

INTERNATIONAL CONGRESS

Russian Supercomputing Days

dtFFT

A High-Performance Library for Distributed Data
Transpositions and Fast Fourier Transforms

Oleg Shatrov

Moscow, 2026

Problem Formulation

Navier-Stokes equations for incompressible fluid

$$\frac{\partial \mathbf{V}}{\partial t} + (\mathbf{V} \nabla) \mathbf{V} = -\nabla p + \Delta \mathbf{V} + f$$

$$\nabla \cdot \mathbf{V} = 0$$

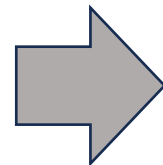
Predictor-corrector scheme

Predictor

$$\tilde{V}_\alpha = V_\alpha + \tau (\Delta_\alpha^h V_\alpha - \mathcal{K}_\alpha^h(\mathbf{V}) V_\alpha - L_{\tilde{x}_\alpha} p + f^h)$$

Corrector

$$\hat{V}_\alpha = \tilde{V}_\alpha - \tau \nabla(\delta p), \quad \delta p = (\hat{p} - p)$$



Nabla operator

$$\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3} \right)$$

Laplace operator

$$\Delta = \nabla^2 = \sum_{\alpha=1}^3 \frac{\partial^2}{\partial x_\alpha^2}$$

Pressure correction equation

$$\Delta(\delta p) = \frac{\nabla \cdot \tilde{\mathbf{V}}}{\tau}$$
$$\left. \frac{\partial(\delta p)}{\partial n} \right|_{\partial\Omega} = 0$$

Poisson Equation Solution

$$\Delta^h u = f^h, \quad \partial u / \partial n = 0$$

Eigenfunctions

$$\psi_k^\alpha(i) = \cos \frac{\pi k(i + \frac{1}{2})}{N_\alpha} \quad \begin{array}{l} i, k = 0, \dots, N_\alpha - 1 \\ \alpha = 1, 2, 3 \end{array}$$

Eigenvalues

$$\lambda_k^\alpha = \frac{4}{h_\alpha^2} \sin^2 \frac{\pi k}{2N_\alpha} \quad k = 0, \dots, N_\alpha - 1$$

Solution methods

- Fourier transform (FFT)
- Multigrid method (MG)
- Direct method (LU, LLT)
- Iterative method with an efficient preconditioner (CG, GMRES, etc.)

Fourier transform

$$-|\lambda_l^1 + \lambda_m^2 + \lambda_n^3| U_{l,m,n} = F_{l,m,n} = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} \sum_{k=0}^{N_3-1} f_{i,j,k} \psi_l^1(i) \psi_m^2(j) \psi_n^3(k)$$

$$U_{l,m,n} = -\frac{F_{l,m,n}}{|\lambda_l^1 + \lambda_m^2 + \lambda_n^3|}$$

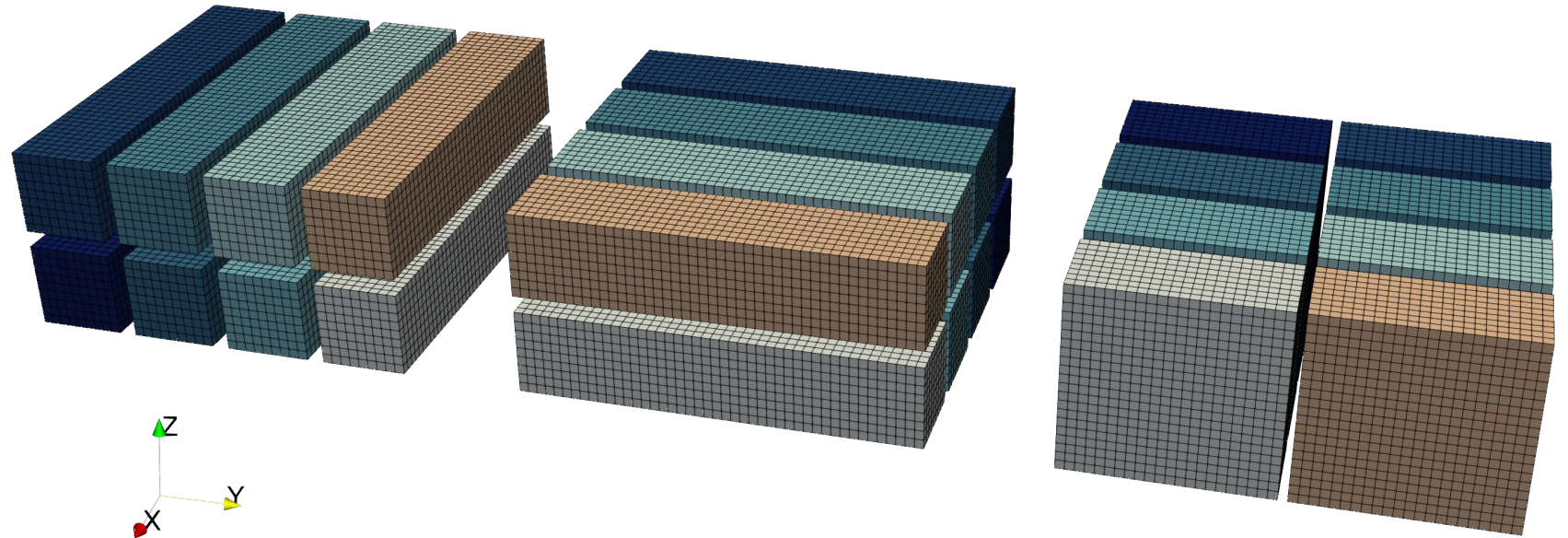
Inverse transform

$$u_{i,j,k} = \frac{8}{N_1 N_2 N_3} \sum_{l=0}^{N_1-1} \sum_{m=0}^{N_2-1} \sum_{n=0}^{N_3-1} U_{l,m,n} \psi_l^1(i) \psi_m^2(j) \psi_n^3(k)$$

Poisson Equation Solution

Fourier transform approach

1. Data distributed across MPI processes; each process owns a full row along X
2. FFT along X
3. Global transpose
→ full row along Y
4. FFT along Y
5. Global transpose
→ full row along Z
6. FFT along Z



