SCI INSTITUTE
TECHNICAL REPORT

Parallelization and Scalability of a Spectral Element Solver

Curtis W. Hamman, Robert M. Kirby, and Martin Berzins*
*Corresponding Author, Email: kirby@sci.utah.edu

UUSCI-2005-011

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

December 12, 2005
Revised: November 9, 2006

Abstract:

Direct numerical simulation (DNS) of turbulent flows is widely recognized to demand fine spatial meshes, small timesteps, and very long run-times to properly resolve the flow field. To overcome these limitations, most DNS is performed on supercomputing machines. With the rapid development of terascale (and, eventually, petascale) computing on thousands of processors, it has become imperative to consider the development of DNS algorithms and parallelization methods that are capable of fully exploiting these massively parallel machines. A highly parallelizable algorithm for the simulation of turbulent channel flow that allows for efficient scaling on several thousand processors is presented. A model that accurately predicts the performance of the algorithm is developed and compared with experimental data. The results demonstrate that the proposed numerical algorithm is capable of scaling well on petascale computing machines and thus will allow for the development and analysis of high Reynolds number channel flows.

THE UNIVERSITY OF UTAH

1 Introduction

Tremendous increases in parallel computing power have made it possible to consider solution techniques that were computationally intractable only a short time ago. One such technique is the direct numerical simulation (DNS) of the incompressible Navier-Stokes equations, a set of three-dimensional, time-dependent, non-linear partial differential equations. DNS is used to solve the incompressible Navier-Stokes by resolving all scales from first principles without the use of additional modeling assumptions. However, the computational requirements for DNS of flow in a channel (or nozzle) remain formidable at even the most moderate of Reynolds numbers.

The spatial discretization must encompass the smallest, physically significant length scales while simultaneously resolving scales that are typically several orders of magnitude larger. The time-advancement of the convective and viscous terms involves the solution of stiff differential equations. Furthermore, very small timesteps and long integration times are often necessary to ensure time accuracy and eliminate the influence of non-physical initial conditions. The underlying physics dictate that scales on the order of the Kolmogorov length scale need be resolved, which implies that the number of degrees of freedom $N_x N_y N_z$ for a uniform grid increases as $Re_\tau^{9/4}$ where Re_τ is the Reynolds number based on the wall shear stress velocity (21). The number of timesteps necessary to properly resolve a fully turbulent, three-dimensional flow based on the Kolmogorov time scale is $N_t \geq O(Re_\tau^{1/2})$ so that the total time-advancement cost is at least $N_x N_y N_z N_t = O(Re_\tau^{11/4})$. The quadratic non-linearities of the Navier-Stokes equations require that steps to eliminate the effects of aliasing error, such as the Orszag Two-Thirds Rule (4), must also be implemented, further increasing the memory and computational requirements. Due to these constraints of time and resources, most DNS simulations have remained restricted to comparatively low Reynolds numbers.

To mitigate these facts, the use of Reynolds-Averaged Navier-Stokes (RANS), Detached Eddy Simulation (DES), and Large Eddy Simulation (LES) methods have grown in popularity (21). In these techniques the statistical development of the flow is sought through the partial modeling of the turbulent energy spectrum. This is in contrast to DNS, which, in principle, seeks a numerically-accurate, instantaneous solution of the governing equations of motion directly from first principles. Moreover, RANS, DES, and LES methods encounter the well-known closure problem, and thus, must devise approximations of unknown correlations in terms of flow properties that are known in the derived equations. DNS requires no such modeling at the expense of a far more computationally intensive simulation.

One of the first direct numerical simulations of fully-developed turbulent channel flow was performed by Kim, Moin and Moser (16). This formulation has effectively become the *de facto* standard in the DNS of turbulent channel flows. Most subsequent efforts have used similar parallelization methods such that the data in the wall-normal direction is stored locally on a single processor for the solution of the Helmholtz system while the xz-planes are redistributed amongst processors during operations involving Fourier Transforms thereby necessitating the use of many communication intensive transpose operations (14; 12; 27).

Following the efforts of Moser *et al.*, there have been a number of papers published based on making use of parallel computing techniques for the DNS of fully-developed turbulent channel flow. For example, Hoyas and Jiménez have resolved simulations up to $Re_\tau = 2003$ using a form of large-scale parallelization (11). Many other efforts (31) have attempted to develop a scalable DNS algorithm, but massive parallelism in this context remains a challenge. In the move to massive parallelism, it is natural to consider approaches based on parallelism in the wall-normal as well as the Fourier directions, concurrently. The approach presented herein parallelizes the Fast Fourier Transforms (FFTs) but also exploits the strong locality of one-dimensional spectral elements for

further concurrency in the wall-normal direction to attain a highly robust parallelization methodology. Accordingly, the numerical procedure is also altered by using a third-order stiffly stable time integration scheme (SE3/SI3) (15).

Given this context, it is beneficial to remain cognizant of the principal physical and computational motivations underlying DNS research (18). DNS solutions are often used to further refine LES subgrid models (22; 2; 24) and develop new turbulence models (20; 26). The field of turbulence control where simulation parameters are modified systematically to study their effect on quantities such as drag and wall shear stress has increasingly relied on the efficacy of DNS (32; 5). The computation of vorticity, pressure-strain correlation tensor, and local topologies of the flow are exceedingly difficult by empirical means while a properly resolved DNS allows for careful analysis of those quantities as well as calibration of experimental devices within the bounds of the underlying numerical and physical assumptions. Studies of topological or spatially local characteristics of the flow (3; 34; 1) benefit from the proper computation of the small-scale interactions provided by DNS. Turbulence scaling laws, statistics, and boundary layer analysis are often improved upon by the results of high Reynolds number DNS flow fields (29; 33; 25; 19).

Thus, the direct numerical simulation of turbulent flows is a valuable resource with wide-ranging and complementary application for experimental, theoretical, and other computational models whose chief difficulty remains a problem of algorithm and software design. As the number of processors increases and the hardware and software infrastructures evolve, software engineering trade-off decisions must be reevaluated (for examples of such a reevaluation of design decisions see (7; 8; 9)). What once were limiting factors which impacted algorithm and software design decisions may no longer be the current bottleneck. Hence, the challenge is then to assess what design decisions need to be reevaluated so as to develop novel algorithms capable of reaching higher and higher Reynolds numbers. We anticipate that new algorithms will meet this challenge by advantageously employing large-scale parallelism on several thousand processors thereby cultivating the next generation of direct numerical simulations.

In discussing the approach considered herein, it is worth noting that most high-end CFD simulations utilize approximately forty seconds of wall clock time per timestep independent of the number of processors (16; 30). This is often the benchmark from which simulation time on supercomputing machines is measured to predict how many days, months, or years are required to complete the simulation. Finally, given future trends in parallel computing to incorporate larger and larger processor counts as parallel computing moves from teraflop towards petaflop performance, the need for algorithms that can fully employ such machines is paramount.

This paper is organized as follows. We discuss the numerical discretization and algorithm in Section 2. The parallel decomposition of the algorithm is detailed in Section 3. A performance model is developed to predict the scaling behavior of the algorithm in Section 4. The performance model is then compared with the empirical scalability data given in Section 5.

