SpFFT Documentation

Release 0.1.0

ETH Zurich, Simon Frasch

Aug 18, 2020

CONTENTS

SpFFT - A 3D FFT library for sparse frequency domain data written in C++ with support for MPI, OpenMP, CUDA and ROCm.

Inspired by the need of some computational material science applications with spherical cutoff data in frequency domain, SpFFT provides Fast Fourier Transformations of sparse frequency domain data. For distributed computations with MPI, slab decomposition in space domain and pencil decomposition in frequency domain (sparse data within a pencil / column must be on one rank) is used.

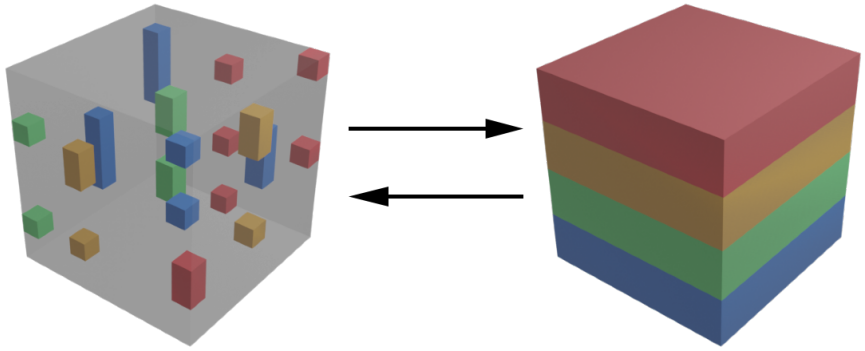

Fig. 1: Illustration of a transform, where data on each MPI rank is identified by color.

CHAPTER

ONE

DESIGN GOALS

- Sparse frequency domain input
- Reuse of pre-allocated memory
- Support of negative indexing for frequency domain data
- Parallelization and acceleration are optional
- Unified interface for calculations on CPUs and GPUs
- Support of Complex-To-Real and Real-To-Complex transforms, where the full hermitian symmetry property is utilized
- C++, C and Fortran interfaces

CHAPTER

TWO

INTERFACE DESIGN

To allow for pre-allocation and reuse of memory, the design is based on two classes:

- **Grid**: Allocates memory for transforms up to a given size in each dimension.
- **Transform**: Is associated with a *Grid* and can have any size up to the *Grid* dimensions. A *Transform* holds a counted reference to the underlying *Grid*. Therefore, *Transforms* created with the same *Grid* share memory, which is only freed, once the *Grid* and all associated *Transforms* are destroyed.

The user provides memory for storing sparse frequency domain data, while a *Transform* provides memory for space domain data. This implies, that executing a *Transform* will override the space domain data of all other *Transforms* associated with the same *Grid*.

Note: The creation of Grids and Transforms, as well as the forward and backward execution may entail MPI calls and must be synchronized between all ranks.

2.1 Installation

2.1.1 Requirements

- C++ Compiler with C++11 support. Supported compilers are:
  - GCC 6 and later
  - Clang 5 and later
  - ICC 18.0 and later
- CMake 3.11 and later
- Library providing a FFTW 3.x interface (FFTW3 or Intel MKL)
- For multi-threading: OpenMP support by the compiler
- For compilation with GPU support:
  - CUDA 9.0 and later for Nvidia hardware
  - ROCm 2.6 and later for AMD hardware

2.1.2 Build

The build system follows the standard CMake workflow. Example:

```
mkdir build
cd build
cmake .. -DSPFFT_OMP=ON -DSPFFT_MPI=ON -DSPFFT_GPU_BACKEND=CUDA -DSPFFT_SINGLE_
↪PRECISION=OFF -DCMAKE_INSTALL_PREFIX=/usr/local
make -j8 install
```

2.1.3 CMake options

Option	Default	Description
SPFFT_MPI	ON	Enable MPI support
SPFFT_OMP	ON	Enable multi-threading with OpenMP
SPFFT_GPU_BACKEND	OFF	Select GPU backend. Can be OFF, CUDA or ROCM
SPFFT_GPU_DIRECT	OFF	Use GPU aware MPI with GPUDirect
SPFFT_SINGLE_PRECISION	OFF	Enable single precision support
SPFFT_STATIC	OFF	Build as static library
SPFFT_BUILD_TESTS	OFF	Build test executables for developement purposes
SPFFT_INSTALL	ON	Add library to install target
SPFFT_FORTRAN	OFF	Build Fortran interface module

2.2 Examples

2.2.1 C++

```
#include <complex>
#include <iostream>
#include <vector>

#include "spfft/spfft.hpp"

int main(int argc, char** argv) {
  const int dimX = 2;
  const int dimY = 2;
  const int dimZ = 2;

  std::cout << "Dimensions: x = " << dimX << ", y = " << dimY << ", z = " << dimZ <<␣
↪std::endl
            << std::endl;

  // Use default OpenMP value
  const int numThreads = -1;

  // use all elements in this example.
  const int numFrequencyElements = dimX * dimY * dimZ;

  // Slice length in space domain. Equivalent to dimZ for non-distributed case.
  const int localZLength = dimZ;
```

(continues on next page)

(continued from previous page)

```
  // interleaved complex numbers
  std::vector<double> frequencyElements;
  frequencyElements.reserve(2 * numFrequencyElements);

  // indices of frequency elements
  std::vector<int> indices;
  indices.reserve(3 * numFrequencyElements);

  // initialize frequency domain values and indices
  double initValue = 0.0;
  for (int xIndex = 0; xIndex < dimX; ++xIndex) {
    for (int yIndex = 0; yIndex < dimY; ++yIndex) {
      for (int zIndex = 0; zIndex < dimZ; ++zIndex) {
        // init with interleaved complex numbers
        frequencyElements.emplace_back(initValue);
        frequencyElements.emplace_back(-initValue);

        // add index triplet for value
        indices.emplace_back(xIndex);
        indices.emplace_back(yIndex);
        indices.emplace_back(zIndex);

        initValue += 1.0;
      }
    }
  }

  std::cout << "Input:" << std::endl;
  for (int i = 0; i < numFrequencyElements; ++i) {
    std::cout << frequencyElements[2 * i] << ", " << frequencyElements[2 * i + 1] <<␣
↪std::endl;
  }

  // create local Grid. For distributed computations, a MPI Communicator has to be␣
↪provided
  spfft::Grid grid(dimX, dimY, dimZ, dimX * dimY, SPFFT_PU_HOST, numThreads);

  // create transform
  spfft::Transform transform =
      grid.create_transform(SPFFT_PU_HOST, SPFFT_TRANS_C2C, dimX, dimY, dimZ,␣
↪localZLength,
                            numFrequencyElements, SPFFT_INDEX_TRIPLETS, indices.
↪data());

  // Get pointer to space domain data. Alignment fullfills requirements for␣
↪std::complex.
  // Can also be read as std::complex elements (guaranteed by the standard to be␣
↪binary compatible
  // since C++11).
  double* spaceDomain = transform.space_domain_data(SPFFT_PU_HOST);

  // transform backward
  transform.backward(frequencyElements.data(), SPFFT_PU_HOST);

  std::cout << std::endl << "After backward transform:" << std::endl;
  for (int i = 0; i < transform.local_slice_size(); ++i) {
    std::cout << spaceDomain[2 * i] << ", " << spaceDomain[2 * i + 1] << std::endl;
```

(continues on next page)

(continued from previous page)

```
  }

  // transform forward
  transform.forward(SPFFT_PU_HOST, frequencyElements.data(), SPFFT_NO_SCALING);

  std::cout << std::endl << "After forward transform (without scaling):" << std::endl;
  for (int i = 0; i < numFrequencyElements; ++i) {
    std::cout << frequencyElements[2 * i] << ", " << frequencyElements[2 * i + 1] <<␣
↪std::endl;
  }

  return 0;
}
```

