Parallel Fast Fourier Transform

Massimiliano Guarrasi

m.guarrasi@cineca.it

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt$$

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{-2\pi i f t} df$$

Frequency Domain ⟺ **Time Domain**

Real Space ⟺ **Reciprocal Space**

Properties

Convolution Theorem

$$\int_{-\infty}^{+\infty} g(\tau)h(t-\tau)d\tau \Leftrightarrow G(f)H(f)$$

Correlation Theorem

$$Corr(h,h) \equiv \int_{-\infty}^{+\infty} h(\tau)h(t+\tau)d\tau \Leftrightarrow |H(f)|^2$$

Power Spectrum

$$Corr(h,h) \equiv \int_{-\infty}^{+\infty} h(\tau)h(t+\tau)d\tau \Leftrightarrow |H(f)|^2$$

In many application contexts the Fourier transform is approximated with a Discrete Fourier Transform (DFT):

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt \approx \sum_{k=0}^{N-1} h_k e^{2\pi i f_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i f_n t_k}$$

$$f_n = n/\Delta$$

$$\begin{cases} t_k = \Delta k / N \\ f_n = n/\Delta \end{cases}$$

$$H(f_n) = \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}$$

The last expression is periodic, with period N. It define a between 2 sets of numbers , $\mathbf{H_n}$ & $\mathbf{h_k}$ ($\mathbf{H(f_n) = \Delta H_n}$)

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N} \qquad h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i k n / N}$$

frequencies from **0** to **fc** (maximum frequency) are mapped in the values with index from **0** to **N/2-1**, while negative ones are up to **-fc** mapped with index values of **N / 2** to **N**

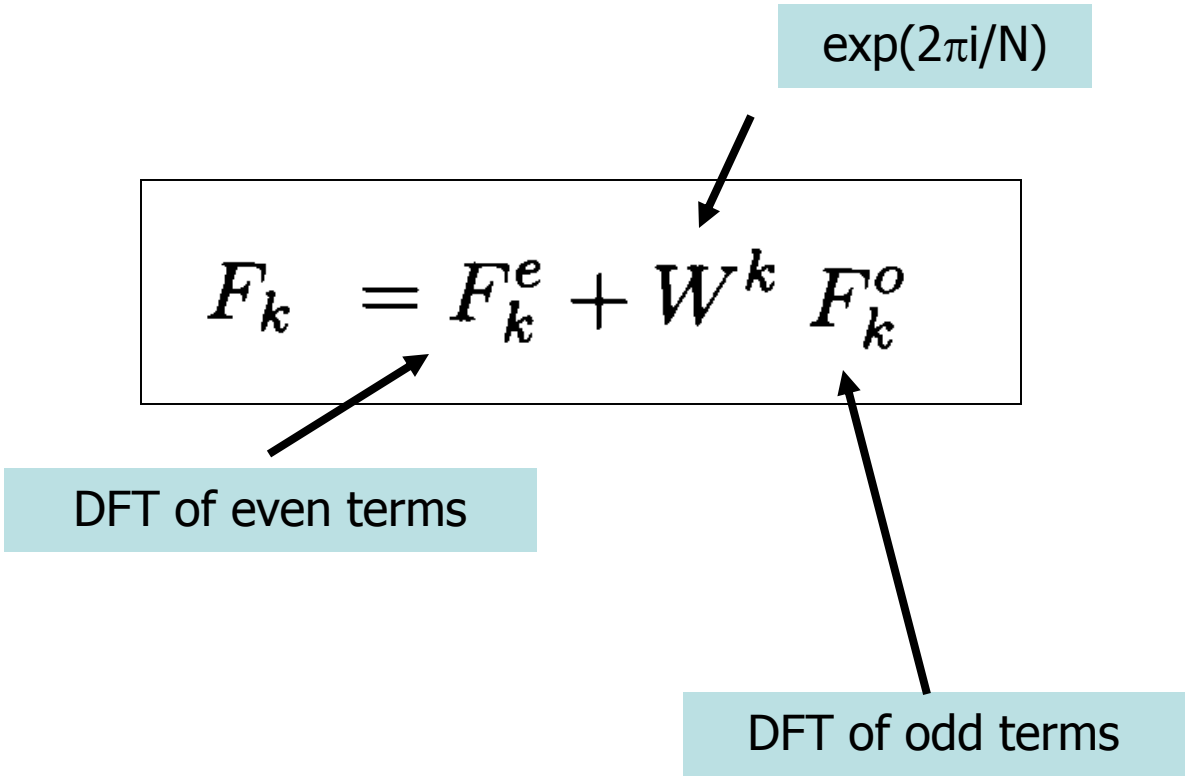
Scale like N*N

Fast Fourier Transform (FFT)

The DFT can be calculated very efficiently using the algorithm known as the FFT, which uses symmetry properties of the DFT s

$$
\begin{aligned}
F_k &= \sum_{j=0}^{N-1} e^{2\pi i j k/N} f_j \\
&= \sum_{j=0}^{N/2-1} e^{2\pi i k(2j)/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i k(2j+1)/N} f_{2j+1} \\
&= \sum_{j=0}^{N/2-1} e^{2\pi i k j/(N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi i k j/(N/2)} f_{2j+1} \\
&= F_k^e + W^k \, F_k^o
\end{aligned}
$$

Fast Fourier Transform (FFT)



Now Iterate:

$$F^{e} = F^{ee} + W^{k/2} F^{eo}$$

$$F^{o} = F^{oe} + W^{k/2} F^{oo}$$

You obtain a series for each value of f_n

$$F^{oeoeooeo..oe} = f_n$$

Scale like N*logN (binary tree)

http://www.fftw.org

Download Mailing List Benchmark Features Documentation FAQ Links Feedback

Introduction

FFTW is a C subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions, of both real and complex data, and of arbitrary input size. We believe that FFTW, which is free software, should become the FFT library of choice for most applications. Our benchmarks, performed on on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software. Moreover, FFTW's performance is *portable*: the program will perform well on most architectures without modification.

It is difficult to summarize in a few words all the complexities that arise when testing many programs, and there is no "best" or "fastest" program. However, FFTW appears to be the fastest program most of the time for in-order transforms, especially in the multi-dimensional and real-complex cases (Kasparov is the best chess player in the world even though he loses some games). Hence the name, "FFTW," which stands for the somewhat whimsical title of "Fastest Fourier Transform in the West." Please visit the benchFFT home page for a more extensive survey of the results.

