_const_col_idxs () const noexcept

  Returns the column indexes of the matrix.
- index_type * get_row_ptrs () noexcept

  Returns the row pointers of the matrix.
- const index_type * get_const_row_ptrs () const noexcept

  Returns the row pointers of the matrix.
- index_type * get_srow () noexcept

  Returns the starting rows.

- const index_type * get_const_srow () const noexcept

    Returns the starting rows.
- size_type get_num_srow_elements () const noexcept

    Returns the number of the srow stored elements (involved warps)
- size_type get_num_stored_elements () const noexcept

    Returns the number of elements explicitly stored in the matrix.
- std::shared_ptr< strategy_type > get_strategy () const noexcept

    Returns the strategy.
- void set_strategy (std::shared_ptr< strategy_type > strategy)

    Set the strategy.
- void scale (const LinOp *alpha)

    Scales the matrix with a scalar.
- void inv_scale (const LinOp *alpha)

    Scales the matrix with the inverse of a scalar.
- std::unique_ptr< Csr< ValueType, IndexType > > create_submatrix (const index_set< IndexType > &row←_index_set, const index_set< IndexType > &column_index_set) const

    Creates a submatrix from this Csr matrix given row and column index_set objects.
- std::unique_ptr< Csr< ValueType, IndexType > > create_submatrix (const span &row_span, const span &column_span) const

    Creates a submatrix from this Csr matrix given row and column spans.

Static Public Member Functions

- static std::unique_ptr< const Csr > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs, gko::detail::ConstArrayView< IndexType > &&row_ptrs, std::shared_ptr< strategy_type > strategy)

    Creates a constant (immutable) Csr matrix from a set of constant arrays.
- static std::unique_ptr< const Csr > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs, gko::detail::ConstArrayView< IndexType > &&row_ptrs)

    This is version of create_const with a default strategy.

41.49.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >

CSR is a matrix format which stores only the nonzero coefficients by compressing each row of the matrix (compressed sparse row format).

The nonzero elements are stored in a 1D array row-wise, and accompanied with a row pointer array which stores the starting index of each row. An additional column index array is used to identify the column of each nonzero element.

The Csr LinOp supports different operations:

```
matrix::Csr *A, *B, *C;       // matrices
matrix::Dense *b, *x;         // vectors tall-and-skinny matrices
matrix::Dense *alpha, *beta; // scalars of dimension 1x1
matrix::Identity *I;         // identity matrix
// Applying to Dense matrices computes an SpMV/SpMM product
A->apply(b, x)              // x = A*b
A->apply(alpha, b, beta, x) // x = alpha*A*b + beta*x
// Applying to Csr matrices computes a SpGEMM product of two sparse matrices
A->apply(B, C)              // C = A*B
A->apply(alpha, B, beta, C) // C = alpha*A*B + beta*C
// Applying to an Identity matrix computes a SpGEAM sparse matrix addition
A->apply(alpha, I, beta, B) // B = alpha*A + beta*B
```

Both the SpGEMM and SpGEAM operation require the input matrices to be sorted by column index, otherwise the algorithms will produce incorrect results.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.49.2 Member Function Documentation

41.49.2.1 column_permute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::column_permute (
            const Array< IndexType > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the column permutation of the Permutable object.

In the resulting LinOp, the column `i` contains the input column `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new column permuted object

Implements gko::Permutable< IndexType >.

41.49.2.2 compute_absolute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<absolute_type> gko::matrix::Csr< ValueType, IndexType >::compute_absolute ( )
const  [override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Csr< ValueType, IndexType > > >.

41.49.2.3 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.49.2.4 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const Csr> gko::matrix::Csr< ValueType, IndexType >::create_const (
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            gko::detail::ConstArrayView< ValueType > && values,
            gko::detail::ConstArrayView< IndexType > && col_idxs,
            gko::detail::ConstArrayView< IndexType > && row_ptrs,
            std::shared_ptr< strategy_type > strategy ) [inline], [static]
```

Creates a constant (immutable) Csr matrix from a set of constant arrays.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
col_idxs	the column index array of the matrix
row_ptrs	the row pointer array of the matrix
strategy	the strategy the matrix uses for SpMV operations

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of these arrays on the correct executor.

```
{
    // cast const-ness away, but return a const object afterwards,
    // so we can ensure that no modifications take place.
    return std::unique_ptr<const Csr>(new Csr{
        exec, size, gko::detail::array_const_cast(std::move(values)),
        gko::detail::array_const_cast(std::move(col_idxs)),
        gko::detail::array_const_cast(std::move(row_ptrs)), strategy});
}
```

Referenced by gko::matrix::Csr< ValueType, IndexType >::create_const().

41.49.2.5 create_submatrix() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Csr<ValueType, IndexType> > gko::matrix::Csr< ValueType, IndexType >::create←
_submatrix (
            const index_set< IndexType > & row_index_set,
            const index_set< IndexType > & column_index_set ) const
```

Creates a submatrix from this Csr matrix given row and column index_set objects.

Parameters

row_index_set	the row index set containing the set of rows to be in the submatrix.
column_index_set	the col index set containing the set of columns to be in the submatrix.

Returns

A new CSR matrix with the elements that belong to the row and columns of this matrix as specified by the index sets.

Note

This is not a view but creates a new, separate CSR matrix.

41.49.2.6 create_submatrix() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Csr<ValueType, IndexType> > gko::matrix::Csr< ValueType, IndexType >::create←
_submatrix (
            const span & row_span,
            const span & column_span ) const
```

Creates a submatrix from this Csr matrix given row and column spans.

Parameters

row_span	the row span containing the contiguous set of rows to be in the submatrix.
column_span	the column span containing the contiguous set of columns to be in the submatrix.

Returns

A new CSR matrix with the elements that belong to the row and columns of this matrix as specified by the index sets.

Note

This is not a view but creates a new, separate CSR matrix.

41.49.2.7 extract_diagonal()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Csr< ValueType, IndexType >::extract_↩
diagonal ( ) const [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.49.2.8 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Csr< ValueType, IndexType >::get_col_idxs ( ) [inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

References gko::Array< ValueType >::get_data().

41.49.2.9 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Csr< ValueType, IndexType >::get_const_col_idxs ( ) const [inline],
[noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.49.2.10 get_const_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Csr< ValueType, IndexType >::get_const_row_ptrs ( ) const [inline],
[noexcept]
```

Returns the row pointers of the matrix.

Returns

the row pointers of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.49.2.11 get_const_srow()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Csr< ValueType, IndexType >::get_const_srow ( ) const [inline],
[noexcept]
```

Returns the starting rows.

Returns

the starting rows.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.49.2.12 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Csr< ValueType, IndexType >::get_const_values ( ) const [inline],
[noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.49.2.13 get_num_srow_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Csr< ValueType, IndexType >::get_num_srow_elements ( ) const [inline],
[noexcept]
```

Returns the number of the srow stored elements (involved warps)

Returns

the number of the srow stored elements (involved warps)

References gko::Array< ValueType >::get_num_elems().

41.49.2.14 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Csr< ValueType, IndexType >::get_num_stored_elements ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.49.2.15 get_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Csr< ValueType, IndexType >::get_row_ptrs ( ) [inline], [noexcept]
```

Returns the row pointers of the matrix.

Returns

the row pointers of the matrix.

References gko::Array< ValueType >::get_data().

41.49.2.16 get_srow()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Csr< ValueType, IndexType >::get_srow ( ) [inline], [noexcept]
```

Returns the starting rows.

Returns

the starting rows.

References gko::Array< ValueType >::get_data().

41.49.2.17 get_strategy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::get_strategy ( )
const [inline], [noexcept]
```

Returns the strategy.

Returns

the strategy

41.49.2.18 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Csr< ValueType, IndexType >::get_values ( ) [inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

References gko::Array< ValueType >::get_data().

41.49.2.19 inv_scale()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::inv_scale (
            const LinOp * alpha ) [inline]
```

Scales the matrix with the inverse of a scalar.

Parameters

alpha	The entire matrix is scaled by 1 / alpha. alpha has to be a 1x1 Dense matrix.

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.49.2.20 inverse_column_permute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::inverse_column_permute (
            const Array< IndexType > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the column `perm[i]` contains the input column `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implements gko::Permutable< IndexType >.

41.49.2.21 inverse_permute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::inverse_permute (
            const Array< IndexType > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(perm[i],perm[j])` contains the input value `(i,j)`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented from gko::Permutable< IndexType >.

41.49.2.22 inverse_row_permute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::inverse_row_permute (
            const Array< IndexType > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the row `perm[i]` contains the input row `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implements gko::Permutable< IndexType >.

41.49.2.23 permute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::permute (
            const Array< IndexType > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the symmetric row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(i,j)` contains the input value `(perm[i],perm[j])`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented from gko::Permutable< IndexType >.

41.49.2.24 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::read (
            const device_mat_data & data )  [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.49.2.25 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::read (
            const mat_data & data ) [override], [virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.49.2.26 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::read (
            device_mat_data && data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.49.2.27 row_permute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::row_permute (
            const Array< IndexType > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the row permutation of the Permutable object.

In the resulting LinOp, the row `i` contains the input row `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Implements gko::Permutable< IndexType >.

41.49.2.28 scale()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::scale (
            const LinOp * alpha )  [inline]
```

Scales the matrix with a scalar.

Parameters

alpha	The entire matrix is scaled by alpha. alpha has to be a 1x1 Dense matrix.

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.49.2.29 set_strategy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::set_strategy (
            std::shared_ptr< strategy_type > strategy )  [inline]
```

Set the strategy.

Parameters

strategy	the csr strategy

41.49.2.30 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Csr< ValueType, IndexType >::transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.49.2.31 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::write (
            mat_data & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/coo.hpp
- ginkgo/core/matrix/csr.hpp

41.50 gko::CublasError Class Reference

CublasError is thrown when a cuBLAS routine throws a non-zero error code.

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- CublasError (const std::string &file, int line, const std::string &func, int64 error_code)

    Initializes a cuBLAS error.

41.50.1 Detailed Description

CublasError is thrown when a cuBLAS routine throws a non-zero error code.

41.50.2 Constructor & Destructor Documentation

41.50.2.1 CublasError()

```
gko::CublasError::CublasError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code )  [inline]
```

Initializes a cuBLAS 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 cuBLAS routine that failed
error_code	The resulting cuBLAS error code

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

- ginkgo/core/base/exception.hpp

41.51 gko::CudaError Class Reference

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

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- CudaError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a CUDA error.

41.51.1 Detailed Description

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

41.51.2 Constructor & Destructor Documentation

41.51.2.1 CudaError()

```
gko::CudaError::CudaError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 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

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

- ginkgo/core/base/exception.hpp

41.52 gko::CudaExecutor Class Reference

This is the Executor subclass which represents the CUDA device.

`#include <ginkgo/core/base/executor.hpp>`

Public Member Functions

- std::shared_ptr< Executor > get_master () noexcept override

    Returns the master OmpExecutor of this Executor.
- std::shared_ptr< const Executor > get_master () const noexcept override

    Returns the master OmpExecutor of this Executor.
- void synchronize () const override

    Synchronize the operations launched on the executor with its master.
- void run (const Operation &op) const override

    Runs the specified Operation using this Executor.
- int get_device_id () const noexcept

    Get the CUDA device id of the device associated to this executor.
- int get_num_warps_per_sm () const noexcept

    Get the number of warps per SM of this executor.
- int get_num_multiprocessor () const noexcept

    Get the number of multiprocessor of this executor.
- int get_num_warps () const noexcept

    Get the number of warps of this executor.
- int get_warp_size () const noexcept

    Get the warp size of this executor.
- int get_max_shared_memory_per_block () const noexcept

    Get maximum shared memory per block.
- int get_major_version () const noexcept

    Get the major verion of compute capability.
- int get_minor_version () const noexcept

    Get the minor verion of compute capability.
- cublasContext ∗ get_cublas_handle () const

    Get the cublas handle for this executor.
- cusparseContext ∗ get_cusparse_handle () const

    Get the cusparse handle for this executor.
- std::vector< int > get_closest_pus () const

    Get the closest PUs.
- int get_closest_numa () const

    Get the closest NUMA node.

Static Public Member Functions

- static std::shared_ptr< CudaExecutor > create (int device_id, std::shared_ptr< Executor > master, bool device_reset=false, allocation_mode alloc_mode=default_cuda_alloc_mode)

    Creates a new CudaExecutor.
- static int get_num_devices ()

    Get the number of devices present on the system.

41.52.1 Detailed Description

This is the Executor subclass which represents the CUDA device.

41.52.2 Member Function Documentation

41.52.2.1 create()

```
static std::shared_ptr<CudaExecutor> gko::CudaExecutor::create (
            int device_id,
            std::shared_ptr< Executor > master,
            bool device_reset = false,
            allocation_mode alloc_mode = default_cuda_alloc_mode )  [static]
```

Creates a new CudaExecutor.

Parameters

device_id	the CUDA device id of this device
master	an executor on the host that is used to invoke the device kernels
device_reset	whether to reset the device after the object exits the scope.
alloc_mode	the allocation mode that the executor should operate on. See @allocation_mode for more details

41.52.2.2 get_closest_numa()

```
int gko::CudaExecutor::get_closest_numa ( ) const  [inline]
```

Get the closest NUMA node.

Returns

the closest NUMA node closest to this device

41.52.2.3 get_closest_pus()

```
std::vector<int> gko::CudaExecutor::get_closest_pus ( ) const [inline]
```

Get the closest PUs.

Returns

the array of PUs closest to this device

41.52.2.4 get_cublas_handle()

```
cublasContext* gko::CudaExecutor::get_cublas_handle ( ) const [inline]
```

Get the cublas handle for this executor.

Returns

the cublas handle (cublasContext*) for this executor

41.52.2.5 get_cusparse_handle()

```
cusparseContext* gko::CudaExecutor::get_cusparse_handle ( ) const [inline]
```

Get the cusparse handle for this executor.

Returns

the cusparse handle (cusparseContext*) for this executor

41.52.2.6 get_master() [1/2]

```
std::shared_ptr<const Executor> gko::CudaExecutor::get_master ( ) const [override], [virtual],
[noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.52.2.7 get_master() [2/2]

```
std::shared_ptr<Executor> gko::CudaExecutor::get_master ( ) [override], [virtual], [noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.52.2.8 run()

```
void gko::CudaExecutor::run (
            const Operation & op ) const [override], [virtual]
```

Runs the specified Operation using this Executor.

Parameters

op	the operation to run

Implements gko::Executor.

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

- ginkgo/core/base/executor.hpp

41.53 gko::CufftError Class Reference

CufftError is thrown when a cuFFT routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- CufftError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a cuFFT error.

41.53.1 Detailed Description

CufftError is thrown when a cuFFT routine throws a non-zero error code.

41.53.2 Constructor & Destructor Documentation

41.53.2.1 CufftError()

```
gko::CufftError::CufftError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code ) [inline]
```

Initializes a cuFFT 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 cuFFT routine that failed
error_code	The resulting cuFFT error code

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

- ginkgo/core/base/exception.hpp

41.54 gko::CurandError Class Reference

CurandError is thrown when a cuRAND routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- CurandError (const std::string &file, int line, const std::string &func, int64 error_code)

    Initializes a cuRAND error.

41.54.1 Detailed Description

CurandError is thrown when a cuRAND routine throws a non-zero error code.

41.54.2 Constructor & Destructor Documentation

41.54.2.1 CurandError()

```
gko::CurandError::CurandError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code ) [inline]
```

Initializes a cuRAND 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 cuRAND routine that failed
error_code	The resulting cuRAND error code

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

- ginkgo/core/base/exception.hpp

41.55 gko::matrix::Csr< ValueType, IndexType >::cusparse Class Reference

cusparse is a strategy_type which uses the sparselib csr.

`#include <ginkgo/core/matrix/csr.hpp>`

Public Member Functions

- cusparse ()
  Creates a cusparse strategy.
- void process (const Array< index_type > &mtx_row_ptrs, Array< index_type > *mtx_srow) override
  Computes srow according to row pointers.
- int64_t clac_size (const int64_t nnz) override
  Computes the srow size according to the number of nonzeros.
- std::shared_ptr< strategy_type > copy () override
  Copy a strategy.

41.55.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >::cusparse

cusparse is a strategy_type which uses the sparselib csr.

Note

cusparse is also known to the hip executor which converts between cuda and hip.

41.55.2 Member Function Documentation

41.55.2.1 clac_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
int64_t gko::matrix::Csr< ValueType, IndexType >::cusparse::clac_size (
            const int64_t nnz ) [inline], [override], [virtual]
```

Computes the srow size according to the number of nonzeros.

Parameters

nnz	the number of nonzeros

Returns

the size of srow

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.55.2.2 copy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::cusparse::copy ( )
[inline], [override], [virtual]
```

Copy a strategy.

This is a workaround until strategies are revamped, since strategies like `automatical` do not work when actually shared.

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.55.2.3 process()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::cusparse::process (
            const Array< index_type > & mtx_row_ptrs,
            Array< index_type > * mtx_srow )  [inline], [override], [virtual]
```

Computes srow according to row pointers.

Parameters

mtx_row_ptrs	the row pointers of the matrix
mtx_srow	the srow of the matrix

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

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

- ginkgo/core/matrix/csr.hpp

41.56 gko::CusparseError Class Reference

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

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- CusparseError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a cuSPARSE error.

41.56.1 Detailed Description

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

41.56.2 Constructor & Destructor Documentation

41.56.2.1 CusparseError()

```
gko::CusparseError::CusparseError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 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

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

- ginkgo/core/base/exception.hpp

41.57 gko::default_converter< S, R > Struct Template Reference

Used to convert objects of type `S` to objects of type `R` using static_cast.

`#include <ginkgo/core/base/math.hpp>`

Public Member Functions

- R operator() (S val)

    Converts the object to result type.

41.57.1 Detailed Description

template<typename S, typename R>
struct gko::default_converter< S, R >

Used to convert objects of type S to objects of type R using static_cast.

Template Parameters

S	source type
R	result type

41.57.2 Member Function Documentation

41.57.2.1 operator()()

```
template<typename S , typename R >
R gko::default_converter< S, R >::operator() (
            S val ) [inline]
```

Converts the object to result type.

Parameters

val	the object to convert

Returns

the converted object

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

- ginkgo/core/base/math.hpp

41.58 gko::matrix::Dense< ValueType > Class Template Reference

Dense is a matrix format which explicitly stores all values of the matrix.

```
#include <ginkgo/core/matrix/dense.hpp>
```

Public Member Functions

- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- void transpose (Dense *output) const
  Writes the transposed matrix into the given output matrix.
- void conj_transpose (Dense *output) const
  Writes the conjugate-transposed matrix into the given output matrix.
- void fill (const ValueType value)
  Fill the dense matrix with a given value.
- std::unique_ptr< LinOp > permute (const Array< int32 > *permutation_indices) const override
  Returns a LinOp representing the symmetric row and column permutation of the Permutable object.
- std::unique_ptr< LinOp > permute (const Array< int64 > *permutation_indices) const override
  Returns a LinOp representing the symmetric row and column permutation of the Permutable object.
- void permute (const Array< int32 > *permutation_indices, Dense *output) const
  Writes the symmetrically permuted matrix into the given output matrix.
- void permute (const Array< int64 > *permutation_indices, Dense *output) const
- std::unique_ptr< LinOp > inverse_permute (const Array< int32 > *permutation_indices) const override
  Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.
- std::unique_ptr< LinOp > inverse_permute (const Array< int64 > *permutation_indices) const override
  Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.
- void inverse_permute (const Array< int32 > *permutation_indices, Dense *output) const
  Writes the inverse symmetrically permuted matrix into the given output matrix.
- void inverse_permute (const Array< int64 > *permutation_indices, Dense *output) const
- std::unique_ptr< LinOp > row_permute (const Array< int32 > *permutation_indices) const override
  Returns a LinOp representing the row permutation of the Permutable object.
- std::unique_ptr< LinOp > row_permute (const Array< int64 > *permutation_indices) const override
  Returns a LinOp representing the row permutation of the Permutable object.
- void row_permute (const Array< int32 > *permutation_indices, Dense *output) const
  Writes the row-permuted matrix into the given output matrix.
- void row_permute (const Array< int64 > *permutation_indices, Dense *output) const
- std::unique_ptr< Dense > row_gather (const Array< int32 > *gather_indices) const
  Create a Dense matrix consisting of the given rows from this matrix.
- std::unique_ptr< Dense > row_gather (const Array< int64 > *gather_indices) const
  Create a Dense matrix consisting of the given rows from this matrix.
- void row_gather (const Array< int32 > *gather_indices, Dense *row_collection) const
  Copies the given rows from this matrix into `row_collection`
- void row_gather (const Array< int64 > *gather_indices, Dense *row_collection) const
  Copies the given rows from this matrix into `row_collection`
- void row_gather (const Array< int32 > *gather_indices, LinOp *row_collection) const
  Copies the given rows from this matrix into `row_collection`
- void row_gather (const Array< int64 > *gather_indices, LinOp *row_collection) const
  Copies the given rows from this matrix into `row_collection`
- void row_gather (const LinOp *alpha, const Array< int32 > *gather_indices, const LinOp *beta, LinOp *row_collection) const
  Copies the given rows from this matrix into `row_collection` with scaling.
- void row_gather (const LinOp *alpha, const Array< int64 > *gather_indices, const LinOp *beta, LinOp *row_collection) const
- std::unique_ptr< LinOp > column_permute (const Array< int32 > *permutation_indices) const override

Returns a LinOp representing the column permutation of the Permutable object.
- std::unique_ptr< LinOp > column_permute (const Array< int64 > *permutation_indices) const override

  Returns a LinOp representing the column permutation of the Permutable object.
- void column_permute (const Array< int32 > *permutation_indices, Dense *output) const

  Writes the column-permuted matrix into the given output matrix.
- void column_permute (const Array< int64 > *permutation_indices, Dense *output) const
- std::unique_ptr< LinOp > inverse_row_permute (const Array< int32 > *permutation_indices) const override

  Returns a LinOp representing the row permutation of the inverse permuted object.
- std::unique_ptr< LinOp > inverse_row_permute (const Array< int64 > *permutation_indices) const override

  Returns a LinOp representing the row permutation of the inverse permuted object.
- void inverse_row_permute (const Array< int32 > *permutation_indices, Dense *output) const

  Writes the inverse row-permuted matrix into the given output matrix.
- void inverse_row_permute (const Array< int64 > *permutation_indices, Dense *output) const
- std::unique_ptr< LinOp > inverse_column_permute (const Array< int32 > *permutation_indices) const override

  Returns a LinOp representing the row permutation of the inverse permuted object.
- std::unique_ptr< LinOp > inverse_column_permute (const Array< int64 > *permutation_indices) const override

  Returns a LinOp representing the row permutation of the inverse permuted object.
- void inverse_column_permute (const Array< int32 > *permutation_indices, Dense *output) const

  Writes the inverse column-permuted matrix into the given output matrix.
- void inverse_column_permute (const Array< int64 > *permutation_indices, Dense *output) const
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override

  Extracts the diagonal entries of the matrix into a vector.
- void extract_diagonal (Diagonal< ValueType > *output) const

  Writes the diagonal of this matrix into an existing diagonal matrix.
- std::unique_ptr< absolute_type > compute_absolute () const override

  Gets the AbsoluteLinOp.
- void compute_absolute (absolute_type *output) const

  Writes the absolute values of this matrix into an existing matrix.
- void compute_absolute_inplace () override

  Compute absolute inplace on each element.
- std::unique_ptr< complex_type > make_complex () const

  Creates a complex copy of the original matrix.
- void make_complex (complex_type *result) const

  Writes a complex copy of the original matrix to a given complex matrix.
- std::unique_ptr< real_type > get_real () const

  Creates a new real matrix and extracts the real part of the original matrix into that.
- void get_real (real_type *result) const

  Extracts the real part of the original matrix into a given real matrix.
- std::unique_ptr< real_type > get_imag () const

  Creates a new real matrix and extracts the imaginary part of the original matrix into that.
- void get_imag (real_type *result) const

  Extracts the imaginary part of the original matrix into a given real matrix.
- value_type * get_values () noexcept

  Returns a pointer to the array of values of the matrix.
- const value_type * get_const_values () const noexcept

  Returns a pointer to the array of values of the matrix.
- size_type get_stride () const noexcept

  Returns the stride of the matrix.
- size_type get_num_stored_elements () const noexcept

Returns the number of elements explicitly stored in the matrix.

- value_type & at (size_type row, size_type col) noexcept

    Returns a single element of the matrix.

- value_type at (size_type row, size_type col) const noexcept

    Returns a single element of the matrix.

- ValueType & at (size_type idx) noexcept

    Returns a single element of the matrix.

- ValueType at (size_type idx) const noexcept

    Returns a single element of the matrix.

- void scale (const LinOp ∗alpha)

    Scales the matrix with a scalar (aka: BLAS scal).

- void inv_scale (const LinOp ∗alpha)

    Scales the matrix with the inverse of a scalar.

- void add_scaled (const LinOp ∗alpha, const LinOp ∗b)

    Adds `b` scaled by `alpha` to the matrix (aka: BLAS axpy).

- void sub_scaled (const LinOp ∗alpha, const LinOp ∗b)

    Subtracts `b` scaled by `alpha` fron the matrix (aka: BLAS axpy).

- void compute_dot (const LinOp ∗b, LinOp ∗result) const

    Computes the column-wise dot product of this matrix and `b`.

- void compute_conj_dot (const LinOp ∗b, LinOp ∗result) const

    Computes the column-wise dot product of `conj(this matrix)` and `b`.

- void compute_norm2 (LinOp ∗result) const

    Computes the column-wise Euclidian (L^2) norm of this matrix.

- void compute_norm1 (LinOp ∗result) const

    Computes the column-wise (L^1) norm of this matrix.

- std::unique_ptr< Dense > create_submatrix (const span &rows, const span &columns, const size_type stride)

    Create a submatrix from the original matrix.

- std::unique_ptr< Dense > create_submatrix (const span &rows, const span &columns)

    Create a submatrix from the original matrix.

- std::unique_ptr< Dense< remove_complex< ValueType > > > create_real_view ()

    Create a real view of the (potentially) complex original matrix.

- std::unique_ptr< const Dense< remove_complex< ValueType > > > create_real_view () const

    Create a real view of the (potentially) complex original matrix.

Static Public Member Functions

- static std::unique_ptr< Dense > create_with_config_of (const Dense ∗other)

    Creates a Dense matrix with the same size and stride as another Dense matrix.

- static std::unique_ptr< Dense > create_with_type_of (const Dense ∗other, std::shared_ptr< const Executor > exec, const dim< 2 > &size=dim< 2 >{})

    Creates a Dense matrix with the same type and executor as another Dense matrix but a different size.

- static std::unique_ptr< Dense > create_with_type_of (const Dense ∗other, std::shared_ptr< const Executor > exec, const dim< 2 > &size, size_type stride)
- static std::unique_ptr< const Dense > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< ValueType > &&values, size_type stride)

    Creates a constant (immutable) Dense matrix from a constant array.

41.58.1 Detailed Description

template<typename ValueType = default_precision>
class gko::matrix::Dense< ValueType >

Dense is a matrix format which explicitly stores all values of the matrix.

The values are stored in row-major format (values belonging to the same row appear consecutive in the memory). Optionally, rows can be padded for better memory access.

Template Parameters

ValueType	precision of matrix elements

Note

While this format is not very useful for storing sparse matrices, it is often suitable to store vectors, and sets of vectors.

41.58.2 Member Function Documentation

41.58.2.1 add_scaled()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::add_scaled (
            const LinOp * alpha,
            const LinOp * b ) [inline]
```

Adds `b` scaled by `alpha` to the matrix (aka: BLAS axpy).

Parameters

alpha	If alpha is 1x1 Dense matrix, the entire matrix is scaled by alpha. If it is a Dense row vector of values, then i-th column of the matrix is scaled with the i-th element of alpha (the number of columns of alpha has to match the number of columns of the matrix).
b	a matrix of the same dimension as this

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.58.2.2 at() [1/4]

```
template<typename ValueType = default_precision>
ValueType gko::matrix::Dense< ValueType >::at (
            size_type idx ) const [inline], [noexcept]
```

Returns a single element of the matrix.

Useful for iterating across all elements of the matrix. However, it is less efficient than the two-parameter variant of this method.

Parameters

idx	a linear index of the requested element (ignoring the stride)

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.58.2.3 at() [2/4]

```
template<typename ValueType = default_precision>
ValueType& gko::matrix::Dense< ValueType >::at (
            size_type idx ) [inline], [noexcept]
```

Returns a single element of the matrix.

Useful for iterating across all elements of the matrix. However, it is less efficient than the two-parameter variant of this method.

Parameters

idx	a linear index of the requested element (ignoring the stride)

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_data().

41.58.2.4 at() [3/4]

```
template<typename ValueType = default_precision>
value_type gko::matrix::Dense< ValueType >::at (
            size_type row,
            size_type col ) const [inline], [noexcept]
```

Returns a single element of the matrix.

Parameters

row	the row of the requested element
col	the column of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.58.2.5 at() [4/4]

```
template<typename ValueType = default_precision>
value_type& gko::matrix::Dense< ValueType >::at (
            size_type row,
            size_type col ) [inline], [noexcept]
```

Returns a single element of the matrix.

Parameters

row	the row of the requested element
col	the column of the requested element

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_data().

Referenced by gko::initialize().

41.58.2.6 column_permute() [1/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::column_permute (
            const Array< int32 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the column permutation of the Permutable object.

In the resulting LinOp, the column `i` contains the input column `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new column permuted object

Implements gko::Permutable< int32 >.

41.58.2.7 column_permute() [2/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::column_permute (
            const Array< int32 > * permutation_indices,
            Dense< ValueType > * output ) const
```

Writes the column-permuted matrix into the given output matrix.

Parameters

permutation_indices	The array containing permutation indices. It must have `this->get_size()[1]` elements.
output	The output matrix. It must have the dimensions `this->get_size()`

See also

Dense::column_permute(const Array<int32>*)

41.58.2.8 column_permute() [3/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::column_permute (
            const Array< int64 > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the column permutation of the Permutable object.

In the resulting LinOp, the column `i` contains the input column `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new column permuted object

Implements gko::Permutable< int64 >.

41.58.2.9 column_permute() [4/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::column_permute (
            const Array< int64 > * permutation_indices,
            Dense< ValueType > * output ) const
```

41.58.2.10 compute_absolute() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<absolute_type> gko::matrix::Dense< ValueType >::compute_absolute ( ) const
[override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Dense< ValueType > > >.

41.58.2.11 compute_absolute() [2/2]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::compute_absolute (
            absolute_type * output ) const
```

Writes the absolute values of this matrix into an existing matrix.

Parameters

output	The output matrix. Its size must match the size of this matrix.

See also

Dense::compute_absolute()

41.58.2.12 compute_conj_dot()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::compute_conj_dot (
            const LinOp * b,
            LinOp * result ) const [inline]
```

Computes the column-wise dot product of `conj(this matrix)` and `b`.

Parameters

b	a Dense matrix of same dimension as this
result	a Dense row vector, used to store the dot product (the number of column in the vector must match the number of columns of this)

References gko::PolymorphicObject::get_executor(), gko::make_temporary_clone(), and gko::make_temporary_↩
output_clone().

41.58.2.13 compute_dot()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::compute_dot (
            const LinOp * b,
            LinOp * result ) const [inline]
```

Computes the column-wise dot product of this matrix and `b`.

Parameters

b	a Dense matrix of same dimension as this
result	a Dense row vector, used to store the dot product (the number of column in the vector must match the number of columns of this)

References gko::PolymorphicObject::get_executor(), gko::make_temporary_clone(), and gko::make_temporary_↩
output_clone().

41.58.2.14 compute_norm1()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::compute_norm1 (
            LinOp * result ) const [inline]
```

Computes the column-wise (L^1) norm of this matrix.

Parameters

result	a Dense row vector, used to store the norm (the number of columns in the vector must match the number of columns of this)

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_output_clone().

41.58.2.15 compute_norm2()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::compute_norm2 (
            LinOp * result ) const  [inline]
```

Computes the column-wise Euclidian (L^2) norm of this matrix.

Parameters

result	a Dense row vector, used to store the norm (the number of columns in the vector must match the number of columns of this)

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_output_clone().

41.58.2.16 conj_transpose() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.58.2.17 conj_transpose() [2/2]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::conj_transpose (
            Dense< ValueType > * output ) const
```

Writes the conjugate-transposed matrix into the given output matrix.

Parameters

output	The output matrix. It must have the dimensions `gko::transpose(this->get_size())`

41.58.2.18 create_const()

```
template<typename ValueType = default_precision>
static std::unique_ptr<const Dense> gko::matrix::Dense< ValueType >::create_const (
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            gko::detail::ConstArrayView< ValueType > && values,
            size_type stride ) [inline], [static]
```

Creates a constant (immutable) Dense matrix from a constant array.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
stride	the row-stride of the matrix

Returns

A smart pointer to the constant matrix wrapping the input array (if it resides on the same executor as the matrix) or a copy of the array on the correct executor.

41.58.2.19 create_real_view() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<Dense<remove_complex<ValueType> > > gko::matrix::Dense< ValueType >::create↩
_real_view ( ) [inline]
```

Create a real view of the (potentially) complex original matrix.

If the original matrix is real, nothing changes. If the original matrix is complex, the result is created by viewing the complex matrix with as real with a reinterpret_cast with twice the number of columns and double the stride.

References gko::PolymorphicObject::get_executor(), gko::matrix::Dense< ValueType >::get_stride(), and gko↩ ::matrix::Dense< ValueType >::get_values().

41.58.2.20 create_real_view() [2/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<const Dense<remove_complex<ValueType> > > gko::matrix::Dense< ValueType >↩
::create_real_view ( ) const  [inline]
```

Create a real view of the (potentially) complex original matrix.

If the original matrix is real, nothing changes. If the original matrix is complex, the result is created by viewing the complex matrix with as real with a reinterpret_cast with twice the number of columns and double the stride.

References gko::matrix::Dense< ValueType >::get_const_values(), gko::PolymorphicObject::get_executor(), and gko::matrix::Dense< ValueType >::get_stride().

41.58.2.21 create_submatrix() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::create_submatrix (
            const span & rows,
            const span & columns )  [inline]
```

Create a submatrix from the original matrix.

Parameters

rows	row span
columns	column span

References gko::matrix::Dense< ValueType >::create_submatrix(), and gko::matrix::Dense< ValueType >::get_↩
stride().

41.58.2.22 create_submatrix() [2/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::create_submatrix (
            const span & rows,
            const span & columns,
            const size_type stride )  [inline]
```

Create a submatrix from the original matrix.

Warning: defining stride for this create_submatrix method might cause wrong memory access. Better use the create_submatrix(rows, columns) method instead.

Parameters

rows	row span
columns	column span
stride	stride of the new submatrix.

Referenced by gko::matrix::Dense< ValueType >::create_submatrix().

41.58.2.23 create_with_config_of()

```
template<typename ValueType = default_precision>
static std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::create_with_config_of (
            const Dense< ValueType > * other ) [inline], [static]
```

Creates a Dense matrix with the same size and stride as another Dense matrix.

Parameters

other	The other matrix whose configuration needs to copied.

41.58.2.24 create_with_type_of() [1/2]

```
template<typename ValueType = default_precision>
static std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::create_with_type_of (
            const Dense< ValueType > * other,
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            size_type stride ) [inline], [static]
```

Parameters

stride	The stride of the new matrix.

Note

This is an overload which allows full parameter specification.

41.58.2.25 create_with_type_of() [2/2]

```
template<typename ValueType = default_precision>
static std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::create_with_type_of (
            const Dense< ValueType > * other,
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size = dim<2>{} ) [inline], [static]
```

Creates a Dense matrix with the same type and executor as another Dense matrix but a different size.

Parameters

other	The other matrix whose type we target.
exec	The executor of the new matrix.
size	The size of the new matrix.
stride	The stride of the new matrix.

Returns

a Dense matrix with the type of other.

41.58.2.26 extract_diagonal() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Dense< ValueType >::extract_diagonal ( )
const [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.58.2.27 extract_diagonal() [2/2]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::extract_diagonal (
            Diagonal< ValueType > * output ) const
```

Writes the diagonal of this matrix into an existing diagonal matrix.

Parameters

output	The output matrix. Its size must match the size of this matrix's diagonal.

See also

Dense::extract_diagonal()

41.58.2.28 fill()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::fill (
            const ValueType value )
```

Fill the dense matrix with a given value.

Parameters

value	the value to be filled

41.58.2.29 get_const_values()

```
template<typename ValueType = default_precision>
const value_type* gko::matrix::Dense< ValueType >::get_const_values ( ) const [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::Dense< ValueType >::create_real_view().

41.58.2.30 get_num_stored_elements()

```
template<typename ValueType = default_precision>
size_type gko::matrix::Dense< ValueType >::get_num_stored_elements ( ) const [inline], [noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.58.2.31 get_stride()

```
template<typename ValueType = default_precision>
size_type gko::matrix::Dense< ValueType >::get_stride ( ) const  [inline], [noexcept]
```

Returns the stride of the matrix.

Returns

the stride of the matrix.

Referenced by gko::matrix::Dense< ValueType >::create_real_view(), and gko::matrix::Dense< ValueType >↩ ::create_submatrix().

41.58.2.32 get_values()

```
template<typename ValueType = default_precision>
value_type* gko::matrix::Dense< ValueType >::get_values ( )  [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

References gko::Array< ValueType >::get_data().

Referenced by gko::matrix::Dense< ValueType >::create_real_view().

41.58.2.33 inv_scale()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inv_scale (
            const LinOp * alpha )  [inline]
```

Scales the matrix with the inverse of a scalar.

Parameters

alpha	If alpha is 1x1 Dense matrix, the entire matrix is scaled by 1 / alpha. If it is a Dense row vector of values, then i-th column of the matrix is scaled with the inverse of the i-th element of alpha (the number of columns of alpha has to match the number of columns of the matrix).

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.58.2.34 inverse_column_permute() [1/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::inverse_column_permute (
            const Array< int32 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the column `perm[i]` contains the input column `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implements gko::Permutable< int32 >.

41.58.2.35 inverse_column_permute() [2/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inverse_column_permute (
            const Array< int32 > * permutation_indices,
            Dense< ValueType > * output ) const
```

Writes the inverse column-permuted matrix into the given output matrix.

Parameters

permutation_indices	The array containing permutation indices. It must have `this->get_size()[1]` elements.
output	The output matrix. It must have the dimensions `this->get_size()`

See also

Dense::inverse_column_permute(const Array<int32>*)

41.58.2.36 inverse_column_permute() [3/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::inverse_column_permute (
            const Array< int64 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the column `perm[i]` contains the input column `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implements gko::Permutable< int64 >.

41.58.2.37 inverse_column_permute() [4/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inverse_column_permute (
            const Array< int64 > * permutation_indices,
            Dense< ValueType > * output ) const
```

41.58.2.38 inverse_permute() [1/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::inverse_permute (
            const Array< int32 > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(perm[i],perm[j])` contains the input value `(i,j)`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented from gko::Permutable< int32 >.

41.58.2.39 inverse_permute() [2/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inverse_permute (
            const Array< int32 > * permutation_indices,
            Dense< ValueType > * output ) const
```

Writes the inverse symmetrically permuted matrix into the given output matrix.

Parameters

permutation_indices	The array containing permutation indices. It must have `this->get_size()[0]` elements.
output	The output matrix. It must have the dimensions `this->get_size()`

See also

Dense::inverse_permute(const Array<int32>*)

41.58.2.40 inverse_permute() [3/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::inverse_permute (
            const Array< int64 > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(perm[i],perm[j])` contains the input value `(i,j)`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented from gko::Permutable< int64 >.

41.58.2.41 inverse_permute() [4/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inverse_permute (
            const Array< int64 > * permutation_indices,
            Dense< ValueType > * output ) const
```

41.58.2.42 inverse_row_permute() [1/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::inverse_row_permute (
            const Array< int32 > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the row `perm[i]` contains the input row `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implements gko::Permutable< int32 >.

41.58.2.43 inverse_row_permute() [2/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inverse_row_permute (
            const Array< int32 > * permutation_indices,
            Dense< ValueType > * output ) const
```

Writes the inverse row-permuted matrix into the given output matrix.

Parameters

permutation_indices	The array containing permutation indices. It must have `this->get_size()[0]` elements.
output	The output matrix. It must have the dimensions `this->get_size()`

See also

Dense::inverse_row_permute(const Array<int32>*)

41.58.2.44 inverse_row_permute() [3/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::inverse_row_permute (
            const Array< int64 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the row `perm[i]` contains the input row `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implements gko::Permutable< int64 >.

41.58.2.45 inverse_row_permute() [4/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::inverse_row_permute (
            const Array< int64 > * permutation_indices,
            Dense< ValueType > * output ) const
```

41.58.2.46 make_complex() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<complex_type> gko::matrix::Dense< ValueType >::make_complex ( ) const
```

Creates a complex copy of the original matrix.

If the original matrix was real, the imaginary part of the result will be zero.

41.58.2.47 make_complex() [2/2]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::make_complex (
            complex_type * result ) const
```

Writes a complex copy of the original matrix to a given complex matrix.

If the original matrix was real, the imaginary part of the result will be zero.

41.58.2.48 permute() [1/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::permute (
            const Array< int32 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the symmetric row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(i,j)` contains the input value `(perm[i],perm[j])`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented from gko::Permutable< int32 >.

41.58.2.49 permute() [2/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::permute (
            const Array< int32 > * permutation_indices,
            Dense< ValueType > * output ) const
```

Writes the symmetrically permuted matrix into the given output matrix.

Parameters

permutation_indices	The array containing permutation indices. It must have `this->get_size()[0]` elements.
output	The output matrix. It must have the dimensions `this->get_size()`

See also

Dense::permute(const Array<int32>*)

41.58.2.50 permute() [3/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::permute (
            const Array< int64 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the symmetric row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(i,j)` contains the input value `(perm[i],perm[j])`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented from gko::Permutable< int64 >.

41.58.2.51 permute() [4/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::permute (
            const Array< int64 > * permutation_indices,
            Dense< ValueType > * output ) const
```

41.58.2.52 row_gather() [1/8]

```
template<typename ValueType = default_precision>
std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::row_gather (
            const Array< int32 > * gather_indices ) const
```

Create a Dense matrix consisting of the given rows from this matrix.

Parameters

gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.

Returns

Dense matrix on the same executor with the same number of columns and `gather_indices->get_num_elems()` rows containing the gathered rows from this matrix: `output(i,j) = input(gather_indices(i), j)`

41.58.2.53 row_gather() [2/8]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_gather (
            const Array< int32 > * gather_indices,
            Dense< ValueType > * row_collection ) const
```

Copies the given rows from this matrix into `row_collection`

Parameters

gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.
row_collection	pointer to a Dense matrix that will store the gathered rows: `row_collection(i,j) = input(gather_indices(i), j)` It must have the same number of columns as this matrix and `gather_indices->get_num_elems()` rows.

41.58.2.54 row_gather() [3/8]

```
template<typename ValueType = default_precision>
```

```
void gko::matrix::Dense< ValueType >::row_gather (
            const Array< int32 > * gather_indices,
            LinOp * row_collection ) const
```

Copies the given rows from this matrix into `row_collection`

Parameters

gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.
row_collection	pointer to a LinOp that will store the gathered rows: `row_collection(i,j) = input(gather_indices(i), j)` It must have the same number of columns as this matrix and `gather_indices->get_num_elems()` rows.

41.58.2.55 row_gather() [4/8]

```
template<typename ValueType = default_precision>
std::unique_ptr<Dense> gko::matrix::Dense< ValueType >::row_gather (
            const Array< int64 > * gather_indices ) const
```

Create a Dense matrix consisting of the given rows from this matrix.

Parameters

gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.

Returns

Dense matrix on the same executor with the same number of columns and `gather_indices->get_num_elems()` rows containing the gathered rows from this matrix: `output(i,j) = input(gather_indices(i), j)`

41.58.2.56 row_gather() [5/8]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_gather (
            const Array< int64 > * gather_indices,
            Dense< ValueType > * row_collection ) const
```

Copies the given rows from this matrix into `row_collection`

Parameters

gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.
row_collection	pointer to a Dense matrix that will store the gathered rows: `row_collection(i,j) = input(gather_indices(i), j)` It must have the same number of columns as this matrix and `gather_indices->get_num_elems()` rows.

41.58.2.57 row_gather() [6/8]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_gather (
            const Array< int64 > * gather_indices,
            LinOp * row_collection ) const
```

Copies the given rows from this matrix into `row_collection`

Parameters

gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.
row_collection	pointer to a LinOp that will store the gathered rows: `row_collection(i,j) = input(gather_indices(i), j)` It must have the same number of columns as this matrix and `gather_indices->get_num_elems()` rows.

41.58.2.58 row_gather() [7/8]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_gather (
            const LinOp * alpha,
            const Array< int32 > * gather_indices,
            const LinOp * beta,
            LinOp * row_collection ) const
```

Copies the given rows from this matrix into `row_collection` with scaling.

Parameters

alpha	scaling the result of row gathering
gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.
beta	scaling the input row_collection
row_collection	pointer to a LinOp that will store the gathered rows: `row_collection(i,j) = input(gather_indices(i), j)` It must have the same number of columns as this matrix and `gather_indices->get_num_elems()` rows.