2.2.2 C

```
#include <stdio.h>
#include <stdlib.h>

#include "spfft/spfft.h"

int main(int argc, char** argv) {
  const int dimX = 2;
  const int dimY = 2;
  const int dimZ = 2;

  printf("Dimensions: x = %d, y = %d, z = %d\n\n", dimX, dimY, dimZ);

  /* Use default OpenMP value */
  const int numThreads = -1;

  /* use all elements in this example. */
  const int numFrequencyElements = dimX * dimY * dimZ;

  /* Slice length in space domain. Equivalent to dimZ for non-distributed case. */
  const int localZLength = dimZ;

  /* interleaved complex numbers */
  double* frequencyElements = (double*)malloc(2 * sizeof(double) *␣
↪numFrequencyElements);

  /* indices of frequency elements */
  int* indices = (int*)malloc(3 * sizeof(int) * numFrequencyElements);

  /* initialize frequency domain values and indices */
  double initValue = 0.0;
  size_t count = 0;
  for (int xIndex = 0; xIndex < dimX; ++xIndex) {
    for (int yIndex = 0; yIndex < dimY; ++yIndex) {
      for (int zIndex = 0; zIndex < dimZ; ++zIndex, ++count) {
        /* init values */
        frequencyElements[2 * count] = initValue;
        frequencyElements[2 * count + 1] = -initValue;

        /* add index triplet for value */
```

(continues on next page)

(continued from previous page)

```
        indices[3 * count] = xIndex;
        indices[3 * count + 1] = yIndex;
        indices[3 * count + 2] = zIndex;

        initValue += 1.0;
      }
    }
  }

  printf("Input:\n");
  for (size_t i = 0; i < dimX * dimY * dimZ; ++i) {
    printf("%f, %f\n", frequencyElements[2 * i], frequencyElements[2 * i + 1]);
  }
  printf("\n");

  SpfftError status = 0;

  /* create local Grid. For distributed computations, a MPI Communicator has to be␣
↪provided */
  SpfftGrid grid;
  status = spfft_grid_create(&grid, dimX, dimY, dimZ, dimX * dimY, SPFFT_PU_HOST,␣
↪numThreads);
  if (status != SPFFT_SUCCESS) exit(status);

  /* create transform */
  SpfftTransform transform;
  status = spfft_transform_create(&transform, grid, SPFFT_PU_HOST, SPFFT_TRANS_C2C,␣
↪dimX, dimY,
                                  dimZ, localZLength, numFrequencyElements, SPFFT_
↪INDEX_TRIPLETS, indices);
  if (status != SPFFT_SUCCESS) exit(status);

  /* grid can be safely destroyed after creating all transforms */
  status = spfft_grid_destroy(grid);
  if (status != SPFFT_SUCCESS) exit(status);

  /* get pointer to space domain data. Alignment is guaranteed to fullfill␣
↪requirements C complex
   types */
  double* spaceDomain;
  status = spfft_transform_get_space_domain(transform, SPFFT_PU_HOST, &spaceDomain);
  if (status != SPFFT_SUCCESS) exit(status);

  /* transform backward */
  status = spfft_transform_backward(transform, frequencyElements, SPFFT_PU_HOST);
  if (status != SPFFT_SUCCESS) exit(status);

  printf("After backward transform:\n");
  for (size_t i = 0; i < dimX * dimY * dimZ; ++i) {
    printf("%f, %f\n", spaceDomain[2 * i], spaceDomain[2 * i + 1]);
  }
  printf("\n");

  /* transform forward */
  status = spfft_transform_forward(transform, SPFFT_PU_HOST, frequencyElements, SPFFT_
↪NO_SCALING);
  if (status != SPFFT_SUCCESS) exit(status);
```

(continues on next page)

(continued from previous page)

```
  printf("After forward transform (without scaling):\n");
  for (size_t i = 0; i < dimX * dimY * dimZ; ++i) {
    printf("%f, %f\n", frequencyElements[2 * i], frequencyElements[2 * i + 1]);
  }

  /* destroying the final transform will free the associated memory */
  status = spfft_transform_destroy(transform);
  if (status != SPFFT_SUCCESS) exit(status);

  return 0;
}
```

2.2.3 Fortran

```
program main
    use iso_c_binding
    use spfft
    implicit none
    integer :: i, j, k, counter
    integer, parameter :: dimX = 2
    integer, parameter :: dimY = 2
    integer, parameter :: dimZ = 2
    integer, parameter :: maxNumLocalZColumns = dimX * dimY
    integer, parameter :: processingUnit = 1
    integer, parameter :: maxNumThreads = -1
    type(c_ptr) :: grid = c_null_ptr
    type(c_ptr) :: transform = c_null_ptr
    integer :: errorCode = 0
    integer, dimension(dimX * dimY * dimZ * 3):: indices = 0
    complex(C_DOUBLE_COMPLEX), dimension(dimX * dimY * dimZ):: frequencyElements
    complex(C_DOUBLE_COMPLEX), pointer :: spaceDomain(:,:,:)
    type(c_ptr) :: realValuesPtr


    counter = 0
    do k = 1, dimZ
        do j = 1, dimY
            do i = 1, dimX
             frequencyElements(counter + 1) = cmplx(counter, -counter)
             indices(counter * 3 + 1) = i - 1
             indices(counter * 3 + 2) = j - 1
             indices(counter * 3 + 3) = k - 1
             counter = counter + 1
            end do
        end do
    end do

    ! print input
    print *, "Input:"
    do i = 1, size(frequencyElements)
         print *, frequencyElements(i)
    end do
```

(continues on next page)

(continued from previous page)

```
    ! create grid and transform
    errorCode = spfft_grid_create(grid, dimX, dimY, dimZ, maxNumLocalZColumns,␣
↪processingUnit, maxNumThreads);
    if (errorCode /= SPFFT_SUCCESS) error stop
    errorCode = spfft_transform_create(transform, grid, processingUnit, 0, dimX, dimY,
↪ dimZ, dimZ,&
         size(frequencyElements), SPFFT_INDEX_TRIPLETS, indices)
    if (errorCode /= SPFFT_SUCCESS) error stop

    ! grid can be safely destroyed after creating all required transforms
    errorCode = spfft_grid_destroy(grid)
    if (errorCode /= SPFFT_SUCCESS) error stop

    ! set space domain array to use memory allocted by the library
    errorCode = spfft_transform_get_space_domain(transform, processingUnit,␣
↪realValuesPtr)
    if (errorCode /= SPFFT_SUCCESS) error stop

    ! transform backward
    errorCode = spfft_transform_backward(transform, frequencyElements, processingUnit)
    if (errorCode /= SPFFT_SUCCESS) error stop


    call c_f_pointer(realValuesPtr, spaceDomain, [dimX,dimY,dimZ])

    print *, ""
    print *, "After backward transform:"
    do k = 1, size(spaceDomain, 3)
        do j = 1, size(spaceDomain, 2)
            do i = 1, size(spaceDomain, 1)
             print *, spaceDomain(i, j, k)
            end do
        end do
    end do

    ! transform forward (will invalidate space domain data)
    errorCode = spfft_transform_forward(transform, processingUnit, frequencyElements,␣
↪0)
    if (errorCode /= SPFFT_SUCCESS) error stop

    print *, ""
    print *, "After forward transform (without scaling):"
    do i = 1, size(frequencyElements)
             print *, frequencyElements(i)
    end do

    ! destroying the final transform will free the associated memory
    errorCode = spfft_transform_destroy(transform)
    if (errorCode /= SPFFT_SUCCESS) error stop

end
```

2.3 Details

2.3.1 Transform Definition

Given a discrete function f, SpFFT computes the Discrete Fourier Transform:

$$z_{k_x,k_y,k_z} = \sum_{n_x=0}^{N_x-1} \omega_{N_x}^{k_x,n_x} \sum_{n_y=0}^{N_y-1} \omega_{N_y}^{k_y,n_y} \sum_{n_z=0}^{N_z-1} \omega_{N_z}^{k_z,n_z} f_{n_x,n_y,n_z}$$

where ω is defined as:

- $\omega_N^{k,n} = e^{-2\pi i \frac{kn}{N}}$: *Forward* transform from space domain to frequency domain
- $\omega_N^{k,n} = e^{2\pi i \frac{kn}{N}}$: *Backward* transform from frequency domain to space domain

2.3.2 Complex Number Format

SpFFT always assumes an interleaved format in double or single precision. The alignment of memory provided for space domain data is guaranteed to fulfill to the requirements for std::complex (for C++11), C complex types and GPU complex types of CUDA or ROCm.

