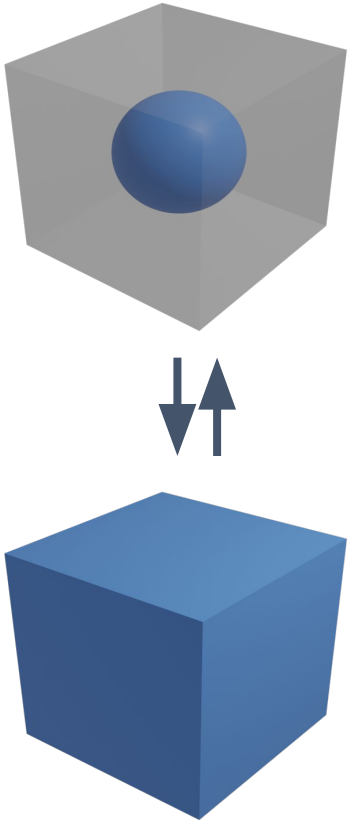
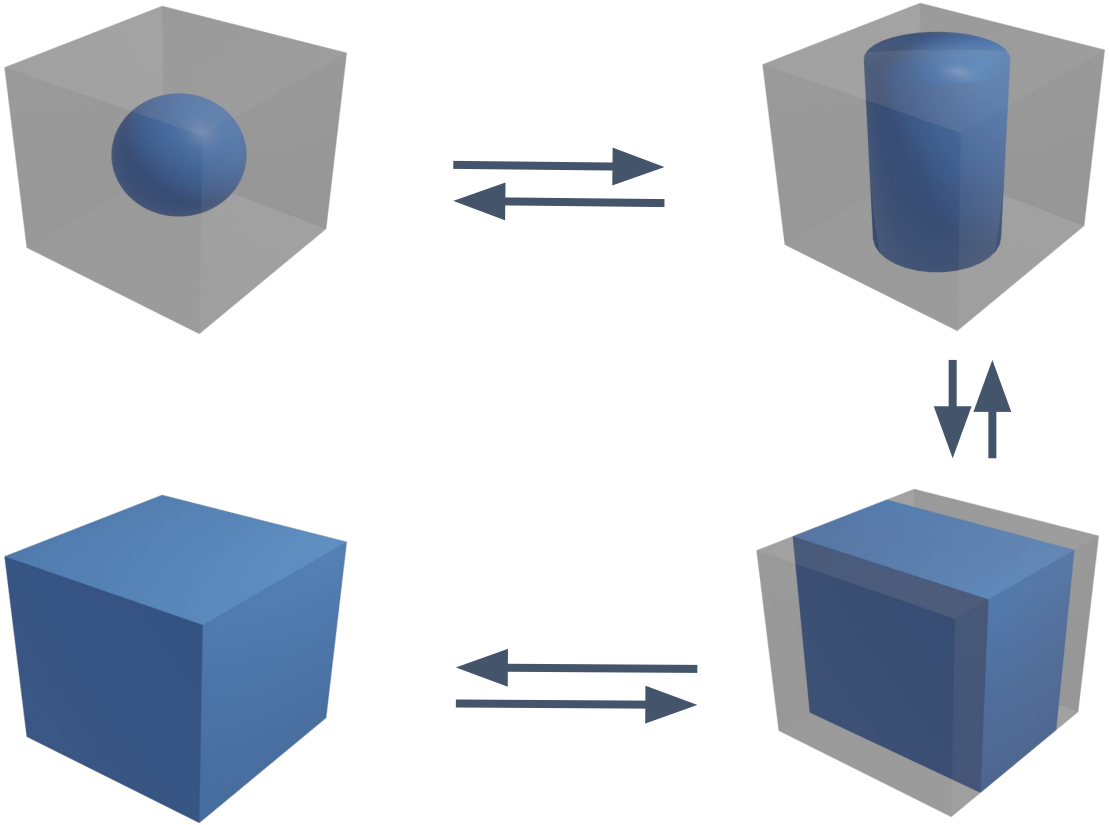
SpFFT library

