# Scalable DNS code for high Reynolds number channel flow simulation on BG/Q

MyoungKyu Lee
mk@ices.utexas.edu

Department of Mechanical Engineering
University of Texas at Austin

MiraCon Mar, 2013

# Contents

Project Overview

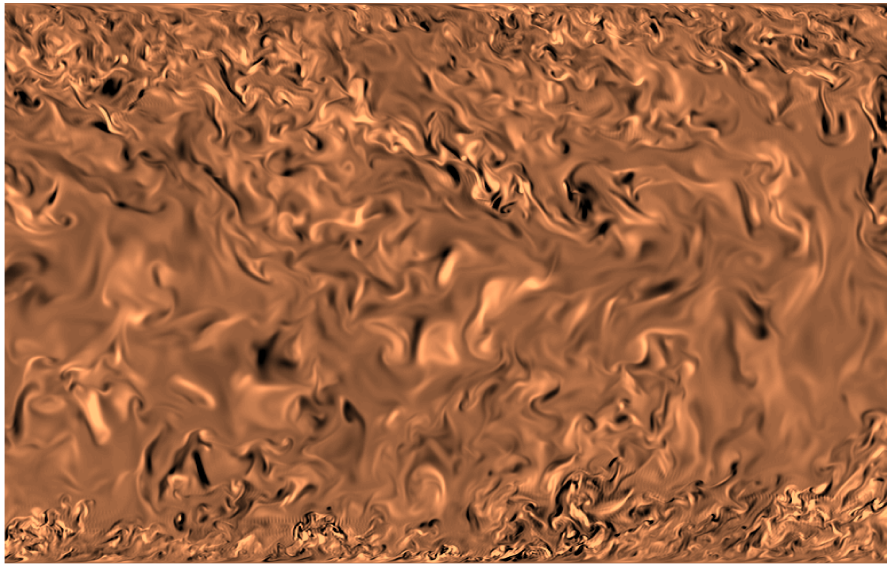
Performance Optimization

Early Result

Conclusion

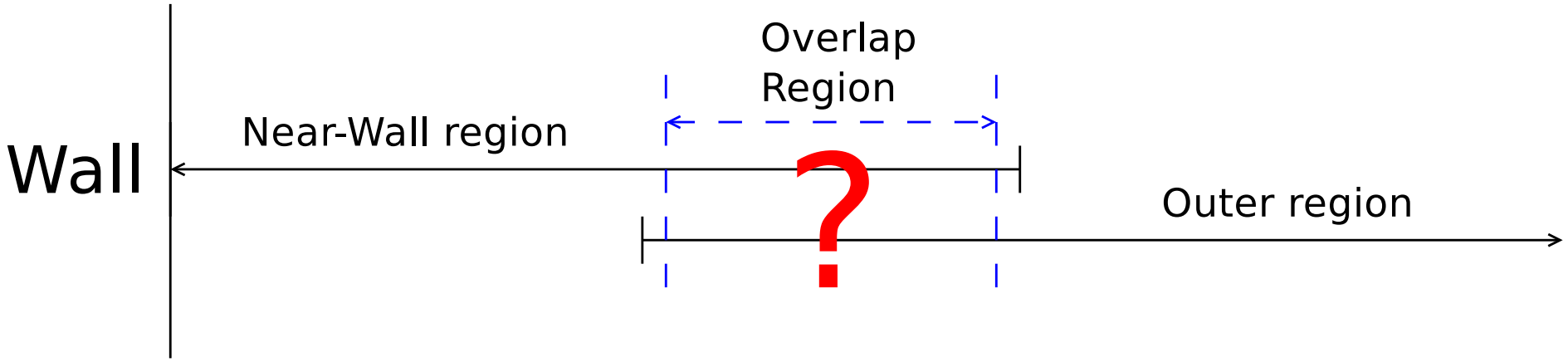
# Project Overview

- Project Title
  - Petascale Direct Numerical Simulations of Turbulent Channel Flow



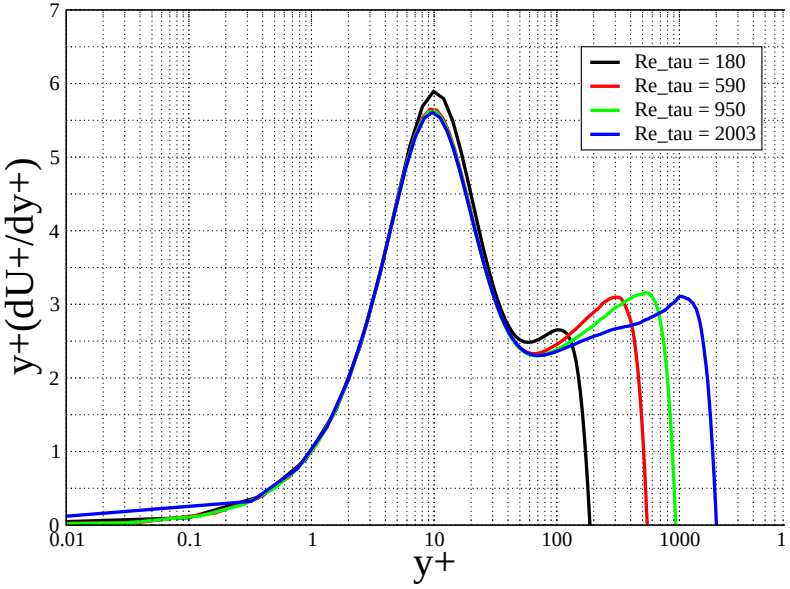
- Goal
  - Expand our understand of wall-bounded turbulence
- Personnel
  - P.I. : Robert Moser
  - Primary Developer : M.K.Lee
  - Software Engineering Support : Nicholas Malaya
  - Catalyst : Ramesh Balakrishnan

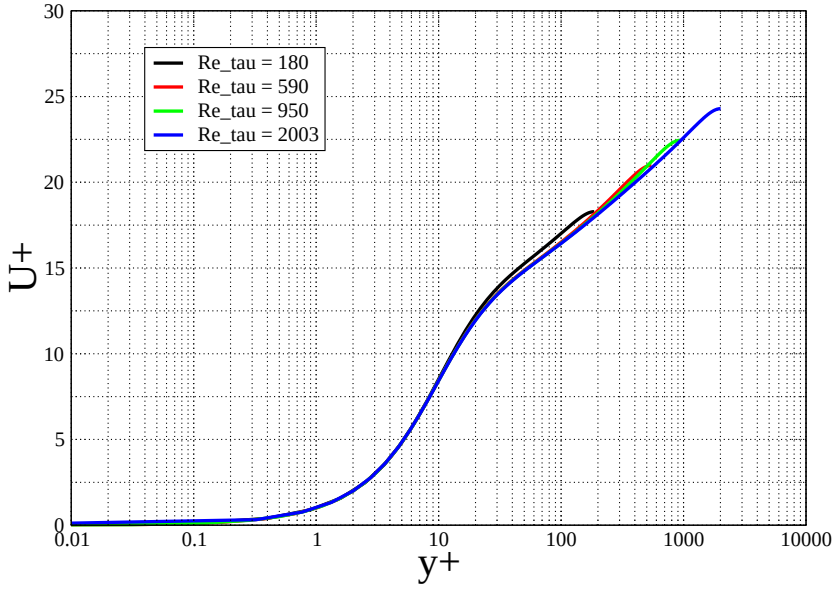
# Overlap Region




- Connection between near-wall region and outer region
- Not understood as well as near-wall region