2.3.3 Indexing

The three dimensions are referred to as x, y and z. An element in space domain is addressed in memory as:

$$(z \cdot N_y + y) \cdot N_x + x$$

For now, the only supported format for providing the indices of sparse frequency domain data are index triplets in an interleaved array.
Example: $x_1, y_1, z_1, x_2, y_2, z_2, ...$

Indices for a dimension of size *n* must be either in the interval $[0, n-1]$ or $\left[\left\lfloor \frac{n}{2} \right\rfloor - n + 1, \left\lfloor \frac{n}{2} \right\rfloor\right]$. For Real-To-Complex transforms additional restrictions apply (see next section).

2.3.4 Real-To-Complex Transforms

The Discrete Fourier Transform $f(x, y, z)$ of a real valued function is hermitian:

$$f(x, y, z) = f^*(-x, -y, -z)$$

Due to this property, only about half the frequency domain data is required without loss of information. Therefore, similar to other FFT libraries, all indices in x *must* be in the interval $\left[0, \left\lfloor \frac{n}{2} \right\rfloor\right]$. To fully utlize the symmetry property, the following steps can be followed:

- Only non-redundent z-coloumns on the y-z plane at $x = 0$ have to be provided. A z-coloumn must be complete and can be provided at either y or $-y$.
- All redundant values in the z-coloumn at $x = 0$, $y = 0$ can be omitted.

2.3.5 Normalization

Normalization is only available for the forward transform with a scaling factor of $\frac{1}{N_x N_y N_z}$. Applying a forward and backwards transform with scaling enabled will therefore yield identical output (within numerical accuracy).

2.3.6 Optimal sizing

The underlying computation is done by FFT libraries such as FFTW and cuFFT, which provide optimized implementations for sizes, which are of the form $2^a 3^b 5^c 7^d$ where a, b, c, d are natural numbers. Typically, smaller prime factors perform better. The size of each dimension is ideally set accordingly.

2.3.7 Data Distribution

SpFFT uses slab decomposition in space domain, where slabs are ideally uniform in size between MPI ranks.
In frequency domain, SpFFT uses a pencil decomposition, where elements within a z-coloumn (same x-y index) *must* be on the same MPI rank. The order and distribution of frequency space elements can have significant impact on performance. Locally, elements are best grouped by z-columns and ordered by their z-index within each column. The ideal distribution of z-columns between MPI ranks differs for execution on host and GPU.

For execution on host:
: Indices of z-columns are ideally continuous in y on each MPI rank.

For execution on GPU:
: Indices of z-columns are ideally continuous in x on each MPI rank.

2.3.8 MPI Exchange

The MPI exchange is based on a collective MPI call. The following options are available:

SPFFT_EXCH_BUFFERED Exchange with MPI_Alltoall. Requires repacking of data into buffer. Possibly best optimized for large number of ranks by MPI implementations, but does not adjust well to non-uniform data distributions.

SPFFT_EXCH_COMPACT_BUFFERED Exchange with MPI_Alltoallv. Requires repacking of data into buffer. Performance is usually close to MPI_alltoall and it adapts well to non-uniform data distributions.

SPFFT_EXCH_UNBUFFERED Exchange with MPI_Alltoallw. Does not require repacking of data into buffer (outside of the MPI library). Performance varies widely between systems and MPI implementations. It is generally difficult to optimize for large number of ranks, but may perform best in certain conditions.

For both *SPFFT_EXCH_BUFFERED* and *SPFFT_EXCH_COMPACT_BUFFERED*, an exchange in single precision can be selected. With transforms in double precision, the number of bytes sent and received is halved. For execution on GPUs without GPUDirect, the data transfer between GPU and host also benefits. This option can provide a

significant speedup, but incurs a slight accuracy loss. The double precision values are converted to and from single precision between the transform in z and the transform in x / y, while all actual calculations are still done in the selected precision.

2.3.9 Thread-Safety

The creation of Grid and Transform objects is thread-safe only if:

- No FFTW library calls are executed concurrently.
- In the distributed case, MPI thread support is set to *MPI_THREAD_MULTIPLE*.

The execution of transforms is thread-safe if

- Each thread executes using its own Grid and associated Transform object.
- In the distributed case, MPI thread support is set to *MPI_THREAD_MULTIPLE*.

2.3.10 GPU

Saving transfer time between host and GPU is key to good performance for execution with GPUs. Ideally, both input and output is located on GPU memory. If host memory pointers are provided as input or output, it is helpful to use pinned memory through the CUDA or ROCm API.

If available, GPU aware MPI can be utilized, to safe on the otherwise required transfers between host and GPU in preparation of the MPI exchange. This can greatly impact performance and is enabled by compiling the library with the CMake option SPFFT_GPU_DIRECT set to ON.

Note: Additional environment variables may have to be set for some MPI implementations, to allow GPUDirect usage.

Note: The execution of a transform is synchronized with the default stream.

2.4 Types

Enums

enum SpfftExchangeType

Values:

enumerator SPFFT_EXCH_DEFAULT

Default exchange.

Equivalent to SPFFT_EXCH_COMPACT_BUFFERED.

enumerator SPFFT_EXCH_BUFFERED

Exchange based on MPI_Alltoall.

enumerator SPFFT_EXCH_BUFFERED_FLOAT
Exchange based on MPI_Alltoall in single precision.

Slight accuracy loss for double precision transforms due to conversion to float prior to MPI exchange.

enumerator SPFFT_EXCH_COMPACT_BUFFERED
Exchange based on MPI_Alltoallv.

enumerator SPFFT_EXCH_COMPACT_BUFFERED_FLOAT
Exchange based on MPI_Alltoallv in single precision.

Slight accuracy loss for double precision transforms due to conversion to float prior to MPI exchange.

enumerator SPFFT_EXCH_UNBUFFERED
Exchange based on MPI_Alltoallw.

enum SpfftProcessingUnitType
Processing unit type.

Values:

enumerator SPFFT_PU_HOST = 1
HOST.

enumerator SPFFT_PU_GPU = 2
GPU.

enum SpfftIndexFormatType
Values:

enumerator SPFFT_INDEX_TRIPLETS
Triplets of x,y,z frequency indices.

enum SpfftTransformType
Values:

enumerator SPFFT_TRANS_C2C
Complex-to-Complex transform.

enumerator SPFFT_TRANS_R2C
Real-to-Complex transform.

enum SpfftScalingType
Values:

enumerator SPFFT_NO_SCALING
No scaling.

enumerator SPFFT_FULL_SCALING
Full scaling.

2.5 Grid

Note: A Grid object can be safely destroyed after Transform objects have been created, since internal reference counting used to prevent the release of resources while still in use.

class *spfft*::**Grid**
A *Grid*, which provides pre-allocated memory for double precision transforms.

Public Functions

Grid(int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, *SpfftProcessingUnitType processingUnit*, int *maxNumThreads*)
Constructor for a local grid.

Parameters

- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

Grid(int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, int *maxLocalZLength*, *SpfftProcessingUnitType processingUnit*, int *maxNumThreads*, MPI_Comm *comm*, *SpfftExchangeType exchangeType*)
Constructor for a distributed grid.

Thread-safe if MPI thread support is set to MPI_THREAD_MULTIPLE.

Parameters

- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain of the local MPI rank.
- `[in] maxLocalZLength`: Maximum length in z in space domain for the local MPI rank.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.
- `[in] comm`: The MPI communicator to use. Will be duplicated for internal use.
- `[in] exchangeType`: The type of MPI exchange to use. Possible values are SPFFT_EXCH_DEFAULT, SPFFT_EXCH_BUFFERED, SPFFT_EXCH_COMPACT_BUFFERED and SPFFT_EXCH_UNBUFFERED.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

Grid(**const** *Grid*&)
Custom copy constructor.

Creates a independent copy. Calls MPI functions for the distributed case.

Grid(*Grid*&&) = default
Default move constructor.

Grid &**operator=**(**const** *Grid*&)
Custom copy operator.

Creates a independent copy. Calls MPI functions for the distributed case.

Grid &**operator=**(*Grid*&&) = default
Default move operator.

