epcc

Parallel Fast Fourier Transforms

Gavin J. Pringle
Joahcim Hein

Introduction

- The Fourier Transform
  - What, who, why?
  - Mathematics and and its inherent properties
- Discrete Fourier Transform
- Fast Fourier Transform, or FFT
- Parallel FFTs
- FFT libraries
- Fastest Fourier Transform in the West
  - Configuration, installation, compilation and runtime tuning
  - Execution times and other users experiences

Fourier Transforms

- Jean Baptiste Joseph Fourier (1768-1830) first employed what we now call Fourier transforms whilst working on the theory of heat
  - The Fourier transform first appeared in “On the Propagation of Heat in Solid Bodies”, memoir to Paris Institute, 21 Dec., 1807.
- Mathematical tool which alters the problem to one which is more easily solved
- Linear transform which converts temporal or spatial information and converts into information which lies in the frequency domain
  - And visa versa
  - Frequency domain also known as Fourier space, Reciprocal space, or G-space

Pictures of Joseph Fourier

Who would use Fourier Transforms?

epcc

- Physics
  - Cosmology (P^3M N-body solvers)
  - Fluid mechanics
  - Quantum physics
  - Signal and image processing
    - Antenna studies
    - Optics
- Numerical analysis
  - Linear systems analysis
  - Boundary value problems
  - Large integer multiplication (Prime finding)
- Statistics
  - Random process modelling
  - Probability theory

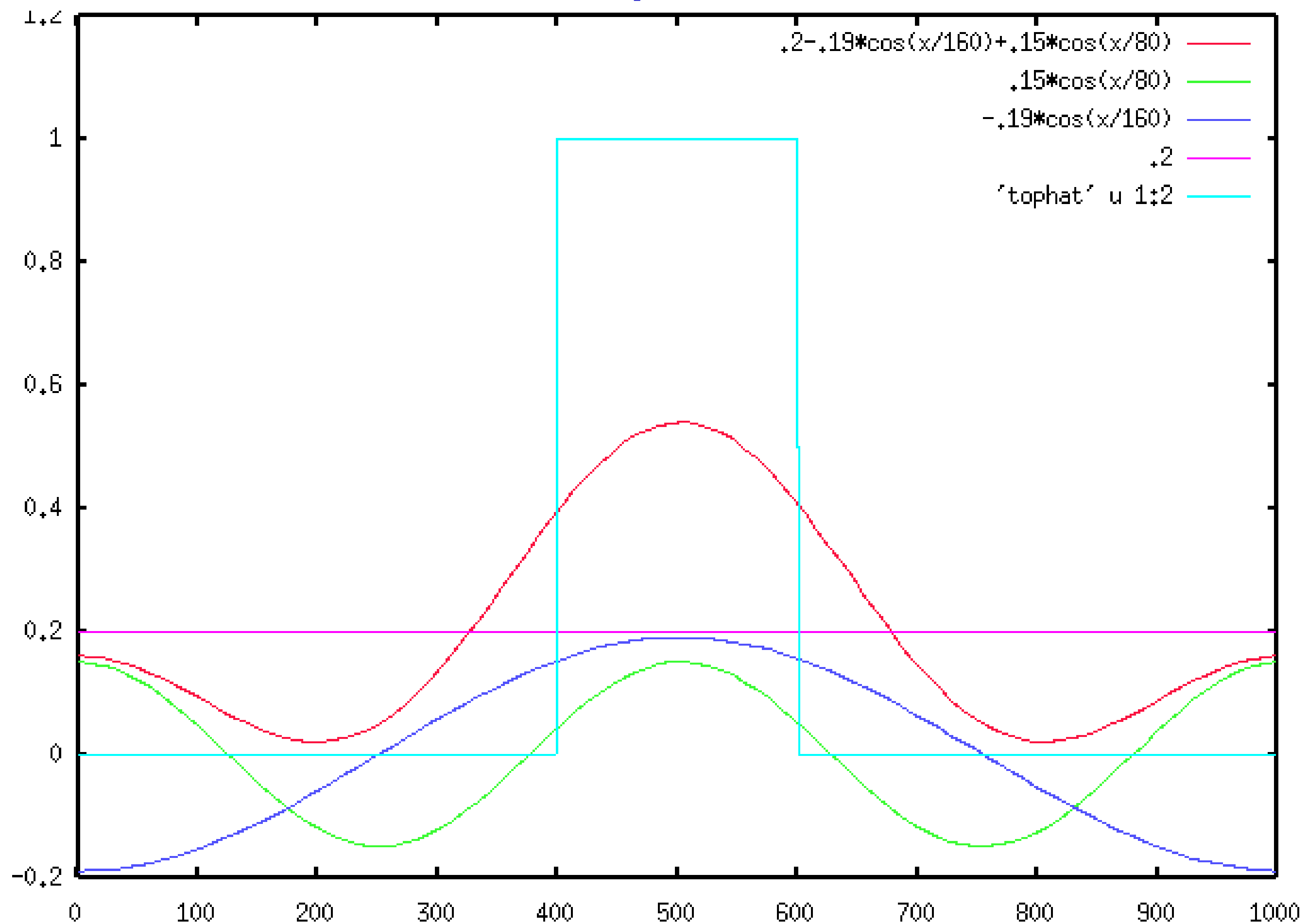
Fourier Transforms in a nutshell

- All periodic signals may be represented by an infinite sum of sines and cosines of different periods and amplitudes. (Fourier’s Theorem)
  - The cosines and sines are associated with the symmetrical and asymmetric information, respectively
  - Any signal can be broken into a sequence of ‘chunks’, where each chunk may be considered periodic.
- Fourier transforms encode this information via

