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

Ning Li and Sylvain Laizet

CUG 2010
Edinburgh
24-27 May 2010

nag® Experts in numerical algorithms and HPC services

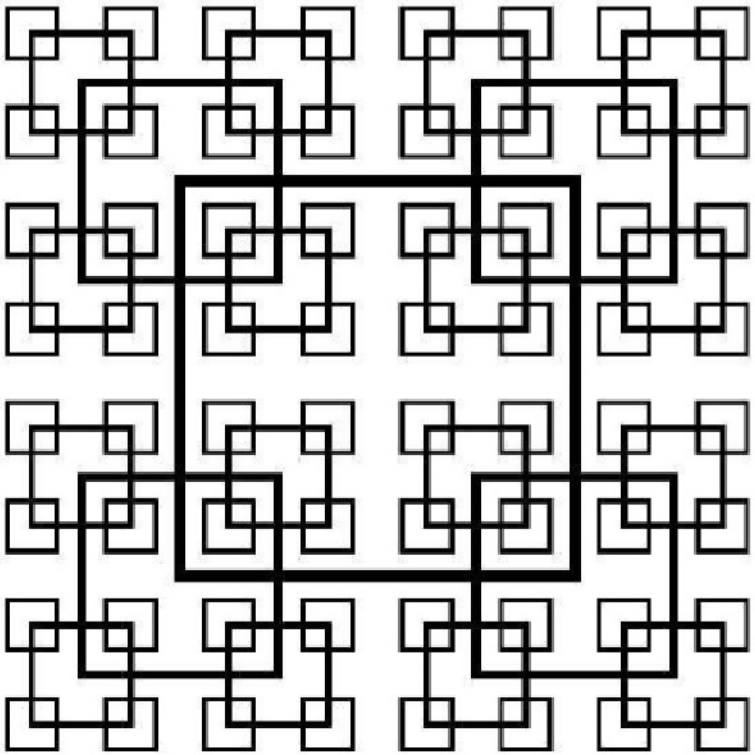
HECToR

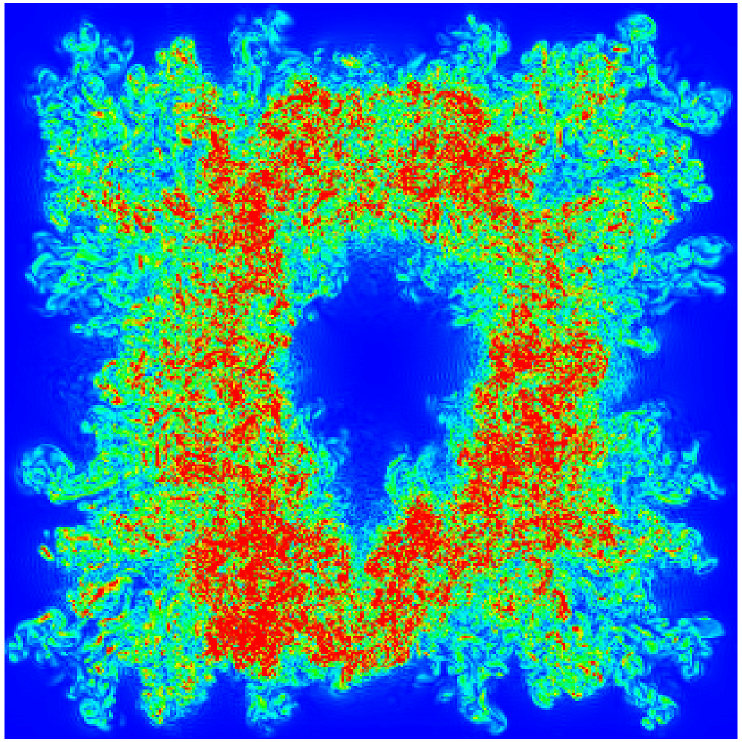
RESEARCH COUNCILS UK

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





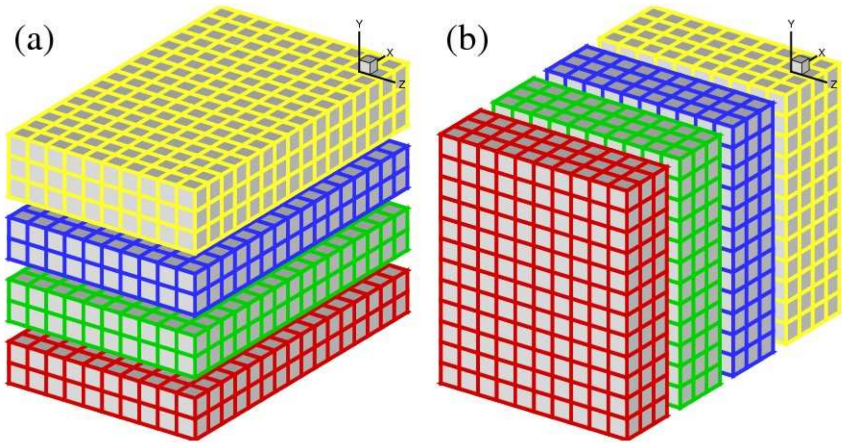
- Flow passing through multi-scale fractal grid
- Energy-efficient way to generate turbulence
- Very fine grid (~billions) required for such simulations