# History



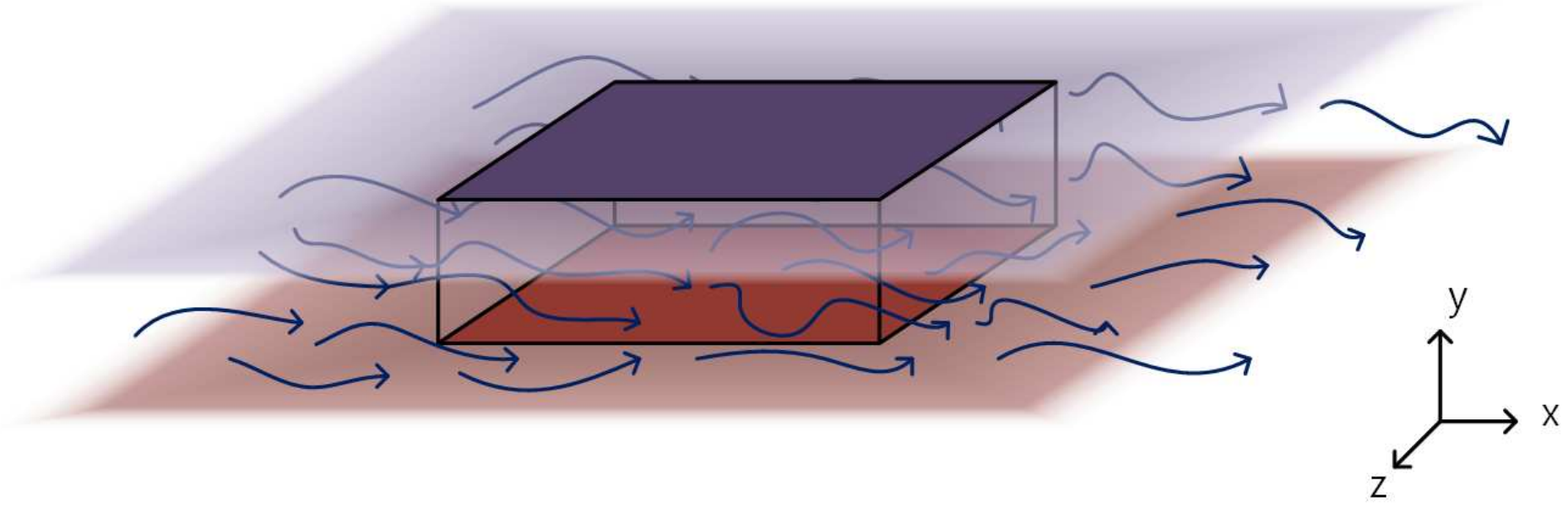

- $Re_\tau = 180$ : Kim, Moin & Moser, 1987
- $Re_\tau = 590$ : Moser, Kim & Mansour, 1999
- $Re_\tau = 940$ : Del Álamo, Jiménez, Zandonade & Moser, 2004
- $Re_\tau = 2003$ : Hoya & Jiménez, 2006

# Computation of Overlap Region

- High Reynolds(Re) number is required to obtain logarithmic layer.
- Even with existing channel flow DNS data at the high Re, $Re_\tau \approx 2000$, the region of log layer is small.
- To expand our understanding of log layer, DNS data at $Re_\tau > 4000 \sim 5000$ is required.
- Cost scales with $Re^3$
- Simulation at high $Re$ is limited by computational power.

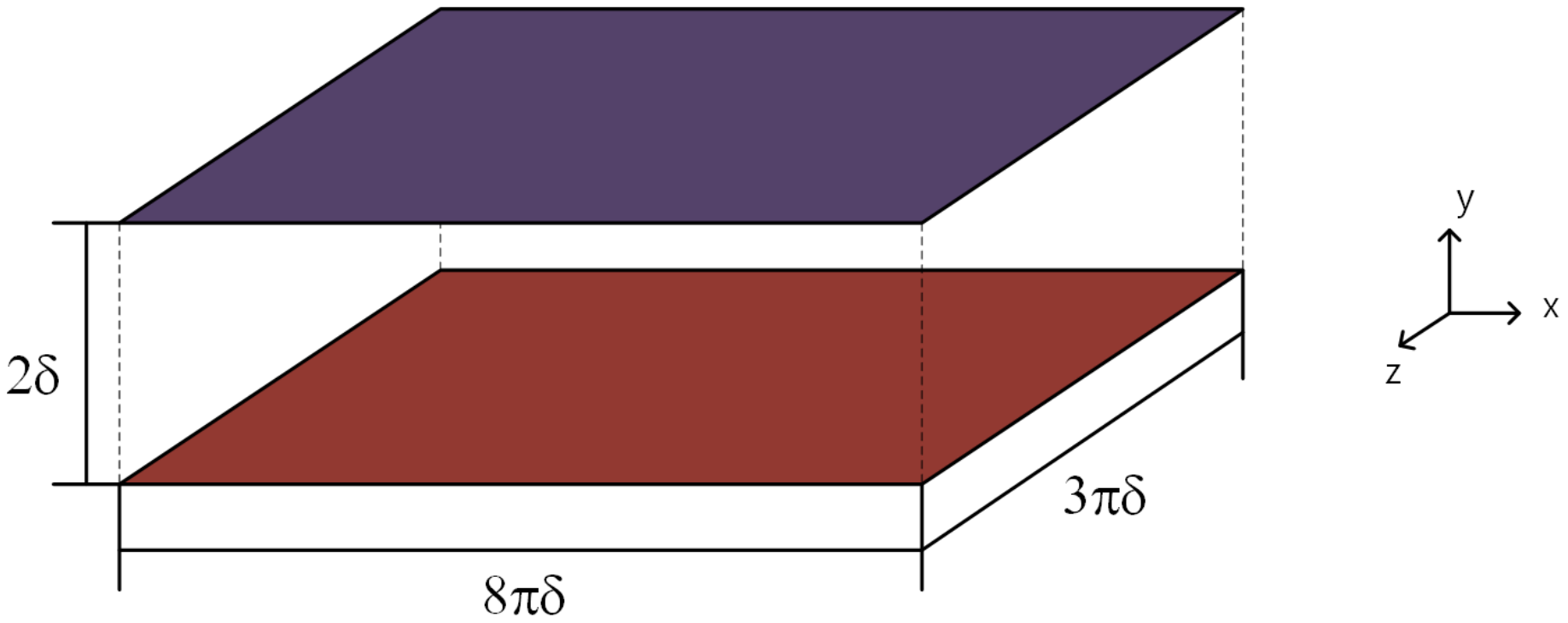
- **BG/Q enable us to simulate such a high $Re$ flow**

# Simulation Geometry




- Flow between two infinite parallel plates
- $\mathbf{Re}_\tau \approx \mathbf{5000}$
- Periodic boundary conditions in streamwise / spanwise directions
- No slip, no mass flux boundary conditions at wall

# Simulation Geometry




- Nx = 15,360 (Uniform) - DFT
- Ny = 1,536 (Stretched, wall-denser) - B-Spline
- Nz = 11,520 (Uniform) - DFT
- 270 billion points $\times$ 5 fields

# Formulation (Kim, Moin & Moser, 1987)

$$\frac{\partial u_i}{\partial t} = -\frac{\partial P}{\partial x_i} + H_i + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j}, \qquad \frac{\partial u_j}{\partial x_j} = 0$$

$$\frac{\partial \omega_y}{\partial t} = h_g + \nu \nabla^2 \omega_y$$
$$\frac{\partial \phi}{\partial t} = h_v + \nu \nabla^2 \phi$$

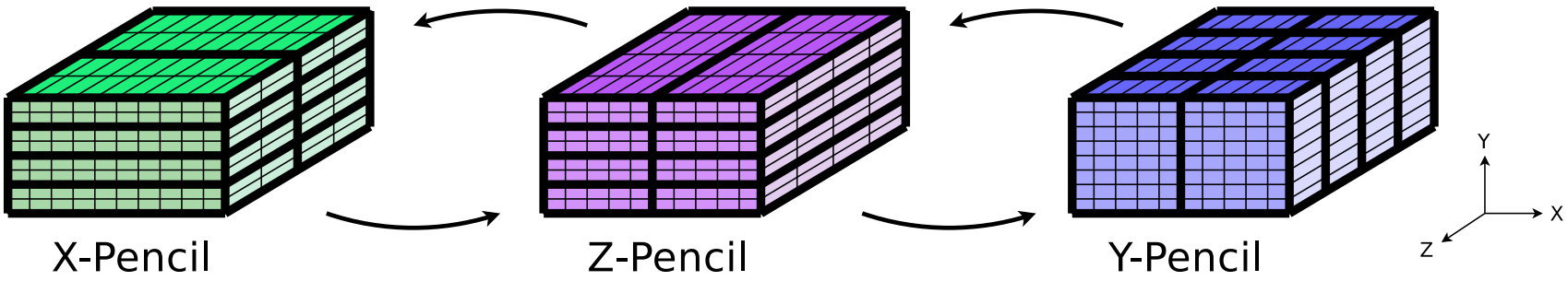
where

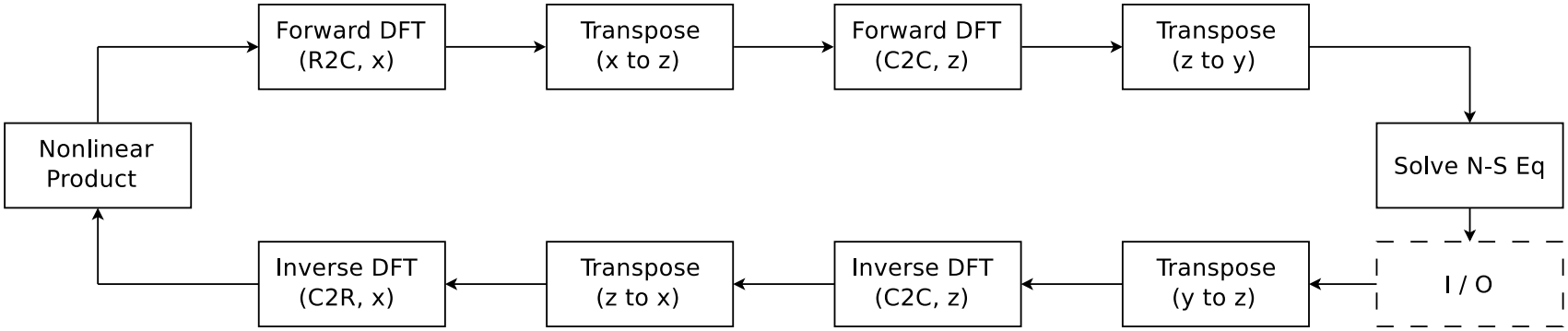
$$\phi = \nabla^2 v, \qquad \text{At wall} \quad \omega_y = 0, \quad v = \frac{\partial v}{\partial y} = 0$$

