Tera-scalable Fourier Spectral Element Code for DNS of Channel Turbulent Flow at High Reynolds Number

Jin Xu

Physics Division
Argonne National Laboratory
Argonne, IL 60439 USA
jin_xu@anl.gov

Abstract

Due to the extensive requirement of memory and speed for direct numerical simulation (DNS) of channel turbulence, people can only perform DNS at moderate Reynolds number before. With the fast development of supercomputers, it has become more and more approachable for researchers to perform DNS of turbulence at high Reynolds number. This makes it imperative to consider the development of tera-scalable DNS codes that are capable of fully exploiting these massively parallel machines. In order to achieve this, three parallel models (1D, 2D and 3D domain decompositions) have been implemented and benchmarked. All these models have been successfully ported on BlueGene/L. We have benchmarked these models on BG/L at ANL and BGW at IBM Watson center. Details of these models have been described, discussed and presented in this paper. The optimized model can be used to perform DNS at high Reynolds number in the near future.

Key word: DNS, Fourier Spectral Element Method, Domain Decomposition, high Reynolds number

1. Background

Turbulence remains one of the unsolved physical problems in 20th century. It has great importance in fluid dynamics, as turbulence nearly exists in all areas of fluids. After many years of research, turbulence still remains open challenge to the scientific and engineering communities. People have tried to solve turbulence in theory for centuries. However, due to its extremely complexity of nonlinearity, it is very difficult to understand turbulence in theory. So people become more interested in solving it through computer simulation. In 1973, Steven Orszag performed the first DNS on a 32^3 mesh at NCAR on a CDC7600 computer with only 50 Mbytes memory. This is a 3D homogeneous turbulence with periodic boundary conditions in all three directions. Since it uses Fourier series as expansion bases, the speed is relatively faster. Direct Numerical Simulation of Turbulence by a Fourier Spectral Method on the Earth Simulator has been reported on SC2002 with 2048^3 mesh. However, turbulence in wall bounded region is more difficult to perform DNS. Usually it is more time consuming than the 3D homogeneous case. In 1987, Kim, Moin and Moser (KMM) published their first paper on DNS of channel turbulence[8]. After that, interests on DNS greatly increased. However, as the cost of channel turbulent DNS is proportional to Re^3, and the limit of computer speed and memory, people can only perform DNS at moderate Reynolds number($< Re_\tau = 600$). Recently with the supercomputer speed becomes faster and faster, there is a revised interest on high Reynolds number DNS. In 1999, Robert Moser has reported his results at $Re_\tau = 595$, which is the highest Reynolds number at that time. In 2003 and 2005, Japanese researchers have reported DNS results at $Re_\tau = 1152$ and $Re_\tau = 2320$, using 512 and 2048 processors on Earth Simulator, respectively. Right now many researchers are working on the optimization of parallel models for DNS of turbulent channel flows[3, 12]. This is a highly competitive field and results at higher Reynolds number are expected to be reported soon.

Our simulation method is similar to that of Kim, Moin and Moser[9]. The computational domain is a plane channel with periodic boundary conditions in streamwise (x) and spanwise (z) directions. The difference of the discretization is in the wall normal (y) direction, where we use spectral element expansion instead of the Chebyshev polynomials in one element. In the streamwise and spanwise directions, we use Fourier series as the expansion bases, which is same as KMM's.

As the cost of channel turbulent DNS is proportional to Re_m^3 ($Re_m = U_m * H/\nu$, U_m is the mean velocity of fluid, H is the channel height and ν is the kinetic viscosity), it remains very challenging to do channel turbulence DNS at high Reynolds number. Since DNS is time and memory consuming at high Reynolds number, we need to design tera-scalable models for the simulation of turbulent channel flow that allows efficient scaling on tens of thousands of processors. In order to do this, we have implemented three different parallel models. The first model uses domain decomposition in the streamwise direction (model A), and it is the simplest one among these three models. However, this model has the inherent limitation on the maximum number of processors that can be used. As we need to have at least two planes allocated to each processor in order to do de-aliasing in the nonlinear step, therefore this model is limited by the total mesh points in the streamwise direction. In order to overcome this bottleneck, the other two parallel models were implemented. In the second model (model B), the domain is decomposed in both streamwise and spanwise directions. In the third model (model C), the domain is decomposed in all three directions (streamwise, spanwise and wall-normal direction). Using models B and C, we can easily scale the simulation to tens of thousands of processors. Model C is more flexible than model B as the number of processors can be altered in all three directions. Implementation details and benchmark results are presented in the following sessions. We will also present our benchmark results on BGW at IBM Watson center, which the second fastest machine on the world now. At last, DNS results at $Re_\tau = 1000$ ($Re_\tau = U_\tau * H/\nu$, U_τ is the friction velocity) are reported, which use about 0.7 billion grid points (8.64G bytes/variable).