Algorithms and Parallel Solutions

- Incompact3D uses
  - Compact Finite Difference method → $af'_{i-1}+bf'_{i}+cf'_{i+1} = RHS$
  - 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

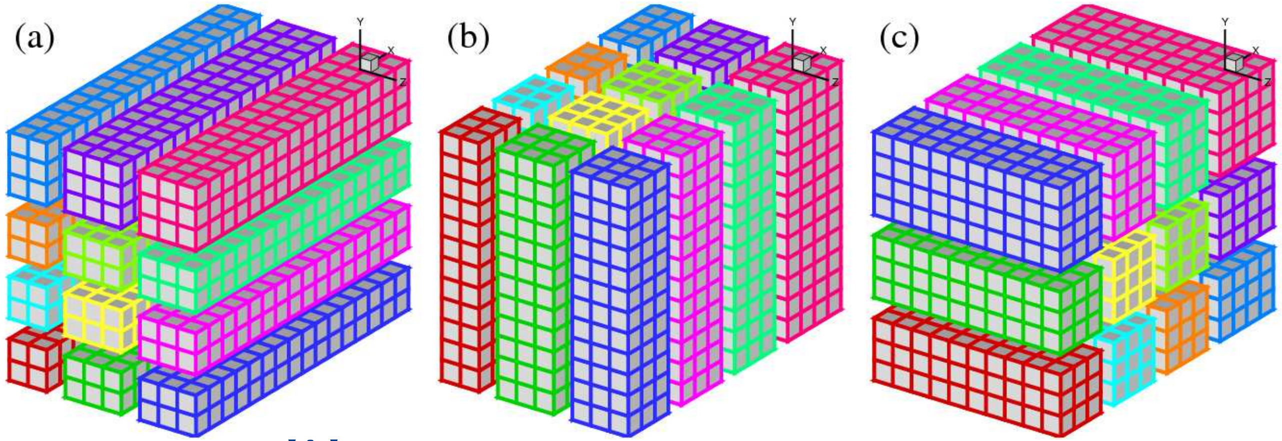
1D Decomposition

- 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_proc < N
  - Also memory limit



Typical Incompact3D simulation
2048*512*512
N_proc < 512
On HECToR
200,000 time steps at 4 seconds each
25 days wall-clock time
(excluding queueing time)

2D Decomposition



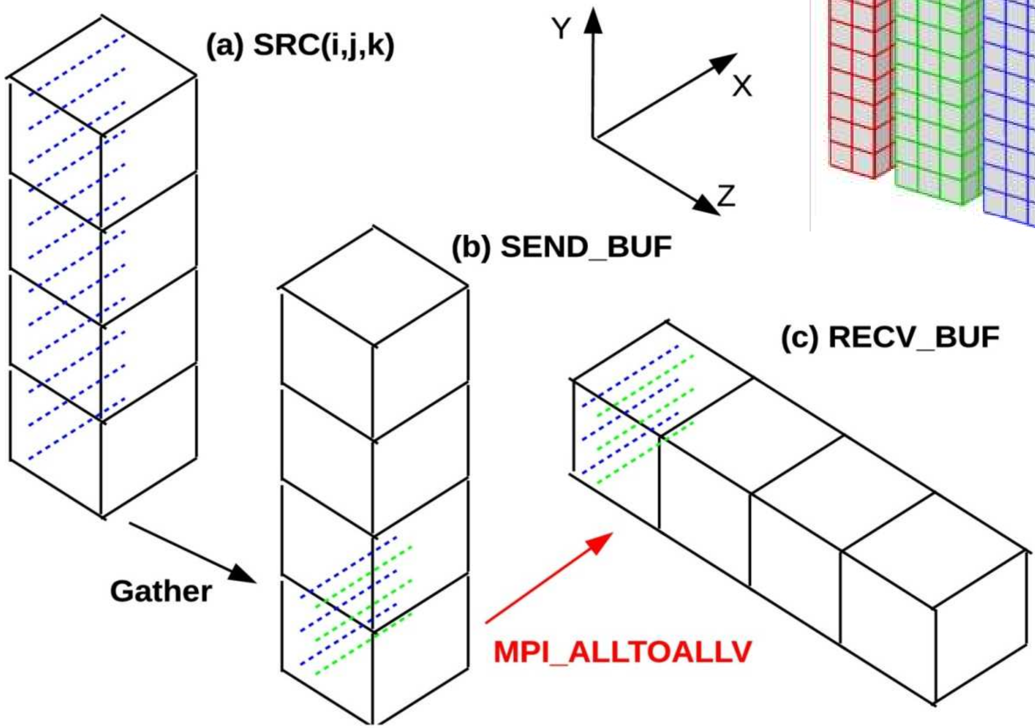
- 2D Decomposition
  - Also known as **pencil** or **drawer** decomposition
  - Local operations in one direction at a time
  - Transpose
    - (a) ⇔ (b) ⇔ (c) ⇔ (b) ⇔ (a)
    - Communication among sub-groups only
  - Constraint relaxed to N_proc < N^2 for cubic mesh

