THE UNIVERSITY *of* York

Parallel Fast Fourier Transform

Matt Probert

August-Wilhelm Scheer Visiting Prof TUM 2015

Condensed Matter Dynamics Group

Department of Physics,

University of York, U.K.

http://www-users.york.ac.uk/~mijp1

Overview

- What is a Fourier Transform?
- What is the Fast Fourier Transform?
  - Maths & Scaling
  - Data layout
  - Handling real data
  - Extensions to 2D, 3D …
- How to parallelize the FFT
  - FFT in parallel vs FFT of parallel data
- Summary

Fourier Transform Basics

Fourier Transforms

- A linear transform that maps a function from time $\leftarrow\rightarrow$ frequency domain
  - Or from space $\leftarrow\rightarrow$ reciprocal space, etc

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

  - Very useful in many areas, for signal processing, spectral analysis, etc.
  - In DFT the KE is simple in reciprocal space and the PE is simple in real space
  - Different conventions on 2π factors etc

Fourier Analysis

Animation from Wikipedia ‘Fourier Analysis’ article

Numerical Fourier Transforms

Discrete Fourier Transform

- How can we do it computationally?
- Need to sample data at discrete intervals:

$$h_k = h\left(t_k\right),\ t_k = k.\delta t,\ k = 0, 1, 2, 3, \ldots N-1$$

- And then do Discrete Fourier Transform:

$$H\left(f_n\right) = \int_{-\infty}^{\infty} h\left(t\right) \exp\left(2\pi i f_n t\right) dt$$

$$\simeq \sum_{k=0}^{N-1} h_k \exp\left(2\pi i f_n t_k\right) \delta t = \delta t \sum_{k=0}^{N-1} h_k \exp\left(2\pi i k n / N\right)$$

DFT continued

- Hence define DFT:

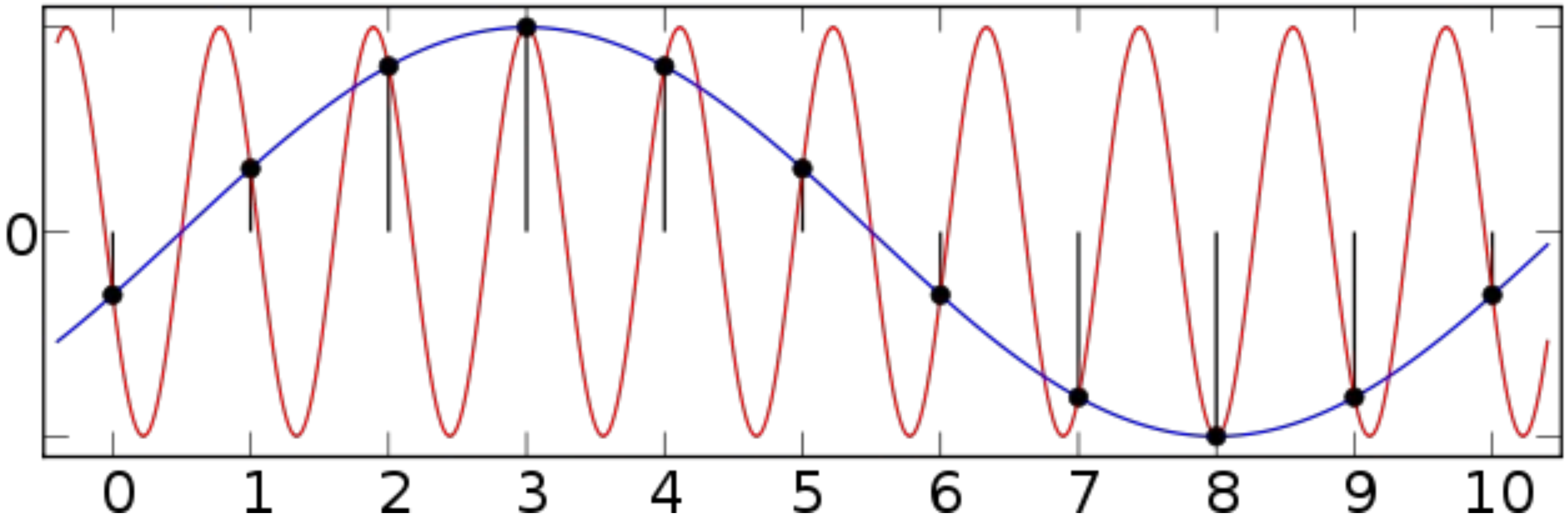
$$H_n = \sum_{k=0}^{N-1} h_k \exp\left(2\pi i k n / N\right)$$