2. Numerical Method

The discretization is similar to that of Kim, Moin and Moser[8]. The difference is in the wall normal direction, where we use a spectral element expansion instead of the Chebyshev polynomials. In the streamwise and spanwise directions, we use a Fourier series as the expansion bases, which is the same as KMM's. Under this framework, the velocity can be expressed in the following form within an element:

$$u(x,y,z,t) = \sum_{m=-M/2}^{M/2-1} \sum_{n=-N/2}^{N/2-1} \sum_{p=0}^{P} \hat{u}(m,p,n,t) e^{-i\alpha m x} e^{-i\beta n z} \phi_p(y) \quad (1)$$

Where $\phi_p(y)$ are spectral modes in each spectral element. The mapping between global coordinate y and local element e, local coordinate ξ $(-1 < \xi < 1)$ can be expressed as following:

$$y = \frac{1-\xi}{2} y_{e-1} + \frac{1+\xi}{2} y_e \quad (2)$$

The modal expansions adopted in this work are Jacobi polynomials, $P_p^{\alpha,\beta}(x)$ [7]. Jacobi polynomials are the family of polynomial solutions to a singular Sturm-Louiville problem and for $-1 < x < 1$, can be written as

$$\frac{d}{dx}\left[(1-x)^{1+\alpha}(1+x)^{1+\beta}\frac{d}{dx}u_p(x)\right] = \lambda_p (1-x)^\alpha (1+x)^\beta u_p(x) \quad (3)$$

with $u_p(x) = P_p^{\alpha,\beta}(x), \lambda_p = -p(\alpha+\beta+p+1)$. Jacobi polynomials have the orthogonality property

$$\int_{-1}^{1} (1+x)^\beta (1-x)^\alpha P_p^{\alpha,\beta}(x) P_q^{\alpha,\beta}(x) dx = C\delta_{pq} \quad (4)$$

with C depending on α, β, p. Thus, $P_p^{\alpha,\beta}(x)$ is orthogonal to all polynomials of order less than p, when integrating with $(1+x)^\beta(1-x)^\alpha$ and the modal expansion basis is then defined as

$$\phi_p(\xi) = \begin{cases} \frac{1-\xi}{2} & ,p=0 \\ \frac{1}{4}(1-\xi)(1+\xi)P_{p-2}^{1,1}(\xi) & ,0<p<P \\ \frac{1+\xi}{2} & ,p=P \end{cases} \quad (5)$$

where $P_p(y)$ are the standard Jacobi Polynomials ($P_p^{\alpha,\beta}(y)$) with $\alpha = \beta = 1$ (Legendre Polynomial) in the form.

Using this method, we can choose the position of an element boundary, so that we can control the number of points in the near wall region. According to [4], the DNS simulation usually needs at least 13 points in the first 10 wall units, so that it is enough to resolve the smallest turbulent structure in the near wall region.

The velocity field $\vec{V}(x,y,z,t)$ of incompressible flow satisfies

$$\frac{\partial \vec{V}}{\partial t} + \vec{V}\cdot\nabla\vec{V} = -\nabla p + \nu\nabla^2\vec{V}, \qquad \nabla\cdot\vec{V} = 0 \quad (6)$$

Where ν is the fluid viscosity and p is the pressure. We use the high-order time splitting method of Karniadakis, Israeli, Orszag (1991)[6] to do the time integration.