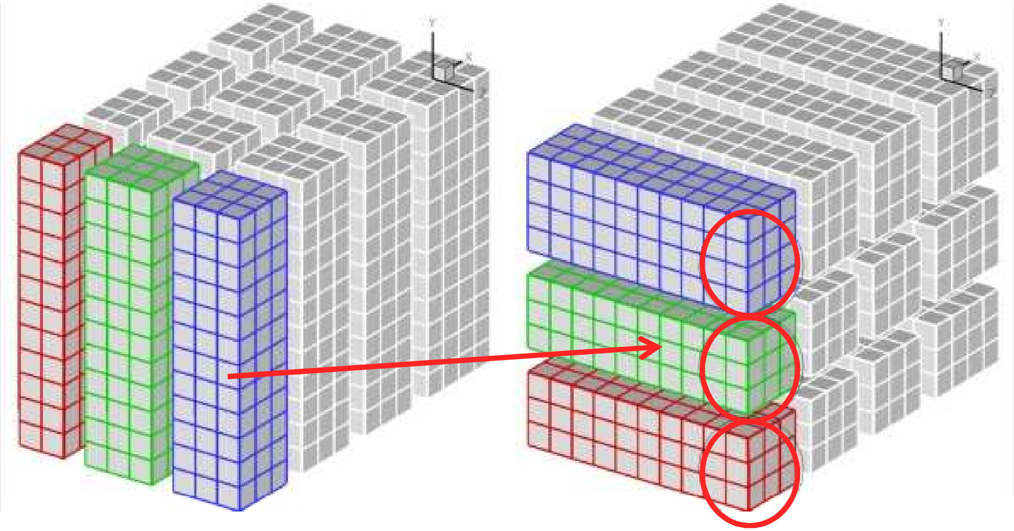
Why a Library Solution?

- 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

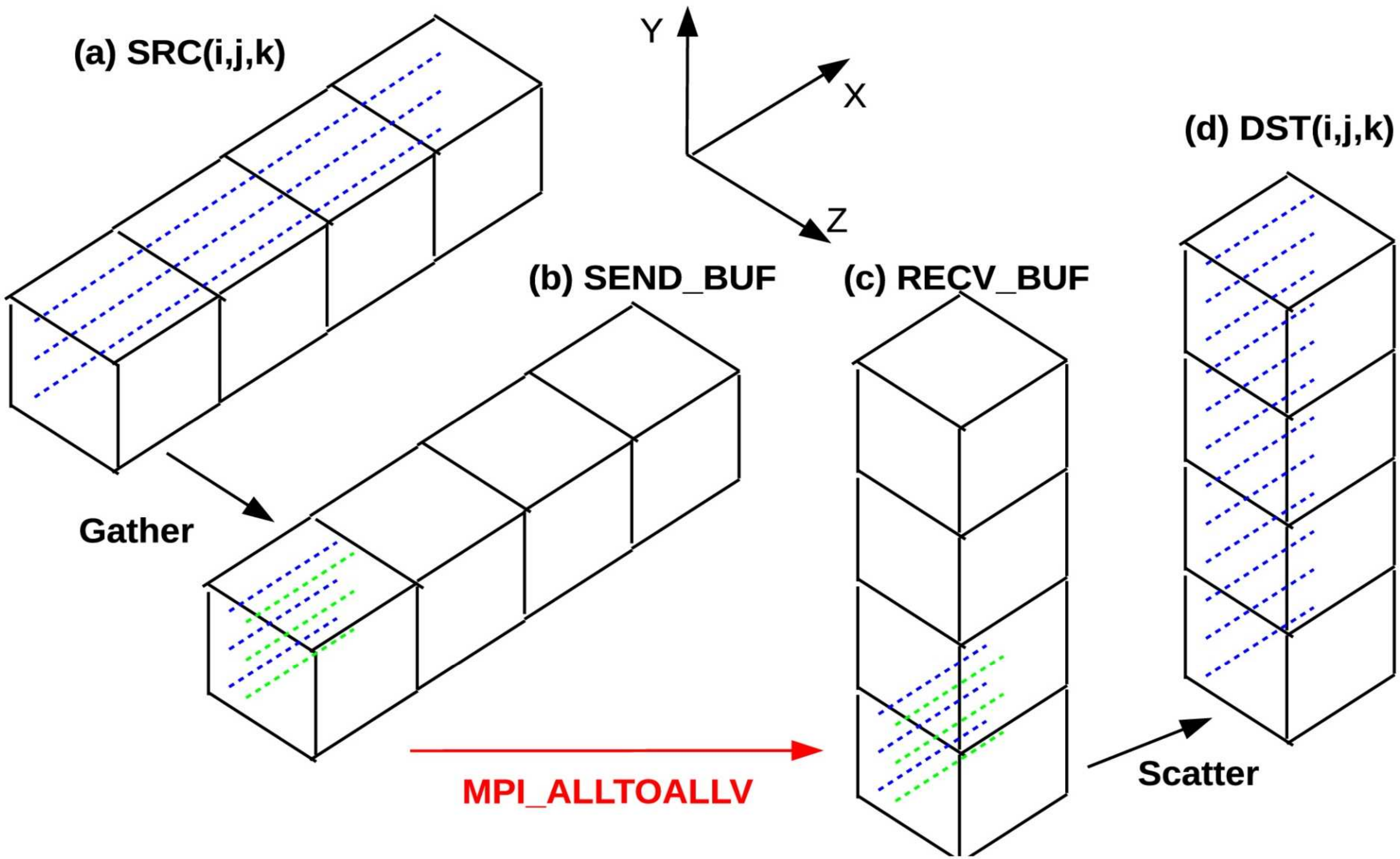
MPI_ALLTOALLV(sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, recvtype, comm)





