

Scientific Computing (Phys 2109/Ast 3100H)

II. Numerical Tools for Physical Scientists

SciNet HPC Consortium, University of Toronto

Lecture 16: More Fast Fourier Transform

Winter 2013

Lessons from HW 3

About FFTW

Supposedly the “Fastest Fourier Transform in the West”

About FFTW

Supposedly the “Fastest Fourier Transform in the West”

version 2 ! = version 3

About FFTW

Supposedly the “Fastest Fourier Transform in the West”

version 2 ! = version 3

Capabilities

- ▶ Complex one-dimensional transforms
- ▶ Complex multi-dimensional transforms.
- ▶ Real-to-half-complex array transforms
- ▶ Format real transforms different in 1d and nd.
- ▶ Threaded, MPI, SIMD vectorized
- ▶ Read the manual!

Notes

- ▶ Always create a plan first.

An `fftw_plan` contains all information necessary to compute the transform, including the pointers to the input and output arrays.

Plans can be reused in the program, and even saved on disk!

Notes

- ▶ Always create a plan first.
An `fftw_plan` contains all information necessary to compute the transform, including the pointers to the input and output arrays.
Plans can be reused in the program, and even saved on disk!
- ▶ When creating a plan, you can have FFTW measure the fastest way of computing dft's of that size (`FFTW_MEASURE`), instead of guessing (`FFTW_ESTIMATE`).

Notes

- ▶ Always create a plan first.
An `fftw_plan` contains all information necessary to compute the transform, including the pointers to the input and output arrays.
Plans can be reused in the program, and even saved on disk!
- ▶ When creating a plan, you can have FFTW measure the fastest way of computing dft's of that size (`FFTW_MEASURE`), instead of guessing (`FFTW_ESTIMATE`).
- ▶ FFTW works with doubles by default, but you can install single precision too.

Symmetries

Even data:

$$f_i = f_{-i} = f_{n-i}$$



$$\hat{f}_k = \hat{f}_{-k} = \hat{f}_{n-k}$$

Symmetries

Even data:

$$f_i = f_{-i} = f_{n-i}$$



$$\hat{f}_k = \hat{f}_{-k} = \hat{f}_{n-k}$$

Odd data:

$$f_i = -f_{-i} = -f_{n-i}$$



$$\hat{f}_k = -\hat{f}_{-k} = -\hat{f}_{n-k}$$

Symmetries

Even data:

$$f_i = f_{-i} = f_{n-i}$$



$$\hat{f}_k = \hat{f}_{-k} = \hat{f}_{n-k}$$

Odd data:

$$f_i = -f_{-i} = -f_{n-i}$$



$$\hat{f}_k = -\hat{f}_{-k} = -\hat{f}_{n-k}$$

Shifted data:

$$f_j = f'_{j+J}$$



$$\hat{f}_k = \exp(2\pi i J k / n) \hat{f}'_k$$

Symmetries for real data

- ▶ All arrays were complex so far.
- ▶ If input $\mathbf{f}$ is real, this can be exploited.

$$\mathbf{f}_j^* = \mathbf{f}_j \leftrightarrow \hat{\mathbf{f}}_k = \hat{\mathbf{f}}_{n-k}^*$$

- ▶ Each complex number holds two real numbers, but for the input $\mathbf{f}$ we only need n real numbers.
- ▶ If n is even, the transform $\hat{\mathbf{f}}$ has real $\hat{\mathbf{f}}_0$ and $\hat{\mathbf{f}}_{n/2}$, and the values of $\hat{\mathbf{f}}_k > n/2$ can be derived from the complex valued $\hat{\mathbf{f}}_{0 < k < n/2}$: again n real numbers need to be stored.

Symmetries for real data

- ▶ A different way of storing the result is in “half-complex storage”. First, the $n/2$ real parts of $\hat{\mathbf{f}}_{0 \leq k < n/2}$ are stored, then their imaginary parts in reversed order.
- ▶ Seems odd, but means that the magnitude of the wave-numbers is like that for a complex-to-complex transform.
- ▶ These kind of implementation dependent storage patterns can be tricky, especially in higher dimensions.

Multidimensional transforms

In principle a straightforward generalization:

- ▶ Given a set of $\mathbf{n} \times \mathbf{m}$ function values on a regular grid:

$$\mathbf{f}_{ab} = \mathbf{f}(\mathbf{a}\Delta\mathbf{x}, \mathbf{b}\Delta\mathbf{y})$$

- ▶ Transform these to $\mathbf{n}$ other values $\hat{\mathbf{f}}_{\mathbf{k}}$

$$\hat{\mathbf{f}}_{kl} = \sum_{a=0}^{n-1} \sum_{b=0}^{m-1} \mathbf{f}_{ab} e^{\pm 2\pi i (a k/n + b l/m)}$$

- ▶ Easily back-transformed:

$$\mathbf{f}_{ab} = \frac{1}{nm} \sum_{k=0}^{n-1} \sum_{l=0}^{m-1} \hat{\mathbf{f}}_{kl} e^{\mp 2\pi i (a k/n + b l/m)}$$

- ▶ Negative frequencies: $\mathbf{f}_{-k, -l} = \mathbf{f}_{n-k, m-l}$.

Multidimensional FFT

- ▶ We could successive apply the FFT to each dimension
- ▶ This may require transposes, can be expensive.
- ▶ Alternatively, could apply FFT on rectangular patches.
- ▶ Mostly should let the libraries deal with this.
- ▶ FFT scaling still **$n \log n$** .
- ▶ Real transform even more convoluted.

Homework

Trigonometric interpolation

Trigonometric interpolation uses a n point Fourier series to find values at intermediate points. It is one way of “downscaling” data, and was a motivation for Gauss, to be applied to planetary motion. The way it works is:

- ▶ You fourier transform your data
- ▶ You add frequencies above the Nyquist frequency (in absolute values), but set all the amplitudes of the new frequencies to zero.
- ▶ Note that the frequencies are stored such that eg. $\hat{f}_{n-1}$ is a low frequency $-1/n$.
- ▶ The resulting $2n$ array can be back transformed, and now gives an

Homework

Assignment 1

Write an application that will read in this image:



as a binary file with a 2d array, in double precision, and creates an image twice the size in all directions.

Use a real-to-half-complex version of fftw (despite the counter-recommendation in the fftw3 documentation).

Homework

PPM image format

The image format to be used is ppm, which goes as follows:

- ▶ first line: "P6\n"
- ▶ second line: "width height\n"
- ▶ Subsequently triplets of 3 (rgb) byte values per pixel.

Homework

Assignment 2

Write an application which reads an image and performs a low pass filter on the image for each of the colors (rgb). I.e., any fourier components with magnitudes k larger than $n/8$ are to be set to zero, after which the fourier inverse is taken and the image is to be printed out. You can use the real-to-half-complex versions of fftw here too.

Due next Friday at 9:00 am!