- And inverse DFT:

$$h_k = \frac{1}{N} \sum_{k=0}^{N-1} H_n \exp\left(-2\pi i k n / N\right)$$

Consequences of sampling

- Data is sampled with time interval of δt
- Hence only finite range of frequencies present: $-f_c < f < f_c$ and $f_c = \dfrac{1}{2.\delta t}$
  - Hence can have *aliasing* – where two different signals appear to be the same



Other numerical issues

- Can also have problems when sampling periodic functions
  - What happens if sample interval is not an integer number of periods?
  - Can get apparent discontinuities in signal
- *Filtering* the data can help
  - Lots of different filters available
  - Trade-off between broadening central peak vs power into side lobes, etc.

Implementing DFT

- Matrix multiplication:

$$H_n = \sum_{k=0}^{N-1} W^{nk} h_k$$

- Input data is vector ***h*** of length *N*
- Output data is vector ***H*** of length *N*
- ***W*** is a matrix of size *N*x*N* whose (*n*,*k*)th element is the constant $W = \exp\left(2\pi i/N\right)$ raised to the power *n* x *k*

Speed of DFT

- Each element of H_k requires N multiplications hence overall $\sim\mathcal{O}(N^2)$
  - which is true of the general transform
- BUT if N is even then can split h into two parts, h_e = even indices h_{2j} and h_o = odd indices h_{2j+1}
- Whereupon find $H_k = H^e_k + W^k H^o_k$
  - Hence cost is now $\sim\mathcal{O}(2(N/2)^2)$
- Repeat … Fast Fourier Transform $\sim\mathcal{O}(N \cdot \log_2 N)$