- Best buffer gathering / scattering strategy?
- Optimisation opportunity?

Transpose from X-pencil to Y-pencil



Decomposition API

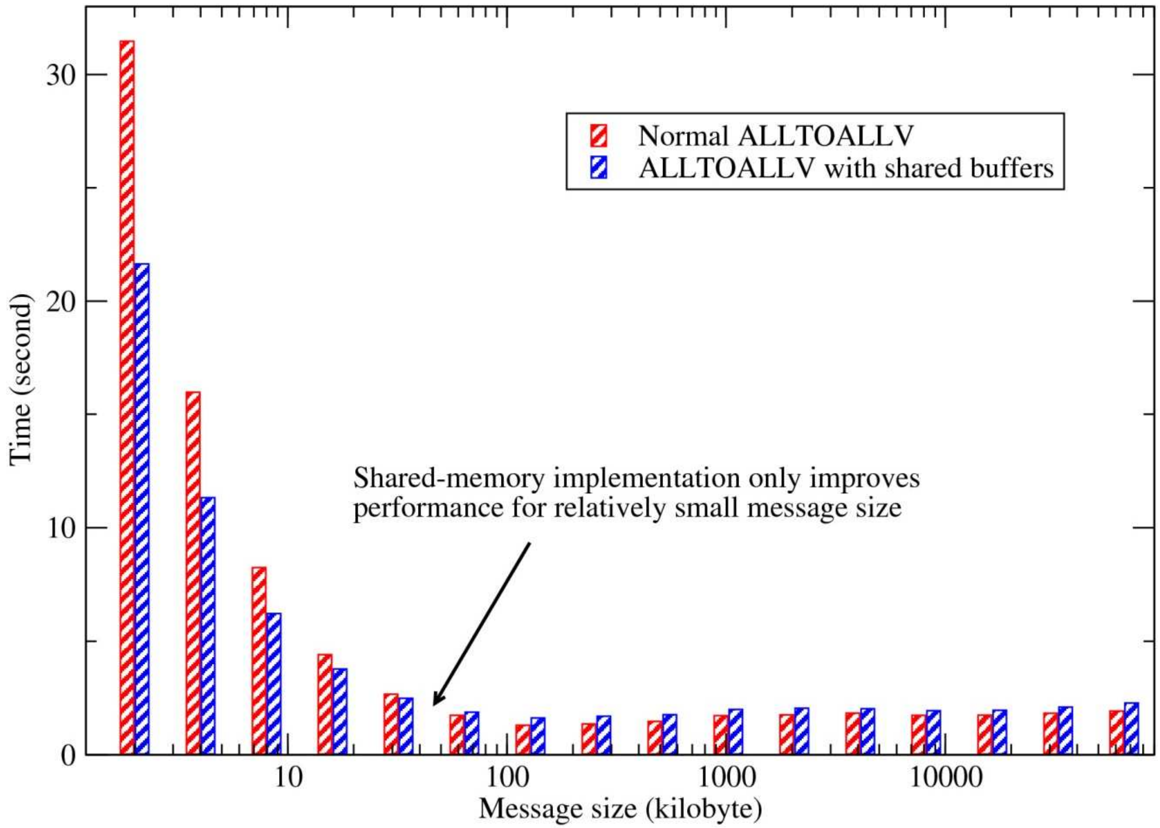
- Fortran module
  - **`use decomp_2d`**
- Global variables
  - Starting/ending index and size of the sub-domain held by current rank, required to define application data structures
    - **`allocate(in(xsize(1),xsize(2),xsize(3))`**
    - **`allocate(out(ystart(1):yend(1), ystart(2):yend(2), ystart(3):yend(3))`**
- 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. Tanqueray).
- 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

