1-9-2012

GPU-Accelerated Large-Eddy Simulation of Turbulent Channel Flows

Rey DeLeon
Boise State University

Inanc Senocak
Boise State University

50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
09 - 12 January 2012, Nashville, Tennessee

AIAA 2012-0722

GPU-accelerated Large-Eddy Simulation of Turbulent Channel Flows

Rey DeLeon* and Inanc Senocak†
Department of Mechanical and Biomedical Engineering Boise State University, Boise, Idaho, 83725

High performance computing clusters that are augmented with cost and power efficient graphics processing unit (GPU) provide new opportunities to broaden the use of large-eddy simulation technique to study high Reynolds number turbulent flows in fluids engineering applications. In this paper, we extend our earlier work on multi-GPU acceleration of an incompressible Navier-Stokes solver to include a large-eddy simulation (LES) capability. In particular, we implement the Lagrangian dynamic subgrid scale model and compare our results against existing direct numerical simulation (DNS) data of a turbulent channel flow at Re_τ = 180. Overall, our LES results match fairly well with the DNS data. Our results show that the Re_τ = 180 case can be entirely simulated on a single GPU, whereas higher Reynolds cases can benefit from a GPU cluster.

I. Introduction

Large eddy simulation (LES) is a numerical technique for calculating a majority of energy containing eddies allowing for the resolution of vortical flow structures and the prediction of turbulent flow statistics. LES technique finds applications in several areas such as atmospheric boundary layers, turbomachinery flows and combustion, to name a few. LES of high Reynolds number flows need spatial and temporal resolutions that can easily overwhelm individual workstations. Researchers continue to target higher Reynolds number flows using LES because of its better predictive capabilities[1,2]. Cheng et al.[3] applied OpenFOAM[4] to pollution transportation in idealized two-dimensional street canyons computation mesh of 13.5 million elements. The computation of 100 dimensionless time units on an eight-core machine was reported to have taken approximately 1,000 hours. Such long turn-around times makes LES impractical for applications in industry. But recent innovations in multi- and many-core computing architectures show great promise in broadening the adoption of LES in industry.

Scientific computing with graphics processing units (GPU) has become a new paradigm in supercomputing. As of November 2011, three of the top five supercomputers in the world adopt GPUs as accelerators[5]. The upcoming Titan supercomputer[6] at Oak Ridge National Laboratory (expected to make its debut in early 2013) is projected to be the fastest supercomputer in the world with a peak performance of over 10 petaflops (i.e. 10 trillion floating point operations per second). Titan will be powered by NVIDIA GPUs that is expected to power efficient. Furthermore, an exascale supercomputer is expected to arrive in 2018[7]. GPUs are currently viewed as a key technology that will contribute to the goal of maintaining an overall power consumption of under 20 MW for a supercomputer[8]. However, this phenomenal growth in supercomputing capacity has to be matched with a software infrastructure. Developing scalable scientific software, massive data storage and scientific visualization of exascale simulations are going to be challenging. To this end, research towards effectively using multiple GPUs on large computing clusters for scientific applications brings us closer to sustaining exascale performance on future supercomputers.

To date, there have been numerous studies on using GPUs in a variety of fields, some focusing on single GPU implementations and others focusing on multiple GPU implementations on clusters. In the field of computational fluid dynamics (CFD), some of the examples include: Thibault and Senocak[9] who focused on an incompressible flow solver, Jacobsen et al. who transformed the work of Thibault and Senocak into a Navier-Stokes solver for GPU clusters[10] with a full-depth amalgamated parallel 3D geometric multigrid method[11], Corrigan et al.[12] explored techniques to accelerate execution of unstructured mesh CFD codes on GPUs, Cohan and Molemaker[13] performed simulations of Rayleigh-Bernard convection problems, Antoniou et al.[14] developed a GPU-accelerated weighted essentially non-oscillatory scheme for supersonic flows, Brandvik and Pullan[15] developed a GPU accelerated 3D Navier-Stokes solver

*Graduate Research Assistant, Student Member AIAA.
†Assistant Professor, Senior Member AIAA.

for turbomachinery, Griebel and Zaspel[16] created a 3D incompressible solver for two phase flows, and Geveler et al. who present libraries for Lattice-Boltzmann based CFD applications on multi- and many-core machines[17].

GPU's massively parallel, many-core architecture allows for fine-grain parallelism making it well-suited for many numerical codes[18]. With the introduction of NVIDIA's Compute Unified Device Architecture[19] (CUDA) in 2007, scientific computing with GPUs made a quantum leap forward. Recent efforts have made CUDA available for a broader set of applications by allowing programs developed with CUDA to be executed on multicore x86 CPU architectures and GPUs from other vendors. In an effort to produce a dynamic compilation framework for heterogeneous computing systems, Kerr et al. created Ocelot[20] which is capable of executing Parallel Thread Execution (PTX), i.e. CUDA programs, on NVIDIA CUDA-enabled GPUs, AMD GPUs, or multicore x86 CPU architectures without recompilation of the CUDA source code. When running PTX on x86 architectures, Ocelot can either emulate PTX on a x86 architecture or translate the PTX binary to the Low Level Virtual Machine (LLVM) compiler infrastructure for execution on multicore x86 CPUs. The Portland Group Inc. recently released CUDA-x86[21] which gives flexibility to programmers by allowing CUDA applications to be tested on CUDA-enabled GPUs, multicore x86 CPU architectures, or both. CUDA-x86 is the first native CUDA compiler capable of directly creating a binary from CUDA source code for x86 platforms. NVIDIA and the Portland Group Inc. are also making the GPU's computational power more readily available to a larger scientific community by creating CUDA FORTRAN[22], an extension to the FORTRAN language for GPU computing.

With CFD simulations taking on ever increasing Reynolds numbers, even the more powerful GPUs such as the Tesla M2090 or the Tesla C2075, both of which have 6 GB of on-board memory, may not be enough. There is a need for GPU clusters and multi-GPU parallel CFD models to study turbulent flows that are common in engineering practice. For the near future, dual-level parallelism that interleaves CUDA with Message Passing Interface (MPI) appear to be an adequate choice to address multi-GPU parallelism[10,15,16]. Jacobsen and Senocak[23] investigated tri-level parallelism using CUDA, MPI, and OpenMP for clusters with multiple GPUs per node. Their results did not show a significant performance gain over dual-level parallelism with MPI-CUDA.