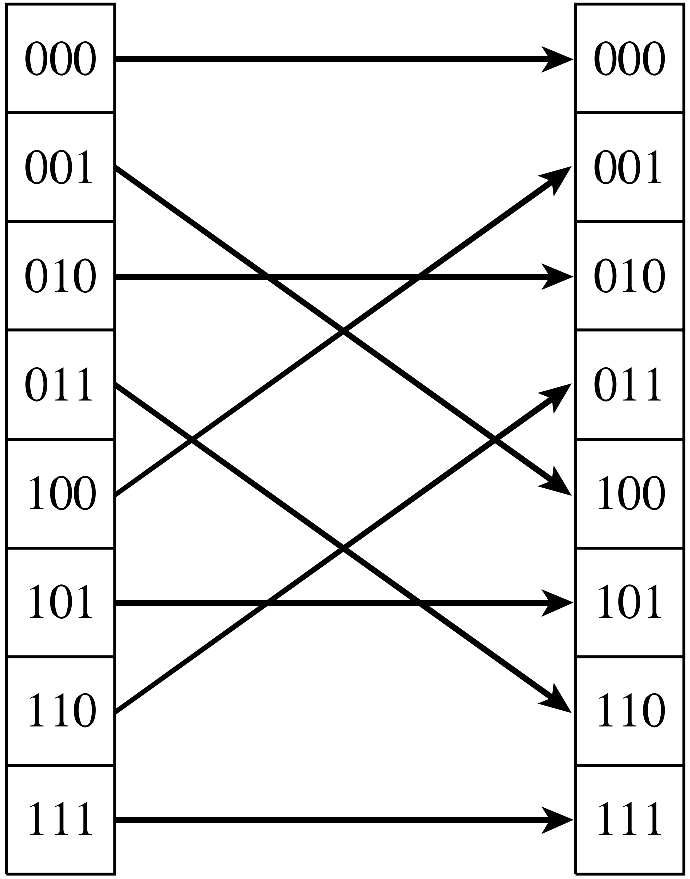
Fast
Fourier Transforms

Fast Fourier Transform

- The FFT algorithm was independently discovered a number of times in past
- Major breakthrough came with computational discovery/ implementation by Cooley & Tukey
  - Voted one of the “Top 10 algorithms of C20”!
  - E.g. key part of JPEG and MP3 encoding
  - Now extended to not just even *N*, but to *N* which has prime factors of 2, 3, 5, …
  - Difference in speed wrt DFT ~$N/\log_2(N)$ so pad data with zeroes until *N* fits and then use FFT!