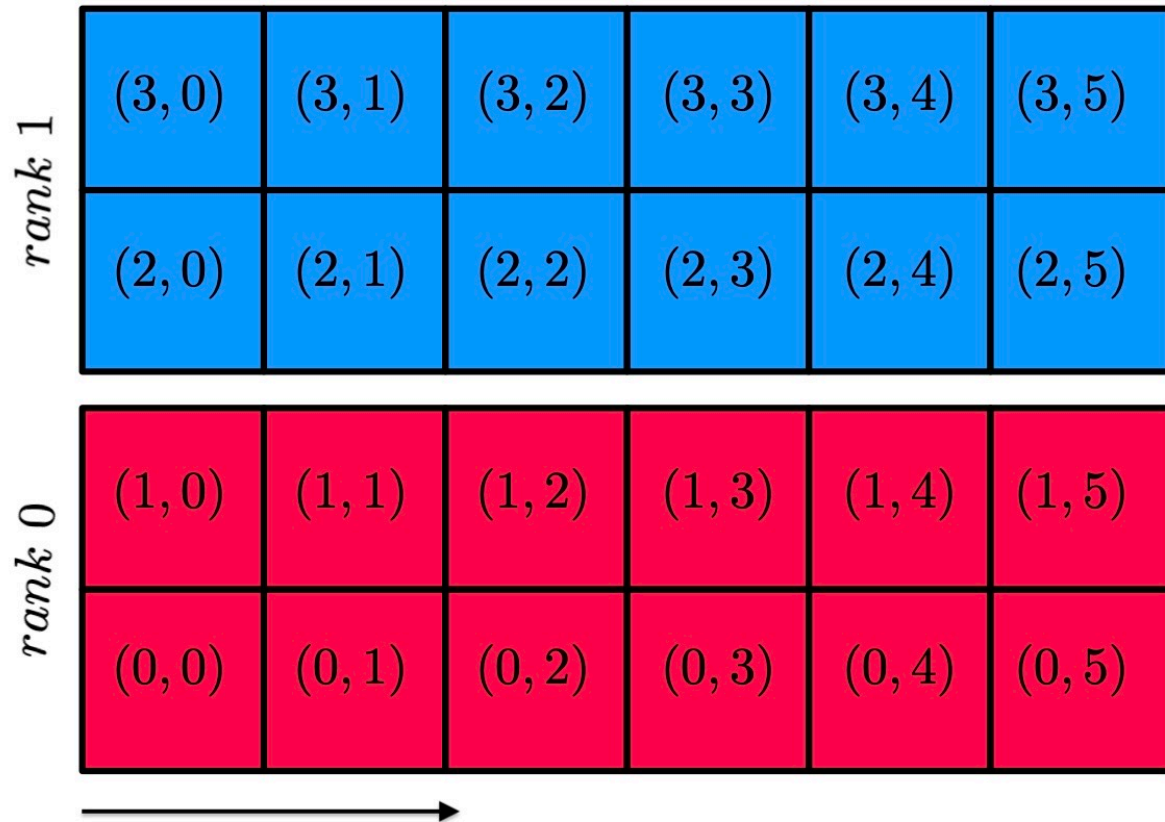
How dtFFT differs

- Both 2D and 3D transforms
- R2R, R2C, and C2C transforms
- Support for slab (1D) and pencil (2D) decompositions
- Single and double precision
- CUDA/OpenMP support
- Universal backends with explicit packing/unpacking: (MPI, NCCL, cuFFTMp)
- Two operation modes: global and local
- Transforms using derived MPI datatypes (zero-copy)
- Support for brick (3D) decomposition
- RMA support
- Runtime autotuning
- Runtime compression
- Fortran, C, C++ and Python APIs

Global Transposition Example

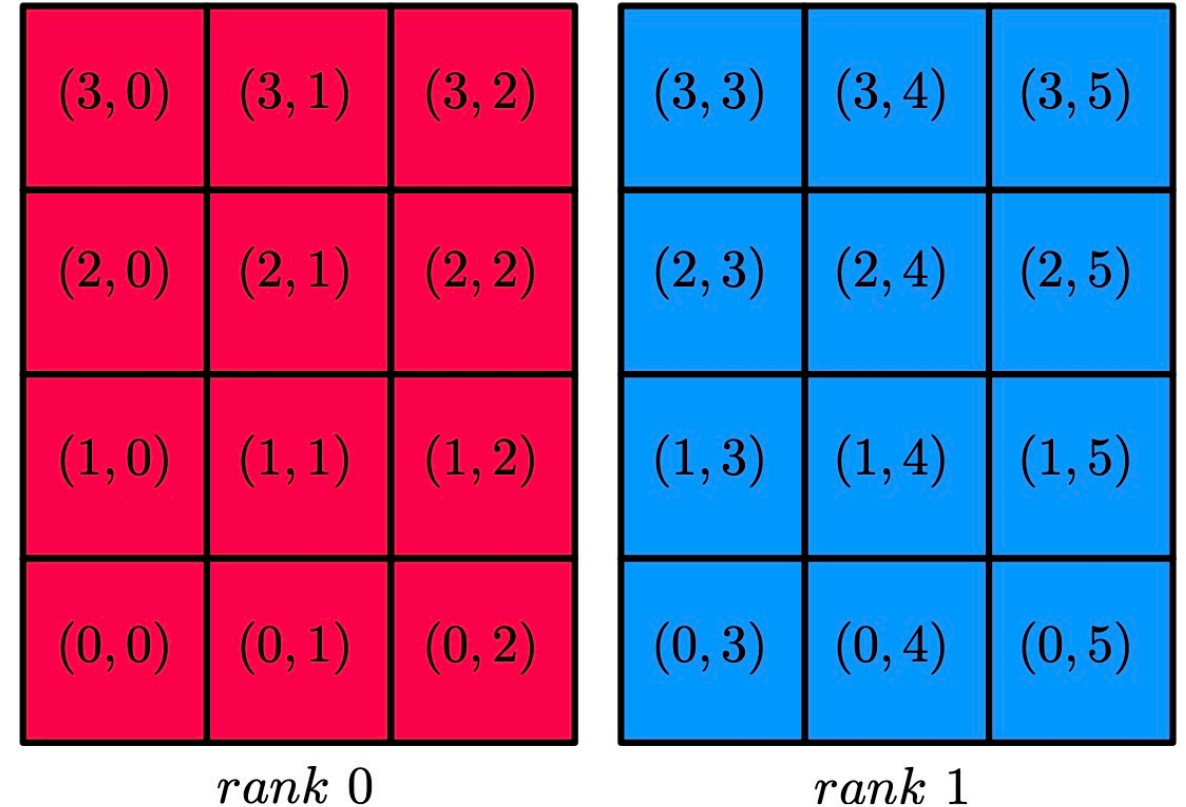
Contiguous data layout along X

<i>rank 1</i>	(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
	(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
<i>rank 0</i>	(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)