$$h_g = \frac{\partial H_1}{\partial z} - \frac{\partial H_3}{\partial x}, \quad h_v = \frac{\partial^2 H_2}{\partial x^2} + \frac{\partial^2 H_2}{\partial z^2} - \frac{\partial^2 H_1}{\partial x \partial y} - \frac{\partial^2 H_3}{\partial y \partial z}$$

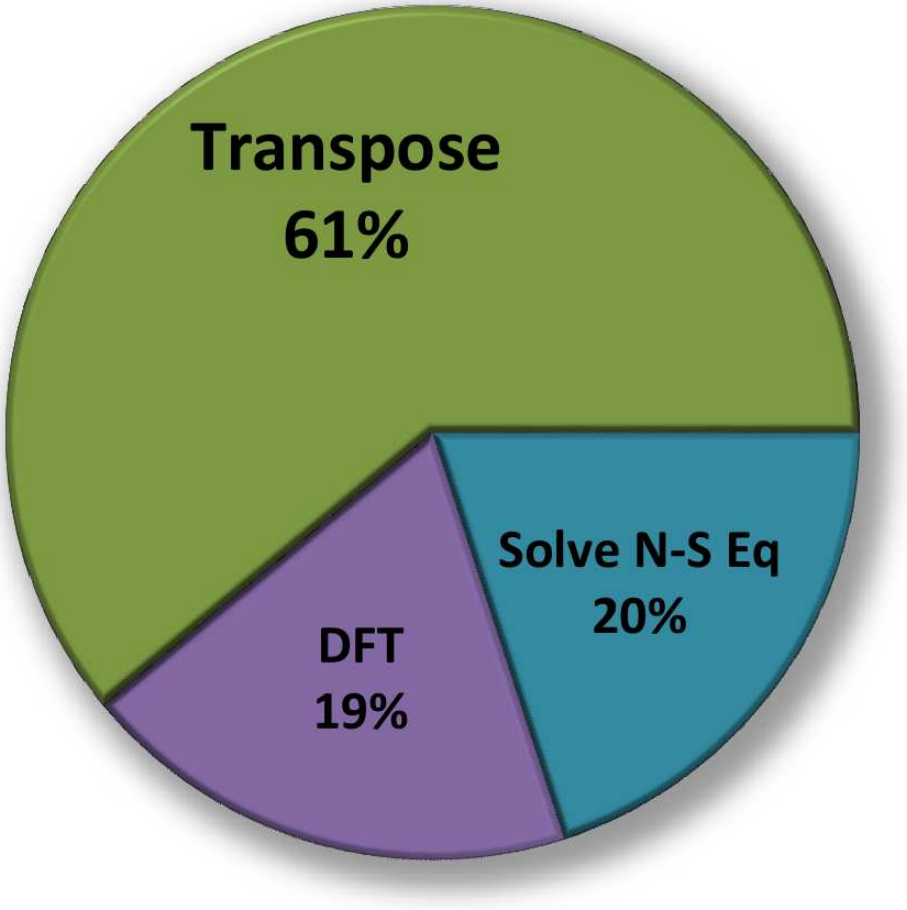
- Time discretization : Implicit/Explicit Low-Storage RK3 method

# Simulation Process - Pencil Decomposition



# For one time step




* 8 Racks, 1 MPI task/node

# Data Transpose

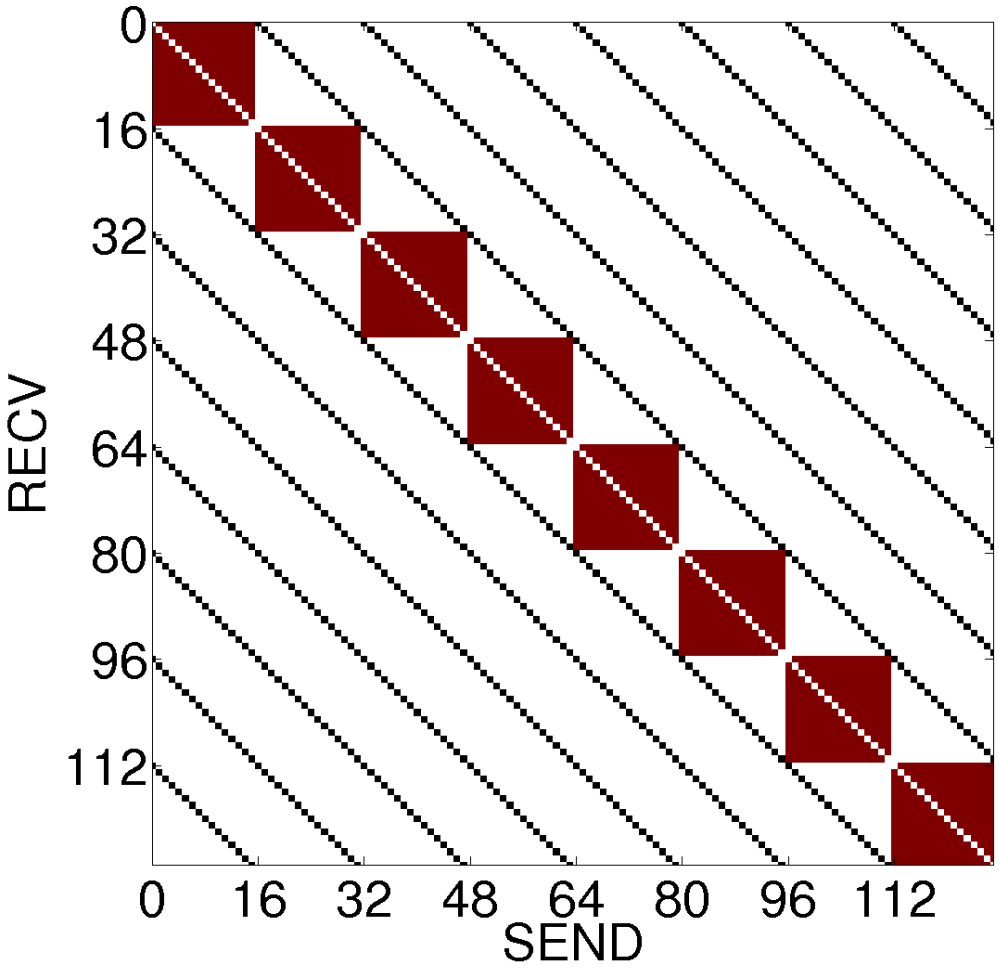
- Sequential data alignment is necessary for DFT and solving linear equations
- 3D version of All-to-All communication
- *FFTW 3.3 library*

- MPI Task per node : $1 \longleftrightarrow 64$

# Sub-Communicator

```
ndims = 2 ! Decomposed by 2 direction (a.k.a. pencil decomposition)
periods(1) = .false. ! no periodicity
periods(2) = .false. ! no periodicity

reorder = .false.
call mpi_cart_create(MPI_COMM_WORLD,ndims,dimsize &
,periods,reorder,COMM2D,ierr)
call mpi_comm_rank(COMM2D,rank2D,ierr)

! Create Communicator in 1st direction
belongs(1) = .true.
belongs(2) = .false.
call mpi_cart_sub(COMM2D,belongs,COMM1DA,ierr)

! Create Communicator in 2nd direction
belongs(1) = .false.
belongs(2) = .true.
call mpi_cart_sub(COMM2D,belongs,COMM1DB,ierr)

! Number of tasks in each direction
nprocs1DA = dimsize(1)
nprocs1DB = dimsize(2)
call mpi_comm_rank(COMM1DA,myrankA,ierr)
call mpi_comm_rank(COMM1DB,myrankB,ierr)
```