2 Numerical Method

For an incompressible, isotropic, isothermal, and Newtonian fluid flow, the governing equations of motion are given by the Navier-Stokes equations and the equation of mass continuity. The Navier-Stokes equations can be written as follows,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{\omega} \times \boldsymbol{u} + \frac{1}{2}\nabla u^2 = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u} + \boldsymbol{F} \ , \tag{1a}$$

where $\boldsymbol{u}(\boldsymbol{x},t)$ is the velocity field, $\boldsymbol{\omega}(\boldsymbol{x},t) = \nabla \times \boldsymbol{u}$ is the vorticity field, $p(\boldsymbol{x},t)$ is the pressure fluctuation field, $\boldsymbol{F}(\boldsymbol{x},t)$ is an appropriate forcing term (such as the mean streamwise pressure gradient), ρ is the (constant) density, and ν is the (constant) kinematic viscosity of the fluid. The equation of mass continuity assuming incompressibility is reduced to the condition

$$\nabla \cdot \boldsymbol{u} = 0 \ . \tag{1b}$$

For computational purposes, we consider only the rotational form of the non-linear, convective acceleration term $(\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} = \boldsymbol{\omega} \times \boldsymbol{u} + \frac{1}{2}\nabla u^2$ as both linear momentum and kinetic energy are conserved in the inviscid limit and fewer derivatives need be computed. However, as a result of this choice, a 2/3-dealiasing method in the streamwise and spanwise directions with 3/2-over-integration in the wall-normal direction was used to eliminate aliasing error (4). The numerical properties of one-dimensional spectral element methods are used to yield comparatively superior performance.

Following Karniadakis *et al.* (13; 15), the Navier-Stokes equations can be discretized in time using a three-step splitting (SE3/SI3) scheme with three distinct stages that compute the non-linear terms, perform a divergence-free pressure projection, and implement a viscous correction to determine the updated velocity field. During each timestep, the three stages are executed sequentially with different parallelization strategies. Stage 1 involves the integration of the non-linear terms. Stage 2 enforces a divergence-free projection of the velocity field and solves a pressure Poisson equation. During Stage 3, a Helmholtz problem derived from the viscous contributions is solved for the updated velocity components.

The streamwise direction corresponds to the x-coordinate axis, the spanwise direction the z-axis, and the wall-normal direction the y-axis. Hence, the xz-plane is periodic, homogeneous, and employs FFTs to compute the derivatives in transformed space. The wall-normal direction employs a spectral element discretization with Gauss-Lobatto-Legendre collocation and quadrature for physical space differentiation and integration to solve the pressure and viscous Helmholtz problems as required in Stage 2 and Stage 3. This mathematical methodology has been successfully used previously for the simulation of flow in a channel (31; 32); the goal in this work is to consider a different algorithmic and implementation choice for parallelization than previously used while maintaining the same tested mathematical framework.

2.1 Stage 1

In the first stage, the non-linear terms are computed and extrapolated with a consistent order, J_e, in time with timestep Δt. The first stage is given by the integration of the convective terms

$$\frac{\hat{\boldsymbol{u}} - \sum_{q=0}^{J_e-1} \alpha_q \boldsymbol{u}^{n-q}}{\Delta t} = -\sum_{q=0}^{J_e-1} \beta_q [(\boldsymbol{\omega} \times \boldsymbol{u} + \frac{1}{2}\nabla u^2) - \boldsymbol{F}]^{n-q} \ , \tag{2}$$

where the integration weights, α_q, β_q and γ_0, can be found in (15).

The non-linear terms are computed in rotational form, i.e. $(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = \boldsymbol{\omega} \times \boldsymbol{u} + \frac{1}{2}\nabla u^2$, using 2/3-dealiasing in the Fourier directions (xz-plane) with 3/2-over-integration in the wall-normal direction to properly compute the quadratic nonlinearities. Notice that this stage is embarrassingly parallel with respect to parallelization in the wall-normal direction as all y-direction derivatives are computed on an elemental basis that is local to each processor (we assume that elements are partitioned across processors; an element is not allowed to be split across parallelization domains). The only communication overhead occurs when processors are split along the streamwise direction, which is diminished due to the resultant concurrency of the wall-normal parallelization. The mean streamwise pressure gradient is included in the forcing term at this stage.

2.2 Stage 2

The second-stage approximation to $\boldsymbol{u}$, $\hat{\hat{\boldsymbol{u}}}$, can be computed as

$$\frac{\hat{\hat{\boldsymbol{u}}} - \hat{\boldsymbol{u}}}{\Delta t} = -\nabla p^{n+1/2} \ , \tag{3}$$

which, assuming $\hat{\hat{\boldsymbol{u}}}$ is divergence-free, can be recast as

$$\nabla^2 p^{n+1/2} = \nabla \cdot \left(\frac{\hat{\boldsymbol{u}}}{\Delta t} \right) \ , \tag{4}$$

which can be solved with the following Neumann boundary conditions

$$\frac{\partial p^{n+1/2}}{\partial n} = -\nu \hat{\mathbf{n}} \cdot \sum_{q=0}^{J_e - 1} \beta_q (\nabla \times \boldsymbol{\omega})^{n-q} \quad \text{on} \quad \partial\Omega \ , \tag{5}$$

where the unit normal $\hat{\mathbf{n}}$ is strictly directed along the y-axis for a planar channel flow (15). Once $p^{n+1/2}$ is found, substitution into Equation 3 immediately yields the second stage approximation to the velocity field.

2.3 Stage 3

In the third stage, the updated velocity field, $\boldsymbol{u}^{n+1}$ is computed as

$$\frac{\gamma_0 \boldsymbol{u}^{n+1} - \hat{\hat{\boldsymbol{u}}}}{\Delta t} = \nu \nabla^2 \boldsymbol{u}^{n+1} \ , \tag{6}$$

which can be recast as

$$\left(\nabla^2 - \frac{\gamma_0}{\nu \Delta t} \right) \boldsymbol{u}^{n+1} = - \left(\frac{\hat{\hat{\boldsymbol{u}}}}{\nu \Delta t} \right) \ , \tag{7}$$

with appropriate boundary conditions (no-slip and stationary walls for the current simulations). Then, the three stage cycle is repeated with the updated velocity field to advance the solution in time.

2.4 Spectral element discretization

The use of a conformal spectral element grid in the wall-normal (y) direction permits both h-type elemental decomposition as well as p-type polynomial expansion modifications to gain further resolution and accuracy without sacrificing performance. For smooth functions, h-type refinement results in an algebraic decay of the numerical error while p-type refinement yields exponential decay of the numerical error. In particular, elements can be positioned arbitrarily close to the wall with a few large elements spanning the channel center so as to appropriately resolve all essential turbulent scales dependent on Reynolds number. The use of stretched grids to concentrate points where the highest frequency perturbations reside (typically, near the wall) is common practice in DNS (16). The simulation parameters α and β control the distribution of the element clustering and the amount of clustering performed at the wall boundaries, respectively. The algebraic mapping (23; 10) between the physical elemental boundaries and uniform mapping domain is given by

$$y = H \frac{(2\alpha + \beta) \left[(\beta + 1)/(\beta - 1) \right]^{(\eta - \alpha)/(1-\alpha)} + 2\alpha - \beta}{(2\alpha + 1) \{ 1 + \left[(\beta + 1)/(\beta - 1) \right]^{(\eta - \alpha)/(1-\alpha)} \}} \ , \tag{8}$$

where α is defined on $[0,1)$, β is defined on $(1,\infty)$, H is the channel height, and η is the mapping of each elemental boundary to a uniform grid defined on $[0,1]$. As α approaches zero (or one), the clustering is concentrated on the top (or bottom) wall, and, for $\alpha = 0.5$, the clustering is distributed equally between the top and bottom walls. As β approaches 1, more grid points are clustered near the wall boundaries while, as $\beta \to \infty$, the elements approach a uniform separation. For all simulations presented, $\alpha = 0.5$ and $\beta = 1.10$.