1. Nonlinear step:

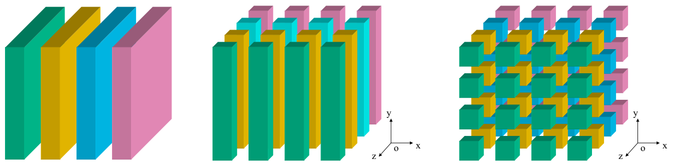

Figure 1. Sketch for three parallel models

$$\frac{\vec{V}^{s+1/3}-\sum_{q=0}^{J_i-1}\alpha_q\vec{V}^{s-q}}{\Delta t}=\sum_{q=0}^{J_e-1}\beta_q N(\vec{V}^{s-q}) \qquad (7)$$

J_i, J_e are orders, and α_q,β_q are coefficients in splitting scheme.

2. Pressure step:

$$\frac{\vec{V}^{s+2/3}-\vec{V}^{s+1/3}}{\Delta t} = -\nabla p^{s+1} \qquad (8)$$

$$\nabla\cdot\vec{V}^{s+2/3} = 0 \qquad (9)$$

$$\begin{aligned}\frac{\partial p^{s+1}}{\partial n} &= \nu\sum_{q=0}^{J_e-1}\beta_q(-\nabla\times(\nabla\times\vec{V}^{s-q}))] \\ &+ \vec{n}\cdot[\sum_{q=0}^{J_e-1}\beta_q N(\vec{V}^{s-q}) \qquad (10)\end{aligned}$$

3. Viscous step:

$$\frac{\vec{V}^{s+1}-\vec{V}^{s+2/3}}{\Delta t}=\nu\nabla^2\vec{V}^{s+1} \qquad (11)$$

In order to eliminate the aliasing error generated in nonlinear step, we perform the nonlinear step in physical space using 3/2 rule. This means that we expand the mesh by 3/2 times larger in both streamwise and spanwise directions. After evaluating the nonlinear terms, we transform them back to the normal mesh. The code uses FFTW[2], which can optimize its performance on different platforms to do Fast Fourier Transform. The high performance numerical libraries such as BLAS, LAPACK have been used in the code. The code has been parallelized using MPI, and benchmarked on different platforms, such as BlueGene/L, SPX, SGI, HP, LINUX Cluster, etc. The benchmark results on BlueGene/L are reported in detail as follows.

3 Parallel Models Implementation

3.1 Domain Decomposition in X Direction (Model A)

In Figure. 1 (left), we present the sketch of model A. The domain has been decomposed only in the streamwise direction, and it is the simplest model for channel flow solver. The domain is divided in streamwise direction, therefore the Fast Fourier Transform (FFT) can not be performed directly in this direction. In order to complete FFT, we need to do transpose of whole data in (x, z) plane. The interpolation and extrapolation in model A for dealiasing have been performed by shifting data to its next neighbor. This need to be performed between each processor and its succeeding processor for N times (N is the total number of processors). That means each time the data is shifted from one processor to its subsequent processor, and after N times, the data is returned to its original position, i.e. $0\rightarrow1\rightarrow2\rightarrow\ldots\rightarrow N-1\rightarrow0$. During this process, each processor can compute to decide which portion of the data should be copied. After one turn, all processors have the data they need for interpolation or extrapolation.

3.2 Domain Decomposition in X and Z Directions (Model B1, B2 and B3)

In Figure. 1 (middle), we present the sketch of model B. The domain has been decomposed in both streamwise and spanwise directions. Since we decomposed the domain in both streamwise and spanwise directions, we cannot perform a FFT directly at any direction. We can not apply transpose in (x, z) plane either, since both of them are divided into different processors. In order to do a FFT, we implemented two different methods. In the first one, we used two new communication groups, comx and comz. Communication group comx contains all processors that have the same x locations, and communication group comz contains all processors that have same z location. They are different from the global communication group, MPI_COMM_WORLD, which contains all processors in the code. While comx and comz only have a portion of all the processors. FFT in x direction is applied after each processor performs transpose of (y, z) plane in corresponding communication group comz, then there is another transpose to transform data back to its original position. FFT in z direction is done similarly by transpose of (y, x) plane in each corresponding communication group comx, and after that, using another transpose to transform data back to its original position. The second method collected data from all processors at several planes, and performed 2D FFT transform, then distributed them back to original processors.

