Fast parallel multidimensional FFT using advanced MPI

Lisandro Dalcin [1] Mikael Mortensen [2] David E. Keyes [1]

[1]Extreme Computing Research Center
King Abdullah University of Science and Technology

[2]Department of Mathematics
University of Oslo

SIAM PP20
February 14, 2020
Seattle, Washington, U.S.

Sequential FFTs of multidimensional arrays

- Full d-dimensional transforms

$$\hat{u}_{k_0,k_1,\ldots,k_{d-1}} = \mathcal{F}_0\left(\mathcal{F}_1\left(\cdots \mathcal{F}_{d-1}(u_{j_0,j_1,\ldots,j_{d-1}})\right)\right)$$

$$u_{j_0,j_1,\ldots,j_{d-1}} = \mathcal{F}_{d-1}^{-1}\left(\cdots \mathcal{F}_1^{-1}\left(\mathcal{F}_0^{-1}(\hat{u}_{k_0,k_1,\ldots,k_{d-1}})\right)\right)$$

- Partial 1-dimensional transforms

$$\tilde{u}_{j_0,\ldots,k_i,\ldots,j_{d-1}} = \mathcal{F}_i(u_{j_0,\ldots,j_i,\ldots,j_{d-1}})$$

$$u_{j_0,\ldots,j_i,\ldots,j_{d-1}} = \mathcal{F}_i^{-1}(\tilde{u}_{j_0,\ldots,k_i,\ldots,j_{d-1}})$$

Parallel FFTs of multidimensional arrays

Notation

- Consider a d-dimensional array $u_{j_0,j_1,\ldots,j_{d-1}}$
- Pick index set $j_m = 0, 1, \ldots, N_m - 1$ with $m \in 0, 1, \ldots, d-1$
- Consider group of processes P of size $|P|$ with process identifiers $p = 0, 1, \ldots, |P| - 1$
- We denote j_m distributed into P as j_m/P

Example

- d-dimensional array distributed in its second axis

$$u_{j_0,j_1/P,\ldots,j_{d-1}}$$

- Partial transform over all but the second axis as

$$\tilde{u}_{k_0,k_1/P,\ldots,k_{d-1}} = \mathcal{F}_0\left(\mathcal{F}_2\left(\cdots \mathcal{F}_{d-1}(u_{j_0,j_1/P,\ldots,j_{d-1}})\right)\right)$$

Balanced block-contiguous decomposition

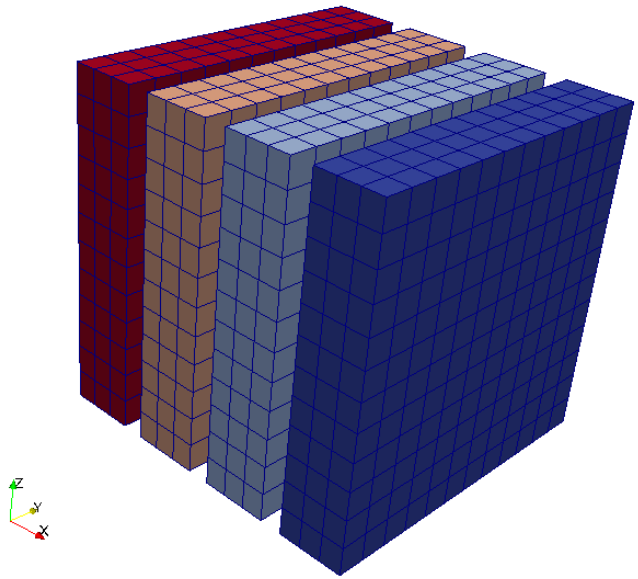
```
1:  function DECOMPOSE(N, M, p)
2:  input N : integer                    ▷ total number of elements, N ≥ 0
3:  input M : integer                    ▷ number of parts, M > 0
4:  input p : integer                    ▷ part index, 0 ≤ p < M
5:  output n : integer                   ▷ number of elements in p-th part
6:  output s : integer                   ▷ start index of p-th part
7:      q ← ⌊N/M⌋
8:      r ← N mod M
9:      if r > p then
10:         n ← q + 1
11:         s ← n · p
12:     else
13:         n ← q
14:         s ← n · p + r
15:     end if
16:     return n, s
17: end function
```