Each of the N_e spectral elements is prescribed a specific modal expansion basis polynomial degree P_e on the standard interval $\Omega_{st} = \{\xi : -1 \leq \xi \leq 1\}$ using the modified Jacobi polynomial basis given by the following expression:

$$\psi_p^e(\xi) = \begin{cases} \left(\frac{1-\xi}{2}\right) & p = 0 \ , \\ \left(\frac{1-\xi}{2}\right)\left(\frac{1-\xi}{2}\right) P_{p-1}^{1,1}(\xi) & 0 < p < P_e \ , \\ \left(\frac{1+\xi}{2}\right) & p = P_e \ , \end{cases} \tag{9}$$

where $P_j^{\alpha,\beta}(\xi)$ is the j^{th} Jacobi polynomial with $\alpha = \beta = 1$. The boundary modes ($p = 0$ and $p = P_e$) are responsible for coupling the adjacent elements. This allows for a combined *hp*-type spectral element discretization where the flow is divided into N_e macro-elements each discretized by a P_e-th order polynomial expansion basis. By varying N_e or increasing P_e, the algorithm will converge to the desired solution. To clarify the exposition, we will assume that the polynomial order P is constant for each element.

The standard element Ω_{st} is the mapping of the global coordinate y in terms of the local element e and local coordinate $\xi \in \Omega_{st}$ as

$$y = \chi^e(\xi) = \frac{(1-\xi)}{2} y_{e-1} + \frac{(1+\xi)}{2} y_e \ , \tag{10}$$

where the elemental boundary locations y_e and y_{e-1} for each element e are determined from Equation 8. Then, Q Gauss-Lobatto-Legendre quadrature points are distributed within each element, i.e. $Q = P + 2$. For over-integration of the non-linear terms, $Q_{nl} = (3/2)(P+2)$. The Jacobian mapping $d\chi^e/d\xi$ from Ω_{st} to the actual domain χ^e is then specified by

$$\frac{d\chi^e}{d\xi} = \frac{y_e - y_{e-1}}{2} \ . \tag{11}$$

The primary benefit of this particular discretization beyond the geometrical flexibility and high-order convergence rates afforded by *hp*-refinement is that the computations are localized at the elemental level thereby providing further concurrency through simultaneous utilization of the distinct parallel communication methods of the wall-normal and xz-plane decomposition.

3 Parallel Decomposition

The data decomposition with $P_y = 2$ wall-normal processors and $P_{xz} = 4$ streamwise/spanwise processors for a total of eight ($P_y \times P_{xz}$) processors is shown in Figure 1. By construction, P_y and P_{xz} are both positive integers. The communication connection schematic of the parallel decomposition is shown in Figure 2. The P_y processor groups denote the Helmholtz (or wall-normal) communications while the P_{xz} processor groups denote the FFT (or xz-plane) communication costs. Note that, after each FFT, the data ordering is left in a transposed state thereby saving an extra transposition of the data.

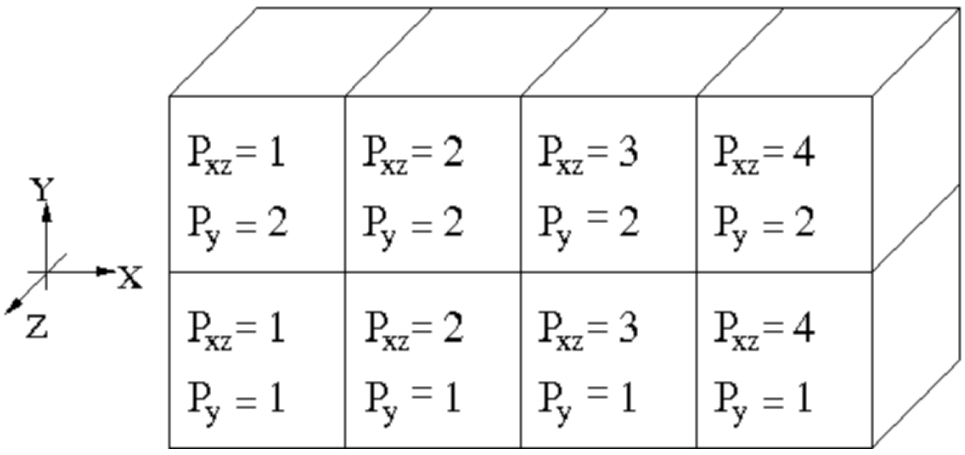


Figure 1: Example data decomposition among eight processors with $P_y = 2$ and $P_{xz} = 4$.

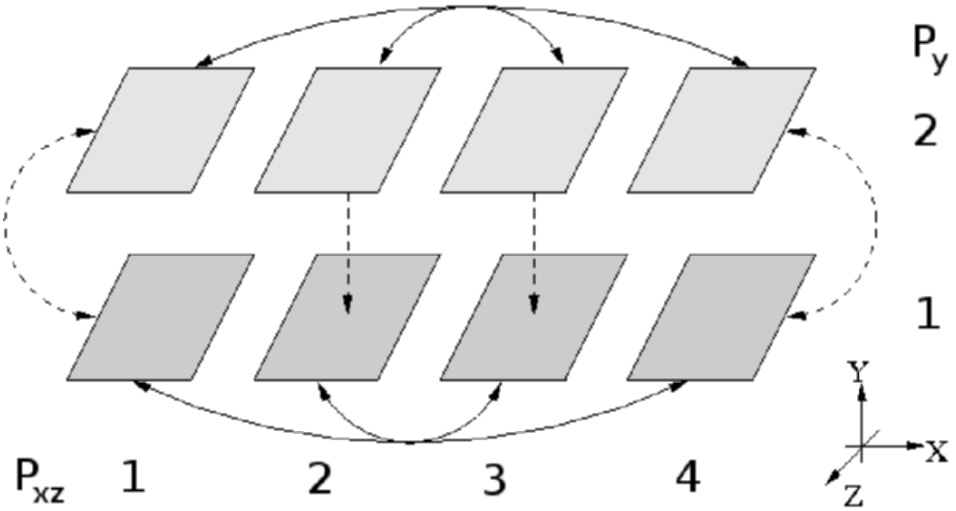


Figure 2: A schematic representation of the communication connection graph for the proposed algorithm and domain decomposition. Communication is performed in both the wall-normal and homogeneous directions with eight processors such that $P_y = 2$ and $P_{xz} = 4$ as in Figure 1. The solid lines denote collective communication between processors of the same xz-plane due to the MPI_Alltoall calls of the FFTs. The dashed lines denote the pairwise and MPI_Allreduce communication between processors of the same yz-plane from the Helmholtz solution process.

3.1 Helmholtz Solver

For Stage 2 and Stage 3, the local elemental Helmholtz equation being solved is of the form

$$\int_{\Omega_e} \left(\frac{\partial \psi_q^e}{\partial \xi} \frac{\partial \psi_p^e}{\partial \xi} + \lambda \psi_q^e \psi_p^e \right) \hat{u}_p^e \frac{d\chi^e}{d\xi} d\xi = \int_{\Omega_e} \psi_p^e f^e \frac{d\chi^e}{d\xi} d\xi \ , \tag{12}$$

where f represents the right-hand side of Equations 4 and 7, respectively, and the tildes denote Fourier transformed quantities along the xz-plane. The homogeneity of the flow along the streamwise and spanwise directions transforms derivatives with respect to x and z into algebraic expressions that may be absorbed into a single constant λ. For the above equation, $\lambda = r^2 m^2 + s^2 n^2$, where $r = 2\pi/L_x$, $s = 2\pi/L_z$, m is the streamwise wavenumber, and n is the spanwise wavenumber where we assume the Fourier transforms are unnormalized; if normalized transforms are used, $r = s = 1$. When $m = n = 0$ during Stage 2, the solution is set to zero since the spatially constant mean streamwise pressure gradient is included in the forcing term.