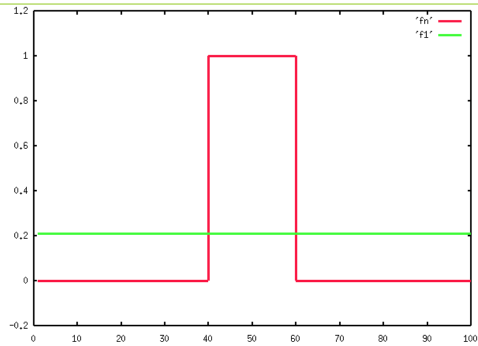
$$e^{i\theta} = \cos\theta + i\sin\theta$$

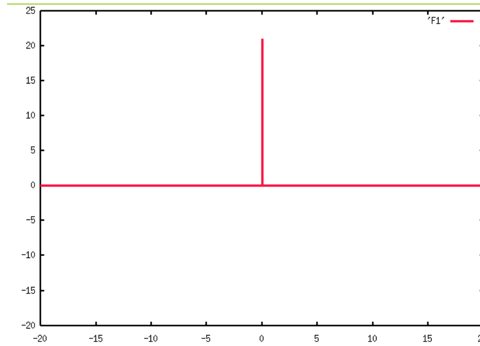
The Top Hat function

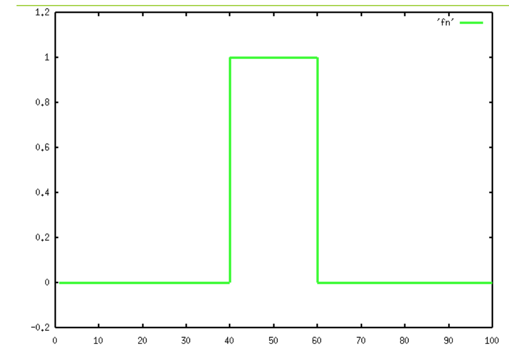
- The top hat function, along with the individual 1st, 2nd and 3rd Fourier components and their sum.

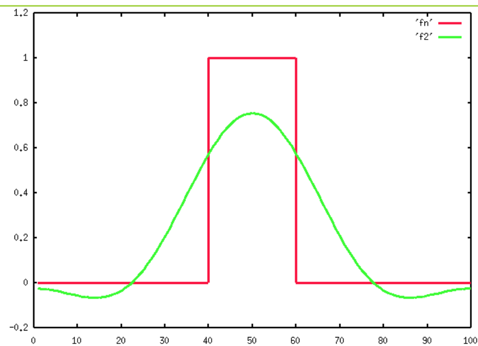


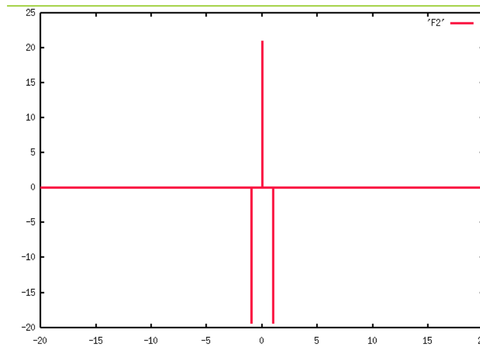
The Top Hat function and its discrete Fourier components