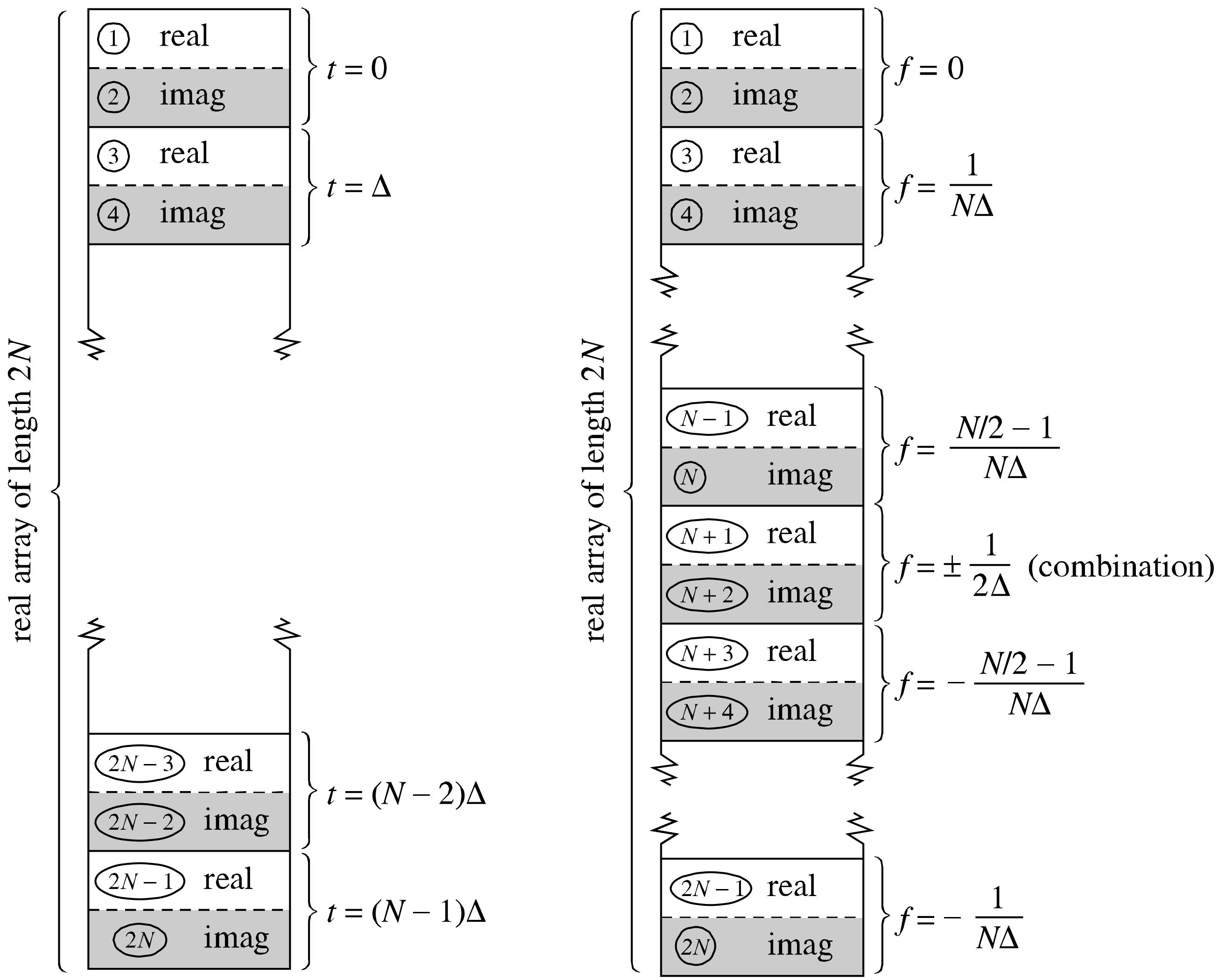
FFT data layout

- The recursive nature of the halving of the FFT means internal data layout is complex:



- Need bit reversal of indices for output data

FFT data layout



Input data
t=0 → (N-1)δ

Output data
f: -(N/2δ) → +(N/2δ)

Complex vs Real data

- In general, the input data has type=complex and so is output
- What if input data has type=real?
  - E.g. real-space charge density?
  - E.g. Γ-point calculations?
- Could either code up separate real ←→ complex transforms => messy & inefficient
- Or pack 2 real vectors into 1 complex vector (1=real, 2=imaginary) and do both at same time => fast and saves memory

Multidimensional FFT

- What about 2D or 3D data?
  - E.g. image manipulations or charge density …
- Easy to generalize analytical formulae and DFT/FFT to 2D and 3D and beyond
- Computationally see that each dimension is independent
  - Simplest to implement using row-column approach – e.g. 2D = set of 1D transforms
  - In 3D best to do 2D planes for fixed z and then iterate over z to get best cache reuse

Parallel FFT

- Two cases:
  - Doing a big FFT in parallel
    - Can exploit the recursive nature of FFT to split up long FFT into a sequence of shorter independent FFTs and hence parallelize
    - Useful for multi-core approach to handling few big FFTs e.g. in digital camera
  - Or doing FFT on data that is already parallel distributed
    - Which is what we have in CASTEP …

FFT in parallel CASTEP

Parallel FFT in CASTEP

- We need to do 3D real ←→ recip FFT at various points in CASTEP
  - E.g. calculations involving wavefunction, density, band overlap matrix, etc.
- And with **G**-vector parallelism we have the basic **G**-vectors and all associated data distributed across cores
  - How best to arrange the data?
  - Dictated by the FFT algorithm!

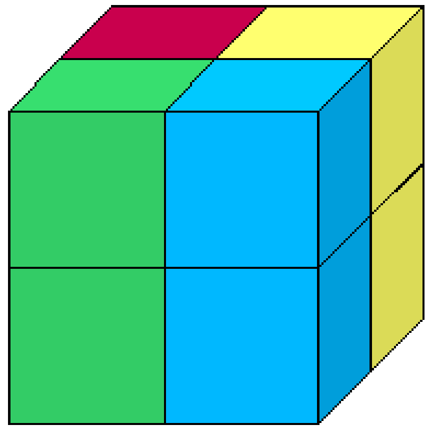
G-vector data distribution

- Distributed data:
  - Each core has some **G**-vectors
  - Each core has some **r**-space data
  - Time reversal symmetry: $\rho(\mathbf{G})=\rho(-\mathbf{G})$
- Fourier transform:
  - ALL **G**-vectors contribute to ALL points in **r**-space and vice versa
  - Hence requires a lot of data movement
  - FFT is a key communications bottleneck