Simon Frasch

SpFFT



SpFFT



SpFFT library

Design goals:

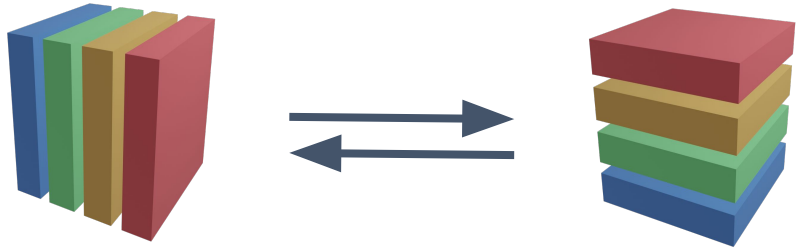
- Distributed 3D FFT computation with sparse input
- Resource reuse for transforms of different sizes
- Support for shifted indexing with centered zero-frequency
- Full use of Hermitian symmetry for complex-to-real transforms

Implementation:

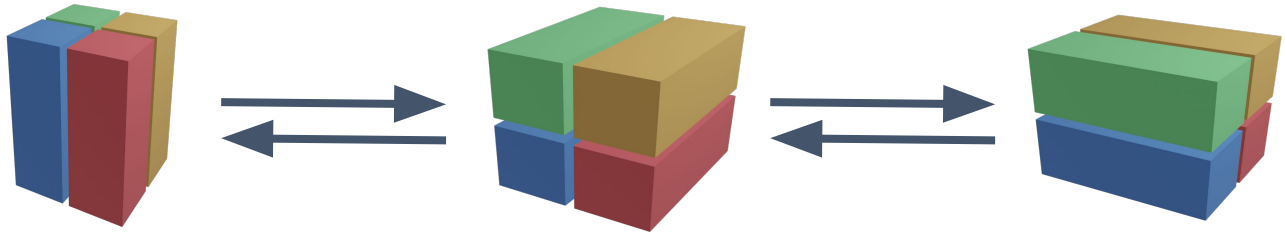
- Written in C++11
- Only mandatory dependency: Library providing a FFTW 3.x interface
- Optional parallelization and acceleration with:
  - OpenMP
  - MPI
  - CUDA or ROCm

SpFFT - Data Decomposition

Slab decomposition:

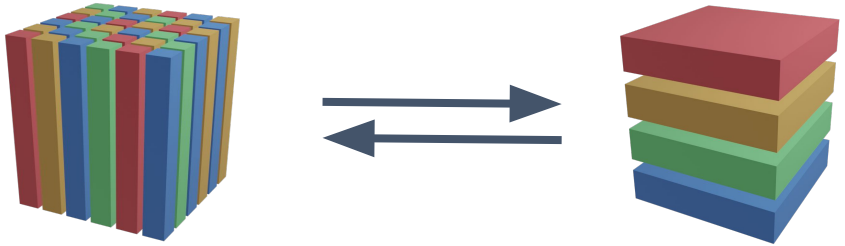


Pencil decomposition:



SpFFT - Data Decomposition

SpFFT uses a mixed decomposition:



Flexible pencil decomposition

Slab decomposition

Advantages:

- Less constraints on the distribution of sparse input data
- Better suited for GPU acceleration
- Typically faster than full pencil decomposition, provided MPI ranks do not outnumber slabs

Disadvantages:

- Parallelization is limited by the size of a dimension

SpFFT - Data exchange

Three MPI exchange methods are supported:

- **MPI_Alltoall**
  - Fixed message sizes
  - Typically best optimized for large number of ranks
- **MPI_Alltoallv**
  - Adapts to non-uniform data distribution with variable message sizes
- **MPI_Alltoallw**
  - Manual packing / unpacking of data before exchange can be avoided by using custom data types for each message

Additional features:

- Optional use of conversion to / from single precision for MPI exchange step
- CUDA aware MPI with GPUDirect to avoid data transfer between host and device

SpFFT - Interface

The interface is based on two constructs:

Grid

- Allocates memory for transforms up to a given maximum size
- Transforms of different sizes can be executed on the same grid, allowing for memory reuse

Transform

- Associated to a reference counted grid
- Created with frequency (Miller) indices of sparse input data
- For GPU acceleration: Accepts host and device pointers. Output can be selected to be placed on host or device memory.
  Note: No CUDA or ROCm API is exposed to the user.