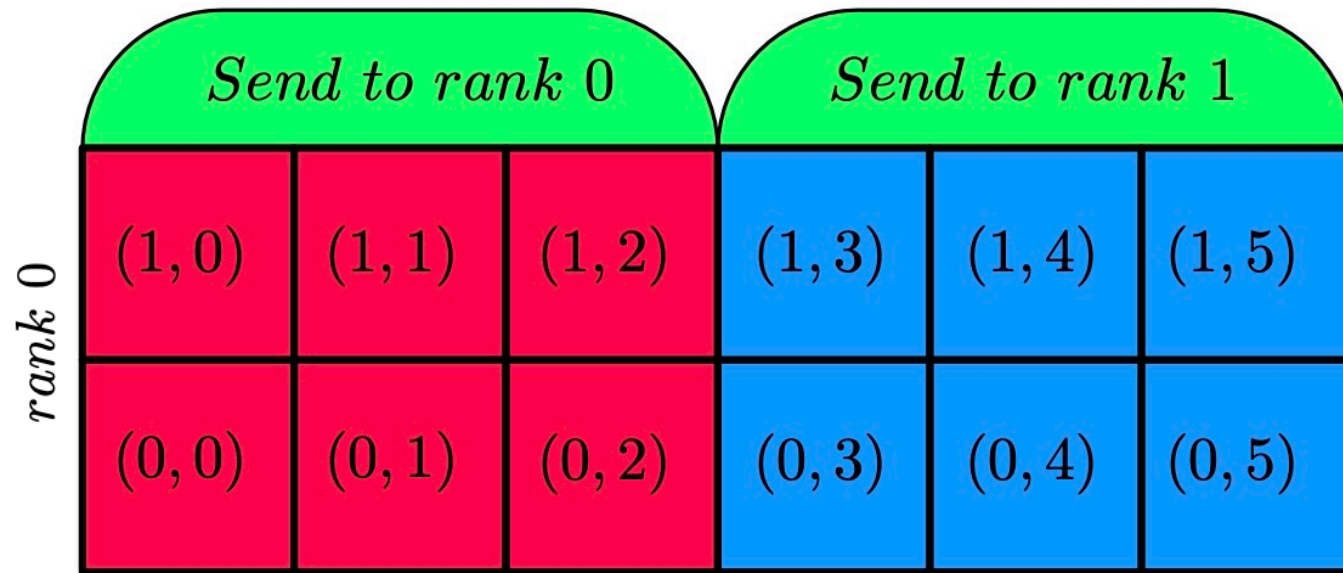
Contiguous data layout along Y

<i>rank 0</i>	(3, 0)	(3, 1)	(3, 2)
	(2, 0)	(2, 1)	(2, 2)
<i>rank 1</i>	(3, 3)	(3, 4)	(3, 5)
	(2, 3)	(2, 4)	(2, 5)



MPI Derived Datatypes

Sending data

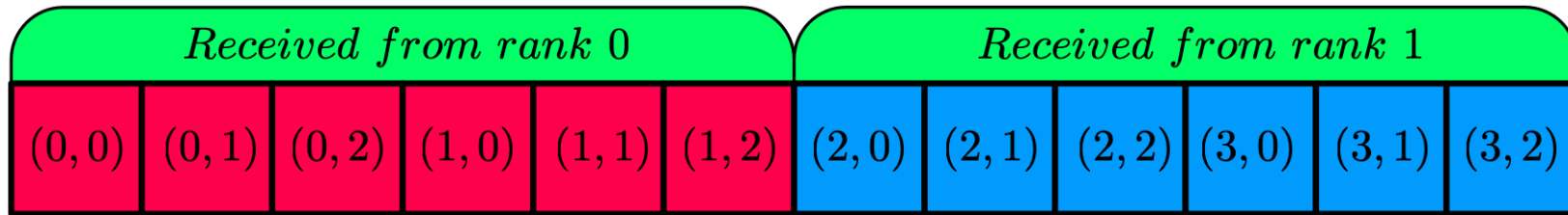


```
MPI_Type_vector(  
    2, 3, 6, MPI_INT, &temp_dtype);  
  
MPI_Type_create_resized(temp_dtype,  
    0, 3 * sizeof(int), &send_dtype);  
  
MPI_Type_commit(&send_dtype);
```

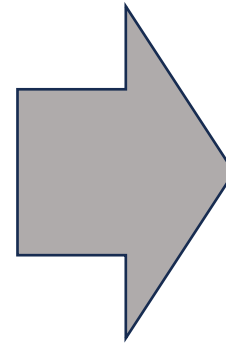


MPI Derived Datatypes

Receiving data



```
MPI_Type_vector(3, 1, 4, MPI_INT, &temp_dtype1);  
MPI_Type_create_resized(  
    temp_dtype1, 0, sizeof(int), &temp_dtype2);  
MPI_Type_contiguous(2, temp_dtype2, &recv_dtype);  
MPI_Type_commit(&recv_dtype);
```



Contiguous data layout
along Y

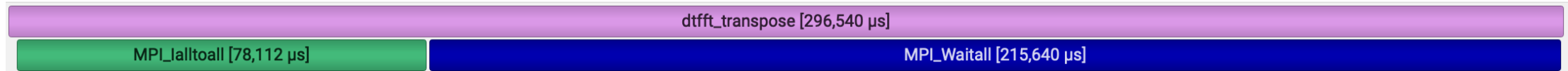
(3,0)	(3,1)	(3,2)
(2,0)	(2,1)	(2,2)
(1,0)	(1,1)	(1,2)
(0,0)	(0,1)	(0,2)

MPI Derived Datatypes

Transpose is launched by single MPI call

```
MPI_Alltoall(buffer_in, 1, send_dtype, buffer_out, 1, recv_dtype, comm);
```

DTFFT_BACKEND_MPI_DATATYPE



Advantages

- High CPU performance with even data distribution across MPI processes
- Asynchronous execution
- Communication requires only two buffers

Disadvantages

- Poor GPU performance
- Higher overhead under uneven data distribution (MPI_Alltoallw required)