41.58.2.59 row_gather() [8/8]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_gather (
            const LinOp * alpha,
            const Array< int64 > * gather_indices,
            const LinOp * beta,
            LinOp * row_collection ) const
```

Parameters

alpha	scaling the result of row gathering
gather_indices	pointer to an array containing row indices from this matrix. It may contain duplicates.
beta	scaling the input row_collection
row_collection	pointer to a LinOp that will store the gathered rows: `row_collection(i,j) = input(gather_indices(i), j)` It must have the same number of columns as this matrix and `gather_indices->get_num_elems()` rows.

41.58.2.60 row_permute() [1/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::row_permute (
            const Array< int32 > * permutation_indices ) const  [override], [virtual]
```

Returns a LinOp representing the row permutation of the Permutable object.

In the resulting LinOp, the row `i` contains the input row `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Implements gko::Permutable< int32 >.

41.58.2.61 row_permute() [2/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_permute (
            const Array< int32 > * permutation_indices,
            Dense< ValueType > * output ) const
```

Writes the row-permuted matrix into the given output matrix.

Parameters

permutation_indices	The array containing permutation indices. It must have `this->get_size()[0]` elements.
output	The output matrix. It must have the dimensions `this->get_size()`

See also

Dense::row_permute(const Array<int32>*)

41.58.2.62 row_permute() [3/4]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::row_permute (
            const Array< int64 > * permutation_indices ) const [override], [virtual]
```

Returns a LinOp representing the row permutation of the Permutable object.

In the resulting LinOp, the row `i` contains the input row `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Implements gko::Permutable< int64 >.

41.58.2.63 row_permute() [4/4]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::row_permute (
            const Array< int64 > * permutation_indices,
            Dense< ValueType > * output ) const
```

41.58.2.64 scale()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::scale (
            const LinOp * alpha ) [inline]
```

Scales the matrix with a scalar (aka: BLAS scal).

Parameters

alpha	If alpha is 1x1 Dense matrix, the entire matrix is scaled by alpha. If it is a Dense row vector of values, then i-th column of the matrix is scaled with the i-th element of alpha (the number of columns of alpha has to match the number of columns of the matrix).

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.58.2.65 sub_scaled()

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::sub_scaled (
            const LinOp * alpha,
            const LinOp * b ) [inline]
```

Subtracts b scaled by alpha fron the matrix (aka: BLAS axpy).

Parameters

alpha	If alpha is 1x1 Dense matrix, b is scaled by alpha. If it is a Dense row vector of values, then i-th column of b is scaled with the i-th element of alpha (the number of columns of alpha has to match the number of columns of the matrix).
b	a matrix of the same dimension as this

References gko::PolymorphicObject::get_executor(), and gko::make_temporary_clone().

41.58.2.66 transpose() [1/2]

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Dense< ValueType >::transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.58.2.67 transpose() [2/2]

```
template<typename ValueType = default_precision>
void gko::matrix::Dense< ValueType >::transpose (
            Dense< ValueType > * output ) const
```

Writes the transposed matrix into the given output matrix.

Parameters

output	The output matrix. It must have the dimensions `gko::transpose(this->get_size())`

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

- ginkgo/core/base/mtx_io.hpp
- ginkgo/core/matrix/dense.hpp

41.59 gko::device_matrix_data< ValueType, IndexType > Class Template Reference

This type is a device-side equivalent to matrix_data.

`#include <ginkgo/core/base/device_matrix_data.hpp>`

Classes

- struct arrays

  Stores the internal arrays of a device_matrix_data object.

Public Member Functions

- device_matrix_data (std::shared_ptr< const Executor > exec, dim< 2 > size={}, size_type num_entries=0)

  Initializes a new device_matrix_data object.
- device_matrix_data (std::shared_ptr< const Executor > exec, const device_matrix_data &data)

  Initializes a device_matrix_data object by copying an existing object on another executor.
- template<typename ValueArray , typename RowIndexArray , typename ColIndexArray >
  device_matrix_data (std::shared_ptr< const Executor > exec, dim< 2 > size, RowIndexArray &&row_idxs, ColIndexArray &&col_idxs, ValueArray &&values)

  Initializes a new device_matrix_data object from existing data.
- host_type copy_to_host () const

  Copies the device_matrix_data entries to the host to return a regular matrix_data object with the same dimensions and entries.
- void sort_row_major ()

  Sorts the matrix entries in row-major order This means that they will be sorted by row index first, and then by column index inside each row.
- void remove_zeros ()

  Removes all zero entries from the storage.
- void sum_duplicates ()

  Sums up all duplicate entries pointing to the same non-zero location.
- std::shared_ptr< const Executor > get_executor () const

  Returns the executor used to store the device_matrix_data entries.
- dim< 2 > get_size () const

  Returns the dimensions of the matrix.
- size_type get_num_elems () const

  Returns the number of stored elements of the matrix.

- index_type ∗ get_row_idxs ()
  Returns a pointer to the row index array.
- const index_type ∗ get_const_row_idxs () const
  Returns a pointer to the constant row index array.
- index_type ∗ get_col_idxs ()
  Returns a pointer to the column index array.
- const index_type ∗ get_const_col_idxs () const
  Returns a pointer to the constant column index array.
- value_type ∗ get_values ()
  Returns a pointer to the value array.
- const value_type ∗ get_const_values () const
  Returns a pointer to the constant value array.
- void resize_and_reset (size_type new_num_entries)
  Resizes the internal storage to the given number of stored matrix entries.
- void resize_and_reset (dim< 2 > new_size, size_type new_num_entries)
  Resizes the matrix and internal storage to the given dimensions.
- arrays empty_out ()
  Moves out the internal arrays of the device_matrix_data object and resets it to an empty 0x0 matrix.

Static Public Member Functions

- static device_matrix_data create_from_host (std::shared_ptr< const Executor > exec, const host_type &data)
  Creates a device_matrix_data object from the given host data on the given executor.

41.59.1 Detailed Description

template<typename ValueType, typename IndexType>
class gko::device_matrix_data< ValueType, IndexType >

This type is a device-side equivalent to matrix_data.

It stores the data necessary to initialize any matrix format in Ginkgo in individual value, column and row index arrays together with associated matrix dimensions. matrix_data uses Array-of-Structs storage (AoS), while device_matrix_data uses Struct-of-Arrays (SoA).

Note

To be used with a Ginkgo matrix type, the entry array must be sorted in row-major order, i.e. by row index, then by column index within rows. This can be achieved by calling the sort_row_major function.

The data must not contain any duplicate (row, column) pairs.

Template Parameters

ValueType	the type used to store matrix values
IndexType	the type used to store matrix row and column indices

41.59.2 Constructor & Destructor Documentation

41.59.2.1 device_matrix_data() [1/3]

```
template<typename ValueType, typename IndexType>
gko::device_matrix_data< ValueType, IndexType >::device_matrix_data (
            std::shared_ptr< const Executor > exec,
            dim< 2 > size = {},
            size_type num_entries = 0 )  [explicit]
```

Initializes a new device_matrix_data object.

It uses the given executor to allocate storage for the given number of entries and matrix dimensions.

Parameters

exec	the executor to be used to store the matrix entries
size	the matrix dimensions
num_entries	the number of entries to be stored

41.59.2.2 device_matrix_data() [2/3]

```
template<typename ValueType, typename IndexType>
gko::device_matrix_data< ValueType, IndexType >::device_matrix_data (
            std::shared_ptr< const Executor > exec,
            const device_matrix_data< ValueType, IndexType > & data )
```

Initializes a device_matrix_data object by copying an existing object on another executor.

Parameters

exec	the executor to be used to store the matrix entries
data	the device_matrix data object to copy, potentially stored on another executor.

41.59.2.3 device_matrix_data() [3/3]

```
template<typename ValueType, typename IndexType>
template<typename ValueArray , typename RowIndexArray , typename ColIndexArray >
gko::device_matrix_data< ValueType, IndexType >::device_matrix_data (
            std::shared_ptr< const Executor > exec,
            dim< 2 > size,
            RowIndexArray && row_idxs,
```

```
            ColIndexArray && col_idxs,
            ValueArray && values )  [inline]
```

Initializes a new device_matrix_data object from existing data.

Parameters

size	the matrix dimensions
values	the array containing the matrix values
col_idxs	the array containing the matrix column indices
row_idxs	the array containing the matrix row indices

```
        : size_{size},
          row_idxs_{exec, std::forward<RowIndexArray>(row_idxs)},
          col_idxs_{exec, std::forward<ColIndexArray>(col_idxs)},
          values_{exec, std::forward<ValueArray>(values)}
    {
        GKO_ASSERT_EQ(values_.get_num_elems(), row_idxs_.get_num_elems());
        GKO_ASSERT_EQ(values_.get_num_elems(), col_idxs_.get_num_elems());
    }
```

41.59.3 Member Function Documentation

41.59.3.1 copy_to_host()

```
template<typename ValueType, typename IndexType>
host_type gko::device_matrix_data< ValueType, IndexType >::copy_to_host ( ) const
```

Copies the device_matrix_data entries to the host to return a regular matrix_data object with the same dimensions and entries.

Returns

a matrix_data object with the same dimensions and entries.

Referenced by gko::ReadableFromMatrixData< ValueType, int32 >::read().

41.59.3.2 create_from_host()

```
template<typename ValueType, typename IndexType>
static device_matrix_data gko::device_matrix_data< ValueType, IndexType >::create_from_host (
            std::shared_ptr< const Executor > exec,
            const host_type & data )  [static]
```

Creates a device_matrix_data object from the given host data on the given executor.

Parameters

exec	the executor to create the device_matrix_data on.
data	the data to be wrapped or copied into a device_matrix_data.

Returns

a device_matrix_data object with the same size and entries as `data` copied to the device executor.

41.59.3.3 empty_out()

```
template<typename ValueType, typename IndexType>
arrays gko::device_matrix_data< ValueType, IndexType >::empty_out ( )
```

Moves out the internal arrays of the device_matrix_data object and resets it to an empty 0x0 matrix.

Returns

a struct containing the internal arrays.

41.59.3.4 get_col_idxs()

```
template<typename ValueType, typename IndexType>
index_type* gko::device_matrix_data< ValueType, IndexType >::get_col_idxs ( ) [inline]
```

Returns a pointer to the column index array.

Returns

a pointer to the column index array

References gko::Array< ValueType >::get_data().

41.59.3.5 get_const_col_idxs()

```
template<typename ValueType, typename IndexType>
const index_type* gko::device_matrix_data< ValueType, IndexType >::get_const_col_idxs ( )
const [inline]
```

Returns a pointer to the constant column index array.

Returns

a pointer to the constant column index array

References gko::Array< ValueType >::get_const_data().

41.59.3.6 get_const_row_idxs()

```
template<typename ValueType, typename IndexType>
const index_type* gko::device_matrix_data< ValueType, IndexType >::get_const_row_idxs ( )
const [inline]
```

Returns a pointer to the constant row index array.

Returns

a pointer to the constant row index array

References gko::Array< ValueType >::get_const_data().

41.59.3.7 get_const_values()

```
template<typename ValueType, typename IndexType>
const value_type* gko::device_matrix_data< ValueType, IndexType >::get_const_values ( ) const
[inline]
```

Returns a pointer to the constant value array.

Returns

a pointer to the constant value array

References gko::Array< ValueType >::get_const_data().

41.59.3.8 get_executor()

```
template<typename ValueType, typename IndexType>
std::shared_ptr<const Executor> gko::device_matrix_data< ValueType, IndexType >::get_executor
( ) const [inline]
```

Returns the executor used to store the device_matrix_data entries.

Returns

the executor used to store the device_matrix_data entries.

References gko::Array< ValueType >::get_executor().

41.59.3.9 get_num_elems()

```
template<typename ValueType, typename IndexType>
size_type gko::device_matrix_data< ValueType, IndexType >::get_num_elems ( ) const [inline]
```

Returns the number of stored elements of the matrix.

Returns

the number of stored elements of the matrix.

References gko::Array< ValueType >::get_num_elems().

41.59.3.10 get_row_idxs()

```
template<typename ValueType, typename IndexType>
index_type* gko::device_matrix_data< ValueType, IndexType >::get_row_idxs ( ) [inline]
```

Returns a pointer to the row index array.

Returns

a pointer to the row index array

References gko::Array< ValueType >::get_data().

41.59.3.11 get_size()

```
template<typename ValueType, typename IndexType>
dim<2> gko::device_matrix_data< ValueType, IndexType >::get_size ( ) const [inline]
```

Returns the dimensions of the matrix.

Returns

the dimensions of the matrix.

41.59.3.12 get_values()

```
template<typename ValueType, typename IndexType>
value_type* gko::device_matrix_data< ValueType, IndexType >::get_values ( ) [inline]
```

Returns a pointer to the value array.

Returns

a pointer to the value array

References gko::Array< ValueType >::get_data().

41.59.3.13 remove_zeros()

```
template<typename ValueType, typename IndexType>
void gko::device_matrix_data< ValueType, IndexType >::remove_zeros ( )
```

Removes all zero entries from the storage.

This does not modify the storage if there are no zero entries, and keeps the relative order of nonzero entries otherwise.

41.59.3.14 resize_and_reset() [1/2]

```
template<typename ValueType, typename IndexType>
void gko::device_matrix_data< ValueType, IndexType >::resize_and_reset (
            dim< 2 > new_size,
            size_type new_num_entries )
```

Resizes the matrix and internal storage to the given dimensions.

The resulting storage should be assumed uninitialized.

Parameters

new_size	the new matrix dimensions.
new_num_entries	the new number of stored matrix entries.

41.59.3.15 resize_and_reset() [2/2]

```
template<typename ValueType, typename IndexType>
void gko::device_matrix_data< ValueType, IndexType >::resize_and_reset (
            size_type new_num_entries )
```

Resizes the internal storage to the given number of stored matrix entries.

The resulting storage should be assumed uninitialized.

Parameters

new_num_entries	the new number of stored matrix entries.

41.59.3.16 sum_duplicates()

```
template<typename ValueType, typename IndexType>
void gko::device_matrix_data< ValueType, IndexType >::sum_duplicates ( )
```

Sums up all duplicate entries pointing to the same non-zero location.

The output will be sorted in row-major order, and it will only reallocate if duplicates exist.

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

- ginkgo/core/base/device_matrix_data.hpp

41.60 gko::matrix::Diagonal< ValueType > Class Template Reference

This class is a utility which efficiently implements the diagonal matrix (a linear operator which scales a vector row wise).

`#include <ginkgo/core/matrix/diagonal.hpp>`

Public Member Functions

- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- std::unique_ptr< absolute_type > compute_absolute () const override
  Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override
  Compute absolute inplace on each element.
- value_type * get_values () noexcept
  Returns a pointer to the array of values of the matrix.
- const value_type * get_const_values () const noexcept
  Returns a pointer to the array of values of the matrix.
- void rapply (const LinOp *b, LinOp *x) const
  Applies the diagonal matrix from the right side to a matrix b, which means scales the columns of b with the according diagonal entries.

Static Public Member Functions

- static std::unique_ptr< const Diagonal > create_const (std::shared_ptr< const Executor > exec, size_type size, gko::detail::ConstArrayView< ValueType > &&values)
  Creates a constant (immutable) Diagonal matrix from a constant array.

41.60.1 Detailed Description

template<typename ValueType = default_precision>
class gko::matrix::Diagonal< ValueType >

This class is a utility which efficiently implements the diagonal matrix (a linear operator which scales a vector row wise).

Objects of the Diagonal class always represent a square matrix, and require one array to store their values.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes of a CSR matrix the diagonal is applied or converted to.

41.60.2 Member Function Documentation

41.60.2.1 compute_absolute()

```
template<typename ValueType = default_precision>
std::unique_ptr<absolute_type> gko::matrix::Diagonal< ValueType >::compute_absolute ( ) const
[override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Diagonal< ValueType > > >.

41.60.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Diagonal< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.60.2.3 create_const()

```
template<typename ValueType = default_precision>
static std::unique_ptr<const Diagonal> gko::matrix::Diagonal< ValueType >::create_const (
            std::shared_ptr< const Executor > exec,
            size_type size,
            gko::detail::ConstArrayView< ValueType > && values )  [inline], [static]
```

Creates a constant (immutable) Diagonal matrix from a constant array.

Parameters

exec	the executor to create the matrix on
size	the size of the square matrix
values	the value array of the matrix

Returns

A smart pointer to the constant matrix wrapping the input array (if it resides on the same executor as the matrix) or a copy of the array on the correct executor.

```
{
    // cast const-ness away, but return a const object afterwards,
    // so we can ensure that no modifications take place.
    return std::unique_ptr<const Diagonal>(new Diagonal{
        exec, size, gko::detail::array_const_cast(std::move(values))});
}
```

41.60.2.4 get_const_values()

```
template<typename ValueType = default_precision>
const value_type* gko::matrix::Diagonal< ValueType >::get_const_values ( ) const  [inline],
[noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.60.2.5 get_values()

```
template<typename ValueType = default_precision>
value_type* gko::matrix::Diagonal< ValueType >::get_values ( )  [inline], [noexcept]
```

Returns a pointer to the array of values of the matrix.

Returns

the pointer to the array of values

References gko::Array< ValueType >::get_data().

41.60.2.6 rapply()

```
template<typename ValueType = default_precision>
void gko::matrix::Diagonal< ValueType >::rapply (
            const LinOp * b,
            LinOp * x ) const  [inline]
```

Applies the diagonal matrix from the right side to a matrix b, which means scales the columns of b with the according diagonal entries.

Parameters

b	the input vector(s) on which the diagonal matrix is applied
x	the output vector(s) where the result is stored

41.60.2.7 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Diagonal< ValueType >::transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/base/lin_op.hpp
- ginkgo/core/matrix/diagonal.hpp

41.61 gko::DiagonalExtractable< ValueType > Class Template Reference

The diagonal of a LinOp implementing this interface can be extracted.

`#include <ginkgo/core/base/lin_op.hpp>`

Public Member Functions

- std::unique_ptr< LinOp > extract_diagonal_linop () const override
  Extracts the diagonal entries of the matrix into a vector.
- virtual std::unique_ptr< matrix::Diagonal< ValueType > > extract_diagonal () const =0
  Extracts the diagonal entries of the matrix into a vector.

41.61.1 Detailed Description

template<typename ValueType>
class gko::DiagonalExtractable< ValueType >

The diagonal of a LinOp implementing this interface can be extracted.

extract_diagonal extracts the elements whose col and row index are the same and stores the result in a min(nrows, ncols) x 1 dense matrix.

41.61.2 Member Function Documentation

41.61.2.1 extract_diagonal()

```
template<typename ValueType >
virtual std::unique_ptr<matrix::Diagonal<ValueType> > gko::DiagonalExtractable< ValueType
>::extract_diagonal ( ) const [pure virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Dense< ValueType >, gko::matrix::Hybrid< ValueType, Inde
gko::matrix::Fbcsr< ValueType, IndexType >, gko::matrix::Coo< ValueType, IndexType >, gko::matrix::Ell< ValueType, IndexType >
and gko::matrix::Sellp< ValueType, IndexType >.

41.61.2.2 extract_diagonal_linop()

```
template<typename ValueType >
std::unique_ptr<LinOp> gko::DiagonalExtractable< ValueType >::extract_diagonal_linop ( )
const [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Returns

linop the linop of diagonal format

Implements gko::DiagonalLinOpExtractable.

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

- ginkgo/core/base/lin_op.hpp

41.62 gko::DiagonalLinOpExtractable Class Reference

The diagonal of a LinOp can be extracted.

`#include <ginkgo/core/base/lin_op.hpp>`

Public Member Functions

- virtual std::unique_ptr< LinOp > extract_diagonal_linop () const =0

  Extracts the diagonal entries of the matrix into a vector.

41.62.1 Detailed Description

The diagonal of a LinOp can be extracted.

It will be implemented by DiagonalExtractable<ValueType>, so the class does not need to implement it. extract↩_diagonal_linop returns a linop which extracts the elements whose col and row index are the same and stores the result in a min(nrows, ncols) x 1 dense matrix.

41.62.2 Member Function Documentation

41.62.2.1 extract_diagonal_linop()

```
virtual std::unique_ptr<LinOp> gko::DiagonalLinOpExtractable::extract_diagonal_linop ( ) const
[pure virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Returns

linop the linop of diagonal format

Implemented in gko::DiagonalExtractable< ValueType >.

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

- ginkgo/core/base/lin_op.hpp

41.63 gko::dim< Dimensionality, DimensionType > Struct Template Reference

A type representing the dimensions of a multidimensional object.

```
#include <ginkgo/core/base/dim.hpp>
```

Public Member Functions

- constexpr dim (const dimension_type &size=dimension_type{})

    Creates a dimension object with all dimensions set to the same value.
- template<typename... Rest>
  constexpr dim (const dimension_type &first, const Rest &... rest)

    Creates a dimension object with the specified dimensions.
- constexpr const dimension_type & operator[ ] (const size_type &dimension) const noexcept

    Returns the requested dimension.
- dimension_type & operator[ ] (const size_type &dimension) noexcept
- constexpr operator bool () const

    Checks if all dimensions evaluate to true.

Friends

- constexpr friend bool operator== (const dim &x, const dim &y)

    Checks if two dim objects are equal.
- constexpr friend dim operator* (const dim &x, const dim &y)

    Multiplies two dim objects.
- std::ostream & operator<< (std::ostream &os, const dim &x)

    A stream operator overload for dim.

41.63.1 Detailed Description

template<size_type Dimensionality, typename DimensionType = size_type>
struct gko::dim< Dimensionality, DimensionType >

A type representing the dimensions of a multidimensional object.

Template Parameters

Dimensionality	number of dimensions of the object
DimensionType	datatype used to represent each dimension

41.63.2 Constructor & Destructor Documentation

41.63.2.1 dim() [1/2]

```
template<size_type Dimensionality, typename DimensionType = size_type>
constexpr gko::dim< Dimensionality, DimensionType >::dim (
            const dimension_type & size = dimension_type{} )  [inline], [constexpr]
```

Creates a dimension object with all dimensions set to the same value.

Parameters

size	the size of each dimension

41.63.2.2 dim() [2/2]

```
template<size_type Dimensionality, typename DimensionType = size_type>
template<typename...  Rest>
constexpr gko::dim< Dimensionality, DimensionType >::dim (
            const dimension_type & first,
            const Rest &...  rest ) [inline], [constexpr]
```

Creates a dimension object with the specified dimensions.

If the number of dimensions given is less than the dimensionality of the object, the remaining dimensions are set to the same value as the last value given.

For example, in the context of matrices `dim<2>{2, 3}` creates the dimensions for a 2-by-3 matrix.

Parameters

first	first dimension
rest	other dimensions

41.63.3 Member Function Documentation

41.63.3.1 operator bool()

```
template<size_type Dimensionality, typename DimensionType = size_type>
constexpr gko::dim< Dimensionality, DimensionType >::operator bool ( ) const  [inline], [explicit],
[constexpr]
```

Checks if all dimensions evaluate to true.

For standard arithmetic types, this is equivalent to all dimensions being different than zero.

Returns

true if and only if all dimensions evaluate to true

Note

This operator is explicit to avoid implicit dim-to-int casts. It will still be used in contextual conversions (if, &&, ||, !)

41.63.3.2 operator[]() [1/2]

```
template<size_type Dimensionality, typename DimensionType = size_type>
constexpr const dimension_type& gko::dim< Dimensionality, DimensionType >::operator[] (
            const size_type & dimension ) const  [inline], [constexpr], [noexcept]
```

Returns the requested dimension.

For example, if `d` is a dim<2> object representing matrix dimensions, `d[0]` returns the number of rows, and `d[1]` returns the number of columns.

Parameters

dimension	the requested dimension

Returns

the `dimension`-th dimension

41.63.3.3 operator[]() [2/2]

```
template<size_type Dimensionality, typename DimensionType = size_type>
dimension_type& gko::dim< Dimensionality, DimensionType >::operator[] (
            const size_type & dimension )  [inline], [noexcept]
```

41.63.4 Friends And Related Function Documentation

41.63.4.1 operator*

```
template<size_type Dimensionality, typename DimensionType = size_type>
constexpr friend dim operator* (
            const dim< Dimensionality, DimensionType > & x,
            const dim< Dimensionality, DimensionType > & y )  [friend]
```

Multiplies two dim objects.

Parameters

x	first object
y	second object

Returns

a dim object representing the size of the tensor product `x * y`

41.63.4.2 operator<<

```
template<size_type Dimensionality, typename DimensionType = size_type>
std::ostream& operator<< (
            std::ostream & os,
            const dim< Dimensionality, DimensionType > & x )  [friend]
```

A stream operator overload for dim.

Parameters

os	stream object
x	dim object

Returns

a stream object appended with the dim output

41.63.4.3 operator==

```
template<size_type Dimensionality, typename DimensionType = size_type>
constexpr friend bool operator== (
            const dim< Dimensionality, DimensionType > & x,
            const dim< Dimensionality, DimensionType > & y )  [friend]
```

Checks if two dim objects are equal.

Parameters

x	first object
y	second object

Returns

true if and only if all dimensions of both objects are equal.

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

- ginkgo/core/base/dim.hpp

41.64 gko::DimensionMismatch Class Reference

DimensionMismatch is thrown if an operation is being applied to LinOps of incompatible size.

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- DimensionMismatch (const std::string &file, int line, const std::string &func, const std::string &first_name, size_type first_rows, size_type first_cols, const std::string &second_name, size_type second_rows, size_type second_cols, const std::string &clarification)

  Initializes a dimension mismatch error.

41.64.1 Detailed Description

DimensionMismatch is thrown if an operation is being applied to LinOps of incompatible size.

41.64.2 Constructor & Destructor Documentation

41.64.2.1 DimensionMismatch()

```
gko::DimensionMismatch::DimensionMismatch (
            const std::string & file,
            int line,
            const std::string & func,
            const std::string & first_name,
            size_type first_rows,
            size_type first_cols,
            const std::string & second_name,
            size_type second_rows,
            size_type second_cols,
            const std::string & clarification )  [inline]
```

Initializes a dimension mismatch 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
first_name	The name of the first operator
first_rows	The output dimension of the first operator
first_cols	The input dimension of the first operator
second_name	The name of the second operator
second_rows	The output dimension of the second operator
second_cols	The input dimension of the second operator
clarification	An additional message describing the error further

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

- ginkgo/core/base/exception.hpp

41.65 gko::DpcppExecutor Class Reference

This is the Executor subclass which represents a DPC++ enhanced device.

```
#include <ginkgo/core/base/executor.hpp>
```

Public Member Functions

- std::shared_ptr< Executor > get_master () noexcept override

  Returns the master OmpExecutor of this Executor.
- std::shared_ptr< const Executor > get_master () const noexcept override

  Returns the master OmpExecutor of this Executor.
- void synchronize () const override

  Synchronize the operations launched on the executor with its master.
- void run (const Operation &op) const override

  Runs the specified Operation using this Executor.
- int get_device_id () const noexcept

  Get the DPCPP device id of the device associated to this executor.
- const std::vector< int > & get_subgroup_sizes () const noexcept

  Get the available subgroup sizes for this device.
- int get_num_computing_units () const noexcept

  Get the number of Computing Units of this executor.
- const std::vector< int > & get_max_workitem_sizes () const noexcept

  Get the maximum work item sizes.
- int get_max_workgroup_size () const noexcept

  Get the maximum workgroup size.
- int get_max_subgroup_size () const noexcept

  Get the maximum subgroup size.
- std::string get_device_type () const noexcept

  Get a string representing the device type.

Static Public Member Functions

- static std::shared_ptr< DpcppExecutor > create (int device_id, std::shared_ptr< Executor > master, std↩ ::string device_type="all")

  Creates a new DpcppExecutor.
- static int get_num_devices (std::string device_type)

  Get the number of devices present on the system.

41.65.1 Detailed Description

This is the Executor subclass which represents a DPC++ enhanced device.

41.65.2 Member Function Documentation

41.65.2.1 create()

```
static std::shared_ptr<DpcppExecutor> gko::DpcppExecutor::create (
            int device_id,
            std::shared_ptr< Executor > master,
            std::string device_type = "all" )  [static]
```

Creates a new DpcppExecutor.

Parameters

device_id	the DPCPP device id of this device
master	an executor on the host that is used to invoke the device kernels
device_type	a string representing the type of device to consider (accelerator, cpu, gpu or all).

41.65.2.2 get_device_id()

```
int gko::DpcppExecutor::get_device_id ( ) const [inline], [noexcept]
```

Get the DPCPP device id of the device associated to this executor.

Returns

the DPCPP device id of the device associated to this executor

41.65.2.3 get_device_type()

```
std::string gko::DpcppExecutor::get_device_type ( ) const [inline], [noexcept]
```

Get a string representing the device type.

Returns

a string representing the device type

41.65.2.4 get_master() [1/2]

```
std::shared_ptr<const Executor> gko::DpcppExecutor::get_master ( ) const [override], [virtual],
[noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.65.2.5 get_master() [2/2]

```
std::shared_ptr<Executor> gko::DpcppExecutor::get_master ( ) [override], [virtual], [noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.65.2.6 get_max_subgroup_size()

```
int gko::DpcppExecutor::get_max_subgroup_size ( ) const [inline], [noexcept]
```

Get the maximum subgroup size.

Returns

the maximum subgroup size

41.65.2.7 get_max_workgroup_size()

```
int gko::DpcppExecutor::get_max_workgroup_size ( ) const [inline], [noexcept]
```

Get the maximum workgroup size.

Returns

the maximum workgroup size

41.65.2.8 get_max_workitem_sizes()

```
const std::vector<int>& gko::DpcppExecutor::get_max_workitem_sizes ( ) const [inline], [noexcept]
```

Get the maximum work item sizes.

Returns

the maximum work item sizes

41.65.2.9 get_num_computing_units()

```
int gko::DpcppExecutor::get_num_computing_units ( ) const [inline], [noexcept]
```

Get the number of Computing Units of this executor.

Returns

the number of Computing Units of this executor

41.65.2.10 get_num_devices()

```
static int gko::DpcppExecutor::get_num_devices (
            std::string device_type ) [static]
```

Get the number of devices present on the system.

Parameters

device_type	a string representing the device type

Returns

the number of devices present on the system

41.65.2.11 get_subgroup_sizes()

```
const std::vector<int>& gko::DpcppExecutor::get_subgroup_sizes ( ) const [inline], [noexcept]
```

Get the available subgroup sizes for this device.

Returns

the available subgroup sizes for this device

41.65.2.12 run()

```
void gko::DpcppExecutor::run (
            const Operation & op ) const [override], [virtual]
```

Runs the specified Operation using this Executor.

Parameters

op	the operation to run

Implements gko::Executor.

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

- ginkgo/core/base/executor.hpp

41.66 gko::matrix::Ell< ValueType, IndexType > Class Template Reference

ELL is a matrix format where stride with explicit zeros is used such that all rows have the same number of stored elements.

`#include <ginkgo/core/matrix/ell.hpp>`

Public Member Functions

- void read (const mat_data &data) override
  Reads a matrix from a matrix_data structure.
- void read (const device_mat_data &data) override
  Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override
  Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override
  Writes a matrix to a matrix_data structure.
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override
  Extracts the diagonal entries of the matrix into a vector.
- std::unique_ptr< absolute_type > compute_absolute () const override
  Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override
  Compute absolute inplace on each element.
- value_type * get_values () noexcept
  Returns the values of the matrix.
- const value_type * get_const_values () const noexcept
  Returns the values of the matrix.
- index_type * get_col_idxs () noexcept
  Returns the column indexes of the matrix.
- const index_type * get_const_col_idxs () const noexcept
  Returns the column indexes of the matrix.
- size_type get_num_stored_elements_per_row () const noexcept
  Returns the number of stored elements per row.
- size_type get_stride () const noexcept
  Returns the stride of the matrix.
- size_type get_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the matrix.

- value_type & val_at (size_type row, size_type idx) noexcept

  Returns the `idx`*-th non-zero element of the* `row`*-th row .*
- value_type val_at (size_type row, size_type idx) const noexcept

  Returns the `idx`*-th non-zero element of the* `row`*-th row .*
- index_type & col_at (size_type row, size_type idx) noexcept

  Returns the `idx`*-th column index of the* `row`*-th row .*
- index_type col_at (size_type row, size_type idx) const noexcept

  Returns the `idx`*-th column index of the* `row`*-th row .*

Static Public Member Functions

- static std::unique_ptr< const Ell > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs, size_type num_stored_elements_per_row, size_type stride)

  Creates a constant (immutable) Ell matrix from a set of constant arrays.

41.66.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Ell< ValueType, IndexType >

ELL is a matrix format where stride with explicit zeros is used such that all rows have the same number of stored elements.

The number of elements stored in each row is the largest number of nonzero elements in any of the rows (obtainable through get_num_stored_elements_per_row() method). This removes the need of a row pointer like in the CSR format, and allows for SIMD processing of the distinct rows. For efficient processing, the nonzero elements and the corresponding column indices are stored in column-major fashion. The columns are padded to the length by user-defined stride parameter whose default value is the number of rows of the matrix.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.66.2 Member Function Documentation

41.66.2.1 col_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type gko::matrix::Ell< ValueType, IndexType >::col_at (
            size_type row,
            size_type idx ) const [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

```
{
    return this->get_const_col_idxs()[this->linearize_index(row, idx)];
}
```

References gko::matrix::Ell< ValueType, IndexType >::get_const_col_idxs().

41.66.2.2 col_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type& gko::matrix::Ell< ValueType, IndexType >::col_at (
            size_type row,
            size_type idx )  [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::matrix::Ell< ValueType, IndexType >::get_col_idxs().

41.66.2.3 compute_absolute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<absolute_type> gko::matrix::Ell< ValueType, IndexType >::compute_absolute ( )
const  [override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Ell< ValueType, IndexType > > >.

41.66.2.4 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const Ell> gko::matrix::Ell< ValueType, IndexType >::create_const (
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            gko::detail::ConstArrayView< ValueType > && values,
            gko::detail::ConstArrayView< IndexType > && col_idxs,
            size_type num_stored_elements_per_row,
            size_type stride ) [inline], [static]
```

Creates a constant (immutable) Ell matrix from a set of constant arrays.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
col_idxs	the column index array of the matrix
num_stored_elements_per_row	the number of stored nonzeros per row
stride	the column-stride of the value and column index array

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of the arrays on the correct executor.

41.66.2.5 extract_diagonal()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Ell< ValueType, IndexType >::extract_↩
diagonal ( ) const [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.66.2.6 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Ell< ValueType, IndexType >::get_col_idxs ( ) [inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

References gko::Array< ValueType >::get_data().

Referenced by gko::matrix::Ell< ValueType, IndexType >::col_at().

41.66.2.7 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Ell< ValueType, IndexType >::get_const_col_idxs ( ) const [inline],
[noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

Referenced by gko::matrix::Ell< ValueType, IndexType >::col_at().

41.66.2.8 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Ell< ValueType, IndexType >::get_const_values ( ) const [inline],
[noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.66.2.9 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Ell< ValueType, IndexType >::get_num_stored_elements ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.66.2.10 get_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Ell< ValueType, IndexType >::get_num_stored_elements_per_row ( ) const
[inline], [noexcept]
```

Returns the number of stored elements per row.

Returns

the number of stored elements per row.

41.66.2.11 get_stride()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Ell< ValueType, IndexType >::get_stride ( ) const [inline], [noexcept]
```

Returns the stride of the matrix.

Returns

the stride of the matrix.

41.66.2.12 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Ell< ValueType, IndexType >::get_values ( ) [inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

References gko::Array< ValueType >::get_data().

41.66.2.13 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Ell< ValueType, IndexType >::read (
            const device_mat_data & data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.66.2.14 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Ell< ValueType, IndexType >::read (
            const mat_data & data )  [override], [virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.66.2.15 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Ell< ValueType, IndexType >::read (
            device_mat_data && data )  [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.66.2.16 val_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type gko::matrix::Ell< ValueType, IndexType >::val_at (
            size_type row,
            size_type idx ) const  [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.66.2.17 val_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type& gko::matrix::Ell< ValueType, IndexType >::val_at (
            size_type row,
            size_type idx ) [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row .

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

References gko::Array< ValueType >::get_data().

41.66.2.18 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Ell< ValueType, IndexType >::write (
            mat_data & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/csr.hpp
- ginkgo/core/matrix/ell.hpp

41.67 gko::enable_parameters_type< ConcreteParametersType, Factory > Class Template Reference

The enable_parameters_type mixin is used to create a base implementation of the factory parameters structure.

```
#include <ginkgo/core/base/abstract_factory.hpp>
```

Public Member Functions

- std::unique_ptr< Factory > on (std::shared_ptr< const Executor > exec) const

    Creates a new factory on the specified executor.

41.67.1 Detailed Description

template<typename ConcreteParametersType, typename Factory>
class gko::enable_parameters_type< ConcreteParametersType, Factory >

The enable_parameters_type mixin is used to create a base implementation of the factory parameters structure.

It provides only the on() method which can be used to instantiate the factory give the parameters stored in the structure.

Template Parameters

ConcreteParametersType	the concrete parameters type which is being implemented [CRTP parameter]
Factory	the concrete factory for which these parameters are being used

41.67.2 Member Function Documentation

41.67.2.1 on()

```
template<typename ConcreteParametersType, typename Factory>
std::unique_ptr<Factory> gko::enable_parameters_type< ConcreteParametersType, Factory >::on (
            std::shared_ptr< const Executor > exec ) const  [inline]
```

Creates a new factory on the specified executor.

Parameters

exec	the executor where the factory will be created

Returns

a new factory instance

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

- ginkgo/core/base/abstract_factory.hpp

41.68 gko::EnableAbsoluteComputation< AbsoluteLinOp > Class Template Reference

The EnableAbsoluteComputation mixin provides the default implementations of `compute_absolute_linop` and the absolute interface.

`#include <ginkgo/core/base/lin_op.hpp>`

Public Member Functions

- std::unique_ptr< LinOp > compute_absolute_linop () const override

  Gets the absolute LinOp.
- virtual std::unique_ptr< absolute_type > compute_absolute () const =0

  Gets the AbsoluteLinOp.

41.68.1 Detailed Description

template<typename AbsoluteLinOp>
class gko::EnableAbsoluteComputation< AbsoluteLinOp >

The EnableAbsoluteComputation mixin provides the default implementations of `compute_absolute_linop` and the absolute interface.

`compute_absolute` gets a new AbsoluteLinOp. `compute_absolute_inplace` applies absolute inplace, so it still keeps the value_type of the class.

Template Parameters

AbsoluteLinOp	the absolute LinOp which is being returned [CRTP parameter]

41.68.2 Member Function Documentation

41.68.2.1 compute_absolute()

```
template<typename AbsoluteLinOp>
virtual std::unique_ptr<absolute_type> gko::EnableAbsoluteComputation< AbsoluteLinOp >↩
::compute_absolute ( ) const  [pure virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Dense< ValueType >, gko::matrix::Hybrid< ValueType, Inde gko::matrix::Fbcsr< ValueType, IndexType >, gko::matrix::Coo< ValueType, IndexType >, gko::matrix::Ell< ValueType, IndexType > gko::matrix::Sellp< ValueType, IndexType >, and gko::matrix::Diagonal< ValueType >.

Referenced by gko::EnableAbsoluteComputation< remove_complex< Ell< ValueType, IndexType > > >↩ ::compute_absolute_linop().

41.68.2.2 compute_absolute_linop()

```
template<typename AbsoluteLinOp>
std::unique_ptr<LinOp> gko::EnableAbsoluteComputation< AbsoluteLinOp >::compute_absolute_↩
linop ( ) const  [inline], [override], [virtual]
```

Gets the absolute LinOp.

Returns

a pointer to the new absolute LinOp

Implements gko::AbsoluteComputable.

```
{
    return this->compute_absolute();
}
```

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

- ginkgo/core/base/lin_op.hpp

41.69 gko::EnableAbstractPolymorphicObject< AbstractObject, PolymorphicBase > Class Template Reference

This mixin inherits from (a subclass of) PolymorphicObject and provides a base implementation of a new abstract object.

```
#include <ginkgo/core/base/polymorphic_object.hpp>
```

41.69.1 Detailed Description

template<typename AbstractObject, typename PolymorphicBase = PolymorphicObject>
class gko::EnableAbstractPolymorphicObject< AbstractObject, PolymorphicBase >

This mixin inherits from (a subclass of) PolymorphicObject and provides a base implementation of a new abstract object.

It uses method hiding to update the parameter and return types from `PolymorphicObject to`AbstractObject` wherever it makes sense. As opposed to EnablePolymorphicObject, it does not implement PolymorphicObject's virtual methods.

Template Parameters

AbstractObject	the abstract class which is being implemented [CRTP parameter]
PolymorphicBase	parent of AbstractObject in the polymorphic hierarchy, has to be a subclass of polymorphic object

See also

EnablePolymorphicObject for creating a concrete subclass of PolymorphicObject.

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

- ginkgo/core/base/polymorphic_object.hpp

41.70 gko::EnableBatchLinOp< ConcreteBatchLinOp, PolymorphicBase > Class Template Reference

The EnableBatchLinOp mixin can be used to provide sensible default implementations of the majority of the Batch↩ LinOp and PolymorphicObject interface.

```
#include <ginkgo/core/base/batch_lin_op.hpp>
```

Additional Inherited Members

41.70.1 Detailed Description

template<typename ConcreteBatchLinOp, typename PolymorphicBase = BatchLinOp>
class gko::EnableBatchLinOp< ConcreteBatchLinOp, PolymorphicBase >

The EnableBatchLinOp mixin can be used to provide sensible default implementations of the majority of the Batch↩ LinOp and PolymorphicObject interface.

The goal of the mixin is to facilitate the development of new BatchLinOp, by enabling the implementers to focus on the important parts of their operator, while the library takes care of generating the trivial utility functions. The mixin will provide default implementations for the entire PolymorphicObject interface, including a default implementation of `copy_from` between objects of the new BatchLinOp type. It will also hide the default BatchLinOp::apply() methods with versions that preserve the static type of the object.

Implementers of new BatchLinOps are required to specify only the following aspects:

1. Creation of the BatchLinOp: This can be facilitated via either EnableCreateMethod mixin (used mostly for matrix formats), or GKO_ENABLE_BATCH_LIN_OP_FACTORY macro (used for operators created from other operators, like preconditioners and solvers).
2. Application of the BatchLinOp: Implementers have to override the two overloads of the BatchLinOp::apply↩ _impl() virtual methods.

Template Parameters

ConcreteBatchLinOp	the concrete BatchLinOp which is being implemented [CRTP parameter]
PolymorphicBase	parent of ConcreteBatchLinOp in the polymorphic hierarchy, has to be a subclass of BatchLinOp

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

- ginkgo/core/base/batch_lin_op.hpp

41.71 gko::EnableBatchScaling Class Reference

Default batch scalable interface with some type-checking.

`#include <ginkgo/core/base/batch_lin_op.hpp>`

Public Member Functions

- void batch_scale (const BatchLinOp ∗const left_scale, const BatchLinOp ∗const right_scale) override
  Scales each matrix in a batch from the left and right.

41.71.1 Detailed Description

Default batch scalable interface with some type-checking.

Provides basic functionality for solvers which need to be composed with batch scaling. If A is a matrix, S is a batch-scaling operation and F is a solver-type such that F(A) is a solver, then if the class for F inherits from this EnableBatchScaling class, it represents a new solver F(S(A)) (F composed with S applied to A).

See also

BatchScalable

41.71.2 Member Function Documentation

41.71.2.1 batch_scale()

```
void gko::EnableBatchScaling::batch_scale (
            const BatchLinOp *const left_scale,
            const BatchLinOp *const right_scale )  [inline], [override], [virtual]
```

Scales each matrix in a batch from the left and right.

Parameters

left_scale	The left scaling batch vector.
right_scale	The right scaling batch vector.

Exceptions

gko::NotSupported	If the arguments do not point to BatchDense objects.

Implements gko::BatchScalable.

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

- ginkgo/core/base/batch_lin_op.hpp

41.72 gko::solver::EnableBatchSolver< ConcreteSolver, PolymorphicBase > Class Template Reference

```
#include <ginkgo/core/solver/batch_solver.hpp>
```

Public Member Functions

- std::shared_ptr< const BatchLinOp > get_system_matrix () const
  Returns the system operator (matrix) of the linear system.

41.72.1 Detailed Description

template<typename ConcreteSolver, typename PolymorphicBase = BatchLinOp>
class gko::solver::EnableBatchSolver< ConcreteSolver, PolymorphicBase >

Template Parameters

PolymorphicBase	The base class; must be a subclass of BatchLinOp.

41.72.2 Member Function Documentation

41.72.2.1 get_system_matrix()

```
template<typename ConcreteSolver, typename PolymorphicBase = BatchLinOp>
std::shared_ptr<const BatchLinOp> gko::solver::EnableBatchSolver< ConcreteSolver, Polymorphic←
Base >::get_system_matrix ( ) const [inline]
```

Returns the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

```
{
    return system_matrix_;
}
```

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

- ginkgo/core/solver/batch_solver.hpp

41.73 gko::EnableCreateMethod< ConcreteType > Class Template Reference

This mixin implements a static `create()` method on `ConcreteType` that dynamically allocates the memory, uses the passed-in arguments to construct the object, and returns an std::unique_ptr to such an object.

```
#include <ginkgo/core/base/polymorphic_object.hpp>
```

41.73.1 Detailed Description

template<typename ConcreteType>
class gko::EnableCreateMethod< ConcreteType >

This mixin implements a static `create()` method on `ConcreteType` that dynamically allocates the memory, uses the passed-in arguments to construct the object, and returns an std::unique_ptr to such an object.

Template Parameters

ConcreteObject	the concrete type for which `create()` is being implemented [CRTP parameter]

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

- ginkgo/core/base/polymorphic_object.hpp

41.74 gko::EnableDefaultFactory< ConcreteFactory, ProductType, ParametersType, PolymorphicBase > Class Template Reference

This mixin provides a default implementation of a concrete factory.

```
#include <ginkgo/core/base/abstract_factory.hpp>
```

Public Member Functions

- const parameters_type & get_parameters () const noexcept
  Returns the parameters of the factory.

