Petascale Direct Numerical Simulation of Turbulent Channel Flow on up to 786K Cores[*]

Myoungkyu Lee
Department of Mechanical Engineering
University of Texas at Austin
Austin, Texas 78735
mk@ices.utexas.edu

Nicholas Malaya
Institute of Computational Engineering and Sciences
University of Texas at Austin
Austin, Texas 78735
nick@ices.utexas.edu

Robert D. Moser
Institute of Computational Engineering and Sciences and Department of Mechanical Engineering
University of Texas at Austin
Austin, Texas 78735
rmoser@ices.utexas.edu

ABSTRACT

We present results of performance optimization for direct numerical simulation (DNS) of wall bounded turbulent flow (channel flow). DNS is a technique in which the fluid flow equations are solved without subgrid modeling. Of particular interest are high Reynolds number (Re) turbulent flows over walls, because of their importance in technological applications. Simulating high Re turbulence is a challenging computational problem, due to the high spatial and temporal resolution requirements.

An optimized code was developed using spectral methods, the method of choice for turbulent flows. Optimization was performed to address three major issues: efficiency of banded matrix linear algebra, cache reuse and memory access, and communication for the global data transposes.

Results show that performance is highly dependent on characteristics of the communication network, rather than single-core performance. In our tests, it exhibits approximately 80% strong scaling parallel efficiency at 786K cores relative to performance on 65K cores.

Categories and Subject Descriptors

[**Algorithms**]: Numerical methods, linear and non-linear systems; [**Applications**]: Computational fluid dynamics

General Terms

Turbulence, MPI_alltoall, Parallel FFT, Petascale, Data transpose

1. INTRODUCTION

Turbulence is a multi-scale fluid flow phenomenon characterized by large unpredictable fluctuations that occur across a wide range of length and time scales. Described by Richard Feynman as, “the most important unsolved problem of classical physics.”, the turbulence problem remains unsolved despite over one hundred years of scientific research. Nevertheless, approximately 20% of global energy consumption is expended on transportation[26], in which vehicles move through the air or water, or fluids are transported through pipes and ducts, and this energy is dissipated primarily in turbulence. Unfortunately, there is currently no theory or model of wall-bounded turbulence of sufficient veracity to support the development of new efficient designs of turbulent fluid systems.

For Direct Numerical Simulations (DNS) of turbulence, the equations of fluid motion (the Navier-Stokes equations) are solved, without further modeling, with sufficient temporal and spatial resolution to represent all the scales of turbulence. Such simulations provide exquisitely detailed and highly reliable data, which have driven a number of discoveries regarding the nature of turbulence [10, 8, 28, 3, 13]. Indeed, for some quantities that are difficult or impossible to measure, DNS is the most reliable or only source of data.

The turbulent flow of fluids past walls at high speed (high Reynolds number) is particularly important in the transportation examples mentioned above, as well as many other applications. The Reynolds number measures the relative importance of inertia relative to viscosity, so that the higher the Re, the weaker the stabilizing effect of viscosity. However, the use of DNS to study flows has been hindered by the extreme computational expense of high Reynolds number turbulence simulation. The ratio of the size of the largest to the size of the smallest turbulent eddies in a flow increases with the Reynolds number. In wall-bounded turbulence like

SC13, November 17-21 2013, Denver, CO, USA
Copyright 2013 ACM 978-1-4503-2378-9/13/11...$15.00.
http://dx.doi.org/10.1145/2503210.2503298

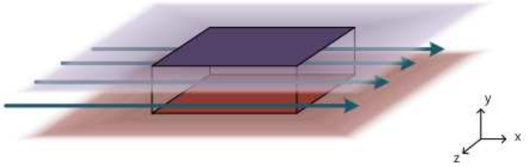

Figure 1: Simulation geometry, fluid flows between two parallel infinite plates

the channel flow simulated here, the result is that the number of degrees of freedom in a DNS and therefore the memory requirement scales with Re^3, while the computational cost scales as Re^4[22]. DNS of turbulence is thus a natural supercomputing problem, which is why the National Science Foundation included turbulence DNS as an essential performance benchmark for the Blue-Waters system[15].

The DNS of turbulent channel flow described here were undertaken to address questions about the nature of wall-bounded turbulence at high Reynolds number. The Reynolds number simulated ($Re_\tau = 5200$ based on the friction velocity) was selected to be large enough to display characteristics of very high Reynolds number wall-bounded turbulence. This Reynolds number requires a simulation with 242 billion degrees of freedom, which to the authors' knowledge is the largest scientific DNS yet conducted, with 3.5 times more degrees of freedom than Kaneda *et al*'s 2004 Isotropic Homogeneous simulation[14] and 15 times larger than the previously largest channel DNS of Hoyas and Jimenez[10], conducted in 2006.

2. THE COMPUTATIONAL PROBLEM

Simulated here is turbulent channel flow, the flow between infinite parallel planes driven by a pressure gradient, as shown in figure 1. For computational purposes the infinite domains in the directions parallel to the wall are truncated to a finite box with periodic boundary conditions in the streamwise (x) and spanwise (z) directions. Zero velocity boundary conditions are imposed at the walls.

The velocity is represented in space as a truncated Fourier series in x and z, and in terms of 7^{th}-order basis splines (B-splines) in the wall-normal direction (y). B-splines were selected for their excellent error characteristics[17] as well as a straightforward formulation using the recursive relation of DeBoor[5]. A Fourier spectral formulation is preferred for turbulence DNS, when applicable, because of its superior resolution properties [2], despite the resulting algorithmic need for expensive all-to-all communication. The communication cost is more than compensated for by the reduced grid size needed by spectral methods for the same accuracy, and the reduced cost of solving the Poisson equation that necessarily arises when solving the incompressible Navier-Stokes equations.

2.1 Numerical Formulation

The incompressible Navier-Stokes equations are solved using the well-known formulation of Kim, Moin and Moser[16]. First, the incompressible Navier-Stokes equations,

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{uu}) = -\nabla p + \nu \nabla^2 \mathbf{u}, \qquad \nabla \cdot \mathbf{u} = 0, \qquad (1)$$

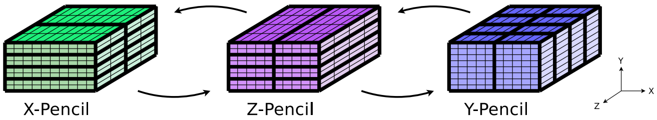

Figure 2: Pencil decomposition and transpose

can be rewritten to eliminate the pressure, and naturally impose the continuity constraint $\nabla \cdot \mathbf{u} = 0$:

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

where ν is the kinematic viscosity, h_g and h_v arise from the nonlinear terms, ω_y is the wall-normal component of the vorticity and ϕ is related to the wall-normal component of the velocity v through $\phi = \nabla^2 v$.