The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson.

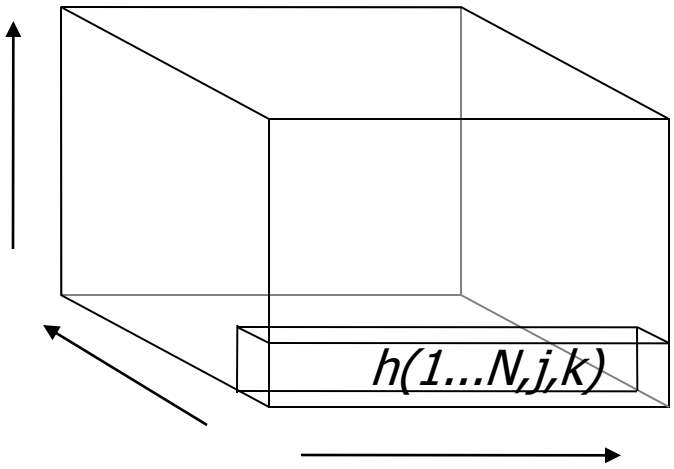
- Written in C
- Fortran wrapper is also provided
- FFTW adapt itself to your machines, your cache, the size of your memory, the number of register, etc...
- FFTW doesn't use a fixed algorithm to make DFT
  - FFTW chose the best algorithm for your machines
- Computation is split in 2 phases:
  - PLAN creation
  - Execution
- FFTW support transforms of data with arbitrary length, rank, multiplicity, and memory layout, and more....

- Many different versions:
  - FFTW 2:
    - Released in 2003
    - Well tested and used in many codes
    - Includes serial and parallel transforms for both shared and distributed memory system
  - FFTW 3:
    - Released in February 2012
    - Includes serial and parallel transforms for both shared and distributed memory system
    - Hybrid implementation MPI-OpenMP

Multi dimensional FFT

$$H(n_1, n_2) = \text{FFT-on-index-1}\left(\text{FFT-on-index-2}\left[h(k_1, k_2)\right]\right)$$

$$= \text{FFT-on-index-2}\left(\text{FFT-on-index-1}\left[h(k_1, k_2)\right]\right)$$



*1) For each value of **j** and **k***

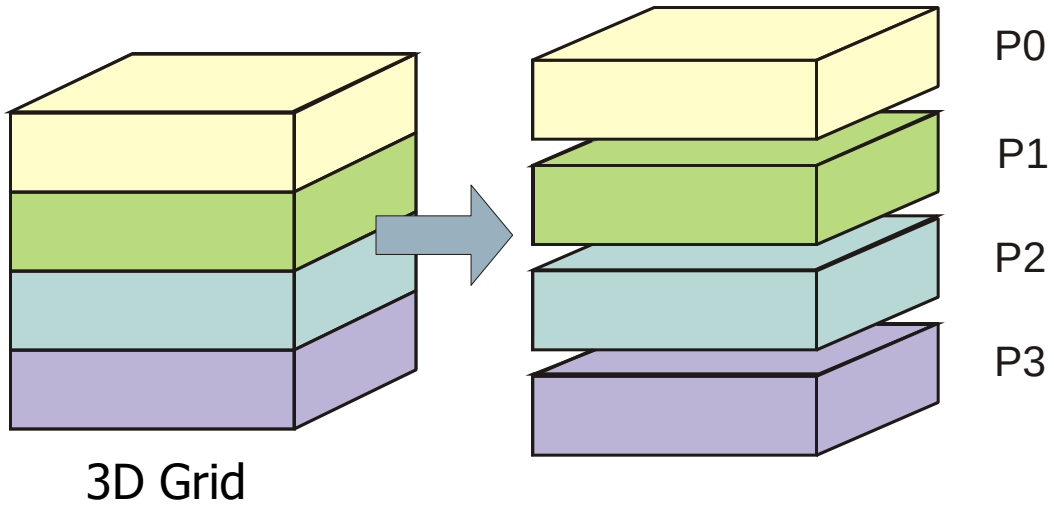
Apply FFT to h(1...N,j,k)

*2) For each value of **i** and **k***

Apply FFT to h(i,1...N,k)

*3) For each value of **i** and **j***
Apply FFT to h(i,j,1...N)