Basic MPI approach

Local data transpose  Alltoall exchange  Data unpacking

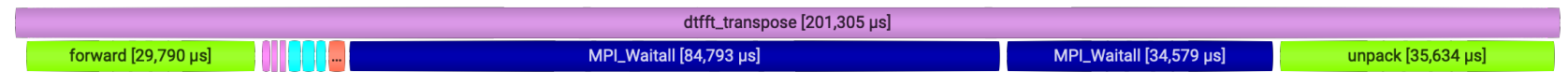
Advantages

- Good performance on both CPU and GPU
- Requires only two buffers

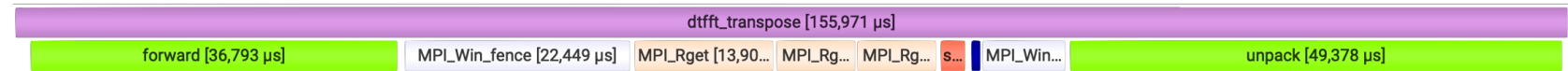
Disadvantages

- No pipelining of data processing

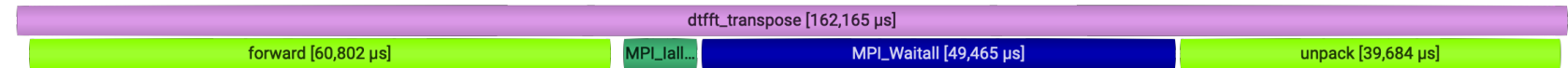
DTFFT_BACKEND_MPI_P2P



DTFFT_BACKEND_MPI_RMA



DTFFT_BACKEND_MPI_A2A



DTFFT_BACKEND_MPI_P2P_SCHEDULED



MPI pipeline

Local data transpose

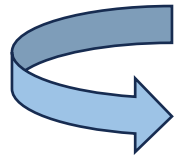


Start of Alltoall exchange



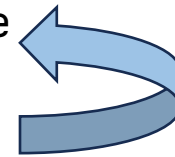
Advantages

- Often outperforms basic exchanges



Wait for new data to arrive

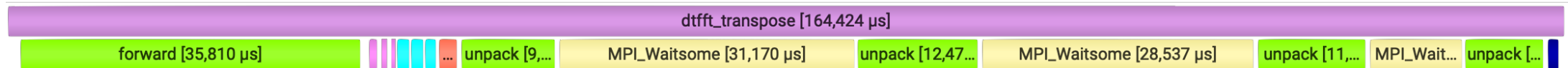
Unpack arrived data



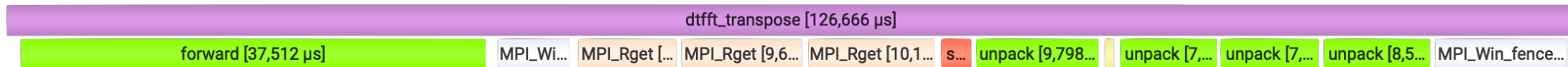
Disadvantages

- Requires at least three buffers

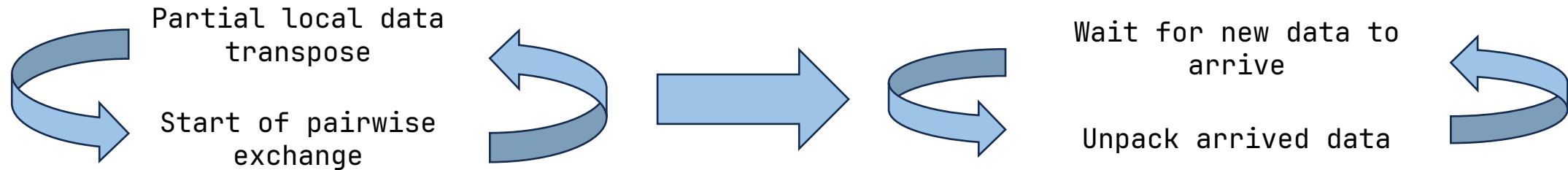
DTFFT_BACKEND_MPI_P2P_PIPELINED



DTFFT_BACKEND_MPI_RMA_PIPELINED



Advanced MPI pipeline



Advantages

- Near-optimal performance

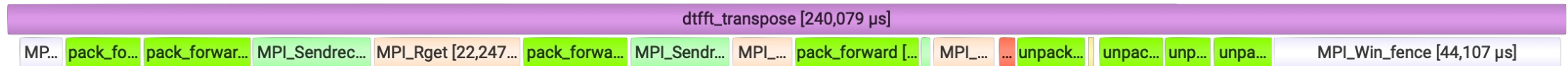
Disadvantages

- Requires at least four buffers

DTFFT_BACKEND_MPI_P2P_FUSED



DTFFT_BACKEND_MPI_RMA_FUSED



Advanced MPI pipeline with runtime compression

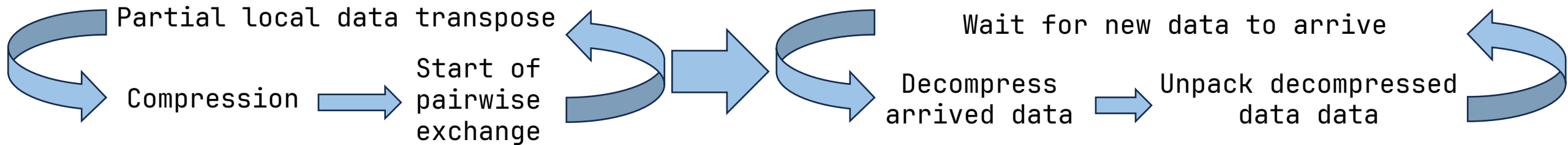
Goal

Reduce communication time by sending less data

Solution

Extending the advanced pipeline with additional steps:

- Compress on send
- Decompress on receive



Library used: ZFP

- Lossy/lossless compression modes
- CUDA/OpenMP
- Optimized for scientific data

Disadvantages

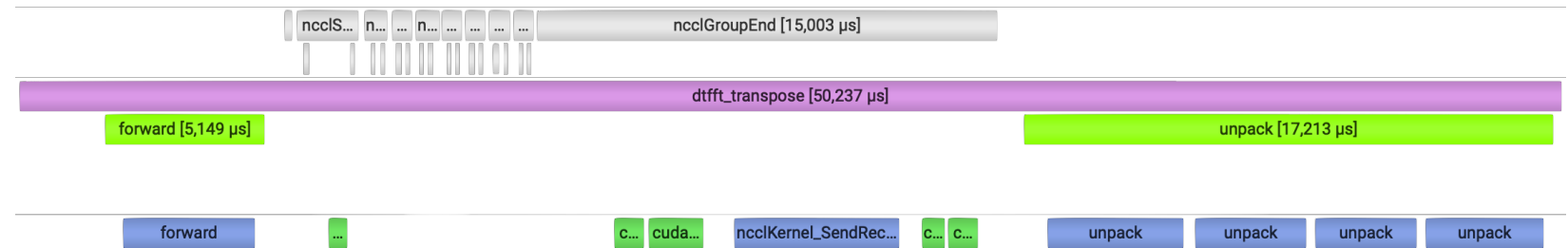
- Compression does not always reduce the buffer size
- Compression is not guaranteed to improve performance
- Requires at least four buffers