These equations are reduced to ordinary differential equations in time for the Fourier-B-spline coefficients of ω_y and ϕ using a Fourier-Galerkin method in x and z and a B-spline collocation method in y. The Galerkin quadratures for the nonlinear terms, h_g and h_v, are performed using Fourier-Gauss quadrature, which, since the nonlinearities are quadratic, requires $3/2$ as many uniformly spaced quadrature points in x and z as there are Fourier modes in those directions (commonly called dealiasing)[20]. The evaluation of the velocities on the quadrature grid, and the computation of the Galerkin integral are done using the fast Fourier transform (FFT) algorithm.

Time advancement is performed using a mixed implicit-explicit (IMEX) scheme based on third-order low-storage Runge-Kutta [23], with the nonlinear (convective) terms treated explicitly and the viscous terms implicitly. Both the implicit time-stepping and the determination of v from ϕ require the solution of linear algebraic systems with dimension determined by the size of the B-spline basis. For each wavenumber in the Fourier representation, three such linear systems must be solved, each of which are equivalent to solving a second-order, two-point boundary value problem in y using B-spline collocation.

2.2 Parallel decomposition

The most effective way to perform the required FFTs and linear algebra in parallel is to have each parallel process work on the FFT in one dimension (in x or z) or linear algebra (in y), with all data in that direction local, and to decompose the other two spatial directions across processes (MPI tasks). The result is that each process works on a "pencil" of data that is long in the direction in which the transform or linear algebra is being done. The algorithm will perform transforms in each of the horizontal directions and linear algebra in the wall-normal direction, requiring the data repeatedly redistributed across processes to change the orientation of the pencils[7]. A schematic detailing the data partitioning is shown in figure 2. This reshuffling is essentially a global transpose and involves a communication pattern like that implemented in `MPI_alltoall`. This pencil decomposition is used rather than the alternative planar decomposition because it provides far greater flexibility with respect to possible MPI communicator topologies and node counts.

2.3 Simulation steps

Our program starts at the beginning of a time step with the Fourier coefficients for the three velocity components u, v and w evaluated at the collocation points in y, distributed in the y-pencils data configuration. Following is an outline of the operations required for a single Runge-Kutta substep starting at this point.

(a) Transpose the data for three velocity components from y-pencils to z-pencils
(b) Pad the data in z with zeros for FFT on 3/2 larger data vector
(c) Perform inverse Fourier transform in z direction
(d) Transpose the data from z-pencils to x-pencils
(e) Pad the data in x with zeros for FFT on 3/2 larger data vector
(f) Perform inverse Fourier transform in x-direction. The three velocity components are now evaluated on a 3D grid distributed in the x-pencil configuration.
(g) Compute five quadratic products of u, v and w at each point
(h) Reverse steps (f) to (a) in reverse order for the five fields of nonlinear terms. Here, the forward Fourier transform is performed in (f) and (c), and the data arrays are truncated rather than padded in (e) and (b).
(i) Perform the linear algebra needed to time advance equations for the Fourier-B-spline coefficients of ω_y and ϕ.
(j) Perform linear algebra needed to evaluate the new values of the velocity components at the collocation points in y.

3. BENCHMARK SYSTEMS

The DNS code described here was benchmarked on four HPC platforms with different system architectures. They are:

1. **Mira**: Argonne National Laboratory, BlueGene/Q, Power BQC 16C 1.60GHz, Custom 5D Torus
2. **Stampede**: Texas Advanced Computing Center, PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi
3. **Lonestar**: Texas Advanced Computing Center, Dell PowerEdge M610 Cluster, Xeon 5680 3.3Ghz, Infiniband QDR
4. **Blue Waters**: National Center for Supercomputing Applications, Cray XE6, AMD 6276 "Interlagos" 2.3Ghz, Cray Gemini 3D torus interconnect

Most of the performance measurements were conducted on Mira because it is the platform being used for the full-scale scientific runs. Second, on Stampede and Blue Waters, the benchmarks were conducted without using the available accelerators. Another study[24] found excellent scaling properties with the Intel MIC using a compute kernel from this simulation. These benchmarks were conducted using the IBM XL compilers for Blue Gene on Mira (Version 14.1), the Intel compilers for Lonestar and Stampede (version 11.1 and 13.0, respectively), and the Cray compilers (version 8.1.4) for Blue Waters. Finally, the `MPI_wtime()` function was used to measure elapsed time on all platforms.

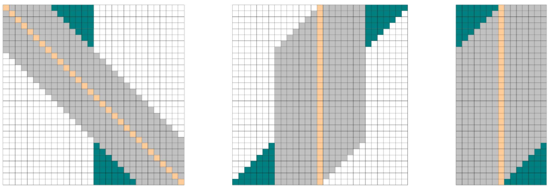

Figure 3: Banded matrix storage (left: original full matrix, center: format for conventional LAPACK banded solver, right: format for customized solver)

4. CODE OPTIMIZATION

A number of optimizations were pursued for the channel DNS code described here to address performance at the single core level, the node level and the system level.

4.1 Single core optimization

There are two primary computation kernels whose single-core performance is important. These are the one-dimensional FFTs and the linear algebra required to time advance the Navier-Stokes equations. We use the FFTW 3.3 library[9] for FFTs, and a custom routine for the linear algebra was developed and optimized as described below.

4.1.1 Linear Algebra

As discussed in section 2.1, time advancing the Navier-Stokes equations using the IMEX scheme reduces to solving a pair of two-point boundary-value problems for ω_y and ϕ for each wavenumber in the Fourier representation. These problems are of the form:

$$\left[I - \alpha\frac{\nu\Delta t}{2}\left(\frac{d^2}{dy^2} - k^2 I\right)\right]\psi = R, \tag{3}$$

where ψ can be either $\hat{\omega}_y$ or $\hat{\phi}$ the Fourier coefficients at the next time level for this wavenumber, I is the identity operator, Δt is the time step, α is a coefficient determined by the time discretization scheme, $k^2 = k_x^2 + k_z^2$ is the sum of the squares of the Fourier wavenumbers in x and z and R is the accumulation of all the terms on the right hand side. Also, determining the wall-normal velocity $\hat{v}$ from ϕ requires the solution of the following ODE:

$$\left[\frac{d^2}{dy^2} - k^2 I\right]\hat{v} = \hat{\phi}. \tag{4}$$

Applying the B-spline collocation method to these equations results in a system of linear algebraic equations, with the matrix representing the operator in brackets in (3) or (4). In both cases the matrix is banded with extra non zero values in the first and last few rows, as shown on the left side of figure 3.