Static Public Member Functions

- static parameters_type create ()

    Creates a new ParametersType object which can be used to instantiate a new ConcreteFactory.

41.74.1 Detailed Description

template<typename ConcreteFactory, typename ProductType, typename ParametersType, typename PolymorphicBase>
class gko::EnableDefaultFactory< ConcreteFactory, ProductType, ParametersType, PolymorphicBase >

This mixin provides a default implementation of a concrete factory.

It implements all the methods of AbstractFactory and PolymorphicObject. Its implementation of the generate↩ _impl() method delegates the creation of the product by calling the `ProductType::ProductType(const ConcreteFactory *, const components_type &)` constructor. The factory also supports parameters by using the `ParametersType` structure, which is defined by the user.

For a simple example, see IntFactory in `core/test/base/abstract_factory.cpp`.

Template Parameters

ConcreteFactory	the concrete factory which is being implemented [CRTP parameter]
ProductType	the concrete type of products which this factory produces, has to be a subclass of PolymorphicBase::abstract_product_type
ParametersType	a type representing the parameters of the factory, has to inherit from the enable_parameters_type mixin
PolymorphicBase	parent of ConcreteFactory in the polymorphic hierarchy, has to be a subclass of AbstractFactory

41.74.2 Member Function Documentation

41.74.2.1 create()

```
template<typename ConcreteFactory , typename ProductType , typename ParametersType , typename
PolymorphicBase >
static parameters_type gko::EnableDefaultFactory< ConcreteFactory, ProductType, Parameters↩
Type, PolymorphicBase >::create ( ) [inline], [static]
```

Creates a new ParametersType object which can be used to instantiate a new ConcreteFactory.

This method does not construct the factory directly, but returns a new parameters_type object, which can be used to set the parameters of the factory. Once the parameters have been set, the parameters_type::on() method can be used to obtain an instance of the factory with those parameters.

Returns

a default parameters_type object

41.74.2.2 get_parameters()

```
template<typename ConcreteFactory , typename ProductType , typename ParametersType , typename
PolymorphicBase >
const parameters_type& gko::EnableDefaultFactory< ConcreteFactory, ProductType, Parameters←
Type, PolymorphicBase >::get_parameters ( ) const [inline], [noexcept]
```

Returns the parameters of the factory.

Returns

the parameters of the factory

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

- ginkgo/core/base/abstract_factory.hpp

41.75 gko::EnableLinOp< ConcreteLinOp, PolymorphicBase > Class Template Reference

The EnableLinOp mixin can be used to provide sensible default implementations of the majority of the LinOp and PolymorphicObject interface.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Additional Inherited Members

41.75.1 Detailed Description

template<typename ConcreteLinOp, typename PolymorphicBase = LinOp>
class gko::EnableLinOp< ConcreteLinOp, PolymorphicBase >

The EnableLinOp mixin can be used to provide sensible default implementations of the majority of the LinOp and PolymorphicObject interface.

The goal of the mixin is to facilitate the development of new LinOp, by enabling the implementers to focus on the important parts of their operator, while the library takes care of generating the trivial utility functions. The mixin will provide default implementations for the entire PolymorphicObject interface, including a default implementation of `copy_from` between objects of the new LinOp type. It will also hide the default LinOp::apply() methods with versions that preserve the static type of the object.

Implementers of new LinOps are required to specify only the following aspects:

1. Creation of the LinOp: This can be facilitated via either EnableCreateMethod mixin (used mostly for matrix formats), or GKO_ENABLE_LIN_OP_FACTORY macro (used for operators created from other operators, like preconditioners and solvers).
2. Application of the LinOp: Implementers have to override the two overloads of the LinOp::apply_impl() virtual methods.

Template Parameters

ConcreteLinOp	the concrete LinOp which is being implemented [CRTP parameter]
PolymorphicBase	parent of ConcreteLinOp in the polymorphic hierarchy, has to be a subclass of LinOp

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

- ginkgo/core/base/lin_op.hpp

41.76 gko::log::EnableLogging< ConcreteLoggable, PolymorphicBase > Class Template Reference

EnableLogging is a mixin which should be inherited by any class which wants to enable logging.

`#include <ginkgo/core/log/logger.hpp>`

41.76.1 Detailed Description

template<typename ConcreteLoggable, typename PolymorphicBase = Loggable>
class gko::log::EnableLogging< ConcreteLoggable, PolymorphicBase >

EnableLogging is a mixin which should be inherited by any class which wants to enable logging.

All the received events are passed to the loggers this class contains.

Template Parameters

ConcreteLoggable	the object being logged [CRTP parameter]
PolymorphicBase	the polymorphic base of this class. By default it is Loggable. Change it if you want to use a new superclass of `Loggable` as polymorphic base of this class.

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

- ginkgo/core/log/logger.hpp

41.77 gko::multigrid::EnableMultigridLevel< ValueType > Class Template Reference

The EnableMultigridLevel gives the default implementation of MultigridLevel with composition and provides `set_multigrid_level` function.

`#include <ginkgo/core/multigrid/multigrid_level.hpp>`

Public Member Functions

- std::shared_ptr< const LinOp > get_fine_op () const override

  Returns the operator on fine level.
- std::shared_ptr< const LinOp > get_restrict_op () const override

  Returns the restrict operator.
- std::shared_ptr< const LinOp > get_coarse_op () const override

  Returns the operator on coarse level.
- std::shared_ptr< const LinOp > get_prolong_op () const override

  Returns the prolong operator.

41.77.1 Detailed Description

template<typename ValueType>
class gko::multigrid::EnableMultigridLevel< ValueType >

The EnableMultigridLevel gives the default implementation of MultigridLevel with composition and provides `set_multigrid_level` function.

A class inherit from EnableMultigridLevel should use the this->get_compositions()->apply(...) as its own apply, which represents op(b) = prolong(coarse(restrict(b))).

41.77.2 Member Function Documentation

41.77.2.1 get_coarse_op()

```
template<typename ValueType >
std::shared_ptr<const LinOp> gko::multigrid::EnableMultigridLevel< ValueType >::get_coarse_op
( ) const [inline], [override], [virtual]
```

Returns the operator on coarse level.

Returns

the operator on coarse level.

Implements gko::multigrid::MultigridLevel.

```
{
    return this->get_operator_at(1);
}
```

References gko::UseComposition< ValueType >::get_operator_at().

41.77.2.2 get_fine_op()

```
template<typename ValueType >
std::shared_ptr<const LinOp> gko::multigrid::EnableMultigridLevel< ValueType >::get_fine_op (
) const [inline], [override], [virtual]
```

Returns the operator on fine level.

Returns

the operator on fine level.

Implements gko::multigrid::MultigridLevel.

41.77.2.3 get_prolong_op()

```
template<typename ValueType >
std::shared_ptr<const LinOp> gko::multigrid::EnableMultigridLevel< ValueType >::get_prolong↩
_op ( ) const [inline], [override], [virtual]
```

Returns the prolong operator.

Returns

the prolong operator.

Implements gko::multigrid::MultigridLevel.

References gko::UseComposition< ValueType >::get_operator_at().

41.77.2.4 get_restrict_op()

```
template<typename ValueType >
std::shared_ptr<const LinOp> gko::multigrid::EnableMultigridLevel< ValueType >::get_restrict↩
_op ( ) const [inline], [override], [virtual]
```

Returns the restrict operator.

Returns

the restrict operator.

Implements gko::multigrid::MultigridLevel.

References gko::UseComposition< ValueType >::get_operator_at().

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

- ginkgo/core/multigrid/multigrid_level.hpp

41.78 gko::EnablePolymorphicAssignment< ConcreteType, ResultType > Class Template Reference

This mixin is used to enable a default PolymorphicObject::copy_from() implementation for objects that have implemented conversions between them.

```
#include <ginkgo/core/base/polymorphic_object.hpp>
```

Public Member Functions

- void convert_to (result_type ∗result) const override

  Converts the implementer to an object of type result_type.
- void move_to (result_type ∗result) override

  Converts the implementer to an object of type result_type by moving data from this object.

41.78.1 Detailed Description

template<typename ConcreteType, typename ResultType = ConcreteType>
class gko::EnablePolymorphicAssignment< ConcreteType, ResultType >

This mixin is used to enable a default PolymorphicObject::copy_from() implementation for objects that have implemented conversions between them.

The requirement is that there is either a conversion constructor from `ConcreteType` in `ResultType`, or a conversion operator to `ResultType` in `ConcreteType`.

Template Parameters

ConcreteType	the concrete type from which the copy_from is being enabled [CRTP parameter]
ResultType	the type to which copy_from is being enabled

41.78.2 Member Function Documentation

41.78.2.1 convert_to()

```
template<typename ConcreteType, typename ResultType = ConcreteType>
void gko::EnablePolymorphicAssignment< ConcreteType, ResultType >::convert_to (
            result_type * result ) const [inline], [override], [virtual]
```

Converts the implementer to an object of type result_type.

Parameters

result	the object used to store the result of the conversion

Implements gko::ConvertibleTo< ResultType >.

41.78.2.2 move_to()

```
template<typename ConcreteType, typename ResultType = ConcreteType>
void gko::EnablePolymorphicAssignment< ConcreteType, ResultType >::move_to (
            result_type * result )  [inline], [override], [virtual]
```

Converts the implementer to an object of type result_type by moving data from this object.

This method is used when the implementer is a temporary object, and move semantics can be used.

Parameters

result	the object used to emplace the result of the conversion

Note

ConvertibleTo::move_to can be implemented by simply calling ConvertibleTo::convert_to. However, this operation can often be optimized by exploiting the fact that implementer's data can be moved to the result.

Implements gko::ConvertibleTo< ResultType >.

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

- ginkgo/core/base/polymorphic_object.hpp

41.79 gko::EnablePolymorphicObject< ConcreteObject, PolymorphicBase > Class Template Reference

This mixin inherits from (a subclass of) PolymorphicObject and provides a base implementation of a new concrete polymorphic object.

`#include <ginkgo/core/base/polymorphic_object.hpp>`

41.79.1 Detailed Description

template<typename ConcreteObject, typename PolymorphicBase = PolymorphicObject>
class gko::EnablePolymorphicObject< ConcreteObject, PolymorphicBase >

This mixin inherits from (a subclass of) PolymorphicObject and provides a base implementation of a new concrete polymorphic object.

The mixin changes parameter and return types of appropriate public methods of PolymorphicObject in the same way EnableAbstractPolymorphicObject does. In addition, it also provides default implementations of PolymorphicObject's vritual methods by using the *executor default constructor* and the assignment operator of ConcreteObject. Consequently, the following is a minimal example of PolymorphicObject:

```
 {c++}
struct MyObject : EnablePolymorphicObject<MyObject> {
    MyObject(std::shared_ptr<const Executor> exec)
        : EnablePolymorphicObject<MyObject>(std::move(exec))
    {}
};
```

In a way, this mixin can be viewed as an extension of default constructor/destructor/assignment operators.

Note

This mixin does not enable copying the polymorphic object to the object of the same type (i.e. it does not implement the ConvertibleTo<ConcreteObject> interface). To enable a default implementation of this interface see the EnablePolymorphicAssignment mixin.

Template Parameters

ConcreteObject	the concrete type which is being implemented [CRTP parameter]
PolymorphicBase	parent of ConcreteObject in the polymorphic hierarchy, has to be a subclass of polymorphic object

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

- ginkgo/core/base/polymorphic_object.hpp

41.80 gko::Error Class Reference

The Error class is used to report exceptional behaviour in library functions.

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- Error (const std::string &file, int line, const std::string &what)
  Initializes an error.
- virtual const char * what () const noexcept override
  Returns a human-readable string with a more detailed description of the error.

41.80.1 Detailed Description

The Error class is used to report exceptional behaviour in library functions.

Ginkgo uses C++ exception mechanism to this end, and the Error class represents a base class for all types of errors. The exact list of errors which could occur during the execution of a certain library routine is provided in the documentation of that routine, along with a short description of the situation when that error can occur. During runtime, these errors can be detected by using standard C++ try-catch blocks, and a human-readable error description can be obtained by calling the Error::what() method.

As an example, trying to compute a matrix-vector product with arguments of incompatible size will result in a DimensionMismatch error, which is demonstrated in the following program.

```
#include <ginkgo.h>
#include <iostream>
using namespace gko;
int main()
{
    auto omp = create<OmpExecutor>();
    auto A = randn_fill<matrix::Csr<float»(5, 5, 0f, 1f, omp);
    auto x = fill<matrix::Dense<float»(6, 1, 1f, omp);
    try {
        auto y = apply(A.get(), x.get());
    } catch(Error e) {
        // an error occured, write the message to screen and exit
        std::cout « e.what() « std::endl;
        return -1;
    }
    return 0;
}
```

41.80.2 Constructor & Destructor Documentation

41.80.2.1 Error()

```
gko::Error::Error (
            const std::string & file,
            int line,
            const std::string & what )  [inline]
```

Initializes an error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
what	The error message

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

- ginkgo/core/base/exception.hpp

41.81 gko::Executor Class Reference

The first step in using the Ginkgo library consists of creating an executor.

```
#include <ginkgo/core/base/executor.hpp>
```

Public Member Functions

- virtual void run (const Operation &op) const =0

  Runs the specified Operation using this Executor.
- template<typename ClosureOmp , typename ClosureCuda , typename ClosureHip , typename ClosureDpcpp >
  void run (const ClosureOmp &op_omp, const ClosureCuda &op_cuda, const ClosureHip &op_hip, const ClosureDpcpp &op_dpcpp) const

  Runs one of the passed in functors, depending on the Executor type.
- template<typename T >
  T * alloc (size_type num_elems) const

  Allocates memory in this Executor.
- void free (void *ptr) const noexcept

  Frees memory previously allocated with Executor::alloc().
- template<typename T >
  void copy_from (const Executor *src_exec, size_type num_elems, const T *src_ptr, T *dest_ptr) const

  Copies data from another Executor.
- template<typename T >
  void copy (size_type num_elems, const T *src_ptr, T *dest_ptr) const

  Copies data within this Executor.

- template<typename T >
  T copy_val_to_host (const T ∗ptr) const

  Retrieves a single element at the given location from executor memory.
- virtual std::shared_ptr< Executor > get_master () noexcept=0

  Returns the master OmpExecutor of this Executor.
- virtual std::shared_ptr< const Executor > get_master () const noexcept=0

  Returns the master OmpExecutor of this Executor.
- virtual void synchronize () const =0

  Synchronize the operations launched on the executor with its master.
- bool memory_accessible (const std::shared_ptr< const Executor > &other) const

  Verifies whether the executors share the same memory.

41.81.1 Detailed Description

The first step in using the Ginkgo library consists of creating an executor.

Executors are used to specify the location for the data of linear algebra objects, and to determine where the operations will be executed. Ginkgo currently supports five different executor types:

- OmpExecutor specifies that the data should be stored and the associated operations executed on an OpenMP-supporting device (e.g. host CPU);
- CudaExecutor specifies that the data should be stored and the operations executed on the NVIDIA GPU accelerator;
- HipExecutor specifies that the data should be stored and the operations executed on either an NVIDIA or AMD GPU accelerator;
- DpcppExecutor specifies that the data should be stored and the operations executed on an hardware supporting DPC++;
- ReferenceExecutor executes a non-optimized reference implementation, which can be used to debug the library.

The following code snippet demonstrates the simplest possible use of the Ginkgo library:
```
auto omp = gko::create<gko::OmpExecutor>();
auto A = gko::read_from_mtx<gko::matrix::Csr<float>>("A.mtx", omp);
```

First, we create a OMP executor, which will be used in the next line to specify where we want the data for the matrix A to be stored. The second line will read a matrix from the matrix market file 'A.mtx', and store the data on the CPU in CSR format (gko::matrix::Csr is a Ginkgo matrix class which stores its data in CSR format). At this point, matrix A is bound to the CPU, and any routines called on it will be performed on the CPU. This approach is usually desired in sparse linear algebra, as the cost of individual operations is several orders of magnitude lower than the cost of copying the matrix to the GPU.

If matrix A is going to be reused multiple times, it could be beneficial to copy it over to the accelerator, and perform the operations there, as demonstrated by the next code snippet:
```
auto cuda = gko::create<gko::CudaExecutor>(0, omp);
auto dA = gko::copy_to<gko::matrix::Csr<float>>(A.get(), cuda);
```

The first line of the snippet creates a new CUDA executor. Since there may be multiple NVIDIA GPUs present on the system, the first parameter instructs the library to use the first device (i.e. the one with device ID zero, as in cudaSetDevice() routine from the CUDA runtime API). In addition, since GPUs are not stand-alone processors, it is required to pass a "master" OmpExecutor which will be used to schedule the requested CUDA kernels on the accelerator.

The second command creates a copy of the matrix A on the GPU. Notice the use of the get() method. As Ginkgo aims to provide automatic memory management of its objects, the result of calling gko::read_from_mtx() is a smart pointer (std::unique_ptr) to the created object. On the other hand, as the library will not hold a reference to A once the copy is completed, the input parameter for gko::copy_to() is a plain pointer. Thus, the get() method is used to convert from a std::unique_ptr to a plain pointer, as expected by gko::copy_to().

As a side note, the gko::copy_to routine is far more powerful than just copying data between different devices. It can also be used to convert data between different formats. For example, if the above code used gko::matrix::Ell as the template parameter, dA would be stored on the GPU, in ELLPACK format.

Finally, if all the processing of the matrix is supposed to be done on the GPU, and a CPU copy of the matrix is not required, we could have read the matrix to the GPU directly:

```
auto omp = gko::create<gko::OmpExecutor>();
auto cuda = gko::create<gko::CudaExecutor>(0, omp);
auto dA = gko::read_from_mtx<gko::matrix::Csr<float»("A.mtx", cuda);
```

Notice that even though reading the matrix directly from a file to the accelerator is not supported, the library is designed to abstract away the intermediate step of reading the matrix to the CPU memory. This is a general design approach taken by the library: in case an operation is not supported by the device, the data will be copied to the CPU, the operation performed there, and finally the results copied back to the device. This approach makes using the library more concise, as explicit copies are not required by the user. Nevertheless, this feature should be taken into account when considering performance implications of using such operations.

41.81.2 Member Function Documentation

41.81.2.1 alloc()

```
template<typename T >
T* gko::Executor::alloc (
              size_type num_elems ) const  [inline]
```

Allocates memory in this Executor.

Template Parameters

T	datatype to allocate

Parameters

num_elems	number of elements of type T to allocate

Exceptions

AllocationError	if the allocation failed

Returns

pointer to allocated memory

41.81.2.2 copy()

```
template<typename T >
void gko::Executor::copy (
            size_type num_elems,
            const T * src_ptr,
            T * dest_ptr ) const [inline]
```

Copies data within this Executor.

Template Parameters

T	datatype to copy

Parameters

num_elems	number of elements of type T to copy
src_ptr	pointer to a block of memory containing the data to be copied
dest_ptr	pointer to an allocated block of memory where the data will be copied to

References copy_from().

41.81.2.3 copy_from()

```
template<typename T >
void gko::Executor::copy_from (
            const Executor * src_exec,
            size_type num_elems,
            const T * src_ptr,
            T * dest_ptr ) const [inline]
```

Copies data from another Executor.

Template Parameters

T	datatype to copy

Parameters

src_exec	Executor from which the memory will be copied
num_elems	number of elements of type T to copy
src_ptr	pointer to a block of memory containing the data to be copied
dest_ptr	pointer to an allocated block of memory where the data will be copied to

References get_master().

Referenced by copy().

41.81.2.4 copy_val_to_host()

```
template<typename T >
T gko::Executor::copy_val_to_host (
            const T * ptr ) const  [inline]
```

Retrieves a single element at the given location from executor memory.

Template Parameters

T	datatype to copy

Parameters

ptr	the pointer to the element to be copied

Returns

the value stored at ptr

References get_master().

41.81.2.5 free()

```
void gko::Executor::free (
            void * ptr ) const  [inline], [noexcept]
```

Frees memory previously allocated with Executor::alloc().

If `ptr` is a `nullptr`, the function has no effect.

Parameters

ptr	pointer to the allocated memory block

41.81.2.6 get_master() [1/2]

```
virtual std::shared_ptr<const Executor> gko::Executor::get_master ( ) const  [pure virtual],
[noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implemented in gko::DpcppExecutor, gko::HipExecutor, gko::CudaExecutor, and gko::OmpExecutor.

41.81.2.7 get_master() [2/2]

```
virtual std::shared_ptr<Executor> gko::Executor::get_master ( ) [pure virtual], [noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implemented in gko::DpcppExecutor, gko::HipExecutor, gko::CudaExecutor, and gko::OmpExecutor.

Referenced by copy_from(), and copy_val_to_host().

41.81.2.8 memory_accessible()

```
bool gko::Executor::memory_accessible (
            const std::shared_ptr< const Executor > & other ) const [inline]
```

Verifies whether the executors share the same memory.

Parameters

other	the other Executor to compare against

Returns

whether the executors this and other share the same memory.

41.81.2.9 run() [1/2]

```
template<typename ClosureOmp , typename ClosureCuda , typename ClosureHip , typename Closure←
Dpcpp >
void gko::Executor::run (
            const ClosureOmp & op_omp,
            const ClosureCuda & op_cuda,
            const ClosureHip & op_hip,
            const ClosureDpcpp & op_dpcpp ) const [inline]
```

Runs one of the passed in functors, depending on the Executor type.

Template Parameters

ClosureOmp	type of op_omp
ClosureCuda	type of op_cuda
ClosureHip	type of op_hip
ClosureDpcpp	type of op_dpcpp

Parameters

op_omp	functor to run in case of a OmpExecutor or ReferenceExecutor
op_cuda	functor to run in case of a CudaExecutor
op_hip	functor to run in case of a HipExecutor
op_dpcpp	functor to run in case of a DpcppExecutor

References run().

41.81.2.10 run() [2/2]

```
virtual void gko::Executor::run (
            const Operation & op ) const  [pure virtual]
```

Runs the specified Operation using this Executor.

Parameters

op	the operation to run

Implemented in gko::DpcppExecutor, gko::HipExecutor, gko::CudaExecutor, and gko::ReferenceExecutor.

Referenced by run().

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

- ginkgo/core/base/executor.hpp

41.82 gko::log::executor_data Struct Reference

Struct representing Executor related data.

```
#include <ginkgo/core/log/record.hpp>
```

41.82.1 Detailed Description

Struct representing Executor related data.

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

- ginkgo/core/log/record.hpp

41.83 gko::executor_deleter< T > Class Template Reference

This is a deleter that uses an executor's `free` method to deallocate the data.

`#include <ginkgo/core/base/executor.hpp>`

Public Member Functions

- executor_deleter (std::shared_ptr< const Executor > exec)

    Creates a new deleter.
- void operator() (pointer ptr) const

    Deletes the object.

41.83.1 Detailed Description

template<typename T>
class gko::executor_deleter< T >

This is a deleter that uses an executor's `free` method to deallocate the data.

Template Parameters

T	the type of object being deleted

41.83.2 Constructor & Destructor Documentation

41.83.2.1 executor_deleter()

```
template<typename T >
gko::executor_deleter< T >::executor_deleter (
            std::shared_ptr< const Executor > exec )  [inline], [explicit]
```

Creates a new deleter.

Parameters

exec	the executor used to free the data

41.83.3 Member Function Documentation

41.83.3.1 operator()()

```
template<typename T >
void gko::executor_deleter< T >::operator() (
            pointer ptr ) const  [inline]
```

Deletes the object.

Parameters

ptr	pointer to the object being deleted

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

- ginkgo/core/base/executor.hpp

41.84 gko::matrix::Fbcsr< ValueType, IndexType > Class Template Reference

Fixed-block compressed sparse row storage matrix format.

`#include <ginkgo/core/matrix/fbcsr.hpp>`

Public Member Functions

- void convert_to (Csr< ValueType, IndexType > *result) const override
  Converts the matrix to CSR format.
- void convert_to (SparsityCsr< ValueType, IndexType > *result) const override
  Get the block sparsity pattern in CSR-like format.
- void read (const mat_data &data) override
  Reads a matrix_data into Fbcsr format.
- void read (const device_mat_data &data) override
  Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override
  Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override
  Writes a matrix to a matrix_data structure.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override
  Extracts the diagonal entries of the matrix into a vector.
- std::unique_ptr< absolute_type > compute_absolute () const override
  Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override
  Compute absolute inplace on each element.
- void sort_by_column_index ()

Sorts the values blocks and block-column indices in each row by column index.
- bool is_sorted_by_column_index () const

  Tests if all row entry pairs (value, col_idx) are sorted by column index.
- value_type ∗ get_values () noexcept
- const value_type ∗ get_const_values () const noexcept
- index_type ∗ get_col_idxs () noexcept
- const index_type ∗ get_const_col_idxs () const noexcept
- index_type ∗ get_row_ptrs () noexcept
- const index_type ∗ get_const_row_ptrs () const noexcept
- size_type get_num_stored_elements () const noexcept
- size_type get_num_stored_blocks () const noexcept
- int get_block_size () const noexcept
- index_type get_num_block_rows () const noexcept
- index_type get_num_block_cols () const noexcept

Static Public Member Functions

- static std::unique_ptr< const Fbcsr > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, int blocksize, gko::detail::ConstArrayView< ValueType > &&values, gko::detail::ConstArrayView< IndexType > &&col_idxs, gko::detail::ConstArrayView< IndexType > &&row_ptrs)

  Creates a constant (immutable) Fbcsr matrix from a constant array.

41.84.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Fbcsr< ValueType, IndexType >

Fixed-block compressed sparse row storage matrix format.

FBCSR is a matrix format meant for matrices having a natural block structure made up of small, dense, disjoint blocks. It is similar to CSR

See also

Csr. However, unlike Csr, each non-zero location stores a small dense block of entries having a constant size. This reduces the number of integers that need to be stored in order to refer to a given non-zero entry, and enables efficient implementation of certain block methods.

The block size is expected to be known in advance and passed to the constructor.

Note

The total number of rows and the number of columns are expected to be divisible by the block size.

The nonzero elements are stored in a 1D array row-wise, and accompanied with a row pointer array which stores the starting index of each block-row. An additional block-column index array is used to identify the block-column of each nonzero block.

The Fbcsr LinOp supports different operations:
```
matrix::Fbcsr *A, *B, *C;      // matrices
matrix::Dense *b, *x;         // vectors tall-and-skinny matrices
matrix::Dense *alpha, *beta; // scalars of dimension 1x1
// Applying to Dense matrices computes an SpMV/SpMM product
A->apply(b, x)              // x = A*b
A->apply(alpha, b, beta, x) // x = alpha*A*b + beta*x
```

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.84.2 Member Function Documentation

41.84.2.1 compute_absolute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<absolute_type> gko::matrix::Fbcsr< ValueType, IndexType >::compute_absolute (
) const [override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Fbcsr< ValueType, IndexType > > >.

41.84.2.2 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Fbcsr< ValueType, IndexType >::conj_transpose ( ) const
[override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.84.2.3 convert_to() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Fbcsr< ValueType, IndexType >::convert_to (
            Csr< ValueType, IndexType > * result ) const [override]
```

Converts the matrix to CSR format.

Note

Any explicit zeros in the original matrix are retained in the converted result.

41.84.2.4 convert_to() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Fbcsr< ValueType, IndexType >::convert_to (
            SparsityCsr< ValueType, IndexType > * result ) const  [override]
```

Get the block sparsity pattern in CSR-like format.

Note

The actual non-zero values are never copied; the result always has a value array of size 1 with the value 1.

41.84.2.5 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const Fbcsr> gko::matrix::Fbcsr< ValueType, IndexType >::create_const
(
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            int blocksize,
            gko::detail::ConstArrayView< ValueType > && values,
            gko::detail::ConstArrayView< IndexType > && col_idxs,
            gko::detail::ConstArrayView< IndexType > && row_ptrs )  [inline], [static]
```

Creates a constant (immutable) Fbcsr matrix from a constant array.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
blocksize	the block size of the matrix
values	the value array of the matrix
col_idxs	the block column index array of the matrix
row_ptrs	the block row pointer array of the matrix

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of the arrays on the correct executor.

```
    {
        // cast const-ness away, but return a const object afterwards,
        // so we can ensure that no modifications take place.
        return std::unique_ptr<const Fbcsr>(
            new Fbcsr{exec, size, blocksize,
                      gko::detail::array_const_cast(std::move(values)),
                      gko::detail::array_const_cast(std::move(col_idxs)),
                      gko::detail::array_const_cast(std::move(row_ptrs))});
    }
```

41.84.2.6 extract_diagonal()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Fbcsr< ValueType, IndexType >::extract_↩
diagonal ( ) const [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.84.2.7 get_block_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
int gko::matrix::Fbcsr< ValueType, IndexType >::get_block_size ( ) const [inline], [noexcept]
```

Returns

The fixed block size for this matrix

41.84.2.8 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Fbcsr< ValueType, IndexType >::get_col_idxs ( ) [inline], [noexcept]
```

Returns

The column indexes of the matrix.

References gko::Array< ValueType >::get_data().

41.84.2.9 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Fbcsr< ValueType, IndexType >::get_const_col_idxs ( ) const
[inline], [noexcept]
```

Returns

The column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.84.2.10 get_const_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Fbcsr< ValueType, IndexType >::get_const_row_ptrs ( ) const
[inline], [noexcept]
```

Returns

The row pointers of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.84.2.11 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Fbcsr< ValueType, IndexType >::get_const_values ( ) const [inline],
[noexcept]
```

Returns

The values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.84.2.12 get_num_block_cols()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type gko::matrix::Fbcsr< ValueType, IndexType >::get_num_block_cols ( ) const [inline],
[noexcept]
```

Returns

The number of block-columns in the matrix

41.84.2.13 get_num_block_rows()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type gko::matrix::Fbcsr< ValueType, IndexType >::get_num_block_rows ( ) const [inline],
[noexcept]
```

Returns

The number of block-rows in the matrix

41.84.2.14 get_num_stored_blocks()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Fbcsr< ValueType, IndexType >::get_num_stored_blocks ( ) const [inline],
[noexcept]
```

Returns

The number of non-zero blocks explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.84.2.15 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Fbcsr< ValueType, IndexType >::get_num_stored_elements ( ) const [inline],
[noexcept]
```

Returns

The number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.84.2.16 get_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Fbcsr< ValueType, IndexType >::get_row_ptrs ( ) [inline], [noexcept]
```

Returns

The row pointers of the matrix.

References gko::Array< ValueType >::get_data().

41.84.2.17 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Fbcsr< ValueType, IndexType >::get_values ( ) [inline], [noexcept]
```

Returns

The values of the matrix.

References gko::Array< ValueType >::get_data().

41.84.2.18 is_sorted_by_column_index()

```
template<typename ValueType = default_precision, typename IndexType = int32>
bool gko::matrix::Fbcsr< ValueType, IndexType >::is_sorted_by_column_index ( ) const
```

Tests if all row entry pairs (value, col_idx) are sorted by column index.

Returns

True if all row entry pairs (value, col_idx) are sorted by column index

41.84.2.19 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Fbcsr< ValueType, IndexType >::read (
            const device_mat_data & data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.84.2.20 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Fbcsr< ValueType, IndexType >::read (
            const mat_data & data ) [override], [virtual]
```

Reads a matrix_data into Fbcsr format.

Requires the block size to be set beforehand

See also

set_block_size.

Warning

Unlike Csr::read, here explicit non-zeros are NOT dropped.

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.84.2.21 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Fbcsr< ValueType, IndexType >::read (
            device_mat_data && data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.84.2.22 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::Fbcsr< ValueType, IndexType >::transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.84.2.23 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Fbcsr< ValueType, IndexType >::write (
            mat_data & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/csr.hpp
- ginkgo/core/matrix/fbcsr.hpp

41.85 gko::solver::Fcg< ValueType > Class Template Reference

FCG or the flexible conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

```
#include <ginkgo/core/solver/fcg.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const

  Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override

  Return true as iterative solvers use the data in x as an initial guess.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const

  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)

  Sets the stopping criterion of the solver.

41.85.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Fcg< ValueType >

FCG or the flexible conjugate gradient method is an iterative type Krylov subspace method which is suitable for symmetric positive definite methods.

Though this method performs very well for symmetric positive definite matrices, it is in general not suitable for general matrices.

In contrast to the standard CG based on the Polack-Ribiere formula, the flexible CG uses the Fletcher-Reeves formula for creating the orthonormal vectors spanning the Krylov subspace. This increases the computational cost of every Krylov solver iteration but allows for non-constant preconditioners.

The implementation in Ginkgo makes use of the merged kernel to make the best use of data locality. The inner operations in one iteration of FCG are merged into 2 separate steps.

Template Parameters

ValueType	precision of matrix elements

41.85.2 Member Function Documentation

41.85.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Fcg< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
107 { return true; }
```

41.85.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Fcg< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.85.2.3 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Fcg< ValueType >::get_stop_←
criterion_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.85.2.4 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Fcg< ValueType >::get_system_matrix ( ) const [inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.85.2.5 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Fcg< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other ) [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.85.2.6 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Fcg< ValueType >::transpose ( ) const [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/fcg.hpp

41.86 gko::matrix::Fft Class Reference

This LinOp implements a 1D Fourier matrix using the FFT algorithm.

```
#include <ginkgo/core/matrix/fft.hpp>
```

Public Member Functions

- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- void write (matrix_data< std::complex< float >, int32 > &data) const override
  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< float >, int64 > &data) const override
  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< double >, int32 > &data) const override
  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< double >, int64 > &data) const override
  Writes a matrix to a matrix_data structure.

41.86.1 Detailed Description

This LinOp implements a 1D Fourier matrix using the FFT algorithm.

It implements forward and inverse DFT.

For a power-of-two size n with corresponding root of unity $\omega = e^{-2\pi i/n}$ for forward DFT and $\omega = e^{2\pi i/n}$ for inverse DFT it computes

$$x_k = \sum_{j=0}^{n-1} \omega^{jk} b_j$$

without normalization factors.

The Reference and OpenMP implementations support only power-of-two input sizes, as they use the Radix-2 algorithm by J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, no. 90, pp. 297–301, 1965, doi: 10.2307/2003354. The CUDA and HIP implementations use cuSPARSE/hipSPARSE with full support for non-power-of-two input sizes and special optimizations for products of small prime powers.

41.86.2 Member Function Documentation

41.86.2.1 conj_transpose()

```
std::unique_ptr<LinOp> gko::matrix::Fft::conj_transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.86.2.2 transpose()

```
std::unique_ptr<LinOp> gko::matrix::Fft::transpose ( ) const [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.86.2.3 write() [1/4]

```
void gko::matrix::Fft::write (
            matrix_data< std::complex< double >, int32 > & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< double >, int32 >.

41.86.2.4 write() [2/4]

```
void gko::matrix::Fft::write (
            matrix_data< std::complex< double >, int64 > & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< double >, int64 >.

41.86.2.5 write() [3/4]

```
void gko::matrix::Fft::write (
            matrix_data< std::complex< float >, int32 > & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< float >, int32 >.

41.86.2.6 write() [4/4]

```
void gko::matrix::Fft::write (
            matrix_data< std::complex< float >, int64 > & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< float >, int64 >.

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

- ginkgo/core/matrix/fft.hpp

41.87 gko::matrix::Fft2 Class Reference

This LinOp implements a 2D Fourier matrix using the FFT algorithm.

```
#include <ginkgo/core/matrix/fft.hpp>
```

Public Member Functions

- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.
- void write (matrix_data< std::complex< float >, int32 > &data) const override

  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< float >, int64 > &data) const override

  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< double >, int32 > &data) const override

  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< double >, int64 > &data) const override

  Writes a matrix to a matrix_data structure.

41.87.1 Detailed Description

This LinOp implements a 2D Fourier matrix using the FFT algorithm.

For indexing purposes, the first dimension is the major axis.

It implements complex-to-complex forward and inverse FFT.

For a power-of-two sizes n_1, n_2 with corresponding root of unity $\omega = e^{-2\pi i/(n_1 n_2)}$ for forward DFT and $\omega = e^{2\pi i/(n_1 n_2)}$ for inverse DFT it computes

$$x_{k_1 n_2 + k_2} = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \omega^{i_1 k_1 + i_2 k_2} b_{i_1 n_2 + i_2}$$

without normalization factors.

The Reference and OpenMP implementations support only power-of-two input sizes, as they use the Radix-2 algorithm by J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, no. 90, pp. 297–301, 1965, doi: 10.2307/2003354. The CUDA and HIP implementations use cuSPARSE/hipSPARSE with full support for non-power-of-two input sizes and special optimizations for products of small prime powers.

41.87.2 Member Function Documentation

41.87.2.1 conj_transpose()

```
std::unique_ptr<LinOp> gko::matrix::Fft2::conj_transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.87.2.2 transpose()

```
std::unique_ptr<LinOp> gko::matrix::Fft2::transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.87.2.3 write() [1/4]

```
void gko::matrix::Fft2::write (
            matrix_data< std::complex< double >, int32 > & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< double >, int32 >.

41.87.2.4 write() [2/4]

```
void gko::matrix::Fft2::write (
            matrix_data< std::complex< double >, int64 > & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< double >, int64 >.

41.87.2.5 write() [3/4]

```
void gko::matrix::Fft2::write (
            matrix_data< std::complex< float >, int32 > & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< float >, int32 >.

41.87.2.6 write() [4/4]

```
void gko::matrix::Fft2::write (
            matrix_data< std::complex< float >, int64 > & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< float >, int64 >.

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

- ginkgo/core/matrix/fft.hpp

41.88 gko::matrix::Fft3 Class Reference

This LinOp implements a 3D Fourier matrix using the FFT algorithm.

```
#include <ginkgo/core/matrix/fft.hpp>
```

Public Member Functions

- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- void write (matrix_data< std::complex< float >, int32 > &data) const override
  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< float >, int64 > &data) const override
  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< double >, int32 > &data) const override
  Writes a matrix to a matrix_data structure.
- void write (matrix_data< std::complex< double >, int64 > &data) const override
  Writes a matrix to a matrix_data structure.

41.88.1 Detailed Description

This LinOp implements a 3D Fourier matrix using the FFT algorithm.

For indexing purposes, the first dimension is the major axis.

It implements complex-to-complex forward and inverse FFT.

For a power-of-two sizes n_1, n_2, n_3 with corresponding root of unity $\omega = e^{-2\pi i/(n_1 n_2 n_3)}$ for forward DFT and $\omega = e^{2\pi i/(n_1 n_2 n_3)}$ for inverse DFT it computes

$$x_{k_1 n_2 n_3 + k_2 n_3 + k_3} = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \sum_{i_3=0}^{n_3-1} \omega^{i_1 k_1 + i_2 k_2 + i_3 k_3} b_{i_1 n_2 n_3 + i_2 n_3 + i_3}$$

without normalization factors.

The Reference and OpenMP implementations support only power-of-two input sizes, as they use the Radix-2 algorithm by J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, no. 90, pp. 297–301, 1965, doi: 10.2307/2003354. The CUDA and HIP implementations use cuSPARSE/hipSPARSE with full support for non-power-of-two input sizes and special optimizations for products of small prime powers.

41.88.2 Member Function Documentation

41.88.2.1 conj_transpose()

```
std::unique_ptr<LinOp> gko::matrix::Fft3::conj_transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.88.2.2 transpose()

```
std::unique_ptr<LinOp> gko::matrix::Fft3::transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.88.2.3 write() [1/4]

```
void gko::matrix::Fft3::write (
            matrix_data< std::complex< double >, int32 > & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< double >, int32 >.

41.88.2.4 write() [2/4]

```
void gko::matrix::Fft3::write (
            matrix_data< std::complex< double >, int64 > & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< double >, int64 >.

41.88.2.5 write() [3/4]

```
void gko::matrix::Fft3::write (
            matrix_data< std::complex< float >, int32 > & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< float >, int32 >.

41.88.2.6 write() [4/4]

```
void gko::matrix::Fft3::write (
            matrix_data< std::complex< float >, int64 > & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< std::complex< float >, int64 >.

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

- ginkgo/core/matrix/fft.hpp

41.89 gko::solver::Gmres< ValueType > Class Template Reference

GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

```
#include <ginkgo/core/solver/gmres.hpp>
```

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const
  Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.
- size_type get_krylov_dim () const
  Gets the Krylov dimension of the solver.
- void set_krylov_dim (size_type other)
  Sets the Krylov dimension.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const
  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)
  Sets the stopping criterion of the solver.

41.89.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Gmres< ValueType >

GMRES or the generalized minimal residual method is an iterative type Krylov subspace method which is suitable for nonsymmetric linear systems.

The implementation in Ginkgo makes use of the merged kernel to make the best use of data locality. The inner operations in one iteration of GMRES are merged into 2 separate steps. Modified Gram-Schmidt is used.

Template Parameters

ValueType	precision of matrix elements

41.89.2 Member Function Documentation

41.89.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Gmres< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
102 { return true; }
```

41.89.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Gmres< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.89.2.3 get_krylov_dim()

```
template<typename ValueType = default_precision>
size_type gko::solver::Gmres< ValueType >::get_krylov_dim ( ) const [inline]
```

Gets the Krylov dimension of the solver.

Returns

the Krylov dimension

41.89.2.4 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Gmres< ValueType >::get_stop_↩
criterion_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.89.2.5 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Gmres< ValueType >::get_system_matrix ( ) const
[inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.89.2.6 set_krylov_dim()

```
template<typename ValueType = default_precision>
void gko::solver::Gmres< ValueType >::set_krylov_dim (
            size_type other )  [inline]
```

Sets the Krylov dimension.

Parameters

other	the new Krylov dimension

41.89.2.7 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Gmres< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other )  [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.89.2.8 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Gmres< ValueType >::transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/gmres.hpp

41.90 gko::solver::has_with_criteria< SolverType, typename > Struct Template Reference

Helper structure to test if the Factory of SolverType has a function `with_criteria`.

`#include <ginkgo/core/solver/solver_traits.hpp>`

41.90.1 Detailed Description

template<typename SolverType, typename = void>
struct gko::solver::has_with_criteria< SolverType, typename >

Helper structure to test if the Factory of SolverType has a function `with_criteria`.

Contains a constexpr boolean `value`, which is true if the Factory class of SolverType has a `with_criteria`, and false otherwise.

Template Parameters

SolverType	Solver to test if its factory has a with_criteria function.

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

- ginkgo/core/solver/solver_traits.hpp

41.91 gko::solver::has_with_criteria< SolverType, xstd::void_t< decltype(SolverType::build().with_criteria(std::shared_ptr< const stop::CriterionFactory >()))> > Struct Template Reference

Helper structure to test if the Factory of SolverType has a function `with_criteria`.

`#include <ginkgo/core/solver/solver_traits.hpp>`

41.91.1 Detailed Description

template<typename SolverType>
struct gko::solver::has_with_criteria< SolverType, xstd::void_t< decltype(SolverType::build().with_criteria(std::shared_ptr< const stop::CriterionFactory >()))> >

Helper structure to test if the Factory of SolverType has a function `with_criteria`.

Contains a constexpr boolean `value`, which is true if the Factory class of SolverType has a `with_criteria`, and false otherwise.

Template Parameters

SolverType	Solver to test if its factory has a with_criteria function.

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

- ginkgo/core/solver/solver_traits.hpp

41.92 gko::HipblasError Class Reference

HipblasError is thrown when a hipBLAS routine throws a non-zero error code.

`#include <ginkgo/core/base/exception.hpp>`

Public Member Functions

- HipblasError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a hipBLAS error.

41.92.1 Detailed Description

HipblasError is thrown when a hipBLAS routine throws a non-zero error code.

41.92.2 Constructor & Destructor Documentation

41.92.2.1 HipblasError()

```
gko::HipblasError::HipblasError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code )  [inline]
```

Initializes a hipBLAS 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 hipBLAS routine that failed
error_code	The resulting hipBLAS error code

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

- ginkgo/core/base/exception.hpp

41.93 gko::HipError Class Reference

HipError is thrown when a HIP routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- HipError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a HIP error.

41.93.1 Detailed Description

HipError is thrown when a HIP routine throws a non-zero error code.

41.93.2 Constructor & Destructor Documentation

41.93.2.1 HipError()

```
gko::HipError::HipError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code )  [inline]
```

Initializes a HIP 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 HIP routine that failed
error_code	The resulting HIP error code

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

- ginkgo/core/base/exception.hpp

41.94 gko::HipExecutor Class Reference

This is the Executor subclass which represents the HIP enhanced device.

`#include <ginkgo/core/base/executor.hpp>`

Public Member Functions

- std::shared_ptr< Executor > get_master () noexcept override

  Returns the master OmpExecutor of this Executor.
- std::shared_ptr< const Executor > get_master () const noexcept override

  Returns the master OmpExecutor of this Executor.
- void synchronize () const override

  Synchronize the operations launched on the executor with its master.
- void run (const Operation &op) const override

  Runs the specified Operation using this Executor.
- int get_device_id () const noexcept

  Get the HIP device id of the device associated to this executor.
- int get_num_warps_per_sm () const noexcept

  Get the number of warps per SM of this executor.
- int get_num_multiprocessor () const noexcept

  Get the number of multiprocessor of this executor.
- int get_major_version () const noexcept

  Get the major verion of compute capability.
- int get_minor_version () const noexcept

  Get the minor verion of compute capability.
- int get_num_warps () const noexcept

  Get the number of warps of this executor.
- int get_warp_size () const noexcept

  Get the warp size of this executor.
- int get_max_shared_memory_per_block () const noexcept

  Get maximum shared memory per block.
- hipblasContext * get_hipblas_handle () const

  Get the hipblas handle for this executor.
- hipsparseContext * get_hipsparse_handle () const

  Get the hipsparse handle for this executor.
- int get_closest_numa () const

  Get the closest NUMA node.
- std::vector< int > get_closest_pus () const

  Get the closest PUs.

