MHPC
Master in High Performance Computing

FFT IN PRACTICE

Ivan Girotto – igirotto@ictp.it
Information & Communication Technology Section (ICTS)
International Centre for Theoretical Physics (ICTP)

SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP The Abdus Salam International Centre for Theoretical Physics

Fields of Application

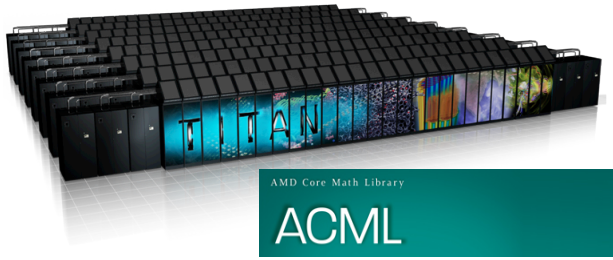
- The FFT is used in physics, astronomy, engineering, applied mathematics, cryptography, and comp. finance
- Method to solve differential equations
- Digital Signal Processing & Image Processing
  - Receive signal in the time domain, but want the frequency spectrum
- Convolutions/Filters
  - Filter can be efficiently represented mathematically by a convolution
- Benchmarking

MHPC
Master in High Performance Computing

Existing Libraries for Scientific High Performance Applications

MathKeisan

ESSL and Parallel ESSL



AMD Core Math Library
ACML
AMD
The future is fusion

cuFFT

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

SDSC
P3DFFT

FFTE

FFTW

2DECOMP & FFT

SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP
The Abdus Salam International Centre for Theoretical Physics

MHPC
Master in High Performance Computing

FFTW, WHY?

- Freely available open-source package. Most used on Linux environment for High Performance Computing
- FFTW is written in portable C and runs well on many architectures and O.S.
- FFTW computes DFTs in O (*n logn*) time for any length *n* and supports arbitrary multi-dimensional data
- Includes parallel (multi-threaded) transforms for shared-memory systems and distributed-memory parallel transforms using MPI libraries
- FFTW supports multiple and/or strided DFTs; for example, to transform a 3-component vector field or a portion of a multi-dimensional array.
- FFTW supports DFTs of real data, as well as of real symmetric/anti-symmetric data (also called discrete cosine/sine transforms).

FFTW basic usage

```
[...]
use, intrinsic :: iso_c_binding
!
implicit none
include 'fftw3.f03'
!
integer, intent( in ) :: n1,n2,n3
!
type(C_PTR) :: plan
complex(C_DOUBLE_COMPLEX), dimension(:,:,:), pointer :: in, out
type(C_PTR) :: p_in, p_out
!
! Aligned memory allocation
p_in = fftw_alloc_complex( int( n1 * n2 * n3, C_SIZE_T ) )
p_out = fftw_alloc_complex( int( n1 * n2 * n3, C_SIZE_T ) )
!
call c_f_pointer( p_in, in, [ n1, n2, n3 ] )
call c_f_pointer( p_out, out, [ n1, n2, n3 ] )
!
! Definition of a FFTW plan
!
plan = fftw_plan_dft_3d( n3, n2, n1, in, out, FFTW_FORWARD,FFTW_ESTIMATE )
!
call fftw_execute_dft( plan, in, out )
!
call fftw_destroy_plan( plan )
call fftw_free( p_in )
call fftw_free( p_out )
```

FFTW from FORTRAN (remarks)

- 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
- For example, consider the three-dimensional (L M N) arrays:

```
complex(C_DOUBLE_COMPLEX), dimension(L,M,N) :: in, out
```