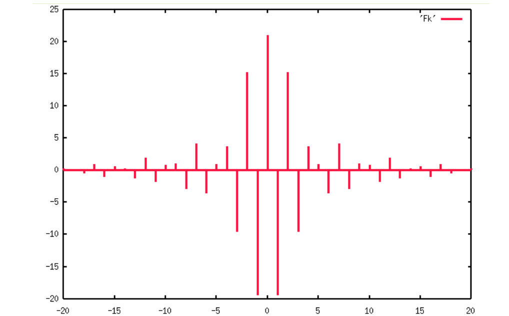


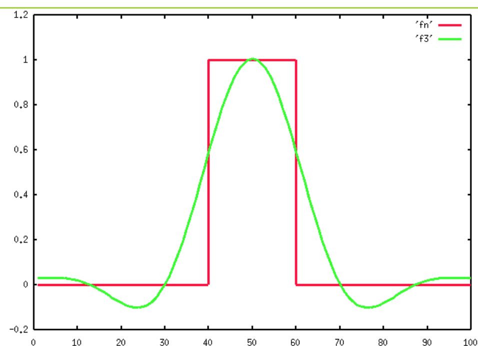


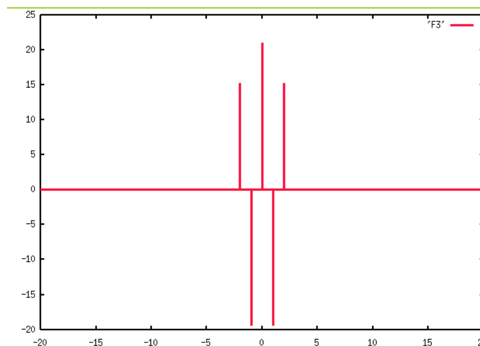






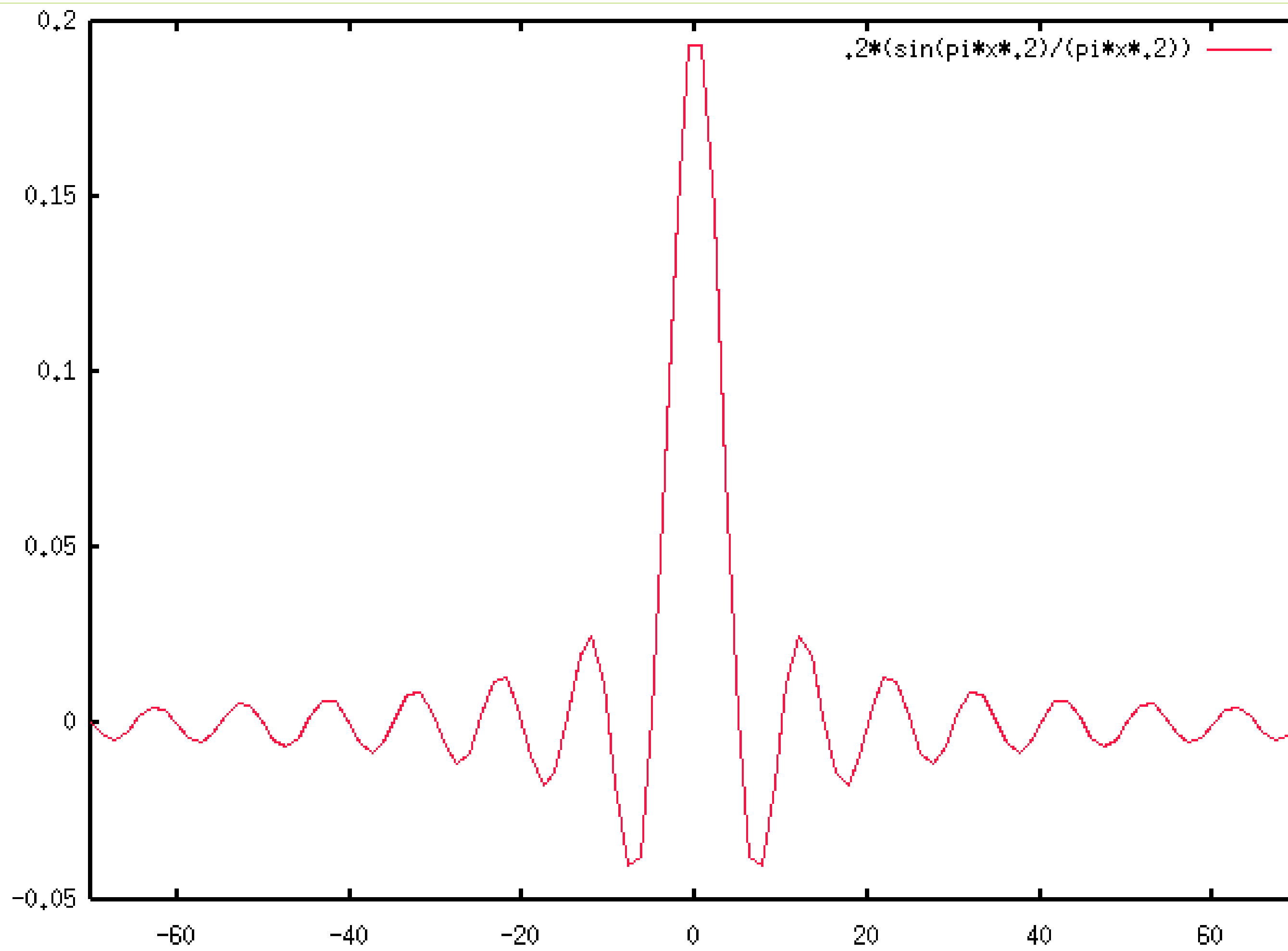






animation

The Fourier transform of a continuous Top Hat function



- The Fourier transform of a complex function *f* (*x*) is given as

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xs} dx$$

- The inverse Fourier transform is given as

$$f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi xs} ds$$

- The Fourier pair is defined as

$$f(x) <=> F(s)$$

Properties 1: Scaling

- Time scaling

$$f(at) \Leftrightarrow \frac{1}{|a|} F\left(\frac{s}{a}\right)$$

- Frequency scaling

$$\frac{1}{|b|} f\left(\frac{t}{b}\right) \Leftrightarrow F(bs)$$

Properties 2: Shifting

- Time shifting

$$f(t-t_0) \Leftrightarrow F(s)e^{2\pi i s t_0}$$

- Frequency shifting

$$f(t)e^{-2\pi i s_0 t} \Leftrightarrow F(s-s_0)$$

Properties 3: Convolution Theorem

- Say we have two functions, $g(t)$ and $h(t)$, then the convolution of the two functions is defined as