Static Public Member Functions

- static std::shared_ptr< HipExecutor > create (int device_id, std::shared_ptr< Executor > master, bool device_reset=false, allocation_mode alloc_mode=default_hip_alloc_mode)

    Creates a new HipExecutor.
- static int get_num_devices ()

    Get the number of devices present on the system.

41.94.1 Detailed Description

This is the Executor subclass which represents the HIP enhanced device.

41.94.2 Member Function Documentation

41.94.2.1 create()

```
static std::shared_ptr<HipExecutor> gko::HipExecutor::create (
            int device_id,
            std::shared_ptr< Executor > master,
            bool device_reset = false,
            allocation_mode alloc_mode = default_hip_alloc_mode ) [static]
```

Creates a new HipExecutor.

Parameters

device_id	the HIP device id of this device
master	an executor on the host that is used to invoke the device kernels
device_reset	whether to reset the device after the object exits the scope.
alloc_mode	the allocation mode that the executor should operate on. See @allocation_mode for more details

41.94.2.2 get_closest_numa()

```
int gko::HipExecutor::get_closest_numa ( ) const [inline]
```

Get the closest NUMA node.

Returns

the closest NUMA node closest to this device

41.94.2.3 get_closest_pus()

```
std::vector<int> gko::HipExecutor::get_closest_pus ( ) const [inline]
```

Get the closest PUs.

Returns

the array of PUs closest to this device

41.94.2.4 get_hipblas_handle()

```
hipblasContext* gko::HipExecutor::get_hipblas_handle ( ) const [inline]
```

Get the hipblas handle for this executor.

Returns

the hipblas handle (hipblasContext*) for this executor

41.94.2.5 get_hipsparse_handle()

```
hipsparseContext* gko::HipExecutor::get_hipsparse_handle ( ) const [inline]
```

Get the hipsparse handle for this executor.

Returns

the hipsparse handle (hipsparseContext*) for this executor

41.94.2.6 get_master() [1/2]

```
std::shared_ptr<const Executor> gko::HipExecutor::get_master ( ) const [override], [virtual],
[noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.94.2.7 get_master() [2/2]

```
std::shared_ptr<Executor> gko::HipExecutor::get_master ( ) [override], [virtual], [noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.94.2.8 run()

```
void gko::HipExecutor::run (
            const Operation & op ) const [override], [virtual]
```

Runs the specified Operation using this Executor.

Parameters

op	the operation to run

Implements gko::Executor.

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

- ginkgo/core/base/executor.hpp

41.95 gko::HipfftError Class Reference

HipfftError is thrown when a hipFFT routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- HipfftError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a hipFFT error.

41.95.1 Detailed Description

HipfftError is thrown when a hipFFT routine throws a non-zero error code.

41.95.2 Constructor & Destructor Documentation

41.95.2.1 HipfftError()

```
gko::HipfftError::HipfftError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code )  [inline]
```

Initializes a hipFFT 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 hipFFT routine that failed
error_code	The resulting hipFFT error code

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

- ginkgo/core/base/exception.hpp

41.96 gko::HiprandError Class Reference

HiprandError is thrown when a hipRAND routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- HiprandError (const std::string &file, int line, const std::string &func, int64 error_code)

    Initializes a hipRAND error.

41.96.1 Detailed Description

HiprandError is thrown when a hipRAND routine throws a non-zero error code.

41.96.2 Constructor & Destructor Documentation

41.96.2.1 HiprandError()

```
gko::HiprandError::HiprandError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code )  [inline]
```

Initializes a hipRAND 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 hipRAND routine that failed
error_code	The resulting hipRAND error code

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

- ginkgo/core/base/exception.hpp

41.97 gko::HipsparseError Class Reference

HipsparseError is thrown when a hipSPARSE routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- HipsparseError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a hipSPARSE error.

41.97.1 Detailed Description

HipsparseError is thrown when a hipSPARSE routine throws a non-zero error code.

41.97.2 Constructor & Destructor Documentation

41.97.2.1 HipsparseError()

```
gko::HipsparseError::HipsparseError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code ) [inline]
```

Initializes a hipSPARSE 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 hipSPARSE routine that failed
error_code	The resulting hipSPARSE error code

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

- ginkgo/core/base/exception.hpp

41.98 gko::matrix::Hybrid< ValueType, IndexType > Class Template Reference

HYBRID is a matrix format which splits the matrix into ELLPACK and COO format.

```
#include <ginkgo/core/matrix/hybrid.hpp>
```

Classes

- class automatic

    automatic is a strategy_type which decides the number of stored elements per row of the ell part automatically.
- class column_limit

    column_limit is a strategy_type which decides the number of stored elements per row of the ell part by specifying the number of columns.
- class imbalance_bounded_limit

    imbalance_bounded_limit is a strategy_type which decides the number of stored elements per row of the ell part.
- class imbalance_limit

    imbalance_limit is a strategy_type which decides the number of stored elements per row of the ell part according to the percent.
- class minimal_storage_limit

    minimal_storage_limit is a strategy_type which decides the number of stored elements per row of the ell part.
- class strategy_type

    strategy_type is to decide how to set the hybrid config.

Public Member Functions

- void read (const mat_data &data) override

    Reads a matrix from a matrix_data structure.
- void read (const device_mat_data &data) override

    Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override

    Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override

    Writes a matrix to a matrix_data structure.
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override

    Extracts the diagonal entries of the matrix into a vector.
- std::unique_ptr< absolute_type > compute_absolute () const override

    Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override

    Compute absolute inplace on each element.
- value_type * get_ell_values () noexcept

    Returns the values of the ell part.
- const value_type * get_const_ell_values () const noexcept

    Returns the values of the ell part.

- index_type ∗ get_ell_col_idxs () noexcept
  Returns the column indexes of the ell part.
- const index_type ∗ get_const_ell_col_idxs () const noexcept
  Returns the column indexes of the ell part.
- size_type get_ell_num_stored_elements_per_row () const noexcept
  Returns the number of stored elements per row of ell part.
- size_type get_ell_stride () const noexcept
  Returns the stride of the ell part.
- size_type get_ell_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the ell part.
- value_type & ell_val_at (size_type row, size_type idx) noexcept
  Returns the `idx`-th non-zero element of the `row`-th row in the ell part.
- value_type ell_val_at (size_type row, size_type idx) const noexcept
  Returns the `idx`-th non-zero element of the `row`-th row in the ell part.
- index_type & ell_col_at (size_type row, size_type idx) noexcept
  Returns the `idx`-th column index of the `row`-th row in the ell part.
- index_type ell_col_at (size_type row, size_type idx) const noexcept
  Returns the `idx`-th column index of the `row`-th row in the ell part.
- const ell_type ∗ get_ell () const noexcept
  Returns the matrix of the ell part.
- value_type ∗ get_coo_values () noexcept
  Returns the values of the coo part.
- const value_type ∗ get_const_coo_values () const noexcept
  Returns the values of the coo part.
- index_type ∗ get_coo_col_idxs () noexcept
  Returns the column indexes of the coo part.
- const index_type ∗ get_const_coo_col_idxs () const noexcept
  Returns the column indexes of the coo part.
- index_type ∗ get_coo_row_idxs () noexcept
  Returns the row indexes of the coo part.
- const index_type ∗ get_const_coo_row_idxs () const noexcept
  Returns the row indexes of the coo part.
- size_type get_coo_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the coo part.
- const coo_type ∗ get_coo () const noexcept
  Returns the matrix of the coo part.
- size_type get_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the matrix.
- std::shared_ptr< strategy_type > get_strategy () const noexcept
  Returns the strategy.
- template< typename HybType >
  std::shared_ptr< typename HybType::strategy_type > get_strategy () const
  Returns the current strategy allowed in given hybrid format.
- Hybrid & operator= (const Hybrid &other)
  Copies data from another Hybrid.

41.98.1 Detailed Description

template< typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >

HYBRID is a matrix format which splits the matrix into ELLPACK and COO format.

Achieve the excellent performance with a proper partition of ELLPACK and COO.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.98.2 Member Function Documentation

41.98.2.1 compute_absolute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<absolute_type> gko::matrix::Hybrid< ValueType, IndexType >::compute_absolute
( ) const  [override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Hybrid< ValueType, IndexType > > >.

41.98.2.2 ell_col_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type gko::matrix::Hybrid< ValueType, IndexType >::ell_col_at (
            size_type row,
            size_type idx ) const  [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row in the ell part.

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

```
{
    return ell_->col_at(row, idx);
}
```

41.98.2.3 ell_col_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type& gko::matrix::Hybrid< ValueType, IndexType >::ell_col_at (
            size_type row,
            size_type idx )  [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row in the ell part.

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

41.98.2.4 ell_val_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type gko::matrix::Hybrid< ValueType, IndexType >::ell_val_at (
            size_type row,
            size_type idx ) const  [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row in the ell part.

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

41.98.2.5 ell_val_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type& gko::matrix::Hybrid< ValueType, IndexType >::ell_val_at (
            size_type row,
            size_type idx )  [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row in the ell part.

Parameters

row	the row of the requested element
idx	the idx-th stored element of the row

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the OMP results in a runtime error)

41.98.2.6 extract_diagonal()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Hybrid< ValueType, IndexType >::extract↩
_diagonal ( ) const  [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.98.2.7 get_const_coo_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Hybrid< ValueType, IndexType >::get_const_coo_col_idxs ( )
const  [inline], [noexcept]
```

Returns the column indexes of the coo part.

Returns

the column indexes of the coo part.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

41.98.2.8 get_const_coo_row_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Hybrid< ValueType, IndexType >::get_const_coo_row_idxs ( )
const [inline], [noexcept]
```

Returns the row indexes of the coo part.

Returns

the row indexes of the coo part.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

41.98.2.9 get_const_coo_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Hybrid< ValueType, IndexType >::get_const_coo_values ( ) const
[inline], [noexcept]
```

Returns the values of the coo part.

Returns

the values of the coo part.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

41.98.2.10 get_const_ell_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Hybrid< ValueType, IndexType >::get_const_ell_col_idxs ( )
const [inline], [noexcept]
```

Returns the column indexes of the ell part.

Returns

the column indexes of the ell part

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

41.98.2.11 get_const_ell_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Hybrid< ValueType, IndexType >::get_const_ell_values ( ) const
[inline], [noexcept]
```

Returns the values of the ell part.

Returns

the values of the ell part

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

41.98.2.12 get_coo()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const coo_type* gko::matrix::Hybrid< ValueType, IndexType >::get_coo ( ) const  [inline],
[noexcept]
```

Returns the matrix of the coo part.

Returns

the matrix of the coo part

41.98.2.13 get_coo_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Hybrid< ValueType, IndexType >::get_coo_col_idxs ( )  [inline], [noexcept]
```

Returns the column indexes of the coo part.

Returns

the column indexes of the coo part.

41.98.2.14 get_coo_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::get_coo_num_stored_elements ( ) const
[inline], [noexcept]
```

Returns the number of elements explicitly stored in the coo part.

Returns

the number of elements explicitly stored in the coo part

41.98.2.15 get_coo_row_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Hybrid< ValueType, IndexType >::get_coo_row_idxs ( ) [inline], [noexcept]
```

Returns the row indexes of the coo part.

Returns

the row indexes of the coo part.

41.98.2.16 get_coo_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Hybrid< ValueType, IndexType >::get_coo_values ( ) [inline], [noexcept]
```

Returns the values of the coo part.

Returns

the values of the coo part.

41.98.2.17 get_ell()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const ell_type* gko::matrix::Hybrid< ValueType, IndexType >::get_ell ( ) const [inline],
[noexcept]
```

Returns the matrix of the ell part.

Returns

the matrix of the ell part

41.98.2.18 get_ell_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Hybrid< ValueType, IndexType >::get_ell_col_idxs ( ) [inline], [noexcept]
```

Returns the column indexes of the ell part.

Returns

the column indexes of the ell part

41.98.2.19 get_ell_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::get_ell_num_stored_elements ( ) const
[inline], [noexcept]
```

Returns the number of elements explicitly stored in the ell part.

Returns

the number of elements explicitly stored in the ell part

41.98.2.20 get_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::get_ell_num_stored_elements_per_row ( )
const [inline], [noexcept]
```

Returns the number of stored elements per row of ell part.

Returns

the number of stored elements per row of ell part

41.98.2.21 get_ell_stride()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::get_ell_stride ( ) const [inline],
[noexcept]
```

Returns the stride of the ell part.

Returns

the stride of the ell part

41.98.2.22 get_ell_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Hybrid< ValueType, IndexType >::get_ell_values ( )  [inline], [noexcept]
```

Returns the values of the ell part.

Returns

the values of the ell part

41.98.2.23 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::get_num_stored_elements ( ) const
[inline], [noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

41.98.2.24 get_strategy() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
template<typename HybType >
std::shared_ptr<typename HybType::strategy_type> gko::matrix::Hybrid< ValueType, IndexType
>::get_strategy ( ) const
```

Returns the current strategy allowed in given hybrid format.

Template Parameters

HybType	hybrid type

Returns

the strategy

41.98.2.25 get_strategy() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr< typename HybType::strategy_type > gko::matrix::Hybrid< ValueType, IndexType
>::get_strategy ( ) const  [inline], [noexcept]
```

Returns the strategy.

Returns

the strategy

41.98.2.26 operator=()

```
template<typename ValueType = default_precision, typename IndexType = int32>
Hybrid& gko::matrix::Hybrid< ValueType, IndexType >::operator= (
            const Hybrid< ValueType, IndexType > & other ) [inline]
```

Copies data from another Hybrid.

Parameters

other	the Hybrid to copy from

Returns

this

41.98.2.27 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Hybrid< ValueType, IndexType >::read (
            const device_mat_data & data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.98.2.28 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Hybrid< ValueType, IndexType >::read (
            const mat_data & data ) [override], [virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.98.2.29 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Hybrid< ValueType, IndexType >::read (
            device_mat_data && data )  [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.98.2.30 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Hybrid< ValueType, IndexType >::write (
            mat_data & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/coo.hpp
- ginkgo/core/matrix/hybrid.hpp

41.99 gko::preconditioner::Ic< LSolverType, IndexType > Class Template Reference

The Incomplete Cholesky (IC) preconditioner solves the equation $LL^H * x = b$ for a given lower triangular matrix L and the right hand side b (can contain multiple right hand sides).

`#include <ginkgo/core/preconditioner/ic.hpp>`

Public Member Functions

- std::shared_ptr< const l_solver_type > get_l_solver () const
  Returns the solver which is used for the provided L matrix.
- std::shared_ptr< const lh_solver_type > get_lh_solver () const
  Returns the solver which is used for the L^H matrix.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.99.1 Detailed Description

template<typename LSolverType = solver::LowerTrs<>, typename IndexType = int32>
class gko::preconditioner::Ic< LSolverType, IndexType >

The Incomplete Cholesky (IC) preconditioner solves the equation $LL^H * x = b$ for a given lower triangular matrix L and the right hand side b (can contain multiple right hand sides).

It allows to set both the solver for L defaulting to solver::LowerTrs, which is a direct triangular solvers. The solver for L^H is the conjugate-transposed solver for L, ensuring that the preconditioner is symmetric and positive-definite. For this L solver, a factory can be provided (using `with_l_solver_factory`) to have more control over their behavior. In particular, it is possible to use an iterative method for solving the triangular systems. The default parameters for an iterative triangluar solver are:

- reduction factor = 1e-4
- max iteration = <number of="" rows="" of="" the="" matrix="" given="" to="" the="" solver>=""> Solvers without such criteria can also be used, in which case none are set.

An object of this class can be created with a matrix or a gko::Composition containing two matrices. If created with a matrix, it is factorized before creating the solver. If a gko::Composition (containing two matrices) is used, the first operand will be taken as the L matrix, the second will be considered the L^H matrix, which helps to avoid the otherwise necessary transposition of L inside the solver. ParIc can be directly used, since it orders the factors in the correct way.

Note

When providing a gko::Composition, the first matrix must be the lower matrix (L), and the second matrix must be its conjugate-transpose (L^H). If they are swapped, solving might crash or return the wrong result.

Do not use symmetric solvers (like CG) for the L solver since both matrices (L and L^H) are, by design, not symmetric.

This class is not thread safe (even a const object is not) because it uses an internal cache to accelerate multiple (sequential) applies. Using it in parallel can lead to segmentation faults, wrong results and other unwanted behavior.

Template Parameters

LSolverType	type of the solver used for the L matrix. Defaults to solver::LowerTrs
IndexType	type of the indices when ParIc is used to generate the L and L^H factors. Irrelevant otherwise.

41.99.2 Member Function Documentation

41.99.2.1 conj_transpose()

```
template<typename LSolverType = solver::LowerTrs<>, typename IndexType = int32>
std::unique_ptr<LinOp> gko::preconditioner::Ic< LSolverType, IndexType >::conj_transpose ( )
const [inline], [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

```
    {
        std::unique_ptr<transposed_type> transposed{
            new transposed_type{this->get_executor()}};
        transposed->set_size(gko::transpose(this->get_size()));
        transposed->l_solver_ =
            share(as<typename lh_solver_type::transposed_type>(
                this->get_lh_solver()->conj_transpose()));
        transposed->lh_solver_ =
            share(as<typename l_solver_type::transposed_type>(
                this->get_l_solver()->conj_transpose()));

        return std::move(transposed);
    }
```

References gko::PolymorphicObject::get_executor(), gko::preconditioner::Ic< LSolverType, IndexType >::get_l_←
solver(), gko::preconditioner::Ic< LSolverType, IndexType >::get_lh_solver(), gko::share(), and gko::transpose().

41.99.2.2 get_l_solver()

```
template<typename LSolverType = solver::LowerTrs<>, typename IndexType = int32>
std::shared_ptr<const l_solver_type> gko::preconditioner::Ic< LSolverType, IndexType >::get←
_l_solver ( ) const [inline]
```

Returns the solver which is used for the provided L matrix.

Returns

the solver which is used for the provided L matrix

Referenced by gko::preconditioner::Ic< LSolverType, IndexType >::conj_transpose(), and gko::preconditioner::Ic< LSolverType, IndexType >::transpose().

41.99.2.3 get_lh_solver()

```
template<typename LSolverType = solver::LowerTrs<>, typename IndexType = int32>
std::shared_ptr<const lh_solver_type> gko::preconditioner::Ic< LSolverType, IndexType >↩
::get_lh_solver ( ) const  [inline]
```

Returns the solver which is used for the L^H matrix.

Returns

the solver which is used for the L^H matrix

Referenced by gko::preconditioner::Ic< LSolverType, IndexType >::conj_transpose(), and gko::preconditioner::Ic< LSolverType, IndexType >::transpose().

41.99.2.4 transpose()

```
template<typename LSolverType = solver::LowerTrs<>, typename IndexType = int32>
std::unique_ptr<LinOp> gko::preconditioner::Ic< LSolverType, IndexType >::transpose ( ) const
[inline], [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

References gko::PolymorphicObject::get_executor(), gko::preconditioner::Ic< LSolverType, IndexType >::get_l_↩ solver(), gko::preconditioner::Ic< LSolverType, IndexType >::get_lh_solver(), gko::share(), and gko::transpose().

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

- ginkgo/core/preconditioner/ic.hpp

41.100 gko::factorization::Ic< ValueType, IndexType > Class Template Reference

Represents an incomplete Cholesky factorization (IC(0)) of a sparse matrix.

```
#include <ginkgo/core/factorization/ic.hpp>
```

Additional Inherited Members

41.100.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class gko::factorization::Ic< ValueType, IndexType >

Represents an incomplete Cholesky factorization (IC(0)) of a sparse matrix.

More specifically, it consists of a lower triangular factor L and its conjugate transpose L^H with sparsity pattern $\mathcal{S}(L + L^H)$ = $\mathcal{S}(A)$ fulfilling $LL^H = A$ at every non-zero location of A.

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

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

- ginkgo/core/factorization/ic.hpp

41.101 gko::matrix::Identity< ValueType > Class Template Reference

This class is a utility which efficiently implements the identity matrix (a linear operator which maps each vector to itself).

```
#include <ginkgo/core/matrix/identity.hpp>
```

Public Member Functions

- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.101.1 Detailed Description

template<typename ValueType = default_precision>
class gko::matrix::Identity< ValueType >

This class is a utility which efficiently implements the identity matrix (a linear operator which maps each vector to itself).

Thus, objects of the Identity class always represent a square matrix, and don't require any storage for their values. The apply method is implemented as a simple copy (or a linear combination).

Note

This class is useful when composing it with other operators. For example, it can be used instead of a preconditioner in Krylov solvers, if one wants to run a "plain" solver, without using a preconditioner.

Template Parameters

ValueType	precision of matrix elements

41.101.2 Member Function Documentation

41.101.2.1 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Identity< ValueType >::conj_transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.101.2.2 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::matrix::Identity< ValueType >::transpose ( ) const  [override],
[virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/matrix/identity.hpp

41.102 gko::matrix::IdentityFactory< ValueType > Class Template Reference

This factory is a utility which can be used to generate Identity operators.

```
#include <ginkgo/core/matrix/identity.hpp>
```

Static Public Member Functions

- static std::unique_ptr< IdentityFactory > create (std::shared_ptr< const Executor > exec)

  Creates a new Identity factory.

Additional Inherited Members

41.102.1 Detailed Description

template<typename ValueType = default_precision>
class gko::matrix::IdentityFactory< ValueType >

This factory is a utility which can be used to generate Identity operators.

The factory will generate the Identity matrix with the same dimension as the passed in operator. It will throw an exception if the operator is not square.

Template Parameters

ValueType	precision of matrix elements

41.102.2 Member Function Documentation

41.102.2.1 create()

```
template<typename ValueType = default_precision>
static std::unique_ptr<IdentityFactory> gko::matrix::IdentityFactory< ValueType >::create (
            std::shared_ptr< const Executor > exec )  [inline], [static]
```

Creates a new Identity factory.

Parameters

exec	the executor where the Identity operator will be stored

Returns

a unique pointer to the newly created factory

```
    {
        return std::unique_ptr<IdentityFactory>(
            new IdentityFactory(std::move(exec)));
    }
```

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

- ginkgo/core/matrix/identity.hpp

41.103 gko::solver::Idr< ValueType > Class Template Reference

IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.

`#include <ginkgo/core/solver/idr.hpp>`

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const

  Gets the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override

Return true as iterative solvers use the data in x as an initial guess.

- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const

  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)

  Sets the stopping criterion of the solver.
- size_type get_subspace_dim () const

  Gets the subspace dimension of the solver.
- void set_subspace_dim (const size_type other)

  Sets the subspace dimension of the solver.
- remove_complex< ValueType > get_kappa () const

  Gets the kappa parameter of the solver.
- void set_kappa (const remove_complex< ValueType > other)

  Sets the kappa parameter of the solver.
- bool get_deterministic () const

  Gets the deterministic parameter of the solver.
- void set_deterministic (const bool other)

  Sets the deterministic parameter of the solver.
- bool get_complex_subspace () const

  Gets the complex_subspace parameter of the solver.
- void set_complex_subpsace (const bool other)

  Sets the complex_subspace parameter of the solver.

41.103.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Idr< ValueType >

IDR(s) is an efficient method for solving large nonsymmetric systems of linear equations.

The implemented version is the one presented in the paper "Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties" by M. B. Van Gijzen and P. Sonneveld.

The method is based on the induced dimension reduction theorem which provides a way to construct subsequent residuals that lie in a sequence of shrinking subspaces. These subspaces are spanned by s vectors which are first generated randomly and then orthonormalized. They are stored in a dense matrix.

Template Parameters

ValueType	precision of the elements of the system matrix.

41.103.2 Member Function Documentation

41.103.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Idr< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

```
111 { return true; }
```

41.103.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Idr< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.103.2.3 get_complex_subspace()

```
template<typename ValueType = default_precision>
bool gko::solver::Idr< ValueType >::get_complex_subspace ( ) const [inline]
```

Gets the complex_subspace parameter of the solver.

Returns

the complex_subspace parameter

41.103.2.4 get_deterministic()

```
template<typename ValueType = default_precision>
bool gko::solver::Idr< ValueType >::get_deterministic ( ) const [inline]
```

Gets the deterministic parameter of the solver.

Returns

the deterministic parameter

41.103.2.5 get_kappa()

```
template<typename ValueType = default_precision>
remove_complex<ValueType> gko::solver::Idr< ValueType >::get_kappa ( ) const [inline]
```

Gets the kappa parameter of the solver.

Returns

the kappa parameter

41.103.2.6 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Idr< ValueType >::get_stop_↩
criterion_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.103.2.7 get_subspace_dim()

```
template<typename ValueType = default_precision>
size_type gko::solver::Idr< ValueType >::get_subspace_dim ( ) const [inline]
```

Gets the subspace dimension of the solver.

Returns

the subspace Dimension

41.103.2.8 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Idr< ValueType >::get_system_matrix ( ) const [inline]
```

Gets the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.103.2.9 set_complex_subpsace()

```
template<typename ValueType = default_precision>
void gko::solver::Idr< ValueType >::set_complex_subpsace (
            const bool other ) [inline]
```

Sets the complex_subspace parameter of the solver.

Parameters

other	the new complex_subspace parameter

41.103.2.10 set_deterministic()

```
template<typename ValueType = default_precision>
void gko::solver::Idr< ValueType >::set_deterministic (
            const bool other ) [inline]
```

Sets the deterministic parameter of the solver.

Parameters

other	the new deterministic parameter

41.103.2.11 set_kappa()

```
template<typename ValueType = default_precision>
void gko::solver::Idr< ValueType >::set_kappa (
            const remove_complex< ValueType > other ) [inline]
```

Sets the kappa parameter of the solver.

Parameters

other	the new kappa parameter

41.103.2.12 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Idr< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other ) [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.103.2.13 set_subspace_dim()

```
template<typename ValueType = default_precision>
void gko::solver::Idr< ValueType >::set_subspace_dim (
            const size_type other )  [inline]
```

Sets the subspace dimension of the solver.

Parameters

other	the new subspace Dimension

41.103.2.14 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Idr< ValueType >::transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/idr.hpp

41.104 gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType > Class Template Reference

The Incomplete LU (ILU) preconditioner solves the equation $LUx = b$ for a given lower triangular matrix L, an upper triangular matrix U and the right hand side b (can contain multiple right hand sides).

```
#include <ginkgo/core/preconditioner/ilu.hpp>
```

Public Member Functions

- std::shared_ptr< const l_solver_type > get_l_solver () const
  Returns the solver which is used for the provided L matrix.
- std::shared_ptr< const u_solver_type > get_u_solver () const
  Returns the solver which is used for the provided U matrix.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.104.1 Detailed Description

template<typename LSolverType = solver::LowerTrs<>, typename USolverType = solver::UpperTrs<>, bool ReverseApply = false, typename IndexType = int32>
class gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >

The Incomplete LU (ILU) preconditioner solves the equation $LUx = b$ for a given lower triangular matrix L, an upper triangular matrix U and the right hand side b (can contain multiple right hand sides).

It allows to set both the solver for L and the solver for U independently, while providing the defaults solver::LowerTrs and solver::UpperTrs, which are direct triangular solvers. For these solvers, a factory can be provided (with `with_l_solver_factory` and `with_u_solver_factory`) to have more control over their behavior. In particular, it is possible to use an iterative method for solving the triangular systems. The default parameters for an iterative triangluar solver are:

- reduction factor = 1e-4
- max iteration = <number of="" rows="" of="" the="" matrix="" given="" to="" the="" solver>=""> Solvers without such criteria can also be used, in which case none are set.

An object of this class can be created with a matrix or a gko::Composition containing two matrices. If created with a matrix, it is factorized before creating the solver. If a gko::Composition (containing two matrices) is used, the first operand will be taken as the L matrix, the second will be considered the U matrix. ParIlu can be directly used, since it orders the factors in the correct way.

Note

When providing a gko::Composition, the first matrix must be the lower matrix (L), and the second matrix must be the upper matrix (U). If they are swapped, solving might crash or return the wrong result.

Do not use symmetric solvers (like CG) for L or U solvers since both matrices (L and U) are, by design, not symmetric.

This class is not thread safe (even a const object is not) because it uses an internal cache to accelerate multiple (sequential) applies. Using it in parallel can lead to segmentation faults, wrong results and other unwanted behavior.

Template Parameters

LSolverType	type of the solver used for the L matrix. Defaults to solver::LowerTrs
USolverType	type of the solver used for the U matrix Defaults to solver::UpperTrs
ReverseApply	default behavior (ReverseApply = false) is first to solve with L (Ly = b) and then with U (Ux = y). When set to true, it will solve first with U, and then with L.
IndexTypeParIlu	Type of the indices when ParIlu is used to generate both L and U factors. Irrelevant otherwise.

41.104.2 Member Function Documentation

41.104.2.1 conj_transpose()

```
template<typename LSolverType = solver::LowerTrs<>, typename USolverType = solver::Upper↩
Trs<>, bool ReverseApply = false, typename IndexType = int32>
std::unique_ptr<LinOp> gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply,
IndexType >::conj_transpose ( ) const  [inline], [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

```
    {
        std::unique_ptr<transposed_type> transposed{
            new transposed_type{this->get_executor()}};
        transposed->set_size(gko::transpose(this->get_size()));
        transposed->l_solver_ =
            share(as<typename u_solver_type::transposed_type>(
                this->get_u_solver()->conj_transpose()));
        transposed->u_solver_ =
            share(as<typename l_solver_type::transposed_type>(
                this->get_l_solver()->conj_transpose()));

        return std::move(transposed);
    }
```

References gko::PolymorphicObject::get_executor(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::get_l_solver(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::get_u_solver(), gko::share(), and gko::transpose().

41.104.2.2 get_l_solver()

```
template<typename LSolverType = solver::LowerTrs<>, typename USolverType = solver::Upper↩
Trs<>, bool ReverseApply = false, typename IndexType = int32>
std::shared_ptr<const l_solver_type> gko::preconditioner::Ilu< LSolverType, USolverType,
ReverseApply, IndexType >::get_l_solver ( ) const  [inline]
```

Returns the solver which is used for the provided L matrix.

Returns

the solver which is used for the provided L matrix

Referenced by gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::conj_transpose(), and gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::transpose().

41.104.2.3 get_u_solver()

```
template<typename LSolverType = solver::LowerTrs<>, typename USolverType = solver::Upper↩
Trs<>, bool ReverseApply = false, typename IndexType = int32>
std::shared_ptr<const u_solver_type> gko::preconditioner::Ilu< LSolverType, USolverType,
ReverseApply, IndexType >::get_u_solver ( ) const  [inline]
```

Returns the solver which is used for the provided U matrix.

Returns

the solver which is used for the provided U matrix

Referenced by gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::conj_transpose(), and gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::transpose().

41.104.2.4 transpose()

```
template<typename LSolverType = solver::LowerTrs<>, typename USolverType = solver::Upper↩
Trs<>, bool ReverseApply = false, typename IndexType = int32>
std::unique_ptr<LinOp> gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply,
IndexType >::transpose ( ) const  [inline], [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

References gko::PolymorphicObject::get_executor(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::get_l_solver(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::get_u_solver(), gko::share(), and gko::transpose().

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

- ginkgo/core/preconditioner/ilu.hpp

41.105 gko::factorization::Ilu< ValueType, IndexType > Class Template Reference

Represents an incomplete LU factorization – ILU(0) – of a sparse matrix.

`#include <ginkgo/core/factorization/ilu.hpp>`

Additional Inherited Members

41.105.1 Detailed Description

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class gko::factorization::Ilu< ValueType, IndexType >

Represents an incomplete LU factorization – ILU(0) – of a sparse matrix.

More specifically, it consists of a lower unitriangular factor L and an upper triangular factor U with sparsity pattern $\mathcal{S}(L + U)$ = $\mathcal{S}(A)$ fulfilling $LU = A$ at every non-zero location of A.

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

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

- ginkgo/core/factorization/ilu.hpp

41.106 gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit Class Reference

imbalance_bounded_limit is a strategy_type which decides the number of stored elements per row of the ell part.

```
#include <ginkgo/core/matrix/hybrid.hpp>
```

Public Member Functions

- imbalance_bounded_limit (double percent=0.8, double ratio=0.0001)

  Creates a imbalance_bounded_limit strategy.
- size_type compute_ell_num_stored_elements_per_row (Array< size_type > ∗row_nnz) const override

  Computes the number of stored elements per row of the ell part.
- auto get_percentage () const

  Get the percent setting.
- auto get_ratio () const

  Get the ratio setting.

41.106.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit

imbalance_bounded_limit is a strategy_type which decides the number of stored elements per row of the ell part.

It uses the imbalance_limit and adds the upper bound of the number of ell's cols by the number of rows.

41.106.2 Member Function Documentation

41.106.2.1 compute_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::compute_ell_←
num_stored_elements_per_row (
            Array< size_type > ∗ row_nnz ) const  [inline], [override], [virtual]
```

Computes the number of stored elements per row of the ell part.

Parameters

row_nnz	the number of nonzeros of each row

Returns

the number of stored elements per row of the ell part

Implements gko::matrix::Hybrid< ValueType, IndexType >::strategy_type.

References gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::compute_ell_num_stored_elements_per_row(), and gko::Array< ValueType >::get_num_elems().

Referenced by gko::matrix::Hybrid< ValueType, IndexType >::automatic::compute_ell_num_stored_elements_per_row().

41.106.2.2 get_percentage()

```
template<typename ValueType = default_precision, typename IndexType = int32>
auto gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::get_percentage ( )
const [inline]
```

Get the percent setting.

@retrun percent

References gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::get_percentage().

41.106.2.3 get_ratio()

```
template<typename ValueType = default_precision, typename IndexType = int32>
auto gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::get_ratio ( ) const
[inline]
```

Get the ratio setting.

@retrun ratio

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

- ginkgo/core/matrix/hybrid.hpp

41.107 gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit Class Reference

imbalance_limit is a strategy_type which decides the number of stored elements per row of the ell part according to the percent.

```
#include <ginkgo/core/matrix/hybrid.hpp>
```

Public Member Functions

- imbalance_limit (double percent=0.8)

  Creates a imbalance_limit strategy.
- size_type compute_ell_num_stored_elements_per_row (Array< size_type > *row_nnz) const override

  Computes the number of stored elements per row of the ell part.
- auto get_percentage () const

  Get the percent setting.

41.107.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit

imbalance_limit is a strategy_type which decides the number of stored elements per row of the ell part according to the percent.

It sorts the number of nonzeros of each row and takes the value at the position `floor(percent * num_row)` as the number of stored elements per row of the ell part. Thus, at least `percent` rows of all are in the ell part.

41.107.2 Constructor & Destructor Documentation

41.107.2.1 imbalance_limit()

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::imbalance_limit (
            double percent = 0.8 )  [inline], [explicit]
```

Creates a imbalance_limit strategy.

Parameters

percent	the row_nnz[floor(num_rows*percent)] is the number of stored elements per row of the ell part

41.107.3 Member Function Documentation

41.107.3.1 compute_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::compute_ell_num_↩
stored_elements_per_row (
            Array< size_type > * row_nnz ) const  [inline], [override], [virtual]
```

Computes the number of stored elements per row of the ell part.

Parameters

row_nnz	the number of nonzeros of each row

Returns

the number of stored elements per row of the ell part

Implements gko::matrix::Hybrid< ValueType, IndexType >::strategy_type.

References gko::Array< ValueType >::get_data(), and gko::Array< ValueType >::get_num_elems().

Referenced by gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::compute_ell_num_stored_elements_per_row(), and gko::matrix::Hybrid< ValueType, IndexType >::minimal_storage_limit::compute_ell_num_stored_elements_per_row().

41.107.3.2 get_percentage()

```
template<typename ValueType = default_precision, typename IndexType = int32>
auto gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::get_percentage ( ) const
[inline]
```

Get the percent setting.

@retrun percent

Referenced by gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit::get_percentage(), and gko::matrix::Hybrid< ValueType, IndexType >::minimal_storage_limit::get_percentage().

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

- ginkgo/core/matrix/hybrid.hpp

41.108 gko::stop::ImplicitResidualNorm< ValueType > Class Template Reference

The ImplicitResidualNorm class is a stopping criterion which stops the iteration process when the implicit residual norm is below a certain threshold relative to.

```
#include <ginkgo/core/stop/residual_norm.hpp>
```

41.108.1 Detailed Description

template<typename ValueType = default_precision>
class gko::stop::ImplicitResidualNorm< ValueType >

The ImplicitResidualNorm class is a stopping criterion which stops the iteration process when the implicit residual norm is below a certain threshold relative to.

1. the norm of the right-hand side, implicit_resnorm / norm(right_hand_side) < threshold
2. the initial residual, implicit_resnorm / norm(initial_residual) < < threshold.
3. one, implicit_resnorm < threshold.

Note

To use this stopping criterion there are some dependencies. The constructor depends on either `b` or the `initial_residual` in order to compute their norms. If this is not correctly provided, an exception ::gko↩ ::NotSupported() is thrown.

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

- ginkgo/core/stop/residual_norm.hpp

41.109 gko::index_set< IndexType > Class Template Reference

An index set class represents an ordered set of intervals.

`#include <ginkgo/core/base/index_set.hpp>`

Public Types

- using index_type = IndexType

  The type of elements stored in the index set.

Public Member Functions

- index_set (std::shared_ptr< const Executor > exec) noexcept

  Creates an empty index_set tied to the specified Executor.
- index_set (std::shared_ptr< const gko::Executor > exec, std::initializer_list< IndexType > init_list, const bool is_sorted=false)

  Creates an index set on the specified executor from the initializer list.
- index_set (std::shared_ptr< const gko::Executor > exec, const index_type size, const gko::Array< index_type > &indices, const bool is_sorted=false)

  Creates an index set on the specified executor and the given size.
- index_set (std::shared_ptr< const Executor > exec, const index_set &other)

  Creates a copy of the input index_set on a different executor.
- index_set (const index_set &other)

Creates a copy of the input index_set.

- index_set (std::shared_ptr< const Executor > exec, index_set &&other)

  Moves the input index_set to a different executor.
- index_set (index_set &&other)

  Moves the input index_set.
- index_set & operator= (const index_set &other)

  Copies data from another index_set.
- index_set & operator= (index_set &&other)

  Moves data from another index_set.
- void clear () noexcept

  Deallocates all data used by the index_set.
- std::shared_ptr< const Executor > get_executor () const

  Returns the executor of the index_set.
- index_type get_size () const

  Returns the size of the index set space.
- bool is_contiguous () const

  Returns if the index set is contiguous.
- index_type get_num_elems () const

  Return the actual number of indices stored in the index set.
- index_type get_global_index (index_type local_index) const

  Return the global index given a local index.
- index_type get_local_index (index_type global_index) const

  Return the local index given a global index.
- Array< index_type > map_local_to_global (const Array< index_type > &local_indices, const bool is_↩ sorted=false) const

  This is an array version of the scalar function above.
- Array< index_type > map_global_to_local (const Array< index_type > &global_indices, const bool is_↩ sorted=false) const

  This is an array version of the scalar function above.
- Array< index_type > to_global_indices () const

  This function allows the user obtain a decompresed global_indices Array from the indices stored in the index set.
- Array< bool > contains (const Array< index_type > &global_indices, const bool is_sorted=false) const

  Checks if the individual global indeices exist in the index set.
- bool contains (const index_type global_index) const

  Checks if the global index exists in the index set.
- index_type get_num_subsets () const

  Returns the number of subsets stored in the index set.
- const index_type ∗ get_subsets_begin () const

  Returns a pointer to the beginning indices of the subsets.
- const index_type ∗ get_subsets_end () const

  Returns a pointer to the end indices of the subsets.
- const index_type ∗ get_superset_indices () const

  Returns a pointer to the cumulative indices of the superset of the subsets.

41.109.1 Detailed Description

template<typename IndexType = int32>
class gko::index_set< IndexType >

An index set class represents an ordered set of intervals.

The index set contains subsets which store the starting and end points of a range, [a,b), storing the first index and one past the last index. As the index set only stores the end-points of ranges, it can be quite efficient in terms of storage.

This class is particularly useful in storing continuous ranges. For example, consider the index set (1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 18, 19, 20, 21, 42). Instead of storing the entire array of indices, one can store intervals ([1,9), [10,13), [18,22), [42,43)), thereby only using half the storage.

We store three arrays, one (subsets_begin) with the starting indices of the subsets in the index set, another (subsets_end) storing one index beyond the end indices of the subsets and the last (superset_cumulative_indices) storing the cumulative number of indices in the subsequent subsets with an initial zero which speeds up the querying. Additionally, the arrays conataining the range boundaries (subsets_begin, subsets_end) are stored in a sorted fashion.

Therefore the storage would look as follows

> index_set = (1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 18, 19, 20, 21, 42) subsets_begin = {1, 10, 18, 42} subsets_end = {9, 13, 22, 43} superset_cumulative_indices = {0, 8, 11, 15, 16}

Template Parameters

index_type	type of the indices being stored in the index set.

41.109.2 Constructor & Destructor Documentation

41.109.2.1 index_set() [1/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
              std::shared_ptr< const Executor > exec )  [inline], [explicit], [noexcept]
```

Creates an empty index_set tied to the specified Executor.

Parameters

exec	the Executor where the index_set data is allocated

```
        : exec_(std::move(exec)),
          index_space_size_{0},
          num_stored_indices_{0},
          subsets_begin_{Array<index_type>(exec_)},
          subsets_end_{Array<index_type>(exec_)},
```

```
        superset_cumulative_indices_{Array<index_type>(exec_)}
    {}
```

41.109.2.2 index_set() [2/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
            std::shared_ptr< const gko::Executor > exec,
            std::initializer_list< IndexType > init_list,
            const bool is_sorted = false ) [inline], [explicit]
```

Creates an index set on the specified executor from the initializer list.

Parameters

exec	the Executor where the index set data will be allocated
init_list	the indices that the index set should hold in an initializer_list.
is_sorted	a parameter that specifies if the indices array is sorted or not. `true` if sorted.

41.109.2.3 index_set() [3/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
            std::shared_ptr< const gko::Executor > exec,
            const index_type size,
            const gko::Array< index_type > & indices,
            const bool is_sorted = false ) [inline], [explicit]
```

Creates an index set on the specified executor and the given size.

Parameters

exec	the Executor where the index set data will be allocated
size	the maximum index the index set it allowed to hold. This is the size of the index space.
indices	the indices that the index set should hold.
is_sorted	a parameter that specifies if the indices array is sorted or not. `true` if sorted.

References gko::Array< ValueType >::get_num_elems().

41.109.2.4 index_set() [4/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
```

```
std::shared_ptr< const Executor > exec,
const index_set< IndexType > & other ) [inline]
```

Creates a copy of the input index_set on a different executor.

Parameters

exec	the executor where the new index_set will be created
other	the index_set to copy from

41.109.2.5 index_set() [5/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
            const index_set< IndexType > & other ) [inline]
```

Creates a copy of the input index_set.

Parameters

other	the index_set to copy from

41.109.2.6 index_set() [6/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
            std::shared_ptr< const Executor > exec,
            index_set< IndexType > && other ) [inline]
```

Moves the input index_set to a different executor.

Parameters

exec	the executor where the new index_set will be moved to
other	the index_set to move from

41.109.2.7 index_set() [7/7]

```
template<typename IndexType = int32>
gko::index_set< IndexType >::index_set (
            index_set< IndexType > && other ) [inline]
```

Moves the input index_set.

Parameters

other	the index_set to move from

41.109.3 Member Function Documentation

41.109.3.1 clear()

```
template<typename IndexType = int32>
void gko::index_set< IndexType >::clear ( )  [inline], [noexcept]
```

Deallocates all data used by the index_set.

The index_set is left in a valid, but empty state, so the same index_set can be used to allocate new memory. Calls to index_set::get_subsets_begin() will return a `nullptr`.

References gko::Array< ValueType >::clear().

41.109.3.2 contains() [1/2]

```
template<typename IndexType = int32>
Array<bool> gko::index_set< IndexType >::contains (
            const Array< index_type > & global_indices,
            const bool is_sorted = false ) const
```

Checks if the individual global indeices exist in the index set.

Parameters

global_indices	the indices to check.
is_sorted	a parameter that specifies if the query array is sorted or not. `true` if sorted.

Returns

the Array that contains element wise whether the corresponding global index in the index set or not.

41.109.3.3 contains() [2/2]

```
template<typename IndexType = int32>
bool gko::index_set< IndexType >::contains (
            const index_type global_index ) const
```

Checks if the global index exists in the index set.

Parameters

global_index	the index to check.

Returns

whether the element exists in the index set.

Warning

This single entry query can have significant kernel launch overheads and should be avoided if possible.

41.109.3.4 get_executor()

```
template<typename IndexType = int32>
std::shared_ptr<const Executor> gko::index_set< IndexType >::get_executor ( ) const [inline]
```

Returns the executor of the index_set.

Returns

the executor.

41.109.3.5 get_global_index()

```
template<typename IndexType = int32>
index_type gko::index_set< IndexType >::get_global_index (
            index_type local_index ) const
```

Return the global index given a local index.

Consider the set idx_set = (0, 1, 2, 4, 6, 7, 8, 9). This function returns the element at the global index k stored in the index set. For example, `idx_set.get_global_index(0) == 0` `idx_set.get_global_index(3) == 4` and `idx_set.get_global_index(7) == 9`

Note

This function returns a scalar value and needs a scalar value. For repeated queries, it is more efficient to use the Array functions that take and return arrays which allow for more throughput.