Domain decomposition in two directions makes it difficult to perform dealiasing by shifting or transpose, since two direction shifting is not easy to implement, and involve $O(N^2)$ times of operations. Transpose in the (x, z) plane is not applicable either, because both x and z directions are

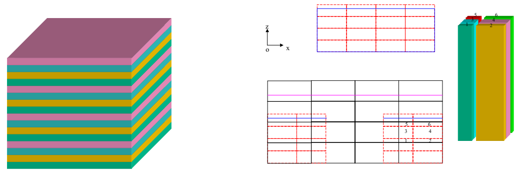

Figure 2. Dealiasing of model B1(left) and B3(right)

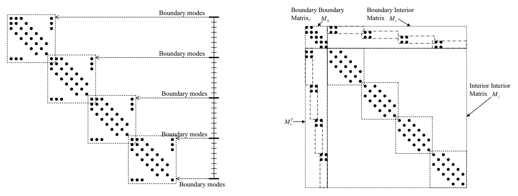
Boundary modes
Boundary modes
Boundary modes
Boundary modes
Boundary modes
Boundary Boundary Matrix, M_b
Boundary Interior Matrix M_c
Interior Interior Matrix M_i
M_c^T

Figure 3. Helmholtz matrix for model C

divided. In order to perform interpolation and extrapolation for the dealiasing, we have implemented three different methods. The first approach (referenced as model B1) is to redistribute data on all processors, as shown in Figure. 2 (left). In the case of 16 processors shown in Figure. 1 (middle), 16 processors have redistributed the data to 16 blocks, and each block has several planes of original data. After that the data on each plane can be expanded to 3/2 in both x and z directions. In the end, the dealiased data are redistributed back to the original processors. The second approach (referenced as model B2) is to transpose separately in (y, z) and (y, x) planes as we do for the FFT step. The interpolation and extrapolation for dealiasing have been performed separately in x and z directions, processors which have different x location need to perform $MPI\_Alltoall$ in different communication group. After the operation in one direction is finished, all processors start the operation in the other direction. The third approach (referenced as model B3) is to send data from all processors to their 6 neighboring processors. In the preprocessing, we need to decide for each processor: on the outbound, the amount of data it needs to send and to which processor it should send those data; and on the inbound, how many data it needs to receive from its peer processor(s). Since the mesh after interpolation is 3/2 the original one in both x and z direction, the maximum number of peer processors that one processor will communicate with can be estimated. Each processor can receive data from at most 6 different processors. This has been done in the preprocessing. We will compare all these three dealiasing models in details later.

As with model A, model B1 also has limitations. When the total mesh points in the wall normal direction is more than the total number of processors, Model B1 usually can achieve similar performance as Model B2. Detailed results of comparison will be shown later. However, when the total mesh points in the wall normal direction is less than the total number of processors, not every processor can have one plane of data for interpolation or extrapolation. For example, when we use 1024 processors, and the total number of mesh points in wall normal direction is only 257, then using Model B1, only 257 processors can perform 2D interpolation and extrapolation, while the other 767 processors will be idle and wait for synchronization. This will waste computing resource and degrade the performance. Some comparisons are shown later.

3.3 Domain Decomposition in X, Y and Z Directions (Model C)

Domain decomposition in two directions has greatly improved the performance and scalability, and this inspired us to do domain decomposition in three directions. In addition, the spectral element method has inherent characteristics of local element, therefore we have divided the domain in y direction to the same number of elements (model C). As shown in Figure. 1 (right), model C has the domain decomposition in both streamwise, spanwise and wall normal directions. This model increases the number of possible combination of (N_x, N_y, N_z) for three directions, where N_x, N_y and N_z are the processor numbers in x, y and z directions, respectively. In model C, the total number of processors in y direction, N_y is equal to the total number of elements in the wall normal direction. In addition to the global communication group MPI_COMM_WORLD, we have generated additional three communication groups (comx, comy and comz), which have all processors at the same x, y and z location. Since BlueGene/L also has 3D torus network structure, this model has potential to fit each processor into its corresponding position in 3D network. This may achieve the best performance.