Our present work on LES on a GPU cluster builds upon early work done by Thibault and Senocak[9] and Jacobsen et al.[10,11] where they developed a 3D MPI-CUDA parallel incompressible flow solver, called GIN3D. Governing equations are solved using a projection algorithm[24] on a staggered, Cartesian grid with a full-depth parallel geometric multigrid pressure Poisson solver. In the present work, we introduce a large-eddy simulation capability in GIN3D. In particular, we implement the Lagrangian dynamic Smagorinsky model[25] for its flexibility for complex geometry flows with no homogeneous directions. For validation purposes, we simulate turbulent channel flows at Re_τ = 180 and compare with the direct numerical simulation (DNS) of Moser, Kim and Mansour[26].

II. Governing Equations and Numerical Approach

A. Governing Equations for LES of Incompressible Flows

The basic principle of LES is to separate a turbulent flow field into large- and small-scales using a mathematical filter. The large-scales are explicitly resolved by the simulation whereas the small-scales, also called subgrid-scales (SGS), are treated as statistically universal and their interaction with the resolved flow is modeled[27]. The governing equations used in LES of incompressible flows are the filtered form of the Navier-Stokes equations,

$$\frac{\partial \overline{u}_j}{\partial x_j} = 0 \tag{1}$$

$$\frac{\partial \overline{u}_i}{\partial t} + \frac{\partial}{\partial x_j}\left(\overline{u}_i \overline{u}_j\right) = -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(2\nu \overline{S}_{ij} - \tau_{ij}\right), \tag{2}$$

where

$$S_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) \tag{3}$$

is the deformation tensor, and

$$\tau_{ij} = \overline{u_i u_j} + \overline{u}_i \overline{u}_j \tag{4}$$

is the tensor representing the interaction of the subgrid-scales on the resolved large-scales. The overbar in these equations represents a filtered quantity.

B. Subgrid Scale Models

For turbulence closure, subgrid-scale motions are treated as statistically universal and replaced by an SGS model. The original Smagorinsky model[28] has well-known deficiencies, particularly near wall boundaries[29]. An ad hoc fix to the problem is to use van Driest damping[30] to make the eddy viscosity vanish near the wall. In 1991, Germano et al.[31] proposed an alternative method to dynamically calculate the empirical parameters in an SGS model using information from the resolved velocity field. The dynamic Smagorinsky model introduced by Germano et al. correctly predicts a decaying eddy viscosity near wall boundaries, but it has the disadvantage of requiring homogeneous directions in the flow problem at hand, which has later been addressed by other researchers[25,32,33]. In the following we briefly present the original and the dynamic Smagorinsky models.

The original Smagorinsky model creates a proportional relationship between the local SGS stresses and the local rate of strain on the large-scale eddies. It is given by

$$\tau_{ij} = -2\mu_t \overline{S}_{ij} + \frac{1}{3}\tau_{ii}\delta_{ij} = -\mu_t \left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i} \right) + \frac{1}{3}\tau_{ii}\delta_{ij}, \tag{5}$$

where μ_t is the turbulent or SGS eddy viscosity and is calculated by

$$\mu_t = \rho (C_S \Delta)^2 \sqrt{2\overline{S}_{ij}\overline{S}_{ij}}. \tag{6}$$

Δ is the filter width and can be defined by either a mathematical filter (e.g., top-hat filter) or by using the numerical grid. C_S is the model coefficient and is a constant parameter in the original Smagorinsky model. Choosing a proper C_S value, which depends upon the mesh and the flow problem being investigated, is critical. For wall boundaries in a channel flow, the model parameter C_S must be adjusted to reflect the vanishing eddies by multiplying C_S with the van-Driest damping function[30] which is given as

$$1 - exp\left(\frac{-y^+}{A} \right), \tag{7}$$

where y^+ is the non-dimensional distance given in wall units and *A* is a constant that is approximately 25. Damping the Smagorinsky coefficient through an arbitrary function significantly improves the LES results, but the procedure is ad-hoc and does not easily extend to complex geometry. This particular shortcoming is overcome by the adopting the dynamic procedure.

The first dynamic subgrid scale model was proposed by Germano et al.[31]. In their dynamic procedure, a second filter with a larger width, denoted by the hat, is applied to a resolved field. The basis of the dynamic procedure is the Germano identity

$$L_{ij} = T_{ij} - \widehat{\tau_{ij}}. \tag{8}$$

The individual terms in this algebraic relation are given by

$$T_{ij} = \widehat{\overline{u_i}\overline{u_j}} - \widehat{\overline{u}_i}\widehat{\overline{u}_j}, \tag{9}$$

$$\widehat{\tau_{ij}} = \widehat{\overline{u_i}\overline{u_j}} - \widehat{\overline{u}_i}\widehat{\overline{u}_j}. \tag{10}$$

The tensor, L_{ij}, is referred to as the Leonard stresses and can be calculated as follows

$$L_{ij} = \widehat{\overline{u}_i}\widehat{\overline{u}_j} - \widehat{\overline{u}_i}\widehat{\overline{u}_i}. \tag{11}$$

Using the Germano identity and the Smagorinsky model, Germano et al.[31] proposed to calculate C_S by

$$C_S^2 = \frac{1}{2}\frac{\langle L_{ij}\overline{S}_{ij} \rangle}{\langle M_{ij}\overline{S}_{ij} \rangle}, \tag{12}$$

which uses spatial averaging in homogeneous directions as denoted by the angle brackets. The advantages of this method are that an arbitrary damping function is no longer required to make eddy viscosity diminish near walls and

determination of an *a priori* C_S is no longer necessary. The dynamic Smagorinsky model was later modified by Lilly[34] who used a least-squares method to obtain

$$C_S^2 = \frac{1}{2}\frac{\langle L_{ij}M_{ij}\rangle}{\langle M_{ij}M_{ij}\rangle}. \tag{13}$$

In both cases, the tensor, M_{ij}, is given by

$$M_{ij} = 2\Delta^2\left[\widehat{|\overline{S}|\overline{S}_{ij}} - \alpha^2|\widehat{\overline{S}}|\widehat{\overline{S}}_{ij}\right], \tag{14}$$

where α represents the ratio of filters and is typically 2.