$$g \otimes h \equiv \int_{-\infty}^{\infty} g(\tau)h(t-\tau)d\tau$$

- The Fourier transform of the convolution is simply the product of the individual Fourier transforms

$$g \otimes h \Leftrightarrow G(s)H(s)$$

Properties 4: Correlation

- The correlation of the two functions is defined by

$$Corr(g,h) \equiv \int_{-\infty}^{\infty} g(\tau + t)h(\tau)d\tau$$

- The Fourier transform of the correlation is simply

$$Corr(g,h) \Leftrightarrow G(s)H(-s)$$

Discrete Fourier Transform

- The discrete Fourier transform of N complex points f_k is defined as

$$F_n \equiv \sum_{k=0}^{N-1} f_k e^{2\pi ikn/N}$$

- The discrete inverse Fourier transform, which recovers the set of f_k s exactly from F_ns is

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{-2\pi ikn/N}$$

- Both the input function and its Fourier transform are periodic

- The DFT can be rewritten as

$$F_n = a_0 + \sum_{k=1}^{N-1}\left(a_k \cos\left(2\pi k \frac{n}{N}\right) + b_k i \sin\left(2\pi k \frac{n}{N}\right)\right)$$

- Thus, DFT routines are basically returning real number values for a_k and b_k , stored in a complex array
  - a_k and b_k are functions of f_k
  - remaining trigonometric constants (twiddle factors) may be pre-computed for a given N
- The scaling, shifting, convolution and correlation relationships, which hold for the continuous case, also hold for the discrete case.

- What is the computational cost of the DFT?
  - Each of the *N* points of the DFT is calculated in terms of all the N points in the original function: $O(N^2)$
- In 1965, J.W. Cooley and J.W. Tukey published an DFT algorithm which is of $O(N \log N)$
  - *N* is a power of 2
  - FFTs are not limited to powers of 2, however, the order may resort to $O(N^2)$
  - Details are beyond the scope of this talk
    - $F(N) = F(N/2)+F(N/2)$
    - Bit reversal
  - In hindsight, faster algorithms were previously, independently discovered
    - Gauss was probably first to use such an algorithm in 1805

Parallel 1D FFT

- Parallelisations of a 1D FFT is hard
- Typically `N≈100` in many scientific codes
- Algorithm is hard to decompose
- Literature example:

Franchetti, Voronenko, Püschel, “FFT Program Generation for Shared Memory: SMP and Multicore”, Paper presented as SC06, Tampa, FL http://sc06.supercomputing.org/schedule/pdf/pap169.pdf

FFTs in two dimensions

- What needs calculating for a 2D FFT:

$$\tilde{f}(k,l) = \sum_{y=1}^{M} \left\{ \sum_{x=1}^{N} \left[f(x,y) \exp\left(-2\pi \mathrm{i} \frac{kx}{N}\right)\right] \exp\left(-2\pi \mathrm{i} \frac{ly}{M}\right)\right\}$$

- We may compute this in a 2 separate calculations
  - as each part is linearly independent

$$\hat{f}(k,y) \equiv \sum_{x=1}^{N} \left[f(x,y) \exp\left(-2\pi \mathrm{i} \frac{kx}{N}\right)\right]$$

$$\tilde{f}(k,l) = \sum_{y=1}^{M} \left\{ \hat{f}(k,y) \exp\left(-2\pi \mathrm{i} \frac{ly}{M}\right)\right\}$$

- Assignment of a 4x4 grid to 4 processors for an array transpose

P1	1	2	3	4
P2	5	6	7	8
P3	9	10	11	12
P4	13	14	15	16

P1	P2	P3	P4
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Algorithm for distributed 2D FFT on a 1D grid of processors

- Calculate 1st FFT in first direction
- Perform parallel transpose
  - MPI_Alltoall
  - Now, what used to be the columns of the original matrix is now processor local
- Now we may perform the 2nd FFT in second direction
- Finally, perform parallel transpose back
  - Sometimes this last expensive step can be avoided
  - Code performs calculations in Fourier space using this new processor grid

Fourier Transformation of a 3D array

- Definition of the Fourier Transformation of a three dimensional array $A_{x,y,z}$