Transform **create_transform**(*SpfftProcessingUnitType processingUnit*, *SpfftTransformType transformType*, int *dimX*, int *dimY*, int *dimZ*, int *localZLength*, int *numLocalElements*, *SpfftIndexFormatType indexFormat*, **const** int **indices*) **const**
Creates a transform from this grid object.

Thread-safe if no FFTW calls are executed concurrently.

Return *Transform*

Parameters

- `[in] processingUnit`: The processing unit type to use. Must be either SPFFT_PU_HOST or SPFFT_PU_GPU and be supported by the grid itself.
- `[in] transformType`: The transform type (complex to complex or real to complex). Can be SPFFT_TRANS_C2C or SPFFT_TRANS_R2C.
- `[in] dimX`: The dimension in x. The maximum allowed depends on the grid parameters.
- `[in] dimY`: The dimension in y. The maximum allowed depends on the grid parameters.
- `[in] dimZ`: The dimension in z. The maximum allowed depends on the grid parameters.
- `[in] localZLength`: The length in z in space domain of the local MPI rank.
- `[in] numLocalElements`: The number of elements in frequency domain of the local MPI rank.
- `[in] indexFormat`: The index format. Only SPFFT_INDEX_TRIPLETS currently supported.
- `[in] indices`: Pointer to the frequency indices. Posive and negative indexing is supported.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

int **max_dim_x**() **const**
Access a grid parameter.

Return Maximum dimension in x.

int **max_dim_y**() **const**
Access a grid parameter.

Return Maximum dimension in y.

int **max_dim_z**() **const**
Access a grid parameter.

Return Maximum dimension in z.

int **max_num_local_z_columns**() **const**
Access a grid parameter.

Return Maximum number of z-columns in frequency domain of the local MPI rank.

int **max_local_z_length**() **const**
Access a grid parameter.

Return Maximum length in z in space domain of the local MPI rank.

SpfftProcessingUnitType **processing_unit**() **const**
Access a grid parameter.

Return The processing unit, the grid has prepared for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.

int **device_id**() **const**
Access a grid parameter.

Return The GPU device id used. Always returns 0, if no GPU support is enabled.

int **num_threads**() **const**
Access a grid parameter.

Return The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

MPI_Comm **communicator**() **const**
Access a grid parameter.

Return The internal MPI communicator.

2.6 GridFloat

Note: This class is only available if single precision support is enabled, in which case the marco SPFFT_SINGLE_PRECISION is defined in config.h.

Note: A Grid object can be safely destroyed after Transform objects have been created, since internal reference counting used to prevent the release of resources while still in use.

class *spfft*::**GridFloat**
A *Grid*, which provides pre-allocated memory for single precision transforms.

Public Functions

GridFloat(int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, *SpfftProcessingUnitType* *processingUnit*, int *maxNumThreads*)

Constructor for a local grid.

Parameters

- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

GridFloat(int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, int *maxLocalZLength*, *SpfftProcessingUnitType* *processingUnit*, int *maxNumThreads*, MPI_Comm *comm*, *SpfftExchangeType* *exchangeType*)

Constructor for a distributed grid.

Thread-safe if MPI thread support is set to MPI_THREAD_MULTIPLE.

Parameters

- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain of the local MPI rank.
- `[in] maxLocalZLength`: Maximum length in z in space domain for the local MPI rank.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.
- `[in] comm`: The MPI communicator to use. Will be duplicated for internal use.
- `[in] exchangeType`: The type of MPI exchange to use. Possible values are SPFFT_EXCH_DEFAULT, SPFFT_EXCH_BUFFERED, SPFFT_EXCH_COMPACT_BUFFERED and SPFFT_EXCH_UNBUFFERED.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

GridFloat(**const** *GridFloat*&)
Custom copy constructor.

Creates a independent copy. Calls MPI functions for the distributed case.

GridFloat(*GridFloat*&&) = default
Default move constructor.

GridFloat &**operator=**(**const** *GridFloat*&)
Custom copy operator.

Creates a independent copy. Calls MPI functions for the distributed case.

GridFloat &**operator=**(*GridFloat*&&) = default
Default move operator.

TransformFloat **create_transform**(*SpfftProcessingUnitType* *processingUnit*, *SpfftTransformType* *transformType*, int *dimX*, int *dimY*, int *dimZ*, int *localZLength*, int *numLocalElements*, *SpfftIndexFormatType* *indexFormat*, **const** int **indices*) **const**
Creates a transform from this grid object.

Thread-safe if no FFTW calls are executed concurrently.

Return *Transform*

Parameters

- `[in] processingUnit`: The processing unit type to use. Must be either SPFFT_PU_HOST or SPFFT_PU_GPU and be supported by the grid itself.
- `[in] transformType`: The transform type (complex to complex or real to complex). Can be SPFFT_TRANS_C2C or SPFFT_TRANS_R2C.
- `[in] dimX`: The dimension in x. The maximum allowed depends on the grid parameters.
- `[in] dimY`: The dimension in y. The maximum allowed depends on the grid parameters.
- `[in] dimZ`: The dimension in z. The maximum allowed depends on the grid parameters.
- `[in] localZLength`: The length in z in space domain of the local MPI rank.
- `[in] numLocalElements`: The number of elements in frequency domain of the local MPI rank.
- `[in] indexFormat`: The index format. Only SPFFT_INDEX_TRIPLETS currently supported.
- `[in] indices`: Pointer to the frequency indices. Posive and negative indexing is supported.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

int **max_dim_x**() **const**
Access a grid parameter.

Return Maximum dimension in x.

int **max_dim_y**() **const**
Access a grid parameter.

Return Maximum dimension in y.

int **max_dim_z**() **const**
Access a grid parameter.

Return Maximum dimension in z.

int **max_num_local_z_columns**() **const**
Access a grid parameter.

Return Maximum number of z-columns in frequency domain of the local MPI rank.

int **max_local_z_length**() **const**
Access a grid parameter.

Return Maximum length in z in space domain of the local MPI rank.

SpfftProcessingUnitType **processing_unit**() **const**
Access a grid parameter.

Return The processing unit, the grid has prepared for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.

int **device_id**() **const**
Access a grid parameter.

Return The GPU device id used. Always returns 0, if no GPU support is enabled.

int **num_threads**() **const**
Access a grid parameter.

Return The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

2.7 Transform

Note: This class only holds an internal reference counted object. The object remains in a usable state even if the associated Grid object is destroyed. In addition, copying a transform only requires an internal copy of a shared pointer.

class *spfft*::**Transform**
A transform in double precision with fixed dimensions.

Shares memory with other transform created from the same *Grid* object.

Public Functions

Transform(**const** *Transform*&) = default
Default copy constructor.

Transform(*Transform*&&) = default
Default move constructor.

Transform &**operator=**(**const** *Transform*&) = default
Default copy operator.

Transform &**operator=**(*Transform*&&) = default
Default move operator.

Transform **clone**() **const**
Clone transform.

Return Independent transform with the same parameters, but with new underlying grid.

SpfftTransformType **type**() **const**
Access a transform parameter.

Return Type of transform.

int **dim_x**() **const**
Access a transform parameter.

Return Dimension in x.

int **dim_y**() **const**
Access a transform parameter.

Return Dimension in y.

int **dim_z**() **const**
Access a transform parameter.

Return Dimension in z.

int **local_z_length**() **const**
Access a transform parameter.

Return Length in z of the space domain slice held by the local MPI rank.

int **local_z_offset**() **const**
Access a transform parameter.

Return Offset in z of the space domain slice held by the local MPI rank.

int **local_slice_size**() **const**
Access a transform parameter.

Return Number of elements in the space domain slice held by the local MPI rank.

long long int **global_size**() **const**
Access a transform parameter.

Return Global number of elements in space domain. Equals *dim_x()* * *dim_y()* * *dim_z()*.

int **num_local_elements**() **const**
Access a transform parameter.

Return Number of elements in frequency domain.

long long int **num_global_elements**() **const**
Access a transform parameter.

Return Global number of elements in frequency domain.

SpfftProcessingUnitType **processing_unit**() **const**
Access a transform parameter.

Return The processing unit used for calculations. Can be SPFFT_PU_HOST or SPFFT_PU_GPU.

int **device_id**() **const**
Access a transform parameter.