Global redistribution operations

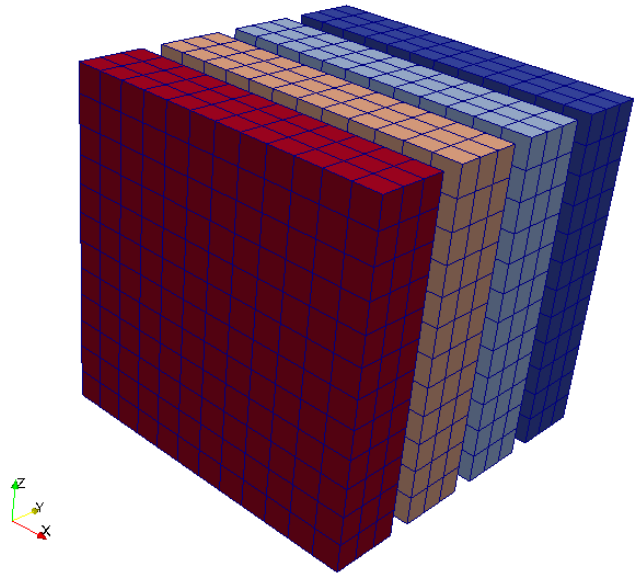
Redistribution from alignment in axis v to alignment in axis w performed within a process group P denoted as

$$u_{\ldots,j_w,\ldots,j_v/P,\ldots} \xleftarrow[P]{v \to w} u_{\ldots,j_w/P,\ldots,j_v,\ldots}$$

Slab decomposition



(a) Decompose in 0-direction



(b) Decompose in 1-direction

$$\tilde{u}_{j_0/P,k_1,\ldots,k_{d-1}} = \mathcal{F}_1\left(\mathcal{F}_2\left(\ldots\mathcal{F}_{d-1}(u_{j_0/P,j_1,\ldots,j_{d-1}})\right)\right)$$

$$\tilde{u}_{j_0,k_1/P,\ldots,k_{d-1}} \xleftarrow[P]{1\to 0} \tilde{u}_{j_0/P,k_1,\ldots,k_{d-1}}$$

$$\hat{u}_{k_0,k_1/P,\ldots,k_{d-1}} = \mathcal{F}_0(\tilde{u}_{j_0,k_1/P,\ldots,k_{d-1}})$$

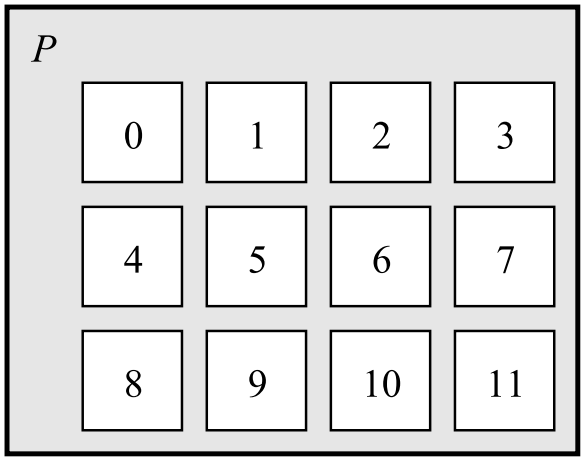
Subarray datatypes

```
1:  function SubArray(T, N, v, M)
2:  input T : datatype                          ▷ elementary datatype descriptor
3:  input N : sequence                  ▷ local sizes of a d-dimensional array
4:  input v : integer                          ▷ axis to partition, 0 ≤ v < d
5:  input M : integer                              ▷ number of parts, M > 0
6:  output S : sequence                      ▷ subarray datatype descriptors
7:      d ← len N
8:      for i ← 0, d − 1 do
9:          n(i) ← N(i)
10:         s(i) ← 0
11:     end for
12:     for p ← 0, M − 1 do
13:         n(v), s(v) ← Decompose(N(v), M, p)
14:         S(p) ← CreateSubArray(T, N, n, s)
15:     end for
16:     return S
17: end function
```