After application of the aforementioned boundary conditions and the global assembly process, the system of equations to be solved then reduces to

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b} \ , \tag{13}$$

where $\boldsymbol{M}$ is the global Helmholtz matrix, which consists of the global assembly of the local mass and Laplacian matrices of the spectral element discretization. The global Helmholtz matrix has a block-diagonal structure with overlap at the boundary modes as shown in Figure 3. The decomposition of the global Helmholtz matrix amongst four distinct processors is also shown in Figure 3. Observe that the only overlapping positions in the global Helmholtz matrix are the boundary-boundary modes formed by local contributions from adjacent processors. Due to the near-orthogonality of the one-dimensional spectral element expansion basis, the regions outside the block-diagonal structure are strictly zero, hence, those entries are ignored in the computation leading to an $O(PN)$ rather than $O(N^2)$ matrix-vector multiplication.

A diagonally preconditioned conjugate gradient method is used to solve the system of equations given in Equation 13. Hence, the only communication necessary between processors occurs at their corresponding processor boundaries (see Figure 3) as part of the matrix-vector multiplication and global assembly of the conjugate gradient method. A total of $2(N-1)$ sends and receives are executed during this communication step.

This process of communicating the boundary-boundary modes between processors is a dominant communication cost during Stage 2 and Stage 3. In order to limit the serialization caused by using blocking sends and receives, the even processors send their last bottom corner mode forward to the next processor while the odd processors send their upper corner mode backward. Then, the even processors send their upper corner mode backward, and, finally, the odd processors send their last bottom corner mode forward. This effectively results in the first two passes of a standard tree algorithm for an all-to-all type of communication. A schematic of this communication procedure is shown in Figure 4. To further reduce communication costs associated with initial message startup times, all streamwise and spanwise planar conjugate gradient solves are performed concurrently such that the boundary modes of each of the $N_x(N_z/2+1)$ complex values in the transformed space are sent to the corresponding processor as a single packaged array. This places the communication burden primarily upon the bandwidth of the parallel system, which is advantageous for most parallel computing architectures where startup latency is dominant. We note that depending on the MPI implementation used, substitution or augmentation of the above scheme with asynchronous sends/receives may provide similar or improved results.

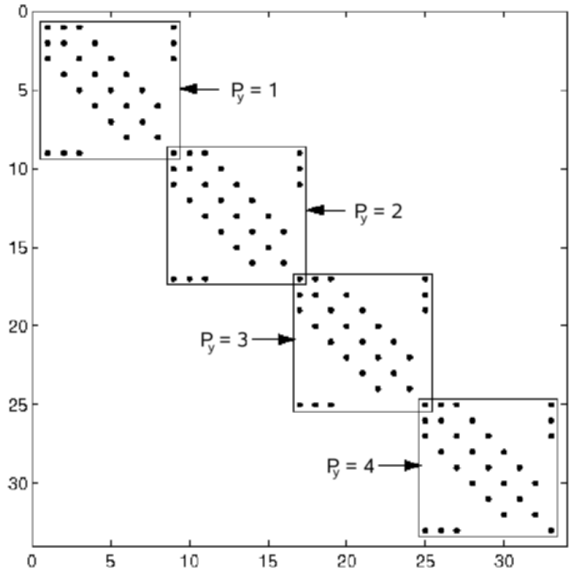


Figure 3: Decomposition of the global Helmholtz matrix with $N_e = 4$, $P = 8$, and Neumann boundary conditions over 4 separate processors ($P_y = 4$), which appears during the Stage 2 pressure solver.

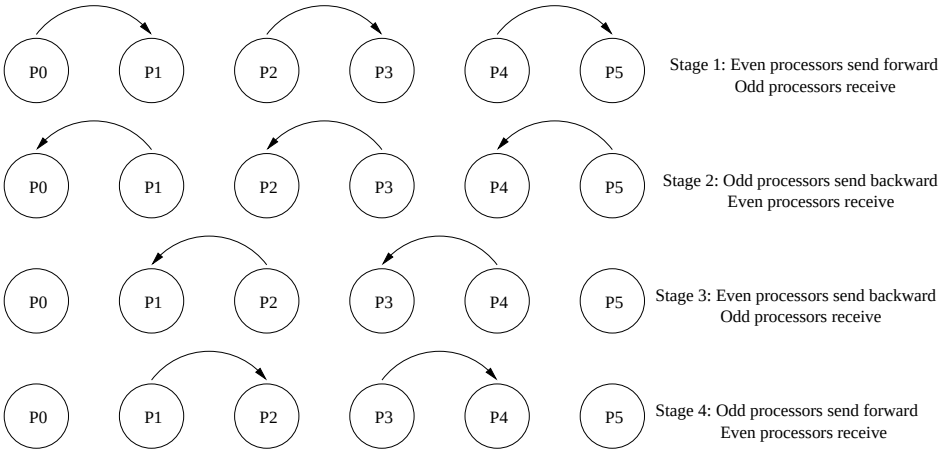


Figure 4: Boundary-boundary communication schematic for a single wall-normal communication group with $P_y = 6$. This communication pattern is used extensively during the matrix-vector multiplication of boundary-boundary modes as shown in Figure 3 during the conjugate gradient iterations of the Helmholtz solution process.

Since the decomposition of the one-dimensional domain across processors is performed at elemental boundaries, the maximum number of processors in the wall-normal direction that can be utilized is equal to the number of elements. This places an upper bound on the number of processors that can be used in the wall-normal direction; however, by parallelizing both the Helmholtz solution process (wall-normal) and the Fast Fourier transforms (xz-plane), the associated difficulties with the parallelization of FFTs can be effectively *deferred* to higher processor counts by adding processors in the wall-normal direction where the additional communication overhead is marginal. If the FFT parallelization begins to lose its advantage at $P_{xz} = \mathscr{P}$ processors for the given problem size, the efficiency limitations which come as a consequence of communications costs can be suppressed until one reaches a higher processor count of $N_e\mathscr{P}$ using the aforementioned parallelization. Hence, if the FFT scalability drops significantly at $\mathscr{P} = 32$ processors for a given xz-plane grid with $N_e = 32$, the aforementioned algorithm can effectively utilize 1024 processors efficiently. As a result, higher Reynolds number turbulent channel flow DNS can be obtained.

4 Performance Model

Scalability is a measure of the ability of a parallelization algorithm to fully utilize an increasing number of processors. Ideally, the speedup should scale linearly as more processors are utilized since the potential computing power increases linearly. Hence, an algorithm that maintains a near linear speedup over a wide range of processors is scalable. Developing performance models to characterize the scalability of an algorithm is becoming increasingly important on modern supercomputers where the ability to both employ several thousand processors efficiently and predict how changing design parameters will impact future simulations is of the utmost importance. A performance model is developed to characterize the scalability of this algorithm dependent upon the processor decomposition topology of the P_{xz} and P_y processors groups. The number of floating point operations per timestep was counted by hand in order to quantify the total computational burden the supercomputing system must support. Furthermore, the communication models used as building blocks in this analysis are the classical models for MPI_Alltoall, MPI_Allreduce, and pairwise communication between processors (28) where local communication between processors on the same node is assumed to be comparable to communication between processors on different nodes. However, we seek to explain our empirical observations and predict algorithmic scaling through control of the simulation parameters, processor speed, startup communication, and bandwidth of the given supercomputer; all factors outside of this class are considered to be negligible.

