Numerical Algorithms for HPC

Parallel Fourier Transforms

|epcc|

1

1

Overview

- Parallel FFT in 1 Dimension – shared memory
- Parallel Fourier Transformations of 2D arrays
- Intro to FFTs of 3D arrays

|epcc|

2

2

Fourier Transformation

- FFTs are often “the” critical bottleneck
  - preventing parallel application from scaling to larger numbers of processors due to communications
- This lecture discusses reasons and how we might parallelise an FFT to overcome this

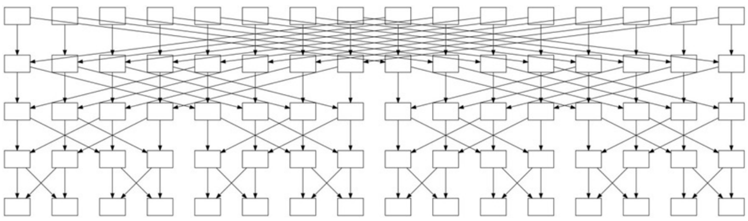
epcc

3

3

Parallel 1D FFT

- Parallelisation of a 1D FFT is hard
  - Combining of data requires a lot of inter-processor communication



- Typically `N≈100-200` in many scientific codes e.g. materials chemistry – small amount of data
- Algorithm is hard to decompose
- Literature examples:

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

Tang et al, “A Framework for Low-Communication 1-D FFT”, SC12, https://software.intel.com/sites/default/files/bd/8b/fft-1d-framework.pdf

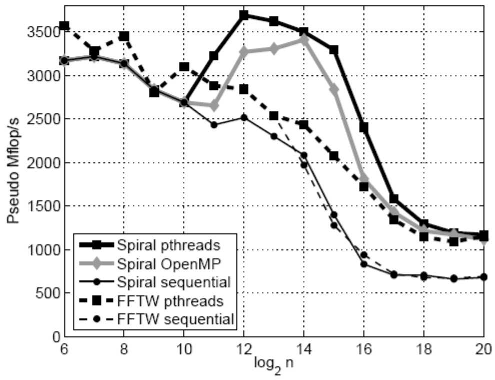
epcc

4

4

"Traditional" SMP

- 4 processor Intel Xeon
  - Communication via shared Memory (Bus)
- Benefits from:
  - N=2048 (Spiral)
  - N=16384 (FFTW)
- Improvement for large problems (Factor about 2 for 4 CPU)

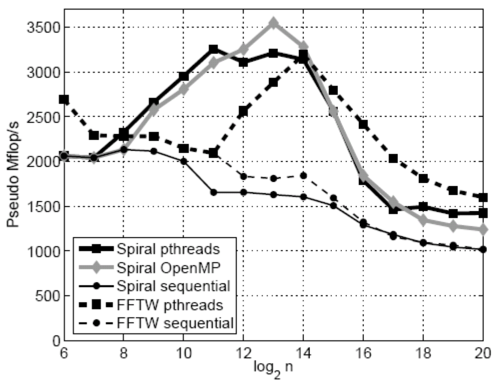


(d) 2.8 GHz Xeon MP (4 processors)

epcc

Multicore Processor

- Intel Core Duo (laptop)
  - Shared L2 used for communication
- Benefits from
  - N=512 (Spiral)
  - N=4096 (FFTW)
- Not as efficient for huge problems

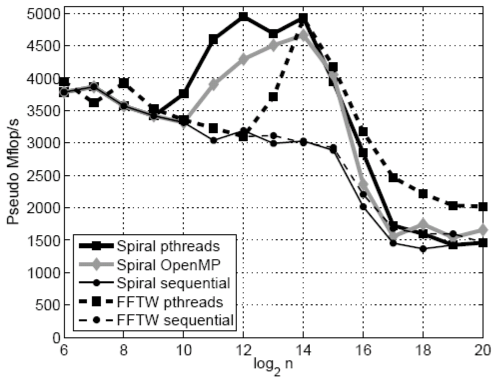


(a) 2.0 GHz Core Duo (2 processors)

epcc

Multicore processor with shared bus

- Intel Pentium D
  - Multicore chip
  - Communication via Bus
- Benefits from:
  - N=2048 (Spiral)
  - N=8192 (FFTW)
- Little benefit for huge problems (shared bus)



(c) 3.6 GHz Pentium D (2 processors)

epcc 7

7

Summary: 1D parallel FFT

- Parallelisation works for large problems only ☹
- Sensitive to contention (shared buses) ☹
- Multicore chips with communications at cache level appear beneficial – might "be there" in a few years time
- Shows speedup, but not always "perfect" 😐
- Presently: **1D FFT is an "expensive sum" of an array which is hard to parallelise**

epcc 8

8

FFTs in two dimensions

- What needs calculating for a 2D FFT:

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

- Do it in a 2 step approach:

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

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

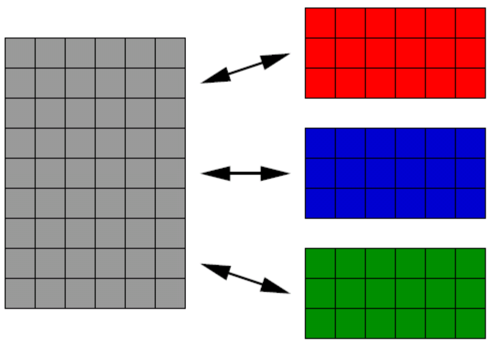
epcc

9

9

Distribute array onto 1D processor grid

- Example:
  - 6 × 9 array
  - 3 processors
  - Assuming row major order (C convention)



- Perform 1st FFT:
  - Each processor transforms 3 arrays of 6 elements
- What next?