# Communication Pattern



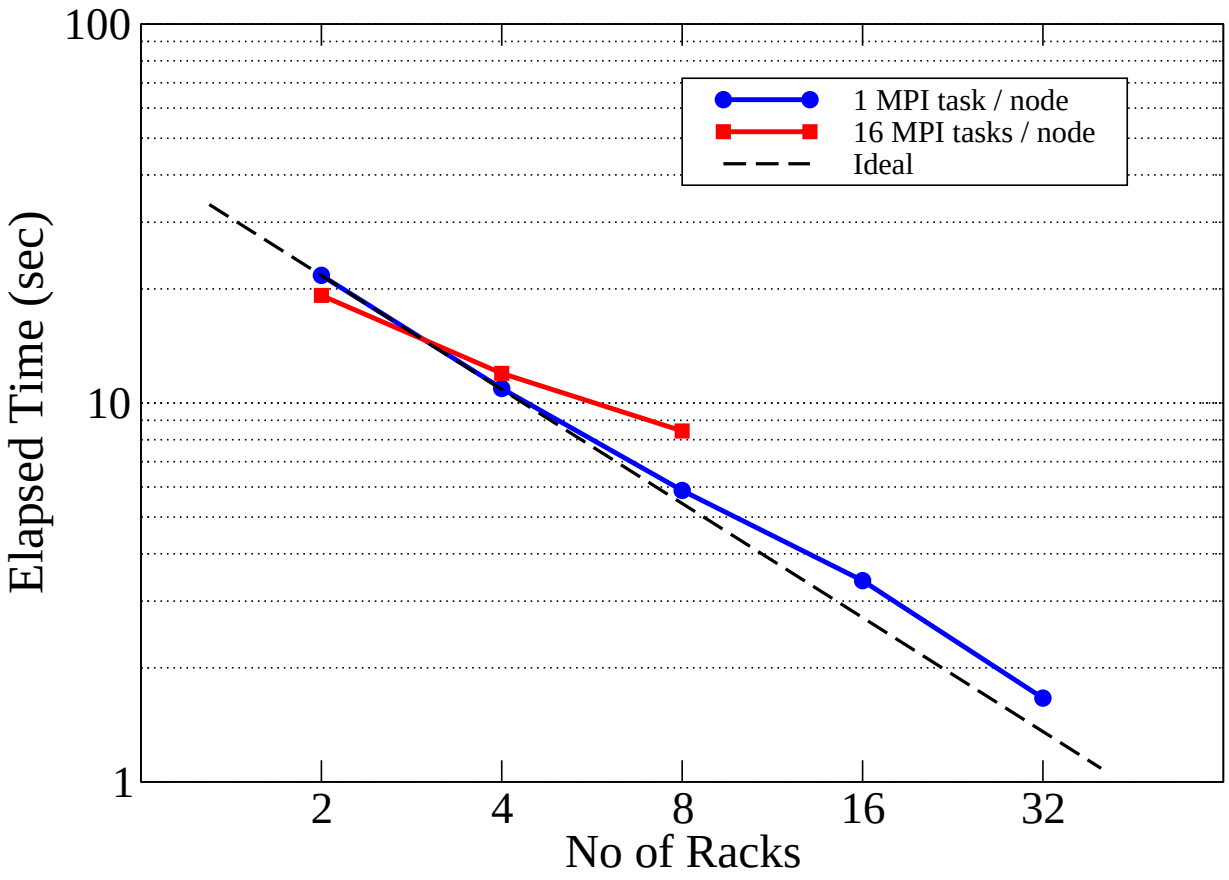

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 16 | 32 | 48 | 64 | 80 | 96 | 112 |
| 1 | 17 | 33 | 49 | 65 | 81 | 97 | 113 |
| 2 | 18 | 34 | 50 | 66 | 82 | 98 | 114 |
| 3 | 19 | 35 | 51 | 67 | 83 | 99 | 115 |
| 4 | 20 | 36 | 52 | 68 | 84 | 100 | 116 |
| 5 | 21 | 37 | 53 | 69 | 85 | 101 | 117 |
| 6 | 22 | 38 | 54 | 70 | 86 | 102 | 118 |
| 7 | 23 | 39 | 55 | 71 | 87 | 103 | 119 |
| 8 | 24 | 40 | 56 | 72 | 88 | 104 | 120 |
| 9 | 25 | 41 | 57 | 73 | 89 | 105 | 121 |
| 10 | 26 | 42 | 58 | 74 | 90 | 106 | 122 |
| 11 | 27 | 43 | 59 | 75 | 91 | 107 | 123 |
| 12 | 28 | 44 | 60 | 76 | 92 | 108 | 124 |
| 13 | 29 | 45 | 61 | 77 | 93 | 109 | 125 |
| 14 | 30 | 46 | 62 | 78 | 94 | 110 | 126 |
| 15 | 31 | 47 | 63 | 79 | 95 | 111 | 127 |

Sub Comm. A

Sub Comm. B

- e.g. 128 MPI tasks = $16 \times 8$
- Sub Comm. A $\rightarrow$ Black / Sub Comm. B $\rightarrow$ Red

# Communication



# Communication - Performance Tips

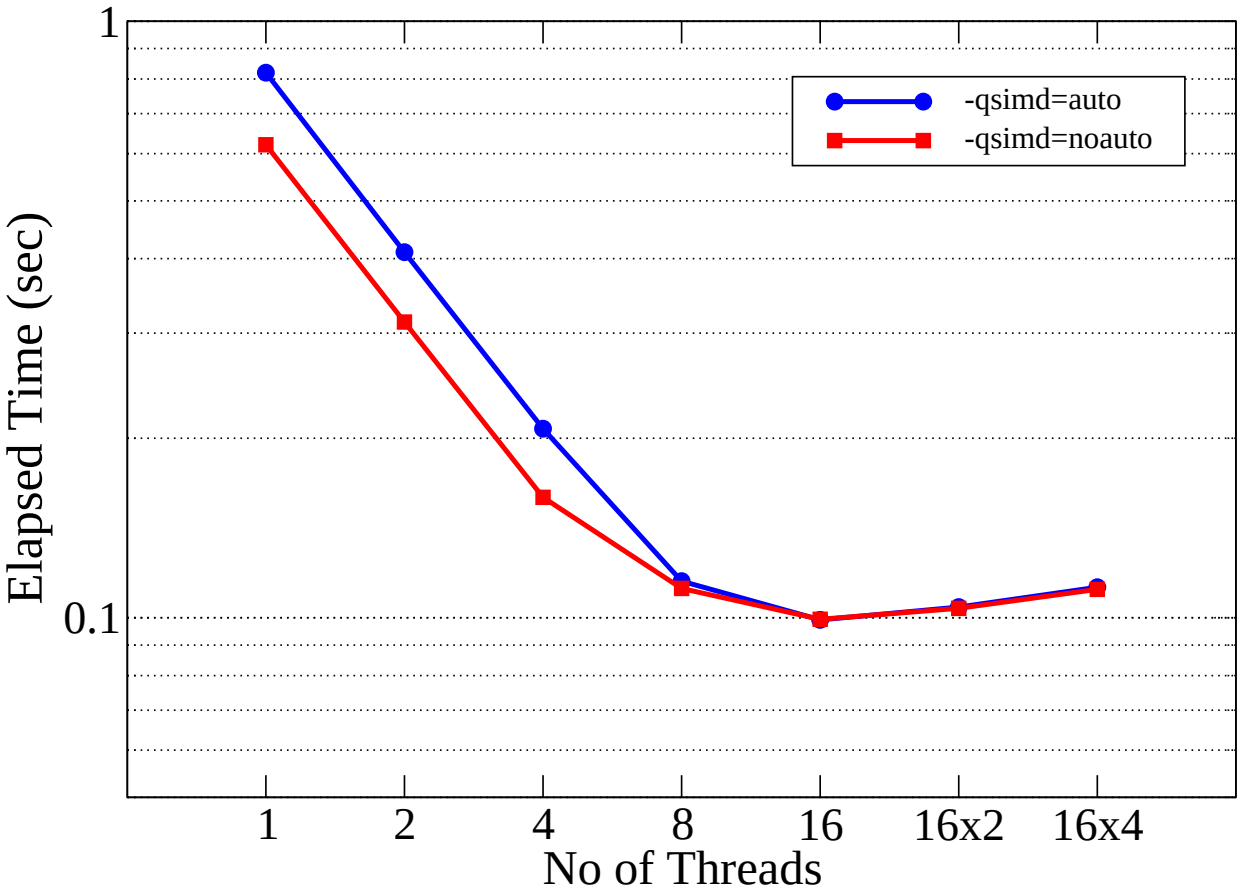
- SubComm. - **Do Not Mess With Torus Boundaries**
  - e.g. 512 Node : <4,4,4,4,2,a>
  - a: number of MPI tasks per node
  - Good : $64\times$ 8a = $(4\times4\times4) \times (4\times2\times a)$
  - Bad : $32\times$ 16a = $(4\times4\times2) \times (2\times4\times2\times a)$
- Generally, best performance with one subcommunicator for a midplane (512 Nodes)
- When changing mapping order (e.g. ABCDET to TECDBA), be sure **reorder is off**.
- For more information: (REDBOOK) *IBM System Blue Gene Solution: Blue Gene/Q Application Development*
  - Appendix A Mapping

