Numerical Algorithms for HPC

FFT Libraries

epcc

1

1

Overview

- Recap of cost of DFT and FFT
- Use of Libraries
- FFT Libraries
  - General usage notes
- FFTW
  - Overview
  - Using the Library
  - Performance
- FFT Practical

epcc

2

2

Recap: Naïve DFT cost

- DFT is a sum. For each of the N values of F_n there is a sum of length N to perform

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

- Cost is therefore $N \times N = N$ i.e. $\mathcal{O}(N^2)$

Recap: FFT cost

- FFT split sum into 2, then into 2 again, etc. E.g. for $N = 4$

f_0	f_2	f_1	f_3
$f_0 + f_2$	$f_0 - f_2$	$f_1 + f_3$	$f_1 - f_3$
$f_0 + f_2 + f_1 + f_3$	$f_0 - f_2 + if_1 - if_3$	$f_0 + f_2 - f_1 - f_3$	$f_0 - f_2 - if_1 + if_3$

Bit reversal: $\mathcal{O}(N)$

- $N = 4$ case: each of the $N = 4$ elements performs 2 sums
- The number of times N (if a power of 2) can be halved until you get to 1 is $\log_2 N$
  - E.g. for N=32: 32/2=16, 16/2=8, 8/2=4, 4/2=2, 2/1=1
  - Divides 5 times: $\log_2 32 = 5$ (i.e. $2^5 = 32$)
- So in general cost is $\mathcal{O}(N \log_2 N)$

Use of Libraries

- What is a library?
  - Collection of pre-implemented routines available in some sort of package (static archive, shared library, Framework, DLL…)
  - Defined (and hopefully documented) API
  - Often provided with operating system (e.g. Scientific Linux), available via package managers, or binary/source downloads
  - Can be combined with your code by linking (and perhaps #including header files)

5

Use of Libraries

- Why use libraries?
  - Easier - save effort and time, no duplicated code
  - Avoid bugs – libraries are (hopefully) well tested
  - Performance – usually contain efficient implementations, perhaps optimised for a given system (vendor-specific architectures), or auto-tuning

6

FFT Libraries

- FFTs do not normalise
  - Each FFT/Inverse FFT pair scales by a factor of N
  - Usually 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
  - May be 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 ($\mathcal{O}(N)$)
    - Around twice as fast as the simple solution
    - Good details in Numerical Recipes book
  - Some libraries have real-to-complex wrappers
  - Similar approach allows us to do 2 independent real-to-complex transforms simultaneously

FFT Libraries

- Multidimensional FFTs
  - simply successive FFTs over each dimension
    - order immaterial: linearly independent operations
  - can place data into 1D array
  - 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
  - 2+D parallel, array transpose operation
    - distributed data is collated on one processor before FFT
    - more in parallel libraries lecture later
  - some libraries have parallel FFT wrappers

FFTW

- 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 N is composed of powers of 2, 3, 5 and 7 (other sizes can be optimised for at configuration time)
    - Also 11 and 13 if to power 0 or 1
  - Powers of 2 particularly fast!

FFTW: FFTW2 vs FFTW3

- Previous version: "FFTW2"
  - Many legacy codes employ FFTW2
  - Simple(r) C interface, with wrappers for many other languages
  - Supports MPI
- New version: "FFTW3"
  - Different interface to FFTW2, though not radically – to allow planner more freedom to optimise
    - Users must rewrite code
  - Now has MPI support
  - Somewhat faster than FFTW2 (~10% or more)
  - Rest of this lecture assumes "FFTW3"

FFTW: Details

- Can perform FFTs on distributed data
  - MPI for distributed memory platforms
  - OpenMP or POSIX for SMPs
- If users rewrite their code to use FFTW 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
    - location of FFTW libraries may vary
- Fortran API is available, though modern best practice is to use bindings to C.
- FFTs can not always be done "in-place"
  - The input and output arrays must be separate and distinct or temp arrays created

FFTW: Plans

- All FFT libraries pre-compute the *twiddle factors*
- FFTW 'plans' also generate the FFT code from *codelets*, hard-coded small transforms
  - 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.
    - Variants widen or restrict range of algorithms to consider.
- Old plans can be reused or even read from file: *wisdom*

FFTW: Installation

- Download library from the website and unpack (gzipped tar file)
- **`./configure; make; make install`**
  - Probes the local environment
  - Compiles the many *codelets* (small C object codes)
  - 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
- Pre-installed with Scientific Linux
- You won't have to install FFTW for exercise – it's already available on Cirrus via a module:
  `module load fftw/3.3.8-gcc8-ompi4`