Return The GPU device id used. Returns always 0, if no GPU support is enabled.

int **num_threads**() **const**
Access a transform parameter.

Return The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

MPI_Comm **communicator**() **const**
Access a transform parameter.

Return The internal MPI communicator.

double ***space_domain_data**(*SpfftProcessingUnitType dataLocation*)
Provides access to the space domain data.

Return Pointer to space domain data on given processing unit. Alignment is guaranteed to fulfill requirements for std::complex and C language complex types.

Parameters

- `[in] dataLocation`: The processing unit to query for the data. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).

Exceptions

- *`GenericError`*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

void **forward**(*SpfftProcessingUnitType inputLocation*, double **output*, *SpfftScalingType scaling* = *SPFFT_NO_SCALING*)
Execute a forward transform from space domain to frequency domain.

Parameters

- `[in] inputLocation`: The processing unit, to take the input from. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).
- `[out] output`: Pointer to memory, where the frequency domain elements are written to. Can be located at Host or GPU memory (if GPU is set as processing unit).
- `[in] scaling`: Controls scaling of output. SPFFT_NO_SCALING to disable or SPFFT_FULL_SCALING to scale by factor 1 / (*dim_x()* * *dim_y()* * *dim_z()*).

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

void **backward**(**const** double *_input_, *SpfftProcessingUnitType* _outputLocation_)

Execute a backward transform from frequency domain to space domain.

Parameters

- `[in] input`: Input data in frequency domain. Must match the indices provided at transform creation. Can be located at Host or GPU memory, if GPU is set as processing unit.
- `[in] outputLocation`: The processing unit, to place the output at. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

2.8 TransformFloat

Note: This class is only available if single precision support is enabled, in which case the marco SPFFT_SINGLE_PRECISION is defined in config.h.

Note: This class only holds an internal reference counted object. The object remains in a usable state even if the associated Grid object is destroyed. In addition, copying a transform only requires an internal copy of a shared pointer.

class *spfft*::**TransformFloat**

A transform in single precision with fixed dimensions.

Shares memory with other transform created from the same *Grid* object.

Public Functions

TransformFloat(**const** *TransformFloat*&) = default
Default copy constructor.

TransformFloat(*TransformFloat*&&) = default
Default move constructor.

TransformFloat &**operator=**(**const** *TransformFloat*&) = default
Default copy operator.

TransformFloat &**operator=**(*TransformFloat*&&) = default
Default move operator.

TransformFloat **clone**() **const**
Clone transform.

Return Independent transform with the same parameters, but with new underlying grid.

SpfftTransformType **type**() **const**
Access a transform parameter.

Return Type of transform.

int **dim_x**() **const**
Access a transform parameter.

Return Dimension in x.

int **dim_y**() **const**
Access a transform parameter.

Return Dimension in y.

int **dim_z**() **const**
Access a transform parameter.

Return Dimension in z.

int **local_z_length**() **const**
Access a transform parameter.

Return Length in z of the space domain slice held by the local MPI rank.

int **local_z_offset**() **const**
Access a transform parameter.

Return Offset in z of the space domain slice held by the local MPI rank.

int **local_slice_size**() **const**
Access a transform parameter.

Return Number of elements in the space domain slice held by the local MPI rank.

long long int **global_size**() **const**
Access a transform parameter.

Return Global number of elements in space domain. Equals *dim_x()* * *dim_y()* * *dim_z()*.

int **num_local_elements**() **const**
Access a transform parameter.

Return Number of elements in frequency domain.

long long int **num_global_elements**() **const**
Access a transform parameter.

Return Global number of elements in frequency domain.

SpfftProcessingUnitType **processing_unit**() **const**
Access a transform parameter.

Return The processing unit used for calculations. Can be SPFFT_PU_HOST or SPFFT_PU_GPU.

int **device_id**() **const**
Access a transform parameter.

Return The GPU device id used. Returns always 0, if no GPU support is enabled.

int **num_threads**() **const**
Access a transform parameter.

Return The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

MPI_Comm **communicator**() **const**
Access a transform parameter.

Return The internal MPI communicator.

float ***space_domain_data**(*SpfftProcessingUnitType dataLocation*)
Provides access to the space domain data.

Return Pointer to space domain data on given processing unit. Alignment is guaranteed to fulfill requirements for std::complex and C language complex types.

Parameters

- `[in] dataLocation`: The processing unit to query for the data. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).

Exceptions

- *`GenericError`*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

void **forward**(*SpfftProcessingUnitType inputLocation*, float **output*, *SpfftScalingType scaling* = *SPFFT_NO_SCALING*)
Execute a forward transform from space domain to frequency domain.

Parameters

- `[in] inputLocation`: The processing unit, to take the input from. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).
- `[out] output`: Pointer to memory, where the frequency domain elements are written to. Can be located at Host or GPU memory (if GPU is set as processing unit).
- `[in] scaling`: Controls scaling of output. SPFFT_NO_SCALING to disable or SPFFT_FULL_SCALING to scale by factor 1 / (*dim_x()* * *dim_y()* * *dim_z()*).

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

void **backward**(**const** float **input*, *SpfftProcessingUnitType outputLocation*)

Execute a backward transform from frequency domain to space domain.

Parameters

- `[in] input`: Input data in frequency domain. Must match the indices provided at transform creation. Can be located at Host or GPU memory, if GPU is set as processing unit.
- `[in] outputLocation`: The processing unit, to place the output at. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

2.9 Multi-Transform

Note: Only fully independent transforms can be executed in parallel.

namespace spfft

Functions

void **multi_transform_forward**(int *numTransforms*, *Transform* **transforms*, *SpfftProcessingUnitType* **inputLocations*, double ***outputPointers*, *SpfftScalingType* **scalingTypes*)

Execute multiple independent forward transforms at once by internal pipelining.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputLocations`: Input locations for each transform.
- `[out] outputPointers`: Output pointers for each transform.
- `[in] scalingTypes`: Scaling types for each transform.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

void **multi_transform_backward**(int *numTransforms*, *Transform* *transforms, double **inputPointers, *SpfftProcessingUnitType* *outputLocations)

Execute multiple independent backward transforms at once by internal pipelining.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputPointers`: Input pointers for each transform.
- `[in] outputLocations`: Output locations for each transform.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

namespace spfft

Functions

void **multi_transform_forward**(int *numTransforms*, *TransformFloat* *transforms, *SpfftProcessingUnitType* *inputLocations, float **outputPointers, *SpfftScalingType* *scalingTypes)

Execute multiple independent forward transforms at once by internal pipelining.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputLocations`: Input locations for each transform.
- `[out] outputPointers`: Output pointers for each transform.
- `[in] scalingTypes`: Scaling types for each transform.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

void **multi_transform_backward**(int *numTransforms*, *TransformFloat* *transforms, float **inputPointers, *SpfftProcessingUnitType* *outputLocations)

Execute multiple independent backward transforms at once by internal pipelining.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputPointers`: Input pointers for each transform.
- `[in] outputLocations`: Output locations for each transform.

Exceptions

- *GenericError*: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

2.10 Exceptions

namespace spfft

class DuplicateIndicesError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Duplicate indices given to transform.

May indicate non-local z-coloumn between MPI ranks.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class FFTWError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> FFTW library error.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GenericError : **public** exception
#include <*exceptions.hpp*> A generic error.

Base type for all other exceptions.

Subclassed by *spfft::DuplicateIndicesError*, *spfft::FFTWError*, *spfft::GPUError*, *spfft::HostAllocationError*, *spfft::HostExecutionError*, *spfft::InternalError*, *spfft::InvalidIndicesError*, *spfft::InvalidParameterError*, *spfft::MPIError*, *spfft::MPIParameterMismatchError*, *spfft::MPISupportError*, *spfft::OverflowError*

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError*

class GPUAllocationError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Failed allocation on GPU.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUCopyError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Failed to copy from / to GPU.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Generic GPU error.

Base type for all GPU related exceptions.