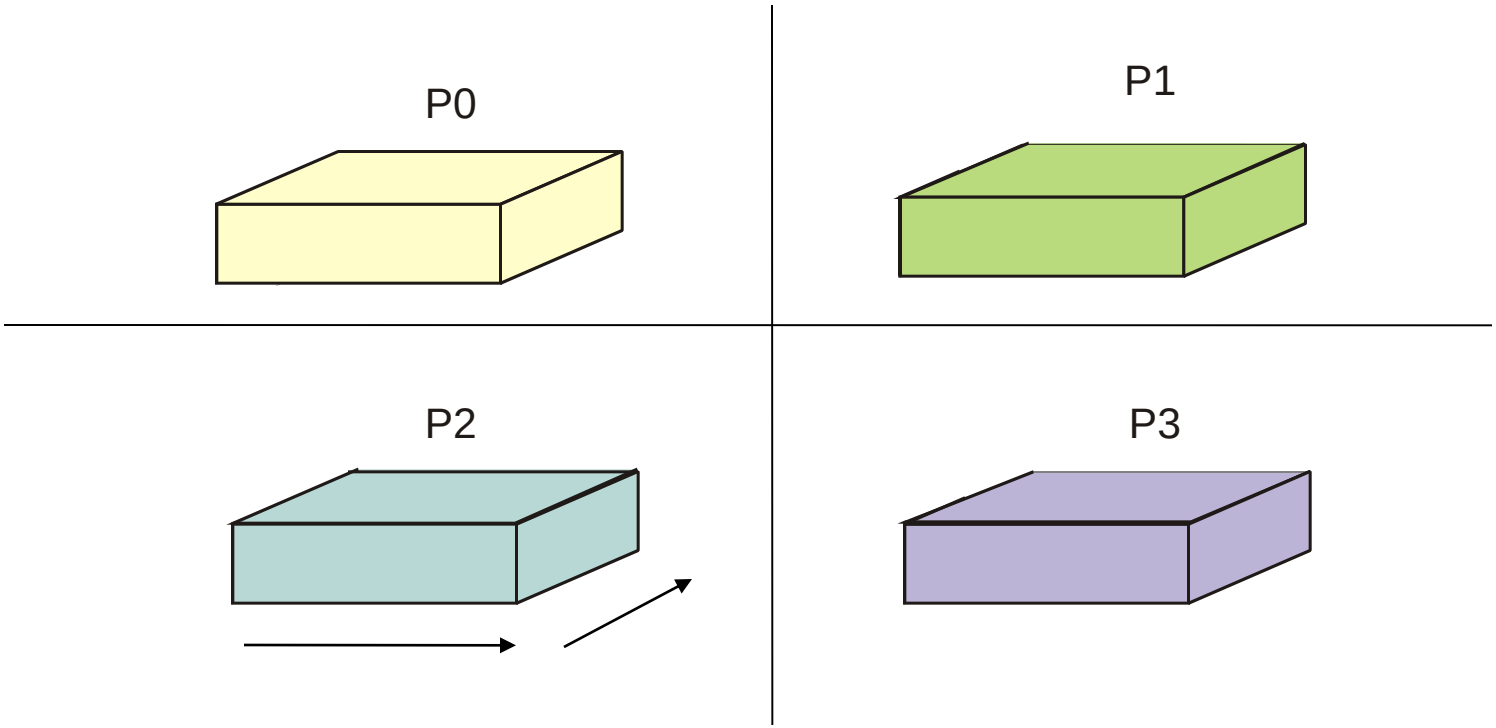
Parallel FFT Data Distribution



Distribute data along one coordinate (e.g. *Z*)

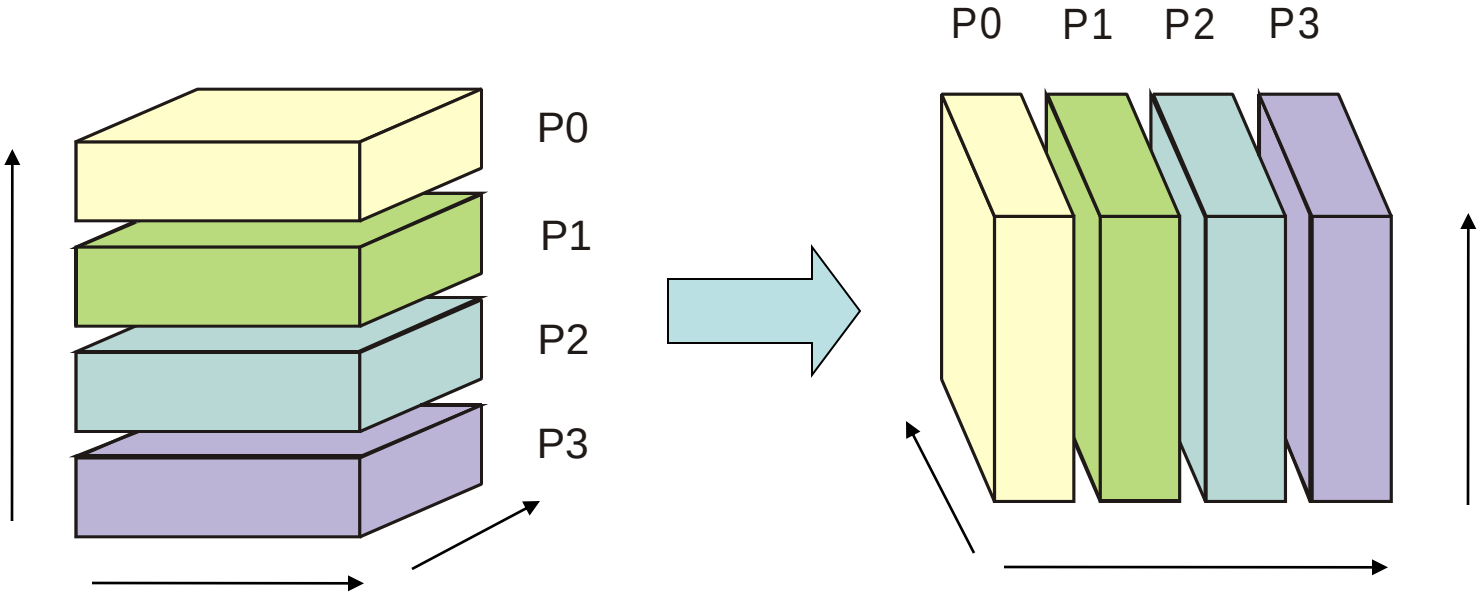
This is know as "Slab Decomposition"

Transform along x and y



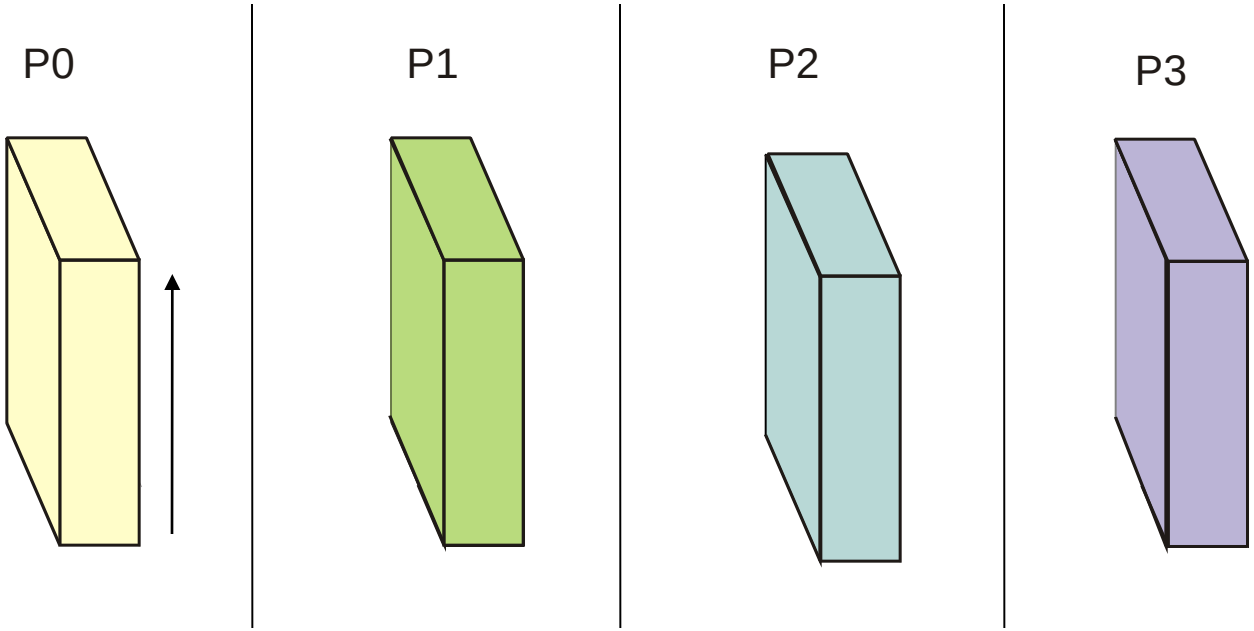
each processor trasform its own sub-grid along the x and y independently of the other

