2DECOMP&FFT – A Highly Scalable 2D Decomposition Library and FFT Interface

Ning Li and Sylvain Laizet

CUG 2010
Edinburgh
24-27 May 2010
Background Information

- HECToR dCSE project ongoing
  - dCSE - dedicated software engineering support to UK research community
  - Support Imperial-based Turbulence, Mixing and Flow Control group, improving a CFD code Incompact3D
  - Opportunities identified to develop reusable software components for a wider range of applications

- Parallel library development
  - A general-purpose 2D decomposition library
    - For applications based on 3D Cartesian data structures
  - A distributed 3-dimensional FFT library
  - A distributed FFT-based Poisson solver
Scientific Applications

- Flow passing through multi-scale fractal grid
- Energy-efficient way to generate turbulence
- Very fine grid (~billions) required for such simulations
Algorithms and Parallel Solutions

- **Incompact3D uses**
  - Compact Finite Difference method → $af'_{i-1} + bf'_{i} + cf'_{i+1} = \text{RHS}$
  - Pressure Poisson solver → 3D FFT → multiple 1D FFTs
  - All values along a global mesh line involved

- **General parallel solutions**
  - Parallelise the elementary algorithms
    - Distributed tri-diagonal solver
    - Distributed 1D FFT
  - Redistribute the data among multiple domain decompositions
    - Often the preferred method
    - Well-developed serial algorithms can be kept unchanged
1D Decomposition

- **Two slab decompositions**

- **Procedure**
  - (a) operate locally in X, Z
  - Transpose to state (b)
  - (b) operate locally in Y
  - Transpose back to state (a)

- **Limitation**
  - For $N^3$ mesh, $N_{\text{proc}} < N$
  - Also memory limit

  Typical Incompact3D simulation
  - $2048 \times 512 \times 512$
  - $N_{\text{proc}} < 512$
  - On HECToR
  - 200,000 time steps at 4 seconds each
  - 25 days wall-clock time
  - (excluding queueing time)
2D Decomposition

- Also known as **pencil** or **drawer** decomposition
- Local operations in one direction at a time
- Transpose
  - (a) $\Leftrightarrow$ (b) $\Leftrightarrow$ (c) $\Leftrightarrow$ (b) $\Leftrightarrow$ (a)
  - Communication among sub-groups only
- **Constraint relaxed to** $N_{\text{proc}} < N^2$ for cubic mesh
Why a Library Solution?

- Many applications.
- For a given global data structure and a given domain decomposition strategy, the corresponding data movement strategy should be identical.
- The implementation is a purely software engineering issue (not relevant to the scientific topics being studied).
- The proper implementation is not easy but important for performance reason.
Transpose from Y-pencil to Z-pencil

MPI_ALLTOALLV(sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, recvtype, comm)

- Best buffer gathering / scattering strategy?
- Optimisation opportunity?
Transpose from X-pencil to Y-pencil

(a) SRC(i,j,k)

(b) SEND_BUF

(c) RECV_BUF

(d) DST(i,j,k)

Gather

Scatter

MPI_ALLTOALLV
Decomposition API

- **Fortran module**
  - `use decomp_2d`

- **Global variables**
  - Starting/ending index and size of the sub-domain held by current rank, required to define application data structures
    - `allocate(in(xsize(1),xsize(2),xsize(3)))`
    - `allocate(out(ystart(1):yend(1),
                  ystart(2):yend(2), ystart(3):yend(3)))`

- **Public subroutines**
  - `decomp_2d_init(nx,ny,nz,p_row,p_col)`
  - `transpose_x_to_y(in,out); transpose_y_to_z(in,out)`
  - `transpose_z_to_y(in,out); transpose_y_to_x(in,out)`
  - `decomp_2d_finalize`
Shared-memory Implementation

- ALLTOALL(V) can be very expensive.
- Supercomputers prefers a small number of large messages.
- HECToR has 8GB memory shared by 4 cores.
- Cores on same node copy data to/from shared buffers.
- Only leaders of the nodes participate in communications.

- Implemented using System V IPC shared-memory API.
- Transparent to applications (switch on by a compiler flag).
- Originally based on Cray’s code (D. Tanquerey).
- Portable implementation using Ian Bush’s FreeIPC.
Shared-memory Performance

- Performance improvement for smaller message size
- Potential on next-generation hardware (24-core HECToR)
## Overview of Distributed FFT Libraries