The performance model of the spectral element channel solver depends upon the computational and communication time models for the three separate stages of execution as a function of number of physical space grid points used in the streamwise, spanwise and wall-normal directions (N_x, N_y and N_z, respectively). Stage 1 principally consists of several FFTs such that the time to complete the Stage 1 computations on a single processor is approximately

$$T_1 = \left(\frac{A_1}{F}\right)\left[(72 + 15Q + 15J_e)\, N_x N_z N_y + 24\left(\frac{5}{2} N_x N_z N_y\right)\log_2(N_x N_z)\right] \ , \tag{14}$$

where F represents the average number of floating point operations each processor can perform per second and A_n is an empirically determined calibration constant that accounts for the different computational and memory access characteristics of the three stages. Similarly, the communications

model for Stage 1 is given by

$$T_{C_1} = \underbrace{7\left[P_{xz}(P_{xz}-1)\tau_s + \left(\frac{3(P_{xz}-1)N_xN_zN_y}{P_{xz}P_y}\right)\tau_w\right]}_{\text{Fast Fourier Transform communication}}, \tag{15}$$

where τ_s is the message startup time and τ_w is the inverse of the bandwidth (time necessary to communicate a single floating point value).

Stage 2 includes several FFTs as well as the pairwise and MPI_Allreduce communication during K iterations of the pressure Helmholtz solver whose computational and communication models are given by

$$\begin{aligned} T_2 =& \left(\frac{A_2}{F}\right)\left[(49+4Q+6J_e)\,N_xN_zN_y + (31+16P+(29+4P)K)N_xN_zN_eP\right] + \\ & \left(\frac{A_2}{F}\right)\left[23\left(\frac{5}{2}N_xN_zN_y\right)\log_2(N_xN_z)\right], \end{aligned} \tag{16}$$

$$\begin{aligned} T_{C_2} =& 6\left[P_{xz}(P_{xz}-1)\tau_s + \left(\frac{3(P_{xz}-1)N_xN_zN_y}{P_{xz}P_y}\right)\tau_w\right] + \\ & \underbrace{5\left[P_{xz}(P_{xz}-1)\tau_s + \left(\frac{(P_{xz}-1)N_xN_zN_y}{P_{xz}P_y}\right)\tau_w\right]}_{\text{Fast Fourier Transform communication}} + \\ & \underbrace{C(K+2)\left[\tau_s + \left(\frac{N_xN_z}{P_{xz}}\right)\tau_w\right]}_{\text{Pairwise communication}} + \underbrace{2(K+1)\log_2(P_y)\left[\tau_s + \left(\frac{N_xN_zN_eP}{P_{xz}P_y}\right)\tau_w\right]}_{\text{MPI\_Allreduce communication}}. \end{aligned} \tag{17}$$

Stage 3 is similar yet differs in that three times as much data is communicated during the viscous Helmholtz solution process yielding

$$\begin{aligned} T_3 =& \left(\frac{A_3}{F}\right)\left[24N_xN_zN_y + 3(31+16P+(29+4P)K)N_xN_zN_eP\right] + \\ & \left(\frac{A_3}{F}\right)\left[9\left(\frac{5}{2}N_xN_zN_y\right)\log_2(N_xN_z)\right], \end{aligned} \tag{18}$$

$$\begin{aligned} T_{C_3} =& \underbrace{3\left[P_{xz}(P_{xz}-1)\tau_s + \left(\frac{3(P_{xz}-1)N_xN_zN_y}{P_{xz}P_y}\right)\tau_w\right]}_{\text{Fast Fourier Transform communication}} + \\ & \underbrace{C(K+2)\left[\tau_s + \left(\frac{3N_xN_z}{P_{xz}}\right)\tau_w\right]}_{\text{Pairwise communication}} + \underbrace{2(K+1)\log_2(P_y)\left[\tau_s + \left(\frac{3N_xN_zN_eP}{P_{xz}P_y}\right)\tau_w\right]}_{\text{MPI\_Allreduce communication}}. \end{aligned} \tag{19}$$

The variable C, dependent on P_y, describes the number of pairwise communications necessary per conjugate gradient iteration (see Figure 4) and is given by

$$C = \begin{cases} 0 & \text{if } P_y = 1\,, \\ 2 & \text{if } P_y = 2\,, \\ 4 & \text{if } P_y \geq 3\,. \end{cases} \tag{20}$$

For Stage 2 and Stage 3, the number of conjugate gradient iterations per timestep K is assumed equal. The number of iterations is, in general, dependent upon the specific discretization used and the topological character of the flow but is typically less than ten.

The total computational model can be rewritten in an approximate form as

$$T_1 + T_2 + T_3 = \left(\frac{1}{F}\right)\left[307 + 83P + 21J_e + 4(29+4P)K + 80\log_2(N_xN_z)\right] N_xN_zN_eP \ , \tag{21}$$

where $N_y \approx N_eP$, $Q = P + 2$, and $A_1 = A_2 = A_3$ were assumed. Hence, the computational cost associated with increasing N_e while holding P constant is linear; however, when N_x or N_z are increased, the computational cost increases super-linearly due to the logarithmic term of the FFTs.

The common similarity between each stage allows for the communications model to be written in a more concise, approximate form as

$$\begin{aligned} T_{C_1} + T_{C_2} + T_{C_3} = \tau_s\left(21P_{xz}^2 + C_1 + C_2\log_2(P_y)\right) + \\ \tau_w\left(\frac{53N_y}{P_y} + \frac{2C_1}{P_{xz}} + \frac{2C_2N_y\log_2(P_y)}{P_{xz}P_y}\right) N_xN_z \ , \end{aligned} \tag{22}$$

where $N_y \approx N_eP$, $C_1 = 2C(K+2)$ and $C_2 = 4(K+1)$. In deriving this last cost, it was assumed that $P_{xz} \approx P_{xz} - 1$. When P_y is increased, the communication costs scale at most logarithmically due to the MPI_Allreduce communication while the pairwise communications remain constant. Furthermore, by increasing the number of processors in the P_y processor group, less data must be transmitted during MPI_Alltoall communications along a given P_{xz} processor group leading to a reduction in the total FFT communication costs. However, when P_{xz} is increased while holding P_y constant, the startup communication costs of the FFT scale as P_{xz}^2. In this form, the parallel performance enhancement by increasing P_y rather than P_{xz} is readily available; all terms that depend on P_y are at most logarithmic while those that depend on P_{xz} are at most quadratic.

The speedup S for this performance model is then given by

$$S_P = \frac{T_1 + T_2 + T_3}{\frac{T_1+T_2+T_3}{P_{xz}P_y} + \left(T_{C_1} + T_{C_2} + T_{C_3}\right)} \ , \tag{23}$$

and the efficiency ε is

$$\varepsilon = \frac{S}{P_{xz}P_y} \ . \tag{24}$$

Using the performance model described above, the wall clock time per timestep for various configurations can be computed. Variations in the communication network with respect to MPI_Alltoall, MPI_Allreduce, or pairwise operations across different computing systems can be accounted for by simply replacing the respective computational characteristics of the aforementioned model by that for a specific architecture. This allows for a robust capability to predict and tune the simulation performance on distinct architectures. All subsequent simulation data uses the predicted serial wall clock time of the performance model on a single processor for reference when computing the speedup, efficiency, and related quantities. Hence, we investigate the absolute speedup and efficiency of the performance model and algorithm.

4.1 Memory Requirements

The amount of memory necessary for the simulations is dictated by the amount of three-dimensional vector arrays necessary to complete the timestepping method. For a J_e order integrator with $N_x \times N_y \times N_z$ grid points (i.e. Reals) given $N_y \approx N_eP$ and N_s scalar fields, the amount of memory (C_M, in Megabytes) required for double-precision is approximately equivalent to

$$C_M \approx N_s\left(\frac{N_xN_zN_eP}{2^{18}}\right) \ , \tag{25}$$

a memory usage estimate consistent with combined Fourier/Spectral element methods (15; 6).

The simulation requires $N_s = 86 + 3J_e$ scalar fields. Hence, for $J_e = 3$, $N_x = N_z = 2048$, $N_e = 128$, and $P = 8$, the amount of memory required is $C_M \approx 1.472$ Terabytes of memory or, distributed over 2048 processors, 736 Megabytes per processor. Memory requirements could be lowered significantly through judicious use of available memory, however, the given arrangement was found to be convenient due to the need to parallelize many data sets simultaneously and to avoid unnecessary memory instantiation or copying.