Parameters

local_index	the local index.

Returns

the global index from the index set.

Warning

This single entry query can have significant kernel launch overheads and should be avoided if possible.

41.109.3.6 get_local_index()

```
template<typename IndexType = int32>
index_type gko::index_set< IndexType >::get_local_index (
            index_type global_index ) const
```

Return the local index given a global index.

Consider the set idx_set = (0, 1, 2, 4, 6, 7, 8, 9). This function returns the local index in the index set of the provided index set. For example, `idx_set.get_local_index(0) == 0` `idx_set.get_local_index(4) == 3` and `idx_set.get_local_index(6) == 4`.

Note

This function returns a scalar value and needs a scalar value. For repeated queries, it is more efficient to use the Array functions that take and return arrays which allow for more throughput.

Parameters

global_index	the global index.

Returns

the local index of the element in the index set.

Warning

This single entry query can have significant kernel launch overheads and should be avoided if possible.

41.109.3.7 get_num_elems()

```
template<typename IndexType = int32>
index_type gko::index_set< IndexType >::get_num_elems ( ) const [inline]
```

Return the actual number of indices stored in the index set.

Returns

number of indices stored in the index set

41.109.3.8 get_num_subsets()

```
template<typename IndexType = int32>
index_type gko::index_set< IndexType >::get_num_subsets ( ) const [inline]
```

Returns the number of subsets stored in the index set.

Returns

the number of stored subsets.

References gko::Array< ValueType >::get_num_elems().

Referenced by gko::index_set< IndexType >::is_contiguous().

41.109.3.9 get_size()

```
template<typename IndexType = int32>
index_type gko::index_set< IndexType >::get_size ( ) const [inline]
```

Returns the size of the index set space.

Returns

the size of the index set space.

41.109.3.10 get_subsets_begin()

```
template<typename IndexType = int32>
const index_type* gko::index_set< IndexType >::get_subsets_begin ( ) const [inline]
```

Returns a pointer to the beginning indices of the subsets.

Returns

a pointer to the beginning indices of the subsets.

References gko::Array< ValueType >::get_const_data().

41.109.3.11 get_subsets_end()

```
template<typename IndexType = int32>
const index_type* gko::index_set< IndexType >::get_subsets_end ( ) const [inline]
```

Returns a pointer to the end indices of the subsets.

Returns

a pointer to the end indices of the subsets.

References gko::Array< ValueType >::get_const_data().

41.109.3.12 get_superset_indices()

```
template<typename IndexType = int32>
const index_type* gko::index_set< IndexType >::get_superset_indices ( ) const [inline]
```

Returns a pointer to the cumulative indices of the superset of the subsets.

Returns

a pointer to the cumulative indices of the superset of the subsets.

References gko::Array< ValueType >::get_const_data().

41.109.3.13 is_contiguous()

```
template<typename IndexType = int32>
bool gko::index_set< IndexType >::is_contiguous ( ) const [inline]
```

Returns if the index set is contiguous.

Returns

if the index set is contiguous.

References gko::index_set< IndexType >::get_num_subsets().

41.109.3.14 map_global_to_local()

```
template<typename IndexType = int32>
Array<index_type> gko::index_set< IndexType >::map_global_to_local (
            const Array< index_type > & global_indices,
            const bool is_sorted = false ) const
```

This is an array version of the scalar function above.

Parameters

global_indices	the global index array.
is_sorted	a parameter that specifies if the query array is sorted or not. `true` if sorted.

Returns

the local index array from the index set.

Note

Whenever possible, passing a sorted array is preferred as the queries can be significantly faster.

41.109.3.15 map_local_to_global()

```
template<typename IndexType = int32>
Array<index_type> gko::index_set< IndexType >::map_local_to_global (
            const Array< index_type > & local_indices,
            const bool is_sorted = false ) const
```

This is an array version of the scalar function above.

Parameters

local_indices	the local index array.
is_sorted	a parameter that specifies if the query array is sorted or not. `true` if sorted .

Returns

the global index array from the index set.

Note

Whenever possible, passing a sorted array is preferred as the queries can be significantly faster.

Passing local indices from [0, size) is equivalent to using the @to_global_indices function.

41.109.3.16 operator=() [1/2]

```
template<typename IndexType = int32>
index_set& gko::index_set< IndexType >::operator= (
            const index_set< IndexType > & other )  [inline]
```

Copies data from another index_set.

The executor of this is preserved. In case this does not have an assigned executor, it will inherit the executor of other.

Parameters

other	the index_set to copy from

Returns

this

41.109.3.17 operator=() [2/2]

```
template<typename IndexType = int32>
index_set& gko::index_set< IndexType >::operator= (
            index_set< IndexType > && other ) [inline]
```

Moves data from another index_set.

The executor of this is preserved. In case this does not have an assigned executor, it will inherit the executor of other.

Parameters

other	the index_set to move from

Returns

this

41.109.3.18 to_global_indices()

```
template<typename IndexType = int32>
Array<index_type> gko::index_set< IndexType >::to_global_indices ( ) const
```

This function allows the user obtain a decompresed global_indices Array from the indices stored in the index set.

Returns

the decompressed set of indices.

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

- ginkgo/core/base/index_set.hpp

41.110 gko::solver::Ir< ValueType > Class Template Reference

Iterative refinement (IR) is an iterative method that uses another coarse method to approximate the error of the current solution via the current residual.

`#include <ginkgo/core/solver/ir.hpp>`

Public Member Functions

- std::shared_ptr< const LinOp > get_system_matrix () const
  Returns the system operator (matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- bool apply_uses_initial_guess () const override
  Return true as iterative solvers use the data in x as an initial guess.
- std::shared_ptr< const LinOp > get_solver () const
  Returns the solver operator used as the inner solver.
- void set_solver (std::shared_ptr< const LinOp > new_solver)
  Sets the solver operator used as the inner solver.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const
  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)
  Sets the stopping criterion of the solver.

41.110.1 Detailed Description

template<typename ValueType = default_precision>
class gko::solver::Ir< ValueType >

Iterative refinement (IR) is an iterative method that uses another coarse method to approximate the error of the current solution via the current residual.

Moreover, it can be also considered as preconditioned Richardson iteration with relaxation factor = 1.

For any approximation of the solution `solution` to the system `Ax = b`, the residual is defined as: `residual = b - A solution`. The error in `solution`, `e = x - solution` (with `x` being the exact solution) can be obtained as the solution to the residual equation `Ae = residual`, since `A e = Ax - A solution = b - A solution = residual`. Then, the real solution is computed as `x = relaxation_factor * solution + e`. Instead of accurately solving the residual equation `Ae = residual`, the solution of the system `e` can be approximated to obtain the approximation `error` using a coarse method `solver`, which is used to update `solution`, and the entire process is repeated with the updated `solution`. This yields the iterative refinement method:

```
solution = initial_guess
while not converged:
    residual = b - A solution
    error = solver(A, residual)
    solution = solution + relaxation_factor * error
```

With `relaxation_factor` equal to 1 (default), the solver is Iterative Refinement, with `relaxation_factor` equal to a value other than `1`, the solver is a Richardson iteration, with possibility for additional preconditioning.

Assuming that `solver` has accuracy `c`, i.e., | `e - error` | `<= c` | `e` |, iterative refinement will converge with a convergence rate of `c`. Indeed, from `e - error = x - solution - error = x - solution*` (where `solution*` denotes the value stored in `solution` after the update) and `e = inv(A) residual = inv(A)b - inv(A) A solution = x - solution` it follows that | x - solution∗ | <= c | x - solution |.

Unless otherwise specified via the `solver` factory parameter, this implementation uses the identity operator (i.e. the solver that approximates the solution of a system Ax = b by setting x := b) as the default inner solver. Such a setting results in a relaxation method known as the Richardson iteration with parameter 1, which is guaranteed to converge for matrices whose spectrum is strictly contained within the unit disc around 1 (i.e., all its eigenvalues `lambda` have to satisfy the equation `|relaxation_factor ∗ lambda - 1| < 1).

Template Parameters

ValueType	precision of matrix elements

41.110.2 Member Function Documentation

41.110.2.1 apply_uses_initial_guess()

```
template<typename ValueType = default_precision>
bool gko::solver::Ir< ValueType >::apply_uses_initial_guess ( ) const [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess.

Returns

true as iterative solvers use the data in x as an initial guess.

41.110.2.2 conj_transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Ir< ValueType >::conj_transpose ( ) const [override],
[virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.110.2.3 get_solver()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Ir< ValueType >::get_solver ( ) const [inline]
```

Returns the solver operator used as the inner solver.

Returns

the solver operator used as the inner solver

41.110.2.4 get_stop_criterion_factory()

```
template<typename ValueType = default_precision>
std::shared_ptr<const stop::CriterionFactory> gko::solver::Ir< ValueType >::get_stop_criterion←
_factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.110.2.5 get_system_matrix()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::solver::Ir< ValueType >::get_system_matrix ( ) const [inline]
```

Returns the system operator (matrix) of the linear system.

Returns

the system operator (matrix)

41.110.2.6 set_solver()

```
template<typename ValueType = default_precision>
void gko::solver::Ir< ValueType >::set_solver (
            std::shared_ptr< const LinOp > new_solver ) [inline]
```

Sets the solver operator used as the inner solver.

Parameters

new_solver	the new inner solver

41.110.2.7 set_stop_criterion_factory()

```
template<typename ValueType = default_precision>
void gko::solver::Ir< ValueType >::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other )  [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

41.110.2.8 transpose()

```
template<typename ValueType = default_precision>
std::unique_ptr<LinOp> gko::solver::Ir< ValueType >::transpose ( ) const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/ir.hpp

41.111 gko::preconditioner::Isai< IsaiType, ValueType, IndexType > Class Template Reference

The Incomplete Sparse Approximate Inverse (ISAI) Preconditioner generates an approximate inverse matrix for a given square matrix A, lower triangular matrix L, upper triangular matrix U or symmetric positive (spd) matrix B.

```
#include <ginkgo/core/preconditioner/isai.hpp>
```

Public Member Functions

- std::shared_ptr< const typename std::conditional< IsaiType==isai_type::spd, Comp, Csr >::type > get_approximate_inverse () const

  Returns the approximate inverse of the given matrix (either a CSR matrix for IsaiType general, upper or lower or a composition of two CSR matrices for IsaiType spd).
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.111.1 Detailed Description

template<isai_type IsaiType, typename ValueType, typename IndexType>
class gko::preconditioner::Isai< IsaiType, ValueType, IndexType >

The Incomplete Sparse Approximate Inverse (ISAI) Preconditioner generates an approximate inverse matrix for a given square matrix A, lower triangular matrix L, upper triangular matrix U or symmetric positive (spd) matrix B.

Using the preconditioner computes $aiA * x$, $aiU * x$, $aiL * x$ or $aiC^T * aiC * x$ (depending on the type of the Isai) for a given vector x (may have multiple right hand sides). aiA, aiU and aiL are the approximate inverses for A, U and L respectively. aiC is an approximation to C, the exact Cholesky factor of B (This is commonly referred to as a Factorized Sparse Approximate Inverse, short FSPAI).

The sparsity pattern used for the approximate inverse of A, L and U is the same as the sparsity pattern of the respective matrix. For B, the sparsity pattern used for the approximate inverse is the same as the sparsity pattern of the lower triangular half of B.

Note that, except for the spd case, for a matrix A generally ISAI(A)^T != ISAI(A^T).

For more details on the algorithm, see the paper `Incomplete Sparse Approximate Inverses for Parallel Preconditioning`, which is the basis for this work.

Note

GPU implementations can only handle the vector unit width `width` (warp size for CUDA) as number of elements per row in the sparse matrix. If there are more than `width` elements per row, the remaining elements will be ignored.

Template Parameters

IsaiType	determines if the ISAI is generated for a general square matrix, a lower triangular matrix, an upper triangular matrix or an spd matrix
ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.111.2 Member Function Documentation

41.111.2.1 conj_transpose()

```
template<isai_type IsaiType, typename ValueType , typename IndexType >
std::unique_ptr<LinOp> gko::preconditioner::Isai< IsaiType, ValueType, IndexType >::conj_↩
transpose ( ) const [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.111.2.2 get_approximate_inverse()

```
template<isai_type IsaiType, typename ValueType , typename IndexType >
std::shared_ptr<const typename std::conditional<IsaiType == isai_type::spd, Comp, Csr>::type>
gko::preconditioner::Isai< IsaiType, ValueType, IndexType >::get_approximate_inverse ( ) const
[inline]
```

Returns the approximate inverse of the given matrix (either a CSR matrix for IsaiType general, upper or lower or a composition of two CSR matrices for IsaiType spd).

Returns

the generated approximate inverse

References gko::as().

41.111.2.3 transpose()

```
template<isai_type IsaiType, typename ValueType , typename IndexType >
std::unique_ptr<LinOp> gko::preconditioner::Isai< IsaiType, ValueType, IndexType >::transpose
( ) const [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/preconditioner/isai.hpp

41.112 gko::stop::Iteration Class Reference

The Iteration class is a stopping criterion which stops the iteration process after a preset number of iterations.

`#include <ginkgo/core/stop/iteration.hpp>`

41.112.1 Detailed Description

The Iteration class is a stopping criterion which stops the iteration process after a preset number of iterations.

Note

to use this stopping criterion, it is required to update the iteration count for the ::check() method.

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

- ginkgo/core/stop/iteration.hpp

41.113 gko::log::iteration_complete_data Struct Reference

Struct representing iteration complete related data.

`#include <ginkgo/core/log/record.hpp>`

41.113.1 Detailed Description

Struct representing iteration complete related data.

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

- ginkgo/core/log/record.hpp

41.114 gko::preconditioner::Jacobi< ValueType, IndexType > Class Template Reference

A block-Jacobi preconditioner is a block-diagonal linear operator, obtained by inverting the diagonal blocks of the source operator.

`#include <ginkgo/core/preconditioner/jacobi.hpp>`

Public Member Functions

- size_type get_num_blocks () const noexcept
  Returns the number of blocks of the operator.
- const block_interleaved_storage_scheme< index_type > & get_storage_scheme () const noexcept
  Returns the storage scheme used for storing Jacobi blocks.
- const value_type * get_blocks () const noexcept
  Returns the pointer to the memory used for storing the block data.
- const remove_complex< value_type > * get_conditioning () const noexcept
  Returns an array of 1-norm condition numbers of the blocks.
- size_type get_num_stored_elements () const noexcept
  Returns the number of elements explicitly stored in the matrix.
- void convert_to (matrix::Dense< value_type > *result) const override
  Converts the implementer to an object of type result_type.
- void move_to (matrix::Dense< value_type > *result) override
  Converts the implementer to an object of type result_type by moving data from this object.
- void write (mat_data &data) const override
  Writes a matrix to a matrix_data structure.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.114.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::preconditioner::Jacobi< ValueType, IndexType >

A block-Jacobi preconditioner is a block-diagonal linear operator, obtained by inverting the diagonal blocks of the source operator.

The Jacobi class implements the inversion of the diagonal blocks using Gauss-Jordan elimination with column pivoting, and stores the inverse explicitly in a customized format.

If the diagonal blocks of the matrix are not explicitly set by the user, the implementation will try to automatically detect the blocks by first finding the natural blocks of the matrix, and then applying the supervariable agglomeration procedure on them. However, if problem-specific knowledge regarding the block diagonal structure is available, it is usually beneficial to explicitly pass the starting rows of the diagonal blocks, as the block detection is merely a heuristic and cannot perfectly detect the diagonal block structure. The current implementation supports blocks of up to 32 rows / columns.

The implementation also includes an improved, adaptive version of the block-Jacobi preconditioner, which can store some of the blocks in lower precision and thus improve the performance of preconditioner application by reducing the amount of memory transfers. This variant can be enabled by setting the Jacobi::Factory's `storage_optimization` parameter. Refer to the documentation of the parameter for more details.

Template Parameters

ValueType	precision of matrix elements
IndexType	integral type used to store pointers to the start of each block

Note

The current implementation supports blocks of up to 32 rows / columns.

When using the adaptive variant, there may be a trade-off in terms of slightly longer preconditioner generation due to extra work required to detect the optimal precision of the blocks.

When the max_block_size is set to 1, specialized kernels are used, both for generation (inverting the diagonals) and application (diagonal scaling) to reduce the overhead involved in the usual (adaptive) block case.

41.114.2 Member Function Documentation

41.114.2.1 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::preconditioner::Jacobi< ValueType, IndexType >::conj_transpose (
) const  [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.114.2.2 convert_to()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::preconditioner::Jacobi< ValueType, IndexType >::convert_to (
            matrix::Dense< value_type > * result ) const  [override], [virtual]
```

Converts the implementer to an object of type result_type.

Parameters

result	the object used to store the result of the conversion

Implements gko::ConvertibleTo< matrix::Dense< ValueType > >.

41.114.2.3 get_blocks()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::preconditioner::Jacobi< ValueType, IndexType >::get_blocks ( ) const
[inline], [noexcept]
```

Returns the pointer to the memory used for storing the block data.

Element (`i, j`) of block `b` is stored in position `(get_block_pointers()[b] + i) * stride + j` of the array.

Returns

the pointer to the memory used for storing the block data

References gko::Array< ValueType >::get_const_data().

41.114.2.4 get_conditioning()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const remove_complex<value_type>* gko::preconditioner::Jacobi< ValueType, IndexType >::get_↩
conditioning ( ) const [inline], [noexcept]
```

Returns an array of 1-norm condition numbers of the blocks.

Returns

an array of 1-norm condition numbers of the blocks

Note

This value is valid only if adaptive precision variant is used, and implementations of the standard non-adaptive variant are allowed to omit the calculation of condition numbers.

References gko::Array< ValueType >::get_const_data().

41.114.2.5 get_num_blocks()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::preconditioner::Jacobi< ValueType, IndexType >::get_num_blocks ( ) const [inline],
[noexcept]
```

Returns the number of blocks of the operator.

Returns

the number of blocks of the operator

41.114.2.6 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::preconditioner::Jacobi< ValueType, IndexType >::get_num_stored_elements ( )
const [inline], [noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.114.2.7 get_storage_scheme()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const block_interleaved_storage_scheme<index_type>& gko::preconditioner::Jacobi< ValueType,
IndexType >::get_storage_scheme ( ) const [inline], [noexcept]
```

Returns the storage scheme used for storing Jacobi blocks.

Returns

the storage scheme used for storing Jacobi blocks

41.114.2.8 move_to()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::preconditioner::Jacobi< ValueType, IndexType >::move_to (
            matrix::Dense< value_type > * result ) [override], [virtual]
```

Converts the implementer to an object of type result_type by moving data from this object.

This method is used when the implementer is a temporary object, and move semantics can be used.

Parameters

result	the object used to emplace the result of the conversion

Note

ConvertibleTo::move_to can be implemented by simply calling ConvertibleTo::convert_to. However, this operation can often be optimized by exploiting the fact that implementer's data can be moved to the result.

Implements gko::ConvertibleTo< matrix::Dense< ValueType > >.

41.114.2.9 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::preconditioner::Jacobi< ValueType, IndexType >::transpose ( )
const  [override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.114.2.10 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::preconditioner::Jacobi< ValueType, IndexType >::write (
            mat_data & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/preconditioner/jacobi.hpp

41.115 gko::KernelNotFound Class Reference

KernelNotFound is thrown if Ginkgo cannot find a kernel which satisfies the criteria imposed by the input arguments.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

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

41.115.1 Detailed Description

KernelNotFound is thrown if Ginkgo cannot find a kernel which satisfies the criteria imposed by the input arguments.

41.115.2 Constructor & Destructor Documentation

41.115.2.1 KernelNotFound()

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

Initializes a KernelNotFound 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 function where the error occurred

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

- ginkgo/core/base/exception.hpp

41.116 gko::log::linop_data Struct Reference

Struct representing LinOp related data.

```
#include <ginkgo/core/log/record.hpp>
```

41.116.1 Detailed Description

Struct representing LinOp related data.

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

- ginkgo/core/log/record.hpp

41.117 gko::log::linop_factory_data Struct Reference

Struct representing LinOp factory related data.

```
#include <ginkgo/core/log/record.hpp>
```

41.117.1 Detailed Description

Struct representing LinOp factory related data.

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

- ginkgo/core/log/record.hpp

41.118 gko::LinOpFactory Class Reference

A LinOpFactory represents a higher order mapping which transforms one linear operator into another.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Additional Inherited Members

41.118.1 Detailed Description

A LinOpFactory represents a higher order mapping which transforms one linear operator into another.

In Ginkgo, every linear solver is viewed as a mapping. For example, given an s.p.d linear system $Ax = b$, the solution $x = A^{-1}b$ can be computed using the CG method. This algorithm can be represented in terms of linear operators and mappings between them as follows:

- A Cg::Factory is a higher order mapping which, given an input operator A, returns a new linear operator A^{-1} stored in "CG format"
- Storing the operator A^{-1} in "CG format" means that the data structure used to store the operator is just a simple pointer to the original matrix A. The application $x = A^{-1}b$ of such an operator can then be implemented by solving the linear system $Ax = b$ using the CG method. This is achieved in code by having a special class for each of those "formats" (e.g. the "Cg" class defines such a format for the CG solver).

Another example of a LinOpFactory is a preconditioner. A preconditioner for a linear operator A is a linear operator M^{-1}, which approximates A^{-1}. In addition, it is stored in a way such that both the data of M^{-1} is cheap to compute from A, and the operation $x = M^{-1}b$ can be computed quickly. These operators are useful to accelerate the convergence of Krylov solvers. Thus, a preconditioner also fits into the LinOpFactory framework:

- The factory maps a linear operator A into a preconditioner M^{-1} which is stored in suitable format (e.g. as a product of two factors in case of ILU preconditioners).
- The resulting linear operator implements the application operation $x = M^{-1}b$ depending on the format the preconditioner is stored in (e.g. as two triangular solves in case of ILU)

41.118.1.1 Example: using CG in Ginkgo

```
{c++}
// Suppose A is a matrix, b a rhs vector, and x an initial guess
// Create a CG which runs for at most 1000 iterations, and stops after
// reducing the residual norm by 6 orders of magnitude
auto cg_factory = solver::Cg<>::build()
    .with_max_iters(1000)
    .with_rel_residual_goal(1e-6)
    .on(cuda);
// create a linear operator which represents the solver
auto cg = cg_factory->generate(A);
// solve the system
cg->apply(gko::lend(b), gko::lend(x));
```

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

- ginkgo/core/base/lin_op.hpp

41.119 gko::matrix::Csr< ValueType, IndexType >::load_balance Class Reference

load_balance is a strategy_type which uses the load balance algorithm.

`#include <ginkgo/core/matrix/csr.hpp>`

Public Member Functions

- load_balance ()
  Creates a load_balance strategy.
- load_balance (std::shared_ptr< const CudaExecutor > exec)
  Creates a load_balance strategy with CUDA executor.
- load_balance (std::shared_ptr< const HipExecutor > exec)
  Creates a load_balance strategy with HIP executor.
- load_balance (std::shared_ptr< const DpcppExecutor > exec)
  Creates a load_balance strategy with DPCPP executor.
- load_balance (int64_t nwarps, int warp_size=32, bool cuda_strategy=true, std::string strategy_name="none")
  Creates a load_balance strategy with specified parameters.
- void process (const Array< index_type > &mtx_row_ptrs, Array< index_type > *mtx_srow) override
  Computes srow according to row pointers.
- int64_t clac_size (const int64_t nnz) override
  Computes the srow size according to the number of nonzeros.
- std::shared_ptr< strategy_type > copy () override
  Copy a strategy.

41.119.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >::load_balance

load_balance is a strategy_type which uses the load balance algorithm.

41.119.2 Constructor & Destructor Documentation

41.119.2.1 load_balance() [1/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Csr< ValueType, IndexType >::load_balance::load_balance ( )  [inline]
```

Creates a load_balance strategy.

Warning

this is deprecated! Please rely on the new automatic strategy instantiation or use one of the other constructors.

41.119.2.2 load_balance() [2/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Csr< ValueType, IndexType >::load_balance::load_balance (
            std::shared_ptr< const CudaExecutor > exec )  [inline]
```

Creates a load_balance strategy with CUDA executor.

Parameters

exec	the CUDA executor

41.119.2.3 load_balance() [3/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Csr< ValueType, IndexType >::load_balance::load_balance (
            std::shared_ptr< const HipExecutor > exec )  [inline]
```

Creates a load_balance strategy with HIP executor.

Parameters

exec	the HIP executor

41.119.2.4 load_balance() [4/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
```

```
gko::matrix::Csr< ValueType, IndexType >::load_balance::load_balance (
            std::shared_ptr< const DpcppExecutor > exec ) [inline]
```

Creates a load_balance strategy with DPCPP executor.

Parameters

exec	the DPCPP executor

Note

TODO: porting - we hardcode the subgroup size is 16 and the number of threads in a SIMD unit is 7

41.119.2.5 load_balance() [5/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Csr< ValueType, IndexType >::load_balance::load_balance (
            int64_t nwarps,
            int warp_size = 32,
            bool cuda_strategy = true,
            std::string strategy_name = "none" ) [inline]
```

Creates a load_balance strategy with specified parameters.

Parameters

nwarps	the number of warps in the executor
warp_size	the warp size of the executor
cuda_strategy	whether the `cuda_strategy` needs to be used.

Note

The warp_size must be the size of full warp. When using this constructor, set_strategy needs to be called with correct parameters which is replaced during the conversion.

41.119.3 Member Function Documentation

41.119.3.1 clac_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
int64_t gko::matrix::Csr< ValueType, IndexType >::load_balance::clac_size (
            const int64_t nnz ) [inline], [override], [virtual]
```

Computes the srow size according to the number of nonzeros.

Parameters

nnz	the number of nonzeros

Returns

the size of srow

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

References gko::ceildiv(), and gko::min().

41.119.3.2 copy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::load_balance::copy (
) [inline], [override], [virtual]
```

Copy a strategy.

This is a workaround until strategies are revamped, since strategies like `automatical` do not work when actually shared.

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.119.3.3 process()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::load_balance::process (
            const Array< index_type > & mtx_row_ptrs,
            Array< index_type > * mtx_srow ) [inline], [override], [virtual]
```

Computes srow according to row pointers.

Parameters

mtx_row_ptrs	the row pointers of the matrix
mtx_srow	the srow of the matrix

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

References gko::ceildiv(), gko::Array< ValueType >::get_const_data(), gko::Array< ValueType >::get_data(), gko::Array< ValueType >::get_executor(), and gko::Array< ValueType >::get_num_elems().

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

- ginkgo/core/matrix/csr.hpp

41.120 gko::log::Loggable Class Reference

Loggable class is an interface which should be implemented by classes wanting to support logging.

`#include <ginkgo/core/log/logger.hpp>`

Public Member Functions

- virtual void add_logger (std::shared_ptr< const Logger > logger)=0

  Adds a new logger to the list of subscribed loggers.
- virtual void remove_logger (const Logger ∗logger)=0

  Removes a logger from the list of subscribed loggers.
- virtual const std::vector< std::shared_ptr< const Logger > > & get_loggers () const =0

  Returns the vector containing all loggers registered at this object.
- virtual void clear_loggers ()=0

  Remove all loggers registered at this object.

41.120.1 Detailed Description

Loggable class is an interface which should be implemented by classes wanting to support logging.

For most cases, one can rely on the EnableLogging mixin which provides a default implementation of this interface.

41.120.2 Member Function Documentation

41.120.2.1 add_logger()

```
virtual void gko::log::Loggable::add_logger (
            std::shared_ptr< const Logger > logger )  [pure virtual]
```

Adds a new logger to the list of subscribed loggers.

Parameters

logger	the logger to add

41.120.2.2 get_loggers()

```
virtual const std::vector<std::shared_ptr<const Logger> >& gko::log::Loggable::get_loggers (
) const  [pure virtual]
```

Returns the vector containing all loggers registered at this object.

Returns

the vector containing all registered loggers.

41.120.2.3 remove_logger()

```
virtual void gko::log::Loggable::remove_logger (
            const Logger * logger ) [pure virtual]
```

Removes a logger from the list of subscribed loggers.

Parameters

logger	the logger to remove

Note

The comparison is done using the logger's object unique identity. Thus, two loggers constructed in the same way are not considered equal.

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

- ginkgo/core/log/logger.hpp

41.121 gko::log::Record::logged_data Struct Reference

Struct storing the actually logged data.

```
#include <ginkgo/core/log/record.hpp>
```

41.121.1 Detailed Description

Struct storing the actually logged data.

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

- ginkgo/core/log/record.hpp

41.122 gko::solver::LowerTrs< ValueType, IndexType > Class Template Reference

LowerTrs is the triangular solver which solves the system L x = b, when L is a lower triangular matrix.

```
#include <ginkgo/core/solver/lower_trs.hpp>
```

Public Member Functions

- std::shared_ptr< const matrix::Csr< ValueType, IndexType > > get_system_matrix () const
  Gets the system operator (CSR matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.122.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::solver::LowerTrs< ValueType, IndexType >

LowerTrs is the triangular solver which solves the system L x = b, when L is a lower triangular matrix.

It works best when passing in a matrix in CSR format. If the matrix is not in CSR, then the generate step converts it into a CSR matrix. The generation fails if the matrix is not convertible to CSR.

Note

As the constructor uses the copy and convert functionality, it is not possible to create a empty solver or a solver with a matrix in any other format other than CSR, if none of the executor modules are being compiled with.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indices

41.122.2 Member Function Documentation

41.122.2.1 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::solver::LowerTrs< ValueType, IndexType >::conj_transpose ( )
const [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.122.2.2 get_system_matrix()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<const matrix::Csr<ValueType, IndexType> > gko::solver::LowerTrs< ValueType,
IndexType >::get_system_matrix ( ) const [inline]
```

Gets the system operator (CSR matrix) of the linear system.

Returns

the system operator (CSR matrix)

```
{
    return system_matrix_;
}
```

41.122.2.3 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::solver::LowerTrs< ValueType, IndexType >::transpose ( ) const
[override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/lower_trs.hpp

41.123 gko::MachineTopology Class Reference

The machine topology class represents the hierarchical topology of a machine, including NUMA nodes, cores and PCI Devices.

```
#include <ginkgo/core/base/machine_topology.hpp>
```

Public Member Functions

- void bind_to_cores (const std::vector< int > &ids, const bool singlify=true) const
  Bind the calling process to the CPU cores associated with the ids.
- void bind_to_core (const int &id) const
  Bind to a single core.
- void bind_to_pus (const std::vector< int > &ids, const bool singlify=true) const
  Bind the calling process to PUs associated with the ids.
- void bind_to_pu (const int &id) const
  Bind to a Processing unit (PU)
- const normal_obj_info ∗ get_pu (size_type id) const
  Get the object of type PU associated with the id.
- const normal_obj_info ∗ get_core (size_type id) const
  Get the object of type core associated with the id.
- const io_obj_info ∗ get_pci_device (size_type id) const
  Get the object of type pci device associated with the id.
- const io_obj_info ∗ get_pci_device (const std::string &pci_bus_id) const
  Get the object of type pci device associated with the PCI bus id.
- size_type get_num_pus () const
  Get the number of PU objects stored in this Topology tree.
- size_type get_num_cores () const
  Get the number of core objects stored in this Topology tree.
- size_type get_num_pci_devices () const
  Get the number of PCI device objects stored in this Topology tree.
- size_type get_num_numas () const
  Get the number of NUMA objects stored in this Topology tree.

Static Public Member Functions

- static MachineTopology ∗ get_instance ()
  Returns an instance of the MachineTopology object.

41.123.1 Detailed Description

The machine topology class represents the hierarchical topology of a machine, including NUMA nodes, cores and PCI Devices.

Various infomation of the machine are gathered with the help of the Hardware Locality library (hwloc).

This class also provides functionalities to bind objects in the topology to the execution objects. Binding can enhance performance by allowing data to be closer to the executing object.

See the hwloc documentation (`https://www.open-mpi.org/projects/hwloc/doc/`) for more detailed information on topology detection and binding interfaces.

Note

A global object of MachineTopology type is created in a thread safe manner and only destroyed at the end of the program. This means that any subsequent queries will be from the same global object and hence use an extra atomic read.

41.123.2 Member Function Documentation

41.123.2.1 bind_to_core()

```
void gko::MachineTopology::bind_to_core (
            const int & id ) const  [inline]
```

Bind to a single core.

Parameters

ids	The ids of the core to be bound to the calling process.

```
{
    MachineTopology::get_instance()->bind_to_cores(std::vector<int>{id});
}
```

References bind_to_cores(), and get_instance().

41.123.2.2 bind_to_cores()

```
void gko::MachineTopology::bind_to_cores (
            const std::vector< int > & ids,
            const bool singlify = true ) const  [inline]
```

Bind the calling process to the CPU cores associated with the ids.

Parameters

ids	The ids of cores to be bound.
singlify	The ids of PUs are singlified to prevent possibly expensive migrations by the OS. This means that the binding is performed for only one of the ids in the set of ids passed in. See hwloc doc for `singlify`

Referenced by bind_to_core().

41.123.2.3 bind_to_pu()

```
void gko::MachineTopology::bind_to_pu (
            const int & id ) const  [inline]
```

Bind to a Processing unit (PU)

Parameters

ids	The ids of PUs to be bound to the calling process.

References bind_to_pus(), and get_instance().

41.123.2.4 bind_to_pus()

```
void gko::MachineTopology::bind_to_pus (
            const std::vector< int > & ids,
            const bool singlify = true ) const [inline]
```

Bind the calling process to PUs associated with the ids.

Parameters

ids	The ids of PUs to be bound.
singlify	The ids of PUs are singlified to prevent possibly expensive migrations by the OS. This means that the binding is performed for only one of the ids in the set of ids passed in. See hwloc doc for `singlify`

Referenced by bind_to_pu().

41.123.2.5 get_core()

```
const normal_obj_info* gko::MachineTopology::get_core (
            size_type id ) const [inline]
```

Get the object of type core associated with the id.

Parameters

id	The id of the core

Returns

the core object struct.

41.123.2.6 get_instance()

```
static MachineTopology* gko::MachineTopology::get_instance ( ) [inline], [static]
```

Returns an instance of the MachineTopology object.

Returns

the MachineTopology instance

Referenced by bind_to_core(), and bind_to_pu().

41.123.2.7 get_num_cores()

```
size_type gko::MachineTopology::get_num_cores ( ) const [inline]
```

Get the number of core objects stored in this Topology tree.

Returns

the number of cores.

41.123.2.8 get_num_numas()

```
size_type gko::MachineTopology::get_num_numas ( ) const [inline]
```

Get the number of NUMA objects stored in this Topology tree.

Returns

the number of NUMA objects.

41.123.2.9 get_num_pci_devices()

```
size_type gko::MachineTopology::get_num_pci_devices ( ) const [inline]
```

Get the number of PCI device objects stored in this Topology tree.

Returns

the number of PCI devices.

41.123.2.10 get_num_pus()

```
size_type gko::MachineTopology::get_num_pus ( ) const [inline]
```

Get the number of PU objects stored in this Topology tree.

Returns

the number of PUs.

41.123.2.11 get_pci_device() [1/2]

```
const io_obj_info* gko::MachineTopology::get_pci_device (
            const std::string & pci_bus_id ) const
```

Get the object of type pci device associated with the PCI bus id.

Parameters

pci_bus↩ _id	The PCI bus id of the pci device

Returns

the PCI object struct.

41.123.2.12 get_pci_device() [2/2]

```
const io_obj_info* gko::MachineTopology::get_pci_device (
            size_type id ) const  [inline]
```

Get the object of type pci device associated with the id.

Parameters

id	The id of the pci device

Returns

the PCI object struct.

41.123.2.13 get_pu()

```
const normal_obj_info* gko::MachineTopology::get_pu (
            size_type id ) const  [inline]
```

Get the object of type PU associated with the id.

Parameters

id	The id of the PU

Returns

the PU object struct.

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

- ginkgo/core/base/machine_topology.hpp

41.124 gko::matrix_assembly_data< ValueType, IndexType > Class Template Reference

This structure is used as an intermediate type to assemble a sparse matrix.

```
#include <ginkgo/core/base/matrix_assembly_data.hpp>
```

Public Member Functions

- void add_value (index_type row, index_type col, value_type val)

  Sets the matrix value at (row, col).
- void set_value (index_type row, index_type col, value_type val)

  Sets the matrix value at (row, col).
- value_type get_value (index_type row, index_type col)

  Gets the matrix value at (row, col).
- bool contains (index_type row, index_type col)

  Returns true iff the matrix contains an entry at (row, col).
- dim< 2 > get_size () const noexcept
- size_type get_num_stored_elements () const noexcept
- matrix_data< ValueType, IndexType > get_ordered_data () const

41.124.1 Detailed Description

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

This structure is used as an intermediate type to assemble a sparse matrix.

The matrix is stored as a set of nonzero elements, where each element is a triplet of the form (row_index, column↵_index, value).

New values can be added by using the matrix_assembly_data::add_value or matrix_assembly_data::set_value

Template Parameters

ValueType	type of matrix values stored in the structure
IndexType	type of matrix indexes stored in the structure

41.124.2 Member Function Documentation

41.124.2.1 add_value()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix_assembly_data< ValueType, IndexType >::add_value (
```

```
            index_type row,
            index_type col,
            value_type val )  [inline]
```

Sets the matrix value at (row, col).

If there is an existing value, it will be set to the sum of the existing and new value, otherwise the value will be inserted.

Parameters

row	the row where the value should be added
col	the column where the value should be added
val	the value to be added to (row, col)

```
{
    auto ind = std::make_pair(row, col);
    nonzeros_[ind] += val;
}
```

41.124.2.2 contains()

```
template<typename ValueType = default_precision, typename IndexType = int32>
bool gko::matrix_assembly_data< ValueType, IndexType >::contains (
            index_type row,
            index_type col )  [inline]
```

Returns true iff the matrix contains an entry at (row, col).

Parameters

row	the row index
col	the column index

Returns

true if the value at (row, col) exists, false otherwise

41.124.2.3 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix_assembly_data< ValueType, IndexType >::get_num_stored_elements ( ) const
[inline], [noexcept]
```

Returns

the number of non-zeros in the (partially) assembled matrix

41.124.2.4 get_ordered_data()

```
template<typename ValueType = default_precision, typename IndexType = int32>
matrix_data<ValueType, IndexType> gko::matrix_assembly_data< ValueType, IndexType >::get_←
ordered_data ( ) const  [inline]
```

Returns

a matrix_data instance containing the assembled non-zeros in row-major order to be used by all matrix formats.

Referenced by gko::ReadableFromMatrixData< ValueType, int32 >::read().

41.124.2.5 get_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
dim<2> gko::matrix_assembly_data< ValueType, IndexType >::get_size ( ) const  [inline], [noexcept]
```

Returns

the dimensions of the matrix being assembled

41.124.2.6 get_value()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type gko::matrix_assembly_data< ValueType, IndexType >::get_value (
             index_type row,
             index_type col )  [inline]
```

Gets the matrix value at (row, col).

Parameters

row	the row index
col	the column index

Returns

the value at (row, col) or 0 if it doesn't exist.

41.124.2.7 set_value()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix_assembly_data< ValueType, IndexType >::set_value (
```

```
index_type row,
index_type col,
value_type val )  [inline]
```

Sets the matrix value at (row, col).

If there is an existing value, it will be overwritten by the new value.

Parameters

row	the row index
col	the column index
val	the value to be written to (row, col)

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

- ginkgo/core/base/matrix_assembly_data.hpp

41.125 gko::matrix_data< ValueType, IndexType > Struct Template Reference

This structure is used as an intermediate data type to store a sparse matrix.

`#include <ginkgo/core/base/matrix_data.hpp>`

Public Member Functions

- matrix_data (dim< 2 > size_=dim< 2 >{}, ValueType value=zero< ValueType >())

  Initializes a matrix filled with the specified value.
- template<typename RandomDistribution , typename RandomEngine >
  matrix_data (dim< 2 > size_, RandomDistribution &&dist, RandomEngine &&engine)

  Initializes a matrix with random values from the specified distribution.
- matrix_data (std::initializer_list< std::initializer_list< ValueType >> values)

  List-initializes the structure from a matrix of values.
- matrix_data (dim< 2 > size_, std::initializer_list< detail::input_triple< ValueType, IndexType >> nonzeros←_)

  Initializes the structure from a list of nonzeros.
- matrix_data (dim< 2 > size_, const matrix_data &block)

  Initializes a matrix out of a matrix block via duplication.
- template<typename Accessor >
  matrix_data (const range< Accessor > &data)

  Initializes a matrix from a range.
- void ensure_row_major_order ()

  Sorts the nonzero vector so the values follow row-major order.
- void remove_zeros ()

  Remove entries with value zero from the matrix data.
- void sum_duplicates ()

  Sum up all values that refer to the same matrix entry.

Static Public Member Functions

- static matrix_data diag (dim< 2 > size_, ValueType value)

  Initializes a diagonal matrix.
- static matrix_data diag (dim< 2 > size_, std::initializer_list< ValueType > nonzeros_)

  Initializes a diagonal matrix using a list of diagonal elements.
- static matrix_data diag (dim< 2 > size_, const matrix_data &block)

  Initializes a block-diagonal matrix.
- template<typename ForwardIterator >
  static matrix_data diag (ForwardIterator begin, ForwardIterator end)

  Initializes a block-diagonal matrix from a list of diagonal blocks.
- static matrix_data diag (std::initializer_list< matrix_data > blocks)

  Initializes a block-diagonal matrix from a list of diagonal blocks.
- template<typename RandomDistribution , typename RandomEngine >
  static matrix_data cond (size_type size, remove_complex< ValueType > condition_number, RandomDistribution &&dist, RandomEngine &&engine, size_type num_reflectors)

  Initializes a random dense matrix with a specific condition number.
- template<typename RandomDistribution , typename RandomEngine >
  static matrix_data cond (size_type size, remove_complex< ValueType > condition_number, RandomDistribution &&dist, RandomEngine &&engine)

  Initializes a random dense matrix with a specific condition number.

Public Attributes

- dim< 2 > size

  Size of the matrix.
- std::vector< nonzero_type > nonzeros

  A vector of tuples storing the non-zeros of the matrix.

41.125.1 Detailed Description

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

This structure is used as an intermediate data type to store a sparse matrix.

The matrix is stored as a sequence of nonzero elements, where each element is a triple of the form (row_index, column_index, value).

Note

All Ginkgo functions returning such a structure will return the nonzeros sorted in row-major order.

All Ginkgo functions that take this structure as input expect that the nonzeros are sorted in row-major order and that the index pair (row_index, column_index) of each nonzero is unique.

This structure is not optimized for usual access patterns and it can only exist on the CPU. Thus, it should only be used for utility functions which do not have to be optimized for performance.

Template Parameters

ValueType	type of matrix values stored in the structure
IndexType	type of matrix indexes stored in the structure

41.125.2 Constructor & Destructor Documentation

41.125.2.1 matrix_data() [1/6]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix_data< ValueType, IndexType >::matrix_data (
            dim< 2 > size_ = dim<2>{},
            ValueType value = zero<ValueType>() )  [inline]
```

Initializes a matrix filled with the specified value.

Parameters

*size_*	dimensions of the matrix
value	value used to fill the elements of the matrix

```
                                        {}, ValueType value = zero<ValueType>())
        : size{size_}
    {
        if (is_zero(value)) {
            return;
        }
        for (size_type row = 0; row < size[0]; ++row) {
            for (size_type col = 0; col < size[1]; ++col) {
                nonzeros.emplace_back(row, col, value);
            }
        }
    }
```

41.125.2.2 matrix_data() [2/6]

```
template<typename ValueType = default_precision, typename IndexType = int32>
template<typename RandomDistribution , typename RandomEngine >
gko::matrix_data< ValueType, IndexType >::matrix_data (
            dim< 2 > size_,
            RandomDistribution && dist,
            RandomEngine && engine )  [inline]
```

Initializes a matrix with random values from the specified distribution.

Template Parameters

RandomDistribution	random distribution type
RandomEngine	random engine type

Parameters

*size_*	dimensions of the matrix
dist	random distribution of the elements of the matrix
engine	random engine used to generate random values

41.125.2.3 matrix_data() [3/6]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix_data< ValueType, IndexType >::matrix_data (
            std::initializer_list< std::initializer_list< ValueType >> values )  [inline]
```

List-initializes the structure from a matrix of values.

Parameters

values	a 2D braced-init-list of matrix values.

41.125.2.4 matrix_data() [4/6]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix_data< ValueType, IndexType >::matrix_data (
            dim< 2 > size_,
            std::initializer_list< detail::input_triple< ValueType, IndexType >> nonzeros_ )
[inline]
```

Initializes the structure from a list of nonzeros.

Parameters

*size_*	dimensions of the matrix
*nonzeros_*	list of nonzero elements

41.125.2.5 matrix_data() [5/6]

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix_data< ValueType, IndexType >::matrix_data (
            dim< 2 > size_,
            const matrix_data< ValueType, IndexType > & block )  [inline]
```

Initializes a matrix out of a matrix block via duplication.

Parameters

size	size of the block-matrix (in blocks)
diag_block	matrix block used to fill the complete matrix

References gko::matrix_data< ValueType, IndexType >::size.

41.125.2.6 matrix_data() [6/6]

```
template<typename ValueType = default_precision, typename IndexType = int32>
template<typename Accessor >
gko::matrix_data< ValueType, IndexType >::matrix_data (
            const range< Accessor > & data )  [inline]
```

Initializes a matrix from a range.