Figure. 3 (left) shows the Helmholtz matrix solved at the viscous step, which is solved globally in models A and B. In order to do domain decomposition in y direction, we transform the matrix into the form shown in Figure. 3 (right), which put all coupled boundary modes together on the top, and separate the interior modes from different elements. On each processor, we keep a copy of boundary boundary matrix and boundary interior matrix of its own. Each processor will first solve the boundary modes, which includes its own boundary. Then we use Shur complement method to subtract the contribution of its boundary modes, and solve its own interior modes in the end.

The spectral element method we used is based on C^0 bases, which means it can only guarantee the continuity

Re_τ	Re	Domain	DNS mesh	DNS data
180	4300	$2\pi \times 2 \times 2\pi$	$128 \times 130 \times 128$	64M
600	18000	$2\pi \times 2 \times \pi$	$384 \times 361 \times 384$	1.5G
1000	27500	$6\pi \times 2 \times 1.5\pi$	$768 \times 521 \times 768$	8.64G

Table 1. Simulation parameters for DNS.

of the function at the boundaries, but cannot guarantee the continuity of the derivatives of the function. Indeed, we usually see different values for the derivative along the y direction on either side of the boundary. We average them to get the correct value. So we need to average them to get the correct value. This needs data communications at the boundaries. We have implemented two different methods. The first one uses MPI_Send and MPI_Recv. Except for the last processor in the communication group comy, each processor sends data of its upper boundary to its upper node in the group; and then each processor sends data of its lower boundary to its lower node in the group, except the first processor in the group. The second method copies upper and lower boundary planes to the corresponding locations in a 3D array, which has same dimensions in the (x, z) plane. The total mesh points of this array in the y direction are equal to the total element number plus one. Each processor simply copies two planes of data to its corresponding positions, and sets all other planes to zero. Then a $MPI\_Allreduce$ is used to do global summation in communication group comy. After that, each processor can obtain the correct value from this array. When the number of elements in the y direction is small, these two methods have similar performance. However as the number of elements in the y direction becomes large, the second method will need to transfer much more data than the first one. Therefore the first method is a better choice.

4 Implementation

4.1 Direct and Iterative Solver

To solve Helmholtz and Laplacian matrices, we applied a direct solver in our models. We will implement iterative solvers and compare them with direct solvers in a future work.

4.2 Memory

As the Reynolds number increases, the number of mesh points needs to increase in a cubic fashion, as shown in Table. 1. At high Reynolds number, the mesh size should be smaller than the smallest scale of vortices. Suppose we need $2048 \times 1025 \times 2048$ mesh to simulate $Re_\tau = 2000$, then the total memory for double precision is $2048 \times 1025 \times 2048 \times 8 = 34.4GB$. In order to save memory, we utilized the characteristics of Fourier coefficients. As the coefficients are conjugate in Fourier space if the values before Fourier Transform are real, we only need to store array of size $2048 \times 1025 \times 1024$ in Fourier space. This leads to the same amount of memory as needed to represent the physical space, since complex numbers occupy twice the memory of real numbers.

4.3 Speed

Kim, Moin, and Moser[8] suggested that the time step required for the simulation should be

$$\Delta t \approx \frac{0.003}{\sqrt{Re_\tau}} \frac{H}{u_\tau}. \tag{12}$$

H is the channel height. Hence, assuming that $5H/u_\tau$ seconds are required to reach a statistically-steady state, then the total time needed are

$$5 \times H/u_\tau = 5 \times \frac{\sqrt{Re_\tau}}{0.003} = 5 \times \frac{\sqrt{2000}}{0.003} = 75000. \tag{13}$$

If each time step takes about 35 seconds, then the total time needed will be $75000 \times 30/3600/24 \approx 26$ days. However, this estimate is based on Chebyshev polynomials in one element. Since we are using the spectral element method, the smallest mesh size in y direction is much larger than that of the Chebyshev polynomial-based method. So the time step Δt can be much larger, and the total time needed for our simulation should be much less than the above estimation. We examine running on the BlueGene system in our benchmark results later.

4.4 Factorization

In order to speedup the simulation, we need to employ factorization techniques. On the other hand, we need to use a larger mesh for DNS at higher Reynolds number. This is a contradiction, and we need to balance the memory and speed requirements. If using factorization technique for the global Helmholtz matrix, the increment of the total number of matrices is P, where P is the total mesh number in the y direction. For example, if P=769, then we need 769 more 3D arrays to save factorization matrices. This is almost impossible, as the total arrays needed for the DNS without factorization is only 25. If we transform the global matrix to separate the boundary matrix and interior matrices, and do the factorization of each local matrix, then the memory requirement will be much lower, as shown in Figure. 3.