CUDA-Optimized Algorithms

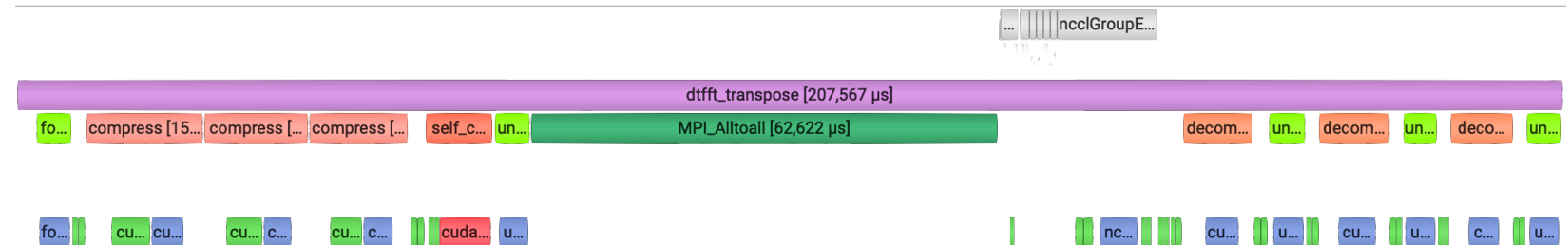
GPU-Specific Aspects:

- Runtime generation of compute kernels tailored to a specific GPU via nvRTC
- Automatic tuning of kernel launch parameters
- Support for CUDA-aware communication libraries (NCCL, cuFFTMp)