The dynamic Smagorinsky model[31] has the disadvantage of requiring spatial averaging in homogeneous directions to smooth C_S and stabilize the computations. Ghosal et al.[32] put the dynamic procedure on a better mathematical foundation through a constrained variational formulation where the averaging of the dynamic coefficient in homogeneous direction is justified. But most practical flow problems lack a homogeneous direction. Therefore, Meneveau et al.[25] proposed a dynamic model from a Lagrangian perspective by averaging along the flow pathlines rather than in homogeneous directions. The idea is to minimize the error caused by using the Smagorinsky model and the Germano identity by taking previous information along the pathline to obtain a current value. This formulation applies to fully inhomogeneous turbulent flows as seen in many engineering applications and requires less computational resources than other localized dynamic models[32], therefore making it a practical option in fluids engineering. The Lagrangian dynamic model uses backward time integration and an exponential weighting function that decreases the weight of past events and allows the backward time integration to be written as the relaxation-transport equations of

$$\frac{\partial \mathcal{J}_{LM}}{\partial t} + \overline{\mathbf{u}}\cdot\nabla\mathcal{J}_{LM} = \frac{1}{T}\left(L_{ij}M_{ij} - \mathcal{J}_{LM}\right), \tag{15}$$

$$\frac{\partial \mathcal{J}_{MM}}{\partial t} + \overline{\mathbf{u}}\cdot\nabla\mathcal{J}_{MM} = \frac{1}{T}\left(M_{ij}M_{ij} - \mathcal{J}_{MM}\right), \tag{16}$$

where T is the relaxation time scale. Meneveau et al. chose to define T as

$$T = 1.5\Delta\left(\mathcal{J}_{LM}\mathcal{J}_{MM}\right)^{-1/8}. \tag{17}$$

C_S is then calculated using the relation,

$$C_S^2 = \frac{\mathcal{J}_{LM}}{\mathcal{J}_{MM}}. \tag{18}$$

C. Numerical Approach

The governing equations are discretized on a Cartesian staggered grid. Second-order central-difference and Adams-Bashforth schemes are used for spatial and temporal derivatives, respectively. The projection algorithm[24] is then adopted to solve the discretized set of equations, in which the velocity field is predicted as follows

$$\mathbf{u}^* = \mathbf{u}^t + \Delta t\left(-\mathbf{u}^t\nabla\cdot\mathbf{u}^t + \nu\nabla^2\mathbf{u}^t\right). \tag{19}$$

A Poisson equation for pressure can then be written by imposing a divergence free condition on the velocity field at time $t+1$,

$$\nabla^2 P^{t+1} = \frac{\rho}{\Delta t}\nabla\cdot\mathbf{u}^*. \tag{20}$$

The pressure field at time $t+1$ is found by solving Equation 20 with a geometric, three-dimensional multigrid method with full-depth amalgamation designed for GPU clusters[11]. The pressure field is then used to correct the predicted velocity, $\mathbf{u}^*$ as follows

$$\mathbf{u}^{t+1} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla P^{t+1}. \tag{21}$$

We use double precision in the computations and use the computational mesh as the base filter. In the case of the Lagrangian dynamic model, which requires a second filter, we used a simple top-hat filter. Time advancement in the Lagrangian dynamic model (Equations 15 and 16) is performed using the first-order schemes recommended by Meneveau et al.[25] which are given by

$$\mathcal{J}_{LM}^{n+1}(\mathbf{x}) = H\left\{\epsilon\left[L_{ij}M_{ij}\right]^{n+1}(\mathbf{x}) + (1-\epsilon)\,\mathcal{J}_{LM}^{n}(\mathbf{x}-\overline{\mathbf{u}}^{n}\Delta t)\right\}, \tag{22}$$

$$\mathcal{J}_{MM}^{n+1}(\mathbf{x}) = \epsilon\left[M_{ij}M_{ij}\right]^{n+1}(\mathbf{x}) + (1-\epsilon)\,\mathcal{J}_{MM}^{n}(\mathbf{x}-\overline{\mathbf{u}}^{n}\Delta t), \tag{23}$$

where

$$\epsilon = \frac{\Delta t/T^{n}}{1+\Delta t/T^{n}} \tag{24}$$

and T is defined in Equation 17. The ramp function in Equation 22 is needed to clip away negative C_S^2 values that result from numerical inaccuracies. We use a trilinear interpolation scheme to evaluate the "upstream" values at $\mathbf{x}-\overline{\mathbf{u}}^{n}\Delta t$.

III. MPI-CUDA Implementation Details

LES capability was integrated into the MPI-CUDA 3D incompressible flow solver developed by Thibault et al.[9] and Jacobsen et al.[10,11] In their implementation, communication is overlapped with the calculations to increase performance. In this section, we give a brief summary of their work. For further details, we direct the reader to three aforementioned references.

SGS models considered in the present study do not require any modifications to the existing code other than adding CUDA kernels and incorporating a turbulent eddy viscosity in the velocity prediction step. The Smagorinsky model was relatively easy to implement, requiring only one kernel call to calculate the sub-grid viscosity. The Lagrangian dynamic model required several more kernel calls, a specific breakdown given in Section B.

A. MPI-CUDA 3D Incompressible Flow Solver

A basic outline for this implementation is shown in Listing 1. At the beginning of a time step, scalar values (e.g. temperature, eddy viscosity) are calculated first. Next, the predicted velocity is calculated as well as the resulting divergence. Using the divergence field, Equation 20 is solved. The resulting pressure field is then used to correct the predicted velocity field. At the end of each time step, the array pointers are rotated.

The dual-level parallelism adopted in GIN3D starts by decomposing the 3D domain of size $\mathrm{NX}\times\mathrm{NY}\times\mathrm{NZ}$ into subdomains of size $\mathrm{NX}\times\mathrm{NY}\times\mathrm{NZ}/\#\mathrm{GPUs}$ and are stored in a single 1D array for efficient memory transfers between host and device. This decomposition allows for dividing up the number of layers that each GPU calculates. The partial domain within a GPU is further decomposed into a top, middle, and bottom sections to accommodate overlapping of communication with computations. The GPU executes the middle section such that MPI communication and host-device memory transfers of the top and bottom sections (edges) happen simultaneously with the calculation of the middle section, which helps with achieving better scalability. The overlapping of computations with MPI communication and GPU transfers is made possible by CUDA streams. The `EXCHANGE` function in Listing 1 is when the ghost cells of each GPU are filled up with the computed edges from neighboring processes.