$Nx \times Ny \times Nz$	Proc. No.	Nl.	Tot.	Nl./Tot.
$512 \times 513 \times 512$	512	40.61	54.78	74.1%
$768 \times 513 \times 768$	1024	40.64	56.21	72.3%
$1024 \times 513 \times 1024$	2048	52.89	68.8	76.9%

Table 2. Time ratio of nonlinear step for model B1

$Nx \times Ny \times Nz$	Proc. No.	Nl.	Tot.	Nl./Tot.
$512 \times 513 \times 512$	512	20.56	34.97	58.8%
$768 \times 513 \times 768$	1024	19.68	35.22	55.9%
$1024 \times 513 \times 1024$	2048	18.19	32.75	55.5%

Table 3. Time ratio of nonlinear step for model C

Since the matrix is local, the maximum bandwidth is the number of spectral modes in each element. Furthermore, if we check the structure of Helmholtz matrix more carefully, we can see it is actually a tridiagnal matrix. That means by using factorization technique for banded matrices, the total number of additional 3D arrays needed is only 6. Speedup of pressure and viscous steps will be twice as fast, as a result.

5 Benchmark Results on BG/L at ANL

5.1 Dealiasing in Nonlinear Step

Table. 2 and 3 show the results for the same grid number in y direction, but with increasing numbers of processors and mesh size for model B1 and C. The percentage of total time spent on the nonlinear step remains relatively constant across increasing numbers of processors because the data transferred through all processors commensurately increases. In addition, when using a very large mesh, most of the simulation time is spent on the nonlinear step (approximately 76%) during each time step. Because we need to scale the "pre-de-aliased" mesh, in preparation of de-aliasing, by a factor of 9/4, we need to exchange a large amount of data between different processors for high Reynolds number simulation. This is the major bottleneck that resulted in the poor performance.

For example, overall we have 64 processors, and we have 16 elements in y direction. So we have 16 processors allocated to y direction for model C, then the total number of processors in (x, z) plane is 4. As we use the same number of processors and same mesh, the total processors used in

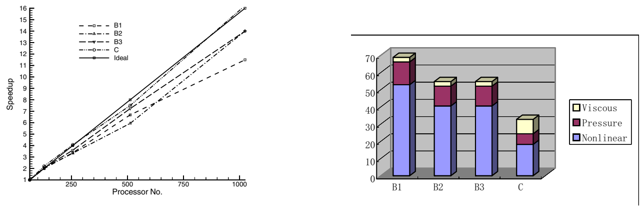

Figure 4. Comparison of model B and C

(x, z) plane of model B is 64. The data allocated to each processor in (x, z) plane of model B is 16 times less than that of model C, but is 16 times larger in y direction than that of model C. So each processor will transfer the same size of data for models B and C. Because there are only 4 processors in the (x, z) plane for model C, while there are 64 processors for model B, the transpose in model C is much quicker than model B. The ratio of nonlinear step drops to about 56%.

As shown in Table. 4, we can verify the assumption that model B1 and B2 will perform differently under different conditions. As the total number of grid points in y direction is more than the total number of processors, these two models will have similar performance. However, as the total number of grid points in y direction is less than the total number of processors, model B2 will have better performance. For 16, 64, 256, 512 and 1024 processors, B1 and B2 do have similar performance for nonlinear step. However, for 2048 processors, model B2 is 25% faster than model B1 for the nonlinear step. There is also some difference between model B2 and B3. B3 has better performance than B2 at small mesh and small number of processors. However, they have similar performance for 2048 processors, as more time will be spent on copying data in and out of the communication buffers.

5.2 Scalability

Figure. 4 (left) shows the scaling of each model on mesh $256 \times 257 \times 256$. We can see from the plot that all models have relatively good scaling. The solid line in the plot is the ideal speedup. Model B2 and B3 are a little bit better than B1. Model C is the best model among all these.