G-vector parallelism

- Do 3D transform as set of 1D transforms
- Give each core all **G**-vectors in a z column
  - Each core does transform in z with own data
- All cores swap data so they have y columns
  - Each core does transform in y with own data
- All cores swap data so they have x columns
  - Each core does transform in x with own data
- Each core starts with **G**-space data in z and ends up with **r**-space data in x

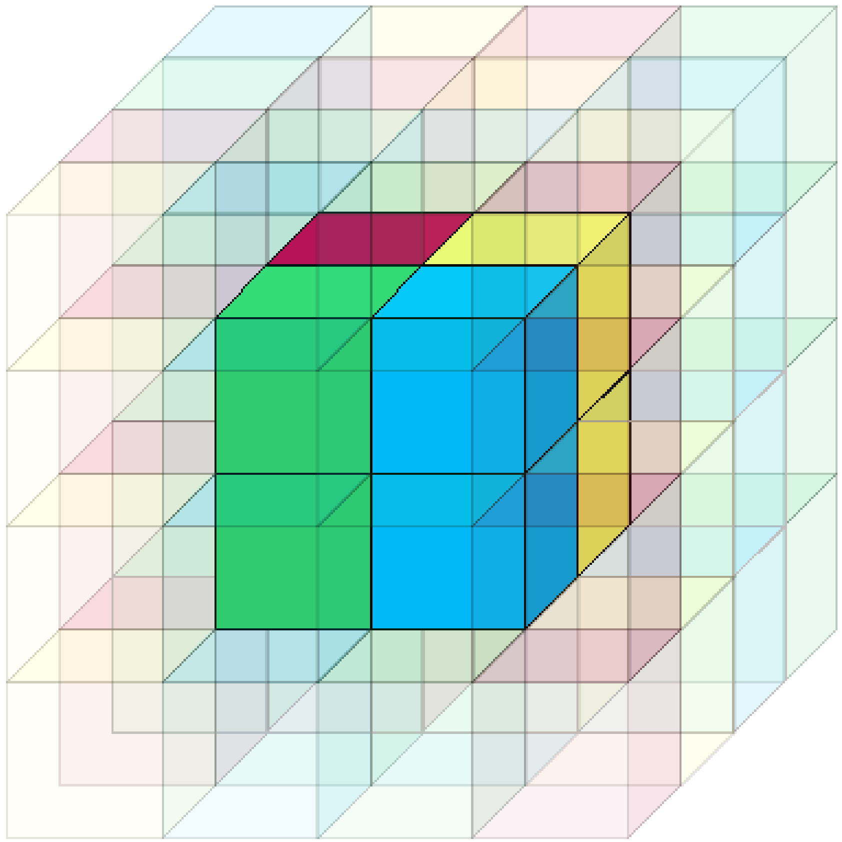
G-vector parallel FFT



Core 1
Core 2
Core 3
Core 4

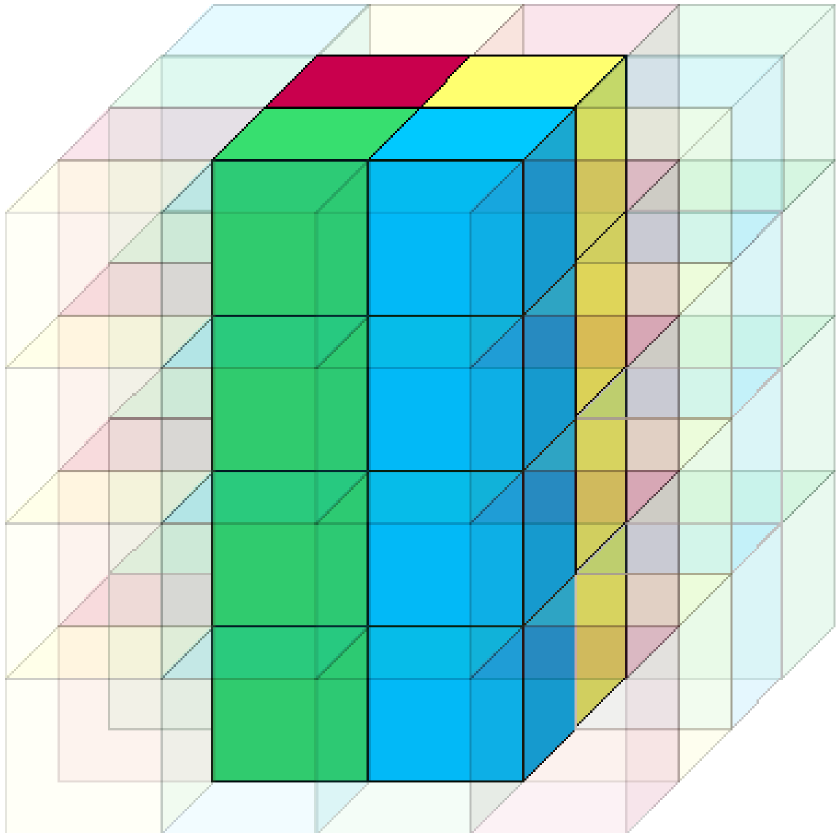
Start: **G**-vectors inside cut-off sphere $\longrightarrow$ put on grid.