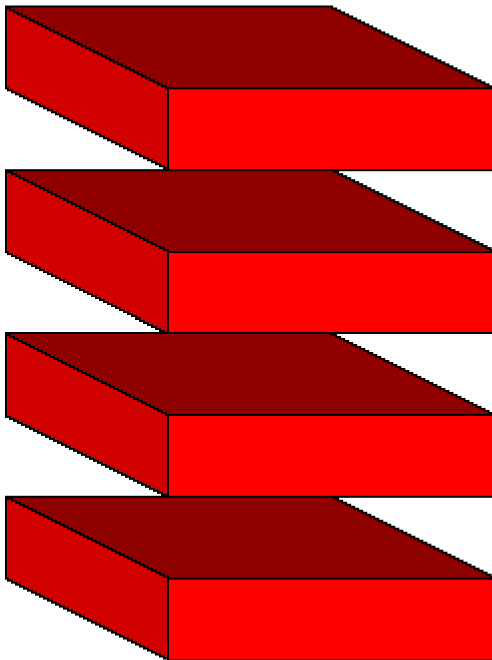
$$\tilde{A}_{u,v,w} :=$$

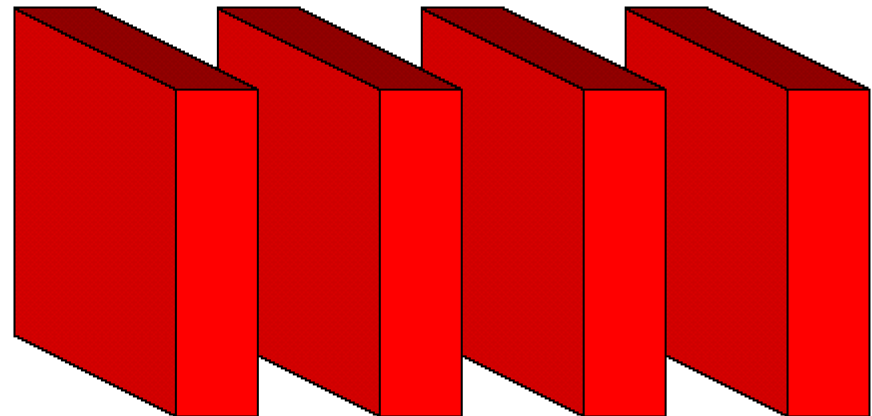
$$\underbrace{\sum_{x=0}^{L-1} \underbrace{\sum_{y=0}^{M-1} \underbrace{\sum_{z=0}^{N-1} A_{x,y,z} \exp(-2\pi \mathrm{i} \frac{wz}{N})}_{\text{1st 1D FT along } z} \exp(-2\pi \mathrm{i} \frac{vy}{M})}_{\text{2nd 1D FT along } y} \exp(-2\pi \mathrm{i} \frac{ux}{L})}_{\text{3rd 1D FT along } x}$$

- Can be performed as three subsequent 1 dimensional Fourier Transformations

Parallel FFT of a 3D array

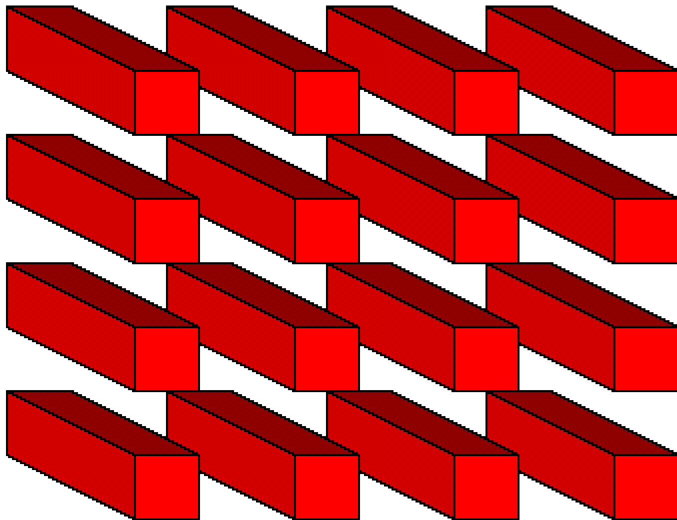
- Traditionally: 1 dimensional processor grid
- Each processor gets several “planes” (or “slices”)
- Perform FFT in two of the three directions
- Single All-to-all before performing FFT in third direction