5 Scalability

The computational scaling and performance results of the implemented Spectral Element Channel Solver (SPECS) are compared to the analytic performance model to characterize the parallel communication of the FFTs, pairwise communication, and collective communication allowing for the accurate prediction of the simulation software performance.

5.1 Hardware

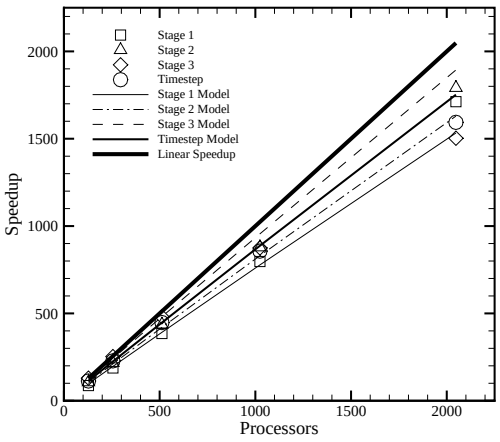
All simulations were performed on the supercomputing machine known as Thunder located at Lawrence Livermore National Laboratory. Thunder utilizes a Quadrics interconnect with $1,024$ nodes, four 1.4 GHz Itanium2 processors per node, and 8.0 GiB of DDR266 SDRAM per node. The communication network supports a 900 MB/s bandwidth with MPI latency less than 4μs. As of June 2006, Thunder ranked fourteenth on the TOP500 list of supercomputers (17) with a 22.9 TFlops peak performance using the LAPACK benchmark. The largest simulations performed utilized only half of the machine corresponding to 512 nodes or 2048 processors.

5.2 Performance measurement technique

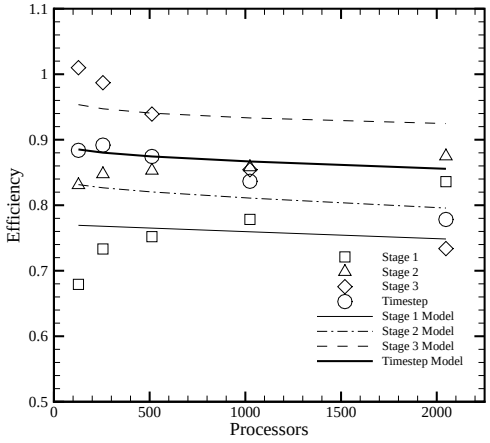
The measurements taken were the time for each stage and the total timestep averaged over ten timesteps after five initialization timesteps. The number of iterations for both the viscous and pressure solvers was fixed at $K = 10$ iterations with $P = 8$ and $J_e = 3$ to allow for direct comparison. The performance model was calibrated yielding the model parameters as $A_1 = 13.0$, $A_2 = 17.0$, $A_3 = 28.0$, $F = 4$ GFlops, $\tau_w = 4.0 \cdot 10^{-9}$ s, and $\tau_s = 4.0 \cdot 10^{-6}$ s.

5.3 Strong scaling

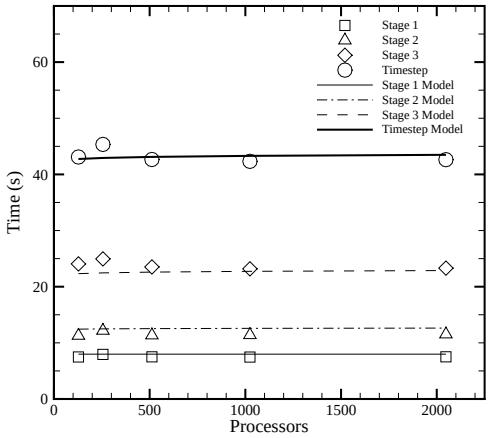
During each strong scaling simulation presented herein, the problem size was held fixed with $N_x = N_z = 512$, $N_e = 64$, and $P = 8$, denoting a characteristic simulation size. Further experiments with different simulation dimensions and data in tabular form are given in the Appendix. All results presented herein represent the typical behavior found across all experiments. The processor group parameters P_y and P_{xz} are varied in three distinct cases, (1) Constant P_{xz}, (2) Constant P_y, and (3) Variable P_{xz} and P_y, to verify the performance model and characterize the factors that influence the scalability of the algorithm. The scaling behavior of the speedup and efficiency for the strong scaling simulations as well as the wall clock time for the isomemory simulations are calculated for each case as shown in Figures 5, 6, and 7. All the plots shown within this paper adhere to the following convention. The abscissa ranges over the number of processors, and the ordinate consists of the dependent quantity of interest (speedup, efficiency or wall-clock time). Symbols are used to denote quantities coming from measured data. Lines are used to denote the predicted (model) results.



(a) Strong scaling speedup with $P_{xz} = 32$.



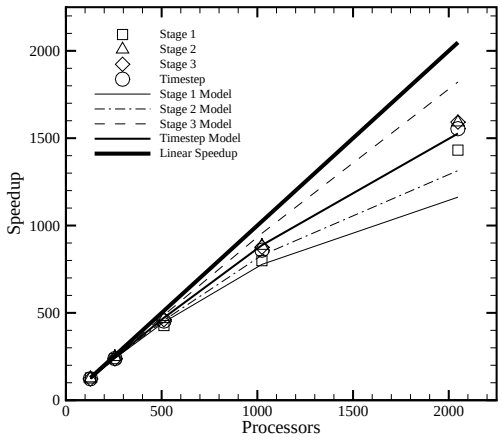
(b) Strong scaling efficiency with $P_{xz} = 32$.



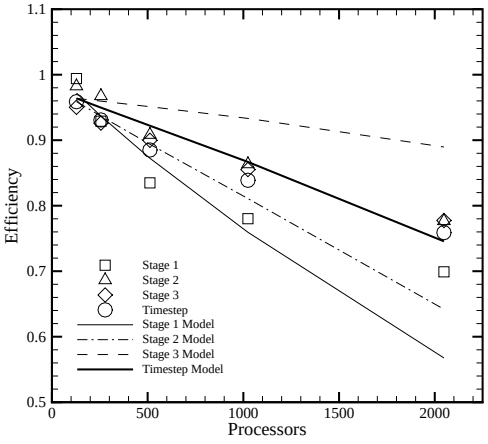
(c) Isomemory scaling wall clock time with $P_{xz} = 32$.

Figure 5: Constant P_{xz} scaling experiments.

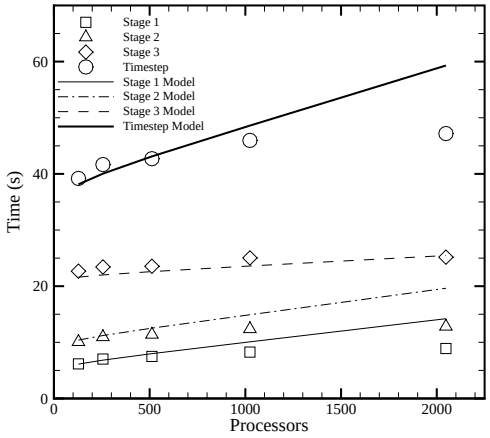
A strong scaling simulation where $P_{xz} = 32$ is held fixed is shown in Figures 5(a) and 5(b). As can be deduced from the performance model, the additional pairwise and MPI_Allreduce communication when varying P_y with constant P_{xz} yields significantly better performance than the opposite case when P_{xz} is varied with constant P_y where the costly MPI_Alltoall communication dominates as shown in Figures 6(a) and 6(b). This example demonstrates how the unfavorable FFT communications costs can be deferred to significantly higher processor counts by concurrent use of the wall-normal parallelization. A hybrid approach where both P_{xz} and P_y are varied simultaneously is exhibited in Figures 7(a) and 7(b). The cases where P_{xz} and P_y are held fixed yield upper and lower bounds, respectively, for the parallel speedup in the case where both parameters vary. The performance model slightly overpredicts the communication costs associated with the FFTs; this is a result of Thunder's use of four processors per node allowing for some local communication to replace the MPI_Alltoall communication inherent in the P_{xz} processor groups.