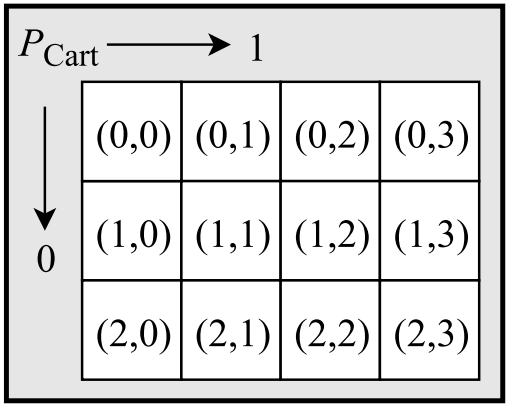
Exchange

1: **procedure** EXCHANGE(P, A, v, B, w)
2: **input** P : communicator ▷ group of communicating processes
3: **input** A : array ▷ local array of elementary datatype T
4: **input** v : integer ▷ axis of alignment for A
5: **output** B : array ▷ local array of elementary datatype T
6: **input** w : integer ▷ axis of alignment for B, $w \neq v$
7: $T \leftarrow$ TYPE(A) ▷ elementary datatype of array A
8: $N_A \leftarrow$ SHAPE(A) ▷ sequence with sizes of array A
9: $N_B \leftarrow$ SHAPE(B) ▷ sequence with sizes of array B
10: $M \leftarrow$ SIZE(P) ▷ number of processes in group
11: $S_A \leftarrow$ SUBARRAY(T, N_A, v, M) ▷ datatypes for sending
12: $S_B \leftarrow$ SUBARRAY(T, N_B, w, M) ▷ datatypes for receiving
13: ALLTOALLW(P, A, S_A, B, S_B) ▷ generalized all-to-all
14: **end procedure**

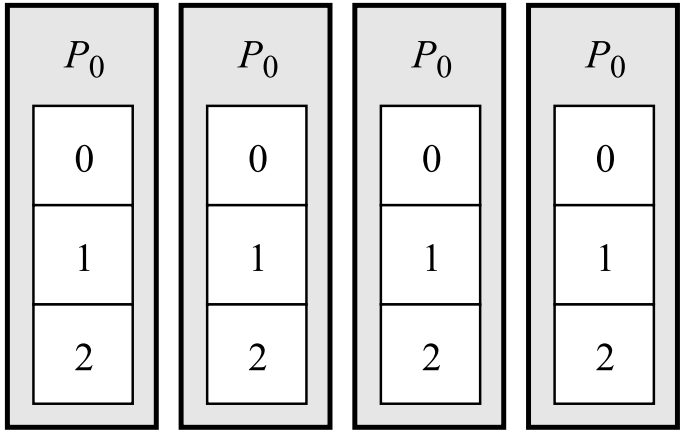
Cartesian process grid (2D) and 1D subgroups