A matrix with this structure can be solved with a general banded matrix solver as in LAPACK, but this is not optimal. First, the full matrix would need to be stored in the form shown in the center of figure 3, and many unnecessary computations would be performed on elements containing zeros. Second, in our case, the matrix is real, while the solution vector and right hand side are complex. This can be solved with the `DBTRF/DGBTRS` routines by rearranging the complex-valued vectors into two sequential real-valued vectors, and then back after solution. To circumvent these

Table 1: Elapsed time for solving a linear system

	Lonestar			Mira	
Bandwidth	MKL^R	MKL^C	Custom	ESSL	Custom
3	0.67	0.65	0.14	0.81	0.16
5	0.55	0.61	0.12	0.85	0.19
7	0.53	0.58	0.11	0.81	0.19
9	0.53	0.56	0.10	0.84	0.19
11	0.47	0.56	0.10	0.88	0.19
13	0.45	0.55	0.11	0.74	0.21
15	0.41	0.53	0.11	0.71	0.20

* $N = 1024$

** Data is normalized by elapsed time using ZBTRF/ZGBTRS of Netlib LAPACK

*** "R" - DBTRF/DGBTRS and "C" ZBTRF/ZGBTRS, For ESSL, DGBF/DGBS, which are equivalent to DBTRF/DGBTRS, are used.

shortcomings and eliminate unnecessary floating point operations, a customized linear algebra solver was developed, based on a custom matrix data structure in which non-zero elements are moved to otherwise empty elements as shown on the right panel of figure 3. In this form, the memory requirement is reduced by half, which is important for cache management. In addition, it is found that compilers inefficiently optimized the low-level operations on matrix elements for the LU decomposition. Instead, loops were unrolled by hand to improve reuse of data in cache.

This customized linear algebra solver was tested on Lonestar and Mira with a problem size $N = 1024$, and results are presented in Table 1, along with benchmarks of implementations from the Intel MKL[4] library (Lonestar) and IBM ESSL[11] (Mira). Note that all times are normalized by the elapsed time for Netlib LAPACK[1]. On Lonestar and Mira, the customized solver is about four times faster than the Intel MKL library or IBM ESSL Library. This speed-up is due to eliminating unnecessary floating point operations and optimizing cache reuse.

4.1.2 Single Core Performance

The single core performance in solving equations (3-4) for time-advancement of the Navier-Stokes equations was measured with the Hardware Performance Monitor(HPM) counter module from IBM on Mira. The quoted theoretical peak Flops per core is 12.8 GFlops, with the LINPACK benchmark attaining 10.38 GFlops. As shown in Table 2, our PDE solver shows about 9% of theoretical peak performance. Here 98.2% of instructions access data in L1 cache (including L1 prefetch). Even though this implies highly efficient reuse of data in cache, the performance of this kernel is clearly limited by the memory bandwidth, given the low number of instructions per cycle and saturated DDR traffic compared to peak performance (18 Bytes/cycle) which is measured by STREAM[27]. The effect of compilation using the SIMD option was also investigated. While compilation with SIMD increased the aggregate Flops, it also increased the computation time. This almost certainly means that compiling with SIMD introduced unfavorable code optimizations which reduced performance. As a result of this finding, all data gathered for benchmarks in this paper are from code compiled without the SIMD option.

Table 2: Single core performance of Navier-Stokes time-advance on Mira

	SIMD	No SIMD
GFlops	4.96 (38.8%)	1.16 (9.05%)
Instruction per Cycle	1.22	0.89
Load hit in L1	98.01	98.2 %
Load hit in L2	1.45	0.92 %
Load hit in DDR	0.53	0.88 %
DDR Traffic (Bytes/cycle)	14.2 (79%)	16.8 (93%)
Elapsed Time (sec)	3.96	3.34

4.2 Threading

There were three major opportunities to take advantage of on-node parallelism through OpenMP threads to enhance single-node performance, rather than relying on MPI. These are the FFTs, the Navier-Stokes time advance and the on-node data reordering that is needed as part of the transpose process. There are two potential advantages of this approach. First, it reduces the total number of MPI messages and thus increases the message size. This reduces the latency cost of the MPI communication. Second, as will be shown below, the optimum degree of on-node parallelism is not the same for all three of the above functions. Using threads rather than MPI allows the degree of parallelism to differ for different functions.

It is straight-forward to thread the FFT calculations because the transform of each full line of data is independent of other data lines. The FFTW 3.3 library is thread-safe and so the FFTW 3.3 calls are simply embedded in OpenMP threads. The FFTW 3.3 library can also use internal threading when processing multiple FFTs at once. This was not used, since it is more efficient to embed the FFTW 3.3 calls together with other threaded operations, such as the padding for the 3/2 expansion of the data lines for dealiasing, or the computation of the nonlinear terms. Indeed, a single OpenMP threaded block spans the inverse x transform, the computation of the nonlinear terms and the forward x transform. This makes the cost of performing the nonlinear products negligible because the data remain in cache across all three operations.

Similar to the FFTs, the linear algebra for the Navier-Stokes time-advance is performed on a single data line (in y for a single wavenumber), with no dependence on other data lines. The entire time advance calculation can thus be performed in a single OpenMP threaded block, which also allows the data to remain in cache throughout the calculation.

As part of the global transpose prosses, the data on each node need to be reordered in an on-node transpose, which takes the form $A(i, j, k) \to A(j, k, i)$. The speed of this on-node transpose is clearly limited by the memory bandwidth, as nothing but memory access is occurring. By dividing this transpose up into independent pieces and threading across the pieces with OpenMP, one is able to improve the utilization of the memory bandwidth by maintaining multiple data streams from memory.

Table 3: Single node performance of FFT/Navier-Stokes time advance

Cores	2	3	4	5	6
Speed up efficiency	2.03/1.99 (101.5%/99.4%)	3.18/2.98 (105.9%/99.3%)	4.07/3.65 7 (101.8%/91.2%)	4.88/4.77 (97.6%/95.5%)	5.49/5.70 (91.4%/95.1%)

Cores	2	4	8	16	16×2	16×4
Speed up efficiency	1.99/2.00 (99.5%/99.9%)	3.96/4.00 (99.0%/99.9%)	7.88/7.97 (98.5%/99.6%)	15.4/15.9 (96.5%/99.6%)	27.6/29.9 (173%/187%)	32.6/34.5 (204%/216%)

* Speed up is normalized by the single core time
** Top(Lonestar), Bottom(Mira)

Table 4: Single node performance of data reordering on Mira

Cores	2	4	8	16	16×2	16×4
DDR Traffic (bytes/cycle)	3.8	7.6	13.6	16.1	15.8	13.6
Speed up	1.98	3.90	5.54	6.24	5.99	5.56
efficiency(%)	99.1	97.5	69.3	39.0	37.4	34.8

* Speed up is normalized by the single core timings
** Maximum DDR traffic (18 bytes/cycle)