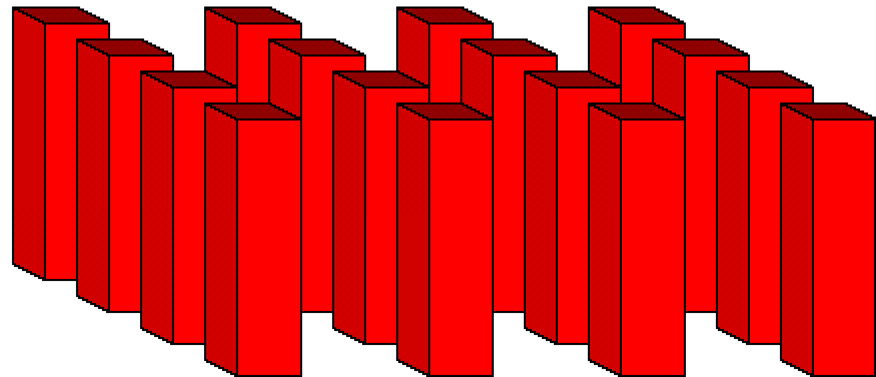




Alternatively: 2D processor grid for 3D FFT

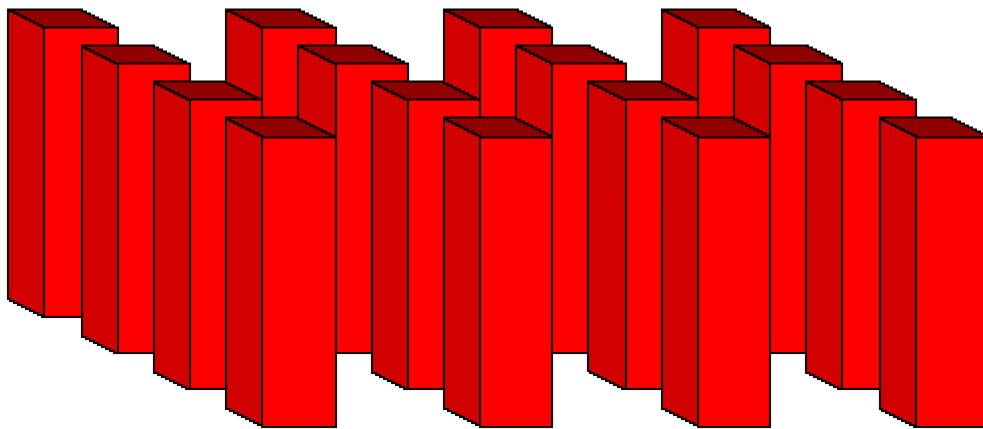
- Each processor gets several “sticks” (or “pencils”) of the 3D array
- Perform FFT in 1st direction
- Perform All-to-all transformation in the columns of the processor grid

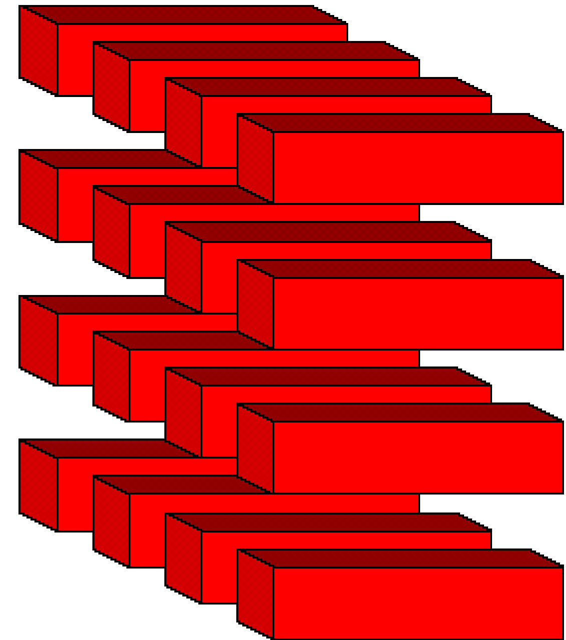




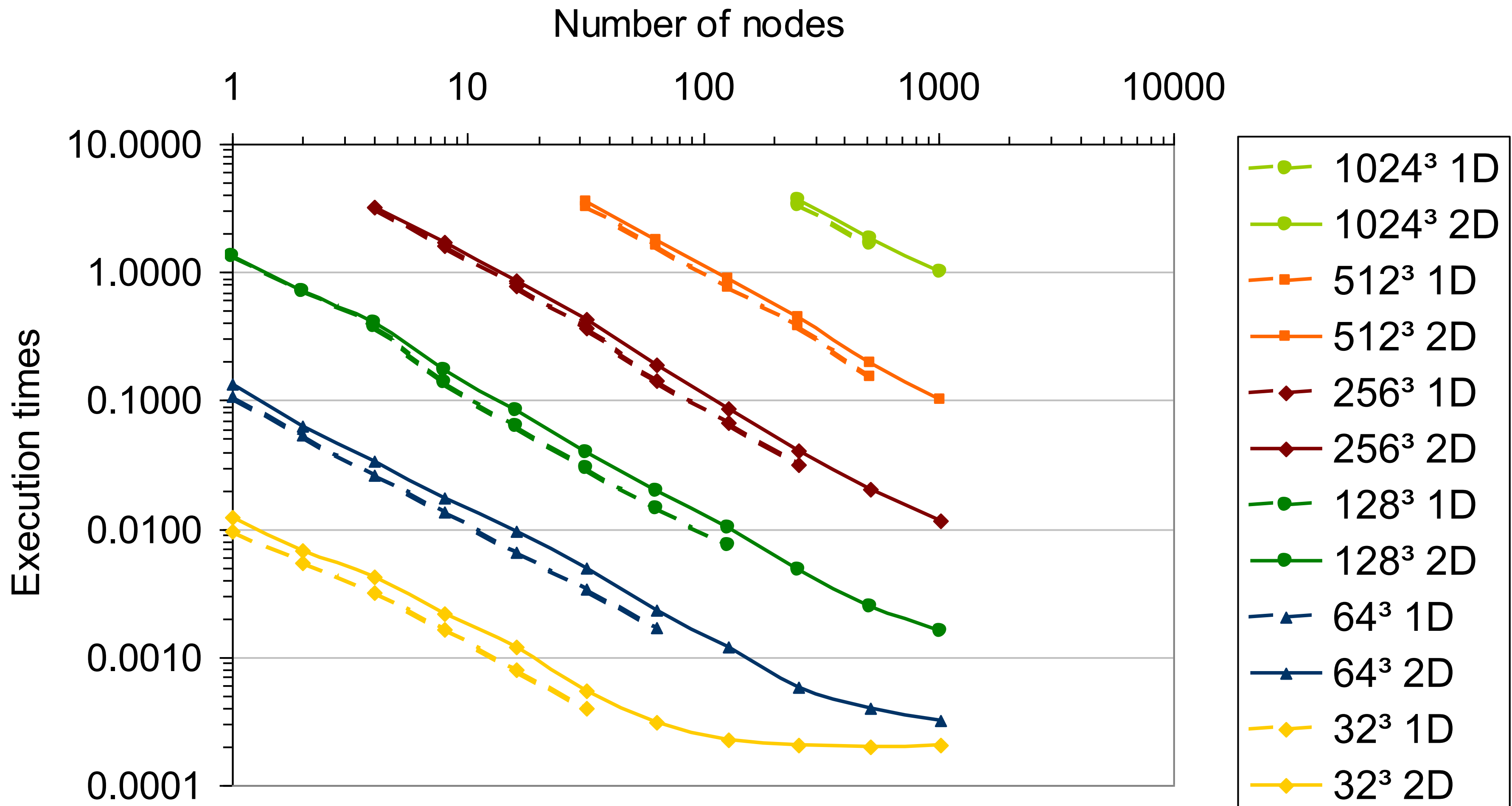
2D processor grid for 3D FFT (cont.)