The pressure Poisson equation (Equation 20) is solved with a full-depth amalgamated geometric multigrid solver[11]. The multigrid method can be separated into four parts: smoothing, restriction, coarse grid solve and prolongation[35,36]. The smoothing operation in our implementation is a weighted Jacobi method that is applied at each level with a weighing coefficient of 0.86. The restriction stage takes a finer mesh and uses full-weighted averaging to transfer it to a coarser mesh that contains half as many grid cells in each direction. The problem is then directly solved at the coarse grid, and then the result is interpolated back to the finer mesh in the prolongation stage. Listing 2 presents the psuedo-code for the multigrid method with a V-cycle.

```
//time stepping loop
for (t=0; t < nsteps; t++)
{
if (TURBULENCE) {
turbulence_model ( u, v, w, nu_turb );
turbulence_model_bc ( nu_turb );
}

if (TEMPERATURE) {
temperature <<< grid, block >>> ( u, v, w, phiold, phi, phinew );
temperature_bc <<< grid, block >>> ( phinew );
EXCHANGE( phinew );
}

momentum <<< grid, block >>> ( uold, u, unew, vold, v, vnew, wold, w, wnew, nu_turb, phinew);
momentum_bc <<< grid, block >>> ( unew, vnew, wnew );
EXCHANGE( unew, vnew, wnew );

divergence <<< grid, block >>> ( unew, vnew, wnew, div );

// mgd is data structure containing pointers to pressure arrays and grid coarsening parameters
multigrid ( mgd, p, pnew, div );

correction <<< grid, block >>> ( pnew, unew, vnew, wnew );
momentum_bc <<< grid, block >>> ( unew, vnew, wnew );

EXCHANGE( unew, vnew, wnew );
ROTATE_ALL_POINTERS();
}
```

Listing 1. Host-side psuedo-code outline for the projection algorithm[24] to solve the incompressible Navier-Stokes equations.

```
// mgdata_t data structure stores all parameters for each level ( grid spacings, pointers, etc. )
// m is the coarsening level
// mgend is the total amount of restrictions that can take place
mgcycle ( mgdata_t* mgd, int m, int mgend )
{
// Smooth the result
mg_smoother(mgd, m, nIterations);


// Coarsen the mesh
restriction(mgd, m);

if ( m+1 >= mgend )
mg_coarse_solve(mgd, m+1);
else
mgcycle(mgd, m+1, mgend);

// Create finer mesh from coarsened mesh
prolongation(mgd, m+1);

// Smooth the result
mg_smoother(mgd, m, nIterations);
}
```

Listing 2. Pseudo-code for a multigrid V-cycle where the fine mesh is restricted until coarsening is no longer possible.

Multigrid methods have excellent convergence rates, but a parallel implementation of a multigrid method requires special care. Grid coarsening should be applied to the entire computational domain, but the coarsening process is prone to loss of data-parallelism on decomposed domains. Truncation of the grid coarsening process hinders convergence rates substantionally. One possible solution that is implemented in Jacobsen et al. is to use amalgamation. Once coarsening is no longer possible within each GPU, the coarsened partial domains are gathered on a single GPU process, and the V-cycle multigrid method proceeds. Once the cycle is finished, the result is scattered back to the other GPU processes and prolongation step starts on each processes.

B. GPU Implementation of the Lagrangian Dynamic Model

The Lagrangian dynamic model requires considerably more kernel calls than the Smagorinsky model: three for velocity filtering; two for the calculation of $|S_{ij}|$ and $|\widehat{S}_{ij}|$; six calls for each component of S_{ij} and $\widehat{S}_{ij}$; 18 for L_{ij} (three per filtering operation for six times); 24 for M_{ij} (three for filtering and one for calculation, each called six times); the calculation of $L_{ij}M_{ij}$ and $M_{ij}M_{ij}$ called six times; and one for the solving of the relaxation-transport equations. Listing 3 is an outline of our CUDA implementation of the Lagrangian dynamic model. In order to minimize memory usage on the GPU, we calculate the product of each component individually and adding them to a running total rather than storing the entire tensor. The `for` loop in Listing 3 serves this purpose. To reduce the amount of code, arrays were created for holding the proper pointers as well as the proper δx_i and δx_j values corresponding to the component being calculated. The arrays `dxi` and `dxj` hold the different values for δx_i and δx_j, respectively, and `ui` and `uj` hold the pointers for u_i and u_j, respectively. The tensor calculations require communication between processes, particularly with filtering operations. The `EXCHANGE` function in Listing 3 is the same as in Listing 1, where edges of subdomains are placed in the ghost rows. The same overlapping techniques are applied here as well.

IV. Simulation Setup

We simulate a periodic turbulent channel flow at Re_τ = 180 for validation purposes. All calculations were performed in double precision. The dimensions of our computational domain are $(2\pi\delta, \pi\delta, 2\delta)$ in (x, y, z) where δ is the channel half-height, x, y and z are the streamwise, spanwise and wall-normal directions, respectively. We used two grids in our simulations, a coarse resolution mesh with 64 $\times$ 64 $\times$ 96 points, and a fine resolution mesh with 128$\times$96$\times$128 points. We did not apply grid stretching in the wall-normal direction as we plan to adopt a structured adaptive mesh refinement strategy in the future.

The flow was initialized in an approach similar to Gowardhan[37] that superimposes a sinusoidal fluctuating component on a logarithmic profile. A C_S value of 0.1 was chosen for the original Smagorinsky model. The Lagrangian dynamic model was initialized with the initial conditions recommended in Meneveau et al.[25] that sets $\mathcal{J}_{LM} = C_S M_{ij} M_{ij}$ and $\mathcal{J}_{MM} = M_{ij} M_{ij}$, with C_S also being 0.1. Periodic boundary conditions were applied in the stream- and span-wise directions to both velocity and scalar quantities. On channel walls, the no-slip condition was imposed on the velocity field, and Neumann boundary conditions for pressure and the scalar quantities found in the Lagrangian dynamic model were set to zero. The flow was maintained by imposing a height independent constant pressure gradient in the streamwise direction that is u_τ^2/δ. The simulation was allowed to develop for 200 dimensionless time units ($u_\tau t/\delta$). Statistics were sampled for 20 dimensionless time units.

```
// Apply the hat filter
filter_sweep_i <<< grid, block >>> (u, v, w, uf, vf, wf);
filter_sweep_j <<< grid, block >>> (uf, vf, wf);
filter_sweep_k <<< grid, block >>> (uf, vf, wf);
velocity_bc <<< grid, block >>> (uf, vf, wf);
EXCHANGE(d_uf, d_vf, d_wf);

// Calculate |S|
magSij <<< grid, block >>> (u, v, w, magS);
scalar_bc <<< grid, block >>> (magS);
EXCHANGE(magS);

// Calculate filter(|S|)
magSij <<< grid, block >>> (uf, vf, wf, magSf);

// Calculate LijMij and MijMij on a component-wise basis
for (m = 0; m < 6; m++)
{
   //Calculate Sij. Store in Lij.
   if (m < 3)             // calculate diagonal components
      Sij_diag <<< grid, block >>> (dxi[m], ui[m], Lij);
   else                   // calculate symmetrical components
      Sij_symm <<< grid, block >>> (dxi[m], dxj[m], ui[m], uj[m], Lij);
   scalar_bc <<< grid, block >>> (Lij);
   EXCHANGE(Lij);

   // Calculate filter(|S|Sij).
   Mij_sweep_i <<< grid, block >>> (Lij, magS, Mij);
   Mij_sweep_j <<< grid, block >>> (Lij, magS, Mij);
   Mij_sweep_k <<< grid, block >>> (Lij, magS, Mij);

   // Calculate filter(Sij). Store in Lij.
   if (m < 3)             // calculate diagonal components
      Sij_diag <<< grid, block >>> (dxi[m], ufi[m], Lij);
   else                   // calculate symmetrical components
      Sij_symm <<< grid, block >>> (dxi[m], dxj[m], ufi[m], ufj[m], Lij);

   // Complete Mij
   Mij_calc <<< grid, block >>> (Lij, magSf, Mij);

   // Calculate Lij
   // Sweeps calculate filter(u_i*u_j).  filter(u_i)*filter(u_j) is in last function
   Lij_sweep_i <<< grid, block >>> (ui[m], uj[m], Lij);
   Lij_sweep_j <<< grid, block >>> (ui[m], uj[m], Lij);
   Lij_sweep_k <<< grid, block >>> (ui[m], uj[m], Lij, ufi[m], ufj[m]);

   // Calculate LijMij and MijMij scalar values
   // symm determines whether a component is multiplied by two
   if ( m < 3 ) {              // calculate diagonal components
      symm = 0.0;
      LijMij_MijMij <<< grid, block >>> (symm, Lij, Mij, LM, MM);
   } else {                    // calculate symmetrical components
      symm = 1.0;
      LijMij_MijMij <<< grid, block >>> (symm, Lij, Mij, LM, MM);
   }
}

// Solve the transport relaxation equations of the Lagrangian dynamic model
solve_transport <<< grid, block >>> ( LM, MM, Jlm, Jlmnew, Jmm, Jmmnew);
scalar_bc <<< grid, block >>> (Jlmnew);
scalar_bc <<< grid, block >>> (Jmmnew);
EXCHANGE(Jlmnew, Jmmnew);
ROTATE_POINTERS(Jlm, Jlmnew);
ROTATE_POINTERS(Jmm, Jmmnew);
```