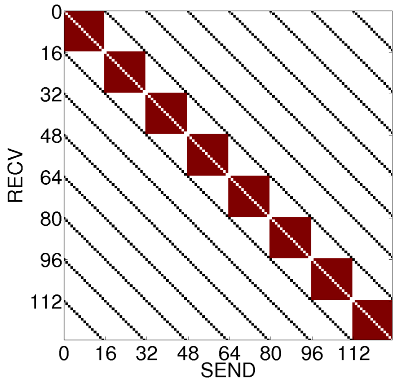

Figure 4: Communication pattern of 128 MPI tasks; Black(CommA) / Red(CommB)

4.2.1 Single node performance

Single node performance was tested on Lonestar and Mira for the FFT and the time-advance kernels. The result in Table 3 shows excellent OpenMP scalability in both cases. While Lonestar possesses 12 cores per node, Table 3 only provides information for up to six cores. This is because each compute node on Lonestar is dual socket, with six cores each, and the threading performance significantly degrades across sockets. On this node, the best parallel strategy is to use one MPI task per socket, with one OpenMP thread per core. On Mira, there are sixteen cores per node with support for four hardware threads per core, so we have tested as many as sixty four OpenMP threads per node. Note that on Mira the parallel efficiency per core exceeds 200% with the use of four hardware threads per core.

Finally, the parallel performance of on-node data reordering was also investigated. As shown in Table 4, scalability of data reordering is not nearly as efficient as the FFTs or Navier-Stokes time advance. This discrepancy can be attributed to memory contention. The FFT and Navier-Stokes time advance kernels are embarrassingly parallel, and the OpenMP threads need not access the same memory locations. The kernel for data reordering, on the other hand, encounters contention. Performance increases with an increasing number of threads until DDR traffic is saturated. After the DDR traffic is saturated, increasing the number of threads only increases contention and decreases performance.

4.3 Global MPI communication

Performance of the global data transposes is dominated by an all-to-all-like communication pattern. Our algorithm requires two different transpose communication patterns: from x-pencils to z-pencils and from z-pencils to y-pencils, and back again. This is accomplished by using two MPI subcommunicators (**CommA** and **CommB**) with the `MPI_cart_create` and `MPI_cart_sub` functions. An example of the communication pattern is shown in Figure 4. Even though the communication pattern in each sub-communicator is of the all-to-all type, it may not be executed using the `MPI_alltoall` routine. This is because we use the FFTW 3.3 communication functionality to orchestrate the global transposes. In the FFTW 3.3 library, multiple implementations of the global transposes are tested (e.g. using `MPI_alltoall` or `MPI_sendrecv`). In this planning stage, the implementation with the best performance on simple tests is selected and used for production.

Table 5 shows the performance measurements for the global MPI communication in our code. The grid size for Mira was $N_x = 2048$, $N_y = 1024$, $N_z = 1024$ and on Lonestar was $N_x = 1536$, $N_y = 384$, $N_z = 1024$. The elapsed time result in Table 5 is the time spent to move the data one full transpose cycle ($x \to z \to y$ then $y \to z \to x$) and it does not include the time for on-node data reordering. There are sixteen cores in a node on Mira and twelve cores to a node on Lonestar. The results in Table 5 indicate that the code performs best when **CommB** stays local to a node. Similarly, when we run the code with many cores $O(10^4 \sim 10^5)$, we localize **CommB** to communication boundaries, e.g. to the interconnect switch boundary, or the 5D torus boundaries. In most cases, FFTW 3.3 chooses `MPI_alltoall` for **CommB**, and chooses either `MPI_sendrecv` or `MPI_alltoall` for **CommA**.

Table 5: Global MPI Communication Performance

System	CommA × CommB	Elapsed Time (sec)
Mira	512 × 16	.386
	256 × 32	.462
	128 × 64	.593
	64 × 128	.609
	32 × 256	.614
	16 × 512	.626
Lonestar	32 × 12	2.966
	16 × 24	3.317
	8 × 48	3.669
	4 × 96	3.775

* Mira - 8192 cores / Lonestar - 384 cores

4.4 Comparison with P3DFFT

Three-dimensional parallel FFTs are an important component of many high performance simulation codes. For instance, the 2012 Gordon Bell Prize winner stated that converting to, "a 3-D parallel FFT library will significantly improve the performance and scalability"[25].

In this section, we provide a direct comparison between our parallel FFT library built on the FFTW 3.3 communication library and P3DFFT [21] on a variety of systems at scale. The P3DFFT library is one of most widely used libraries for 3D parallel FFTs, and it was developed to support direct numerical simulation of turbulence, among other applications. This makes P3DFFT an ideal point of comparison.

The benchmarks reported here were performed using the latest released version of P3DFFT (2.5.1). To ensure that this benchmark is as close as possible to the production calculations in the channel code, the FFT after the last transpose is not performed; that is, the data is Fourier transformed in only two directions. For one full parallel-FFT cycle, four data transpose operations and four FFT operations are performed. Finally, the padding and truncating of data for 3/2 dealiasing is not performed, as this is not supported in P3DFFT.

There are several significant implementation differences between P3DFFT and our customized parallel FFT kernel. First, when a real data line of N elements (N even) is transformed, a complex data line of $(N/2)+1$ elements is stored which is equivalent to $N+2$ real elements. In some applications, including turbulence simulations, the last complex data element (the Nyquist mode) is set to zero because this is a coefficient for a Fourier mode that is not carried in the Fourier representation of the solution. Our parallel FFT library, unlike P3DFFT, recognizes that the Nyquist mode is not necessary and does not store it or include it in transposes. Second, as stated in section 4.2, our FFT kernel is implemented with OpenMP threading for the FFTs and on-node data reordering. P3DFFT does not at present support any shared memory parallelism, such as OpenMP. Finally, our customized kernel is significantly more memory efficient. P3DFFT uses a buffer that is three times the size of the input arrays while our customized kernel only requires a buffer the size of input array.