# Data Shuffling

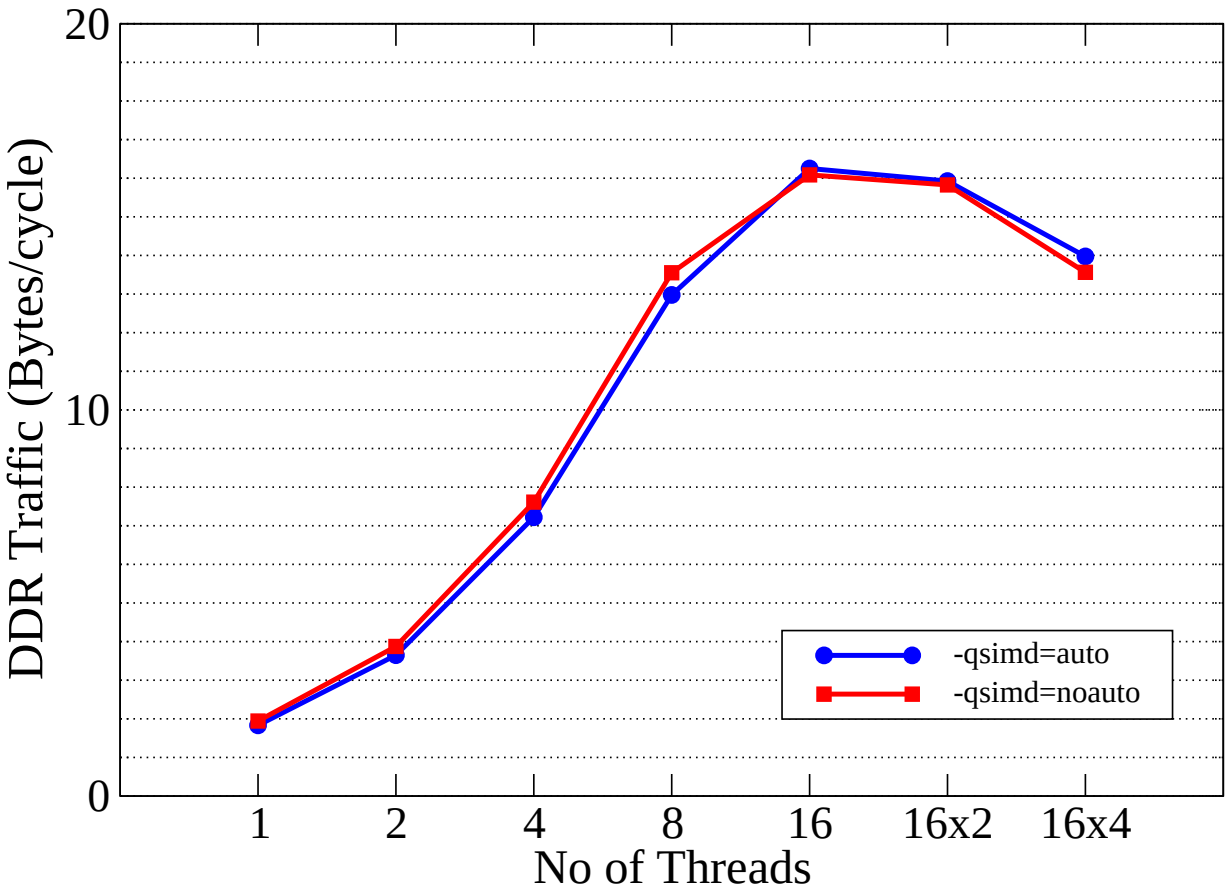
- Necessary procedure for the data alignment after communication

```
! Nice Memory Access
do k = 1,nc
  do j = 1,nb
    do i = 1,na
      B(:,j,k,i) = A(:,i,j,k)
    enddo
  enddo
enddo

! Ugly Memory Access (Unavoidable)
do j = 1,nb
  do k = 1,nc
    do i = 1,na
      do l = 1,nd
        B(i,j,k,l) = A(l,i,k,j)
      enddo
    enddo
  enddo
enddo
```

# Data Shuffling



# Data Shuffling

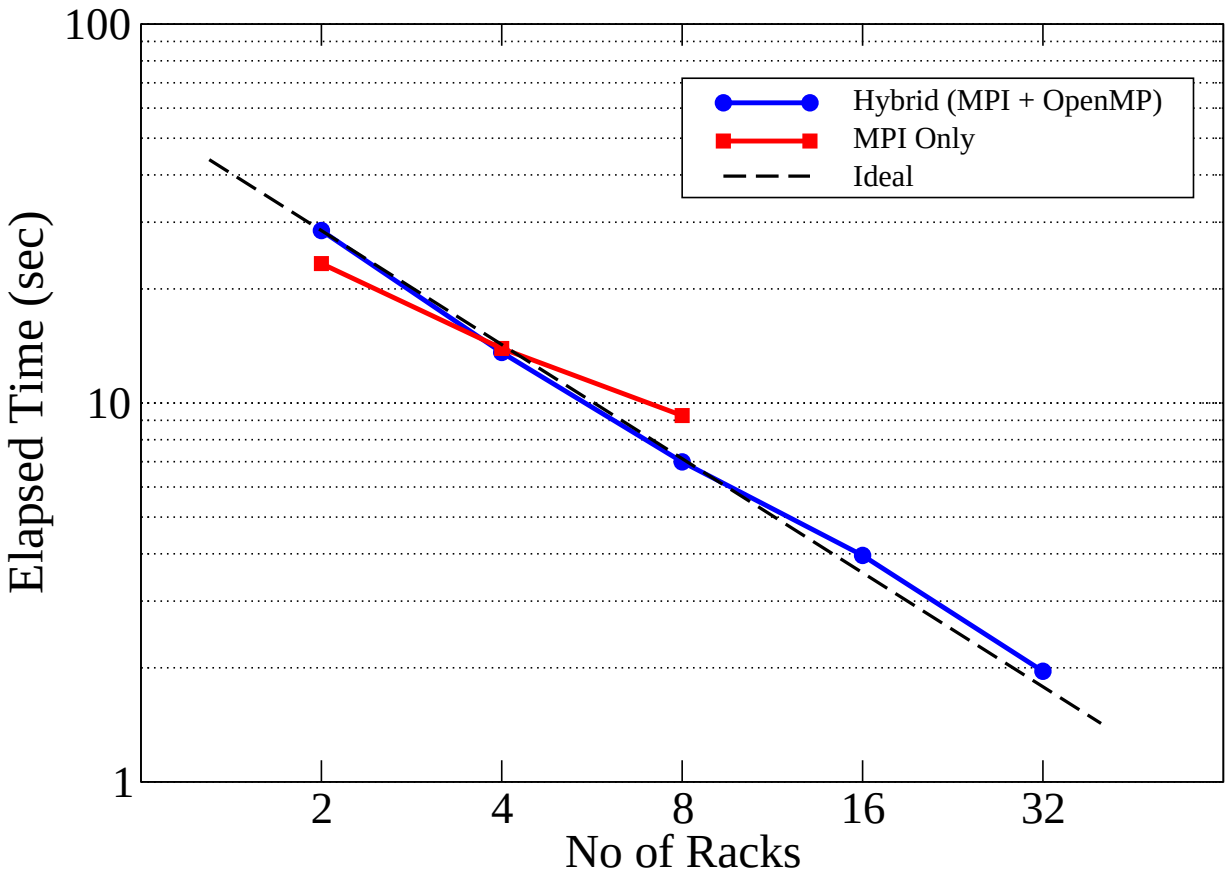



* Maximum DDR Traffic $\approx$ 18 Byte/cycle

# Data Shuffling - Performance Tips

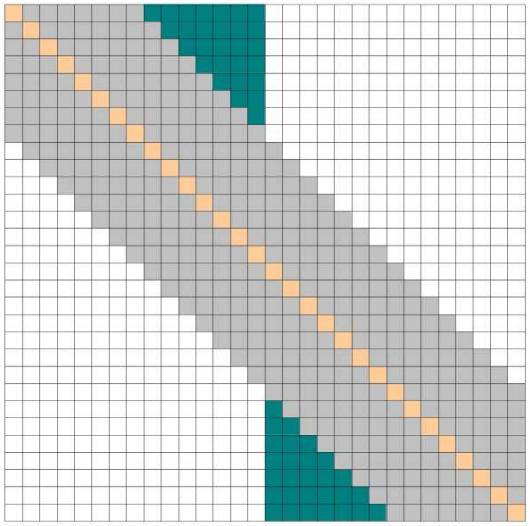
- Try to make **sequential memory access** at stride 1
  - Most of data load hit in L1D Cache or L1P prefetch ($\sim$99%)
- Make sequential memory access for data load rather than data write when you have to choose one of them
- Using hardware threads does not guarantee better performance when DDR traffic is saturated
- Compiling with SIMD flags does not guarantee better performance