G-vector parallel FFT



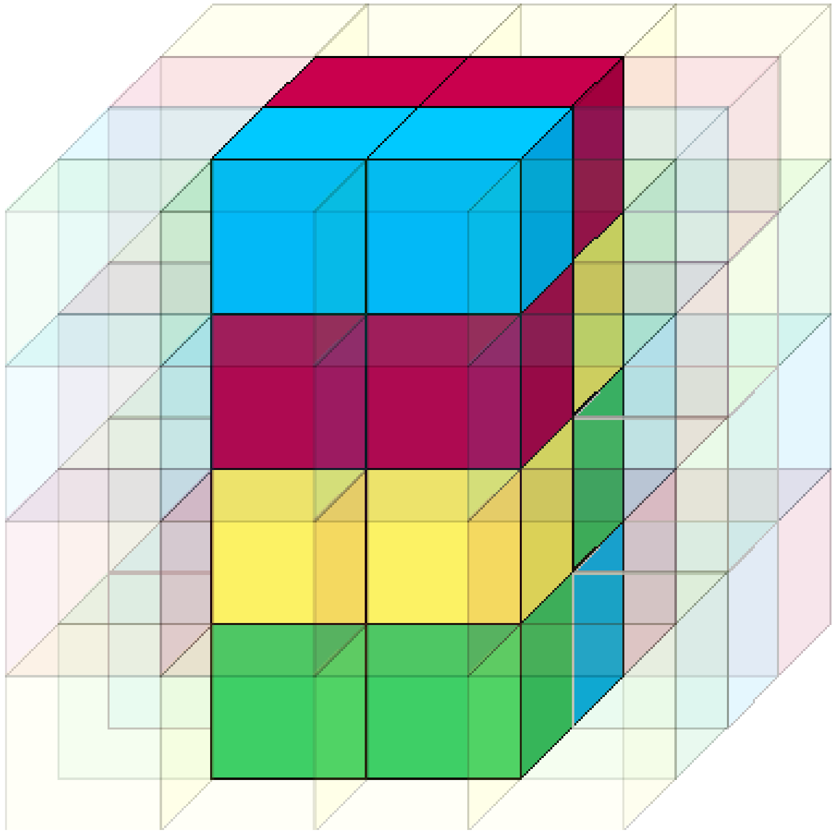
Now perform FFT in z-direction...