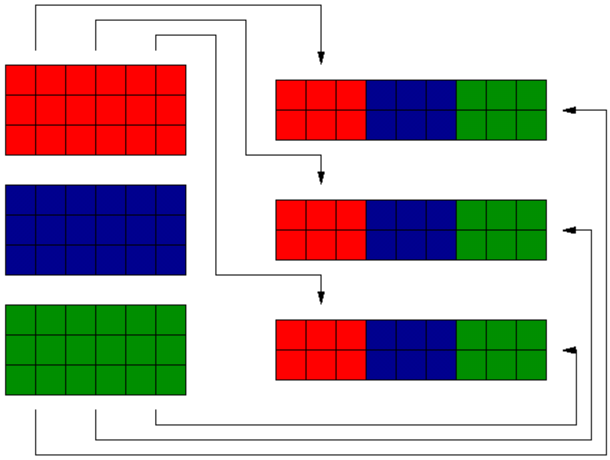
epcc

10

10

Data transposition

- Divide up the array by columns for 2nd FFT
- Depending on FFT library simultaneous transpose can be advantageous (shown on figure)



epcc

11

11

Perform 2nd FFT

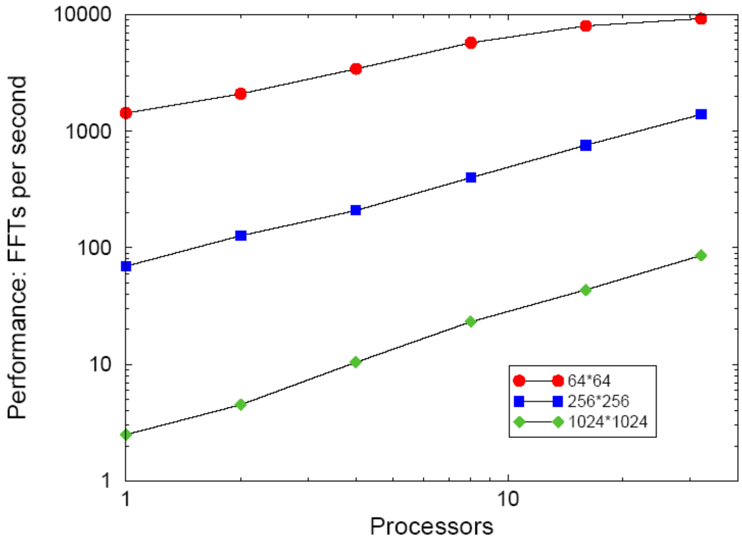
- What used to be the columns of the original array are now in row-major order ☺
- Do the 2nd FFT
  - In the example:
    - Each processor performs 2 FFTs of an array of length 9
- Rearrange data as required by following code
  - Examples:
    - Undoing the transpose
    - Redistributing data onto 2D grid
    - Sometimes: nothing needs to be done ☺

epcc

12

12

Example 2D-FFT on 32 BlueGene/L CPUs



epcc 13

13

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

epcc 14

14

FFT: Decomposition

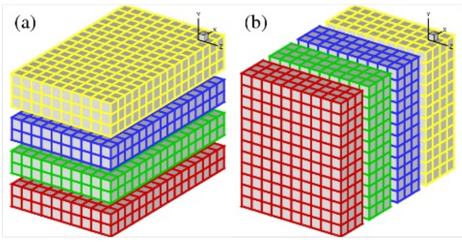
- For a *d*-dimensional problem we decompose in up to *d*-1 dimensions
  - i.e. one dimension should be left “intact” at any one time so that 1D FFTs can be performed on it
  - E.g. for a 2D problem we can parallelise over rows, carry out an FFT on each row, then transpose and do the same again
- For 3D, we have a choice of whether to do a 1D processor decomp (“slab”) or a 2D decomp (“pencil”)

epcc

15

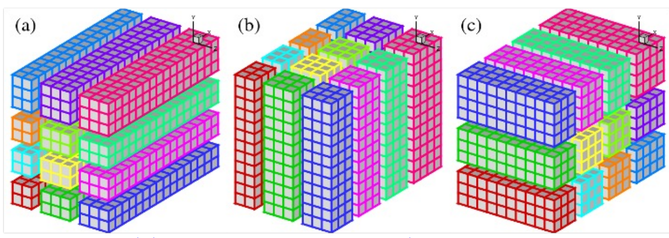
15

FFT Slab and Pencil decomposition



Slab: decomp in Y and X directions using 4 cores

Pencil: (a) X-, (b) Y- and (c) Z-pencils using 12 cores



Images from 2DECOMP&FFT library http://www.2decomp.org/decomp.html

epcc

16

16

FFT: Slab vs Pencil

- Slab
  - Pros: Simple with moderate amount of inter-processor communication
  - Cons: Limited to N procs for N^3 data
- Pencil
  - Pros: faster on massively parallel supercomputers (i.e. lots of cores)
  - Cons: More communications and now more complicated
- Pencil generally better with high core count but not so good for larger arrays on moderate number of cores
- Note: FFTW only does slab!

epcc 17

17

Summary

- Parallelisation of an individual 1D FFT is hard
  - Presently works best for large problems
  - Recent advances in algorithms & hardware encouraging
- Multidimensional problems need to calculate many 1D FFTs
  - Parallelisable by distributing entire FFTs onto the processors and using a standard serial 1D FFT library
  - Requires redistributing the data between FFT dimensions
  - Need to think about decomposition (e.g. slab vs pencil)
- FFT can be used to reduce computational complexity of Fourier transform calculations from $O(n^2)$ to $O(n \log(n))$
  - Applications in signal processing, CFD, probability, etc.

epcc 18

18