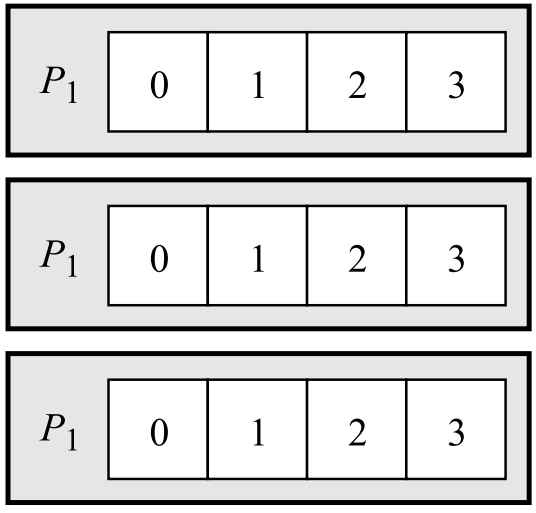
(a) Process group of 12 processes



(b) Cartesian grid of 3×4 processes

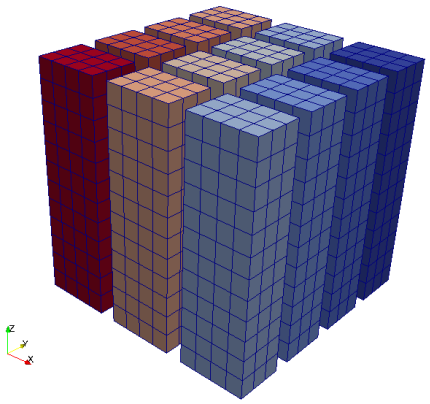


(a) Subgroups P_0 in first direction

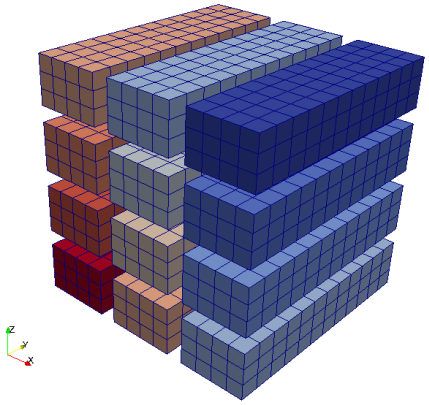


(b) Subgroups P_1 in second direction

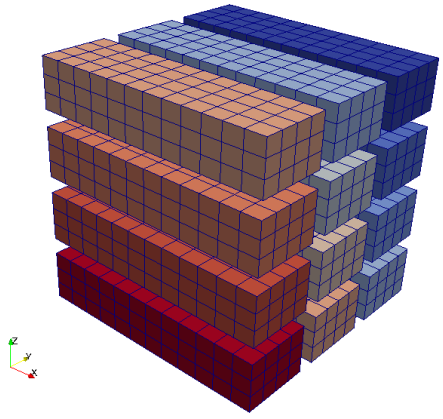
Pencil decomposition



(a) Align in 2-direction



(b) Align in 1-direction



(c) Align in 0-direction

$$\tilde{u}_{j_0/P_0,j_1/P_1,k_2} = \mathcal{F}_2\left(u_{j_0/P_0,j_1/P_1,j_2}\right)$$

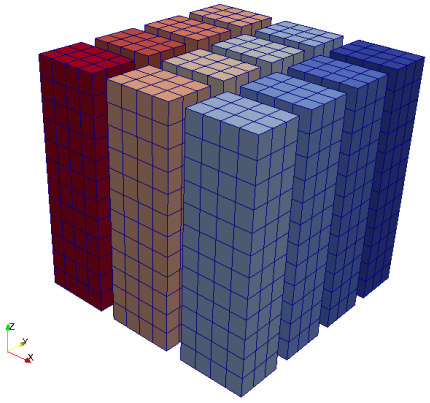
$$\tilde{u}_{j_0/P_0,j_1,k_2/P_1} \xleftarrow[P_1]{2\to 1} \tilde{u}_{j_0/P_0,j_1/P_1,k_2}$$

$$\tilde{u}_{j_0/P_0,k_1,k_2/P_1} = \mathcal{F}_1(\tilde{u}_{j_0/P_0,j_1,k_2/P_1})$$

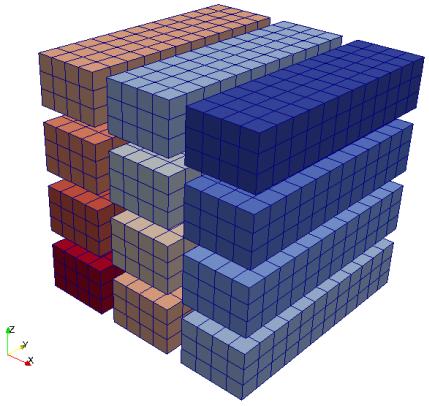
$$\tilde{u}_{j_0,k_1/P_0,k_2/P_1} \xleftarrow[P_0]{1\to 0} \tilde{u}_{j_0/P_0,k_1,k_2/P_1}$$

$$\hat{u}_{k_0,k_1/P_0,k_2/P_1} = \mathcal{F}_0(\tilde{u}_{j_0,k_1/P_0,k_2/P_1})$$

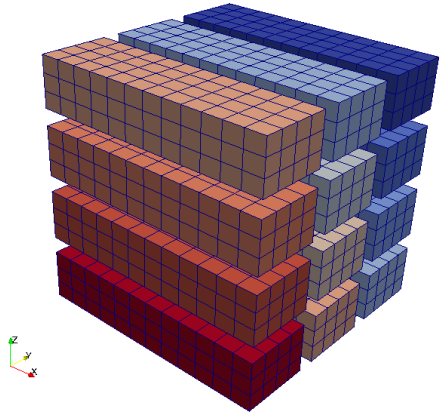
Pencils are a just a bunch of slabs!