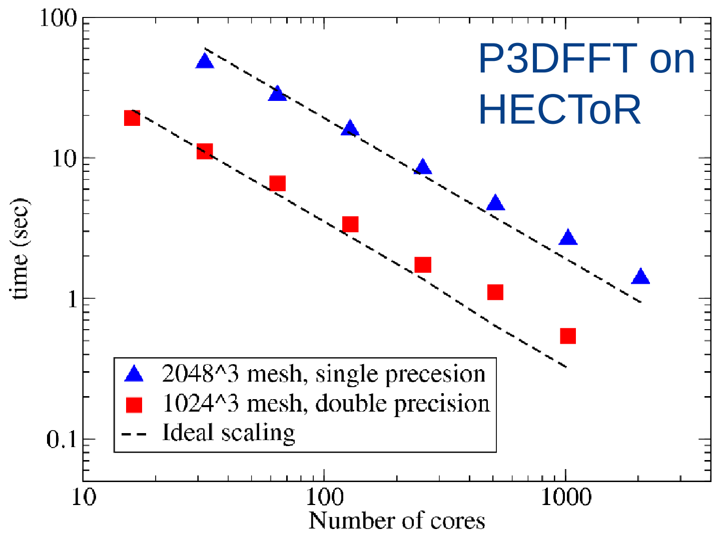
FFT Library	Comments
FFTW 2.x	MPI interface with 1D decomposition
FFTW 3.x	α-version MPI interface
CRAFFT (xt-libsci)	Evenly distributed 1D decomposition
Plimpton's parallel FFT# *	Complex-to-complex transforms only
Takahashi's FFTE #	Evenly distributed data; small prime factors
P3DFFT #	Real-to-complex/complex-to-real transforms only

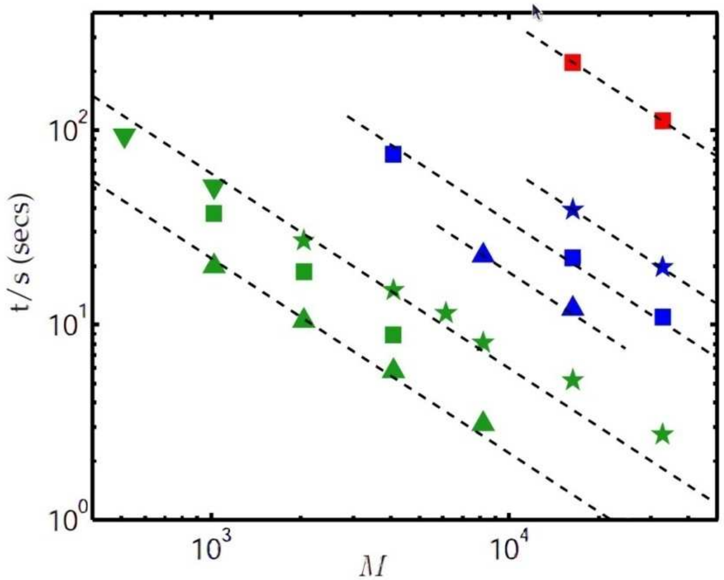
- # based on 2D decomposition
- * user-callable communication routines
- All with some limitations
- Having developed the underlying decomposition library, building a distributed FFT library on top is easy

nag

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**

	r1	r2	r3	r4	r5	r6	r7	r8
	c1	c2	c3	c4	c5	c6=$\underline{\text{c4}}$	c7=$\underline{\text{c3}}$	c8=$\underline{\text{c2}}$
(1)	a1	a2	a3	a4	a5	b4	b3	b2
(2)	c1	c2	c3	c4	c5	--	--	--