# Transpose = Data Shuffling + Communication

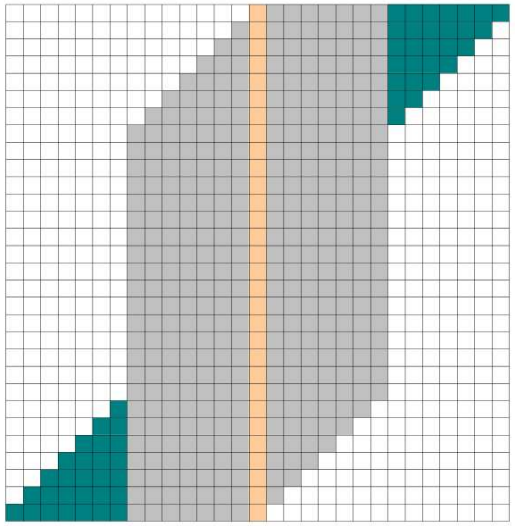


# Solving Linear Equations : $A\mathbf{x} = \mathbf{b}$

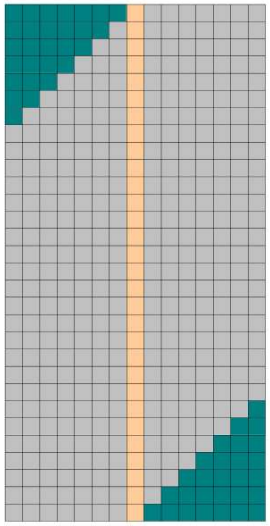
- In Y-direction, B-Spline with Non-periodic boundary condition
- A matrix
  - Banded matrix with additional components near the wall



Original Full Matrix



Banded Matrix Form



Compact Matrix Form

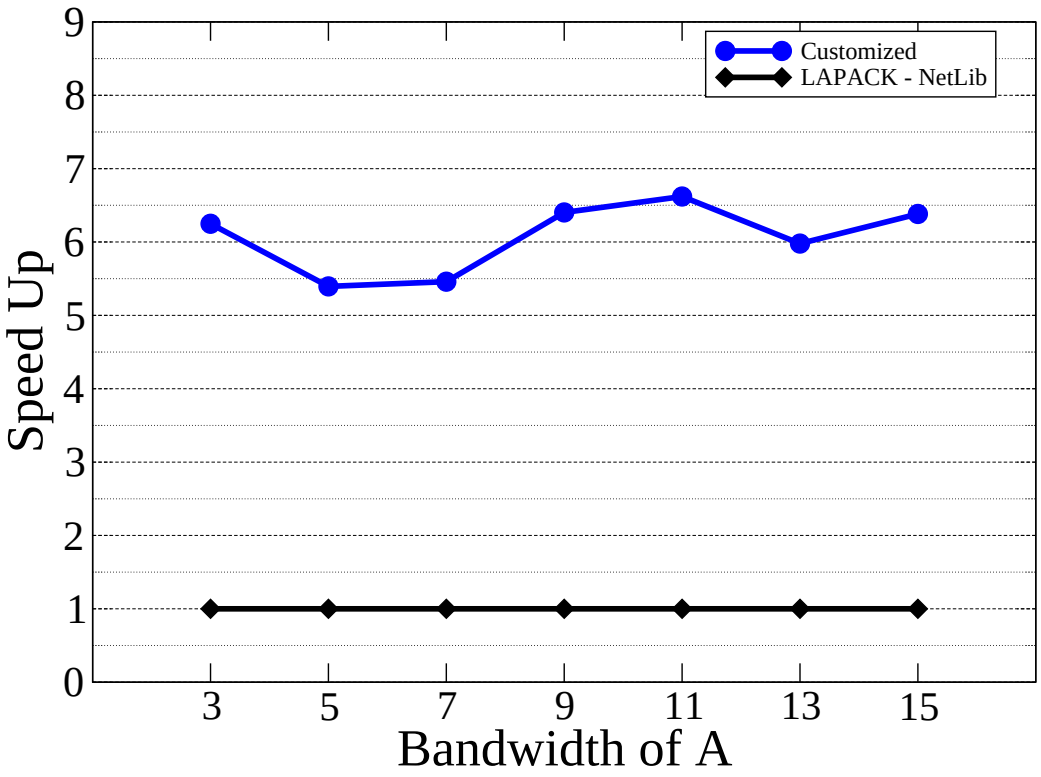
- Conventional LAPACK library - ZGBTRF / ZGBTRS
  - ESSL does not support it
  - MKL-intel, Netlib
    - ★ When **x** and **b** are complex vector, A has to be a complex matrix
    - ★ Need more memory allocation / operation on unnecessary portions

# Solving Linear Equations : $A\mathbf{x} = \mathbf{b}$

- Hand Loop-Unrolling : Maximize **cache memory efficiency**

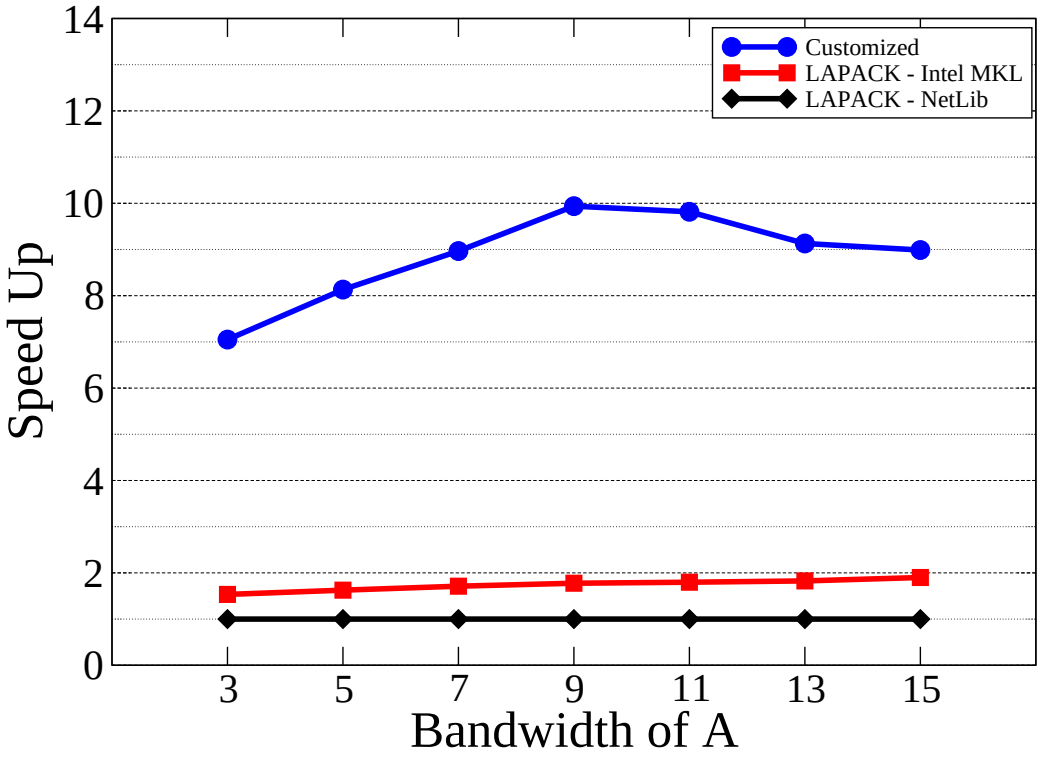
```
...
dummy = 1.d0 / A(m,2)
A(m+1,2) = dummy * (A(m+1,2) -  A(m-1,2) * A(m+2,1))
A(m+2,2) = dummy * (A(m+2,2) -  A(m-1,2) * A(m-1,1))
A(m-2,2) = dummy * (A(m-2,2) -  A(m-1,2) * A(m-2,1))
...
dummy = 1.d0 / A(m,n-2*bw+2)
A(m+1,n-2*bw+2) = dummy * (A(m+1,n-2*bw+2) &
 - A(m-1,n-2*bw+2) * A(m+2,n-2*bw+1))
A(m+2,n-2*bw+2) = dummy *  A(m+2,n-2*bw+2)
...
A(m-2,n-2*bw+9)   = A(m-2,n-2*bw+9)  &
- A(m-3,n-2*bw+9)  * A(m+1,n-2*bw+6) &
- A(m-4,n-2*bw+9)  * A(m+2,n-2*bw+5) &
- A(m-5,n-2*bw+9)  * A(m+3,n-2*bw+4) &
- A(m-6,n-2*bw+9)  * A(m+4,n-2*bw+3) &
- A(m-7,n-2*bw+9)  * A(m+5,n-2*bw+2) &
- A(m+6,n-2*bw+9)  * A(m+6,n-2*bw+1) &
- A(m+7,n-2*bw+9) * A(m+7,n-2*bw}
...
```