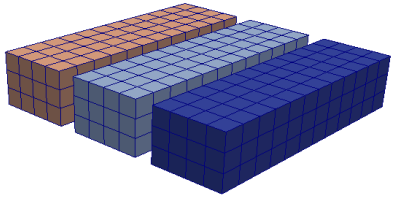
(a)



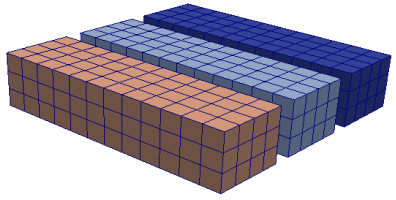
(b)



(c)



Top subslab of (b)



Top subslab of (c)

Implementation - Core routines (dimension-independent!)

```c
#define min(x, y) (((x) < (y)) ? (x) : (y))
void decompose(int N, int M, int p, int *n, int *s)
{
  int q = N / M;
  int r = N % M;
  *n = q + (r > p);
  *s = q * p + min(r, p);
}
```

```c
void subcomm(MPI_Comm comm,
             int      ndims,
             MPI_Comm subcomms[ndims])
{
  MPI_Comm comm_cart;
  int nprocs, dims[ndims], periods[ndims], remdims[ndims];
  for (int i = 0; i < ndims; i++)
    { dims[i] = periods[i] = remdims[i] = 0; }
  MPI_Comm_size(comm, &nprocs);
  MPI_Dims_create(nprocs, ndims, dims);
  MPI_Cart_create(comm, ndims, dims, periods, 1, &comm_cart);
  for (int i = 0; i < ndims; i++) {
    remdims[i] = 1;
    MPI_Cart_sub(comm_cart, remdims, &subcomms[i]);
    remdims[i] = 0;
  }
  MPI_Comm_free(&comm_cart);
}
```

```c
void subarray(MPI_Datatype datatype,
              int          ndims,
              int          sizes[ndims],
              int          axis,
              int          nparts,
              MPI_Datatype subarrays[nparts])
{
  int subsizes[ndims], substarts[ndims], n, s;
  for (int i = 0; i < ndims; i++)
    { subsizes[i] = sizes[i]; substarts[i] = 0; }
  for (int p = 0; p < nparts; p++) {
    decompose(sizes[axis], nparts, p, &n, &s);
    subsizes[axis] = n; substarts[axis] = s;
    MPI_Type_create_subarray(
        ndims, sizes, subsizes, substarts,
        MPI_ORDER_C, datatype, &subarrays[p]);
    MPI_Type_commit(&subarrays[p]);
  }
}
```

```c
void exchange(MPI_Comm     comm,
              MPI_Datatype datatype,
              int ndims,
              int sizesA[ndims], void *arrayA, int axisA,
              int sizesB[ndims], void *arrayB, int axisB)
{
  int nparts;
  MPI_Comm_size(comm, &nparts);
  MPI_Datatype subarraysA[nparts], subarraysB[nparts];
  subarray(datatype, ndims, sizesA, axisA, nparts, subarraysA);
  subarray(datatype, ndims, sizesB, axisB, nparts, subarraysB);
  int counts[nparts], displs[nparts];
  for (int p = 0; p < nparts; p++)
    { counts[p] = 1; displs[p] = 0; }
  MPI_Alltoallw(arrayA, counts, displs, subarraysA,
                arrayB, counts, displs, subarraysB, comm);
  for (int p = 0; p < nparts; p++) {
    MPI_Type_free(&subarraysA[p]);
    MPI_Type_free(&subarraysB[p]);
  }
}
```