- Perform FFT in the 2nd direction
- Perform All-to-all in the rows of the processor grid
- Perform 3rd FFT in the last direction





Performance comparison of 1D pencils vs 2D slabs: IBM BlueGene/L



- Heike Jagode, MSc thesis, University of Edinburgh, 2006

Pencils vs Slabs

- For 3D data points, users employ 1D or 2D processor grid
  - 1D processor grid: sticks/pencils
    - More communications
    - Requires less memory
    - In general, better scalability
  - 2D processor grid: slabs/slices
    - Less communications
    - Requires more memory
- The optimum choice depends on both the problem and the target platform
- Tip: let the physics be your guide and pick the decomposition that suits your problem
  - Try not to make your code platform-specific

Use of general FFT libraries 1

- FFTs do not normalise
  - Each FFT/Inverse FFT pair scales by a factor of N
  - Left as an exercise for the programmer.
- DFTs are complex-to-complex transforms, however, most applications require real-to-complex transforms
  - Simple solution: set imaginary part of input data to be zero
    - This will be relatively slow
  - Better to pack and unpack data
    - Place all the real data into all slots of the input, complex array (of length (n/2) and then unpack the result on the other side (O(n))
    - Around twice as fast as the simple solution
    - Good details in Numerical Recipes
  - Some libraries have real-to-complex wrappers

Use of general FFT libraries 2

- Multidimensional FFTs
  - simply successive FFTs over each dimension
    - order immaterial: linearly independent operations
  - pack data into 1D array – see practical
  - strided FFTs
  - some libraries have multidimensional FFT wrappers
- Parallel FFTs
  - performing FFT on distributed data
  - 1D FFTs are cumbersome to parallelise
    - Suitable only for huge N
  - parallel, array transpose operation
    - distributed data is collated on one processor before FFT
    - diagram on next slide
  - most FFT libraries have parallel FFT wrappers

Introduction to the FFTW library

- Fastest Fourier Transform in the West
  - www.fftw.org
- The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson
- Free under GNU General Public License
- Portable, self-optimising C code
  - Runs on a wide range of platforms
- Arbitrary sized FFTs of one or more dimensions
  - Fastest routines where extents are composed of powers of 2, 3, 5 and 7 (other sizes can be optimised for at configuration time)

FFTW2 versus FFTW3

- Previous version: “FFTW2”
  - Many legacy codes employ FFTW2
  - Simple(r) C interface, with wrappers for many other languages
  - Supports MPI
  - Rest of this lecture assumes “FFTW2”
- New version: “FFTW3”
  - Different interface to FFTW2 – to allow “planner” more freedom to optimise
    - Users must rewrite code
  - Doesn’t support MPI (currently in alpha release)
    - Most codes implement the parallel transpose, and perform the 1D FFTs using FFTW
  - Somewhat faster than FFTW2 (~10% or more)