# Solving Linear Equations : $A\mathbf{x} = \mathbf{b}$, $Ny = 1536$




- Normalized by elapsed time using NetLib Lapack

# Solving Linear Equations : $A\mathbf{x} = \mathbf{b}$, $Ny = 1536$

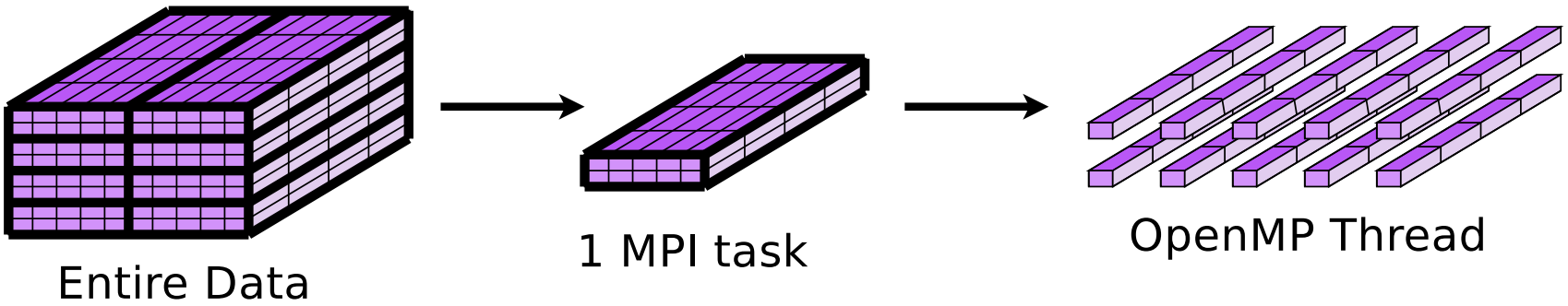



- Normalized by elapsed time using NetLib Lapack
- Intel Xeon 5680, Lonestar, *Texas Advanced Computing Center*