Subclassed by *spfft::GPUAllocationError*, *spfft::GPUCopyError*, *spfft::GPUFFTError*, *spfft::GPUInvalidDevicePointerError*, *spfft::GPUInvalidValueError*, *spfft::GPULaunchError*, *spfft::GPUNoDeviceError*, *spfft::GPUPrecedingError*, *spfft::GPUSupportError*

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUFFTError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Failure in GPU FFT library call.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUInvalidDevicePointerError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Invalid device pointer used.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUInvalidValueError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Invalid value passed to GPU API.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPULaunchError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Failed to launch kernel on GPU.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUNoDeviceError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> No GPU device detected.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUPrecedingError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Detected error on GPU from previous GPU API / kernel calls.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class GPUSupportError : **public** *spfft*::*GPUError*
#include <*exceptions.hpp*> Library not compiled with GPU support.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class HostAllocationError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Failed allocation on host.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class HostExecutionError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Failed execution on host.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class InternalError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Unknown internal error.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class InvalidIndicesError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Invalid indices given to transform.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class InvalidParameterError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Invalid parameter.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class MPIError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> MPI error.

Only thrown if error code of MPI API calls is non-zero.

Public Functions

auto **what** () **const noexcept** -> **const** char* **override**

auto **error_code** () **const noexcept** -> *SpfftError* **override**

class MPIParameterMismatchError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Parameters differ between MPI ranks.

Public Functions

auto **what**() **const noexcept** -> **const** char* **override**

auto **error_code**() **const noexcept** -> *SpfftError* **override**

class MPISupportError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Library not compiled with MPI support.

Public Functions

auto **what**() **const noexcept** -> **const** char* **override**

auto **error_code**() **const noexcept** -> *SpfftError* **override**

class OverflowError : **public** *spfft*::*GenericError*
#include <*exceptions.hpp*> Overflow of integer values.

Public Functions

auto **what**() **const noexcept** -> **const** char* **override**

auto **error_code**() **const noexcept** -> *SpfftError* **override**

2.11 Grid

Note: A Grid handle can be safely destroyed after Transform handles have been created, since internal reference counting used to prevent the release of resources while still in use.

Typedefs

typedef void ***SpfftGrid**
Grid handle.

Functions

SpfftError **spfft_grid_create**(*SpfftGrid* **grid*, int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, *SpfftProcessingUnitType* *processingUnit*, int *maxNumThreads*)
Constructor for a local grid.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[out] grid`: Handle to grid.
- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.

- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.

SpfftError **spfft_grid_create_distributed**(*SpfftGrid* **grid*, int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, int *maxLocalZLength*, *SpfftProcessingUnitType* *processingUnit*, int *maxNumThreads*, MPI_Comm *comm*, *SpfftExchangeType* *exchangeType*)

Constructor for a distributed grid.

Thread-safe if MPI thread support is set to MPI_THREAD_MULTIPLE.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[out] grid`: Handle to grid.
- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain of the local MPI rank.
- `[in] maxLocalZLength`: Maximum length in z in space domain for the local MPI rank.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.
- `[in] comm`: The MPI communicator to use. Will be duplicated for internal use.
- `[in] exchangeType`: The type of MPI exchange to use. Possible values are SPFFT_EXCH_DEFAULT, SPFFT_EXCH_BUFFERED, SPFFT_EXCH_COMPACT_BUFFERED and SPFFT_EXCH_UNBUFFERED.

SpfftError **spfft_grid_destroy**(*SpfftGrid* *grid*)

Destroy a grid.

A grid can be safely destroyed independet from any related transforms. The internal memory is released, once all associated transforms are destroyed as well (through internal reference counting).

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.

SpfftError **spfft_grid_max_dim_x**(*SpfftGrid* *grid*, int **dimX*)

Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] dimX`: Maximum dimension in x.

SpfftError **spfft_grid_max_dim_y** (*SpfftGrid grid*, int **dimY*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] dimY`: Maximum dimension in y.

SpfftError **spfft_grid_max_dim_z** (*SpfftGrid grid*, int **dimZ*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] dimZ`: Maximum dimension in z.

SpfftError **spfft_grid_max_num_local_z_columns** (*SpfftGrid grid*, int **maxNumLocalZColumns*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain of the local MPI rank.

SpfftError **spfft_grid_max_local_z_length** (*SpfftGrid grid*, int **maxLocalZLength*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] maxLocalZLength`: Maximum length in z in space domain of the local MPI rank. rank.

SpfftError **spfft_grid_processing_unit** (*SpfftGrid grid*, *SpfftProcessingUnitType *processingUnit*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] processingUnit`: The processing unit, the grid has prepared for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.

SpfftError **spfft_grid_device_id**(*SpfftGrid grid*, int **deviceId*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] deviceId`: The GPU device id used. Returns always 0, if no GPU support is enabled.

SpfftError **spfft_grid_num_threads**(*SpfftGrid grid*, int **numThreads*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] numThreads`: The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

SpfftError **spfft_grid_communicator**(*SpfftGrid grid*, MPI_Comm **comm*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] comm`: The internal MPI communicator.

2.12 GridFloat

Note: A Grid handle can be safely destroyed after Transform handles have been created, since internal reference counting used to prevent the release of resources while still in use.

Note: These functions are only available if single precision support is enabled, in which case the marco SPFFT_SINGLE_PRECISION is defined in config.h.

Typedefs

typedef void ***SpfftFloatGrid**
Grid handle.

Functions

SpfftError **spfft_float_grid_create**(*SpfftFloatGrid* **grid*, int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, *SpfftProcessingUnitType* *processingUnit*, int *maxNumThreads*)

Constructor for a single precision local grid.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[out] grid`: Handle to grid.
- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.

SpfftError **spfft_float_grid_create_distributed**(*SpfftFloatGrid* **grid*, int *maxDimX*, int *maxDimY*, int *maxDimZ*, int *maxNumLocalZColumns*, int *maxLocalZLength*, *SpfftProcessingUnitType* *processingUnit*, int *maxNumThreads*, MPI_Comm *comm*, *SpfftExchangeType* *exchangeType*)

Constructor for a single precision distributed grid.

Thread-safe if MPI thread support is set to MPI_THREAD_MULTIPLE.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[out] grid`: Handle to grid.
- `[in] maxDimX`: Maximum dimension in x.
- `[in] maxDimY`: Maximum dimension in y.
- `[in] maxDimZ`: Maximum dimension in z.
- `[in] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain of the local MPI rank.
- `[in] maxLocalZLength`: Maximum length in z in space domain for the local MPI rank.
- `[in] processingUnit`: The processing unit type to prepare for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.
- `[in] maxNumThreads`: The maximum number of threads, transforms created with this grid are allowed to use. If smaller than 1, the OpenMP default value is used.
- `[in] comm`: The MPI communicator to use. Will be duplicated for internal use.

- `[in] exchangeType`: The type of MPI exchange to use. Possible values are SPFFT_EXCH_DEFAULT, SPFFT_EXCH_BUFFERED, SPFFT_EXCH_COMPACT_BUFFERED and SPFFT_EXCH_UNBUFFERED.

SpfftError **spfft_float_grid_destroy**(*SpfftFloatGrid grid*)

Destroy a grid.

A grid can be safely destroyed independet from any related transforms. The internal memory is released, once all associated transforms are destroyed as well (through internal reference counting).

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.

SpfftError **spfft_float_grid_max_dim_x**(*SpfftFloatGrid grid*, int **dimX*)

Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] dimX`: Maximum dimension in x.

SpfftError **spfft_float_grid_max_dim_y**(*SpfftFloatGrid grid*, int **dimY*)

Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] dimY`: Maximum dimension in y.

SpfftError **spfft_float_grid_max_dim_z**(*SpfftFloatGrid grid*, int **dimZ*)

Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] dimZ`: Maximum dimension in z.

SpfftError **spfft_float_grid_max_num_local_z_columns**(*SpfftFloatGrid grid*, int **maxNumLocalZColumns*)

Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] maxNumLocalZColumns`: Maximum number of z-columns in frequency domain of the local MPI rank.

SpfftError **spfft_float_grid_max_local_z_length**(*SpfftFloatGrid grid*, int **maxLocalZLength*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] maxLocalZLength`: Maximum length in z in space domain of the local MPI rank. rank.

SpfftError **spfft_float_grid_processing_unit**(*SpfftFloatGrid grid*, *SpfftProcessingUnitType* **processingUnit*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] processingUnit`: The processing unit, the grid has prepared for. Can be SPFFT_PU_HOST or SPFFT_PU_GPU or SPFFT_PU_HOST | SPFFT_PU_GPU.