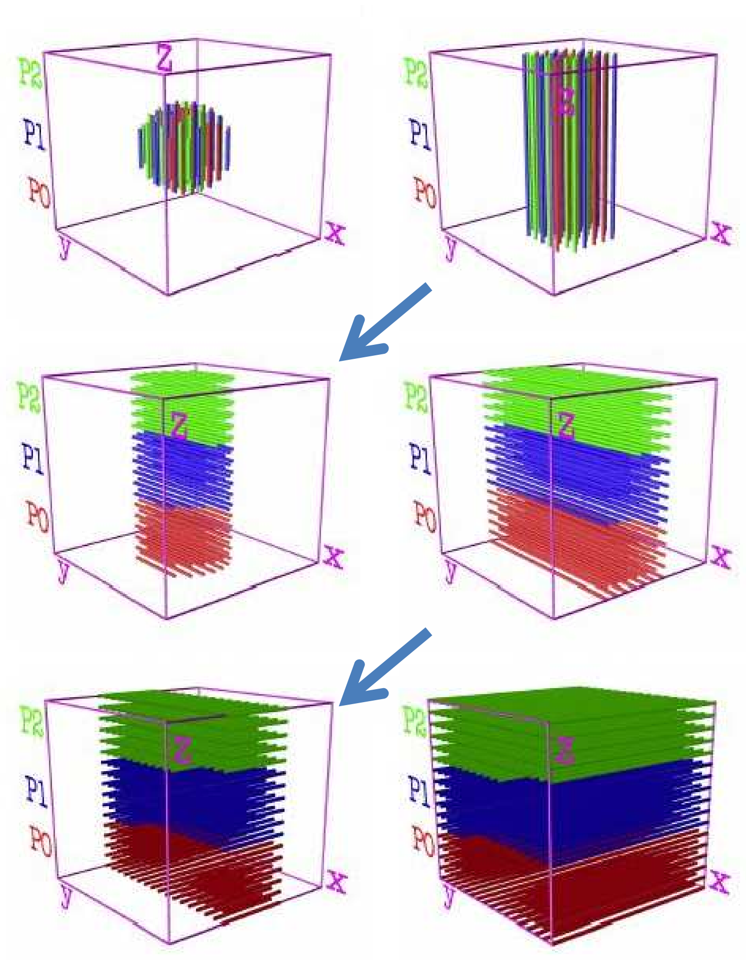
Extension of Base Communication Library

- Requirement
  - FFT real input: nx*ny*nz; complex output: (nx/2+1)*ny*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.
    - `TYPE(DECOMP_INFO) :: decomp`
    - `call decomp_info_init(nx,ny,nz,decomp)`
  - Optional third parameter to transposition routines
    - `call transpose_x_to_y(in,out,decomp)`

nag®

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.

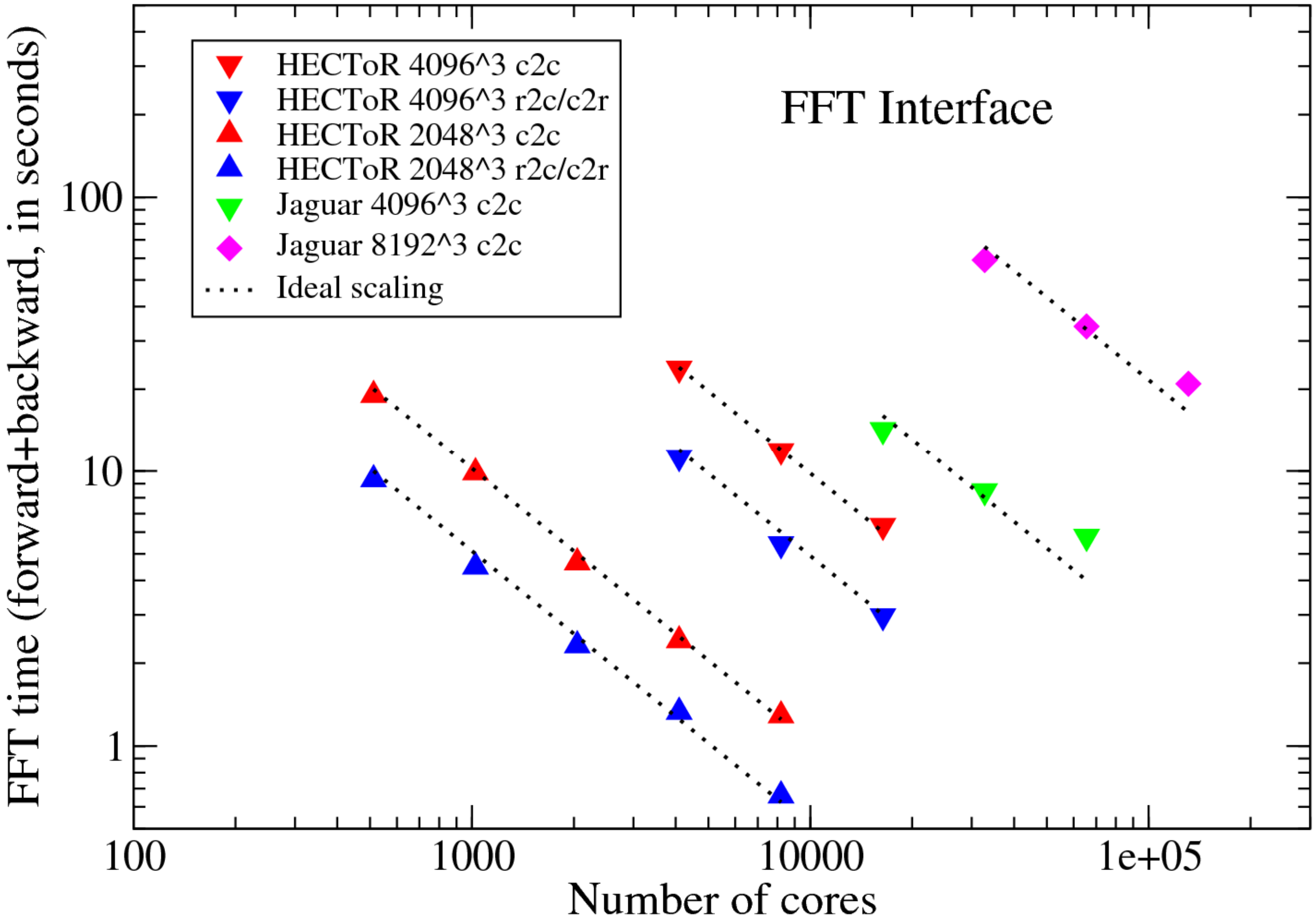
Library	Open-source	Hardware-tuned	Programming experience	Easy parallel coding
Generic (default)	Y	N	Slow but no dependency on external library	Y
FFTW 3.x	Y	Auto-tuning	Planning hard to use in parallel coding	N
ACML	N	For AMD	Limited r2c/c2r API	Y
fftpack	Y	N	Slow but used in many legacy applications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