G-vector parallel FFT



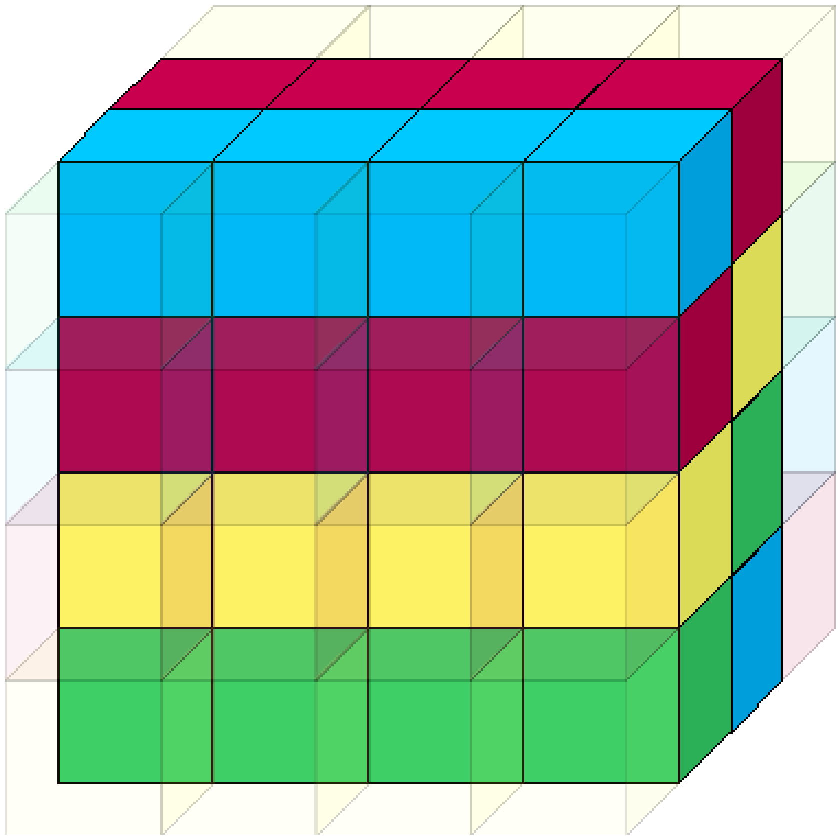
Transpose (swap) data into y-columns.

G-vector parallel FFT



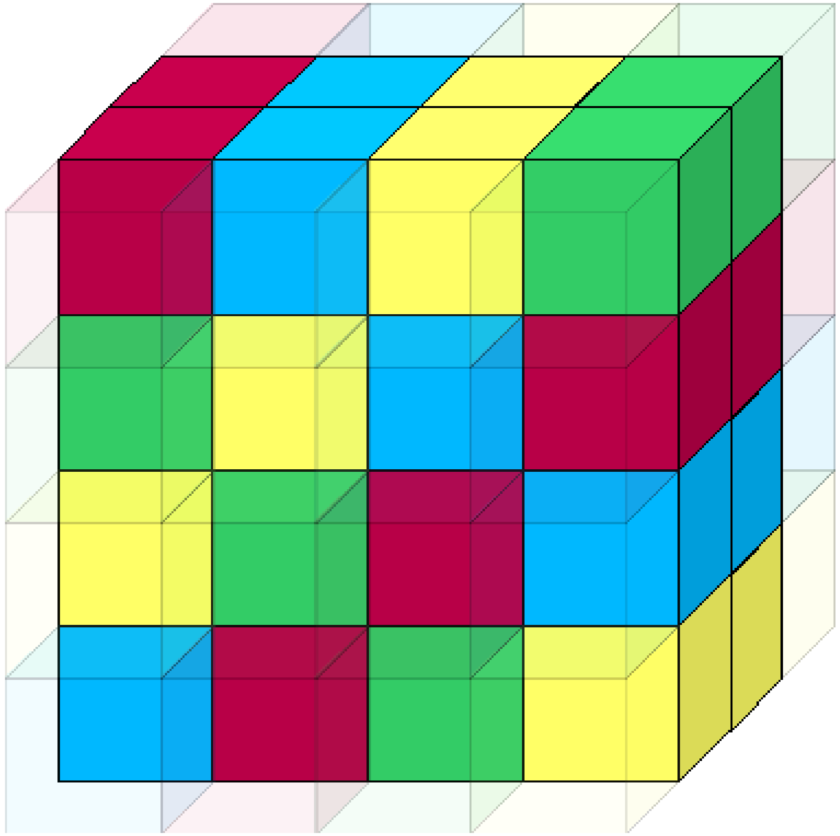
Now perform FFT in y-direction...