(a) Strong scaling speedup with $P_y = 32$.

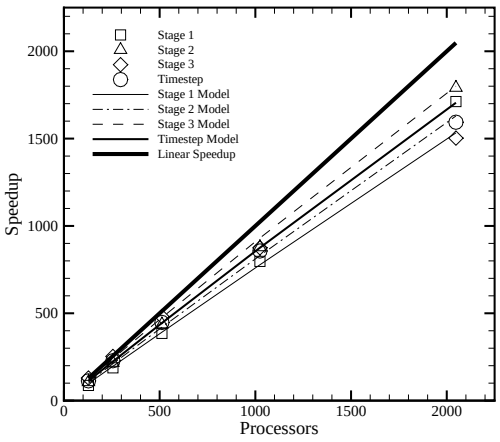


(b) Strong scaling efficiency with $P_y = 32$.

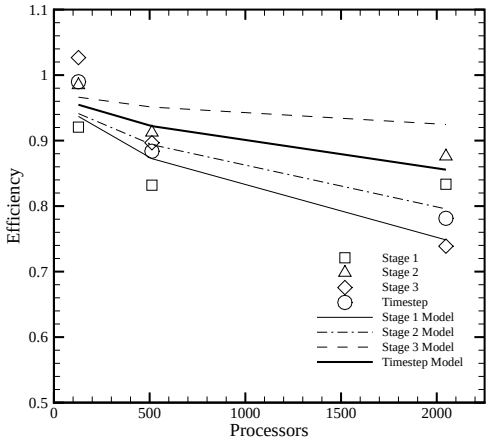


(c) Isomemory scaling wall clock time with $P_y = 16$.

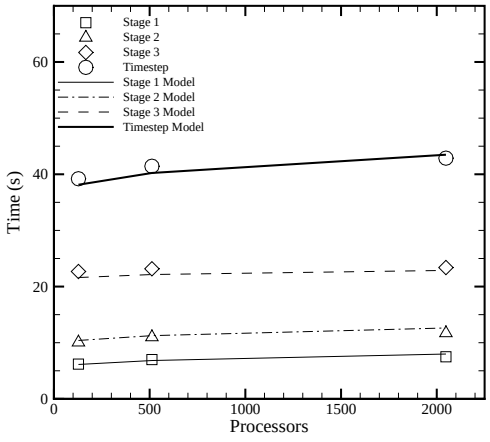
Figure 6: Constant P_y scaling experiments.



(a) Strong scaling speedup with $P_y = P_{xz}/2$.



(b) Strong scaling efficiency with $P_y = P_{xz}/2$.



(c) Isomemory scaling wall clock time with $P_y = P_{xz}/2$.

Figure 7: Variable P_{xz} and P_y scaling experiments.

5.4 Isomemory scaling

For an isomemory study of a parallel algorithm, the problem size is scaled with the number of processors so that the average memory per processor is held approximately constant. For this particular algorithm, the total memory scales linearly with the parameters N_x, N_z, N_e, and P as shown in Equation 25. Hence, if the number of processors is doubled, one of the aforementioned parameters is doubled so as to keep the average memory per processor constant.

An isomemory scaling simulation where $P_{xz} = 32$ is held fixed is shown in Figure 5(c). The total number of elements is doubled when the number of processors (in the wall-normal direction) is doubled. For this case, the computational effort per processor remains approximately constant as shown in Equation 21. Effectively, increasing P_y for a given P_{xz} while proportionately increasing the problem size provides a near constant parallel efficiency and constant execution time due to the lack of additional MPI_Alltoall type communications.

A similar set of simulation data where the number of processors controlling the Fourier planes is doubled while doubling the problem size with $P_y = 16$ held constant is shown in Figure 6(c). Even though the average domain size per processor is held constant, the total computational effort per processor is not constant due to the logarithmic term in Equation 21, hence, the execution time is expected to increase over the range investigated. Furthermore, the MPI_Alltoall communication dominates the total communication cost when increasing P_{xz} leading to further performance degradation relative to the case when holding P_{xz} constant; however, the performance model overpredicts the experimental data. We attribute this to Thunder's processor topology with four processors per node allowing for some local communication during an MPI_Alltoall.

Another simulation where both the P_{xz} and P_y processor groups were increased simultaneously such that $P_y = P_{xz}/2$ is shown in Figure 7(c). The expected trends of the FFT scaling are exhibited relative to the case where P_{xz} was fixed; however, the FFT communication cost is deferred to higher processors due to the concurrent increase in P_y processor groups. Thus, the time per timestep effectively remains constant when the number of wall-normal processors increases yet slowly degrades as the number of streamwise/spanwise processors increases. This supports the remarks concerning how the performance decrease due to the FFTs can be deferred by appropriate use of the wall-normal processor topology.

6 Conclusions

We have shown an effective parallelization method for use in the direct numerical simulation of turbulent channel flow. The proposed parallelization method was derived as a consequence of reevaluating algorithmic designs based upon assumptions that may no longer hold true. What once were limiting factors which impacted algorithm and software design decisions may no longer be the current bottleneck. Hence, the challenge is then to assess what design decisions need to be reevaluated and to consider developing novel algorithms that exploit large-scale parallelism on several thousand processors to solve problems with higher and higher Reynolds numbers thereby developing the next generation of simulations. To aid our algorithmic evaluation, a performance model was developed based upon common communication pattern building blocks that accurately predicts the parallel performance of the simulation software while maintaining a high-degree of modularity in that the various components that arise from communication overhead are easily identifiable and replaceable depending upon the underlying hardware architecture. This fidelity allows for the realistic prediction of parallel performance on arbitrary architectures.

7 Acknowledgements

This work was partially supported by the U.S. Department of Energy through the Center for the Simulation of Accidental Fires and Explosions (C-SAFE) under grant W-7405-ENG-48. We appreciate the computational support provided by C-SAFE and thank Justin Luitjens for running the simulations on Thunder. The second author acknowledges the support of NSF Career Award CCF-0347791.

8 Appendix

Further analysis of the parallel performance of the spectral element channel flow solver algorithm on the Thunder machine is given below.

Procs	P_y	N_e	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Timestep (s)
128	4	8	7.48153	11.2789	24.0591	43.1007
256	8	16	7.94185	12.1739	24.9496	45.3444
512	16	32	7.50835	11.3485	23.5179	42.6556
1024	32	64	7.46941	11.3989	23.1797	42.3284
2048	64	128	7.51132	11.5223	23.3107	42.6240

Table 1: Weak scaling of the solver for $N_x = N_z = 1024$, $P = 8$, and $P_{xz} = 32$.

Procs	P_y	N_e	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Timestep (s)
128	4	8	7.98352	12.4409	22.3218	42.7462
256	8	16	7.98352	12.4871	22.4603	42.9309
512	16	32	7.98352	12.5333	22.5988	43.1156
1024	32	64	7.98352	12.5795	22.7373	43.3003
2048	64	128	7.98352	11.6258	22.8758	43.3851

Table 2: Weak scaling of the model for $N_x = N_z = 1024$, $P = 8$, and $P_{xz} = 32$.