<table>
<thead>
<tr>
<th>FFT Library</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFTW 2.x</td>
<td>MPI interface with 1D decomposition</td>
</tr>
<tr>
<td>FFTW 3.x</td>
<td>( \alpha )-version MPI interface</td>
</tr>
<tr>
<td>CRAFFT (xt-libsci)</td>
<td>Evenly distributed 1D decomposition</td>
</tr>
<tr>
<td>Plimpton’s parallel FFT*</td>
<td>Complex-to-complex transforms only</td>
</tr>
<tr>
<td>Takahashi’s FFTE #</td>
<td>Evenly distributed data; small prime factors</td>
</tr>
<tr>
<td>P3DFFT #</td>
<td>Real-to-complex/complex-to-real transforms only</td>
</tr>
</tbody>
</table>

- # based on 2D decomposition
- * user-callable communication routines
- All with some limitations
- Having developed the underlying decomposition library, building a distributed FFT library on top is easy
P3DFFT

- P3DFFT
  - Open-source software by Pekurovsky (SDSC)
  - Only r2c/c2r transforms
  - Private data transposition routines

- Application
  - Turbulence research using spectral DNS code by Yeung, et al.
  - Internally using P3DFFT

- Aim to achieve at least similar scaling
Distributed FFT API

- Fortran module
  - use decomp_2d_fft

- Public subroutines
  - decomp_2d_fft_init
    - By default, physical space in X-pencil, spectral space in Z-pencil
    - Optional parameter to use the opposite
  - decomp_2d_fft_3d (generic interface)
    - (complex in, complex out, direction) complex to complex
    - (real in_r, complex out_c) real to complex
    - (complex in_c, real out_r) complex to real
  - decomp_2d_get_fft_size (allocate memory for c2r/r2c)
  - decomp_2d_fft_finalize
Implementing Distributed FFTs

- **Complex to complex (c2c) -- easy**
  - Update decomposition routines to support complex data type (Fortran generic interface)

- **Real-to-complex (r2c) and complex-to-real (c2r)**
  - Data storage considering conjugate symmetry
  - For nx real input $r_k$, the complex output: $c_k = a_k + ib_k$
    - (1) also nx real numbers (Hermitian storage)
    - (2) nx/2+1 complex numbers – easier to extend to multi-dimension

<table>
<thead>
<tr>
<th></th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
<th>r6</th>
<th>r7</th>
<th>r8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c1</td>
<td>c2</td>
<td>c3</td>
<td>c4</td>
<td>c5</td>
<td>c6=c4</td>
<td>c7=c3</td>
<td>c8=c2</td>
</tr>
<tr>
<td>(1)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
<td>b4</td>
<td>b3</td>
<td>b2</td>
</tr>
<tr>
<td>(2)</td>
<td>c1</td>
<td>c2</td>
<td>c3</td>
<td>c4</td>
<td>c5</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
Extension of Base Communication Library

- **Requirement**
  - FFT real input: \( nx \times ny \times nz \); complex output: \( (nx/2+1) \times ny \times nz \)
  - Both need to be distributed as 2D pencils

- **Solution**
  - Object-oriented style design
  - Store decomposition information per global size in a Fortran derived data type
    - Containing sub-domain sizes; starting/ending indices; Mesh distribution and MPI_ALLTOALLV buffer parameters; etc.
    - \texttt{TYPE(DECOMP_INFO) :: decomp}
    - \texttt{call decomp_info_init(nx,ny,nz,decomp)}
  - Optional third parameter to transposition routines
    - \texttt{call transpose_x_to_y(in,out,decomp)}
Other Multi-global-size Examples

- Plane-wave electronic structure calculations
  - Fourier space confined in a sphere of diameter $d$
  - Real space in a $2d^3$ cube
  - Only transpose non-zero data to improve efficiency
    - $d*d*2d$; $d*2d*2d$

- CFD application using staggered mesh
  - Cell-centre variables and cell-interface variables different global sizes

nag®
# FFT Engines

- Distributed library performs data management only.
- Actual 1D FFT delegates to a third-party FFT library.
- Multiple third-party libraries supported.

<table>
<thead>
<tr>
<th>Library</th>
<th>Open-source</th>
<th>Hardware-tuned</th>
<th>Programming experience</th>
<th>Easy parallel coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic (default)</td>
<td>Y</td>
<td>N</td>
<td>Slow but no dependency on external library</td>
<td>Y</td>
</tr>
<tr>
<td>FFTW 3.x</td>
<td>Y</td>
<td>Auto-tuning</td>
<td>Planning hard to use in parallel coding</td>
<td>N</td>
</tr>
<tr>
<td>ACML</td>
<td>N</td>
<td>For AMD</td>
<td>Limited r2c/c2r API</td>
<td>Y</td>
</tr>
<tr>
<td>fftpack</td>
<td>Y</td>
<td>N</td>
<td>Slow but used in many legacy applications</td>
<td>Y</td>
</tr>
<tr>
<td>MKL</td>
<td>N</td>
<td>For Intel</td>
<td>Flawed API design</td>
<td>Y</td>
</tr>
<tr>
<td>ESSL</td>
<td>N</td>
<td>For IBM</td>
<td>Limited transform lengths</td>
<td>N</td>
</tr>
</tbody>
</table>
## FFT Library Performance