Data redistribution involving x and z



The data are now distributed along x

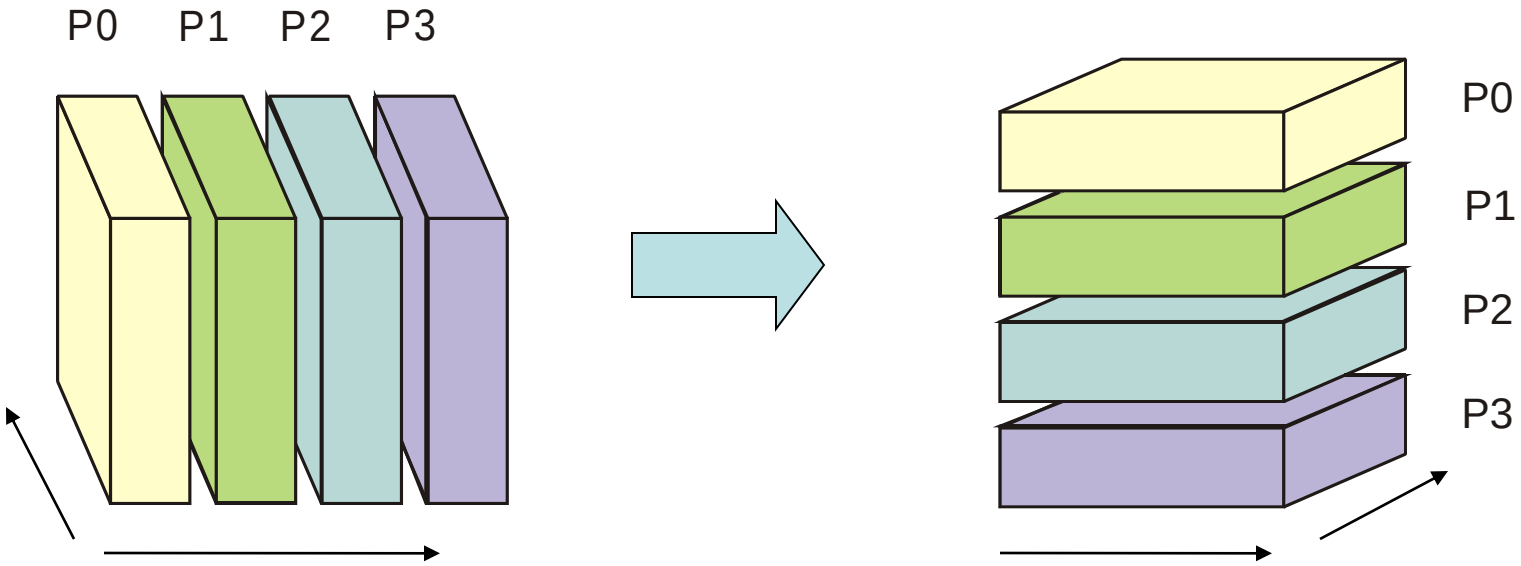
FFT along z



each processor transform its own sub-grid

along the z dimension independently of the other

Data are re-distributed, back from x to z



The 3D array now has the original layout, but each element

Has been substituted with its FFT.

Some Useful Instructions

Compile a FFTW program on FERMI:

- Module Loading:

  module load fftw/3.3.3—bgq-xl—1.0

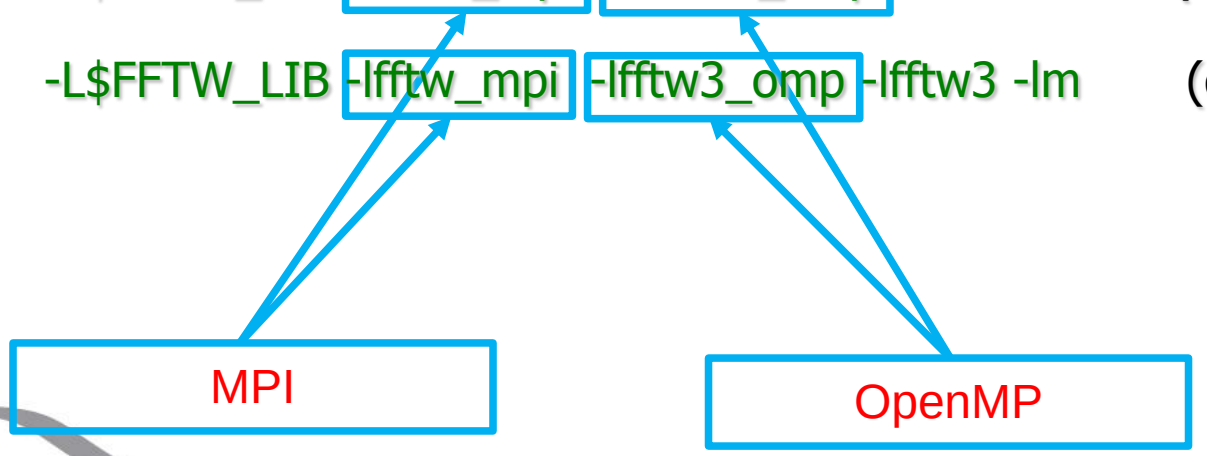
- Including header:

  - -I$FFTW_INC

- Linking:

  -L$FFTW_LIB -lfftwf_mpi -lfftw3f_omp -lfftw3f -lm (single precision)

  -L$FFTW_LIB -lfftw_mpi -lfftw3_omp -lfftw3 -lm (double precision)



INCLUDING FFTW Lib:

Serial

- C:

```
#include <fftw.h>
```

- FORTRAN:

```
include 'fftw3.f03‘
```

Parallel (Pure MPI)