DTFFT_BACKEND_NCCL



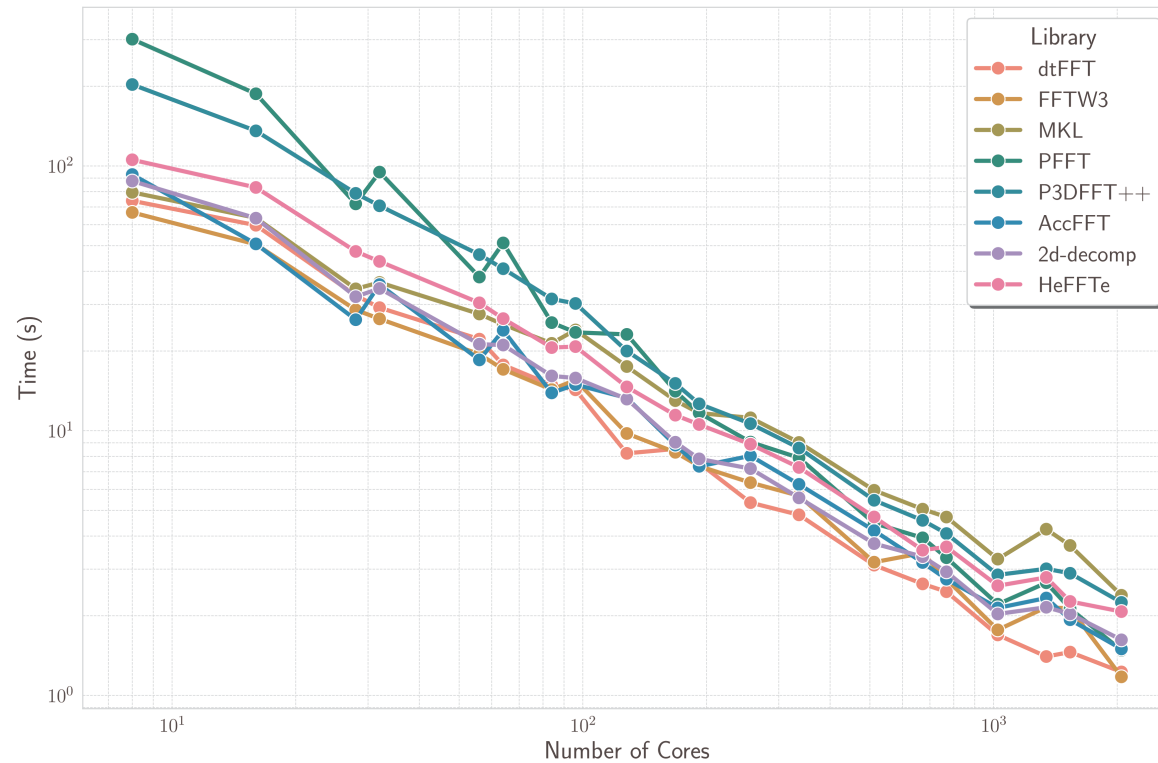
DTFFT_BACKEND_NCCL_COMPRESSED



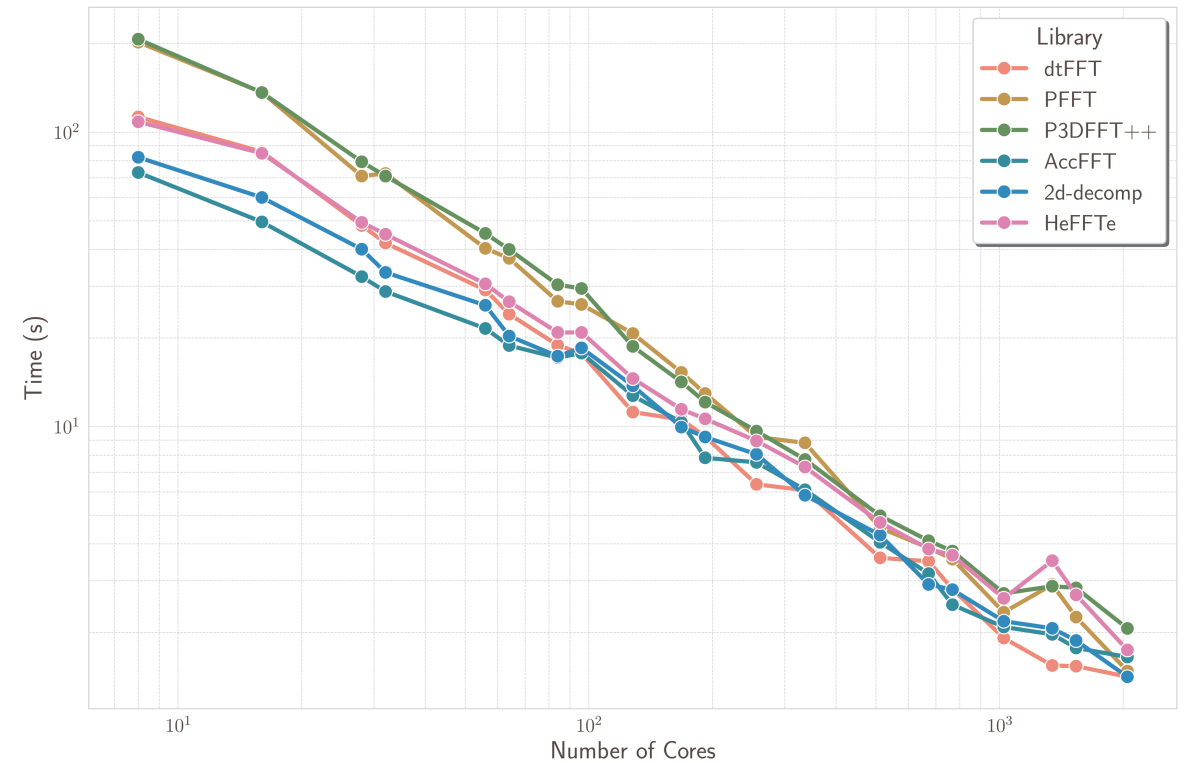
Strong Scalability on CPU

3D complex transform, double precision

Grid size: 64x2048x2048



Grid size: 2048x2048x64



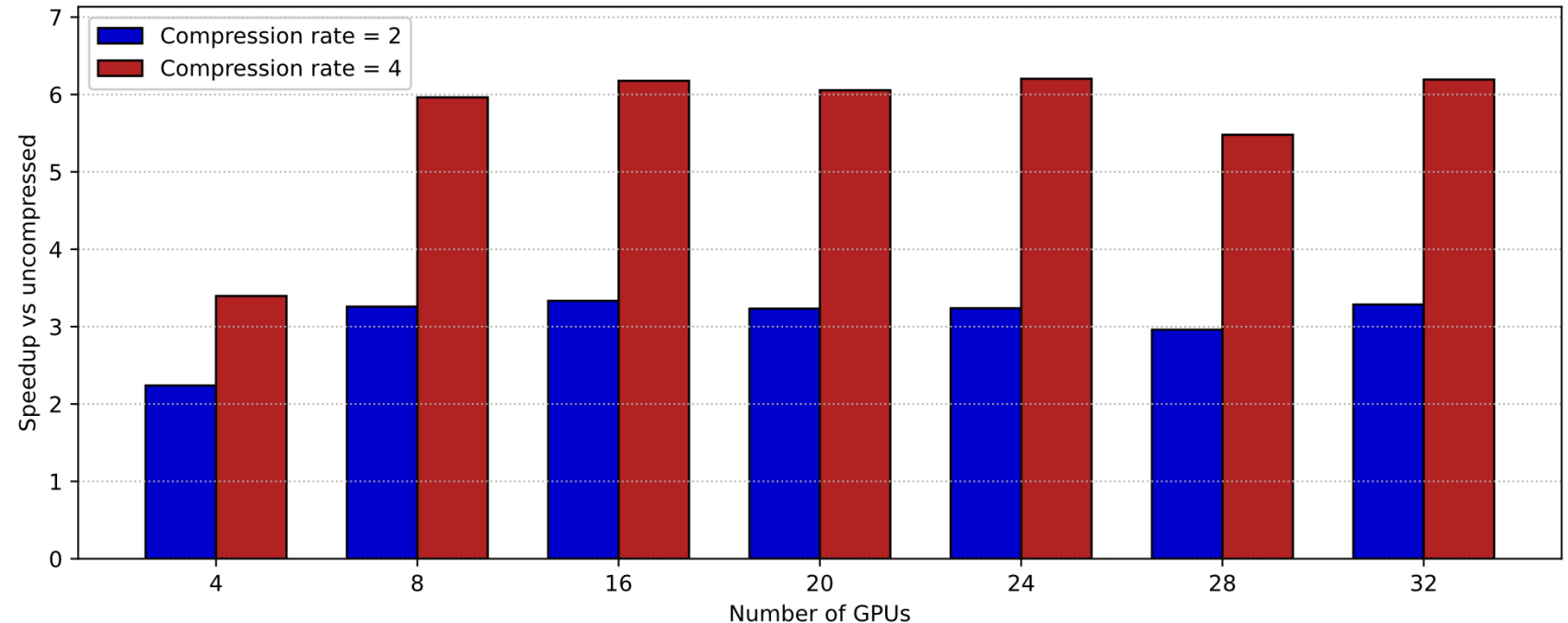
Runtime Compression for CUDA

3D complex transform, double precision
Grid size 1024x1024x1024

DTFFT_BACKEND_NCCL_COMPRESSED

Lossy compression

Fixed-rate
compression mode is
used, guaranteeing
compression by a
given factor



Future Work

In progress

- Caching of autotuning results
- Additional data compression libraries

Planned

- HIP platform support
- APIs/bindings for more programming languages
- Specialized NVSHMEM-based all-to-all exchange methods
- Support for hybrid execution across CPU and GPU
- Support for long double and quad precision
- Support for 64-bit indexing
- Support for non-uniform transforms (NFFT)
- Mixed transform types per direction (e.g., C2C along X, R2R along Y, C2C along Z)

Any questions?



Available Operation Modes

Global Decomposition Mode

- The user specifies the global problem dimensions
- Automatic grid decomposition tuning
- Brick decomposition is not applied
- Local domain dimensions can be obtained from dtFFT