Table 6: Strong scaling benchmark of parallel FFT

		Elapsed Time (Efficiency)		
System	Cores	P3DFFT	Customized	**Ratio**
Mira^1	128	11.5 (100.%)	5.38 (100.%)	**2.14**
	256	5.88 (98.0%)	2.78 (96.8%)	**2.12**
	512	2.95 (97.7%)	1.18 (115.%)	**2.51**
	1,024	1.46 (98.6%)	.580 (116.%)	**2.52**
	2,048	.724 (99.5%)	.287 (117.%)	**2.52**
	4,096	.360 (100.%)	.139 (121.%)	**2.59**
	8,192	.179 (101.%)	.068 (123.%)	**2.61**
Mira^2	65,536	N/A	30.5 (100.%)	N/A
	131,072	N/A	16.2 (94.1%)	N/A
	262,144	12.4 (100.%)	8.51 (89.5%)	**1.45**
	393,216	10.1 (81.4%)	5.85 (86.9%)	**1.73**
	524,288	6.90 (89.6%)	4.04 (94.3%)	**1.71**
	786,432	4.55 (90.5%)	3.12 (81.5%)	**1.46**
Lonestar	12	N/A	6.00 (100.%)	N/A
	24	2.67 (100.%)	3.63 (82.7%)	**.735**
	48	1.57 (84.7%)	2.13 (70.4%)	**.739**
	96	.873 (76.4%)	1.12 (66.8%)	**.777**
	192	.547 (60.9%)	.580 (64.7%)	**.943**
	384	.294 (56.6%)	.297 (63.2%)	**.992**
	768	.212 (39.2%)	.172 (54.5%)	**1.23**
	1,536	.193 (21.6%)	.111 (42.2%)	**1.74**
Stampede	16	N/A	6.88 (100.%)	N/A
	32	N/A	4.42 (77.8%)	N/A
	64	2.16 (100.%)	2.51 (68.5%)	**.860**
	128	1.32 (81.9%)	1.39 (61.5%)	**.942**
	256	.676 (79.8%)	.718 (59.8%)	**.942**
	512	.421 (64.0%)	.377 (56.9%)	**1.12**
	1,024	.296 (45.6%)	.199 (53.9%)	**1.48**
	2,048	.201 (33.5%)	.113 (47.6%)	**1.78**
	4,096	.194 (17.4%)	.0636 (42.2%)	**3.05**

* "N/A" denotes inadequate memory
** N_x/N_y=N_z: Mira^1(2,048/1,024), Mira^2(18,432/12,288), Lonestar(768/768), Stampede(1,024/1,024)

As a result of these implementation differences, our parallel FFT kernel has significantly shorter execution times and better parallel efficiency than P3DFFT, as shown in Table 6. In this table, parallel efficiencies are calculated by normalizing the execution times by the time using the smallest number of cores in each case. P3DFFT is more efficient when run on a relatively small number of cores, while our customized kernel becomes significantly more efficient on large numbers of cores. In some cases, the customized kernel shows super-scaling. We conjecture that this is a result of the on-node data reordering process becoming more cache efficient higher core counts, because the dimensions of the array being transposed is smaller.

5. PERFORMANCE FOR A TIMESTEP

As a final test of the performance characteristics of the code, the performance on a complete third-order Runge-Kutta timestep was measured on the four benchmark platforms listed before. The grid sizes used for the strong scaling and weak scaling benchmarks are provided in Tables 7 and 8, respectively. The results of these benchmarks appear in Tables 9 and 10.

Table 7: Test cases for strong scaling benchmarks of a timestep

System	N_x	N_y	N_z	Degree of freedom
Mira	18,432	1,536	12,288	$696. \times 10^9$
Lonestar	1,024	384	1,536	1.21×10^9
Stampede	2,048	512	4,096	8.59×10^9
Blue Waters	2,048	1,024	2,048	8.59×10^9

Table 8: Test cases for weak scaling benchmarks of one full timestep

System	N_x	N_y	N_z
Mira	4,608/9,216/18,432/ 27,648/36,864/55,296	1,536	12,288
Lonestar	512/1,024/2,048/4,096	384	1,536
Stampede	512/1,024/2,048/4,096	512	4,096
Blue Waters	1,024/2,048/4,096/8,192	1,024	2,048

5.1 Strong Scaling Benchmark

On Mira, two execution cases were tested. There are sixteen physical cores per node and each core supports four hardware threads. For the case denoted as "MPI", sixteen MPI tasks were assigned per node and each core (task) was hardware threaded with four OpenMP threads for FFT and the time-advance kernel. For the case denoted as "Hybrid", only one MPI task was assigned per node and sixteen OpenMP threads were employed for data reordering and sixty four threads were used for the FFTs and time-advance kernel, exercising four hardware threads per core. In "MPI" cases, every section of the code shows excellent strong scaling over the range of benchmark cases. Even with 786K cores, the strong scaling parallel efficiency is 97% relative to performance on 131K cores. The results from the "Hybrid" benchmark are more complicated. Solution time is generally lower in the hybrid case than the MPI case, but the magnitude of this advantage decreases with increasing numbers of cores until with the largest core count, the execution times are essentially equivalent. The result is that the measured scaling efficiency is reduced to 80% for the largest core count. As is clear in the tables, the efficiency degradation in the hybrid case is primarily due to the transpose. When using one MPI task per node, the communication overhead is smaller resulting in better performance than assigning MPI tasks to each core. Apparently, when using large numbers of nodes, the total number of MPI tasks is large enough to cause communcation contention even with just one MPI task per node.

On Lonestar and Stampede, the on-node computation kernels scale almost perfectly, as does the transpose on Lonestar. However, on Stampede, the performance of the data transpose degrades with large core-counts. On Blue waters, the on-node performance generally scales well, similar to the other systems, while the cost of the transpose scales very poorly, with parallel efficiency degrading to 24% over a factor of eight increase in cores. Communication was expected to account for a large fraction of the execution time on Blue Waters because of the relative speed of the nodes and the communication network. However, this serious degradation in efficiency was not expected. One consequence is that communication accounts for a fraction of the execution time that increases from 80% at 2048 cores to 93% at 16,384 cores.

5.2 Weak Scaling Benchmark

The weak scaling benchmarks demonstrate generally similar behaviour as the strong scaling benchmarks, with the exception of the FFTs. The weak scaling benchmarks were performed by increased the data length in the streamwise (x) direction. It appears that as the size of the grid in the x direction increases, the degree to which the calculations done in the x-pencil configuration can remain in cache is degraded resulting in performance degradation. However, even in the absence of this, the FFTs cannot scale perfectly because the number of floating point operations scale like $O(N \log N)$. As with the strong scaling benchmarks, the primary reason for degradation in overall parallel efficiency is the transpose. However, there is significant degradation with problem size/core count on Mira for both MPI and Hybrid, which was not the case for the strong scaling benchmark.

5.3 MPI vs Hybrid on Mira

Table 11 shows a comparison of different choices of parallelism that were tested on Mira. There is no significant performance difference in on-node computation as shown in Tables 9 and 10. Rather, the performance of the global data transposes dominates the performance difference.

It is interesting to note that for these test cases, our code is faster using a hybrid MPI/OpenMP parallelism model. Using only MPI results in sixteen times more MPI tasks that issue 256 times more messages that are 256 times smaller. The latency and contention caused by so many more messages degraded data transpose performance. The fact that at the largest core count the MPI and hybrid performance are the same suggests that increasing the number of cores in the hybrid case finally saturate the interconnect in the same way that the extra MPI tasks did in the MPI case.

The channel code running on all 48 racks of Mira (786k nodes) on the strong scaling benchmark problem attained an aggregate compute speed of 271 Tflops, which is about 2.7% of the hardware theoretical peak. This modest flop rate compared to the maximum possible is due in part to the transpose required in the algorithm. Considering only the on-node computation, the aggregate flop rate is about 906 Tflops, or approximately 9.0% of peak. As shown in Table 2, the on-node compute speed is limited by memory bandwidth. Hence, the scarce resources on Mira for the turbulence DNS algorithms used here are inter-node communication capacity and memory bandwidth, and our optimizations have thus focused on managing these resources.