Full 3D complex FFT with 2D pencil decomposition

```
16  // External routine in charge of computing
17  // multidimensional complex FFTs in-place
18  enum { FORWARD=-1, BACKWARD=+1 };
19  extern void seqxfftn(int ndims, int sizes[ndims],
20                       double complex *array,
21                       int axis, int sign);

45    // Create subgroups from 2D process grid
46    MPI_Comm P[2];
47    subcomm(MPI_COMM_WORLD, 2, P);
48    // Define elementary MPI datatype
49    MPI_Datatype T = MPI_C_DOUBLE_COMPLEX;

63    // Forward FFT
64    seqxfftn(3, sizesA, arrayA, 2, FORWARD);
65    exchange(P[1], T, 3, sizesA, arrayA, 2, sizesB, arrayB, 1);
66    seqxfftn(3, sizesB, arrayB, 1, FORWARD);
67    exchange(P[0], T, 3, sizesB, arrayB, 1, sizesC, arrayC, 0);
68    seqxfftn(3, sizesC, arrayC, 0, FORWARD);

70    // Backward FFT
71    seqxfftn(3, sizesC, arrayC, 0, BACKWARD);
72    exchange(P[0], T, 3, sizesC, arrayC, 0, sizesB, arrayB, 1);
73    seqxfftn(3, sizesB, arrayB, 1, BACKWARD);
74    exchange(P[1], T, 3, sizesB, arrayB, 1, sizesA, arrayA, 2);
75    seqxfftn(3, sizesA, arrayA, 2, BACKWARD);
```

Full 4D complex FFT with 3D pencil decomposition

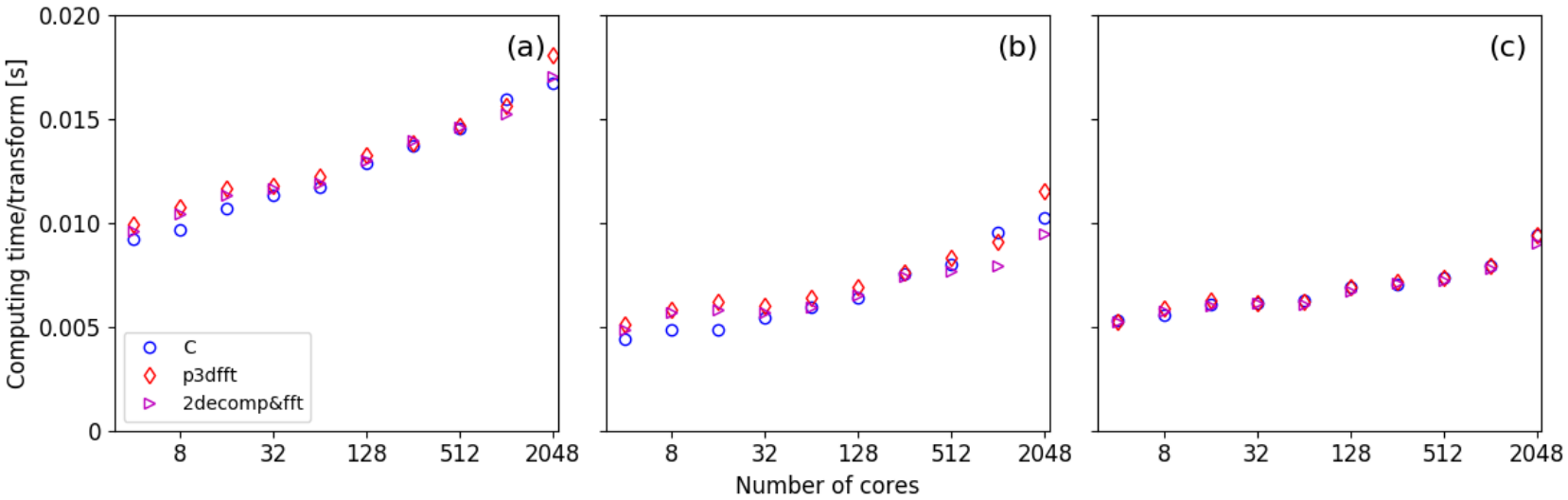
```
45    // Create subgroups from 3D process grid
46    MPI_Comm P[3];
47    subcomm(MPI_COMM_WORLD, 3, P);
48    // Define elementary MPI datatype
49    MPI_Datatype T = MPI_C_DOUBLE_COMPLEX;

71    // Forward FFT
72    seqxfftn(4, sizesA, arrayA, 3, FORWARD);
73    exchange(P[2], T, 3, sizesA, arrayA, 3, sizesB, arrayB, 2);
74    seqxfftn(4, sizesB, arrayB, 2, FORWARD);
75    exchange(P[1], T, 3, sizesB, arrayB, 2, sizesC, arrayC, 1);
76    seqxfftn(4, sizesC, arrayC, 1, FORWARD);
77    exchange(P[0], T, 3, sizesC, arrayC, 1, sizesD, arrayD, 0);
78    seqxfftn(4, sizesD, arrayD, 0, FORWARD);

80    // Backward FFT
81    seqxfftn(4, sizesD, arrayD, 0, BACKWARD);
82    exchange(P[0], T, 3, sizesD, arrayD, 0, sizesC, arrayC, 1);
83    seqxfftn(4, sizesC, arrayC, 1, BACKWARD);
84    exchange(P[1], T, 3, sizesC, arrayC, 1, sizesB, arrayB, 2);
85    seqxfftn(4, sizesB, arrayB, 2, BACKWARD);
86    exchange(P[2], T, 3, sizesB, arrayB, 2, sizesA, arrayA, 3);
87    seqxfftn(4, sizesA, arrayA, 3, BACKWARD);
```