Some technical details of the FFTW2

- Can perform FFTs on distributed data
  - MPI for distributed memory platforms
  - OpenMP or POSIX for SMPs
- If users rewrite their code to this FFT just once then the user is saved from
  - learning platform dependent, proprietary FFT routines
  - rewriting their code every time they port their code
    - No standard interface to FFTs
  - drastically rewriting their makefiles
    - Although the location of FFTW libraries may vary
- FORTRAN wrappers for the majority of routines
  - Currently FORTRAN FFTs are not “in-place”
    - The input and output arrays must be separate and distinct
  - Nor are the strided FFT calls (in FFTW 2)
    - The input/output arrays must be contiguous

FFTW2 Plans in a nutshell

- All FFT libraries pre-compute the *twiddle factors*
- FFTW 'plans' also generates the FFT code from *codelets*
  - Codelets compiled when FFTW configured
- Two forms of plans
  - Estimated
    - The best numerical routines are guessed, based on information gleaned from the configuration process.
  - Measured
    - Different numerical routines are actually run and timed with the fastest being used for all future FFTW calls using this plan.
- Old plans can be reused or even read from file: *wisdom*

Configuration and Installation

- Download library from the website and unpack (gzipped tar file)
- `./configure; make; make install`
  - Probes the local environment
  - Compiles many small C object codes called *codelets*
  - User can provide non-standard compiler optimisation flags
  - Libraries (both static and dynamic) are then installed along with online documentation and header files
- Includes test suite
  - Very important for any numerical library

Compiling code

- **`gfortran fft_code.f -O3 -lfftw`**
  - If using C, FFTW must be linked with `-lfftw -lm`
  - If the FFTW library configured for both single and double precision, then link with **`-lsfftw`** and **`-lfftw`**, respectively.
- Example FORTRAN code:

```
integer plan
integer, parameter :: n = 1024
complex in(n), out(n)
! plan the computation
call fftw_f77_create_plan(…)
! execute the plan
call fftw_f77_one(…)
```

  - NB: actual correct incantations are not given here as reading documentation is integral to utilising any numerical library

Performance

- The FFTW homepage, www.fftw.org, details the performance of the library compared to proprietary FFTs on a wide range of platforms.
- The FFTW library is faster than any other portable FFT library
- Comparable with machine-specific libraries provided by vendors

- Performance results from http://www.fftw.org/speed/

epcc

AMD Opteron 275 2.2 GHz

double-precision complex, 1d transforms

powers of two

speed (mflops)

3000
2500
2000
1500
1000
500
0

2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144

fftw3 out-of-place
fftw3 in-place
ooura-sgf
intel-mkl-dfti in-place
intel-mkl-dfti out-of-place
acml
green
kissfft
harm
bloodworth
sciport
dfftpack
rmayer-unstable
numutils
monnier
mixfft
mpfun77
scimark2c
cross
esrfft
jmfftc
cwplib

epcc

Intel Core Duo 3.0 GHz

double-precision complex, 1d transforms

powers of two

speed (mflops)

10000 9500 9000 8500 8000 7500 7000 6500 6000 5500 5000 4500 4000 3500 3000 2500 2000 1500 1000 500 0

2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576 2097152 4194304 8388608 16777216 33554432

intel-ipps
intel-mkl-dfti in-place
fftw3 out-of-place
fftw3 in-place
intel-mkl-dfti out-of-place
ffte
ooura-sgf
fftw2
spiral-egner-fft
green
emayer
dfftpack
gsl-mixed-radix
mpfun90
bloodworth
harm
sciport
kissfft
monnier
numutils
esrfft
mixfft
cross
scimark2c
cwplib
jmfftc

epcc

IBM POWER5 1.65 GHz

double-precision complex, 1d transforms

powers of two

speed (mflops)

0
500
1000
1500
2000
2500
3000
3500

2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144

essl in-place
essl out-of-place
fftw3 out-of-place
fftw3 in-place
ooura-sgf
green
dfftpack
sciport
kissfft
bloodworth
monnier
arprec
harm
numutils
scimark2c
cross
cwplib
mixfft
esrfft
jmfftc
valkenburg

Accolades

- Winner of the 1998 J.H. Wilkinson Prize for Numerical Software
  - awarded every four years to the software that "best addresses all phases of the preparation of high quality numerical software."
- Quotes from www.fftw.org.
  - “It’s the best FFT package I have ever seen”
  - “It performs my standard 256 iterations of 1024pt complex FFT about 20 times faster than the previous one I used.”
  - “FFTW is the best thing since microwave popcorn”
- Dr. Richard Field
  - Former Vice-principal of University of Edinburgh and Chairman of NAG
  - “I think FFTW is terrific. It’s the best piece of software I’ve seen written in a bunch of years. […] I give FFTW very high marks (probably as high marks as I would ever give).”

Summary

- Introduced both the continuous and discrete forms of the Fourier Transform
- Stated the translation theorems of the Fourier transform
  - Scaling, Shifting, Convolution and Correlation
- Fast Fourier Transform
- Parallel FFTs
- FFTW
  - Fast, robust and portable
  - FORTRAN and C, serial and parallel.
  - Simple to use
  - Recommended and used in major projects by EPCC

epcc

Thank you

- Any questions?

- gavin@epcc.ed.ac.uk