FFT Library Performance

N^3	Serial FFTW		Distributed (FFTW engine)		
	Planning	Execution	16 cores	64 cores	256 cores
64^3	0.359	0.00509	0.00222	-	-
128^3	1.98	0.0525	0.0223	0.00576	0.00397
256^3	8.03	0.551	0.179	0.0505	0.0138
512^3	37.5	5.38	1.74	0.536	0.249
1024^3	-	-	-	4.59	1.27
2048^3	-	-	-	-	17.9

- 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

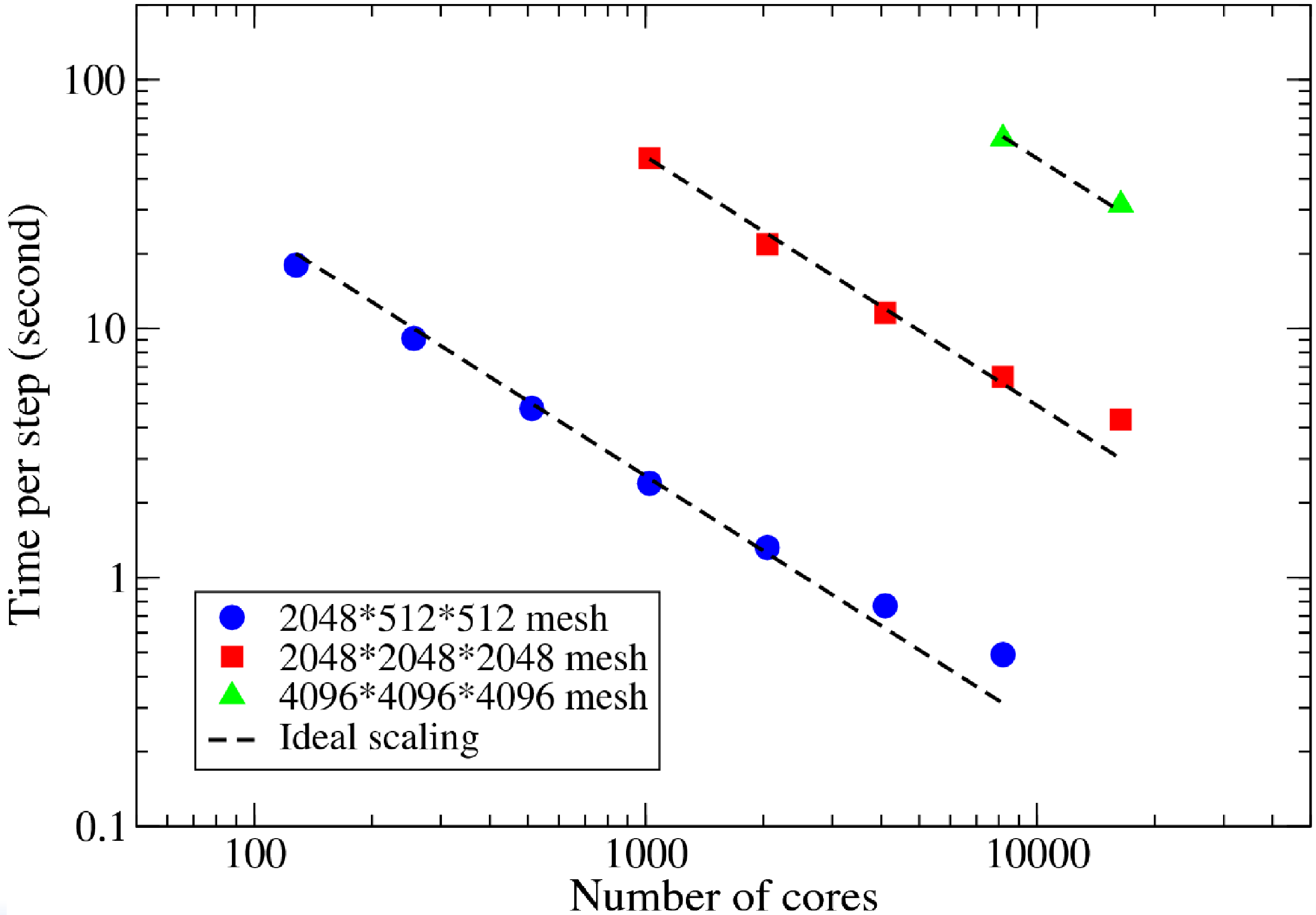
Boundary Condition	Global Transpositions	1024^3 case on 128 cores	2048^3 case on 1024 cores	4096^3 case on 8192 cores
000	FFT only	4.81	6.26	7.59
100	FFT + 8	7.38	10.38	14.41
010	FFT + 6	6.81	8.86	12.63
110	FFT + 12	8.23	11.56	16.31
111		8.41	11.67	16.48