Procs	P_{xz}	N_x	N_z	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Timestep (s)
128	8	512	512	6.16306	10.0791	22.6770	39.1914
256	16	1024	512	7.02060	10.9436	23.4233	41.6655
512	32	1024	1024	7.48652	11.3869	23.5495	42.7035
1024	64	2048	1024	8.25493	12.3714	25.0389	45.9641
2048	128	2048	2048	8.89703	12.8236	25.1725	47.1861

Table 3: Weak scaling of the solver for $N_e = 32$, $P = 8$, and $P_y = 16$.

Procs	P_{xz}	N_x	N_z	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Timestep (s)
128	8	512	512	6.12492	10.4045	21.6084	38.1378
256	16	1024	512	6.82607	11.2152	22.0058	40.0471
512	32	1024	1024	7.98352	12.5333	22.5988	43.1156
1024	64	2048	1024	10.0858	14.9168	23.5967	48.5993
2048	128	2048	2048	14.2069	19.6337	25.4597	59.3004

Table 4: Weak scaling of the model for $N_e = 32$ $P = 8$, and $P_y = 16$.

Procs	P_{xz}	P_y	N_x	N_z	N_e	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Timestep (s)
128	8	16	512	512	32	6.18899	10.0611	22.6838	39.2047
512	16	32	1024	512	64	6.99806	10.9894	23.1659	41.4313
2048	32	64	1024	1024	128	7.49435	11.7120	23.3879	42.8742

Table 5: Weak scaling of the solver when varying both P_y and P_{xz} with $P = 8$.

Procs	P_{xz}	P_y	N_x	N_z	N_e	Stage 1 (s)	Stage 2 (s)	Stage 3 (s)	Timestep (s)
128	8	16	512	512	32	6.12492	10.4045	21.6084	38.1378
512	16	32	1024	512	64	6.82607	11.2615	22.1443	40.2319
2048	32	64	1024	1024	128	7.98352	12.6258	22.8758	43.4851

Table 6: Weak scaling of the model when varying both P_y and P_{xz} with $P = 8$.

References

[1] R.J. Adrian, C.D. Meinhart, and C.D. Tomkins. Vortex organization in the outer region of the turbulent boundary layer. *Journal of Fluid Mechanics*, 422: 1 – 54, 2000.

[2] V. Armenio, U. Piomelli, and V. Fiorotto. Effect on the subgrid scales on particle motion. *Physics of Fluids*, 11(10): 3030 – 3042, 1999.

[3] H.M. Blackburn, N.N. Mansour, and B.J. Cantwell. Topology of fine-scale motions in turbulent channel flow. *Journal of Fluid Mechanics*, 310: 269 – 292, 1996.

[4] C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods in Fluid Dynamics*. Springer-Verlag, New York, 1988.

[5] H. Choi, P. Moin, and J. Kim. Active turbulence control for drag reduction in wall-bounded flows. *Journal of Fluid Mechanics*, 262: 75 – 110, 1994.

[6] M.O. Deville, E.H. Mund and P.F. Fischer. *High Order Methods for Incompressible Fluid Flow*. Cambridge University Press, 2002.

[7] S. Dong and G.E. Karniadakis. P-refinement and P-threads. *Com. Meth. Appl. Mech. Engr.*, 192:2191–2201, 2003.

[8] S. Dong and G.E. Karniadakis. Dual-level parallelism for high-order CFD methods. *Parallel Computing*, 30(1):1–20, January 2004.

[9] S. Dong, G.E. Karniadakis, and N.T. Karonis. Cross-site computations on the TeraGrid. *Computing in Science and Engineering*, 7(5): 14–23, 2005.

[10] K.A. Hoffmann and S.T. Chiang. *Computational Fluid Dynamics for Engineers*. Engineering Education System, Austin, Texas, USA, 1989.

[11] S. Hoyas and J. Jiménez. Scaling of the velocity fluctuation in turbulent channels up to $Re_\tau = 2003$. *Physics of Fluids*, 18: 2006.

[12] G-S Karamanos, C. Evangelinos, R.C. Boes, R.M. Kirby, and G.E. Karniadakis. *Direct Numerical Simulation of Turbulence with a PC/Linux Cluster: Fact or Fiction?*, ACM/IEEE SC '99, Portland.

[13] G.E. Karniadakis, M. Israeli, and S.A. Orszag. High-order splitting methods for incompressible Navier-Stokes equations. *Journal of Computational Physics*, 97:414 – 443, 1991.

[14] G.E. Karniadakis and S.A. Orszag. Nodes modes and flow codes. *Physics Today*, 46(3): 34 – 42, 1993.

[15] G.E. Karniadakis and S.J. Sherwin. *Spectral/hp Element Methods for CFD*. Oxford University Press, Second Edition, 2005.

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

[17] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. *TOP500 Supercomputer Sites*. URL: www.top500.org .

[18] P. Moin and K. Mahesh. Direct Numerical Simulation: A Tool in Turbulence Research. *Annual Review of Fluid Mechanics*, 30: 539 – 578, 1998.

[19] R.D. Moser, J. Kim, and N.N. Mansour. Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$. *Physics of Fluids*, 11(4): 943 – 945, 1999.

[20] B. Perot. Turbulence modeling using body force potentials. *Physics of Fluids*, 11(9): 2645 – 2656.

[21] U. Piomelli and E. Balaras. Wall-layer models for large-eddy simulations. *Annual Review of Fluid Mechanics*, 34: 349 – 374, 2002.

[22] U. Piomelli, W.H. Cabot, P. Moin, and S. Lee. Subgrid-scale backscatter in turbulent and transitional flows. *Physics of Fluids*, 3(7): 1766 – 1771, 1991.

[23] G.O. Roberts. Computational meshes for boundary layer problems. In *Second Intl Conf. on Numerical Methods in Fluid Dynamics* (ed. M. Holt). Lecture Notes in Physics, vol. 8, Springer, pp. 171 – 177, 1970.

[24] W. Rodi and N.N. Mansour. Low Reynolds number k-ϵ modeling with the aid of direct simulation data. *Journal of Fluid Mechanics*, 250: 509 – 529,1993.

[25] P.R. Spalart. Direct simulation of a turbulent boundary layer up to $Re_\theta = 1410$. *Journal of Fluid Mechanics*, 187: 61 – 98, 1988.

[26] C.G. Speziale. Analytical methods for the development of Reynolds-stress closures in turbulence. *Annual Review of Fluid Mechanics*, 23: 107 – 157, 1991.

[27] H.M. Tufo and P.F. Fischer. *Terascale Spectral Element Algorithms and Implementations*, ACM/IEEE SC '99, Portland.

[28] E.F. Van de Velde. *Concurrent Scientific Computing*. Springer-Verlag: Texts in Applied Mathematical Sciences Series, 1994.

[29] T. Wei, P. Fife, J. Klewicki, and P. McMurtry. Properties of the mean momentum balance in turbulent boundary layer, pipe and channel flows. *Journal of Fluid Mechanics*, 522: 303 – 327, 2005.

[30] D.C. Wilcox. *Turbulence Modeling For CFD*. D.C.W. Industries, 2006.

[31] J. Xu. Studies of high order finite/spectral element methods for unsteady incompressible viscous flow. Ph.D. Thesis Brown University 2001.

[32] J. Xu, M.R. Maxey, and G.E. Karniadakis. Numerical simulation of turbulent drag reduction using micro-bubbles. *Journal of Fluid Mechanics*, 468: 271 – 281.

[33] M.V. Zagarola and A.J. Smits Mean-flow scaling of turbulent pipe flow. *Journal of Fluid Mechanics*, 373: 33 – 79, 1998.

[34] J. Zhou, R.J. Adrian, S. Balachandar, and T.M. Kendall. Mechanisms for generating coherent packets of hairpin vortices in channel flow. *Journal of Fluid Mechanics*, 387: 353 – 396, 1999.