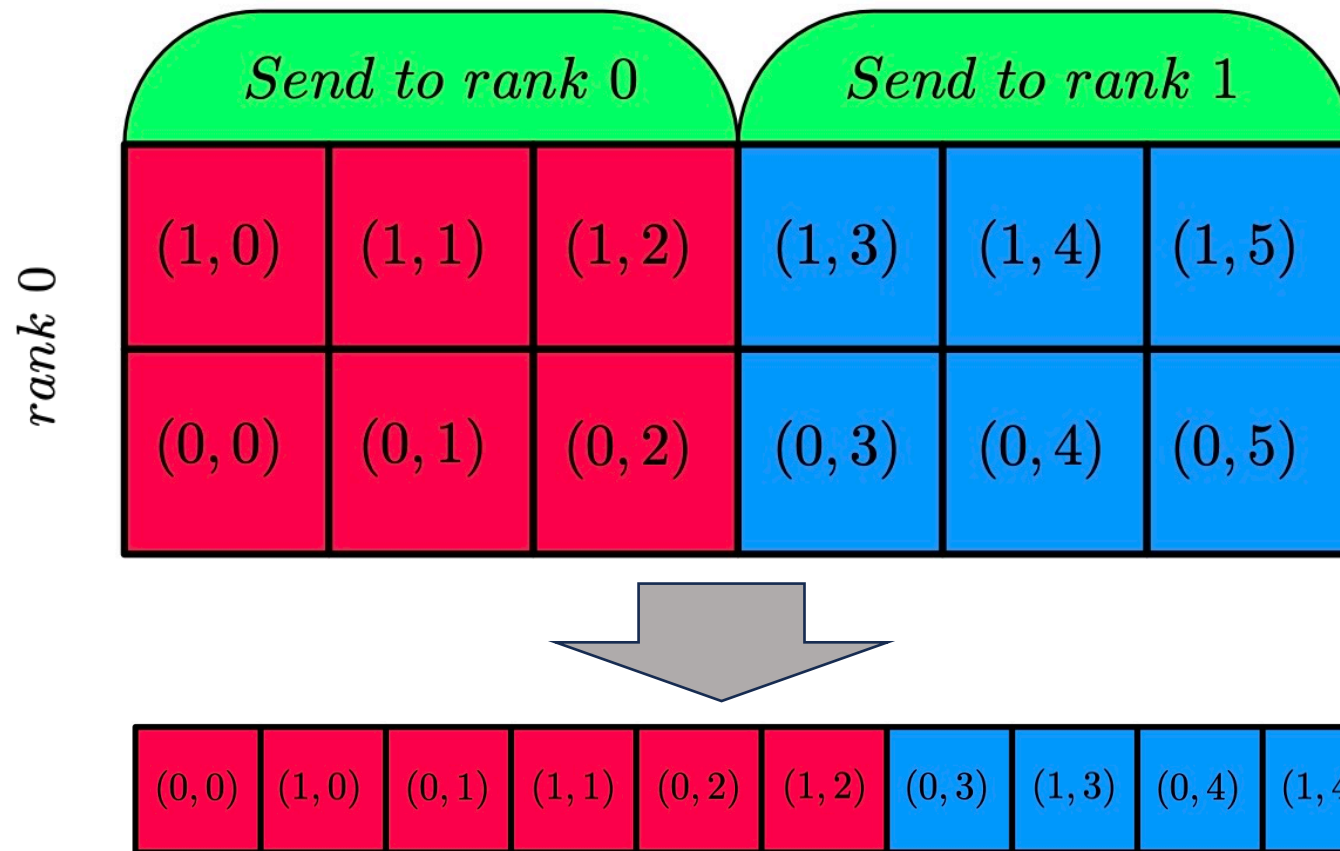
Problem: local domain dimensions returned by dtFFT may not match those used in the rest of the application

Local Decomposition Mode

- The user specifies the local domain dimensions used in the application
- The user gains full control over data locality and aligns dtFFT with existing computational domain decompositions
- Brick decomposition becomes possible

MPI Derived Datatypes

Sending data (option 2)



```
MPI_Type_vector(  
    2, 1, 6, MPI_INT, &temp_dtype1);
```

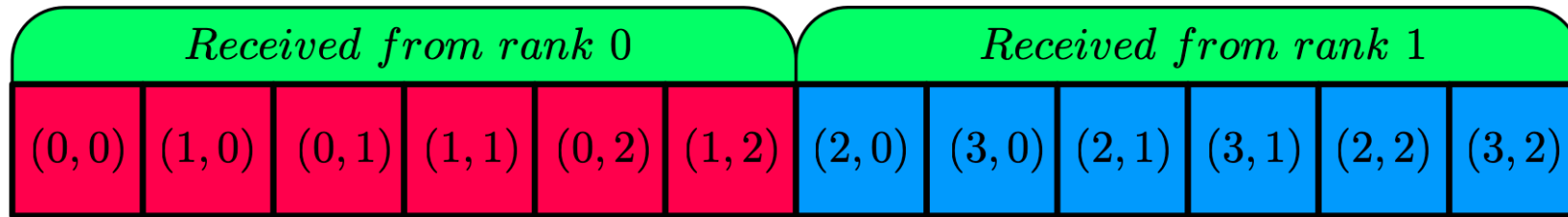
```
MPI_Type_create_resized(  
    temp_dtype1, 0, sizeof(int),  
    &temp_dtype2);
```

```
MPI_Type_contiguous(  
    3, temp_dtype2, &send_dtype);
```

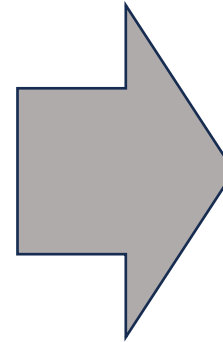
```
MPI_Type_commit(&send_dtype);
```

MPI Derived Datatypes

Receiving data (option 2)



```
MPI_Type_vector(3, 2, 4, MPI_INT, &temp_dtype);  
MPI_Type_create_resized(  
    temp_dtype, 0, 2 * sizeof(int), &recv_dtype);  
MPI_Type_commit(&recv_dtype);
```



Contiguous data layout
along Y

(3,0)	(3,1)	(3,2)
(2,0)	(2,1)	(2,2)
(1,0)	(1,1)	(1,2)
(0,0)	(0,1)	(0,2)

Runtime autotuning

Autotuning parameters

4 levels of autotuning:

- DTFFT_ESTIMATE
 - DTFFT_MEASURE
 - DTFFT_PATIENT
 - DTFFT_EXHAUSTIVE
- Global domain distribution among MPI processes
 - Slabs (1D)
 - Pencils (2D)
 - 2D FFT algorithm vs. two 1D FFTs + transposition
 - All-to-all communication algorithm for transposition and reshape (pencils $\leftrightarrow$ bricks)
 - DTFFT_BACKEND_MPI_DATATYPE, DTFFT_BACKEND_MPI_RMA_FUSED, etc.
 - Local transposition stage
 - During data packing
 - During data unpacking
 - Compute kernels for transposition
 - Loop order
 - Tiling
 - CUDA kernel launch parameters

Host compute kernels for transposition

Contiguous read

```
do z = 0, nz - 1
  do y = 0, ny - 1
    do x = 0, nx - 1
      iidx = z * nx * ny + y * nx + x
      oidx = x * ny * nz + z * ny + y
      out(oidx) = in(iidx)
```

Contiguous write

```
do x = 0, nx - 1
  do z = 0, nz - 1
    do y = 0, ny - 1
      iidx = z * nx * ny + y * nx + x
      oidx = x * ny * nz + z * ny + y
      out(oidx) = in(iidx)
```