Table. 4 shows the time for each step on different mesh with different number of processors. From this table, we can see that model B1, B2 and B3 have similar performance. Model C is the best, its speed is about 40% faster than model B1. Interesting thing is that not all steps in model C are faster than those of model B. Actually the viscous step takes more time in model C than any other models, the reason may be due to the extra communication in the communication group comy. In model B, there is no communication; unlike model C, which has much communication.

CPU =16	NX =128	NY =129	NZ =128	KE =4
$Model$	NL	PRES	VIS	Total
B1	8.4	2.37	1.27	12.04
B2	9.39	2.46	1.27	13.12
B3	9.08	2.42	1.27	12.77
C	6.88	1.88	1.68	10.44
CPU =64	NX =256	NY =257	NZ =256	KE =8
$Model$	NL	PRES	VIS	Total
B1	23.5	6.9	2.7	33.1
B2	25.11	7.06	2.71	34.88
B3	24.97	7.06	2.71	34.74
C	15.6	4.48	4.76	24.84
CPU =256	NX =512	NY =257	NZ =512	KE =16
$Model$	NL	PRES	VIS	Total
B1	30.8	7.89	2.71	41.4
B2	31.67	7.92	2.72	42.31
B3	30.66	7.95	2.71	41.32
C	20.02	4.42	2.92	27.36
CPU =512	NX =512	NY =513	NZ =512	KE =16
$Model$	NL	PRES	VIS	Total
B1	40.61	11.53	2.64	54.78
B2	42.64	11.62	2.58	56.84
B3	34.15	11.61	2.58	48.34
C	20.56	6.26	8.15	34.97
CPU =1024	NX =768	NY =513	NZ =768	KE =16
$Model$	NL	PRES	VIS	Total
B1	40.64	12.51	3.06	56.21
B2	41.14	12.43	3.06	56.63
B3	39.43	12.08	3.04	54.55
C	19.68	6.62	8.92	35.22
CPU =2048	NX =1024	NY =513	NZ =1024	KE =16
$Model$	NL	PRES	VIS	Total
B1	52.89	13.26	2.65	68.8
B2	40.53	11.52	2.65	54.7
B3	40.54	11.5	2.65	54.66
C	18.19	6.18	8.38	32.75

Table 4. Weak Scaling on different meshes

Ca.	Mesh	Proc.	El.	Rack	T.
I	$2048\times513\times2048$	16384	16	16	17.0
J	$2048\times513\times2048$	32768	16	16	14.8
K	$2048\times769\times2048$	16384	24	8	39.6
L	$2048\times769\times2048$	16384	24	16	29.0
M	$2048\times769\times2048$	32768	24	16	22.6

Table 5. 2 to 16 racks on BGW at IBM Watson center

The encouraging thing is that the nonlinear step in model C occupies half the time of that in the other models. This contributes to the overall speedup of about 40%.

5.3 Comparison of Model B and C

We also compared different models using the same mesh and the same number of processors. The mesh is still $512 \times 257 \times 512$, with 8 elements in the wall normal direction and 33 points per element. Since there are 8 elements in the y direction, when we use model C, we have 8 processors in the y direction. Then we can test model C with varying combinations of N_x and N_z. Figure. 4 (right) compares the total speed of Model B1, B2, B3 and C. Model C is the best model as shown in the figure.

6 Benchmark results on BGW

In order to check the scalability of Model C on large number of processors, we have done the tests on BGW at IBM Watson center, which is the second fastest machine on the world now.

We test Model C on 16 racks, with 32768 processors. We also obtained quite good scalability as shown in table. 5. From the table, we can see that as each compute node has two processors for BG/L, so there are two ways to use it: virtual node (vn) means to use two processor on each node, and co-processor (co) mode mean use only one processor on each node. Running same number of processors in vn mode is usually about 40% faster than in co mode. Case K and L show about 36.6% speedup in vn mode. And model C can be successfully run on 16 racks with 32768 process.