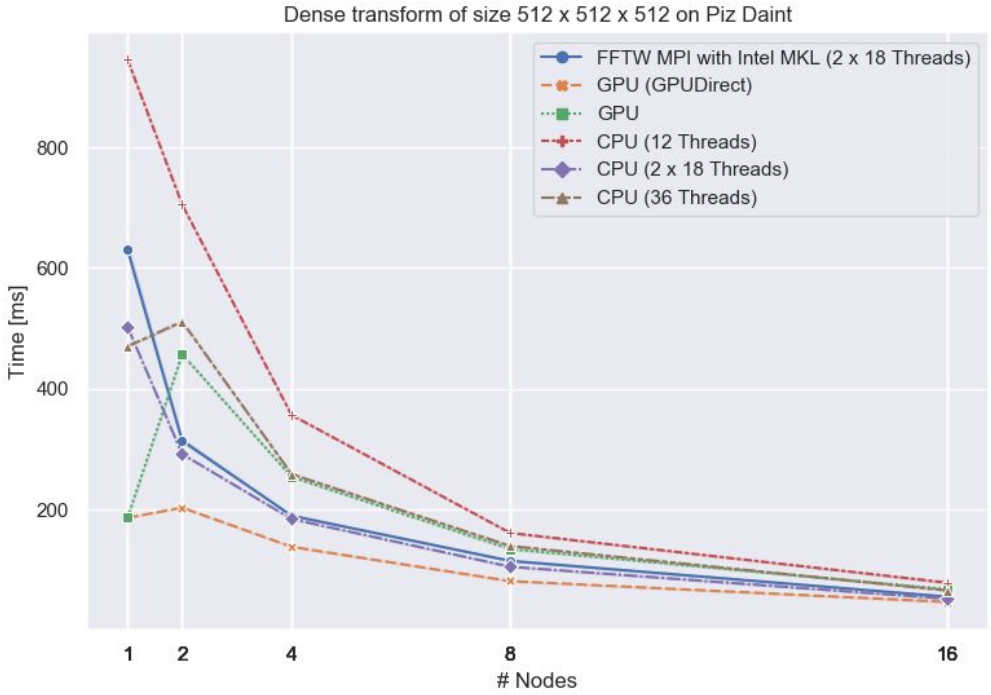
SpFFT - Example

```
! Create grid, which allocates necessary resources
spfft_grid_create_distributed(grid, dffts%nr1, dffts%nr2, dffts%nr3,&
                                    dffts%nsp(dffts%mype+1), dffts%my_nr3p,&
                                    SPFFT_PU_HOST, 1, MPI_COMM_WORLD, SPFFT_EXCH_BUFFERED)

! Create transform on grid with Miller indices
spfft_transform_create(transform, grid, SPFFT_PU_HOST, SPFFT_TRANS_C2C,&
                       dffts%nr1, dffts%nr2, dffts%nr3, dffts%my_nr3p,&
                       size(mill)/3, SPFFT_INDEX_TRIPLETS, mill)

! Grids are reference counted. Can be safely destroyed, since resources are only freed
! after associated transforms have been destroyed as well.
spfft_grid_destroy(grid)

! Memory for storing real space data is provided by transform.
! Must be translated from a C pointer.
spfft_transform_get_space_domain(transform, SPFFT_PU_HOST, psic_ptr)
call c_f_pointer(psic_ptr, psic, [n])

! Transform nbnd times forward and backwards.
do ib = 1, nbnd, 1
    spfft_transform_backward(transform, psi(:,ib), SPFFT_PU_HOST)

       ! Work on real space data here...

    spfft_transform_forward(transform, SPFFT_PU_HOST, hpsi(:,ib), SPFFT_FULL_SCALING)
enddo

! All resources are freed by destroying the only / last transform
spfft_transform_destroy(transform)
```