Template Parameters

Accessor	accessor type of the input range

Parameters

data	range used to initialize the matrix

References gko::range< Accessor >::length().

41.125.3 Member Function Documentation

41.125.3.1 cond() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
template<typename RandomDistribution , typename RandomEngine >
static matrix_data gko::matrix_data< ValueType, IndexType >::cond (
            size_type size,
            remove_complex< ValueType > condition_number,
            RandomDistribution && dist,
            RandomEngine && engine )  [inline], [static]
```

Initializes a random dense matrix with a specific condition number.

The matrix is generated by applying a series of random Hausholder reflectors to a diagonal matrix with diagonal entries uniformly distributed between `sqrt(condition_number)` and `1/sqrt(condition_number)`.

This version of the function applies `size - 1` reflectors to each side of the diagonal matrix.

Template Parameters

RandomDistribution	the type of the random distribution
RandomEngine	the type of the random engine

Parameters

size	number of rows and columns of the matrix
condition_number	condition number of the matrix
dist	random distribution used to generate reflectors
engine	random engine used to generate reflectors

Returns

the dense matrix with the specified condition number

References gko::matrix_data< ValueType, IndexType >::cond(), and gko::matrix_data< ValueType, IndexType >↩
::size.

41.125.3.2 cond() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
template<typename RandomDistribution , typename RandomEngine >
static matrix_data gko::matrix_data< ValueType, IndexType >::cond (
            size_type size,
            remove_complex< ValueType > condition_number,
            RandomDistribution && dist,
            RandomEngine && engine,
            size_type num_reflectors ) [inline], [static]
```

Initializes a random dense matrix with a specific condition number.

The matrix is generated by applying a series of random Hausholder reflectors to a diagonal matrix with diagonal entries uniformly distributed between `sqrt(condition_number)` and `1/sqrt(condition_number)`.

Template Parameters

RandomDistribution	the type of the random distribution
RandomEngine	the type of the random engine

Parameters

size	number of rows and columns of the matrix
condition_number	condition number of the matrix
dist	random distribution used to generate reflectors
engine	random engine used to generate reflectors
num_reflectors	number of reflectors to apply from each side

Returns

the dense matrix with the specified condition number

References gko::matrix_data< ValueType, IndexType >::size.

Referenced by gko::matrix_data< ValueType, IndexType >::cond().

41.125.3.3 diag() [1/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
static matrix_data gko::matrix_data< ValueType, IndexType >::diag (
            dim< 2 > size_,
            const matrix_data< ValueType, IndexType > & block )  [inline], [static]
```

Initializes a block-diagonal matrix.

Parameters

*size_*	the size of the matrix
diag_block	matrix used to fill diagonal blocks

Returns

the block-diagonal matrix

References gko::matrix_data< ValueType, IndexType >::nonzeros, and gko::matrix_data< ValueType, IndexType >::size.

41.125.3.4 diag() [2/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
static matrix_data gko::matrix_data< ValueType, IndexType >::diag (
            dim< 2 > size_,
            std::initializer_list< ValueType > nonzeros_ )  [inline], [static]
```

Initializes a diagonal matrix using a list of diagonal elements.

Parameters

*size_*	dimensions of the matrix
*nonzeros_*	list of diagonal elements

Returns

the diagonal matrix

References gko::matrix_data< ValueType, IndexType >::nonzeros.

41.125.3.5 diag() [3/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
static matrix_data gko::matrix_data< ValueType, IndexType >::diag (
            dim< 2 > size_,
            ValueType value )  [inline], [static]
```

Initializes a diagonal matrix.

Parameters

*size_*	dimensions of the matrix
value	value used to fill the elements of the matrix

Returns

the diagonal matrix

References gko::is_nonzero(), and gko::matrix_data< ValueType, IndexType >::nonzeros.

Referenced by gko::matrix_data< ValueType, IndexType >::diag().

41.125.3.6 diag() [4/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
template<typename ForwardIterator >
static matrix_data gko::matrix_data< ValueType, IndexType >::diag (
            ForwardIterator begin,
            ForwardIterator end )  [inline], [static]
```

Initializes a block-diagonal matrix from a list of diagonal blocks.

Template Parameters

ForwardIterator	type of list iterator

Parameters

begin	the first iterator of the list
end	the last iterator of the list

Returns

the block-diagonal matrix with diagonal blocks set to the blocks between begin (inclusive) and end (exclusive)

References gko::matrix_data< ValueType, IndexType >::nonzeros.

41.125.3.7 diag() [5/5]

```
template<typename ValueType = default_precision, typename IndexType = int32>
static matrix_data gko::matrix_data< ValueType, IndexType >::diag (
            std::initializer_list< matrix_data< ValueType, IndexType > > blocks )  [inline],
[static]
```

Initializes a block-diagonal matrix from a list of diagonal blocks.

Parameters

blocks	a list of blocks to initialize from

Returns

the block-diagonal matrix with diagonal blocks set to the blocks passed in blocks

References gko::matrix_data< ValueType, IndexType >::diag().

41.125.3.8 sum_duplicates()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix_data< ValueType, IndexType >::sum_duplicates ( )  [inline]
```

Sum up all values that refer to the same matrix entry.

The result is sorted in row-major order.

References gko::matrix_data< ValueType, IndexType >::ensure_row_major_order(), and gko::matrix_data< ValueType, IndexType >::nonzeros.

41.125.4 Member Data Documentation

41.125.4.1 nonzeros

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::vector<nonzero_type> gko::matrix_data< ValueType, IndexType >::nonzeros
```

A vector of tuples storing the non-zeros of the matrix.

The first two elements of the tuple are the row index and the column index of a matrix element, and its third element is the value at that position.

Referenced by gko::matrix_data< ValueType, IndexType >::diag(), gko::matrix_data< ValueType, IndexType >↩::ensure_row_major_order(), gko::matrix_data< ValueType, IndexType >::remove_zeros(), and gko::matrix_data< ValueType, IndexType >::sum_duplicates().

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

- ginkgo/core/base/matrix_data.hpp

41.126 gko::matrix_data_entry< ValueType, IndexType > Struct Template Reference

Type used to store nonzeros.

```
#include <ginkgo/core/base/matrix_data.hpp>
```

41.126.1 Detailed Description

template<typename ValueType, typename IndexType>
struct gko::matrix_data_entry< ValueType, IndexType >

Type used to store nonzeros.

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

- ginkgo/core/base/matrix_data.hpp

41.127 gko::matrix::Csr< ValueType, IndexType >::merge_path Class Reference

merge_path is a strategy_type which uses the merge_path algorithm.

```
#include <ginkgo/core/matrix/csr.hpp>
```

Public Member Functions

- merge_path ()
  Creates a merge_path strategy.
- void process (const Array< index_type > &mtx_row_ptrs, Array< index_type > *mtx_srow) override
  Computes srow according to row pointers.
- int64_t clac_size (const int64_t nnz) override
  Computes the srow size according to the number of nonzeros.
- std::shared_ptr< strategy_type > copy () override
  Copy a strategy.

41.127.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >::merge_path

merge_path is a strategy_type which uses the merge_path algorithm.

merge_path is according to Merrill and Garland: Merge-Based Parallel Sparse Matrix-Vector Multiplication

41.127.2 Member Function Documentation

41.127.2.1 clac_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
int64_t gko::matrix::Csr< ValueType, IndexType >::merge_path::clac_size (
            const int64_t nnz )  [inline], [override], [virtual]
```

Computes the srow size according to the number of nonzeros.

Parameters

nnz	the number of nonzeros

Returns

the size of srow

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.127.2.2 copy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::merge_path::copy ( )
[inline], [override], [virtual]
```

Copy a strategy.

This is a workaround until strategies are revamped, since strategies like `automatical` do not work when actually shared.

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.127.2.3 process()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::merge_path::process (
            const Array< index_type > & mtx_row_ptrs,
            Array< index_type > * mtx_srow ) [inline], [override], [virtual]
```

Computes srow according to row pointers.

Parameters

mtx_row_ptrs	the row pointers of the matrix
mtx_srow	the srow of the matrix

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

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

- ginkgo/core/matrix/csr.hpp

41.128 gko::matrix::Hybrid< ValueType, IndexType >::minimal_storage_limit Class Reference

minimal_storage_limit is a strategy_type which decides the number of stored elements per row of the ell part.

`#include <ginkgo/core/matrix/hybrid.hpp>`

Public Member Functions

- minimal_storage_limit ()
  Creates a minimal_storage_limit strategy.
- size_type compute_ell_num_stored_elements_per_row (Array< size_type > ∗row_nnz) const override
  Computes the number of stored elements per row of the ell part.
- auto get_percentage () const
  Get the percent setting.

41.128.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >::minimal_storage_limit

minimal_storage_limit is a strategy_type which decides the number of stored elements per row of the ell part.

It is determined by the size of ValueType and IndexType, the storage is the minimum among all partition.

41.128.2 Member Function Documentation

41.128.2.1 compute_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::minimal_storage_limit::compute_ell_←
num_stored_elements_per_row (
            Array< size_type > * row_nnz ) const [inline], [override], [virtual]
```

Computes the number of stored elements per row of the ell part.

Parameters

row_nnz	the number of nonzeros of each row

Returns

the number of stored elements per row of the ell part

Implements gko::matrix::Hybrid< ValueType, IndexType >::strategy_type.

References gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::compute_ell_num_stored_elements_per_row().

41.128.2.2 get_percentage()

```
template<typename ValueType = default_precision, typename IndexType = int32>
auto gko::matrix::Hybrid< ValueType, IndexType >::minimal_storage_limit::get_percentage ( )
const [inline]
```

Get the percent setting.

@retrun percent

References gko::matrix::Hybrid< ValueType, IndexType >::imbalance_limit::get_percentage().

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

- ginkgo/core/matrix/hybrid.hpp

41.129 gko::MpiError Class Reference

MpiError is thrown when a MPI routine throws a non-zero error code.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- MpiError (const std::string &file, int line, const std::string &func, int64 error_code)
  Initializes a MPI error.

41.129.1 Detailed Description

MpiError is thrown when a MPI routine throws a non-zero error code.

41.129.2 Constructor & Destructor Documentation

41.129.2.1 MpiError()

```
gko::MpiError::MpiError (
            const std::string & file,
            int line,
            const std::string & func,
            int64 error_code ) [inline]
```

Initializes a MPI 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 MPI routine that failed
error_code	The resulting MPI error code

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

- ginkgo/core/base/exception.hpp

41.130 gko::solver::Multigrid Class Reference

Multigrid methods have a hierarchy of many levels, whose corase level is a subset of the fine level, of the problem.

```
#include <ginkgo/core/solver/multigrid.hpp>
```

Public Member Functions

- bool apply_uses_initial_guess () const override

  Return true as iterative solvers use the data in x as an initial guess or false if multigrid always set the input as zero.
- std::shared_ptr< const stop::CriterionFactory > get_stop_criterion_factory () const

  Gets the stopping criterion factory of the solver.
- void set_stop_criterion_factory (std::shared_ptr< const stop::CriterionFactory > other)

  Sets the stopping criterion of the solver.
- std::shared_ptr< const LinOp > get_system_matrix () const

  Gets the system operator of the linear system.
- std::vector< std::shared_ptr< const gko::multigrid::MultigridLevel > > get_mg_level_list () const

  Gets the list of MultigridLevel operators.
- std::vector< std::shared_ptr< const LinOp > > get_pre_smoother_list () const

  Gets the list of pre-smoother operators.
- std::vector< std::shared_ptr< const LinOp > > get_mid_smoother_list () const

  Gets the list of mid-smoother operators.
- std::vector< std::shared_ptr< const LinOp > > get_post_smoother_list () const

  Gets the list of post-smoother operators.
- std::shared_ptr< const LinOp > get_coarsest_solver () const

  Gets the operator at the coarsest level.
- multigrid::cycle get_cycle () const

  Get the cycle of multigrid.
- void set_cycle (multigrid::cycle cycle)

  Set the cycle of multigrid.

41.130.1 Detailed Description

Multigrid methods have a hierarchy of many levels, whose corase level is a subset of the fine level, of the problem.

The coarse level solves the system on the residual of fine level and fine level will use the coarse solution to correct its own result. Multigrid solves the problem by relatively cheap step in each level and refining the result when prolongating back.

The main step of each level

- Presmooth (solve on the fine level)
- Calculate residual
- Restrict (reduce the problem dimension)
- Solve residual in next level
- Prolongate (return to the fine level size)
- Postsmooth (correct the answer in fine level)

Ginkgo uses the index from 0 for finest level (original problem size) $\sim$ N for the coarsest level (the coarsest solver), and its level counts is N (N multigrid level generation).

41.130.2 Member Function Documentation

41.130.2.1 apply_uses_initial_guess()

```
bool gko::solver::Multigrid::apply_uses_initial_guess ( ) const  [inline], [override]
```

Return true as iterative solvers use the data in x as an initial guess or false if multigrid always set the input as zero.

Returns

bool it is related to parameters variable zero_guess

```
{
    return !parameters_.zero_guess;
}
```

41.130.2.2 get_coarsest_solver()

```
std::shared_ptr<const LinOp> gko::solver::Multigrid::get_coarsest_solver ( ) const  [inline]
```

Gets the operator at the coarsest level.

Returns

the coarsest operator

41.130.2.3 get_cycle()

```
multigrid::cycle gko::solver::Multigrid::get_cycle ( ) const [inline]
```

Get the cycle of multigrid.

Returns

the multigrid::cycle

41.130.2.4 get_mg_level_list()

```
std::vector<std::shared_ptr<const gko::multigrid::MultigridLevel> > gko::solver::Multigrid←
::get_mg_level_list ( ) const [inline]
```

Gets the list of MultigridLevel operators.

Returns

the list of MultigridLevel operators

41.130.2.5 get_mid_smoother_list()

```
std::vector<std::shared_ptr<const LinOp> > gko::solver::Multigrid::get_mid_smoother_list ( )
const [inline]
```

Gets the list of mid-smoother operators.

Returns

the list of mid-smoother operators

41.130.2.6 get_post_smoother_list()

```
std::vector<std::shared_ptr<const LinOp> > gko::solver::Multigrid::get_post_smoother_list ( )
const [inline]
```

Gets the list of post-smoother operators.

Returns

the list of post-smoother operators

41.130.2.7 get_pre_smoother_list()

```
std::vector<std::shared_ptr<const LinOp> > gko::solver::Multigrid::get_pre_smoother_list ( )
const [inline]
```

Gets the list of pre-smoother operators.

Returns

the list of pre-smoother operators

41.130.2.8 get_stop_criterion_factory()

```
std::shared_ptr<const stop::CriterionFactory> gko::solver::Multigrid::get_stop_criterion_←
factory ( ) const [inline]
```

Gets the stopping criterion factory of the solver.

Returns

the stopping criterion factory

41.130.2.9 get_system_matrix()

```
std::shared_ptr<const LinOp> gko::solver::Multigrid::get_system_matrix ( ) const [inline]
```

Gets the system operator of the linear system.

Returns

the system operator

41.130.2.10 set_cycle()

```
void gko::solver::Multigrid::set_cycle (
            multigrid::cycle cycle ) [inline]
```

Set the cycle of multigrid.

Parameters

multigrid::cycle	the new cycle

41.130.2.11 set_stop_criterion_factory()

```
void gko::solver::Multigrid::set_stop_criterion_factory (
            std::shared_ptr< const stop::CriterionFactory > other )  [inline]
```

Sets the stopping criterion of the solver.

Parameters

other	the new stopping criterion factory

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

- ginkgo/core/solver/multigrid.hpp

41.131 gko::multigrid::MultigridLevel Class Reference

This class represents two levels in a multigrid hierarchy.

```
#include <ginkgo/core/multigrid/multigrid_level.hpp>
```

Public Member Functions

- virtual std::shared_ptr< const LinOp > get_fine_op () const =0

  Returns the operator on fine level.
- virtual std::shared_ptr< const LinOp > get_restrict_op () const =0

  Returns the restrict operator.
- virtual std::shared_ptr< const LinOp > get_coarse_op () const =0

  Returns the operator on coarse level.
- virtual std::shared_ptr< const LinOp > get_prolong_op () const =0

  Returns the prolong operator.

41.131.1 Detailed Description

This class represents two levels in a multigrid hierarchy.

The MultigridLevel is an interface that allows to get the individual components of multigrid level. Each implementation of a multigrid level should inherit from this interface. Use EnableMultigridLevel<ValueType> to implement this interface with composition by default.

41.131.2 Member Function Documentation

41.131.2.1 get_coarse_op()

```
virtual std::shared_ptr<const LinOp> gko::multigrid::MultigridLevel::get_coarse_op ( ) const
[pure virtual]
```

Returns the operator on coarse level.

Returns

the operator on coarse level.

Implemented in gko::multigrid::EnableMultigridLevel< ValueType >.

41.131.2.2 get_fine_op()

```
virtual std::shared_ptr<const LinOp> gko::multigrid::MultigridLevel::get_fine_op ( ) const
[pure virtual]
```

Returns the operator on fine level.

Returns

the operator on fine level.

Implemented in gko::multigrid::EnableMultigridLevel< ValueType >.

41.131.2.3 get_prolong_op()

```
virtual std::shared_ptr<const LinOp> gko::multigrid::MultigridLevel::get_prolong_op ( ) const
[pure virtual]
```

Returns the prolong operator.

Returns

the prolong operator.

Implemented in gko::multigrid::EnableMultigridLevel< ValueType >.

41.131.2.4 get_restrict_op()

```
virtual std::shared_ptr<const LinOp> gko::multigrid::MultigridLevel::get_restrict_op ( ) const
[pure virtual]
```

Returns the restrict operator.

Returns

the restrict operator.

Implemented in gko::multigrid::EnableMultigridLevel< ValueType >.

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

- ginkgo/core/multigrid/multigrid_level.hpp

41.132 gko::NotCompiled Class Reference

NotCompiled is thrown when attempting to call an operation which is a part of a module that was not compiled on the system.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- NotCompiled (const std::string &file, int line, const std::string &func, const std::string &module)
  Initializes a NotCompiled error.

41.132.1 Detailed Description

NotCompiled is thrown when attempting to call an operation which is a part of a module that was not compiled on the system.

41.132.2 Constructor & Destructor Documentation

41.132.2.1 NotCompiled()

```
gko::NotCompiled::NotCompiled (
            const std::string & file,
            int line,
            const std::string & func,
            const std::string & module )  [inline]
```

Initializes a NotCompiled 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 function that has not been compiled
module	The name of the module which contains the function

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

- ginkgo/core/base/exception.hpp

41.133 gko::NotImplemented Class Reference

NotImplemented is thrown in case an operation has not yet been implemented (but will be implemented in the future).

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

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

41.133.1 Detailed Description

NotImplemented is thrown in case an operation has not yet been implemented (but will be implemented in the future).

41.133.2 Constructor & Destructor Documentation

41.133.2.1 NotImplemented()

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

Initializes a NotImplemented 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 not-yet implemented function

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

- ginkgo/core/base/exception.hpp

41.134 gko::NotSupported Class Reference

NotSupported is thrown in case it is not possible to perform the requested operation on the given object type.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- NotSupported (const std::string &file, int line, const std::string &func, const std::string &obj_type)
  Initializes a NotSupported error.

41.134.1 Detailed Description

NotSupported is thrown in case it is not possible to perform the requested operation on the given object type.

41.134.2 Constructor & Destructor Documentation

41.134.2.1 NotSupported()

```
gko::NotSupported::NotSupported (
            const std::string & file,
            int line,
            const std::string & func,
            const std::string & obj_type )  [inline]
```

Initializes a NotSupported 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 function where the error occured
obj_type	The object type on which the requested operation cannot be performed.

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

- ginkgo/core/base/exception.hpp

41.135 gko::null_deleter< T > Class Template Reference

This is a deleter that does not delete the object.

```
#include <ginkgo/core/base/utils_helper.hpp>
```

Public Member Functions

- void operator() (pointer) const noexcept
  Deletes the object.

41.135.1 Detailed Description

template<typename T>
class gko::null_deleter< T >

This is a deleter that does not delete the object.

It is useful where the object has been allocated elsewhere and will be deleted manually.

41.135.2 Member Function Documentation

41.135.2.1 operator()()

```
template<typename T >
void gko::null_deleter< T >::operator() (
            pointer ) const [inline], [noexcept]
```

Deletes the object.

Parameters

ptr	pointer to the object being deleted

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

- ginkgo/core/base/utils_helper.hpp

41.136 gko::nvidia_device Class Reference

nvidia_device handles the number of executor on Nvidia devices and have the corresponding recursive_mutex.

```
#include <ginkgo/core/base/device.hpp>
```

41.136.1 Detailed Description

nvidia_device handles the number of executor on Nvidia devices and have the corresponding recursive_mutex.

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

- ginkgo/core/base/device.hpp

41.137 gko::OmpExecutor Class Reference

This is the Executor subclass which represents the OpenMP device (typically CPU).

`#include <ginkgo/core/base/executor.hpp>`

Public Member Functions

- std::shared_ptr< Executor > get_master () noexcept override
  Returns the master OmpExecutor of this Executor.
- std::shared_ptr< const Executor > get_master () const noexcept override
  Returns the master OmpExecutor of this Executor.
- void synchronize () const override
  Synchronize the operations launched on the executor with its master.

Static Public Member Functions

- static std::shared_ptr< OmpExecutor > create ()
  Creates a new OmpExecutor.

41.137.1 Detailed Description

This is the Executor subclass which represents the OpenMP device (typically CPU).

41.137.2 Member Function Documentation

41.137.2.1 get_master() [1/2]

```
std::shared_ptr<const Executor> gko::OmpExecutor::get_master ( ) const [override], [virtual],
[noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

41.137.2.2 get_master() [2/2]

```
std::shared_ptr<Executor> gko::OmpExecutor::get_master ( ) [override], [virtual], [noexcept]
```

Returns the master OmpExecutor of this Executor.

Returns

the master OmpExecutor of this Executor.

Implements gko::Executor.

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

- ginkgo/core/base/executor.hpp

41.138 gko::Operation Class Reference

Operations can be used to define functionalities whose implementations differ among devices.

`#include <ginkgo/core/base/executor.hpp>`

Public Member Functions

- virtual const char ∗ get_name () const noexcept

  Returns the operation's name.

41.138.1 Detailed Description

Operations can be used to define functionalities whose implementations differ among devices.

This is done by extending the Operation class and implementing the overloads of the Operation::run() method for all Executor types. When invoking the Executor::run() method with the Operation as input, the library will select the Operation::run() overload corresponding to the dynamic type of the Executor instance.

Consider an overload of `operator<<` for Executors, which prints some basic device information (e.g. device type and id) of the Executor to a C++ stream:

```
std::ostream& operator«(std::ostream &os, const gko::Executor &exec);
```

One possible implementation would be to use RTTI to find the dynamic type of the Executor, However, using the Operation feature of Ginkgo, there is a more elegant approach which utilizes polymorphism. The first step is to define an Operation that will print the desired information for each Executor type.

```
class DeviceInfoPrinter : public gko::Operation {
public:
    explicit DeviceInfoPrinter(std::ostream &os) : os_(os) {}
    void run(const gko::OmpExecutor *) const override { os_ « "OMP"; }
    void run(const gko::CudaExecutor *exec) const override
    { os_ « "CUDA(" « exec->get_device_id() « ")"; }
    void run(const gko::HipExecutor *exec) const override
    { os_ « "HIP(" « exec->get_device_id() « ")"; }
    void run(const gko::DpcppExecutor *exec) const override
    { os_ « "DPC++(" « exec->get_device_id() « ")"; }
    // This is optional, if not overloaded, defaults to OmpExecutor overload
    void run(const gko::ReferenceExecutor *) const override
    { os_ « "Reference CPU"; }
private:
```

```
    std::ostream &os_;
};
```

Using DeviceInfoPrinter, the implementation of `operator<<` is as simple as calling the run() method of the executor.

```
std::ostream& operator«(std::ostream &os, const gko::Executor &exec)
{
    DeviceInfoPrinter printer(os);
    exec.run(printer);
    return os;
}
```

Now it is possible to write the following code:

```
auto omp = gko::OmpExecutor::create();
std::cout « *omp « std::endl
          « *gko::CudaExecutor::create(0, omp) « std::endl
          « *gko::HipExecutor::create(0, omp) « std::endl
          « *gko::DpcppExecutor::create(0, omp) « std::endl
          « *gko::ReferenceExecutor::create() « std::endl;
```

which produces the expected output:

```
OMP
CUDA(0)
HIP(0)
DPC++(0)
Reference CPU
```

One might feel that this code is too complicated for such a simple task. Luckily, there is an overload of the Executor::run() method, which is designed to facilitate writing simple operations like this one. The method takes four closures as input: one which is run for OMP, one for CUDA executors, one for HIP executors, and the last one for DPC++ executors. Using this method, there is no need to implement an Operation subclass:

```
std::ostream& operator«(std::ostream &os, const gko::Executor &exec)
{
    exec.run(
        [&]() { os « "OMP"; },  // OMP closure
        [&]() { os « "CUDA("    // CUDA closure
                   « static_cast<gko::CudaExecutor&>(exec)
                       .get_device_id()
                   « ")"; },
        [&]() { os « "HIP("    // HIP closure
                   « static_cast<gko::HipExecutor&>(exec)
                       .get_device_id()
                   « ")"; });
        [&]() { os « "DPC++("    // DPC++ closure
                   « static_cast<gko::DpcppExecutor&>(exec)
                       .get_device_id()
                   « ")"; });
    return os;
}
```

Using this approach, however, it is impossible to distinguish between a OmpExecutor and ReferenceExecutor, as both of them call the OMP closure.

41.138.2 Member Function Documentation

41.138.2.1 get_name()

`virtual const char* gko::Operation::get_name ( ) const [virtual], [noexcept]`

Returns the operation's name.

Returns

the operation's name

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

- ginkgo/core/base/executor.hpp

41.139 gko::log::operation_data Struct Reference

Struct representing Operator related data.

```
#include <ginkgo/core/log/record.hpp>
```

41.139.1 Detailed Description

Struct representing Operator related data.

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

- ginkgo/core/log/record.hpp

41.140 gko::OutOfBoundsError Class Reference

OutOfBoundsError is thrown if a memory access is detected to be out-of-bounds.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- OutOfBoundsError (const std::string &file, int line, size_type index, size_type bound)
  Initializes an OutOfBoundsError.

41.140.1 Detailed Description

OutOfBoundsError is thrown if a memory access is detected to be out-of-bounds.

41.140.2 Constructor & Destructor Documentation

41.140.2.1 OutOfBoundsError()

```
gko::OutOfBoundsError::OutOfBoundsError (
            const std::string & file,
            int line,
            size_type index,
            size_type bound )  [inline]
```

Initializes an OutOfBoundsError.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
index	The position that was accessed
bound	The first out-of-bound index

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

- ginkgo/core/base/exception.hpp

41.141 gko::factorization::ParIc< ValueType, IndexType > Class Template Reference

ParIC is an incomplete Cholesky factorization which is computed in parallel.

```
#include <ginkgo/core/factorization/par_ic.hpp>
```

Additional Inherited Members

41.141.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::factorization::ParIc< ValueType, IndexType >

ParIC is an incomplete Cholesky factorization which is computed in parallel.

L is a lower triangular matrix, which approximates a given matrix A with $A \approx LL^H$. Here, $L + L^H$ has the same sparsity pattern as A, which is also called IC(0).

The ParIC algorithm generates the incomplete factors iteratively, using a fixed-point iteration of the form

$$F(L) = \begin{cases} \sqrt{a_{ii} - \sum_{k=1}^{i-1} |l_{ik}|^2}, & i == j \\ a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, & i < j \end{cases}$$

In general, the entries of L can be iterated in parallel and in asynchronous fashion, the algorithm asymptotically converges to the incomplete factors L and L^H fulfilling $\left(R = A - L \cdot L^H\right)|_{\mathcal{S}} = 0|_{\mathcal{S}}$ where $\mathcal{S}$ is the pre-defined sparsity pattern (in case of IC(0) the sparsity pattern of the system matrix A). The number of ParIC sweeps needed for convergence depends on the parallelism level: For sequential execution, a single sweep is sufficient, for fine-grained parallelism, the number of sweeps necessary to get a good approximation of the incomplete factors depends heavily on the problem. On the OpenMP executor, 3 sweeps usually give a decent approximation in our experiments, while GPU executors can take 10 or more iterations.

The ParIC algorithm in Ginkgo follows the design of E. Chow and A. Patel, Fine-grained Parallel Incomplete LU Factorization, SIAM Journal on Scientific Computing, 37, C169-C193 (2015).

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

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

- ginkgo/core/factorization/par_ic.hpp

41.142 gko::factorization::ParIct< ValueType, IndexType > Class Template Reference

ParICT is an incomplete threshold-based Cholesky factorization which is computed in parallel.

```
#include <ginkgo/core/factorization/par_ict.hpp>
```

Additional Inherited Members

41.142.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::factorization::ParIct< ValueType, IndexType >

ParICT is an incomplete threshold-based Cholesky factorization which is computed in parallel.

L is a lower triangular matrix which approximates a given symmetric positive definite matrix A with $A \approx LL^T$. Here, L has a sparsity pattern that is improved iteratively based on its element-wise magnitude. The initial sparsity pattern is chosen based on the lower triangle of A.

One iteration of the ParICT algorithm consists of the following steps:

1. Calculating the residual $R = A - LL^T$
2. Adding new non-zero locations from R to L. The new non-zero locations are initialized based on the corresponding residual value.
3. Executing a fixed-point iteration on L according to $F(L) = \begin{cases} \frac{1}{l_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}\right), & i \neq j \\ \sqrt{a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}}, & i = j \end{cases}$
4. Removing the smallest entries (by magnitude) from L
5. Executing a fixed-point iteration on the (now sparser) L

This ParICT algorithm thus improves the sparsity pattern and the approximation of L simultaneously.

The implementation follows the design of H. Anzt et al., ParILUT - A Parallel Threshold ILU for GPUs, 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 231–241.

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

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

- ginkgo/core/factorization/par_ict.hpp

41.143 gko::factorization::ParIlu< ValueType, IndexType > Class Template Reference

ParILU is an incomplete LU factorization which is computed in parallel.

`#include <ginkgo/core/factorization/par_ilu.hpp>`

Additional Inherited Members

41.143.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::factorization::ParIlu< ValueType, IndexType >

ParILU is an incomplete LU factorization which is computed in parallel.

L is a lower unitriangular, while U is an upper triangular matrix, which approximate a given matrix A with $A \approx LU$. Here, L and U have the same sparsity pattern as A, which is also called ILU(0).

The ParILU algorithm generates the incomplete factors iteratively, using a fixed-point iteration of the form

$$F(L,U) = \begin{cases} \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}\right), & i > j \\ a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, & i \leq j \end{cases}$$

In general, the entries of L and U can be iterated in parallel and in asynchronous fashion, the algorithm asymptotically converges to the incomplete factors L and U fulfilling $(R = A - L \cdot U)\vert_{\mathcal{S}} = 0\vert_{\mathcal{S}}$ where $\mathcal{S}$ is the pre-defined sparsity pattern (in case of ILU(0) the sparsity pattern of the system matrix A). The number of ParILU sweeps needed for convergence depends on the parallelism level: For sequential execution, a single sweep is sufficient, for fine-grained parallelism, the number of sweeps necessary to get a good approximation of the incomplete factors depends heavily on the problem. On the OpenMP executor, 3 sweeps usually give a decent approximation in our experiments, while GPU executors can take 10 or more iterations.

The ParILU algorithm in Ginkgo follows the design of E. Chow and A. Patel, Fine-grained Parallel Incomplete LU Factorization, SIAM Journal on Scientific Computing, 37, C169-C193 (2015).

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

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

- ginkgo/core/factorization/par_ilu.hpp

41.144 gko::factorization::ParIlut< ValueType, IndexType > Class Template Reference

ParILUT is an incomplete threshold-based LU factorization which is computed in parallel.

```
#include <ginkgo/core/factorization/par_ilut.hpp>
```

Additional Inherited Members

41.144.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::factorization::ParIlut< ValueType, IndexType >

ParILUT is an incomplete threshold-based LU factorization which is computed in parallel.

L is a lower unitriangular, while U is an upper triangular matrix, which approximate a given matrix A with $A \approx LU$. Here, L and U have a sparsity pattern that is improved iteratively based on their element-wise magnitude. The initial sparsity pattern is chosen based on the $ILU(0)$ factorization of A.

One iteration of the ParILUT algorithm consists of the following steps:

1. Calculating the residual $R = A - LU$
2. Adding new non-zero locations from R to L and U. The new non-zero locations are initialized based on the corresponding residual value.
3. Executing a fixed-point iteration on L and U according to $F(L,U) = \begin{cases} \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right), & i > j \\ a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, & i \leq j \end{cases}$

   For a more detailed description of the fixed-point iteration, see ParIlu.
4. Removing the smallest entries (by magnitude) from L and U
5. Executing a fixed-point iteration on the (now sparser) L and U

This ParILUT algorithm thus improves the sparsity pattern and the approximation of L and U simultaneously.

The implementation follows the design of H. Anzt et al., ParILUT - A Parallel Threshold ILU for GPUs, 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 231–241.

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

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

- ginkgo/core/factorization/par_ilut.hpp

41.145 gko::Permutable< IndexType > Class Template Reference

Linear operators which support permutation should implement the Permutable interface.

`#include <ginkgo/core/base/lin_op.hpp>`

Public Member Functions

- virtual std::unique_ptr< LinOp > permute (const Array< IndexType > *permutation_indices) const

    Returns a LinOp representing the symmetric row and column permutation of the Permutable object.
- virtual std::unique_ptr< LinOp > inverse_permute (const Array< IndexType > *permutation_indices) const

    Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.
- virtual std::unique_ptr< LinOp > row_permute (const Array< IndexType > *permutation_indices) const =0

    Returns a LinOp representing the row permutation of the Permutable object.
- virtual std::unique_ptr< LinOp > column_permute (const Array< IndexType > *permutation_indices) const =0

    Returns a LinOp representing the column permutation of the Permutable object.
- virtual std::unique_ptr< LinOp > inverse_row_permute (const Array< IndexType > *permutation_indices) const =0

    Returns a LinOp representing the row permutation of the inverse permuted object.
- virtual std::unique_ptr< LinOp > inverse_column_permute (const Array< IndexType > *permutation_↩ indices) const =0

    Returns a LinOp representing the row permutation of the inverse permuted object.

41.145.1 Detailed Description

template<typename IndexType>
class gko::Permutable< IndexType >

Linear operators which support permutation should implement the Permutable interface.

It provides functions to permute the rows and columns of a LinOp, independently or symmetrically, and with a regular or inverted permutation.

After a regular row permutation with permutation array `perm` the row `i` in the output LinOp contains the row `perm[i]` from the input LinOp. After an inverse row permutation, the row `perm[i]` in the output LinOp contains the row `i` from the input LinOp. Equivalently, after a column permutation, the output stores in column `i` the column `perm[i]` from the input, and an inverse column permutation stores in column `perm[i]` the column `i` from the input. A symmetric permutation is functionally equivalent to calling `as<Permutable>(A->row_↩ permute(perm))->column_permute(perm)`, but the implementation can provide better performance due to kernel fusion.

41.145.1.1 Example: Permuting a Csr matrix:

```
{c++}
//Permuting an object of LinOp type.
//The object you want to permute.
auto op = matrix::Csr::create(exec);
//Permute the object by first converting it to a Permutable type.
auto perm = op->row_permute(permutation_indices);
```

41.145.2 Member Function Documentation

41.145.2.1 column_permute()

```
template<typename IndexType>
virtual std::unique_ptr<LinOp> gko::Permutable< IndexType >::column_permute (
            const Array< IndexType > * permutation_indices ) const [pure virtual]
```

Returns a LinOp representing the column permutation of the Permutable object.

In the resulting LinOp, the column `i` contains the input column `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new column permuted object

Implemented in gko::matrix::Dense< ValueType >, gko::matrix::Dense< ValueType >, and gko::matrix::Csr< ValueType, IndexType >

41.145.2.2 inverse_column_permute()

```
template<typename IndexType>
virtual std::unique_ptr<LinOp> gko::Permutable< IndexType >::inverse_column_permute (
            const Array< IndexType > * permutation_indices ) const [pure virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the column `perm[i]` contains the input column `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implemented in gko::matrix::Dense< ValueType >, gko::matrix::Dense< ValueType >, and gko::matrix::Csr< ValueType, IndexType

41.145.2.3 inverse_permute()

```
template<typename IndexType>
virtual std::unique_ptr<LinOp> gko::Permutable< IndexType >::inverse_permute (
            const Array< IndexType > * permutation_indices ) const  [inline], [virtual]
```

Returns a LinOp representing the symmetric inverse row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(perm[i],perm[j])` contains the input value `(i,j)`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented in gko::matrix::Dense< ValueType >, gko::matrix::Dense< ValueType >, and gko::matrix::Csr< ValueType, IndexTyp

41.145.2.4 inverse_row_permute()

```
template<typename IndexType>
virtual std::unique_ptr<LinOp> gko::Permutable< IndexType >::inverse_row_permute (
            const Array< IndexType > * permutation_indices ) const  [pure virtual]
```

Returns a LinOp representing the row permutation of the inverse permuted object.

In the resulting LinOp, the row `perm[i]` contains the input row `i`.

Parameters

permutation_indices	the array of indices containing the permutation order `perm`.

Returns

a pointer to the new inverse permuted object

Implemented in gko::matrix::Dense< ValueType >, gko::matrix::Dense< ValueType >, and gko::matrix::Csr< ValueType, IndexType

41.145.2.5 permute()

```
template<typename IndexType>
virtual std::unique_ptr<LinOp> gko::Permutable< IndexType >::permute (
            const Array< IndexType > * permutation_indices ) const  [inline], [virtual]
```

Returns a LinOp representing the symmetric row and column permutation of the Permutable object.

In the resulting LinOp, the entry at location `(i,j)` contains the input value `(perm[i],perm[j])`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Reimplemented in gko::matrix::Dense< ValueType >, gko::matrix::Dense< ValueType >, and gko::matrix::Csr< ValueType, IndexTyp

41.145.2.6 row_permute()

```
template<typename IndexType>
virtual std::unique_ptr<LinOp> gko::Permutable< IndexType >::row_permute (
            const Array< IndexType > * permutation_indices ) const  [pure virtual]
```

Returns a LinOp representing the row permutation of the Permutable object.

In the resulting LinOp, the row `i` contains the input row `perm[i]`.

Parameters

permutation_indices	the array of indices containing the permutation order.

Returns

a pointer to the new permuted object

Implemented in gko::matrix::Dense< ValueType >, gko::matrix::Dense< ValueType >, and gko::matrix::Csr< ValueType, IndexType

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

- ginkgo/core/base/lin_op.hpp

41.146 gko::matrix::Permutation< IndexType > Class Template Reference

Permutation is a matrix "format" which stores the row and column permutation arrays which can be used for re-ordering the rows and columns a matrix.

`#include <ginkgo/core/matrix/permutation.hpp>`

Public Member Functions

- index_type ∗ get_permutation () noexcept

  Returns a pointer to the array of permutation.
- const index_type ∗ get_const_permutation () const noexcept

  Returns a pointer to the array of permutation.
- size_type get_permutation_size () const noexcept

  Returns the number of elements explicitly stored in the permutation array.
- mask_type get_permute_mask () const

  Get the permute masks.
- void set_permute_mask (mask_type permute_mask)

  Set the permute masks.

Static Public Member Functions

- static std::unique_ptr< const Permutation > create_const (std::shared_ptr< const Executor > exec, size_type size, gko::detail::ConstArrayView< IndexType > &&perm_idxs, mask_type enabled_↩ permute=row_permute)

  Creates a constant (immutable) Permutation matrix from a constant array.

41.146.1 Detailed Description

template< typename IndexType = int32>
class gko::matrix::Permutation< IndexType >

Permutation is a matrix "format" which stores the row and column permutation arrays which can be used for re-ordering the rows and columns a matrix.

Template Parameters

IndexType	precision of permutation array indices.

Note

This format is used mainly to allow for an abstraction of the permutation/re-ordering and provides the user with an apply method which calls the respective LinOp's permute operation if the respective LinOp implements the Permutable interface. As such it only stores an array of the permutation indices.

41.146.2 Member Function Documentation

41.146.2.1 create_const()

```
template<typename IndexType = int32>
static std::unique_ptr<const Permutation> gko::matrix::Permutation< IndexType >::create_const
(
```

```
std::shared_ptr< const Executor > exec,
size_type size,
gko::detail::ConstArrayView< IndexType > && perm_idxs,
mask_type enabled_permute = row_permute )  [inline], [static]
```

Creates a constant (immutable) Permutation matrix from a constant array.

Parameters

exec	the executor to create the matrix on
size	the size of the square matrix
perm_idxs	the permutation index array of the matrix
enabled_permute	the mask describing the type of permutation

Returns

A smart pointer to the constant matrix wrapping the input array (if it resides on the same executor as the matrix) or a copy of the array on the correct executor.

```
    {
        // cast const-ness away, but return a const object afterwards,
        // so we can ensure that no modifications take place.
        return std::unique_ptr<const Permutation>(new Permutation{
            exec, size, gko::detail::array_const_cast(std::move(perm_idxs)),
            enabled_permute});
    }
```

41.146.2.2 get_const_permutation()

```
template<typename IndexType = int32>
const index_type* gko::matrix::Permutation< IndexType >::get_const_permutation ( ) const  [inline],
[noexcept]
```

Returns a pointer to the array of permutation.

Returns

the pointer to the row permutation array.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.146.2.3 get_permutation()

```
template<typename IndexType = int32>
index_type* gko::matrix::Permutation< IndexType >::get_permutation ( )  [inline], [noexcept]
```

Returns a pointer to the array of permutation.

Returns

the pointer to the row permutation array.

References gko::Array< ValueType >::get_data().