- C:

```
#include <fftw-mpi.h>
```

- FORTRAN:

```
include 'fftw3-mpi.f03'
```

Some important Remarks for FORTRAN users

- Function in C became function in FORTRAN if they have a return value, and subroutines otherwise.
- All C types are mapped via the iso_c_binning standard.
- FFTW plans are type(C_PTR) in FORTRAN.
- The ordering of FORTRAN array dimensions must be reversed when they are passed to the FFTW plan creation

Initializzation

Including FFTW Lib:

- C:
  - Serial:

    #include <fftw.h>
  - MPI:

    include 'fftw3.f03'
- FORTRAN:
  - Serial:

    #include <fftw-mpi.h>
  - MPI:

    include 'fftw3-mpi.f03'

MPI initializzation:

- C:

void fftw_mpi_init(void)

- FORTRAN:

fftw_mpi_init()

Create arrays

C:

- Fixed size array:

```
fftx_complex data[n0][n1][n2]
```

- Dynamic array:

```
data = fftw_alloc_complex(n0*n1*n2)
```

- MPI dynamic arrays:

```
fftw_complex *data
ptrdiff_t alloc_local, local_no, local_no_start
alloc_local= fftw_mpi_local_size_3d(n0, n1, n2, MPI_COMM_WORLD, &local_n0,&local_n0_start)
data = fftw_alloc_complex(alloc _local)
```

FORTRAN:

- Fixed size array (simplest way):

```
complex(C_DOUBLE_COMPLEX), dimension(n0,n1,n2) :: data
```

- Dynamic array (simplest way):

```
complex(C_DOUBLE_COMPLEX), allocatable, dimension(:, :, :) :: data
allocate (data(n0, n1, n2))
```

- Dynamic array (fastest method):

```
complex(C_DOUBLE_COMPLEX), pointer :: data(:, :, :) )
type(C_PTR) :: cdata
cdata = fftw_alloc_complex(n0*n1*n2)
call c_f_pointer(cdata, data, [n0,n1,n2])
```

- MPI dynamic arrays:

```
complex(C_DOUBLE_COMPLEX), pointer :: data(:, :, :)
type(C_PTR) :: cdata
integer(C_INTPTR_T) :: alloc_local, local_n2, local_n2_offset
alloc_local = fftw_mpi_local_size_3d(n2, n1, n0, MPI_COMM_WORLD, local_n2, local_n2_offset)
cdata = fftw_alloc_complex(alloc_local)
call c_f_pointer(cdata, data, [n0,n1,local_n2])
```

Plan Creation (C2C)

1D Complex to complex DFT:

• C:

fftw_plan = fftw_plan_dft_1d(int nx, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)

•FORTRAN:

plan = ftw_plan_dft_1d(nz, in, out, dir, flags)

FFTW_FORWARD
FFTW_BACKWARD

FFTW_ESTIMATE
FFTW_MEASURE

3D Complex to complex DFT:

• C:

fftw_plan = fftw_plan_dft_2d(int nx, int ny, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)

fftw_plan = fftw_mpi_plan_dft_2d(int nx, int ny, fftw_complex *in, fftw_complex *out,MPI_COMM_WORLD, fftw_direction dir, int flags)

•FORTRAN:

plan = ftw_plan_dft_2d(ny, nx, in, out, dir, flags)

plan = ftw_mpi_plan_dft_2d(ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

3D Complex to complex DFT:

• C:

fftw_plan = fftw_plan_dft_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)

fftw_plan = fftw_mpi_plan_dft_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out,MPI_COMM_WORLD, fftw_direction dir, int flags)

•FORTRAN:

plan = ftw_plan_dft_3d(nz, ny, nx, in, out, dir, flags)

plan = ftw_mpi_plan_dft_3d(nz, ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

Plan Creation (R2C)

1D Complex to complex DFT:

• C:

fftw_plan = fftw_plan_dft_r2c_1d(int nx, double *in, fftw_complex *out, fftw_direction dir, unsigned flags)

•FORTRAN:

ftw_plan_dft_r2c_1d(nz, in, out, dir, flags)

FFTW_FORWARD
FFTW_BACKWARD

FFTW_ESTIMATE
FFTW_MEASURE

3D Complex to complex DFT:

• C:

fftw_plan = fftw_plan_dft_r2c_2d(int nx, int ny, double *in, fftw_complex *out, fftw_direction dir, unsigned flags)

fftw_plan = fftw_mpi_plan_dft_r2c_2d(int nx, int ny, double *in, fftw_complex *out,MPI_COMM_WORLD, fftw_direction dir, int flags)

•FORTRAN:

ftw_plan_dft_r2c_2d(ny, nx, in, out, dir, flags)

ftw_mpi_plan_dft_r2c_2d(ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

3D Complex to complex DFT:

• C:

fftw_plan = fftw_plan_dft_r2c_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)

fftw_plan = fftw_mpi_plan_dft_r2c_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out,MPI_COMM_WORLD, fftw_direction dir, int flags)

•FORTRAN:

ftw_plan_dft_r2c_3d(nz, ny, nx, in, out, dir, flags)

ftw_mpi_plan_dft_r2c_3d(nz, ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

Plan Execution

Complex to complex DFT:

• C:

void fftw_execute_dft(fftw_plan plan, fftw_complex *in, fftw_complex *out)
void fftw_mpi_execute_dft (fftw_plan plan, fftw_complex *in, fftw_complex *out)

•FORTRAN:

fftw_execute_dft (plan, in, out)
fftw_mpi_execute_dft (plan, in, out)

Complex to complex DFT:

• C:

void fftw_execute_dft (fftw_plan plan, double *in, fftw_complex *out)
void fftw_mpi_execute_dft (fftw_plan plan, double *in, fftw_complex *out)

•FORTRAN:

fftw_execute_dft (plan, in, out)
Fftw_mpi_execute_dft (plan, in, out)

Finalizing FFTW

Destroying PLAN:

- C:

```
void  fftw_destroy_plan(fftw_plan plan)
```

- FORTRAN:

```
fftw_destroy_plan(plan)
```

FFTW MPI cleanup:

- C:

```
void  fftw_mpi_cleanup ()
```

- FORTRAN:

```
fftw_mpi_cleanup ()
```

Deallocate data:

- C:

```
void  fftw_free (fftw_complex data)
```

- FORTRAN:

```
fftw_free (data)
```

Some Useful Examples

1D Serial FFT - Fortran


```
 program FFTW1D
       use, intrinsic :: iso_c_binding
       implicit none
       include 'fftw3.f03'
       integer(C_INTPTR_T):: L = 1024
       integer(C_INT) :: LL
       type(C_PTR) :: plan1
       complex(C_DOUBLE_COMPLEX), dimension(1024) :: idata, odata
       integer :: i
       character(len=41), parameter :: filename='serial_data.txt'
       LL = int(L,C_INT)
!! create MPI plan for in-place forward DF
        plan1 = fftw_plan_dft_1d(LL, idata, odata, FFTW_FORWARD, FFTW_ESTIMATE)
!! initialize data
        do i = 1, L
          if (i .le. (L/2)) then
            idata(i) = (1.,0.)
          else
            idata(i) = (0.,0.)
          endif
        end do
!! compute transform (as many times as desired)
        call fftw_execute_dft(plan1, idata, odata)
!! deallocate and destroy plans
        call fftw_destroy_plan(plan1)
      end
```

1D Serial FFT - Fortran

Summer School on PARALLEL COMPUTING

```
 program FFTW1D
 use, intrinsic :: iso_c_binding
       implicit none
       include 'fftw3.f03'
       integer(C_INTPTR_T):: L = 1024
       integer(C_INT) :: LL
       type(C_PTR) :: plan1
       type(C_PTR) :: p_idata, p_odata
       complex(C_DOUBLE_COMPLEX), dimension(:), pointer :: idata,odata
       integer :: i
!! Allocate
         LL = int(L,C_INT)
         p_idata = fftw_alloc_complex(L)
         p_odata = fftw_alloc_complex(L)
         call c_f_pointer(p_idata,idata,(/L/))
         call c_f_pointer(p_odata,odata,(/L/))
!!   create MPI plan for in-place forward DF
         plan1 = fftw_plan_dft_1d(LL, idata, odata, FFTW_FORWARD, FFTW_ESTIMATE)
!! initialize data
        do i = 1, L
           if (i .le. (L/2)) then
              idata(i) = (1.,0.)
           else
              idata(i) = (0.,0.)
           endif
        end do
!! compute transform (as many times as desired)
        call fftw_execute_dft(plan1, idata, odata)
!! deallocate and destroy plans
        call fftw_destroy_plan(plan1)
        call fftw_free(p_idata)
        call fftw_free(p_odata)
     end
```

CINECA

1D Serial FFT - C


```c
# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <fftw3.h>

int main ( void )

{
  ptrdiff_t i;
  const ptrdiff_t n = 1024;
  fftw_complex *in;
  fftw_complex *out;
  fftw_plan plan_forward;
/* Create arrays. */
   in = fftw_malloc ( sizeof ( fftw_complex ) * n );
   out = fftw_malloc ( sizeof ( fftw_complex ) * n );
/* Initialize data */
   for ( i = 0; i < n; i++ )   {
      if (i <= (n/2-1))   {
         in[i][0] = 1.;
         in[i][1] = 0.;
       }
      else {
         in[i][0] = 0.;
         in[i][1] = 0.;
       }
   }
/* Create plans. */
   plan_forward = fftw_plan_dft_1d ( n, in, out, FFTW_FORWARD, FFTW_ESTIMATE );
/* Compute transform (as many times as desired) */
   fftw_execute ( plan_forward );
/* deallocate and destroy plans */
   fftw_destroy_plan ( plan_forward );
   fftw_free ( in );
   fftw_free ( out );
  return 0;
}
```

2D Parallel FFT – Fortran (part 1)


```
program FFT_MPI_3D
      use, intrinsic :: iso_c_binding
      implicit none
            include 'mpif.h'
      include 'fftw3-mpi.f03'
      integer(C_INTPTR_T), parameter :: L = 1024
      integer(C_INTPTR_T), parameter :: M = 1024
      type(C_PTR) :: plan, cdata
      complex(C_DOUBLE_COMPLEX), pointer :: fdata(:,:)
      integer(C_INTPTR_T) :: alloc_local, local_M, local_j_offset
      integer(C_INTPTR_T) :: i, j
      complex(C_DOUBLE_COMPLEX) :: fout
      integer :: ierr, myid, nproc
! Initialize
            call mpi_init(ierr)
            call MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
            call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
            call fftw_mpi_init()
!   get local data size and allocate (note dimension reversal)
            alloc_local = fftw_mpi_local_size_2d(M, L, MPI_COMM_WORLD, local_M, local_j_offset)
            cdata = fftw_alloc_complex(alloc_local)
            call c_f_pointer(cdata, fdata, [L,local_M])
!   create MPI plan for in-place forward DFT (note dimension reversal)
      plan = fftw_mpi_plan_dft_2d(M, L, fdata, fdata, MPI_COMM_WORLD, FFTW_FORWARD, FFTW_MEASURE)
```

2D Serial FFT – Fortran (part 2)


```
! initialize data to some function my_function(i,j)
                do j = 1, local_M
                  do i = 1, L
                    call initial(i, (j + local_j_offset), L, M, fout)
                    fdata(i, j) = fout
                  end do
                end do
! compute transform (as many times as desired)
                call fftw_mpi_execute_dft(plan, fdata, fdata)!
! deallocate and destroy plans
                call fftw_destroy_plan(plan)
                call fftw_mpi_cleanup()
                call fftw_free(cdata)
                call mpi_finalize(ierr)
        end
```