Autotuning algorithm

- Loop order
- Tiling
- Kernel selection

Tiling

```
do xb = 0, nx - 1, BLOCK_SIZE
  do zb = 0, nz - 1, BLOCK_SIZE
    do yb = 0, ny - 1, BLOCK_SIZE
      do x = xb, min(xb + BLOCK_SIZE - 1, nx - 1)
        do z = zb, min(zb + BLOCK_SIZE - 1, nz - 1)
          do y = yb, min(yb + BLOCK_SIZE - 1, ny - 1)
            ...
```

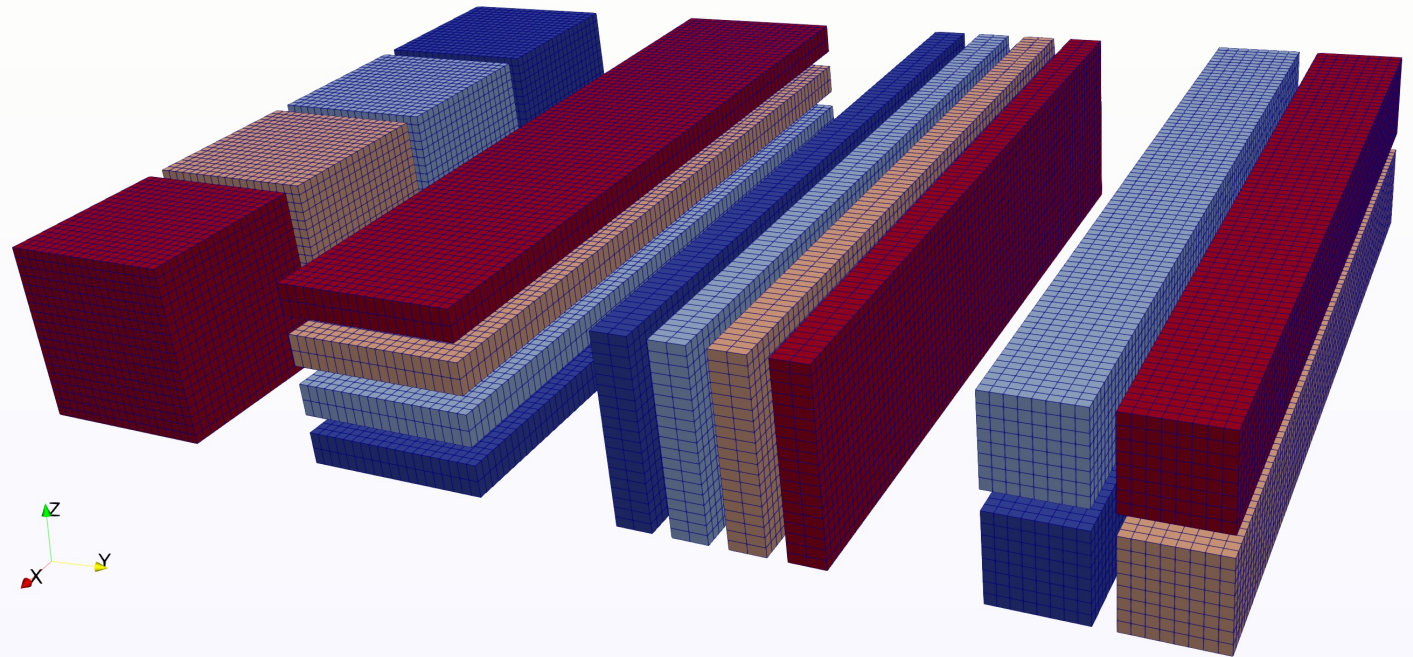
Bricks decomposition

Motivation

- Bricks distribute data over all 3 dimensions
 - better scalability and fit to existing solvers
- FFT needs a full local line
 - data must be realigned into pencils

Reshape strategies

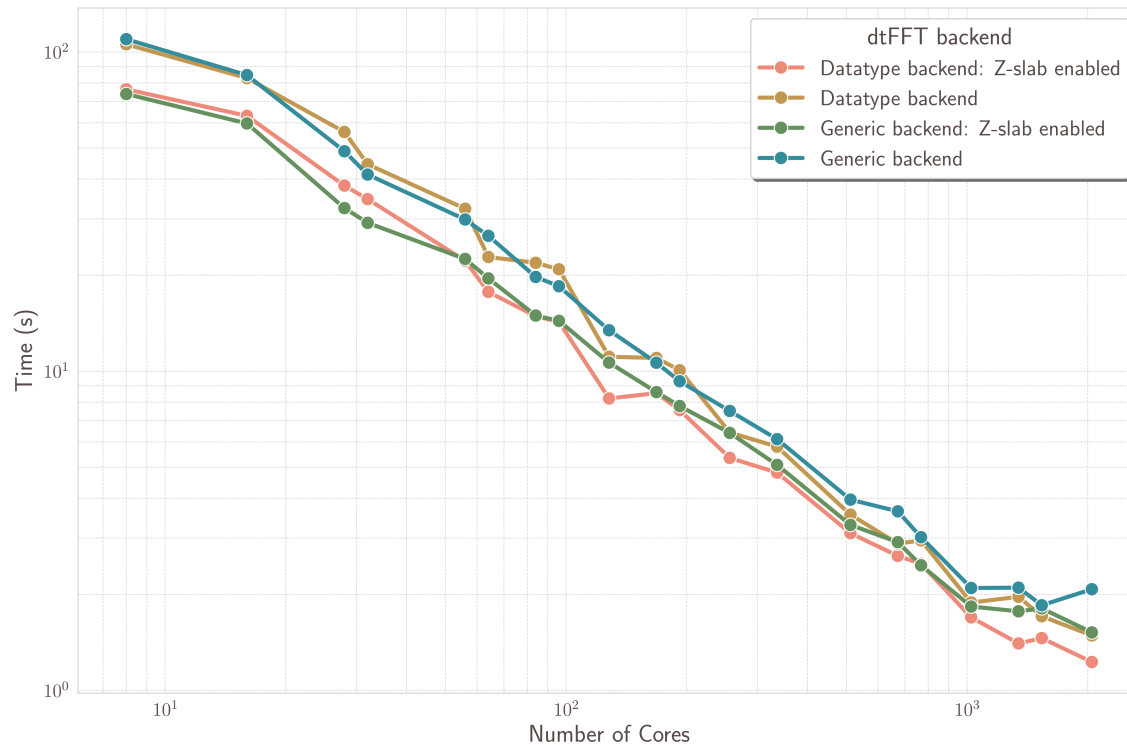
- Gather X, redistribute along Z
- Gather X, redistribute along Y
- Fallback: 2D redistribution



Strong Scalability on CPU

3D complex transform, double precision

Grid size: 64x2048x2048



Grid size: 2048x2048x64