SpfftError **spfft_float_grid_device_id**(*SpfftFloatGrid grid*, int **deviceId*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] deviceId`: The GPU device id used. Returns always 0, if no GPU support is enabled.

SpfftError **spfft_float_grid_num_threads**(*SpfftFloatGrid grid*, int **numThreads*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] numThreads`: The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

SpfftError **spfft_float_grid_communicator**(*SpfftFloatGrid grid*, MPI_Comm **comm*)
Access a grid parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] grid`: Handle to grid.
- `[out] comm`: The internal MPI communicator.

2.13 Transform

Note: This class only holds an internal reference counted object. The object remains in a usable state even if the associated Grid object is destroyed. In addition, copying a transform only requires an internal copy of a shared pointer.

Typedefs

typedef void ***SpfftTransform**
Transform handle.

Functions

SpfftError **spfft_transform_create**(*SpfftTransform* **transform*, *SpfftGrid* *grid*, *SpfftProcessingUnitType* *processingUnit*, *SpfftTransformType* *transformType*, int *dimX*, int *dimY*, int *dimZ*, int *localZLength*, int *numLocalElements*, *SpfftIndexFormatType* *indexFormat*, **const** int **indices*)
Creates a transform from a grid handle.

Thread-safe if no FFTW calls are executed concurrently.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[out] transform`: Handle to the transform.
- `[in] grid`: Handle to the grid, with which the transform is created.
- `[in] processingUnit`: The processing unit type to use. Must be either SPFFT_PU_HOST or SPFFT_PU_GPU and be supported by the grid itself.
- `[in] transformType`: The transform type (complex to complex or real to complex). Can be SPFFT_TRANS_C2C or SPFFT_TRANS_R2C.
- `[in] dimX`: The dimension in x. The maximum allowed depends on the grid parameters.
- `[in] dimY`: The dimension in y. The maximum allowed depends on the grid parameters.
- `[in] dimZ`: The dimension in z. The maximum allowed depends on the grid parameters.
- `[in] localZLength`: The length in z in space domain of the local MPI rank.
- `[in] numLocalElements`: The number of elements in frequency domain of the local MPI rank.
- `[in] indexFormat`: The index format. Only SPFFT_INDEX_TRIPLETS currently supported.
- `[in] indices`: Pointer to the frequency indices. Posive and negative indexing is supported.

SpfftError **spfft_transform_destroy**(*SpfftTransform* *transform*)
Destroy a transform.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.

SpfftError **spfft_transform_clone**(*SpfftTransform transform*, *SpfftTransform *newTransform*)
Clone a transform.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] newTransform`: Independent transform with the same parameters, but with new underlying grid.

SpfftError **spfft_transform_forward**(*SpfftTransform transform*, *SpfftProcessingUnitType inputLocation*, double **output*, *SpfftScalingType scaling*)
Execute a forward transform from space domain to frequency domain.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[in] inputLocation`: The processing unit, to take the input from. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).
- `[out] output`: Pointer to memory, where the frequency domain elements are written to. Can be located at Host or GPU memory (if GPU is set as processing unit).
- `[in] scaling`: Controls scaling of output. SPFFT_NO_SCALING to disable or SPFFT_FULL_SCALING to scale by factor 1 / (dim_x() * dim_y() * dim_z()).

SpfftError **spfft_transform_backward**(*SpfftTransform transform*, **const** double **input*, *SpfftProcessingUnitType outputLocation*)
Execute a backward transform from frequency domain to space domain.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[in] input`: Input data in frequency domain. Must match the indices provided at transform creation. Can be located at Host or GPU memory, if GPU is set as processing unit.
- `[in] outputLocation`: The processing unit, to place the output at. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).

SpfftError **spfft_transform_get_space_domain**(*SpfftTransform transform*, *SpfftProcessingUnitType dataLocation*, double **data*)
Provides access to the space domain data.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[in] dataLocation`: The processing unit to query for the data. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).
- `[out] data`: Pointer to space domain data on given processing unit. Alignment is guaranteed to fulfill requirements for std::complex and C language complex types.

Exceptions

- `GenericError`: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

SpfftError **spfft_transform_dim_x** (*SpfftTransform transform*, int **dimX*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] dimX`: Dimension in x.

SpfftError **spfft_transform_dim_y** (*SpfftTransform transform*, int **dimY*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] dimY`: Dimension in y.

SpfftError **spfft_transform_dim_z** (*SpfftTransform transform*, int **dimZ*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] dimZ`: Dimension in z.

SpfftError **spfft_transform_local_z_length** (*SpfftTransform transform*, int **localZLength*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] localZLength`: size in z of the slice in space domain on the local MPI rank.

SpfftError **spfft_transform_local_slice_size** (*SpfftTransform transform*, int **size*)
Access a transform parameter.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] size`: Number of elements in the space domain slice held by the local MPI rank.

SpfftError **spfft_transform_local_z_offset** (*SpfftTransform transform*, int **offset*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] offset`: Offset in z of the space domain slice held by the local MPI rank.

SpfftError **spfft_transform_global_size**(*SpfftTransform transform*, long long int **globalSize*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] globalSize`: Global number of elements in space domain. Equals dim_x() * dim_y() * dim_z().

SpfftError **spfft_transform_num_local_elements**(*SpfftTransform transform*, int **numLocalElements*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] numLocalElements`: Number of local elements in frequency domain.

SpfftError **spfft_transform_num_global_elements**(*SpfftTransform transform*, long long int **numGlobalElements*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] numGlobalElements`: Global number of elements in space domain. Equals dim_x() * dim_y()
  - dim_z().

SpfftError **spfft_transform_device_id**(*SpfftTransform transform*, int **deviceId*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] deviceId`: The GPU device id used. Returns always 0, if no GPU support is enabled.

SpfftError **spfft_transform_num_threads**(*SpfftTransform transform*, int **numThreads*)
Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] numThreads`: The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

SpfftError **spfft_transform_communicator**(*SpfftTransform transform*, MPI_Comm **comm*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] comm`: The internal MPI communicator.

2.14 TransformFloat

Note: These functions are only available if single precision support is enabled, in which case the marco SPFFT_SINGLE_PRECISION is defined in config.h.

Typedefs

typedef void ***SpfftFloatTransform**

Transform handle.

Functions

SpfftError **spfft_float_transform_create**(*SpfftFloatTransform *transform*, *SpfftFloatGrid grid*, *SpfftProcessingUnitType processingUnit*, *SpfftTransformType transformType*, int *dimX*, int *dimY*, int *dimZ*, int *localZLength*, int *numLocalElements*, *SpfftIndexFormatType indexFormat*, **const** int **indices*)

Creates a single precision transform from a single precision grid handle.

Thread-safe if no FFTW calls are executed concurrently.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[out] transform`: Handle to the transform.
- `[in] grid`: Handle to the grid, with which the transform is created.
- `[in] processingUnit`: The processing unit type to use. Must be either SPFFT_PU_HOST or SPFFT_PU_GPU and be supported by the grid itself.
- `[in] transformType`: The transform type (complex to complex or real to complex). Can be SPFFT_TRANS_C2C or SPFFT_TRANS_R2C.
- `[in] dimX`: The dimension in x. The maximum allowed depends on the grid parameters.
- `[in] dimY`: The dimension in y. The maximum allowed depends on the grid parameters.

- `[in] dimZ`: The dimension in z. The maximum allowed depends on the grid parameters.
- `[in] localZLength`: The length in z in space domain of the local MPI rank.
- `[in] numLocalElements`: The number of elements in frequency domain of the local MPI rank.
- `[in] indexFormat`: The index format. Only SPFFT_INDEX_TRIPLETS currently supported.
- `[in] indices`: Pointer to the frequency indices. Posive and negative indexing is supported.

SpfftError **spfft_float_transform_destroy**(*SpfftFloatTransform transform*)

Destroy a transform.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.

SpfftError **spfft_float_transform_clone**(*SpfftFloatTransform transform*, *SpfftFloatTransform *newTransform*)

Clone a transform.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] newTransform`: Independent transform with the same parameters, but with new underlying grid.