Listing 3. The host code for the Lagrangian dynamic model. Diagonal refers to equal tensor indices, i.e. i = j, and symmetric to unequal indices.

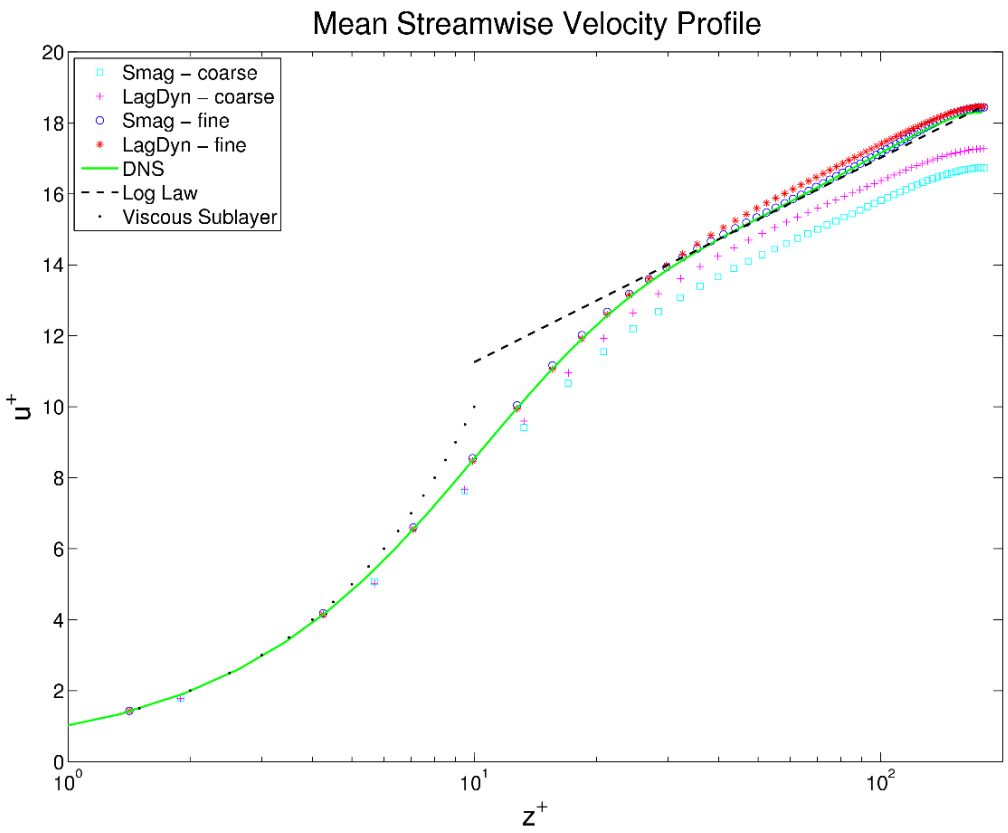

Figure 1. A comparison of the mean streamwise velocity profiles using different models and mesh sizes (coarse - 64 $\times$ 64 $\times$ 96, fine - 128 $\times$ 96 $\times$ 128): $\square$, Smagorinsky on coarse grid; +, Lagrangian dynamic on coarse grid; $\circ$, Smagorinsky on fine grid; $*$, Lagrangian dynamic on fine grid.

V. Results and Discussion

Figure 1 shows the mean velocity profiles for the fine and coarse grids with the Smagorinsky model with van Driest damping and the Lagrangian dynamic model. The profiles were compared to both the theoretical law of the wall and the DNS performed by Moser, Kim and Mansour[26]. As expected, the finer mesh did considerably better than the coarse mesh, having two points in the viscous sublayer as opposed to one.

The x-z component of the Reynolds shear stress is depicted in Figure 2. All the simulations did quite well with this statistic. The Smagorinsky model gave a higher Reynolds shear stress near the wall than the DNS while the Lagrangian dynamic model gave lower values than the DNS. Both models gave larger values of Reynolds shear stress away from the wall, particularly the Lagrangian dynamic at the coarse grid resolution.

The root mean square (rms) values of the velocity fluctuations are shown in Figures. 3, 4 and 5. With the streamwise velocity fluctuations in Figure 3, the coarse grid Lagrangian dynamic model does worse than the coarse grid Smagorinsky toward the center of the channel but better toward the wall. However, the fine grid approaches yields the exact opposite, with the Smagorinsky model performing better toward the wall but worse away from the wall. With the velocity fluctuations in the spanwise direction (Figure 4) and the wall-normal direction (Figure 5), the Smagorinsky model gives better results than the Lagrangian dynamic model, on both grids. The streamwise energy spectra from both models on the fine resolution mesh were compared to the theoretical -5/3 slope of the Kolmogorov spectrum in Figure 6. Both models produced very similar results and gave a slope close to theoretical one roughly around wavenumbers five through ten.