G-vector parallel FFT



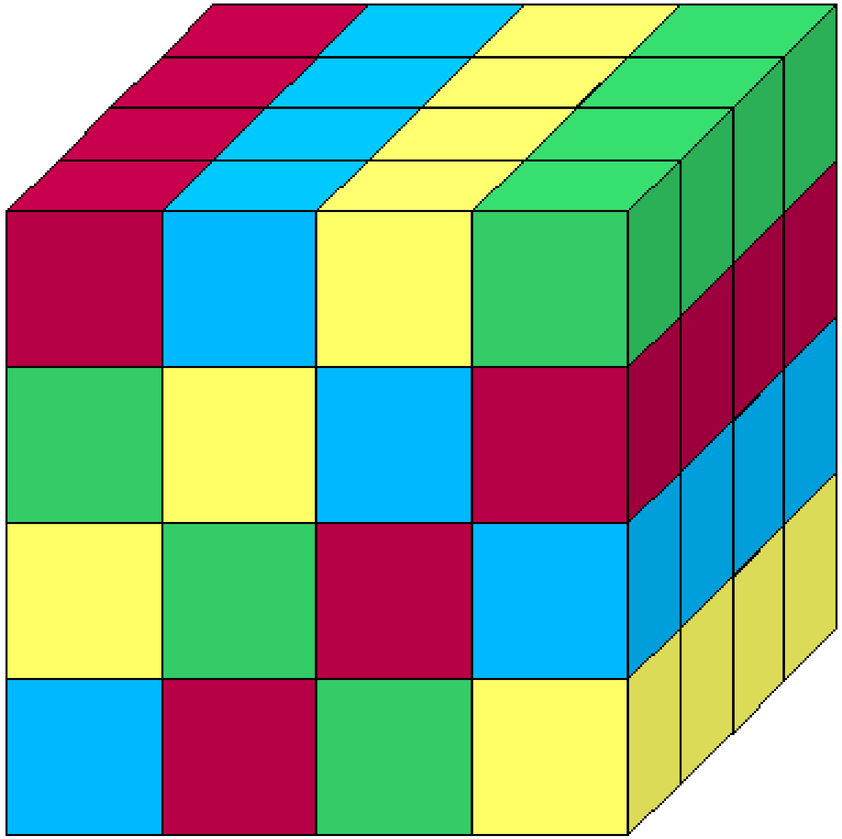
Transpose data into x-columns.

G-vector parallel FFT



Now perform FFT in x-direction...

G-vector parallel FFT



Now have real-space data in x-columns.

G-vector parallel consequences

- Each FFT is relatively fast and scales well
- But requires an *all-to-all* communication at each step to do the data transposition
  - Hence time scales as N^2_{core}
- As N_{core} increases the FFT process will take longer due to increasing comms time
- When comms time ~ calculation time then have reached the limit of scaling
  - More cores will make the calculation go slower!

Summary

Summary

- FFT is a key algorithm in many areas of science and technology
  - Good to understand how it works, and implications of aliasing, filtering etc
  - Key to efficient plane wave DFT
    - KE is diagonal in reciprocal space
    - PE is local in real space
  - But FFT on distributed data requires lots of communications
    - Ultimate limit to CASTEP scaling

References

- MC Payne et al., Rev. Mod. Phys **64**, 1045 (1992)
- WH Press et al, "*Numerical Recipes: The Art of Scientific Computing*", Cambridge University Press (1989 – 2007)
- JW Cooley and JW Tukey, Math. Comput. **19**, 297 (1965)