Table 9: Strong scaling benchmarks of a timestep

		Time(sec) / Efficiency			
System	Cores	Transpose	FFT	N-S time advance	**Total**
Mira (MPI)	131,072	26.9 / 100. %	7.32 / 100. %	6.98 / 100. %	**41.2 / 100. %**
	262,144	13.6 / 98.6 %	4.02 / 91.1 %	3.44 / 101. %	**21.1 / 97.6 %**
	393,216	8.92 / 100. %	2.61 / 93.6 %	2.28 / 102. %	**13.8 / 99.3 %**
	524,288	6.81 / 98.6 %	2.09 / 87.7 %	1.75 / 99.9 %	**10.6 / 96.7 %**
	786,432	4.50 / 99.6 %	1.36 / 89.6 %	1.21 / 96.5 %	**7.06 / 97.1 %**
Mira (Hybrid)	65,536	39.8 / 100. %	13.8 / 100. %	13.6 / 100. %	**67.2 / 100. %**
	131,072	20.9 / 95.4 %	7.03 / 98.2 %	6.76 / 101. %	**34.7 / 97.0 %**
	262,144	11.8 / 84.5 %	3.61 / 95.6 %	3.34 / 102. %	**18.7 / 89.7 %**
	393,216	8.83 / 75.1 %	2.43 / 94.8 %	2.22 / 102. %	**13.5 / 83.1 %**
	524,288	5.73 / 86.7 %	1.89 / 91.5 %	1.67 / 102. %	**9.29 / 90.4 %**
	786,432	4.70 / 70.5 %	1.27 / 90.5 %	1.11 / 102. %	**7.09 / 79.0 %**
Lonestar	192	9.53 / 100. %	2.06 / 100. %	3.00 / 100. %	**14.6 / 100. %**
	384	4.70 / 101. %	1.04 / 98.7 %	1.50 / 100. %	**7.24 / 101. %**
	768	2.38 / 100. %	0.51 / 100. %	0.75 / 99.4 %	**3.65 / 100. %**
	1,536	1.29 / 92.5 %	0.26 / 97.9 %	0.37 / 99.9 %	**1.93 / 94.6 %**
Stampede	512	18.9 / 100. %	5.30 / 100. %	6.85 / 100. %	**31.0 / 100. %**
	1,024	10.9 / 86.7 %	2.68 / 98.8 %	3.40 / 101. %	**17.0 / 91.4 %**
	2,048	7.60 / 62.1 %	1.36 / 97.5 %	1.72 / 99.3 %	**10.7 / 72.6 %**
	4,096	3.83 / 61.5 %	0.67 / 98.2 %	0.84 / 102. %	**5.35 / 72.5 %**
Blue Waters	2,048	17.9 / 100. %	2.73 / 100. %	3.53 / 100. %	**24.2 / 100. %**
	4,096	16.2 / 55.1 %	1.37 / 99.4 %	1.76 / 100. %	**19.4 / 62.3 %**
	8,192	16.2 / 27.7 %	.650 / 105. %	.880 / 100. %	**17.7 / 34.1 %**
	16,384	9.88 / 22.7 %	.356 / 95.8 %	.440 / 100. %	**10.7 / 28.3 %**

Table 10: Weak scaling benchmarks of a timestep

		Time(sec) / Efficiency			
System	Cores	Transpose	FFT	N-S time advance	**Total**
Mira (MPI)	65,536	9.87 / 100. %	3.30 / 100. %	3.46 / 100. %	**16.6 / 100. %**
	131,072	13.6 / 72.6 %	3.52 / 93.8 %	3.45 / 100. %	**20.6 / 80.9 %**
	262,144	13.6 / 72.5 %	4.02 / 82.1 %	3.44 / 101. %	**21.1 / 78.9 %**
	393,216	16.0 / 61.6 %	4.41 / 74.8 %	3.43 / 101. %	**23.9 / 69.7 %**
	524,288	13.5 / 73.2 %	5.50 / 59.9 %	3.48 / 99.5 %	**22.5 / 74.0 %**
	786,432	13.7 / 72.2 %	7.28 / 45.3 %	3.50 / 99.1 %	**24.5 / 68.0 %**
Mira (Hybrid)	65,536	9.83 / 100. %	3.17 / 100. %	3.34 / 100. %	**16.3 / 100. %**
	131,072	10.3 / 95.2 %	3.36 / 94.3 %	3.34 / 99.9 %	**17.0 / 95.9 %**
	262,144	11.8 / 83.5 %	3.61 / 87.7 %	3.34 / 100. %	**18.7 / 87.2 %**
	393,216	13.4 / 73.5 %	4.14 / 76.6 %	3.34 / 100. %	**20.8 / 78.4 %**
	524,288	11.8 / 83.6 %	5.08 / 62.4 %	3.35 / 99.6 %	**20.2 / 80.9 %**
	786,432	14.5 / 67.6 %	7.60 / 41.7 %	3.34 / 99.9 %	**25.5 / 64.1 %**
Lonestar	192	4.73 / 100. %	1.00 / 100. %	1.51 / 100. %	**7.24 / 100. %**
	384	4.70 / 101. %	1.04 / 96.2 %	1.50 / 101. %	**7.24 / 100. %**
	768	4.70 / 101. %	1.17 / 85.6 %	1.50 / 101. %	**7.37 / 98.1 %**
	1,536	5.01 / 94.4 %	1.31 / 76.9 %	1.50 / 101. %	**7.81 / 92.7 %**
Stampede	512	4.85 / 100. %	1.21 / 100. %	1.71 / 100. %	**7.77 / 100. %**
	1,024	5.66 / 85.8 %	1.24 / 97.5 %	1.75 / 97.9 %	**8.65 / 89.9 %**
	2,048	6.78 / 71.5 %	1.34 / 90.1 %	1.73 / 98.8 %	**9.86 / 78.9 %**
	4,096	7.11 / 68.3 %	1.47 / 82.6 %	1.73 / 98.7 %	**10.3 / 75.4 %**
Blue Waters	2,048	11.1 / 100. %	1.26 / 100. %	1.76 / 100. %	**14.1 / 100. %**
	4,096	16.2 / 68.5 %	1.37 / 91.9 %	1.76 / 100. %	**19.4 / 72.9 %**
	8,192	20.44 / 54.4 %	1.49 / 84.6 %	1.76 / 100. %	**23.7 / 59.6 %**
	16,384	25.66 / 43.2 %	1.70 / 74.1 %	1.76 / 100. %	**29.1 / 48.5 %**