2D Serial FFT – C

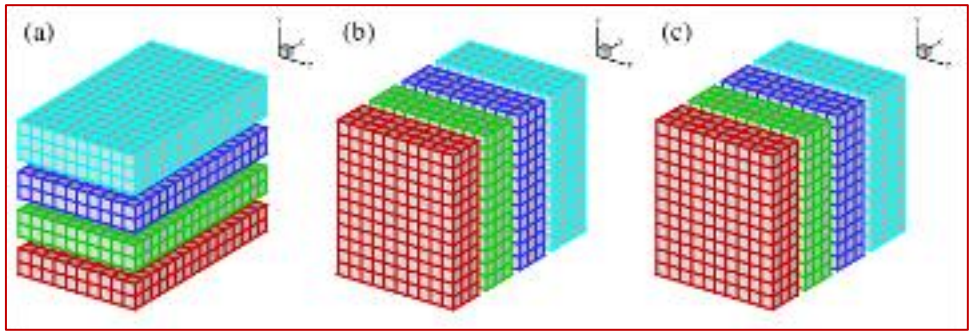

```
# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <mpi.h>
# include <fftw3-mpi.h>

    int main(int argc, char **argv)
    {
        const ptrdiff_t L = 1024, M = 1024;
        fftw_plan plan;
        fftw_complex *data ;
        ptrdiff_t alloc_local, local_L, local_L_start, i, j, ii;
        double xx, yy, rr, r2, t0, t1, t2, t3, tplan, texec;
        const double amp = 0.25;
        /* Initialize */
        MPI_Init(&argc, &argv);
        fftw_mpi_init();

        /* get local data size and allocate */
        alloc_local = fftw_mpi_local_size_2d(L, M, MPI_COMM_WORLD, &local_L, &local_L_start);
        data = fftw_alloc_complex(alloc_local);
        /* create plan for in-place forward DFT */
       plan = fftw_mpi_plan_dft_2d(L, M, data, data, MPI_COMM_WORLD, FFTW_FORWARD, FFTW_ESTIMATE);
```

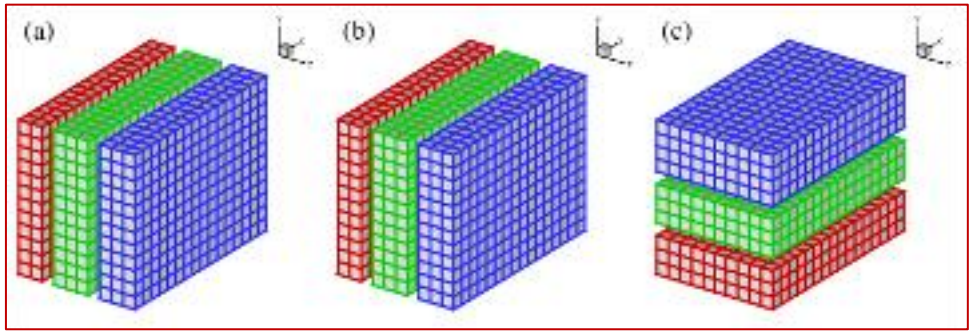
2D Serial FFT – C


```
/* initialize data to some function my_function(x,y) */
/*……..*/
/* compute transforms, in-place, as many times as desired */
        fftw_execute(plan);
/* deallocate and destroy plans */
        fftw_destroy_plan(plan);
        fftw_mpi_cleanup();
        fftw_free ( data );
        MPI_Finalize();
    }
```

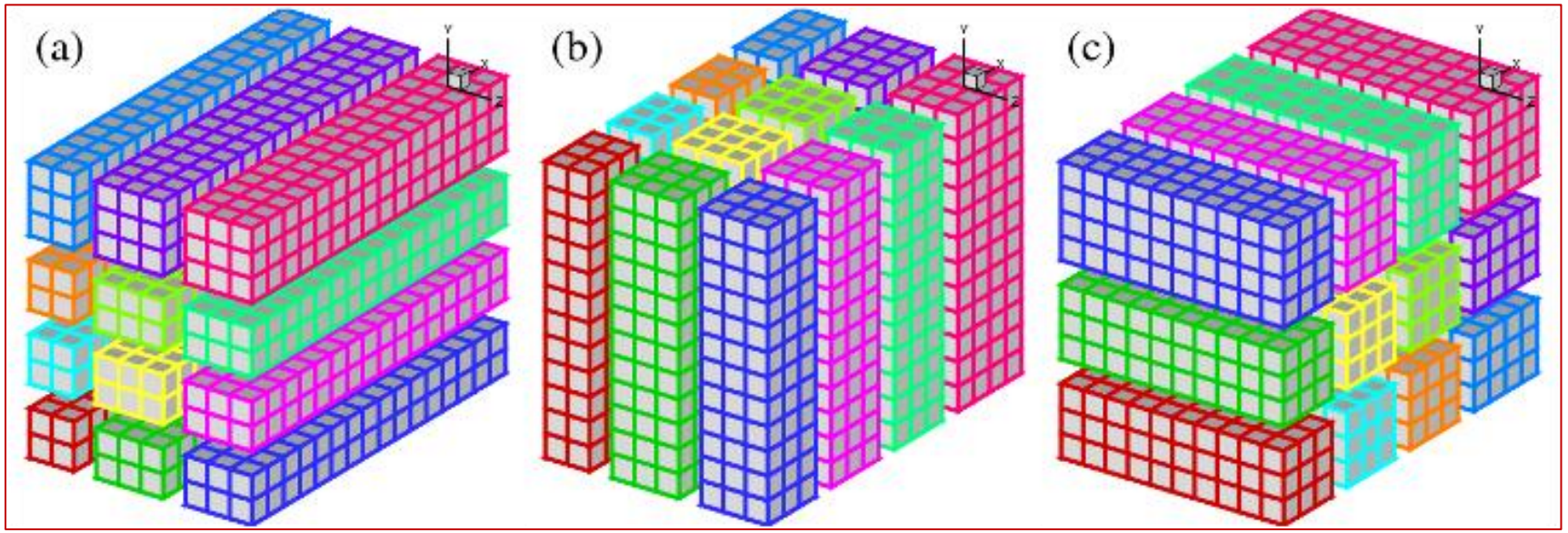
LIMITATIONS

Slab Decomposition





2D Domain Decomposition



2DECOMP & FFT

A FORTRAN Library that use 2D Domain Decomposition

Compile a FFTW program on FERMI:

- Module Loading:

```
module load profile/advanced

module load autoload 2Decomp_fft/1.5.847--bgq-xl--1.0
```

- Including header:

```
-I$FFTW_INC  -I$DECOMP_2D_FFT_LIB
```

- Linking:

```
-L$DECOMP_2D_FFT_LIB  -L$FFTW_LIB  -l2decomp_fft  -lfftw3_mpi -lfftw3f -lfftw3 –lm
```

Some useful instructions

Including 2Decomp&FFT Lib:

```
use decomp_2d
use decomp_2d_fft
```

Initial declarations:

```
integer, parameter :: p_row = ..
integer, parameter :: p_col = ...
complex(mytype), allocatable, dimension(:,:,:) :: in, out
```

Please note that: p_row*p_col = N_core

Initialization:

```
call decomp_2d_init(n0,n1,n2,p_row,p_col)
call decomp_2d_fft_init
```