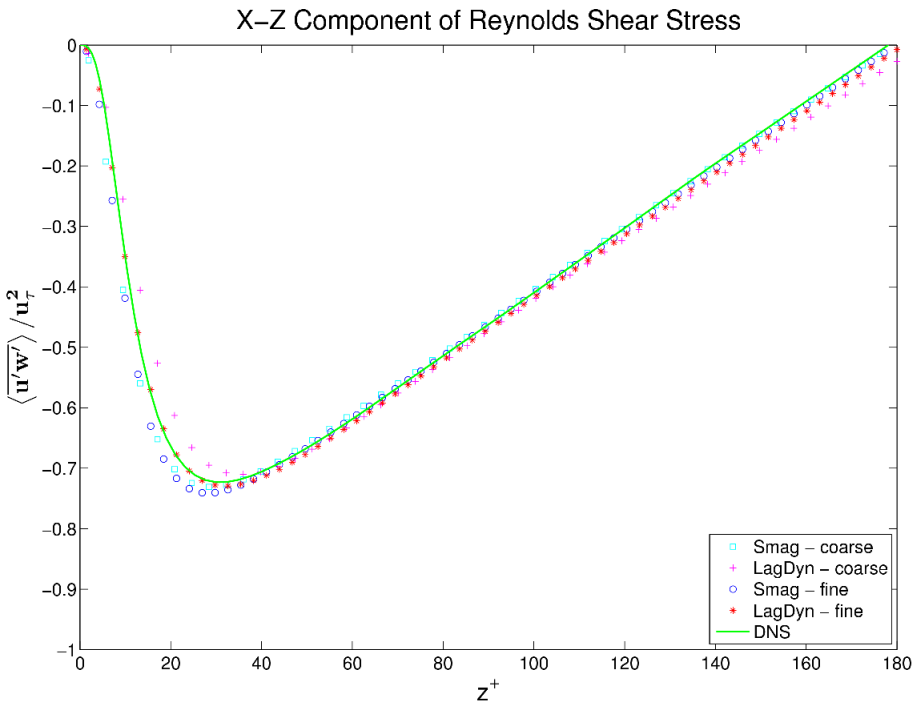

Figure 2. A comparison of the x-z component of the Reynolds shear stress tensor using different turbulence models at different grid resolutions. Note our non-traditional coordinate system has z being the wall-normal direction. □, Smagorinsky on coarse grid (64 × 64 × 96); +, Lagrangian dynamic on coarse grid; ○, Smagorinsky on fine grid (128 × 96 × 128); ∗, Lagrangian dynamic on fine grid.

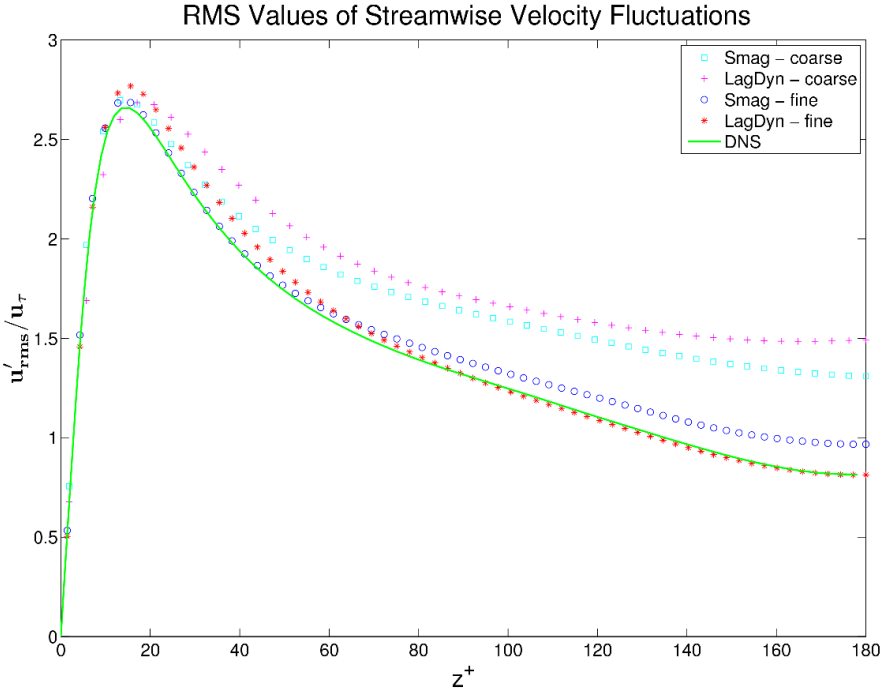

Figure 3. The rms values of streamwise velocity fluctuations: □, Smagorinsky on coarse grid (64 × 64 × 96); +, Lagrangian dynamic on coarse grid; ○, Smagorinsky on fine grid (128 × 96 × 128); ∗, Lagrangian dynamic on fine grid.

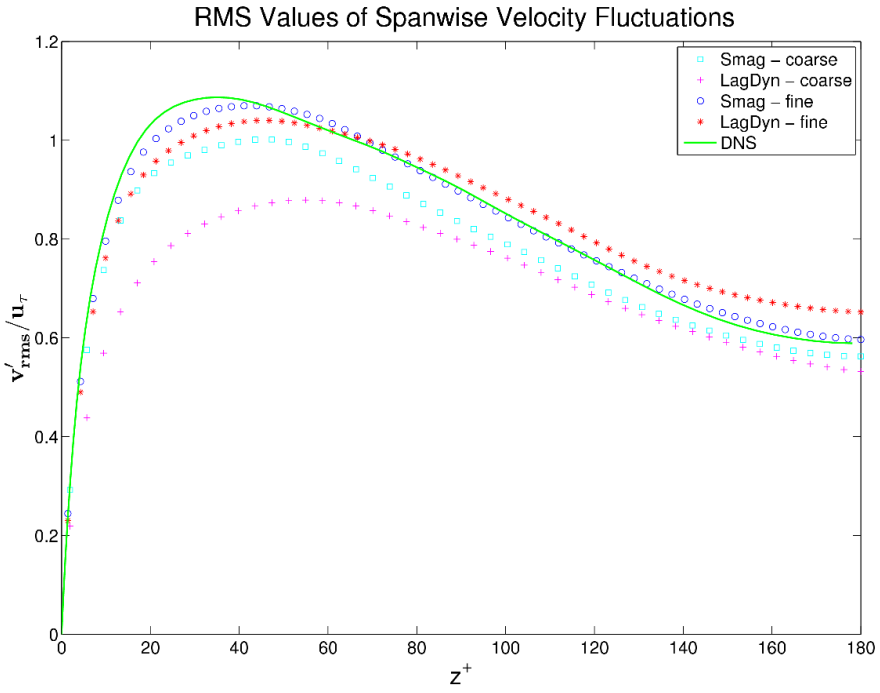

Figure 4. The rms values of spanwise velocity fluctuations: □, Smagorinsky on coarse grid (64 × 64 × 96); +, Lagrangian dynamic on coarse grid; ○, Smagorinsky on fine grid (128 × 96 × 128); ∗, Lagrangian dynamic on fine grid.