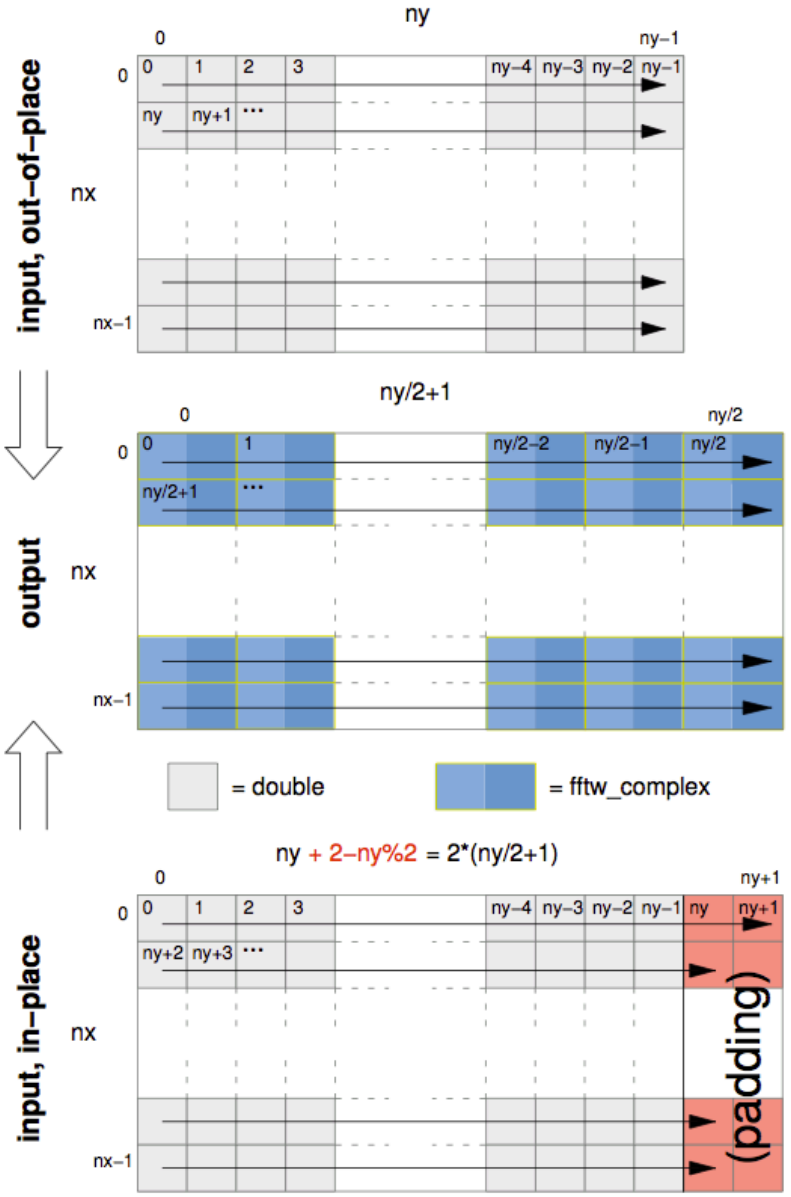
- To plan a DFT for these arrays using fftw_plan_dft_3d, you could do:

```
plan = fftw_plan_dft_3d(N,M,L, in,out, FFTW_FORWARD,FFTW_ESTIMATE)
```

- That is, from FFTW's perspective this is a N M L array. No data transposition need occur, as this is only notation.

Transform Real Data

- The DFT output satisfies the “Hermitian” redundancy: out[i] is the conjugate of out[n - i]
- Substantial optimization
- From r2c, the input is in real numbers, while the output is composed by n/2+1 complex numbers (developers must handle the different data types)



Multithread

- Works on shared-memory multi-cores system that supports POSIX threads and/or OpenMP
- Quick access to scalable FFT (How many threads?)
- Uses the same execution schema as for sequential but the following

```
[...]
/* Initialize FFTW multithreaded environment (to be called before call any FFTW routines) */
int fftw_init_threads(void);

[...]
/* Set the MAX number of threads the for creations of next plans */
void fftw_plan_with_nthreads(int nthreads);

[...]
/* Clean FFTW multithreaded environment */
void fftw_cleanup_threads(void);
```

- Calling FFTW from a thread region, developers must consider *fftw_execute(...)* is the only thread-safe subroutine

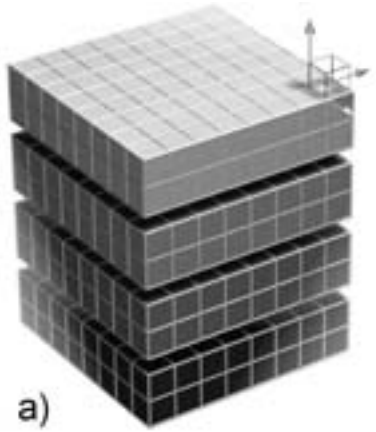
FFTW (MPI)

- each process only stores a portion of the data to be transformed => data structures and programming-interface are quite different from the serial or threads versions

```
#include <fftw3-mpi.h>

[...]
MPI_Init(&argc, &argv);
/* Initalizes FFTW_MPI environment. Must be included between MPI_Init & MPI_Finalize
fftw_mpi_init();

[...]
/* get local data size and allocate */
alloc_local = fftw_mpi_local_size_2d(N0, N1, MPI_COMM_WORLD, &local_n0, &local_0_start);
data = fftw_alloc_complex(alloc_local);

[...]
/* create plan for in-place forward DFT */
plan = fftw_mpi_plan_dft_2d(N0, N1, data, data, MPI_COMM_WORLD, FFTW_FORWARD, FFTW_ESTIMATE);
/* compute transforms, in-place */
fftw_execute(plan);

fftw_destroy_plan(plan);
MPI_Finalize();
```

FFTW (MPI)

- With a distributed-memory FFT, the inputs and outputs are broken into disjoint blocks, one per process
- In particular, FFTW uses a *1d block distribution* of the data, distributed along the *first dimension*



Installation and linking

- Download the package from www.fftw.org
- By default only double precision sequential FFT are compiled and included into the -lfftw3
- Other possible version must be enabled at ./configure time:
  - --enable-float, --enable-long-double for single/long-double precision
  - --enable-threads, --enable-openmp for multithread versions
  - --enable-mpi for MPI version (-lfftw3_mpi)
  - --enable-avx for SIMD extensions (many others available)

References

- FFTW3 manual @ www.fftw.org
- Matteo Frigo and Steven G. Johnson, *The design and implementation of fftw3*, Proc. IEEE, 93(2):2168211;231, 2005
- Matteo Frigo and Steven G. Johnson, Implementing FFTs in practice, http://cnx.org/content/m16336/latest/