FFTW: Compiling code

- **`gfortran fft_code.f -O3 -lfftw3`**
  - If using C, FFTW must be linked with **`-lfftw3 -lm`**
  - Different precision libraries may be available (**`fftw3f`** and **`fftw3l`** for float and long double respectively).
  - May need to include location of libraries (**`-L/`**…) or include files (**`-I/`**…).
- Example Fortran code using C bindings:

```
type(c_ptr) :: plan
integer(c_int), parameter :: n = 1024
complex(c_double_complex) :: in(n), out(n)
! plan the computation
plan = fftw_plan_dft_1d(…)
! execute the plan
call fftw_execute_dft(…)
```

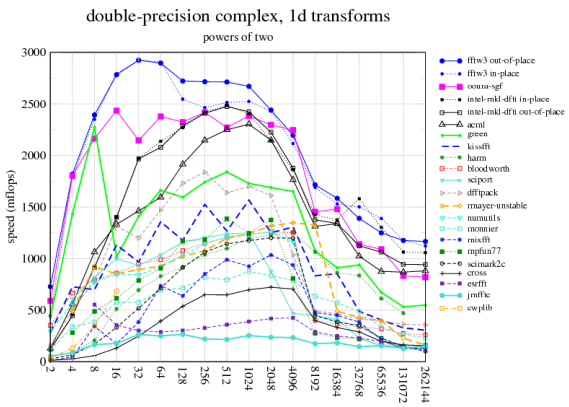
  - NB: full incantations are not given here as reading documentation is integral to utilising any numerical library

FFTW: 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/

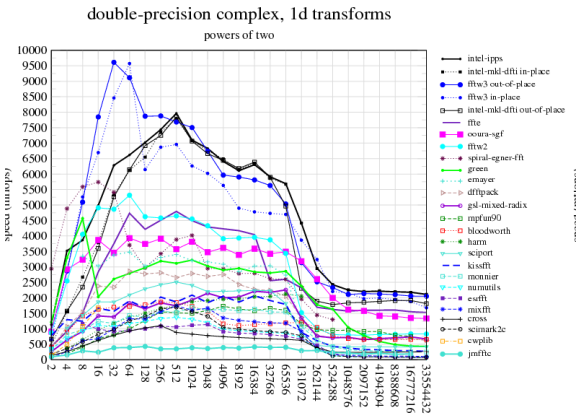
epcc

FFTW: AMD Opteron 275 2.2 GHz



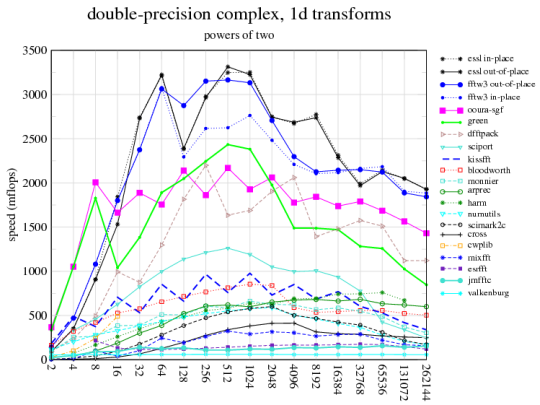
epcc

FFTW: Intel Core Duo 3.0 GHz



17

FFTW: IBM POWER5 1.65 GHz



18

FFTW: 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.
  - "Its 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. Its 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)."

FFT Summary

- First lecture:
  - Introduced both the continuous and discrete forms of the Fourier Transform
  - Fast Fourier Transform
- Second lecture:
  - FFTW
    - Fast, robust and portable
    - Fortran and C, serial and parallel.
    - Simple to use
    - Recommended and used in major projects by EPCC

FFT Practical

- FFTs are also employed in JPEG image compression and smoothing…
  - MPEG also utilises FFTs along with other graphical techniques
- Examine how manipulating image data in Fourier space alters the transformed image using FFTW 3

epcc

FFT Practical

- Full instructions in practical handout sheet
- Two main image transformations to attempt:
  - How many higher frequencies can be removed, reducing the size of the image in memory, without visibly altering the image?
  - Performing edge detection by removing only lower frequencies.
- Hint: remember that Fourier space is periodic
- Begin with an estimated plan and a 1D, non-strided, complex-to-complex transform

epcc

FFT Practical

- Fortran and C makefiles and templates available
- For exercises
  - Fortran routines with C bindings:
    - `fftw_plan_dft_1d, fftw_execute_dft, fftw_destroy_plan`
  - C routines:
    - `fftw_plan_dft_1d, fftw_execute, fftw_destroy_plan`
  - NB: Read the documentation for arguments, usage etc.
- Online documentation on FFTW
  - the `info fftw` command or www.fftw.org

FFT Practical : Further work

- How competitive is FFTW compared to other FFT libraries?
  - Alter previous code and time sections for plans and FFTs for FFTW and FFTPACK, say.
    - At present, FFTW is the only FFT library available
    - If interested, install FFTPACK in your home space
  - Extend data array to compare FFT libraries over a range of N
    - Including non-power-of-two N
    - fill array with zeros first
- Convert image manipulation code to use
  - 2D FFTW
  - Investigate the use of "wisdom"