Table 11: MPI vs Hybrid on Mira

	Strong Scaling			Weak Scaling		
Cores	MPI (sec)	Hybrid (sec)	**Ratio**	MPI (sec)	Hybrid (sec)	**Ratio**
65,536	N/A	67.2	N/A	16.6	16.3	**1.02**
132,072	41.2	34.7	**1.19**	20.6	17.0	**1.21**
262,144	21.1	18.7	**1.13**	21.1	18.7	**1.13**
393,216	13.8	13.5	**1.02**	23.9	20.8	**1.15**
514,288	10.6	9.29	**1.14**	22.5	20.2	**1.11**
786,432	7.06	7.09	**1.00**	24.5	25.5	**.96**

6. SIMULATION RESULTS

The code discussed here was developed and optimized to simulate a turbulent channel flow at friction

Reynolds number of 5000 to study the high-Reynolds number behavior of turbulence in the overlap region between the flow near the wall whose dynamics is dominated by the wall, and the outer flow that is only weakly influenced by the wall. A single simulation is currently being run on 32 racks (524,288 cores) on the BG/Q system, Mira, at ALCF. A Fourier-B-spline representation with 10,240 modes in the x direction, 1,536 in the y direction and 7,680 in the z direction is being used to represent the velocity, for a total of 242 billion degrees of freedom. This is significantly larger (by a factor of 15) than that of the previously largest channel simulation conducted by Hoyas *et al.*[10] in 2006 at $Re_\tau = 2000$. The two and a half times larger Reynolds number will be high enough for unambiguous scale separation between the wall and outer scales, which will enable definitive study of the overlap region discussed above.

Currently, the most complete sequence of cases for turbulent channels are in the friction-Reynolds-number range $Re_\tau = 180 - 2000$, which constitute the standard reference data set in the field[19, 6, 10]. Supplementing these data with $Re_\tau = 5200$ will establish a reference data set that will remain useful for the turbulence research community for many years.

In the study of turbulence, many (most) of the quantities that one wants to study are statistical. The channel flow is a good DNS vehicle for computing turbulent statistics because the flow is statistically stationary, allowing statistics to be computed as time averages. Our channel flow simulation must therefore be run for long enough for the turbulence to become statistically stationary, and then long enough to compute converged statistics. The current simulation will be run a total of about 13 flow-throughs, where a flow through is the average amount of time required for a fluid particle to traverse the simulation domain (in x). Each flow through requires approximately 50,000 time steps to simulate, so overall the simulation will require 650,000 time steps or 260 million core hours on Mira.

At the time of this writing, the simulation has completed approximately 8 flow-throughs. This includes the transient period during which the simulation evolves to a statistically stationary state. Currently, about 4.5 flow-throughs have been completed in statistically steady state, and the preliminary results of these statistics will be the subject of an upcoming publication. As an example, the mean velocity and variance of the velocity fluctuations and turbulent shear stress are plotted in figures 5 and 6. The mean velocity plotted in a semi-log plot displays the famous logarithmic velocity profile in the overlap region.

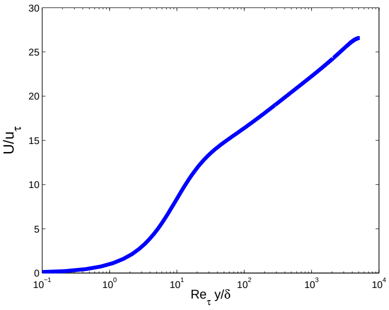

Figure 5: Mean velocity profile, velocity normalized by the friction velocity and y normalized by the channel half-width and friction Reynolds number.

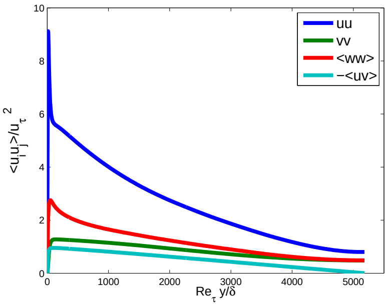

Figure 6: Variances of the turbulent velocity fluctuations ($\langle uu \rangle$, etc.) and the turbulent shear stress ($-\langle uv \rangle$).

The interaction between near-wall turbulence and the outer flow turbulence in the overlap region is critical to understanding high Reynolds number turbulent wall layers. To investigate this interaction, it is necessary that the Reynolds number be sufficiently high so that there is a substantial disparity in scale between the inner and outer flows. There will then be a reasonable expectation that the results can be extrapolated to much higher Reynolds numbers[12]. The disparity in scales between the near-wall turbulence and the outer layer turbulence is displayed in the visualization of the streamwise velocity and spanwise vorticity shown in figures 7 and 8.

7. CONCLUSIONS

According to the Top500 list as of April, 2013[18], Mira (Bluegene/Q) has the second largest core count in the world. Also, Sequoia(Bluegene/Q), which has the largest number of cores in the world, has the same system architecture as Mira. Performance characteristics of large core-count HPC systems like Mira are indicative of what can be expected in next generation HPC platforms. We therefore expect that

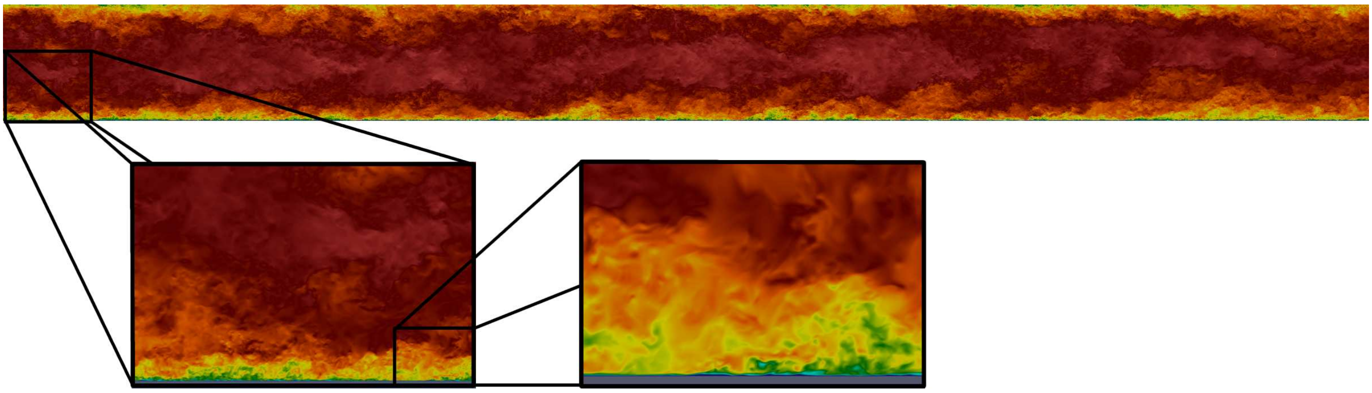

Figure 7: Instantaneous streamwise velocity component over the entire streamwise length of the simulated channel. Zooming in on the flow highlights the multi-scale nature of the turbulence.