SpfftError **spfft_float_transform_forward**(*SpfftFloatTransform transform*, *SpfftProcessingUnitType inputLocation*, float **output*, *SpfftScalingType scaling*)

Execute a forward transform from space domain to frequency domain.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[in] inputLocation`: The processing unit, to take the input from. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).
- `[out] output`: Pointer to memory, where the frequency domain elements are written to. Can be located at Host or GPU memory (if GPU is set as processing unit).
- `[in] scaling`: Controls scaling of output. SPFFT_NO_SCALING to disable or SPFFT_FULL_SCALING to scale by factor 1 / (dim_x() * dim_y() * dim_z()).

SpfftError **spfft_float_transform_backward**(*SpfftFloatTransform transform*, **const** float **input*, *SpfftProcessingUnitType outputLocation*)

Execute a backward transform from frequency domain to space domain.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.

- `[in] input`: Input data in frequency domain. Must match the indices provided at transform creation. Can be located at Host or GPU memory, if GPU is set as processing unit.
- `[in] outputLocation`: The processing unit, to place the output at. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).

SpfftError **spfft_float_transform_get_space_domain**(*SpfftFloatTransform transform*, *SpfftProcessingUnitType dataLocation*, float **data*)

Provides access to the space domain data.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[in] dataLocation`: The processing unit to query for the data. Can be SPFFT_PU_HOST or SPFFT_PU_GPU (if GPU is set as execution unit).
- `[out] data`: Pointer to space domain data on given processing unit. Alignment is guaranteed to fulfill requirements for std::complex and C language complex types.

Exceptions

- `GenericError`: SpFFT error. Can be a derived type.
- `std::exception`: Error from standard library calls. Can be a derived type.

SpfftError **spfft_float_transform_dim_x**(*SpfftFloatTransform transform*, int **dimX*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] dimX`: Dimension in x.

SpfftError **spfft_float_transform_dim_y**(*SpfftFloatTransform transform*, int **dimY*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] dimY`: Dimension in y.

SpfftError **spfft_float_transform_dim_z**(*SpfftFloatTransform transform*, int **dimZ*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] dimZ`: Dimension in z.

SpfftError **spfft_float_transform_local_z_length**(*SpfftFloatTransform transform*, int **localZLength*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] localZLength`: size in z of the slice in space domain on the local MPI rank.

SpfftError **spfft_float_transform_local_slice_size**(*SpfftFloatTransform transform*, int **size*)

Access a transform parameter.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] size`: Number of elements in the space domain slice held by the local MPI rank.

SpfftError **spfft_float_transform_global_size**(*SpfftFloatTransform transform*, long long int **globalSize*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] globalSize`: Global number of elements in space domain. Equals dim_x() * dim_y() * dim_z().

SpfftError **spfft_float_transform_local_z_offset**(*SpfftFloatTransform transform*, int **offset*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] offset`: Offset in z of the space domain slice held by the local MPI rank.

SpfftError **spfft_float_transform_num_local_elements**(*SpfftFloatTransform transform*, int **numLocalElements*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] numLocalElements`: Number of local elements in frequency domain.

SpfftError **spfft_float_transform_num_global_elements**(*SpfftFloatTransform transform*, long long int **numGlobalElements*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] numGlobalElements`: Global number of elements in space domain. Equals dim_x() * dim_y()
  - dim_z().

SpfftError **spfft_float_transform_device_id**(*SpfftFloatTransform transform*, int **deviceId*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] deviceId`: The GPU device id used. Returns always 0, if no GPU support is enabled.

SpfftError **spfft_float_transform_num_threads**(*SpfftFloatTransform transform*, int **numThreads*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] numThreads`: The exact number of threads used by transforms created from this grid. May be less than the maximum given to the constructor. Always 1, if not compiled with OpenMP support.

SpfftError **spfft_float_transform_communicator**(*SpfftFloatTransform transform*, MPI_Comm **comm*)

Access a transform parameter.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] transform`: Handle to the transform.
- `[out] comm`: The internal MPI communicator.

2.15 Multi-Transform

Note: Only fully independent transforms can be executed in parallel.

Functions

SpfftError **spfft_multi_transform_forward**(int *numTransforms*, *SpfftTransform* **transforms*, *SpfftProcessingUnitType* **inputLocations*, double ***outputPointers*, *SpfftScalingType* **scalingTypes*)

Execute multiple independent forward transforms at once by internal pipelining.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputLocations`: Input locations for each transform.
- `[out] outputPointers`: Output pointers for each transform.
- `[in] scalingTypes`: Scaling types for each transform.

SpfftError **spfft_multi_transform_backward**(int *numTransforms*, *SpfftTransform* **transforms*, double ***inputPointers*, *SpfftProcessingUnitType* **outputLocations*)

Execute multiple independent backward transforms at once by internal pipelining.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputPointers`: Input pointers for each transform.
- `[in] outputLocations`: Output locations for each transform.

Functions

SpfftError **spfft_float_multi_transform_forward**(int *numTransforms*, *SpfftFloatTransform* **transforms*, *SpfftProcessingUnitType* **inputLocations*, float ***outputPointers*, *SpfftScalingType* **scalingTypes*)

Execute multiple independent forward transforms at once by internal pipelining.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputLocations`: Input locations for each transform.
- `[out] outputPointers`: Output pointers for each transform.
- `[in] scalingTypes`: Scaling types for each transform.

SpfftError **spfft_float_multi_transform_backward**(int *numTransforms*, *SpfftFloatTransform* **transforms*, float ***inputPointers*, *SpfftProcessingUnitType* **outputLocations*)

Execute multiple independent backward transforms at once by internal pipelining.

Return Error code or SPFFT_SUCCESS.

Parameters

- `[in] numTransforms`: Number of transforms to execute.
- `[in] transforms`: Transforms to execute.
- `[in] inputPointers`: Input pointers for each transform.
- `[in] outputLocations`: Output locations for each transform.

2.16 Errors

Enums

enum SpfftError

Values:

enumerator SPFFT_SUCCESS
Success.

No error.

enumerator SPFFT_UNKNOWN_ERROR
Unknown error.

enumerator SPFFT_INVALID_HANDLE_ERROR
Invalid Grid or Transform handle.

enumerator SPFFT_OVERFLOW_ERROR
Integer overflow.

enumerator SPFFT_ALLOCATION_ERROR
Failed to allocate memory on host.

enumerator SPFFT_INVALID_PARAMETER_ERROR
Invalid parameter.

enumerator SPFFT_DUPLICATE_INDICES_ERROR
Duplicate indices given to transform.

May indicate non-local z-coloumn between MPI ranks.

enumerator SPFFT_INVALID_INDICES_ERROR
Invalid indices given to transform.

enumerator SPFFT_MPI_SUPPORT_ERROR
Library not compiled with MPI support.

enumerator SPFFT_MPI_ERROR
MPI error.

Only returned if error code of MPI API calls is non-zero.

enumerator SPFFT_MPI_PARAMETER_MISMATCH_ERROR
Parameters differ between MPI ranks.

enumerator SPFFT_HOST_EXECUTION_ERROR
Failed execution on host.

enumerator SPFFT_FFTW_ERROR
FFTW library error.

enumerator SPFFT_GPU_ERROR
Generic GPU error.

enumerator SPFFT_GPU_PRECEDING_ERROR
Detected error on GPU from previous GPU API / kernel calls.

enumerator SPFFT_GPU_SUPPORT_ERROR
Library not compiled with GPU support.

enumerator SPFFT_GPU_ALLOCATION_ERROR
Failed allocation on GPU.

enumerator SPFFT_GPU_LAUNCH_ERROR
Failed to launch kernel on GPU.

enumerator SPFFT_GPU_NO_DEVICE_ERROR
No GPU device detected.

enumerator SPFFT_GPU_INVALID_VALUE_ERROR
Invalid value passed to GPU API.

enumerator SPFFT_GPU_INVALID_DEVICE_PTR_ERROR
Invalid device pointer used.

enumerator SPFFT_GPU_COPY_ERROR
Failed to copy from / to GPU.

enumerator SPFFT_GPU_FFT_ERROR
Failure in GPU FFT library call.

INDEX

S