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

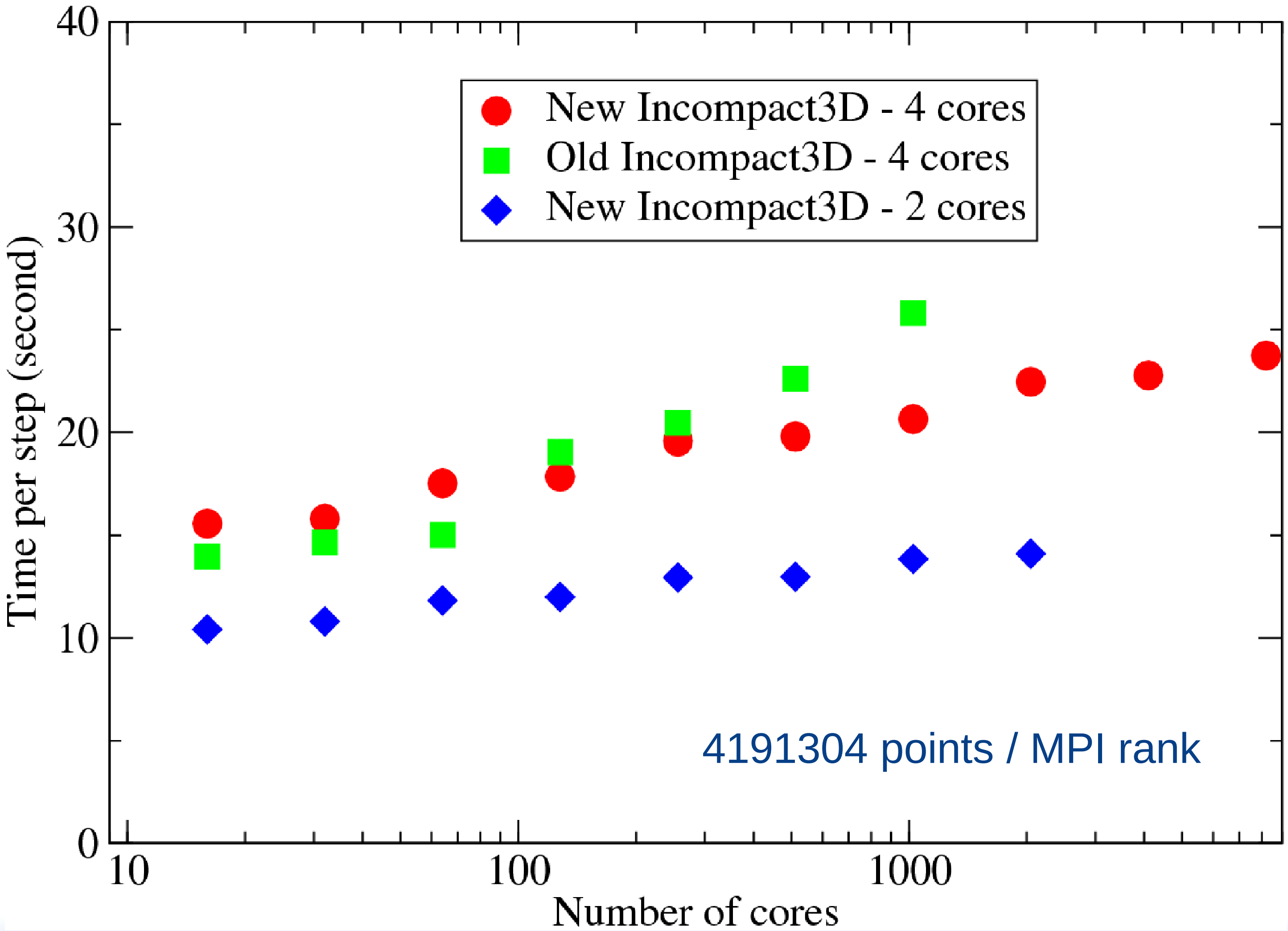
Putting all together

- 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



Incompact3D Weak Scaling on HECToR



nag

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?

HECToR

HIGH END COMPUTING TERASCALE RESOURCE