SpFFT - Benchmark

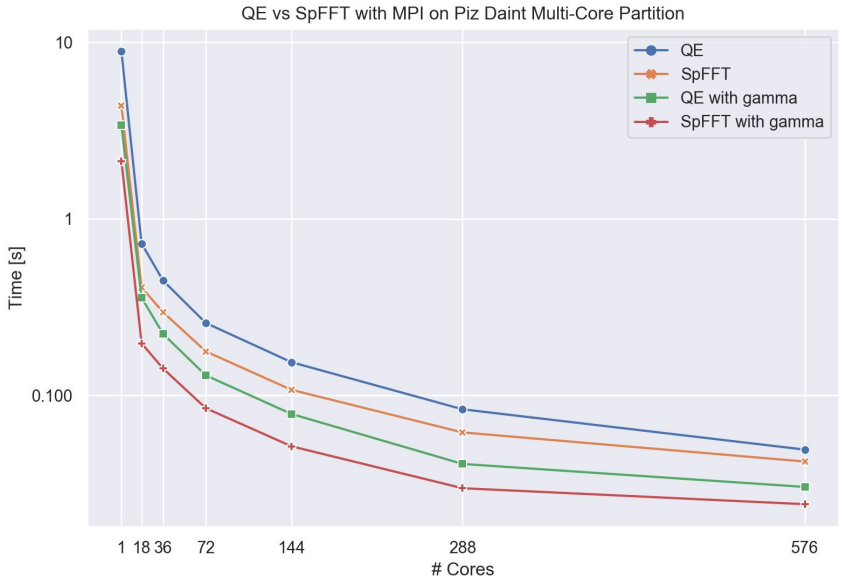
Benchmark:

- FFTW is executed with transposed output (only one internal MPI call necessary)
- GPU node: Intel Xeon E5-2690 v3 (12 cores), Nvidia Tesla P100
- Multi-Core node: 2x Intel Xeon E5-2695 v4 (2 x 18 cores)

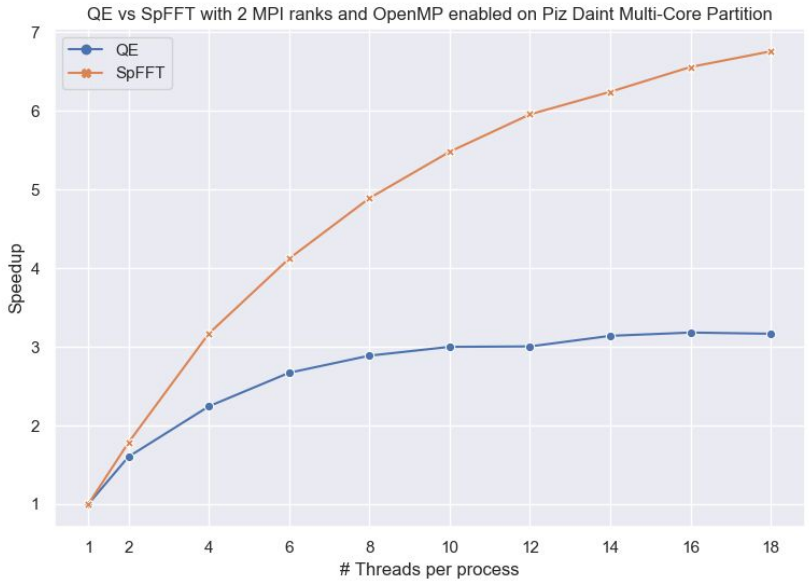


SpFFT vs QE

Timings with single thread per rank:



Multi-thread scaling with two MPI ranks:



Size: 243 x 384 x 576
Density: 34%