Allocation:

```
allocate (in(xstart(1):xend(1),xstart(2):xend(2),xstart(3):xend(3)))
allocate (out(zstart(1):zend(1),zstart(2):zend(2),zstart(3):zend(3)))
```

Execution:

```
call decomp_2d_fft_3d(in, out, DECOMP_2D_FFT_FORWARD)
```

Finalizing:

```
call decomp_2d_fft_finalize
call decomp_2d_finalize
```

2DECOMP & FFT

An example

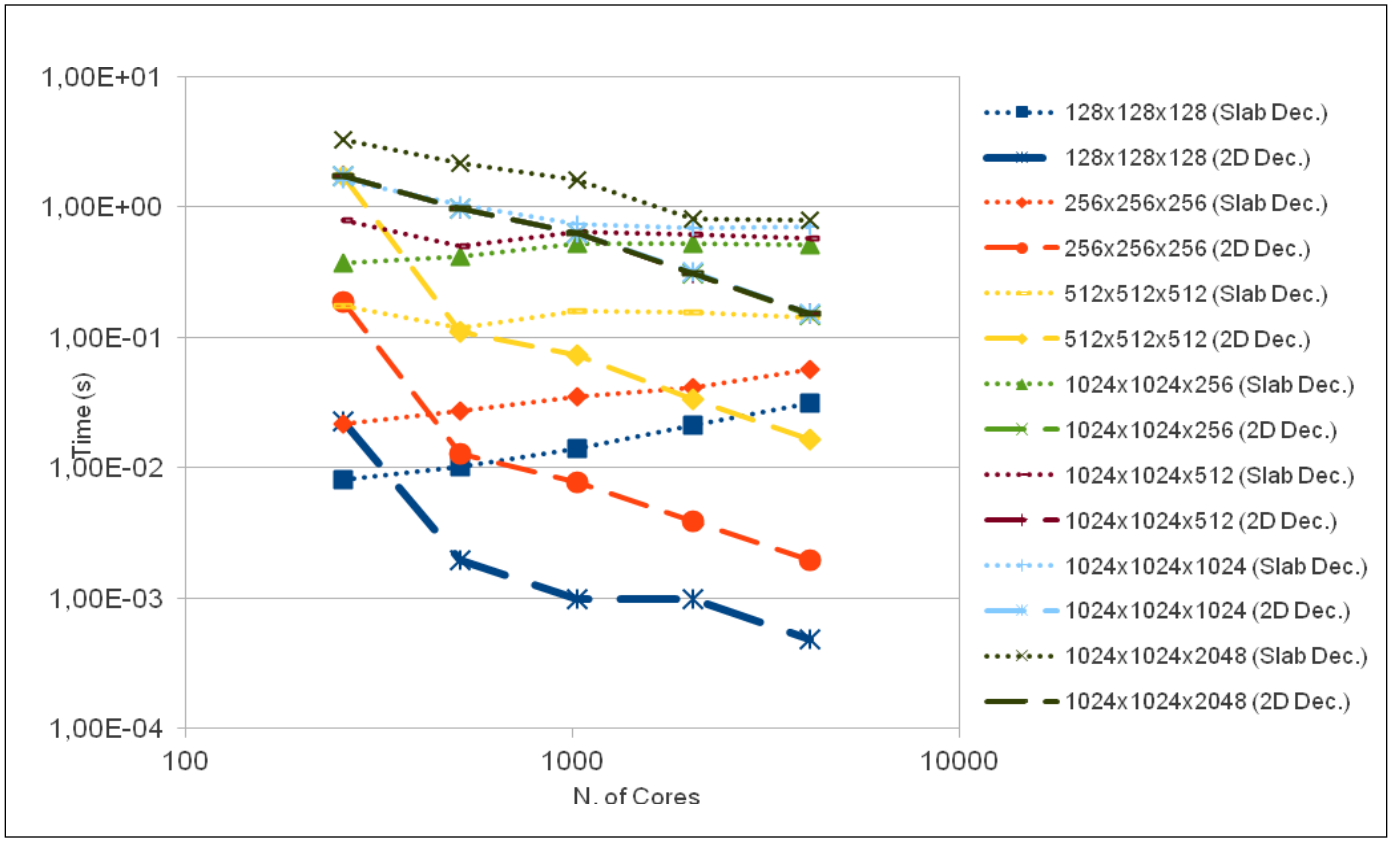
3D – FFT usig 2Decomp&FFT lib


```
PROGRAM FFT_3D_2Decomp_MPI
      use mpi
      use, intrinsic :: iso_c_binding
      use decomp_2d
      use decomp_2d_fft
     implicit none
     integer, parameter :: L = 128
     integer, parameter :: M = 128
     integer, parameter :: N = 128
     integer, parameter :: p_row = 16
     integer, parameter :: p_col = 16
     integer :: nx, ny, nz
     complex(mytype), allocatable, dimension(:,:,:) :: in, out
     complex(mytype) :: fout
     integer :: ierror, i,j,k, numproc, mype
     integer, dimension(3) :: sizex, sizez
! ===== Initialize
      call MPI_INIT(ierror)
      call decomp_2d_init(L,M,N,p_row,p_col)
      call decomp_2d_fft_init
```

3D – FFT usig 2Decomp&FFT lib


```
      allocate (in(xstart(1):xend(1),xstart(2):xend(2),xstart(3):xend(3)))
      allocate (out(zstart(1):zend(1),zstart(2):zend(2),zstart(3):zend(3)))
! ===== each processor gets its local portion of global data =====
      do k=xstart(3),xend(3)
        do j=xstart(2),xend(2)
          do i=xstart(1),xend(1)
            call initial(i, j, k, L, M, N, fout)
            in(i,j,k) = fout
          end do
        end do
       end do
! ===== 3D forward FFT =====
      call decomp_2d_fft_3d(in, out, DECOMP_2D_FFT_FORWARD)
! ==========================
      call decomp_2d_fft_finalize
      call decomp_2d_finalize
      deallocate(in,out)
      call MPI_FINALIZE(ierror)
    end
```

Some results



Links:

FFTW Homepage : http://www.fftw.org/

Download FFTW-3: http://www.fftw.org/fftw-3.3.3.tar.gz

Manual FFTW-3: http://www.fftw.org/fftw3.pdf

2Decomp&FFT homepage: http://www.2decomp.org/

Dowload 2Decomp&FFT: http://www.2decomp.org/download/2decomp_fft-1.5.847.tar.gz

Thank You