41.146.2.4 get_permutation_size()

```
template<typename IndexType = int32>
size_type gko::matrix::Permutation< IndexType >::get_permutation_size ( ) const  [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the permutation array.

Returns

the number of elements explicitly stored in the permutation array.

References gko::Array< ValueType >::get_num_elems().

41.146.2.5 get_permute_mask()

```
template<typename IndexType = int32>
mask_type gko::matrix::Permutation< IndexType >::get_permute_mask ( ) const  [inline]
```

Get the permute masks.

Returns

permute_mask the permute masks

41.146.2.6 set_permute_mask()

```
template<typename IndexType = int32>
void gko::matrix::Permutation< IndexType >::set_permute_mask (
             mask_type permute_mask )  [inline]
```

Set the permute masks.

Parameters

permute_mask	the permute masks

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

- ginkgo/core/matrix/permutation.hpp

41.147 gko::Perturbation< ValueType > Class Template Reference

The Perturbation class can be used to construct a LinOp to represent the operation `(identity + scalar * basis * projector)`.

`#include <ginkgo/core/base/perturbation.hpp>`

Public Member Functions

- const std::shared_ptr< const LinOp > get_basis () const noexcept
  Returns the basis of the perturbation.
- const std::shared_ptr< const LinOp > get_projector () const noexcept
  Returns the projector of the perturbation.
- const std::shared_ptr< const LinOp > get_scalar () const noexcept
  Returns the scalar of the perturbation.

41.147.1 Detailed Description

template<typename ValueType = default_precision>
class gko::Perturbation< ValueType >

The Perturbation class can be used to construct a LinOp to represent the operation `(identity + scalar * basis * projector)`.

This operator adds a movement along a direction constructed by `basis` and `projector` on the LinOp. `projector` gives the coefficient of `basis` to decide the direction.

For example, the Householder matrix can be represented with the Perturbation operator as follows. If u is the Householder factor then we can generate the `Householder transformation`, H = (I - 2 u u*). In this case, the parameters of Perturbation class are scalar = -2, basis = u, and projector = u*.

Template Parameters

ValueType	precision of input and result vectors

Note

the apply operations of Perturbation class are not thread safe

41.147.2 Member Function Documentation

41.147.2.1 get_basis()

```
template<typename ValueType = default_precision>
const std::shared_ptr<const LinOp> gko::Perturbation< ValueType >::get_basis ( ) const [inline],
[noexcept]
```

Returns the basis of the perturbation.

Returns

the basis of the perturbation

```
{
    return basis_;
}
```

41.147.2.2 get_projector()

```
template<typename ValueType = default_precision>
const std::shared_ptr<const LinOp> gko::Perturbation< ValueType >::get_projector ( ) const
[inline], [noexcept]
```

Returns the projector of the perturbation.

Returns

the projector of the perturbation

41.147.2.3 get_scalar()

```
template<typename ValueType = default_precision>
const std::shared_ptr<const LinOp> gko::Perturbation< ValueType >::get_scalar ( ) const [inline],
[noexcept]
```

Returns the scalar of the perturbation.

Returns

the scalar of the perturbation

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

- ginkgo/core/base/perturbation.hpp

41.148 gko::log::polymorphic_object_data Struct Reference

Struct representing PolymorphicObject related data.

`#include <ginkgo/core/log/record.hpp>`

41.148.1 Detailed Description

Struct representing PolymorphicObject related data.

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

- ginkgo/core/log/record.hpp

41.149 gko::PolymorphicObject Class Reference

A PolymorphicObject is the abstract base for all "heavy" objects in Ginkgo that behave polymorphically.

`#include <ginkgo/core/base/polymorphic_object.hpp>`

Public Member Functions

- std::unique_ptr< PolymorphicObject > create_default (std::shared_ptr< const Executor > exec) const

  Creates a new "default" object of the same dynamic type as this object.
- std::unique_ptr< PolymorphicObject > create_default () const

  Creates a new "default" object of the same dynamic type as this object.
- std::unique_ptr< PolymorphicObject > clone (std::shared_ptr< const Executor > exec) const

  Creates a clone of the object.
- std::unique_ptr< PolymorphicObject > clone () const

  Creates a clone of the object.
- PolymorphicObject * copy_from (const PolymorphicObject *other)

  Copies another object into this object.
- PolymorphicObject * copy_from (std::unique_ptr< PolymorphicObject > other)

  Moves another object into this object.
- PolymorphicObject * clear ()

  Transforms the object into its default state.
- std::shared_ptr< const Executor > get_executor () const noexcept

  Returns the Executor of the object.

41.149.1 Detailed Description

A PolymorphicObject is the abstract base for all "heavy" objects in Ginkgo that behave polymorphically.

It defines the basic utilities (copying moving, cloning, clearing the objects) for all such objects. It takes into account that these objects are dynamically allocated, managed by smart pointers, and used polymorphically. Additionally, it assumes their data can be allocated on different executors, and that they can be copied between those executors.

Note

Most of the public methods of this class should not be overridden directly, and are thus not virtual. Instead, there are equivalent protected methods (ending in <method_name>_impl) that should be overriden instead. This allows polymorphic objects to implement default behavior around virtual methods (parameter checking, type casting).

See also

EnablePolymorphicObject if you wish to implement a concrete polymorphic object and have sensible defaults generated automatically. EnableAbstractPolymorphicObject if you wish to implement a new abstract polymorphic object, and have the return types of the methods updated to your type (instead of having them return PolymorphicObject).

41.149.2 Member Function Documentation

41.149.2.1 clear()

`PolymorphicObject* gko::PolymorphicObject::clear ( ) [inline]`

Transforms the object into its default state.

Equivalent to `this->copy_from(this->create_default()).`

See also

clear_impl() when implementing this method

Returns

this

41.149.2.2 clone() [1/2]

```
std::unique_ptr<PolymorphicObject> gko::PolymorphicObject::clone ( ) const [inline]
```

Creates a clone of the object.

This is a shorthand for clone(std::shared_ptr<const Executor>) that uses the executor of this object to construct the new object.

Returns

A clone of the LinOp.

41.149.2.3 clone() [2/2]

```
std::unique_ptr<PolymorphicObject> gko::PolymorphicObject::clone (
            std::shared_ptr< const Executor > exec ) const [inline]
```

Creates a clone of the object.

This is the polymorphic equivalent of the *executor copy constructor* `decltype(*this)(exec, this)`.

Parameters

exec	the executor where the clone will be created

Returns

A clone of the LinOp.

References create_default().

41.149.2.4 copy_from() [1/2]

```
PolymorphicObject* gko::PolymorphicObject::copy_from (
            const PolymorphicObject * other ) [inline]
```

Copies another object into this object.

This is the polymorphic equivalent of the copy assignment operator.

See also

copy_from_impl(const PolymorphicObject ∗)

Parameters

other	the object to copy

Returns

this

41.149.2.5 copy_from() [2/2]

```
PolymorphicObject* gko::PolymorphicObject::copy_from (
            std::unique_ptr< PolymorphicObject > other ) [inline]
```

Moves another object into this object.

This is the polymorphic equivalent of the move assignment operator.

See also

copy_from_impl(std::unique_ptr<PolymorphicObject>)

Parameters

other	the object to move from

Returns

this

41.149.2.6 create_default() [1/2]

```
std::unique_ptr<PolymorphicObject> gko::PolymorphicObject::create_default ( ) const [inline]
```

Creates a new "default" object of the same dynamic type as this object.

This is a shorthand for create_default(std::shared_ptr<const Executor>) that uses the executor of this object to construct the new object.

Returns

a polymorphic object of the same type as this

Referenced by clone().

41.149.2.7 create_default() [2/2]

```
std::unique_ptr<PolymorphicObject> gko::PolymorphicObject::create_default (
            std::shared_ptr< const Executor > exec ) const [inline]
```

Creates a new "default" object of the same dynamic type as this object.

This is the polymorphic equivalent of the *executor default constructor* `decltype(*this)(exec);`.

Parameters

exec	the executor where the object will be created

Returns

a polymorphic object of the same type as this

41.149.2.8 get_executor()

```
std::shared_ptr<const Executor> gko::PolymorphicObject::get_executor ( ) const [inline],
[noexcept]
```

Returns the Executor of the object.

Returns

Executor of the object

Referenced by gko::matrix::BatchDense< ValueType >::add_scaled(), gko::matrix::Dense< ValueType >::add_scaled(), gko::matrix::Coo< ValueType, IndexType >::apply2(), gko::matrix::Dense< ValueType >::compute_conj_dot(), gko::matrix::BatchDense< ValueType >::compute_dot(), gko::matrix::Dense< ValueType >::compute_dot(), gko::matrix::Dense< ValueType >::compute_norm1(), gko::matrix::BatchDense< ValueType >::compute_norm2(), gko::matrix::Dense< ValueType >::compute_norm2(), gko::preconditioner::Ic< LSolverType, IndexType >::conj_transpose(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::conj_transpose(), gko::matrix::Dense< ValueType >::create_real_view(), gko::matrix::Dense< ValueType >::inv_scale(), gko::matrix::Csr< ValueType, IndexType >::inv_scale(), gko::matrix::BatchDense< ValueType >::scale(), gko::matrix::Dense< ValueType >::scale(), gko::matrix::Csr< ValueType, IndexType >::scale(), gko::matrix::Dense< ValueType >::sub_scaled(), gko::preconditioner::Ic< LSolverType, IndexType >::transpose(), gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >::transpose(), gko::matrix::BatchCsr< ValueType, IndexType >::unbatch(), gko::matrix::BatchEll< ValueType, IndexType >::unbatch(), gko::matrix::BatchDiagonal< ValueType >::unbatch(), and gko::matrix::BatchDense< ValueType >::unbatch().

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

- ginkgo/core/base/polymorphic_object.hpp

41.150 gko::precision_reduction Class Reference

This class is used to encode storage precisions of low precision algorithms.

```
#include <ginkgo/core/base/types.hpp>
```

Public Types

- using storage_type = uint8

  The underlying datatype used to store the encoding.

Public Member Functions

- constexpr precision_reduction () noexcept

  Creates a default precision_reduction encoding.
- constexpr precision_reduction (storage_type preserving, storage_type nonpreserving) noexcept

  Creates a precision_reduction encoding with the specified number of conversions.
- constexpr operator storage_type () const noexcept

  Extracts the raw data of the encoding.
- constexpr storage_type get_preserving () const noexcept

  Returns the number of preserving conversions in the encoding.
- constexpr storage_type get_nonpreserving () const noexcept

  Returns the number of non-preserving conversions in the encoding.

Static Public Member Functions

- constexpr static precision_reduction autodetect () noexcept

  Returns a special encoding which instructs the algorithm to automatically detect the best precision.
- constexpr static precision_reduction common (precision_reduction x, precision_reduction y) noexcept

  Returns the common encoding of input encodings.

41.150.1 Detailed Description

This class is used to encode storage precisions of low precision algorithms.

Some algorithms in Ginkgo can improve their performance by storing parts of the data in lower precision, while doing computation in full precision. This class is used to encode the precisions used to store the data. From the user's perspective, some algorithms can provide a parameter for fine-tuning the storage precision. Commonly, the special value returned by precision_reduction::autodetect() should be used to allow the algorithm to automatically choose an appropriate value, though manually selected values can be used for fine-tuning.

In general, a lower precision floating point value can be obtained by either dropping some of the insignificant bits of the significand (keeping the same number of exponent bits, and thus preserving the range of representable values) or using one of the hardware or software supported conversions between IEEE formats, such as double to float or float to half (reducing both the number of exponent, as well as significand bits, and thus decreasing the range of representable values).

The precision_reduction class encodes the lower precision format relative to the base precision used and the algorithm in question. The encoding is done by specifying the amount of range non-preserving conversions and the amount of range preserving conversions that should be done on the base precision to obtain the lower precision format. For example, starting with a double precision value (11 exp, 52 sig. bits), the encoding specifying 1 non-preserving conversion and 1 preserving conversion would first use a hardware-supported non-preserving conversion to obtain a single precision value (8 exp, 23 sig. bits), followed by a preserving bit truncation to obtain a value with 8 exponent and 7 significand bits. Note that non-preserving conversion are always done first, as preserving conversions usually result in datatypes that are not supported by builtin conversions (thus, it is generally not possible to apply a non-preserving conversion to the result of a preserving conversion).

If the specified conversion is not supported by the algorithm, it will most likely fall back to using full precision for storing the data. Refer to the documentation of specific algorithms using this class for details about such special cases.

41.150.2 Constructor & Destructor Documentation

41.150.2.1 precision_reduction() [1/2]

```
constexpr gko::precision_reduction::precision_reduction ( )  [inline], [constexpr], [noexcept]
```

Creates a default precision_reduction encoding.

This encoding represents the case where no conversions are performed.

Referenced by common().

41.150.2.2 precision_reduction() [2/2]

```
constexpr gko::precision_reduction::precision_reduction (
            storage_type preserving,
            storage_type nonpreserving )  [inline], [constexpr], [noexcept]
```

Creates a precision_reduction encoding with the specified number of conversions.

Parameters

preserving	the number of range preserving conversion
nonpreserving	the number of range non-preserving conversions

41.150.3 Member Function Documentation

41.150.3.1 autodetect()

```
constexpr static precision_reduction gko::precision_reduction::autodetect ( )  [inline], [static],
[constexpr], [noexcept]
```

Returns a special encoding which instructs the algorithm to automatically detect the best precision.

Returns

a special encoding instructing the algorithm to automatically detect the best precision.

41.150.3.2 common()

```
constexpr static precision_reduction gko::precision_reduction::common (
            precision_reduction x,
            precision_reduction y )  [inline], [static], [constexpr], [noexcept]
```

Returns the common encoding of input encodings.

The common encoding is defined as the encoding that does not have more preserving, nor non-preserving conversions than the input encodings.

Parameters

x	an encoding
y	an encoding

Returns

the common encoding of `x` and `y`

References precision_reduction().

41.150.3.3 get_nonpreserving()

```
constexpr storage_type gko::precision_reduction::get_nonpreserving ( ) const  [inline], [constexpr],
[noexcept]
```

Returns the number of non-preserving conversions in the encoding.

Returns

the number of non-preserving conversions in the encoding.

41.150.3.4 get_preserving()

```
constexpr storage_type gko::precision_reduction::get_preserving ( ) const  [inline], [constexpr],
[noexcept]
```

Returns the number of preserving conversions in the encoding.

Returns

the number of preserving conversions in the encoding.

41.150.3.5 operator storage_type()

```
constexpr gko::precision_reduction::operator storage_type ( ) const [inline], [constexpr],
[noexcept]
```

Extracts the raw data of the encoding.

Returns

the raw data of the encoding

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

- ginkgo/core/base/types.hpp

41.151 gko::Preconditionable Class Reference

A LinOp implementing this interface can be preconditioned.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Public Member Functions

- virtual std::shared_ptr< const LinOp > get_preconditioner () const
  Returns the preconditioner operator used by the Preconditionable.
- virtual void set_preconditioner (std::shared_ptr< const LinOp > new_precond)
  Sets the preconditioner operator used by the Preconditionable.

41.151.1 Detailed Description

A LinOp implementing this interface can be preconditioned.

41.151.2 Member Function Documentation

41.151.2.1 get_preconditioner()

```
virtual std::shared_ptr<const LinOp> gko::Preconditionable::get_preconditioner ( ) const
[inline], [virtual]
```

Returns the preconditioner operator used by the Preconditionable.

Returns

the preconditioner operator used by the Preconditionable

41.151.2.2 set_preconditioner()

```
virtual void gko::Preconditionable::set_preconditioner (
            std::shared_ptr< const LinOp > new_precond ) [inline], [virtual]
```

Sets the preconditioner operator used by the Preconditionable.

Parameters

new_precond	the new preconditioner operator used by the Preconditionable

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

- ginkgo/core/base/lin_op.hpp

41.152 gko::syn::range< Start, End, Step > Struct Template Reference

range records start, end, step in template

```
#include <ginkgo/core/synthesizer/containers.hpp>
```

41.152.1 Detailed Description

template<int Start, int End, int Step = 1>
struct gko::syn::range< Start, End, Step >

range records start, end, step in template

Template Parameters

Start	start of range
End	end of range
Step	step of range. default is 1

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

- ginkgo/core/synthesizer/containers.hpp

41.153 gko::range< Accessor > Class Template Reference

A range is a multidimensional view of the memory.

```
#include <ginkgo/core/base/range.hpp>
```

Public Types

- using accessor = Accessor
  The type of the underlying accessor.

Public Member Functions

- ~range ()=default

  Use the default destructor.
- template<typename... AccessorParams>
  constexpr range (AccessorParams &&... params)

  Creates a new range.
- template<typename... DimensionTypes>
  constexpr auto operator() (DimensionTypes &&... dimensions) const -> decltype(std::declval< accessor >()(std::forward< DimensionTypes >(dimensions)...))

  Returns a value (or a sub-range) with the specified indexes.
- template<typename OtherAccessor >
  const range & operator= (const range< OtherAccessor > &other) const
- const range & operator= (const range &other) const

  Assigns another range to this range.
- constexpr size_type length (size_type dimension) const

  Returns the length of the specified dimension of the range.
- constexpr const accessor ∗ operator-> () const noexcept

  Returns a pointer to the accessor.
- constexpr const accessor & get_accessor () const noexcept

  `Returns a reference to the accessor.

Static Public Attributes

- static constexpr size_type dimensionality = accessor::dimensionality

  The number of dimensions of the range.

41.153.1 Detailed Description

template<typename Accessor>
class gko::range< Accessor >

A range is a multidimensional view of the memory.

The range does not store any of its values by itself. Instead, it obtains the values through an accessor (e.g. accessor::row_major) which describes how the indexes of the range map to physical locations in memory.

There are several advantages of using ranges instead of plain memory pointers:

1. Code using ranges is easier to read and write, as there is no need for index linearizations.
2. Code using ranges is safer, as it is impossible to accidentally miscalculate an index or step out of bounds, since range accessors perform bounds checking in debug builds. For performance, this can be disabled in release builds by defining the `NDEBUG` flag.
3. Ranges enable generalized code, as algorithms can be written independent of the memory layout. This does not impede various optimizations based on memory layout, as it is always possible to specialize algorithms for ranges with specific memory layouts.
4. Ranges have various pointwise operations predefined, which reduces the amount of loops that need to be written.

41.153.1.1 Range operations

Ranges define a complete set of pointwise unary and binary operators which extend the basic arithmetic operators in C++, as well as a few pointwise operations and mathematical functions useful in ginkgo, and a couple of non-pointwise operations. Compound assignment (+=, *=, etc.) is not yet supported at this moment. Here is a complete list of operations:

- standard unary operations: +, -, `!`, ~
- standard binary operations: +, * (this is pointwise, not matrix multiplication), `/`, `%`, <, >, <=, >=, ==, `!=`, ||, `&&`, |, `&`, ^, <<, >>
- useful unary functions: `zero`, `one`, `abs`, `real`, `imag`, `conj`, `squared_norm`
- useful binary functions: `min`, `max`

All binary pointwise operations also work as expected if one of the operands is a scalar and the other is a range. The scalar operand will have the effect as if it was a range of the same size as the other operand, filled with the specified scalar.

Two "global" functions `transpose` and `mmul` are also supported. `transpose` transposes the first two dimensions of the range (i.e. `transpose(r)(i, j, ...) == r(j, i, ...)`). `mmul` performs a (batched) matrix multiply of the ranges - the first two dimensions represent the matrices, while the rest represent the batch. For example, given the ranges `r1` and `r2` of dimensions `(3, 2, 3)` and `(2, 4, 3)`, respectively, `mmul(r1, r2)` will return a range of dimensions `(3, 4, 3)`, obtained by multiplying the 3 frontal slices of the range, and stacking the result back vertically.

41.153.1.2 Compound operations

Multiple range operations can be combined into a single expression. For example, an "axpy" operation can be obtained using `y = alpha * x + y`, where `x` an `y` are ranges, and `alpha` is a scalar. Range operations are optimized for memory access, and the above code does not allocate additional storage for intermediate ranges `alpha * x` or `aplha * x + y`. In fact, the entire computation is done during the assignment, and the results of operations + and * only register the data, and the types of operations that will be computed once the results are needed.

It is possible to store and reuse these intermediate expressions. The following example will overwrite the range `x` with it's 4th power:

```
{c++}
auto square = x * x;  // this is range constructor, not range assignment!
x = square;  // overwrites x with x*x (this is range assignment)
x = square;  // overwrites new x (x*x) with (x*x)*(x*x) (as is this)
```

41.153.1.3 Caveats

__`mmul` is not a highly-optimized BLAS-3 version of the matrix multiplication.__ The current design of ranges and accessors prevents that, so if you need a high-perfromance matrix multiplication, you should use one of the libraries that provide that, or implement your own (you can use pointwise range operations to help simplify that). However, range design might get improved in the future to allow efficient implementations of BLAS-3 kernels.

Aliasing the result range in `mmul` and `transpose` is not allowed. Constructs like `A = transpose(A)`, `A = mmul(A, A)`, or `A = mmul(A, A) + C` lead to undefined behavior. However, aliasing input arguments is allowed: `C = mmul(A, A)`, and even `C = mmul(A, A) + C` is valid code (in the last example, only pointwise operations are aliased). `C = mmul(A, A + C)` is not valid though.

41.153.1.4 Examples

The range unit tests in core/test/base/range.cpp contain lots of simple 1-line examples of range operations. The accessor unit tests in core/test/base/range.cpp show how to use ranges with concrete accessors, and how to use range slices using `span`s as arguments to range function call operator. Finally, examples/range contains a complete example where ranges are used to implement a simple version of the right-looking LU factorization.

Template Parameters

Accessor	underlying accessor of the range

41.153.2 Constructor & Destructor Documentation

41.153.2.1 range()

```
template<typename Accessor>
template<typename... AccessorParams>
constexpr gko::range< Accessor >::range (
            AccessorParams &&... params ) [inline], [explicit], [constexpr]
```

Creates a new range.

Template Parameters

AccessorParam	types of parameters forwarded to the accessor constructor

Parameters

params	parameters forwarded to Accessor constructor.

```
332         : accessor_{std::forward<AccessorParams>(params)...}
333     {}
```

41.153.3 Member Function Documentation

41.153.3.1 get_accessor()

```
template<typename Accessor>
constexpr const accessor& gko::range< Accessor >::get_accessor ( ) const [inline], [constexpr],
[noexcept]
```

`Returns a reference to the accessor.

Returns

reference to the accessor

Referenced by gko::range< Accessor >::operator=().

41.153.3.2 length()

```
template<typename Accessor>
constexpr size_type gko::range< Accessor >::length (
            size_type dimension ) const  [inline], [constexpr]
```

Returns the length of the specified dimension of the range.

Parameters

dimension	the dimensions whose length is returned

Returns

the length of the `dimension`-th dimension of the range

Referenced by gko::matrix_data< ValueType, IndexType >::matrix_data().

41.153.3.3 operator()()

```
template<typename Accessor>
template<typename... DimensionTypes>
constexpr auto gko::range< Accessor >::operator() (
            DimensionTypes &&... dimensions ) const -> decltype(std::declval<accessor>()(
std::forward<DimensionTypes>(dimensions)...))   [inline], [constexpr]
```

Returns a value (or a sub-range) with the specified indexes.

Template Parameters

DimensionTypes	The types of indexes. Supported types depend on the underlying accessor, but are usually either integer types or spans. If at least one index is a span, the returned value will be a sub-range.

Parameters

dimensions	the indexes of the values.

Returns

a value on position `(dimensions...)`.

References gko::range< Accessor >::dimensionality.

41.153.3.4 operator->()

```
template<typename Accessor>
constexpr const accessor* gko::range< Accessor >::operator-> ( ) const  [inline], [constexpr],
[noexcept]
```

Returns a pointer to the accessor.

Can be used to access data and functions of a specific accessor.

Returns

pointer to the accessor

41.153.3.5 operator=() [1/2]

```
template<typename Accessor>
const range& gko::range< Accessor >::operator= (
            const range< Accessor > & other ) const  [inline]
```

Assigns another range to this range.

The order of assignment is defined by the accessor of this range, thus the memory access will be optimized for the resulting range, and not for the other range. If the sizes of two ranges do not match, the result is undefined. Sizes of the ranges are checked at runtime in debug builds.

Note

Temporary accessors are allowed to define the implementation of the assignment as deleted, so do not expect `r1 * r2 = r2` to work.

Parameters

other	the range to copy the data from

References gko::range< Accessor >::get_accessor().

41.153.3.6 operator=() [2/2]

```
template<typename Accessor>
template<typename OtherAccessor >
const range& gko::range< Accessor >::operator= (
            const range< OtherAccessor > & other ) const  [inline]
```

This is a version of the function which allows to copy between ranges of different accessors.

Template Parameters

OtherAccessor	accessor of the other range

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

- ginkgo/core/base/range.hpp

41.154 gko::reorder::Rcm< ValueType, IndexType > Class Template Reference

Rcm is a reordering algorithm minimizing the bandwidth of a matrix.

`#include <ginkgo/core/reorder/rcm.hpp>`

Public Member Functions

- std::shared_ptr< const PermutationMatrix > get_permutation () const
  Gets the permutation (permutation matrix, output of the algorithm) of the linear operator.
- std::shared_ptr< const PermutationMatrix > get_inverse_permutation () const
  Gets the inverse permutation (permutation matrix, output of the algorithm) of the linear operator.

41.154.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::reorder::Rcm< ValueType, IndexType >

Rcm is a reordering algorithm minimizing the bandwidth of a matrix.

Such a reordering typically also significantly reduces fill-in, though usually not as effective as more complex algorithms, specifically AMD and nested dissection schemes. The advantage of this algorithm is its low runtime.

Note

> This class is derived from polymorphic object but is not a LinOp as it does not make sense for this class to implement the apply methods. The objective of this class is to generate a reordering/permutation vector (in the form of the Permutation matrix), which can be used to apply to reorder a matrix as required.

There are two "starting strategies" currently available: minimum degree and pseudo-peripheral. These strategies control how a starting vertex for a connected component is choosen, which is then renumbered as first vertex in the component, starting the algorithm from there. In general, the bandwidths obtained by choosing a pseudo-peripheral vertex are slightly smaller than those obtained from choosing a vertex of minimum degree. On the other hand, this strategy is much more expensive, relatively. The algorithm for finding a pseudo-peripheral vertex as described in "Computer Solution of Sparse Linear Systems" (George, Liu, Ng, Oak Ridge National Laboratory, 1994) is implemented here.

Template Parameters

ValueType	Type of the values of all matrices used in this class
IndexType	Type of the indices of all matrices used in this class

41.154.2 Member Function Documentation

41.154.2.1 get_inverse_permutation()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<const PermutationMatrix> gko::reorder::Rcm< ValueType, IndexType >::get_↵
inverse_permutation ( ) const [inline]
```

Gets the inverse permutation (permutation matrix, output of the algorithm) of the linear operator.

Returns

the inverse permutation (permutation matrix)

```
{
    return inv_permutation_;
}
```

41.154.2.2 get_permutation()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<const PermutationMatrix> gko::reorder::Rcm< ValueType, IndexType >::get_↵
permutation ( ) const [inline]
```

Gets the permutation (permutation matrix, output of the algorithm) of the linear operator.

Returns

the permutation (permutation matrix)

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

- ginkgo/core/reorder/rcm.hpp

41.155 gko::ReadableFromMatrixData< ValueType, IndexType > Class Template Reference

A LinOp implementing this interface can read its data from a matrix_data structure.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Public Member Functions

- virtual void read (const matrix_data< ValueType, IndexType > &data)=0
  Reads a matrix from a matrix_data structure.
- void read (const matrix_assembly_data< ValueType, IndexType > &data)
  Reads a matrix from a matrix_assembly_data structure.
- virtual void read (const device_matrix_data< ValueType, IndexType > &data)
  Reads a matrix from a device_matrix_data structure.
- virtual void read (device_matrix_data< ValueType, IndexType > &&data)
  Reads a matrix from a device_matrix_data structure.

41.155.1 Detailed Description

template< typename ValueType, typename IndexType >
class gko::ReadableFromMatrixData< ValueType, IndexType >

A LinOp implementing this interface can read its data from a matrix_data structure.

41.155.2 Member Function Documentation

41.155.2.1 read() [1/4]

```
template<typename ValueType, typename IndexType>
virtual void gko::ReadableFromMatrixData< ValueType, IndexType >::read (
            const device_matrix_data< ValueType, IndexType > & data )  [inline], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Hybrid< ValueType, IndexType >, gko::matrix::Fbcsr< ValueType, IndexType >, gko::matrix::Coo< ValueType, IndexType >, gko::matrix::Ell< ValueType, IndexType > gko::matrix::SparsityCsr< ValueType, IndexType >, and gko::matrix::Sellp< ValueType, IndexType >.

41.155.2.2 read() [2/4]

```
template<typename ValueType, typename IndexType>
void gko::ReadableFromMatrixData< ValueType, IndexType >::read (
            const matrix_assembly_data< ValueType, IndexType > & data )  [inline]
```

Reads a matrix from a matrix_assembly_data structure.

Parameters

data	the matrix_assembly_data structure

41.155.2.3 read() [3/4]

```
template<typename ValueType, typename IndexType>
virtual void gko::ReadableFromMatrixData< ValueType, IndexType >::read (
            const matrix_data< ValueType, IndexType > & data )  [pure virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Hybrid< ValueType, IndexType >, gko::matrix::Fbcsr< ValueType, IndexType >, gko::matrix::Coo< ValueType, IndexType >, gko::matrix::Ell< ValueType, IndexType > gko::matrix::SparsityCsr< ValueType, IndexType >, and gko::matrix::Sellp< ValueType, IndexType >.

41.155.2.4 read() [4/4]

```
template<typename ValueType, typename IndexType>
virtual void gko::ReadableFromMatrixData< ValueType, IndexType >::read (
            device_matrix_data< ValueType, IndexType > && data )  [inline], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Hybrid< ValueType, IndexType >, gko::matrix::Fbcsr< ValueType, IndexType >, gko::matrix::Coo< ValueType, IndexType >, gko::matrix::Ell< ValueType, IndexType > gko::matrix::SparsityCsr< ValueType, IndexType >, and gko::matrix::Sellp< ValueType, IndexType >.

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

- ginkgo/core/base/lin_op.hpp

41.156 gko::log::Record Class Reference

Record is a Logger which logs every event to an object.

```
#include <ginkgo/core/log/record.hpp>
```

Classes

- struct logged_data

  Struct storing the actually logged data.

Public Member Functions

- const logged_data & get () const noexcept

  Returns the logged data.
- logged_data & get () noexcept

Static Public Member Functions

- static std::unique_ptr< Record > create (std::shared_ptr< const Executor > exec, const mask_type &enabled_events=Logger::all_events_mask, size_type max_storage=1)

  Creates a Record logger.

41.156.1 Detailed Description

Record is a Logger which logs every event to an object.

The object can then be accessed at any time by asking the logger to return it.

Note

Please note that this logger can have significant memory and performance overhead. In particular, when logging events such as the `check` events, all parameters are cloned. If it is sufficient to clone one parameter, consider implementing a specific logger for this. In addition, it is advised to tune the history size in order to control memory overhead.

41.156.2 Member Function Documentation

41.156.2.1 create()

```
static std::unique_ptr<Record> gko::log::Record::create (
            std::shared_ptr< const Executor > exec,
            const mask_type & enabled_events = Logger::all_events_mask,
            size_type max_storage = 1 )  [inline], [static]
```

Creates a Record logger.

This dynamically allocates the memory, constructs the object and returns an std::unique_ptr to this object.

Parameters

exec	the executor
enabled_events	the events enabled for this logger. By default all events.
max_storage	the size of storage (i.e. history) wanted by the user. By default 0 is used, which means unlimited storage. It is advised to control this to reduce memory overhead of this logger.

Returns

an std::unique_ptr to the the constructed object

```
{
    return std::unique_ptr<Record>(
        new Record(exec, enabled_events, max_storage));
}
```

41.156.2.2 get() [1/2]

`const logged_data& gko::log::Record::get ( ) const [inline], [noexcept]`

Returns the logged data.

Returns

the logged data

41.156.2.3 get() [2/2]

`logged_data& gko::log::Record::get ( ) [inline], [noexcept]`

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

- ginkgo/core/log/record.hpp

41.157 gko::ReferenceExecutor Class Reference

This is a specialization of the OmpExecutor, which runs the reference implementations of the kernels used for debugging purposes.

`#include <ginkgo/core/base/executor.hpp>`

Public Member Functions

- void run (const Operation &op) const override

  Runs the specified Operation using this Executor.

Additional Inherited Members

41.157.1 Detailed Description

This is a specialization of the OmpExecutor, which runs the reference implementations of the kernels used for debugging purposes.

41.157.2 Member Function Documentation

41.157.2.1 run()

```
void gko::ReferenceExecutor::run (
            const Operation & op ) const [inline], [override], [virtual]
```

Runs the specified Operation using this Executor.

Parameters

op	the operation to run

Implements gko::Executor.

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

- ginkgo/core/base/executor.hpp

41.158 gko::stop::RelativeResidualNorm< ValueType > Class Template Reference

The RelativeResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the right-hand side, i.e.

`#include <ginkgo/core/stop/residual_norm.hpp>`

41.158.1 Detailed Description

template<typename ValueType = default_precision>
class gko::stop::RelativeResidualNorm< ValueType >

The RelativeResidualNorm class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the right-hand side, i.e.

when norm(residual) / norm(right_hand_side) < threshold. For better performance, the checks are run thanks to kernels on the executor where the algorithm is executed.

Note

> To use this stopping criterion there are some dependencies. The constructor depends on `b` in order to compute the norm of the right-hand side. If this is not correctly provided, an exception ::gko::NotSupported() is thrown.

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

- ginkgo/core/stop/residual_norm.hpp

41.159 gko::reorder::ReorderingBase Class Reference

The ReorderingBase class is a base class for all the reordering algorithms.

`#include <ginkgo/core/reorder/reordering_base.hpp>`

Additional Inherited Members

41.159.1 Detailed Description

The ReorderingBase class is a base class for all the reordering algorithms.

It contains a factory to instantiate the reorderings. It is up to each specific reordering to decide what to do with the data that is passed to it.

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

- ginkgo/core/reorder/reordering_base.hpp

41.160 gko::reorder::ReorderingBaseArgs Struct Reference

This struct is used to pass parameters to the EnableDefaultReorderingBaseFactory::generate() method.

`#include <ginkgo/core/reorder/reordering_base.hpp>`

41.160.1 Detailed Description

This struct is used to pass parameters to the EnableDefaultReorderingBaseFactory::generate() method.

It is the ComponentsType of ReorderingBaseFactory.

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

- ginkgo/core/reorder/reordering_base.hpp

41.161 gko::stop::ResidualNorm< ValueType > Class Template Reference

The ResidualNorm class is a stopping criterion which stops the iteration process when the actual residual norm is below a certain threshold relative to.

`#include <ginkgo/core/stop/residual_norm.hpp>`

41.161.1 Detailed Description

template<typename ValueType = default_precision>
class gko::stop::ResidualNorm< ValueType >

The ResidualNorm class is a stopping criterion which stops the iteration process when the actual residual norm is below a certain threshold relative to.

1. the norm of the right-hand side, norm(residual) / norm(right_hand_side) < threshold
2. the initial residual, norm(residual) / norm(initial_residual) < threshold.
3. one, norm(residual) < threshold.

For better performance, the checks are run on the executor where the algorithm is executed.

Note

To use this stopping criterion there are some dependencies. The constructor depends on either `b` or the `initial_residual` in order to compute their norms. If this is not correctly provided, an exception ::gko↩ ::NotSupported() is thrown.

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

- ginkgo/core/stop/residual_norm.hpp

41.162 gko::stop::ResidualNormBase< ValueType > Class Template Reference

The ResidualNormBase class provides a framework for stopping criteria related to the residual norm.

```
#include <ginkgo/core/stop/residual_norm.hpp>
```

41.162.1 Detailed Description

template<typename ValueType>
class gko::stop::ResidualNormBase< ValueType >

The ResidualNormBase class provides a framework for stopping criteria related to the residual norm.

These criteria differ in the way they initialize starting_tau_, so in the value they compare the residual norm against. The provided check_impl uses the actual residual to check for convergence.

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

- ginkgo/core/stop/residual_norm.hpp

41.163 gko::stop::ResidualNormReduction< ValueType > Class Template Reference

The ResidualNormReduction class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the initial residual, i.e.

```
#include <ginkgo/core/stop/residual_norm.hpp>
```

41.163.1 Detailed Description

template<typename ValueType = default_precision>
class gko::stop::ResidualNormReduction< ValueType >

The ResidualNormReduction class is a stopping criterion which stops the iteration process when the residual norm is below a certain threshold relative to the norm of the initial residual, i.e.

when norm(residual) / norm(initial_residual) < threshold. For better performance, the checks are run thanks to kernels on the executor where the algorithm is executed.

Note

> To use this stopping criterion there are some dependencies. The constructor depends on `initial_residual` in order to compute the first relative residual norm. The check method depends on either the `residual_norm` or the `residual` being set. When any of those is not correctly provided, an exception ::gko::NotSupported() is thrown.

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

- ginkgo/core/stop/residual_norm.hpp

41.164 gko::accessor::row_major< ValueType, Dimensionality > Class Template Reference

A row_major accessor is a bridge between a range and the row-major memory layout.

```
#include <ginkgo/core/base/range_accessors.hpp>
```

Public Types

- using value_type = ValueType
  Type of values returned by the accessor.
- using data_type = value_type ∗
  Type of underlying data storage.

Public Member Functions

- constexpr value_type & operator() (size_type row, size_type col) const
  Returns the data element at position (row, col)
- constexpr range< row_major > operator() (const span &rows, const span &cols) const
  Returns the sub-range spanning the range (rows, cols)
- constexpr size_type length (size_type dimension) const
  Returns the length in dimension `dimension`.
- template<typename OtherAccessor >
  void copy_from (const OtherAccessor &other) const
  Copies data from another accessor.

Public Attributes

- const data_type data
  Reference to the underlying data.
- const std::array< const size_type, dimensionality > lengths
  An array of dimension sizes.
- const size_type stride
  Distance between consecutive rows.

Static Public Attributes

- static constexpr size_type dimensionality = 2
  Number of dimensions of the accessor.

41.164.1 Detailed Description

template<typename ValueType, size_type Dimensionality>
class gko::accessor::row_major< ValueType, Dimensionality >

A row_major accessor is a bridge between a range and the row-major memory layout.

You should never try to explicitly create an instance of this accessor. Instead, supply it as a template parameter to a range, and pass the constructor parameters for this class to the range (it will forward it to this class).

Warning

The current implementation is incomplete, and only allows for 2-dimensional ranges.

Template Parameters

ValueType	type of values this accessor returns
Dimensionality	number of dimensions of this accessor (has to be 2)

41.164.2 Member Function Documentation

41.164.2.1 copy_from()

```
template<typename ValueType , size_type Dimensionality>
template<typename OtherAccessor >
void gko::accessor::row_major< ValueType, Dimensionality >::copy_from (
            const OtherAccessor & other ) const  [inline]
```

Copies data from another accessor.

Warning

Do not use this function since it is not optimized for a specific executor. It will always be performed sequentially. Please write an optimized version (adjusted to the architecture) by iterating through the values yourself.

Template Parameters

OtherAccessor	type of the other accessor

Parameters

other	other accessor

```
    {
        for (size_type i = 0; i < lengths[0]; ++i) {
            for (size_type j = 0; j < lengths[1]; ++j) {
                (*this)(i, j) = other(i, j);
            }
        }
    }
```

References gko::accessor::row_major< ValueType, Dimensionality >::lengths.

41.164.2.2 length()

```
template<typename ValueType , size_type Dimensionality>
constexpr size_type gko::accessor::row_major< ValueType, Dimensionality >::length (
            size_type dimension ) const  [inline], [constexpr]
```

Returns the length in dimension `dimension`.

Parameters

dimension	a dimension index

Returns

length in dimension `dimension`

References gko::accessor::row_major< ValueType, Dimensionality >::lengths.

41.164.2.3 operator()() [1/2]

```
template<typename ValueType , size_type Dimensionality>
constexpr range<row_major> gko::accessor::row_major< ValueType, Dimensionality >::operator()
(
            const span & rows,
            const span & cols ) const [inline], [constexpr]
```

Returns the sub-range spanning the range (rows, cols)

Parameters

rows	row span
cols	column span

Returns

sub-range spanning the range (rows, cols)

References gko::span::begin, gko::accessor::row_major< ValueType, Dimensionality >::data, gko::span::end, gko::span::is_valid(), gko::accessor::row_major< ValueType, Dimensionality >::lengths, and gko::accessor::row_major< ValueType, Dimensionality >::stride.

41.164.2.4 operator()() [2/2]

```
template<typename ValueType , size_type Dimensionality>
constexpr value_type& gko::accessor::row_major< ValueType, Dimensionality >::operator() (
            size_type row,
            size_type col ) const [inline], [constexpr]
```

Returns the data element at position (row, col)

Parameters

row	row index
col	column index

Returns

data element at (row, col)

References gko::accessor::row_major< ValueType, Dimensionality >::data, gko::accessor::row_major< Value↔
Type, Dimensionality >::lengths, and gko::accessor::row_major< ValueType, Dimensionality >::stride.

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

- ginkgo/core/base/range_accessors.hpp

41.165 gko::matrix::RowGatherer< IndexType > Class Template Reference

RowGatherer is a matrix "format" which stores the gather indices arrays which can be used to gather rows to another matrix.

```
#include <ginkgo/core/matrix/row_gatherer.hpp>
```

Public Member Functions

- index_type ∗ get_row_idxs () noexcept
  Returns a pointer to the row index array for gathering.
- const index_type ∗ get_const_row_idxs () const noexcept
  Returns a pointer to the row index array for gathering.

Static Public Member Functions

- static std::unique_ptr< const RowGatherer > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< IndexType > &&row_idxs)
  Creates a constant (immutable) RowGatherer matrix from a constant array.

41.165.1 Detailed Description

template<typename IndexType = int32>
class gko::matrix::RowGatherer< IndexType >

RowGatherer is a matrix "format" which stores the gather indices arrays which can be used to gather rows to another matrix.

Template Parameters

IndexType	precision of rowgatherer array indices.

Note

This format is used mainly to allow for an abstraction of the rowgatherer and provides the user with an apply method which calls the respective Dense rowgatherer operation. As such it only stores an array of the rowgatherer indices.

41.165.2 Member Function Documentation

41.165.2.1 create_const()

```
template<typename IndexType = int32>
static std::unique_ptr<const RowGatherer> gko::matrix::RowGatherer< IndexType >::create_const
(
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            gko::detail::ConstArrayView< IndexType > && row_idxs ) [inline], [static]
```

Creates a constant (immutable) RowGatherer matrix from a constant array.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
row_idxs	the gathered row indices of the matrix

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of the arrays on the correct executor.

```
{
    // cast const-ness away, but return a const object afterwards,
    // so we can ensure that no modifications take place.
    return std::unique_ptr<const RowGatherer>(new RowGatherer{
        exec, size, gko::detail::array_const_cast(std::move(row_idxs))});
}
```

41.165.2.2 get_const_row_idxs()

```
template<typename IndexType = int32>
const index_type* gko::matrix::RowGatherer< IndexType >::get_const_row_idxs ( ) const [inline],
[noexcept]
```

Returns a pointer to the row index array for gathering.

Returns

the pointer to the row index array for gathering.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.165.2.3 get_row_idxs()

```
template<typename IndexType = int32>
index_type* gko::matrix::RowGatherer< IndexType >::get_row_idxs ( )  [inline], [noexcept]
```

Returns a pointer to the row index array for gathering.

Returns

the pointer to the row index array for gathering.

References gko::Array< ValueType >::get_data().

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

- ginkgo/core/matrix/row_gatherer.hpp

41.166 gko::ScaledIdentityAddable Class Reference

Adds the operation M <- a I + b M for matrix M, identity operator I and scalars a and b, where M is the calling object.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Public Member Functions

- void add_scaled_identity (const LinOp ∗const a, const LinOp ∗const b)
  Scales this and adds another scalar times the identity to it.

41.166.1 Detailed Description

Adds the operation M <- a I + b M for matrix M, identity operator I and scalars a and b, where M is the calling object.

41.166.2 Member Function Documentation

41.166.2.1 add_scaled_identity()

```
void gko::ScaledIdentityAddable::add_scaled_identity (
            const LinOp *const a,
            const LinOp *const b )  [inline]
```

Scales this and adds another scalar times the identity to it.

Parameters

a	Scalar to multiply the identity operator before adding.
b	Scalar to multiply this before adding the scaled identity to it.

References gko::make_temporary_clone().

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

- ginkgo/core/base/lin_op.hpp

41.167 gko::matrix::Sellp< ValueType, IndexType > Class Template Reference

SELL-P is a matrix format similar to ELL format.

```
#include <ginkgo/core/matrix/sellp.hpp>
```

Public Member Functions

- void read (const mat_data &data) override
  Reads a matrix from a matrix_data structure.
- void read (const device_mat_data &data) override
  Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override
  Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override
  Writes a matrix to a matrix_data structure.
- std::unique_ptr< Diagonal< ValueType > > extract_diagonal () const override
  Extracts the diagonal entries of the matrix into a vector.
- std::unique_ptr< absolute_type > compute_absolute () const override
  Gets the AbsoluteLinOp.
- void compute_absolute_inplace () override
  Compute absolute inplace on each element.
- value_type * get_values () noexcept
  Returns the values of the matrix.
- const value_type * get_const_values () const noexcept
  Returns the values of the matrix.
- index_type * get_col_idxs () noexcept
  Returns the column indexes of the matrix.
- const index_type * get_const_col_idxs () const noexcept
  Returns the column indexes of the matrix.
- size_type * get_slice_lengths () noexcept
  Returns the lengths(columns) of slices.
- const size_type * get_const_slice_lengths () const noexcept
  Returns the lengths(columns) of slices.
- size_type * get_slice_sets () noexcept
  Returns the offsets of slices.

- const size_type * get_const_slice_sets () const noexcept

    Returns the offsets of slices.
- size_type get_slice_size () const noexcept

    Returns the size of a slice.
- size_type get_stride_factor () const noexcept

    Returns the stride factor(t) of SELL-P.
- size_type get_total_cols () const noexcept

    Returns the total column number.
- size_type get_num_stored_elements () const noexcept

    Returns the number of elements explicitly stored in the matrix.
- value_type & val_at (size_type row, size_type slice_set, size_type idx) noexcept

    Returns the `idx`-th non-zero element of the `row`-th row with `slice_set` slice set.
- value_type val_at (size_type row, size_type slice_set, size_type idx) const noexcept

    Returns the `idx`-th non-zero element of the `row`-th row with `slice_set` slice set.
- index_type & col_at (size_type row, size_type slice_set, size_type idx) noexcept

    Returns the `idx`-th column index of the `row`-th row with `slice_set` slice set.
- index_type col_at (size_type row, size_type slice_set, size_type idx) const noexcept

    Returns the `idx`-th column index of the `row`-th row with `slice_set` slice set.

41.167.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Sellp< ValueType, IndexType >

SELL-P is a matrix format similar to ELL format.

The difference is that SELL-P format divides rows into smaller slices and store each slice with ELL format.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indexes

41.167.2 Member Function Documentation

41.167.2.1 col_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type gko::matrix::Sellp< ValueType, IndexType >::col_at (
            size_type row,
            size_type slice_set,
            size_type idx ) const  [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row with `slice_set` slice set.

Parameters

row	the row of the requested element in the slice
slice_set	the slice set of the slice
idx	the idx-th stored element of the row in the slice

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the CPU results in a runtime error)

```
{
    return this
        ->get_const_col_idxs()[this->linearize_index(row, slice_set, idx)];
}
```

41.167.2.2 col_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type& gko::matrix::Sellp< ValueType, IndexType >::col_at (
            size_type row,
            size_type slice_set,
            size_type idx ) [inline], [noexcept]
```

Returns the `idx`-th column index of the `row`-th row with `slice_set` slice set.

Parameters

row	the row of the requested element in the slice
slice_set	the slice set of the slice
idx	the idx-th stored element of the row in the slice

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the CPU results in a runtime error)

References gko::matrix::Sellp< ValueType, IndexType >::get_col_idxs().

41.167.2.3 compute_absolute()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<absolute_type> gko::matrix::Sellp< ValueType, IndexType >::compute_absolute (
) const [override], [virtual]
```

Gets the AbsoluteLinOp.

Returns

a pointer to the new absolute object

Implements gko::EnableAbsoluteComputation< remove_complex< Sellp< ValueType, IndexType > > >.

41.167.2.4 extract_diagonal()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<Diagonal<ValueType> > gko::matrix::Sellp< ValueType, IndexType >::extract_↩
diagonal ( ) const [override], [virtual]
```

Extracts the diagonal entries of the matrix into a vector.

Parameters

diag	the vector into which the diagonal will be written

Implements gko::DiagonalExtractable< ValueType >.

41.167.2.5 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::Sellp< ValueType, IndexType >::get_col_idxs ( ) [inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

References gko::Array< ValueType >::get_data().

Referenced by gko::matrix::Sellp< ValueType, IndexType >::col_at().

41.167.2.6 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::Sellp< ValueType, IndexType >::get_const_col_idxs ( ) const
[inline], [noexcept]
```

Returns the column indexes of the matrix.