<table>
<thead>
<tr>
<th>N^3</th>
<th>Serial FFTW</th>
<th>Distributed (FFTW engine)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Planning</td>
<td>Execution</td>
</tr>
<tr>
<td>64^3</td>
<td>0.359</td>
<td>0.00509</td>
</tr>
<tr>
<td>128^3</td>
<td>1.98</td>
<td>0.0525</td>
</tr>
<tr>
<td>256^3</td>
<td>8.03</td>
<td>0.551</td>
</tr>
<tr>
<td>512^3</td>
<td>37.5</td>
<td>5.38</td>
</tr>
<tr>
<td>1024^3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2048^3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Problem size increased by 8.
- Serial FFTW’s execution time increased by ~10.
- Distributed FFT follows serial trend.
FFT Library Scaling

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.
Distributed Poisson Solver

- Fourier-based matrix decomposition method
  - Idea:
    - Finite difference discretisation of 3D Poisson results in matrix with 7 diagonal lines
    - Applying FFT in one dimension $\rightarrow$ 2D pentadiagonal systems
    - Applying FFT in a second dimension $\rightarrow$ 1D tridiagonal systems
    - FFT in X $\rightarrow$ FFT in Y $\rightarrow$ tridiagonal solver in Z $\rightarrow$ Inverse FFT in Y $\rightarrow$ Inverse FFT in X
  - Non-periodic data sets
    - Discrete sine/cosine/quarter-wave transforms
    - Passed to standard FFT library with pre- & post-processing
  - Library code available: FISHPACK, FFTPACK
  - Fit in current framework for parallelisation
Distributed Poisson Solver (2)

- Fourier-based spectral method
  - Algorithm
    - Pre-processing in physical space
    - 3D forward FFT
    - Pre-processing in spectral space
    - Solve Poisson by division of modified wave numbers
    - Post-processing in spectral space
    - 3D inverse FFT
    - Post-processing in physical space
  - Standard 3D FFT in use even with non-periodic data sets
  - Pre- and post-processing can be local (done in any pencil orientation) or global (data transpositions required)
# Poisson Solver Performance

<table>
<thead>
<tr>
<th>Boundary Condition</th>
<th>Global Transpositions</th>
<th>1024^3 case on 128 cores</th>
<th>2048^3 case on 1024 cores</th>
<th>4096^3 case on 8192 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>FFT only</td>
<td>4.81</td>
<td>6.26</td>
<td>7.59</td>
</tr>
<tr>
<td>100</td>
<td>FFT + 8</td>
<td>7.38</td>
<td>10.38</td>
<td>14.41</td>
</tr>
<tr>
<td>010</td>
<td>FFT + 6</td>
<td>6.81</td>
<td>8.86</td>
<td>12.63</td>
</tr>
<tr>
<td>110</td>
<td>FFT + 12</td>
<td>8.23</td>
<td>11.56</td>
<td>16.31</td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>8.41</td>
<td>11.67</td>
<td>16.48</td>
</tr>
</tbody>
</table>

- **Boundary conditions:**
  - 0 – periodic
  - 1 – homogeneous Neumann (symmetric)
- **FFT (forward + inverse) contain 4 global transpositions**
- **Computationally dominant algorithm even with extra communications**
Putting all together

- CFD code Incompact3D uses
  - Explicit data transpositions for its finite difference part when
    - Computing spatial derivatives
    - Doing spatial interpolations
    - Doing spatial filtering
  - A modified version of the Poisson solver for pressure Poisson problem
    - Indirectly using the FFT library
  - In total up to 66 transposition calls per time step
  - An I/O library, also built using the decomposition data
Incompact3D Strong Scaling on HECToR

![Graph showing strong scaling performance](image)
Incompact3D Weak Scaling on HECToR

![Graph showing time per step (second) vs. number of cores for different configurations of Incompact3D. The graph includes data points for New Incompact3D with 4 cores, Old Incompact3D with 4 cores, and New Incompact3D with 2 cores. Each point represents 4191304 points per MPI rank.](image)

- New Incompact3D - 4 cores
- Old Incompact3D - 4 cores
- New Incompact3D - 2 cores

4191304 points / MPI rank
Summary

- Highly scalable and user-friendly 2D decomposition library and distributed FFT library developed.
- Framework for parallelising other algorithms as long as they are
  - Based on 3D Cartesian data structures
  - Operating on direction by direction basis
- Very successful application in Incompact3D

- Source code available upon request
  - Email ning.li@nag.co.uk
  - Collaboration opportunities?
Questions?