Performance: Weak scaling

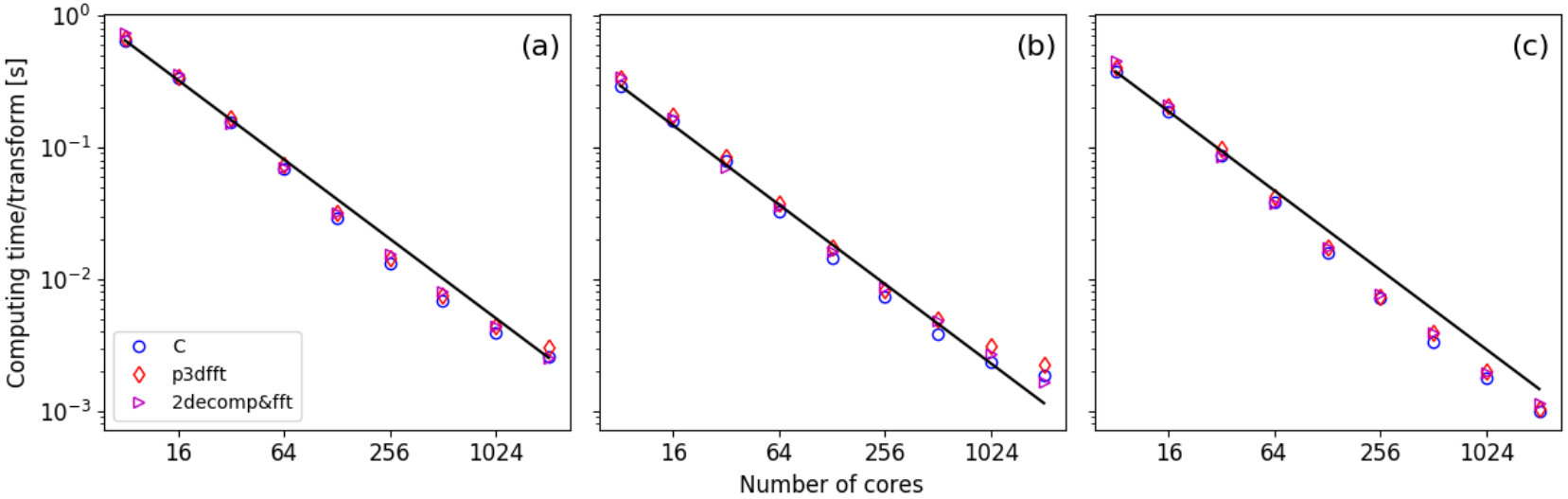
- 2D pencil decomposition, real-to-complex + complex-to-real
- $524,288$ entries per process, **1 core per node**
- Showing fastest times measured