7 Simulation Results at High Reynolds Number

Table. 6 compares the mesh and parameters used by different researchers for $Re_\tau = 1000$, The first two cases have been done by Moser[10], and the third case has been done by Iwamoto[4]. This last case is our DNS result[12]. We

$Nx \times Ny \times Nz$	Re_τ	Lx	Lz	Δx^+	Δy^+	Δz^+
$768 \times 769 \times 768$	1901	π	$\pi/2$	7.8	7.8	3.9
$3072 \times 385 \times 2304$	934	8π	3π	7.6	3.8	7.6
$1152 \times 513 \times 1024$	1160	6π	2π	19	-	7.1
$768 \times 521 \times 768$	934	6π	1.5π	22.9	7.66	5.7

Table 6. DNS at $Re_\tau = 1000$.

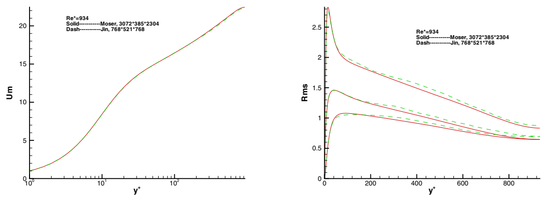

Figure 5. Compare mean and rms at $Re_\tau = 1000$

can see this simulation can be easily done on one rack BG/L system. And we can perform much higher Reynolds number DNS on BGW using 16 racks of 32768 processors.

At last, we give our simulation results at $Re_\tau = 1000$ running on PSC, using 1024 processors with model B1. The mesh is $768 \times 513 \times 768$, and $\delta t = 1/2000$, $Re_m = 28000$.

8 Summary

We have developed and benchmarked several parallel models for DNS of channel turbulent flow. We investigated each parallel model in detail. All these models have shown good scaling up to thousands of processors. The parallel model with 3D domain decomposition has the best performance. Three models based on 2D domain decomposition have similar performance, where the nonlinear step has relatively poor scaling because large data transfers are needed for dealiasing. The viscous step has the best scaling among all three steps. The viscous step in model C is slower than in other models. However, the nonlinear step is much faster than the others. The overall performance of model C is about 40% faster than other models. We also present our recent benchmark result on BGW at IBM Watson center, which show great scalability and prospect to perform higher Reynolds number DNS in the future. At last, some simulation results at high Reynolds number ($Re_\tau = 1000$) has been given.

9 Acknowledgement

I would gratefully acknowledge Assistant Prof. Mike Kirby at Utah University for his help on discussion of parallel algorithms. Special thanks give to Fred Mintzer, James Sexton and Bob Walkup at IBM Watson center for their help on using BGW. I would also like to thank Paul Fischer and Ed Jedlicka in MCS at ANL for their help on BGW tests. This work used machines at other supercomputing centers, such as NAVO, ARSC and NCSA.

References

[1] C. Canuto, M.Y. Hussaini, A. Quarteroni and T.A. Zang. Spectral Methods in Fluid Mechanics. *Springer-Verlag*, New York 1987.

[2] http://www.fftw.org.

[3] C. W. Hamman, R. M. Kirby and M. Berzins. Parallelization and scalability of a spectral element channel flow solver for incompressible Navier Stokes equations. *Concurrency and Computation: Practice and Experience* 2006.

[4] K. Iwamoto and N. Kasagi. DNS of high Reynolds number turbulent channel flow. *Proc. 3th Symp. Smart Control of Turbulence*, Tokyo, March 2003.

[5] K. Iwamoto. Direct Numerical Simulation of Turbulent Channel flow at Re_τ = 2320. *Proc. 6th Symp. Smart Control of Turbulence*, Tokyo, March 2005.

[6] G.E. Karniadakis, M. Israeli and S.A. Orszag. High-order splitting methods for incompressible Navier-Stokes equations. *J. Comp. Phys.*, 97:414-443 1991.

[7] G.E. Karniadakis and S.J. Sherwin. Spectral/hp element methods for CFD. *Oxford University Press*, London 1999.

[8] J. Kim, P. Moin and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *J. Fluid Mechanics* 177:133-166 1987.

[9] P. Moin and J. Kim. Numerical investigation of turbulent channel flow. *J. Fluid Mech.* 118:341-377 1982.

[10] R. Moser, J. Kim and N. Mansour. Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$. *Phys. Fluids* 11:943-945 1999.

[11] http://www.top500.org.

[12] J. Xu. High Reynolds number simulation and drag reduction techniques. *PhD. Thesis, Division of Applied Math.*, Brown University, 2005.