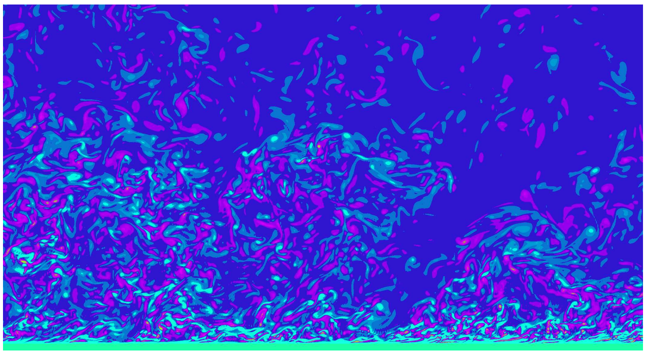

Figure 8: The instantaneous z-component of vorticity near the wall.

our experiences with performance optimization and characterization on Mira will be relevant as HPC systems continue to evolve.

Our code's performance was highly dependent on the characteristics of the communication network. The importance of the interconnect is expected to be increasingly significant as the number of cores increases. The results of the current benchmarks show that the 5D torus interconnect in Mira performed very well on massive data communication, but even with this good communication capability, the results show that the network becomes saturated. It is also clear that for algorithms that require global communication like that employed here, it is critical that interconnect speed improve with node speed.

Limitations of the communication interconnect can be ameliorated somewhat using hybrid parallelism. The necessary condition to make a hybrid scheme competitive compared to MPI only is, of course, good data-parallel performance. Our turbulent direct numerical simulation code had very good threaded parallel performance for on-node computation on all platforms. Particularly impressive was the parallel efficiency of the linear algebra for the Navier-Stokes time-advance. It was shown that using a hybrid scheme can be beneficial when the number of cores per node is large so that core-level MPI tasks would saturate the network. This seems significant for future HPC systems.

While on-node threaded parallel efficiency was very good, the actual core-level performance measured in flops was rather modest, at 1.16 Gflops or 9% of theoretical peak. However, the statistics in Table 2 show that on Mira, the aggregate DDR data traffic in this code is very close to its theoretical peak. The limiting on-node hardware resource in these calculations is thus memory bandwidth, so our most important optimization consideration for on-node performance was reducing memory traffic. This is expected to be an increasingly important issue on the road to exascale.

Finally, the turbulent channel DNS code discussed here is currently in production performing a turbulence simulation of unprecedented magnitude. The simulation results are expected to enable important advances in the science and engineering of wall-bounded turbulent flows.

8. ACKNOWLEDGMENTS

The work described here was enabled by a generous grant of computer time from Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. The financial support of the National Science Foundation under PetaApps grant OCI-0749223 and PRAC Grant 0832634 has also been critical, and is gratefully acknowledged. In addition computing resources for benchmarking were provided by the Texas Advanced Computing Center (TACC) at The University of Texas at Austin and the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

9. REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] C. Canuto. *Spectral Methods in Fluid Dynamics*. Springer, 1988.

[3] A. W. Cook and J. J. Riley. Direct numerical

simulation of a turbulent reactive plume on a parallel computer. *J. Comp. Physics*, 129:263–283, 1996.

[4] I. Corp. Math kernel library. http://software.intel.com/en-us/intel-mkl, 2012.

[5] C. DeBoor. *A practical guide to splines*. Springer, Aug. 1978.

[6] J. C. del Alamo, J. Jiménez, P. Zandonade, and R. D. Moser. Scaling of the energy spectra of turbulent channels. *J. Fluid Mech.*, 117:133–166, 2004.

[7] D. A. Donzis, P. K. Yeung, and D. Pekurovsky. Turbulence simulation on $O(10^4)$ processors. *Teragrid'08*, 2008.

[8] D. A. Donzis, P. K. Yeung, and K. R. Sreenivasan. Dissipation and enstrophy in isotropic turbulence: Resolution effects and scaling in direct numerical simulations. *Physics of Fluids*, 20(4):16, 2008.

[9] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[10] S. Hoyas and J. Jiménez. Scaling of the velocity fluctuations in turbulent channels up to re[sub tau] = 2003. *Physics of Fluids*, 18(1):011702, Jan. 2006.

[11] IBM. Engineering and scientific subroutine library (essl) reference guide. http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp, 2013.

[12] J. Jiménez and R. D. Moser. What are we learning from simulating wall turbulence? *Phil. Trans. Royal Soc. A*, 365:715–732, 2007.

[13] J. Jimenez and A. Pinelli. The autonomous cycle of near-wall turbulence. *J. Fluid Mech.*, 389, 1999.

[14] T. Kaneda, Y; Ishihara and M. Yokokawa. Energy dissipation rate and energy spectrum in high resolution direct numerical simulations of turbulence in a periodic box. *Physics of Fluids*, 15(2), 2003.

[15] J. Kim. Blue waters : Sustained petascale computing. 2010.

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

[17] W. Y. Kwok, R. D. Moser, and J. Jiménez. A critical evaluation of the resolution properties of B-Spline and compact finite difference methods. *Journal of Computational Physics*, 174(2):510–551, Dec. 2001.

[18] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. The Top500 List. http://www.top500.org/, 2012.

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

[20] S. A. Orszag. On the elimination of aliasing in finite-difference schemes by filtering high-wavenumber components. *Journal of the Atmospheric Sciences*, 28(6):1074, 1971.

[21] D. Pekurovsky. P3DFFT: User guide. http://code.google.com/p/p3dfft/, 2008.

[22] S. B. Pope. *Turbulent Flows*. Cambridge University Press, Oct. 2000.

[23] S. P. R., R. D. Moser, and M. M. Rogers. Spectral methods for the Navier-Stokes equations with one infinite and two periodic directions. *J. Comput. Phys.*, 96:297–324, 1991.

[24] K. W. Schulz, R. Ulerich, N. Malaya, P. T. Bauman, R. Stogner, and C. Simmons. Early experiences porting scientific applications to the Many Integrated Core (MIC) platform. In *TACC-Intel Highly Parallel Computing Symposium*, Austin, Texas, April 2012.

[25] J. M. Tomoaki Ishiyama, Keigo Nitadori. 4.45 Pflops Astrophysical N-Body Simulation on K Computer - The Gravitational Trillion-Body Problem. *ACM Gordon Bell Prize 2012*, 2012.

[26] U.S. Energy Information Administration (EIA). INTERNATIONAL ENERGY OUTLOOK 2011.

[27] B. Walkup. Application performance characterization and analysis on blue gene/q. In *Proc. ACM/IEEE Conf. Supercomputing (SC)*, November.

[28] P. K. Yeung, S. B. Pope, and B. L. Sawford. Reynolds number dependence of Lagrangian statistics in large numerical simulations of isotropic turbulence. *Journal of Turbulence*, 7(58):1–12, 2006.