(a) total time (b) redistribution (c) local FFTs

Performance: Strong scaling

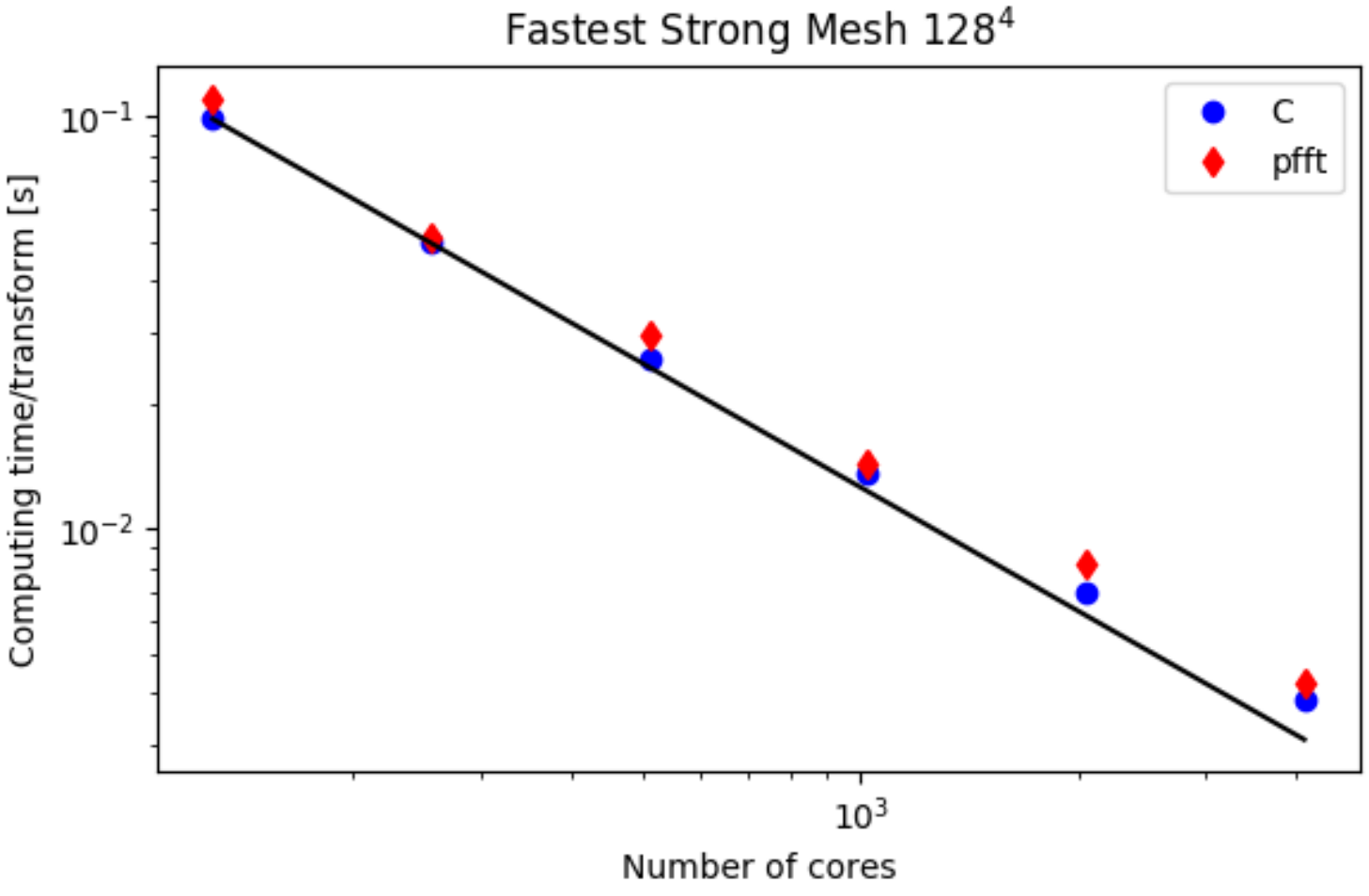
- 2D pencil decomposition, real-to-complex + complex-to-real
- Global size 512^3, **1 core per node**
- Showing fastest times measured



(a) total time (b) redistribution (c) local FFTs

Performance: Strong scaling (4D FFT on 3D pencil)

- Global size 128^4, real-to-complex + complex-to-real
- Showing fastest times measured



Resources

- **Fast parallel multidimensional FFT using advanced MPI**
  L. Dalcin, M. Mortensen, D.E. Keyes, JPDC 128:137-150
  `https://doi.org/10.1016/j.jpdc.2019.02.006`
  `https://arxiv.org/abs/1804.09536`
- **mpi4py-fft**: Distributed FFTs in Python
  `https://doi.org/10.21105/joss.01340`
  `https://mpi4py-fft.readthedocs.io`
  `https://bitbucket.org/mpi4py/mpi4py-fft`
- **shenfun**: Spectral Galerkin framework in Python
  `https://doi.org/10.21105/joss.01071`
  `https://shenfun.readthedocs.io`
  `https://github.com/spectralDNS/shenfun`