Returns

the column indexes of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.167.2.7 get_const_slice_lengths()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const size_type* gko::matrix::Sellp< ValueType, IndexType >::get_const_slice_lengths ( ) const
[inline], [noexcept]
```

Returns the lengths(columns) of slices.

Returns

the lengths(columns) of slices.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.167.2.8 get_const_slice_sets()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const size_type* gko::matrix::Sellp< ValueType, IndexType >::get_const_slice_sets ( ) const
[inline], [noexcept]
```

Returns the offsets of slices.

Returns

the offsets of slices.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.167.2.9 get_const_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::Sellp< ValueType, IndexType >::get_const_values ( ) const  [inline],
[noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.167.2.10 get_num_stored_elements()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Sellp< ValueType, IndexType >::get_num_stored_elements ( ) const  [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.167.2.11 get_slice_lengths()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type* gko::matrix::Sellp< ValueType, IndexType >::get_slice_lengths ( )  [inline], [noexcept]
```

Returns the lengths(columns) of slices.

Returns

the lengths(columns) of slices.

References gko::Array< ValueType >::get_data().

41.167.2.12 get_slice_sets()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type* gko::matrix::Sellp< ValueType, IndexType >::get_slice_sets ( )  [inline], [noexcept]
```

Returns the offsets of slices.

Returns

the offsets of slices.

References gko::Array< ValueType >::get_data().

41.167.2.13 get_slice_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Sellp< ValueType, IndexType >::get_slice_size ( ) const  [inline],
[noexcept]
```

Returns the size of a slice.

Returns

the size of a slice.

41.167.2.14 get_stride_factor()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Sellp< ValueType, IndexType >::get_stride_factor ( ) const  [inline],
[noexcept]
```

Returns the stride factor(t) of SELL-P.

Returns

the stride factor(t) of SELL-P.

41.167.2.15 get_total_cols()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Sellp< ValueType, IndexType >::get_total_cols ( ) const  [inline],
[noexcept]
```

Returns the total column number.

Returns

the total column number.

References gko::Array< ValueType >::get_num_elems().

41.167.2.16 get_values()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::Sellp< ValueType, IndexType >::get_values ( )  [inline], [noexcept]
```

Returns the values of the matrix.

Returns

the values of the matrix.

References gko::Array< ValueType >::get_data().

41.167.2.17 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Sellp< ValueType, IndexType >::read (
            const device_mat_data & data )  [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.167.2.18 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Sellp< ValueType, IndexType >::read (
            const mat_data & data ) [override], [virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.167.2.19 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Sellp< ValueType, IndexType >::read (
            device_mat_data && data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.167.2.20 val_at() [1/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type gko::matrix::Sellp< ValueType, IndexType >::val_at (
            size_type row,
            size_type slice_set,
            size_type idx ) const [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row with `slice_set` slice set.

Parameters

row	the row of the requested element in the slice
slice_set	the slice set of the slice
idx	the idx-th stored element of the row in the slice

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the CPU results in a runtime error)

References gko::Array< ValueType >::get_const_data().

41.167.2.21 val_at() [2/2]

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type& gko::matrix::Sellp< ValueType, IndexType >::val_at (
            size_type row,
            size_type slice_set,
            size_type idx ) [inline], [noexcept]
```

Returns the `idx`-th non-zero element of the `row`-th row with `slice_set` slice set.

Parameters

row	the row of the requested element in the slice
slice_set	the slice set of the slice
idx	the idx-th stored element of the row in the slice

Note

the method has to be called on the same Executor the matrix is stored at (e.g. trying to call this method on a GPU matrix from the CPU results in a runtime error)

References gko::Array< ValueType >::get_data().

41.167.2.22 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Sellp< ValueType, IndexType >::write (
            mat_data & data ) const [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/csr.hpp
- ginkgo/core/matrix/sellp.hpp

41.168 gko::span Struct Reference

A span is a lightweight structure used to create sub-ranges from other ranges.

`#include <ginkgo/core/base/range.hpp>`

Public Member Functions

- constexpr span (size_type point) noexcept
  Creates a span representing a point `point`*.*
- constexpr span (size_type begin, size_type end) noexcept
  Creates a span.
- constexpr bool is_valid () const
  Checks if a span is valid.
- constexpr size_type length () const
  Returns the length of a span.

Public Attributes

- const size_type begin
  Beginning of the span.
- const size_type end
  End of the span.

41.168.1 Detailed Description

A span is a lightweight structure used to create sub-ranges from other ranges.

A `span s` represents a contiguous set of indexes in one dimension of the range, starting on index `s.begin` (inclusive) and ending at index `s.end` (exclusive). A span is only valid if its starting index is smaller than its ending index.

Spans can be compared using the `==` and `!=` operators. Two spans are identical if both their `begin` and `end` values are identical.

Spans also have two distinct partial orders defined on them:

1. `x < y` (`y > x`) if and only if `x.end < y.begin`
2. `x <= y` (`y >= x`) if and only if `x.end <= y.begin`

Note that the orders are in fact partial - there are spans `x` and `y` for which none of the following inequalities holds: `x < y`, `x > y`, `x == y`, `x <= y`, `x >= y`. An example are spans `span{0, 2}` and `span{1, 3}`.

In addition, `<=` is a distinct order from `<`, and not just an extension of the strict order to its weak equivalent. Thus, `x <= y` is not equivalent to `x < y || x == y`.

41.168.2 Constructor & Destructor Documentation

41.168.2.1 span() [1/2]

```
constexpr gko::span::span (
            size_type point )  [inline], [constexpr], [noexcept]
```

Creates a span representing a point `point`.

The `begin` of this span is set to `point`, and the `end` to `point + 1`.

Parameters

point	the point which the span represents

41.168.2.2 span() [2/2]

```
constexpr gko::span::span (
            size_type begin,
            size_type end )  [inline], [constexpr], [noexcept]
```

Creates a span.

Parameters

begin	the beginning of the span
end	the end of the span

References begin.

41.168.3 Member Function Documentation

41.168.3.1 is_valid()

```
constexpr bool gko::span::is_valid ( ) const  [inline], [constexpr]
```

Checks if a span is valid.

Returns

true if and only if `this->begin < this->end`

References begin, and end.

Referenced by gko::accessor::row_major< ValueType, Dimensionality >::operator()().

41.168.3.2 length()

```
constexpr size_type gko::span::length ( ) const  [inline], [constexpr]
```

Returns the length of a span.

Returns

`this->end - this->begin`

References begin, and end.

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

- ginkgo/core/base/range.hpp

41.169 gko::matrix::Csr< ValueType, IndexType >::sparselib Class Reference

sparselib is a strategy_type which uses the sparselib csr.

```
#include <ginkgo/core/matrix/csr.hpp>
```

Public Member Functions

- sparselib ()
  Creates a sparselib strategy.
- void process (const Array< index_type > &mtx_row_ptrs, Array< index_type > *mtx_srow) override
  Computes srow according to row pointers.
- int64_t clac_size (const int64_t nnz) override
  Computes the srow size according to the number of nonzeros.
- std::shared_ptr< strategy_type > copy () override
  Copy a strategy.

41.169.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >::sparselib

sparselib is a strategy_type which uses the sparselib csr.

Note

Uses cusparse in cuda and hipsparse in hip.

41.169.2 Member Function Documentation

41.169.2.1 clac_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
int64_t gko::matrix::Csr< ValueType, IndexType >::sparselib::clac_size (
            const int64_t nnz )  [inline], [override], [virtual]
```

Computes the srow size according to the number of nonzeros.

Parameters

nnz	the number of nonzeros

Returns

the size of srow

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.169.2.2 copy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::sparselib::copy ( )
[inline], [override], [virtual]
```

Copy a strategy.

This is a workaround until strategies are revamped, since strategies like `automatical` do not work when actually shared.

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

41.169.2.3 process()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Csr< ValueType, IndexType >::sparselib::process (
            const Array< index_type > & mtx_row_ptrs,
            Array< index_type > * mtx_srow )  [inline], [override], [virtual]
```

Computes srow according to row pointers.

Parameters

mtx_row_ptrs	the row pointers of the matrix
mtx_srow	the srow of the matrix

Implements gko::matrix::Csr< ValueType, IndexType >::strategy_type.

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

- ginkgo/core/matrix/csr.hpp

41.170 gko::matrix::SparsityCsr< ValueType, IndexType > Class Template Reference

SparsityCsr is a matrix format which stores only the sparsity pattern of a sparse matrix by compressing each row of the matrix (compressed sparse row format).

```
#include <ginkgo/core/matrix/sparsity_csr.hpp>
```

Public Member Functions

- void read (const mat_data &data) override
  Reads a matrix from a matrix_data structure.
- void read (const device_mat_data &data) override
  Reads a matrix from a device_matrix_data structure.
- void read (device_mat_data &&data) override
  Reads a matrix from a device_matrix_data structure.
- void write (mat_data &data) const override
  Writes a matrix to a matrix_data structure.
- std::unique_ptr< LinOp > transpose () const override
  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override
  Returns a LinOp representing the conjugate transpose of the Transposable object.
- std::unique_ptr< SparsityCsr > to_adjacency_matrix () const
  Transforms the sparsity matrix to an adjacency matrix.
- void sort_by_column_index ()
  Sorts each row by column index.
- index_type * get_col_idxs () noexcept
  Returns the column indices of the matrix.
- const index_type * get_const_col_idxs () const noexcept
  Returns the column indices of the matrix.
- index_type * get_row_ptrs () noexcept
  Returns the row pointers of the matrix.
- const index_type * get_const_row_ptrs () const noexcept
  Returns the row pointers of the matrix.
- value_type * get_value () noexcept
  Returns the value stored in the matrix.
- const value_type * get_const_value () const noexcept
  Returns the value stored in the matrix.
- size_type get_num_nonzeros () const noexcept
  Returns the number of elements explicitly stored in the matrix.

Static Public Member Functions

- static std::unique_ptr< const SparsityCsr > create_const (std::shared_ptr< const Executor > exec, const dim< 2 > &size, gko::detail::ConstArrayView< IndexType > &&col_idxs, gko::detail::ConstArrayView< IndexType > &&row_ptrs, ValueType value=one< ValueType >())
  Creates a constant (immutable) SparsityCsr matrix from constant arrays.

41.170.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::SparsityCsr< ValueType, IndexType >

SparsityCsr is a matrix format which stores only the sparsity pattern of a sparse matrix by compressing each row of the matrix (compressed sparse row format).

The values of the nonzero elements are stored as a value array of length 1. All the values in the matrix are equal to this value. By default, this value is set to 1.0. A row pointer array also stores the linearized starting index of each row. An additional column index array is used to identify the column where a nonzero is present.

Template Parameters

ValueType	precision of vectors in apply
IndexType	precision of matrix indexes

41.170.2 Member Function Documentation

41.170.2.1 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::SparsityCsr< ValueType, IndexType >::conj_transpose ( )
const [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.170.2.2 create_const()

```
template<typename ValueType = default_precision, typename IndexType = int32>
static std::unique_ptr<const SparsityCsr> gko::matrix::SparsityCsr< ValueType, IndexType >↩
::create_const (
            std::shared_ptr< const Executor > exec,
            const dim< 2 > & size,
            gko::detail::ConstArrayView< IndexType > && col_idxs,
            gko::detail::ConstArrayView< IndexType > && row_ptrs,
            ValueType value = one<ValueType>() ) [inline], [static]
```

Creates a constant (immutable) SparsityCsr matrix from constant arrays.

Parameters

exec	the executor to create the matrix on
size	the dimensions of the matrix
values	the value array of the matrix
col_idxs	the column index array of the matrix
row_ptrs	the row pointer array of the matrix
strategy	the strategy the matrix uses for SpMV operations

Returns

A smart pointer to the constant matrix wrapping the input arrays (if they reside on the same executor as the matrix) or a copy of these arrays on the correct executor.

```
    {
        // cast const-ness away, but return a const object afterwards,
        // so we can ensure that no modifications take place.
        return std::unique_ptr<const SparsityCsr>(new SparsityCsr{
            exec, size, gko::detail::array_const_cast(std::move(col_idxs)),
            gko::detail::array_const_cast(std::move(row_ptrs)), value});
    }
```

41.170.2.3 get_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::SparsityCsr< ValueType, IndexType >::get_col_idxs ( )  [inline],
[noexcept]
```

Returns the column indices of the matrix.

Returns

the column indices of the matrix.

References gko::Array< ValueType >::get_data().

41.170.2.4 get_const_col_idxs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::SparsityCsr< ValueType, IndexType >::get_const_col_idxs ( )
const  [inline], [noexcept]
```

Returns the column indices of the matrix.

Returns

the column indices of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.170.2.5 get_const_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const index_type* gko::matrix::SparsityCsr< ValueType, IndexType >::get_const_row_ptrs ( )
const [inline], [noexcept]
```

Returns the row pointers of the matrix.

Returns

the row pointers of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.170.2.6 get_const_value()

```
template<typename ValueType = default_precision, typename IndexType = int32>
const value_type* gko::matrix::SparsityCsr< ValueType, IndexType >::get_const_value ( ) const
[inline], [noexcept]
```

Returns the value stored in the matrix.

Returns

the value of the matrix.

Note

This is the constant version of the function, which can be significantly more memory efficient than the non-constant version, so always prefer this version.

References gko::Array< ValueType >::get_const_data().

41.170.2.7 get_num_nonzeros()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::SparsityCsr< ValueType, IndexType >::get_num_nonzeros ( ) const [inline],
[noexcept]
```

Returns the number of elements explicitly stored in the matrix.

Returns

the number of elements explicitly stored in the matrix

References gko::Array< ValueType >::get_num_elems().

41.170.2.8 get_row_ptrs()

```
template<typename ValueType = default_precision, typename IndexType = int32>
index_type* gko::matrix::SparsityCsr< ValueType, IndexType >::get_row_ptrs ( ) [inline],
[noexcept]
```

Returns the row pointers of the matrix.

Returns

the row pointers of the matrix.

References gko::Array< ValueType >::get_data().

41.170.2.9 get_value()

```
template<typename ValueType = default_precision, typename IndexType = int32>
value_type* gko::matrix::SparsityCsr< ValueType, IndexType >::get_value ( ) [inline], [noexcept]
```

Returns the value stored in the matrix.

Returns

the value of the matrix.

References gko::Array< ValueType >::get_data().

41.170.2.10 read() [1/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::SparsityCsr< ValueType, IndexType >::read (
            const device_mat_data & data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.170.2.11 read() [2/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
```

```
void gko::matrix::SparsityCsr< ValueType, IndexType >::read (
            const mat_data & data ) [override], [virtual]
```

Reads a matrix from a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::ReadableFromMatrixData< ValueType, IndexType >.

41.170.2.12 read() [3/3]

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::SparsityCsr< ValueType, IndexType >::read (
            device_mat_data && data ) [override], [virtual]
```

Reads a matrix from a device_matrix_data structure.

The structure may be emptied by this function.

Parameters

data	the device_matrix_data structure.

Reimplemented from gko::ReadableFromMatrixData< ValueType, IndexType >.

41.170.2.13 to_adjacency_matrix()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<SparsityCsr> gko::matrix::SparsityCsr< ValueType, IndexType >::to_adjacency↩
_matrix ( ) const
```

Transforms the sparsity matrix to an adjacency matrix.

As the adjacency matrix has to be square, the input SparsityCsr matrix for this function to work has to be square.

Note

> The adjacency matrix in this case is the sparsity pattern but with the diagonal ones removed. This is mainly used for the reordering/partitioning as taken in by graph libraries such as METIS.

41.170.2.14 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::matrix::SparsityCsr< ValueType, IndexType >::transpose ( ) const
[override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

41.170.2.15 write()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::SparsityCsr< ValueType, IndexType >::write (
            mat_data & data ) const  [override], [virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implements gko::WritableToMatrixData< ValueType, IndexType >.

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

- ginkgo/core/matrix/csr.hpp
- ginkgo/core/matrix/sparsity_csr.hpp

41.171 gko::stopping_status Class Reference

This class is used to keep track of the stopping status of one vector.

```
#include <ginkgo/core/stop/stopping_status.hpp>
```

Public Member Functions

- bool has_stopped () const noexcept

    Check if any stopping criteria was fulfilled.
- bool has_converged () const noexcept

    Check if convergence was reached.
- bool is_finalized () const noexcept

Check if the corresponding vector stores the finalized result.
- uint8 get_id () const noexcept

  Get the id of the stopping criterion which caused the stop.
- void reset () noexcept

  Clear all flags.
- void stop (uint8 id, bool set_finalized=true) noexcept

  Call if a stop occured due to a hard limit (and convergence was not reached).
- void converge (uint8 id, bool set_finalized=true) noexcept

  Call if convergence occured.
- void finalize () noexcept

  Set the result to be finalized (it needs to be stopped or converged first).

Friends

- bool operator== (const stopping_status &x, const stopping_status &y) noexcept

  Checks if two stopping statuses are equivalent.
- bool operator!= (const stopping_status &x, const stopping_status &y) noexcept

  Checks if two stopping statuses are different.

41.171.1 Detailed Description

This class is used to keep track of the stopping status of one vector.

41.171.2 Member Function Documentation

41.171.2.1 converge()

```
void gko::stopping_status::converge (
            uint8 id,
            bool set_finalized = true )  [inline], [noexcept]
```

Call if convergence occured.

Parameters

id	id of the stopping criteria.
set_finalized	Controls if the current version should count as finalized (set to true) or not (set to false).

References has_stopped().

41.171.2.2 get_id()

```
uint8 gko::stopping_status::get_id ( ) const  [inline], [noexcept]
```

Get the id of the stopping criterion which caused the stop.

Returns

Returns the id of the stopping criterion which caused the stop.

Referenced by has_stopped().

41.171.2.3 has_converged()

```
bool gko::stopping_status::has_converged ( ) const [inline], [noexcept]
```

Check if convergence was reached.

Returns

Returns true if convergence was reached.

41.171.2.4 has_stopped()

```
bool gko::stopping_status::has_stopped ( ) const [inline], [noexcept]
```

Check if any stopping criteria was fulfilled.

Returns

Returns true if any stopping criteria was fulfilled.

References get_id().

Referenced by converge(), finalize(), and stop().

41.171.2.5 is_finalized()

```
bool gko::stopping_status::is_finalized ( ) const [inline], [noexcept]
```

Check if the corresponding vector stores the finalized result.

Returns

Returns true if the corresponding vector stores the finalized result.

41.171.2.6 stop()

```
void gko::stopping_status::stop (
            uint8 id,
            bool set_finalized = true ) [inline], [noexcept]
```

Call if a stop occured due to a hard limit (and convergence was not reached).

Parameters

id	id of the stopping criteria.
set_finalized	Controls if the current version should count as finalized (set to true) or not (set to false).

References has_stopped().

41.171.3 Friends And Related Function Documentation

41.171.3.1 operator"!=

```
bool operator!= (
            const stopping_status & x,
            const stopping_status & y )  [friend]
```

Checks if two stopping statuses are different.

Parameters

x	a stopping status
y	a stopping status

Returns

true if and only if `!(x == y)`

41.171.3.2 operator==

```
bool operator== (
            const stopping_status & x,
            const stopping_status & y )  [friend]
```

Checks if two stopping statuses are equivalent.

Parameters

x	a stopping status
y	a stopping status

Returns

true if and only if both `x` and `y` have the same mask and converged and finalized state

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

- ginkgo/core/stop/stopping_status.hpp

41.172 gko::matrix::Csr< ValueType, IndexType >::strategy_type Class Reference

strategy_type is to decide how to set the csr algorithm.

`#include <ginkgo/core/matrix/csr.hpp>`

Public Member Functions

- strategy_type (std::string name)
  Creates a strategy_type.
- std::string get_name ()
  Returns the name of strategy.
- virtual void process (const Array< index_type > &mtx_row_ptrs, Array< index_type > *mtx_srow)=0
  Computes srow according to row pointers.
- virtual int64_t clac_size (const int64_t nnz)=0
  Computes the srow size according to the number of nonzeros.
- virtual std::shared_ptr< strategy_type > copy ()=0
  Copy a strategy.

41.172.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Csr< ValueType, IndexType >::strategy_type

strategy_type is to decide how to set the csr algorithm.

The practical strategy method should inherit strategy_type and implement its `process`, `clac_size` function and the corresponding device kernel.

41.172.2 Constructor & Destructor Documentation

41.172.2.1 strategy_type()

```
template<typename ValueType = default_precision, typename IndexType = int32>
gko::matrix::Csr< ValueType, IndexType >::strategy_type::strategy_type (
            std::string name )  [inline]
```

Creates a strategy_type.

Parameters

name	the name of strategy

41.172.3 Member Function Documentation

41.172.3.1 clac_size()

```
template<typename ValueType = default_precision, typename IndexType = int32>
virtual int64_t gko::matrix::Csr< ValueType, IndexType >::strategy_type::clac_size (
            const int64_t nnz ) [pure virtual]
```

Computes the srow size according to the number of nonzeros.

Parameters

nnz	the number of nonzeros

Returns

the size of srow

Implemented in gko::matrix::Csr< ValueType, IndexType >::load_balance, gko::matrix::Csr< ValueType, IndexType >::sparselib, gko::matrix::Csr< ValueType, IndexType >::cusparse, gko::matrix::Csr< ValueType, IndexType >::merge_path, and gko::matrix::Csr< ValueType, IndexType >::classical.

41.172.3.2 copy()

```
template<typename ValueType = default_precision, typename IndexType = int32>
virtual std::shared_ptr<strategy_type> gko::matrix::Csr< ValueType, IndexType >::strategy_←
type::copy ( ) [pure virtual]
```

Copy a strategy.

This is a workaround until strategies are revamped, since strategies like `automatical` do not work when actually shared.

Implemented in gko::matrix::Csr< ValueType, IndexType >::load_balance, gko::matrix::Csr< ValueType, IndexType >::sparselib, gko::matrix::Csr< ValueType, IndexType >::cusparse, gko::matrix::Csr< ValueType, IndexType >::merge_path, and gko::matrix::Csr< ValueType, IndexType >::classical.

41.172.3.3 get_name()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::string gko::matrix::Csr< ValueType, IndexType >::strategy_type::get_name ( ) [inline]
```

Returns the name of strategy.

Returns

the name of strategy

41.172.3.4 process()

```
template<typename ValueType = default_precision, typename IndexType = int32>
virtual void gko::matrix::Csr< ValueType, IndexType >::strategy_type::process (
            const Array< index_type > & mtx_row_ptrs,
            Array< index_type > * mtx_srow ) [pure virtual]
```

Computes srow according to row pointers.

Parameters

mtx_row_ptrs	the row pointers of the matrix
mtx_srow	the srow of the matrix

Implemented in gko::matrix::Csr< ValueType, IndexType >::load_balance, gko::matrix::Csr< ValueType, IndexType >::sparselib, gko::matrix::Csr< ValueType, IndexType >::cusparse, gko::matrix::Csr< ValueType, IndexType >::merge_path, and gko::matrix::Csr< ValueType, IndexType >::classical.

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

- ginkgo/core/matrix/csr.hpp

41.173 gko::matrix::Hybrid< ValueType, IndexType >::strategy_type Class Reference

strategy_type is to decide how to set the hybrid config.

```
#include <ginkgo/core/matrix/hybrid.hpp>
```

Public Member Functions

- strategy_type ()

    Creates a strategy_type.
- void compute_hybrid_config (const Array< size_type > &row_nnz, size_type *ell_num_stored_elements_↩ per_row, size_type *coo_nnz)

Computes the config of the Hybrid matrix (ell_num_stored_elements_per_row and coo_nnz).
- size_type get_ell_num_stored_elements_per_row () const noexcept

  Returns the number of stored elements per row of the ell part.
- size_type get_coo_nnz () const noexcept

  Returns the number of nonzeros of the coo part.
- virtual size_type compute_ell_num_stored_elements_per_row (Array< size_type > *row_nnz) const =0

  Computes the number of stored elements per row of the ell part.

41.173.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::matrix::Hybrid< ValueType, IndexType >::strategy_type

strategy_type is to decide how to set the hybrid config.

It computes the number of stored elements per row of the ell part and then set the number of residual nonzeros as the number of nonzeros of the coo part.

The practical strategy method should inherit strategy_type and implement its `compute_ell_num_stored_elements_per_row` function.

41.173.2 Member Function Documentation

41.173.2.1 compute_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
virtual size_type gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::compute_ell_num_stored_elements_per_row (
            Array< size_type > * row_nnz ) const  [pure virtual]
```

Computes the number of stored elements per row of the ell part.

Parameters

row_nnz	the number of nonzeros of each row

Returns

the number of stored elements per row of the ell part

Implemented in gko::matrix::Hybrid< ValueType, IndexType >::automatic, gko::matrix::Hybrid< ValueType, IndexType >::minimal_sto
gko::matrix::Hybrid< ValueType, IndexType >::imbalance_bounded_limit, gko::matrix::Hybrid< ValueType, IndexType >::imbalance_
and gko::matrix::Hybrid< ValueType, IndexType >::column_limit.

Referenced by gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::compute_hybrid_config().

41.173.2.2 compute_hybrid_config()

```
template<typename ValueType = default_precision, typename IndexType = int32>
void gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::compute_hybrid_config (
            const Array< size_type > & row_nnz,
            size_type * ell_num_stored_elements_per_row,
            size_type * coo_nnz ) [inline]
```

Computes the config of the Hybrid matrix (ell_num_stored_elements_per_row and coo_nnz).

For now, it copies row_nnz to the reference executor and performs all operations on the reference executor.

Parameters

row_nnz	the number of nonzeros of each row
ell_num_stored_elements_per_row	the output number of stored elements per row of the ell part
coo_nnz	the output number of nonzeros of the coo part

References gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::compute_ell_num_stored_elements_↩ per_row(), gko::Array< ValueType >::get_executor(), and gko::Array< ValueType >::get_num_elems().

41.173.2.3 get_coo_nnz()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::get_coo_nnz ( ) const
[inline], [noexcept]
```

Returns the number of nonzeros of the coo part.

Returns

the number of nonzeros of the coo part

41.173.2.4 get_ell_num_stored_elements_per_row()

```
template<typename ValueType = default_precision, typename IndexType = int32>
size_type gko::matrix::Hybrid< ValueType, IndexType >::strategy_type::get_ell_num_stored_↩
elements_per_row ( ) const [inline], [noexcept]
```

Returns the number of stored elements per row of the ell part.

Returns

the number of stored elements per row of the ell part

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

- ginkgo/core/matrix/hybrid.hpp

41.174 gko::log::Stream< ValueType > Class Template Reference

Stream is a Logger which logs every event to a stream.

```
#include <ginkgo/core/log/stream.hpp>
```

Static Public Member Functions

- static std::unique_ptr< Stream > create (std::shared_ptr< const Executor > exec, const Logger::mask_type &enabled_events=Logger::all_events_mask, std::ostream &os=std::cout, bool verbose=false)

    Creates a Stream logger.

41.174.1 Detailed Description

template<typename ValueType = default_precision>
class gko::log::Stream< ValueType >

Stream is a Logger which logs every event to a stream.

This can typically be used to log to a file or to the console.

Template Parameters

ValueType	the type of values stored in the class (i.e. ValueType template parameter of the concrete Loggable this class will log)

41.174.2 Member Function Documentation

41.174.2.1 create()

```
template<typename ValueType = default_precision>
static std::unique_ptr<Stream> gko::log::Stream< ValueType >::create (
            std::shared_ptr< const Executor > exec,
            const Logger::mask_type & enabled_events = Logger::all_events_mask,
            std::ostream & os = std::cout,
            bool verbose = false ) [inline], [static]
```

Creates a Stream logger.

This dynamically allocates the memory, constructs the object and returns an std::unique_ptr to this object.

Parameters

exec	the executor
enabled_events	the events enabled for this logger. By default all events.
os	the stream used for this logger
verbose	whether we want detailed information or not. This includes always printing residuals and other information which can give a large output.

Returns

an std::unique_ptr to the the constructed object

```
{
    return std::unique_ptr<Stream>(
        new Stream(exec, enabled_events, os, verbose));
}
```

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

- ginkgo/core/log/stream.hpp

41.175 gko::StreamError Class Reference

StreamError is thrown if accessing a stream failed.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- StreamError (const std::string &file, int line, const std::string &func, const std::string &message)

  Initializes a file access error.

41.175.1 Detailed Description

StreamError is thrown if accessing a stream failed.

41.175.2 Constructor & Destructor Documentation

41.175.2.1 StreamError()

```
gko::StreamError::StreamError (
            const std::string & file,
            int line,
            const std::string & func,
            const std::string & message )  [inline]
```

Initializes a file access 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 function that tried to access the file
message	The error message

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

- ginkgo/core/base/exception.hpp

41.176 gko::stop::Time Class Reference

The Time class is a stopping criterion which stops the iteration process after a certain amout of time has passed.

`#include <ginkgo/core/stop/time.hpp>`

41.176.1 Detailed Description

The Time class is a stopping criterion which stops the iteration process after a certain amout of time has passed.

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

- ginkgo/core/stop/time.hpp

41.177 gko::Transposable Class Reference

Linear operators which support transposition should implement the Transposable interface.

`#include <ginkgo/core/base/lin_op.hpp>`

Public Member Functions

- virtual std::unique_ptr< LinOp > transpose () const =0

  Returns a LinOp representing the transpose of the Transposable object.
- virtual std::unique_ptr< LinOp > conj_transpose () const =0

  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.177.1 Detailed Description

Linear operators which support transposition should implement the Transposable interface.

It provides two functionalities, the normal transpose and the conjugate transpose.

The normal transpose returns the transpose of the linear operator without changing any of its elements representing the operation, $B = A^T$.

The conjugate transpose returns the conjugate of each of the elements and additionally transposes the linear operator representing the operation, $B = A^H$.

41.177.1.1 Example: Transposing a Csr matrix:

```
{c++}
//Transposing an object of LinOp type.
//The object you want to transpose.
auto op = matrix::Csr::create(exec);
//Transpose the object by first converting it to a transposable type.
auto trans = op->transpose();
```

41.177.2 Member Function Documentation

41.177.2.1 conj_transpose()

```
virtual std::unique_ptr<LinOp> gko::Transposable::conj_transpose ( ) const [pure virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Fft3, gko::preconditioner::Jacobi< ValueType, IndexType >, gko::matrix::Dense< ValueType >, gko::matrix::Fbcsr< ValueType, IndexType >, gko::preconditioner::Isai< IsaiType, ValueType, Ind gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >, gko::matrix::Fft2, gko::preconditioner::Ic< LSolver gko::solver::Ir< ValueType >, gko::matrix::SparsityCsr< ValueType, IndexType >, gko::solver::LowerTrs< ValueType, IndexType >, gko::solver::UpperTrs< ValueType, IndexType >, gko::solver::Idr< ValueType >, gko::matrix::Diagonal< ValueType >, gko::solver::Bicg< ValueType >, gko::solver::Fcg< ValueType >, gko::solver::Bicgstab< ValueType >, gko::solver::Cg< ValueType gko::solver::Gmres< ValueType >, gko::matrix::Fft, gko::solver::Cgs< ValueType >, gko::Combination< ValueType >, gko::Composition< ValueType >, and gko::matrix::Identity< ValueType >.

41.177.2.2 transpose()

```
virtual std::unique_ptr<LinOp> gko::Transposable::transpose ( ) const [pure virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implemented in gko::matrix::Csr< ValueType, IndexType >, gko::matrix::Fft3, gko::preconditioner::Jacobi< ValueType, IndexType >, gko::matrix::Dense< ValueType >, gko::matrix::Fbcsr< ValueType, IndexType >, gko::preconditioner::Isai< IsaiType, ValueType, Ind gko::matrix::Fft2, gko::preconditioner::Ilu< LSolverType, USolverType, ReverseApply, IndexType >, gko::preconditioner::Ic< LSolver gko::solver::Ir< ValueType >, gko::matrix::SparsityCsr< ValueType, IndexType >, gko::solver::LowerTrs< ValueType, IndexType >, gko::solver::UpperTrs< ValueType, IndexType >, gko::solver::Idr< ValueType >, gko::matrix::Diagonal< ValueType >, gko::solver::Bicg< ValueType >, gko::solver::Fcg< ValueType >, gko::solver::Bicgstab< ValueType >, gko::solver::Cg< ValueType gko::solver::Gmres< ValueType >, gko::matrix::Fft, gko::solver::Cgs< ValueType >, gko::Combination< ValueType >, gko::Composition< ValueType >, and gko::matrix::Identity< ValueType >.

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

- ginkgo/core/base/lin_op.hpp

41.178 gko::syn::type_list< Types > Struct Template Reference

type_list records several types in template

```
#include <ginkgo/core/synthesizer/containers.hpp>
```

41.178.1 Detailed Description

template<typename... Types>
struct gko::syn::type_list< Types >

type_list records several types in template

Template Parameters

Types	the types in the list

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

- ginkgo/core/synthesizer/containers.hpp

41.179 gko::UnsupportedMatrixProperty Class Reference

Exception throws if a matrix does not have a property required by a numerical method.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- UnsupportedMatrixProperty (const std::string &file, const int line, const std::string &msg)
  Initializes the UnsupportedMatrixProperty error.

41.179.1 Detailed Description

Exception throws if a matrix does not have a property required by a numerical method.

Currently, a message is specified at the call-site manually.

41.179.2 Constructor & Destructor Documentation

41.179.2.1 UnsupportedMatrixProperty()

```
gko::UnsupportedMatrixProperty::UnsupportedMatrixProperty (
            const std::string & file,
            const int line,
            const std::string & msg )  [inline]
```

Initializes the UnsupportedMatrixProperty error.

Parameters

file	The name of the offending source file
line	The source code line number where the error occurred
msg	A message describing the property required.

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

- ginkgo/core/base/exception.hpp

41.180 gko::stop::Criterion::Updater Class Reference

The Updater class serves for convenient argument passing to the Criterion's check function.

```
#include <ginkgo/core/stop/criterion.hpp>
```

Public Member Functions

- Updater (const Updater &)=delete

  Prevent copying and moving the object This is to enforce the use of argument passing and calling check at the same time.
- bool check (uint8 stopping_id, bool set_finalized, Array< stopping_status > *stop_status, bool *one_↩ changed) const

  Calls the parent Criterion object's check method.

41.180.1 Detailed Description

The Updater class serves for convenient argument passing to the Criterion's check function.

The pattern used is a Builder, except Updater builds a function's arguments before calling the function itself, and does not build an object. This allows calling a Criterion's check in the form of: stop_criterion->update() .num↩ _iterations(num_iterations) .residual_norm(residual_norm) .implicit_sq_residual_norm(implicit_sq_residual_norm) .residual(residual) .solution(solution) .check(converged);

If there is a need for a new form of data to pass to the Criterion, it should be added here.

41.180.2 Member Function Documentation

41.180.2.1 check()

```
bool gko::stop::Criterion::Updater::check (
            uint8 stopping_id,
            bool set_finalized,
            Array< stopping_status > * stop_status,
            bool * one_changed ) const  [inline]
```

Calls the parent Criterion object's check method.

References gko::stop::Criterion::check().

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

- ginkgo/core/stop/criterion.hpp

41.181 gko::solver::UpperTrs< ValueType, IndexType > Class Template Reference

UpperTrs is the triangular solver which solves the system U x = b, when U is an upper triangular matrix.

`#include <ginkgo/core/solver/upper_trs.hpp>`

Public Member Functions

- std::shared_ptr< const matrix::Csr< ValueType, IndexType > > get_system_matrix () const

  Gets the system operator (CSR matrix) of the linear system.
- std::unique_ptr< LinOp > transpose () const override

  Returns a LinOp representing the transpose of the Transposable object.
- std::unique_ptr< LinOp > conj_transpose () const override

  Returns a LinOp representing the conjugate transpose of the Transposable object.

41.181.1 Detailed Description

template<typename ValueType = default_precision, typename IndexType = int32>
class gko::solver::UpperTrs< ValueType, IndexType >

UpperTrs is the triangular solver which solves the system U x = b, when U is an upper triangular matrix.

It works best when passing in a matrix in CSR format. If the matrix is not in CSR, then the generate step converts it into a CSR matrix. The generation fails if the matrix is not convertible to CSR.

Note

As the constructor uses the copy and convert functionality, it is not possible to create a empty solver or a solver with a matrix in any other format other than CSR, if none of the executor modules are being compiled with.

Template Parameters

ValueType	precision of matrix elements
IndexType	precision of matrix indices

41.181.2 Member Function Documentation

41.181.2.1 conj_transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::solver::UpperTrs< ValueType, IndexType >::conj_transpose ( )
const [override], [virtual]
```

Returns a LinOp representing the conjugate transpose of the Transposable object.

Returns

a pointer to the new conjugate transposed object

Implements gko::Transposable.

41.181.2.2 get_system_matrix()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::shared_ptr<const matrix::Csr<ValueType, IndexType> > gko::solver::UpperTrs< ValueType,
IndexType >::get_system_matrix ( ) const [inline]
```

Gets the system operator (CSR matrix) of the linear system.

Returns

the system operator (CSR matrix)

```
{
    return system_matrix_;
}
```

41.181.2.3 transpose()

```
template<typename ValueType = default_precision, typename IndexType = int32>
std::unique_ptr<LinOp> gko::solver::UpperTrs< ValueType, IndexType >::transpose ( ) const
[override], [virtual]
```

Returns a LinOp representing the transpose of the Transposable object.

Returns

a pointer to the new transposed object

Implements gko::Transposable.

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

- ginkgo/core/solver/lower_trs.hpp
- ginkgo/core/solver/upper_trs.hpp

41.182 gko::UseComposition< ValueType > Class Template Reference

The UseComposition class can be used to store the composition information in LinOp.

```
#include <ginkgo/core/base/composition.hpp>
```

Public Member Functions

- std::shared_ptr< Composition< ValueType > > get_composition () const
  Returns the composition operators.
- std::shared_ptr< const LinOp > get_operator_at (size_type index) const
  Returns the operator at index-th poistion of composition.

41.182.1 Detailed Description

template<typename ValueType = default_precision>
class gko::UseComposition< ValueType >

The UseComposition class can be used to store the composition information in LinOp.

Template Parameters

ValueType	precision of input and result vectors

41.182.2 Member Function Documentation

41.182.2.1 get_composition()

```
template<typename ValueType = default_precision>
std::shared_ptr<Composition<ValueType> > gko::UseComposition< ValueType >::get_composition (
) const [inline]
```

Returns the composition operators.

Returns

composition

41.182.2.2 get_operator_at()

```
template<typename ValueType = default_precision>
std::shared_ptr<const LinOp> gko::UseComposition< ValueType >::get_operator_at (
            size_type index ) const [inline]
```

Returns the operator at index-th poistion of composition.

Returns

index-th operator

Note

when this composition is not set, this function always returns nullptr. However, when this composition is set, it will throw exception when exceeding index.

Exceptions

std::out_of_range	if index is out of bound when composition is existed.

Referenced by gko::multigrid::EnableMultigridLevel< ValueType >::get_coarse_op(), gko::multigrid::EnableMultigridLevel< ValueType >::get_prolong_op(), and gko::multigrid::EnableMultigridLevel< ValueType >::get_restrict_op().

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

- ginkgo/core/base/composition.hpp

41.183 gko::syn::value_list< T, Values > Struct Template Reference

value_list records several values with the same type in template.

```
#include <ginkgo/core/synthesizer/containers.hpp>
```

41.183.1 Detailed Description

template< typename T, T... Values>
struct gko::syn::value_list< T, Values >

value_list records several values with the same type in template.

Template Parameters

T	the value type of the list
Values	the values in the list

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

- ginkgo/core/synthesizer/containers.hpp

41.184 gko::ValueMismatch Class Reference

ValueMismatch is thrown if two values are not equal.

```
#include <ginkgo/core/base/exception.hpp>
```

Public Member Functions

- ValueMismatch (const std::string &file, int line, const std::string &func, size_type val1, size_type val2, const std::string &clarification)

    Initializes a value mismatch error.

41.184.1 Detailed Description

ValueMismatch is thrown if two values are not equal.

41.184.2 Constructor & Destructor Documentation

41.184.2.1 ValueMismatch()

```
gko::ValueMismatch::ValueMismatch (
            const std::string & file,
            int line,
            const std::string & func,
            size_type val1,
            size_type val2,
            const std::string & clarification )  [inline]
```

Initializes a value mismatch 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
val1	The first value to be compared.
val2	The second value to be compared.
clarification	An additional message further describing the error

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

- ginkgo/core/base/exception.hpp

41.185 gko::version Struct Reference

This structure is used to represent versions of various Ginkgo modules.

```
#include <ginkgo/core/base/version.hpp>
```

Public Attributes

- const uint64 major

  The major version number.
- const uint64 minor

  The minor version number.
- const uint64 patch

  The patch version number.
- const char ∗const tag

  Addition tag string that describes the version in more detail.

41.185.1 Detailed Description

This structure is used to represent versions of various Ginkgo modules.

Version structures can be compared using the usual relational operators.

41.185.2 Member Data Documentation

41.185.2.1 tag

```
const char* const gko::version::tag
```

Addition tag string that describes the version in more detail.

It does not participate in comparisons.

Referenced by gko::operator<<().

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

- ginkgo/core/base/version.hpp

41.186 gko::version_info Class Reference

Ginkgo uses version numbers to label new features and to communicate backward compatibility guarantees:

```
#include <ginkgo/core/base/version.hpp>
```

Static Public Member Functions

- static const version_info & get ()

  Returns an instance of version_info.

Public Attributes

- version header_version

  Contains version information of the header files.
- version core_version

  Contains version information of the core library.
- version reference_version

  Contains version information of the reference module.
- version omp_version

  Contains version information of the OMP module.
- version cuda_version

  Contains version information of the CUDA module.
- version hip_version

  Contains version information of the HIP module.
- version dpcpp_version

  Contains version information of the DPC++ module.

41.186.1 Detailed Description

Ginkgo uses version numbers to label new features and to communicate backward compatibility guarantees:

1. Versions with different major version number have incompatible interfaces (parts of the earlier interface may not be present anymore, and new interfaces can appear).
2. Versions with the same major number X, but different minor numbers Y1 and Y2 numbers keep the same interface as version X.0.0, but additions to the interface in X.0.0 present in X.Y1.0 may not be present in X.Y2.0 and vice versa.
3. Versions with the same major an minor version numbers, but different patch numbers have exactly the same interface, but the functionality may be different (something that is not implemented or has a bug in an earlier version may have this implemented or fixed in a later version).

This structure provides versions of different parts of Ginkgo: the headers, the core and the kernel modules (reference, OpenMP, CUDA, HIP, DPCPP). To obtain an instance of version_info filled with information about the current version of Ginkgo, call the version_info::get() static method.

41.186.2 Member Function Documentation

41.186.2.1 get()

`static const version_info& gko::version_info::get ( )  [inline], [static]`

Returns an instance of version_info.

Returns

an instance of version info

41.186.3 Member Data Documentation

41.186.3.1 core_version

`version gko::version_info::core_version`

Contains version information of the core library.

This is the version of the static/shared library called "ginkgo".

41.186.3.2 cuda_version

```
version gko::version_info::cuda_version
```

Contains version information of the CUDA module.

This is the version of the static/shared library called "ginkgo_cuda".

41.186.3.3 dpcpp_version

```
version gko::version_info::dpcpp_version
```

Contains version information of the DPC++ module.

This is the version of the static/shared library called "ginkgo_dpcpp".

41.186.3.4 hip_version

```
version gko::version_info::hip_version
```

Contains version information of the HIP module.

This is the version of the static/shared library called "ginkgo_hip".

41.186.3.5 omp_version

```
version gko::version_info::omp_version
```

Contains version information of the OMP module.

This is the version of the static/shared library called "ginkgo_omp".

41.186.3.6 reference_version

```
version gko::version_info::reference_version
```

Contains version information of the reference module.

This is the version of the static/shared library called "ginkgo_reference".

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

- ginkgo/core/base/version.hpp

41.187 gko::WritableToMatrixData< ValueType, IndexType > Class Template Reference

A LinOp implementing this interface can write its data to a matrix_data structure.

```
#include <ginkgo/core/base/lin_op.hpp>
```

Public Member Functions

- virtual void write (matrix_data< ValueType, IndexType > &data) const =0

  Writes a matrix to a matrix_data structure.

41.187.1 Detailed Description

template<typename ValueType, typename IndexType>
class gko::WritableToMatrixData< ValueType, IndexType >

A LinOp implementing this interface can write its data to a matrix_data structure.

41.187.2 Member Function Documentation

41.187.2.1 write()

```
template<typename ValueType, typename IndexType>
virtual void gko::WritableToMatrixData< ValueType, IndexType >::write (
            matrix_data< ValueType, IndexType > & data ) const  [pure virtual]
```

Writes a matrix to a matrix_data structure.

Parameters

data	the matrix_data structure

Implemented in gko::matrix::Fft3, gko::matrix::Fft2, gko::matrix::Fft, gko::matrix::Fft3, gko::matrix::Fft2, gko::matrix::Fft, gko::matrix::Fft3, gko::matrix::Fft2, gko::matrix::Fft, gko::matrix::Fft3, gko::matrix::Fft2, gko::matrix::Fft, gko::matrix::Csr< ValueType, I gko::matrix::Hybrid< ValueType, IndexType >, gko::preconditioner::Jacobi< ValueType, IndexType >, gko::matrix::Fbcsr< ValueType gko::matrix::Coo< ValueType, IndexType >, gko::matrix::Ell< ValueType, IndexType >, gko::matrix::SparsityCsr< ValueType, IndexT and gko::matrix::Sellp< ValueType, IndexType >.

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

- ginkgo/core/base/lin_op.hpp

Index