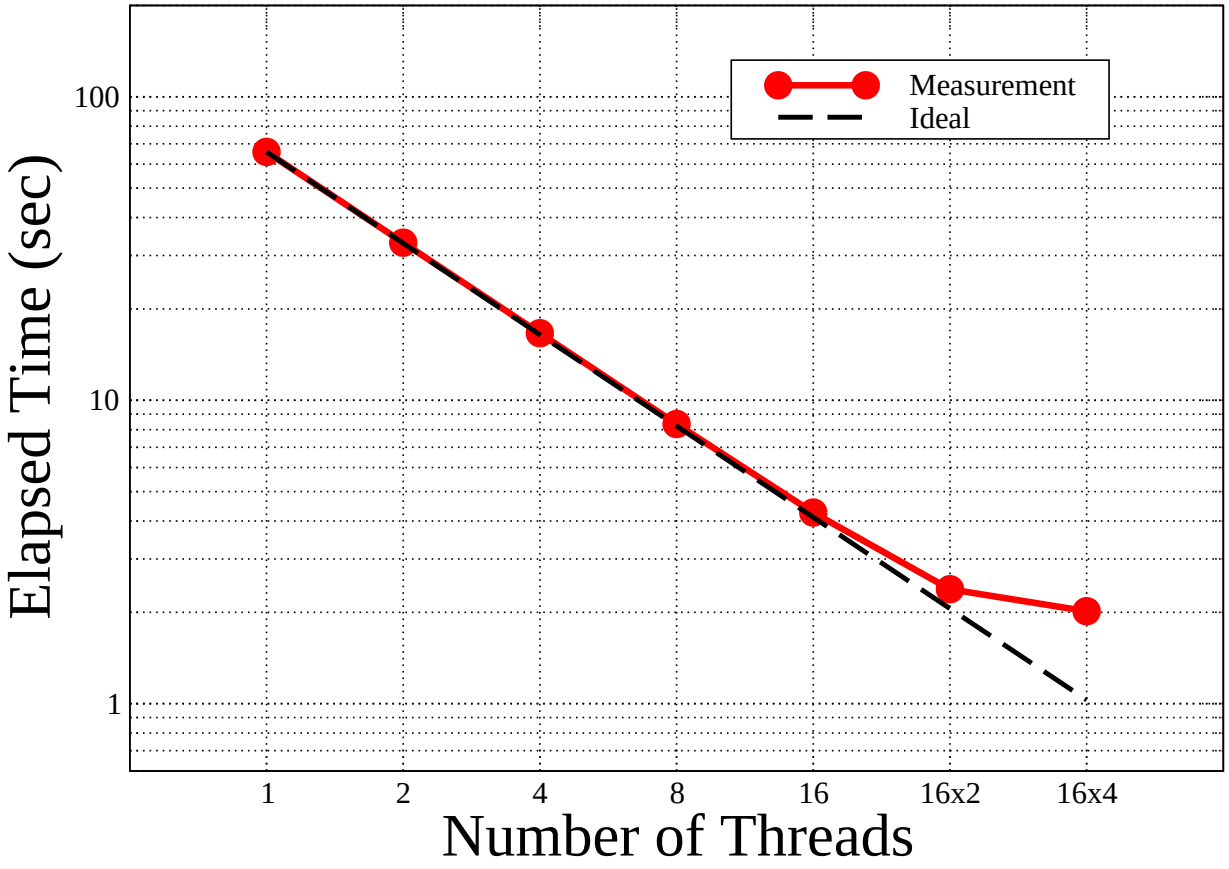
# Threading

After data transpose (with nice data alignment),

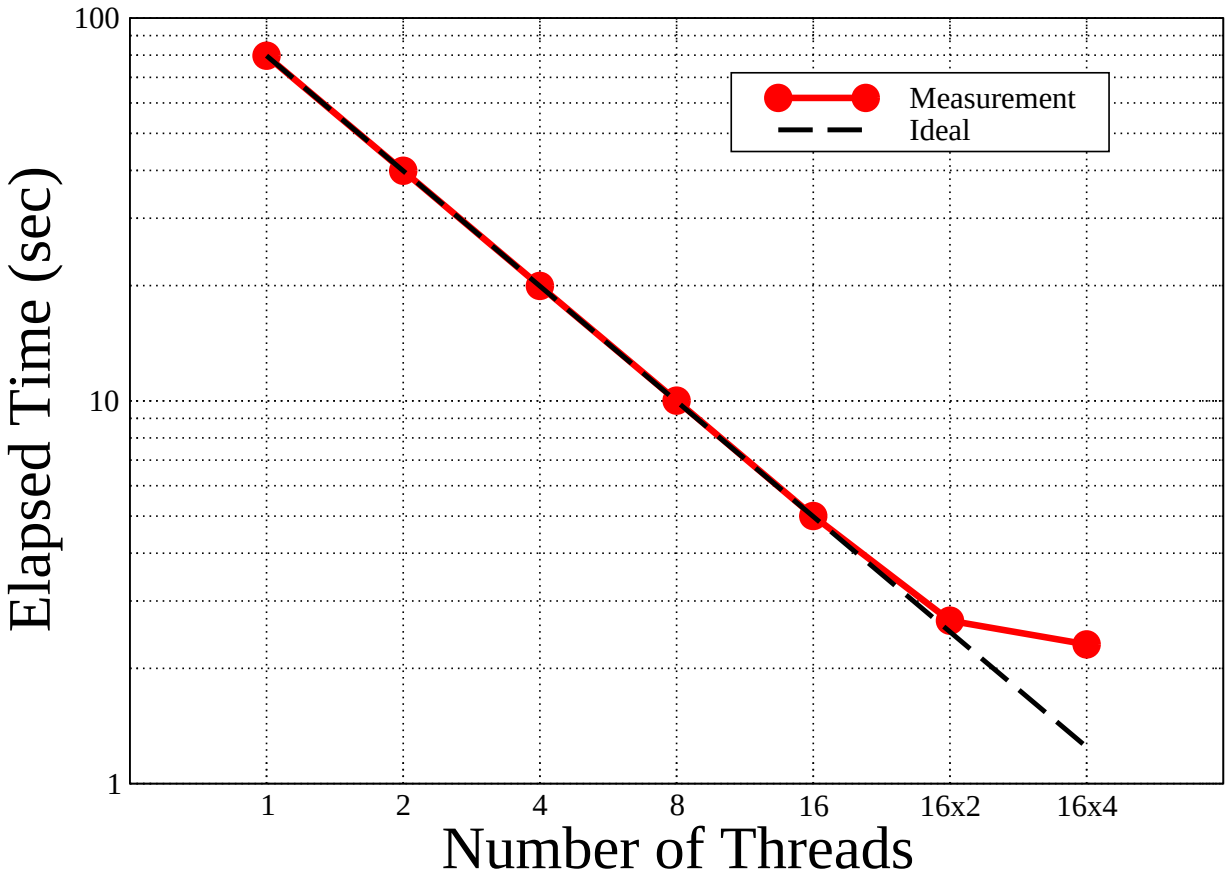
- Discrete Fourier Transform
  - *FFTW*
- Solving Navier-Stokes Eq
  - After DFT in X and Z direction, the problem becomes One dimensional PDE
  - $\partial/\partial x \rightarrow ik_x, \quad \partial/\partial z \rightarrow ik_z$
- Each data line is independent of the other data lines.
- Feasible data structure for data parallelization (OpenMP)



# Threading Result (Discrete Fourier Transform)



# Threading Result (Solving Navier Stokes Eq)



# Threading - Performance Tips

- Load Balance

```
!-----------------------------------------------------
!When OMP_NUM_THREADS=16, x_size=24, y_size=18
!-----------------------------------------------------
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(i,j)
!$OMP DO SCHEDULE(STATIC)
do i=1,x_size
  do j=1,y_size
    ...                    !This is BAD
  enddo
enddo
!$OMP ENDDO
!$OMP END PARALLEL
!-----------------------------------------------------
total_size = x_size * y_size !(= 24x18 = 16x27)
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(i)
!$OMP DO SCHEDULE(STATIC)
do i=1,total_size
  ...                    !This is GOOD
enddo
!$OMP ENDDO
!$OMP END PARALLEL
!-----------------------------------------------------
```

# Threading - Performance Tips

- Memory Allocation / Access

```
complex(C_DOUBLE_COMPLEX) :: U(ny,xsz,zsz,5)
real(C_DOUBLE),dimension(2*bw+1,ny) :: D0,D1,D2
...
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(i,j,k)
!$OMP DO SCHEDULE(STATIC)
do i=1,n_total
   k=mod(int(i-1),xsz) + 1
   j = (i-1)/xsz + 1
      call subroutineX(U(:,k,j,1),U(:,k,j,2),U(:,k,j,3),&
        U(:,k,j,4),U(:,k,j,5),D0, D1, D2)
enddo
!$OMP END DO
!$OMP END PARALLEL
```

- Every thread access to shared memories, D0, D1, D2
  - Heavy memory contention

# Threading - Performance Tips

- Memory Allocation / Access
  - Avoiding memory contention by using FIRSTPRIVATE

```
complex(C_DOUBLE_COMPLEX) :: U(ny,xsz,zsz,5)
real(C_DOUBLE),dimension(2*bw+1,ny) :: D0,D1,D2
...
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(i,j,k) FIRSTPRIVATE(D0,D1,D2)
!$OMP DO SCHEDULE(STATIC)
do i=1,n_total
   k=mod(int(i-1),xsz) + 1
   j = (i-1)/xsz + 1
   call subroutineX(U(:,k,j,1),U(:,k,j,2),U(:,k,j,3),&
        U(:,k,j,4),U(:,k,j,5),D0, D1, D2)
enddo
!$OMP END DO
!$OMP END PARALLEL
```

- Better performance, but cost for allocation/deallocation

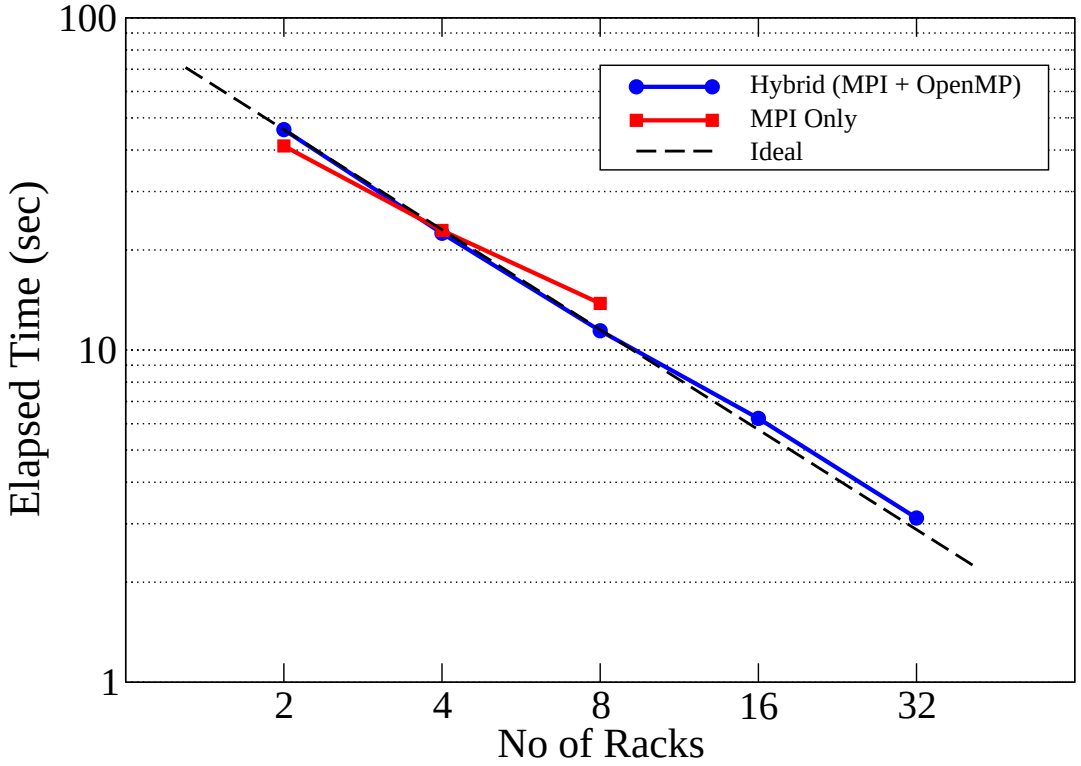
# Threading - Performance Tips

- Memory Allocation / Access
  - Try to allocate all required memory beforehand

```
complex(C_DOUBLE_COMPLEX) :: U(ny,xsz,zsz,5)
real(C_DOUBLE),dimension(2*bw+1,ny,max_omp_num) :: D0,D1,D2
...
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(i,j,k,l)
l = omp_get_thread_num() + 1
!$OMP DO SCHEDULE(STATIC)
do i=1,n_total
   k=mod(int(i-1),xsz) + 1
   j = (i-1)/xsz + 1
   call subroutineX(U(:,k,j,1),U(:,k,j,2),U(:,k,j,3),&
        U(:,k,j,4),U(:,k,j,5),D0(:,:,l), D1(:,:,l), D2(:,:,l))
enddo
!$OMP END DO
!$OMP END PARALLEL
```

- Best performance !!

# Full Timestep Computation




- 8 Racks : 100.8% parallel efficiency
- 32 Racks : 92.3% parallel efficiency

# Summary and Future Work

- Summary
  - Achieved good scalability with hybrid parallelization
  - Remarkable performance improvement with some code tuning techniques
- Future work
  - Running code until it statistically converges
  - Collecting statistical data
  - Analyzing interaction between inner layer and outer layer

THANK YOU!!

Questions?