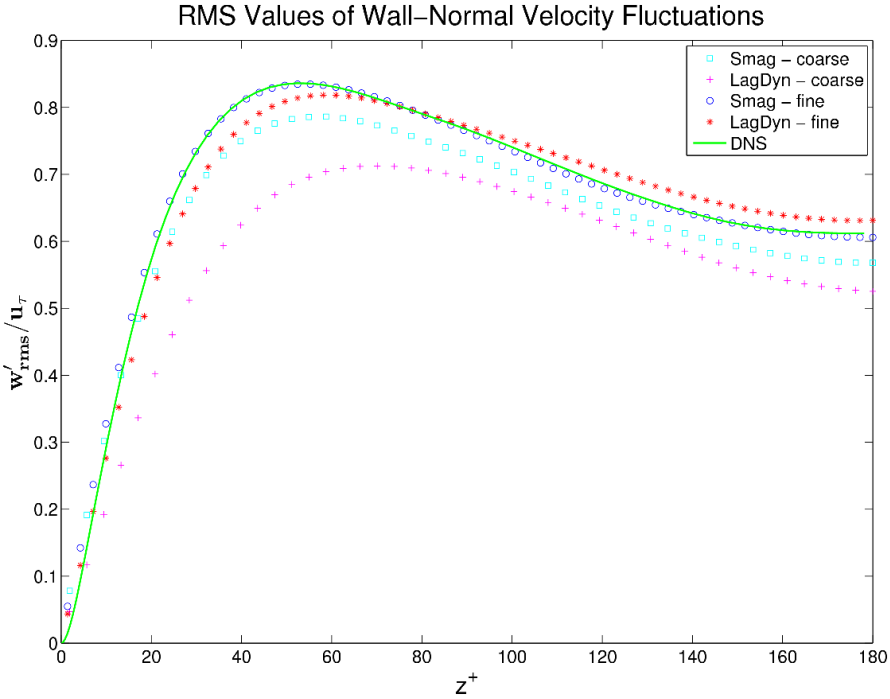

Figure 5. The rms values of wall-normal velocity fluctuations: □, Smagorinsky on coarse grid (64 × 64 × 96); +, Lagrangian dynamic on coarse grid; ○, Smagorinsky on fine grid (128 × 96 × 128); ∗, Lagrangian dynamic on fine grid.

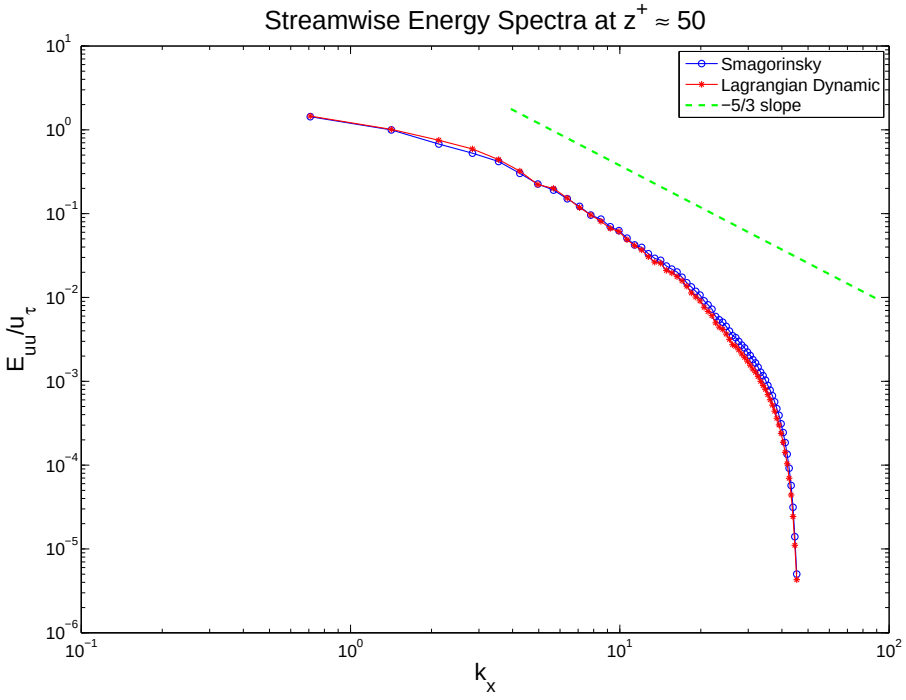

Figure 6. Streamwise spectra of turbulent kinetic energy normalized with friction velocity at approximately $z^+ \approx 50$ for Re_τ = 180 on fine resolution mesh (128 × 96 × 128).

VI. Conclusions and Future Work

We demonstrated LES of incompressible turbulent channel flows on GPU computing platforms. We considered both the original and Lagrangian dynamic Smagorinsky models. In our implementation we use MPI for coarse-grain parallelism and CUDA for fine-grain parallelism on GPUs. Our results compared fairly well with existing DNS results. While the original Smagorinsky model with the van-Driest damping function and a fine-tuned coefficient showed better turbulent statistics on the fine mesh than the Lagrangian dynamic model, the Lagrangian dynamic model produced results that are in good agreement with DNS data without any tuning in the model. Lagrangian dynamic model also has the advantage of being applicable to turbulent flows with a complex geometry, which will be the focus of our future work.

When compared to the original Smagorinsky model, the Lagrangian dynamic model increased the overall simulation time by approximately 30%. This is a significant number and reducing it would be worthwhile. A possibility that we haven't investigated yet would be to use concurrent kernels to calculate the tensors. Concurrent kernels is a new feature provided by the CUDA Toolkit version 4.0 [19] that allows kernels to be run simultaneously on NVIDIA's Fermi GPU architecture and will be investigated in a future work.

Acknowledgments

This work is partially funded by grants from National Science Foundation and through the NASA EPSCoR Graduate Fellowship program. We thank Marty Lukes of Boise State University for his continuous help maintaining our GPU computing resources.

References

[1]Hoyas, S. and Jimenez, J., "Scaling of the velocity fluctuations in turbulent channels up to Re_τ = 2003," *Physics of Fluids*, Vol. 18, No. 1, 2006, pp. 011702.

[2]Chung, D. and McKeon, B. J., "Large-eddy simulation of large-scale structures in long channel flow," *Journal of Fluid Mechanics*, Vol. 661, 2010, pp. 341–364.

[3]Cheng, W. and Liu, C.-H., "Large-Eddy Simulation of Flow and Pollutant Transports in and Above Two-Dimensional Idealized Street Canyons," *Boundary-Layer Meteorology*, Vol. 139, 2011, pp. 411–437, 10.1007/s10546-010-9584-y.

[4]The OpenFOAM Foundation, "OpenFOAM User Guide," http://http://www.openfoam.org/docs/user/, 2011.

[5]TOP500 Supercomputing Site, "Top 10 Systems," http://www.top500.org, 2011.

[6]Oak Ridge Leadership Computing Facility, "Titan," http://www.olcf.ornl.gov/titan/, 2011.

[7]Ashby, S. and et al., "The Opportunities and Challenges of Exascale Computing," DOE Office of Science, 2010.

[8]Kogge, P. M. and et al., "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," DARPA Information Processing Techniques Office, 2008.

[9]Thibault, J. C. and Senocak, I., "Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU platforms," *The Journal of Supercomputing*, Vol. 59, 2010, pp. 693–719.

[10]Jacobsen, D. A., Thibault, J. C., and Senocak, I., "An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters," *49th AIAA Aerospace Science Meeting*, 2010.

[11]Jacobsen, D. A. and Senocak, I., "A full-depth amalgamated parallel 3D geometric multigrid solver for GPU clusters," *49th AIAA Aerospace Science Meeting*, 2011.

[12]Corrigan, A., Camelli, F. F., Löhner, R., and Wallin, J., "Running Unstructured grid-based CFD solvers on modern graphics hardware," *International Journal for Numerical Methods in Fluids*, Feb. 2010.

[13]Cohen, J. M. and Molemaker, M. J., "A Fast Double Precision CFD Code using CUDA," *Proceedings of Parallel CFD*, 2009.

[14]Antoniou, A. S., Karantasis, K. I., Polychronopoulos, E. D., and Ekaterinaris, J. A., "Acceleration of a finite-difference WENO scheme for large-scale simulations on many-core architectures," Orlando, FL, United states, 2010.

[15]Brandvik, T. and Pullan, G., "An Accelerated 3D Navier-Stokes Solver for Flows in Turbomachines," *Journal of Turbomachinery*, Vol. 133, No. 2, 2011, pp. 021025.

[16]Griebel, M. and Zaspel, P., "A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations," *Computer Science - Research and Development*, Vol. 25, 2010, pp. 65–73, 10.1007/s00450-010-0111-7.

[17]Geveler, M., Ribbrock, D., Mallach, S., Göddeke, D., and Turek, S., "A Simulation Suite for Lattice-Boltzmann based Real-Time CFD Applications Exploiting Multi-Level Parallelism on modern Multi- and Many-Core Architectures," *Journal of Computational Science*, Vol. 2, jan 2011, pp. 113–123.

[18]Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., and Phillips, J., "GPU Computing," *Proceedings of the IEEE*, Vol. 96, No. 5, May 2008, pp. 879 –899.

[19]NVIDIA, *NVIDIA CUDA C Programming Guide 4.0*, 2011.

[20]Kerr, A., Diamos, G., and Yalamanchili, S., "gpuocelot - A Dynamic Compilation Framework for PTX," http://code.google.com/p/gpuocelot/, 2011.

[21]The Portland Group, "PGI CUDA-x86," http://http://www.pgroup.com/resources/cuda-x86.htm, 2011.

[22]The Portland Group, "PGI CUDA Fortran Compiler," http://http://www.pgroup.com/resources/cudafortran.htm, 2011.

[23]Jacobsen, D. A. and Senocak, I., "Scalability of Incompressible Flow Computations on Multi-GPU Clusters Using Dual-Level and Tri-Level Parallelism," *49th AIAA Aerospace Science Meeting*, jan 2011.

[24]Chorin, A. J., "Numerical Solution of the Navier-Stokes Equations," *Mathematics of Computation*, Vol. 22, No. 104, 1968, pp. pp. 745–762.

[25]Meneveau, C., Lund, T., and Cabot, W., "A Lagrangian dynamic subgrid-scale model of turbulence," *Journal of Fluid Mechanics*, Vol. 319, 1996, pp. 353 – 85.

[26]Moser, R. D., Kim, J., and Mansour, N. N., "Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$," *Physics of Fluids*, Vol. 11, No. 4, 1999, pp. 943–945.

[27]Lesieur, M., Métais, O., and Comte, P., *Large-Eddy Simulations of Turbulence*, Cambridge University Press, 2005.

[28]Smagorinksy, J., "General circulation experiments with the primitive equations," *Monthly Weather Review*, Vol. 91, No. 3, 1963, pp. 99–164.

[29]Deardorff, J. W., "A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers," *Journal of Fluid Mechanics*, Vol. 41, No. 02, 1970, pp. 453–480.

[30]Driest, E. V., "On Turbulent Flow Near a Wall," *Journal of Aeronautical Sciences*, Vol. 23, No. 11, 1956, pp. 1007–1011.

[31]Germano, M., Piomelli, U., Moin, P., and Cabot, W. H., "A dynamic subgrid-scale eddy viscosity model," *Physics of Fluids A: Fluid Dynamics*, Vol. 3, No. 7, 1991, pp. 1760–1765.

[32]Ghosal, S., Lund, T., Moin, P., and Akselvoll, K., "A dynamic localization model for large-eddy simulation of turbulent flow," *Journal of Fluid Mechanics*, Vol. 286, 1995, pp. 229–255.

[33]Piomelli, U. and Liu, J., "Large-eddy simulation of rotating channel flow using a localized dynamic model," *Physics of Fluids A: Fluid Dynamics*, Vol. 7, No. 4, 1995, pp. 839–848.

[34]Lilly, D., "A proposed modification of the Germano subgrid-scale closure method," *Physics of Fluids A: Fluid Dynamics*, Vol. 4, No. 3, 1992, pp. 633–35.

[35]Briggs, W. L., Henson, V. E., and McCormick, S. F., *A Multigrid Tutorial (2nd ed.)*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[36]Trottenberg, U., Oosterlee, C., and Schüller, A., *Multigrid*, Elsevier, 2001.

[37]Gowardhan, A. A., *Towards Understanding Flow and Dispersion in Urban Areas Using Numerical Tools*, Ph.D. thesis, University of Utah, 2008.