50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
09 - 12 January 2012, Nashville, Tennessee
AIAA 2012-0723

GPU Parallelization of an Unstructured Overset Grid Incompressible Navier-Stokes Solver for Moving Bodies

Dominic D.J. Chandar* , Jay Sitaraman†
and Dimitri Mavriplis‡

In pursuit of obtaining high fidelity solutions to the fluid flow equations in a short span of time, Graphics Processing Units (GPUs) which were originally intended for gaming applications, are currently being used to accelerate Computational Fluid Dynamics codes. With a high peak throughput of about 1 TFLOPS on a PC, GPUs seem to be favorable for many high resolution computations. One such computation that involves a lot of number crunching, is computing time accurate flow solutions past flapping wings at low speeds. The aim of the present paper is thus to discuss the development of an incompressible Navier-Stokes (INS) solver on unstructured and overset grids, and its implementation on GPUs. In its present form, the GPUINS solver solves the fluid flow equations on an unstructured/hybrid grid in the full Pressure-Poisson formulation with consistent treatment of the derivatives for the Pressure-Poisson equation (PPE) in three-dimensions. Since the equations are solved in a semi-implicit form (with viscous terms treated implicitly), the discretization results in a set of linear equations also for velocity. Hence the backbone of the GPU computation relies on developing efficient iterative linear solvers. The BiCGSTAB[1] iterative algorithm is parallelized in a matrix free approach using several GPU *kernels* such as gradient, Laplacian and reduction. Some of the simple arithmetic vector calculations are implemented using the CU++ET[2] approach where *kernels* are automatically generated at compile time. The solver is validated using standard test cases such as the (1) flow in a driven cavity, (2) flow past a cylinder, (3) flow past a plunging airfoil and (3) flow past a sphere (with and without overset grids).

I. Introduction and Background

GPUs are increasingly becoming popular among the CFD community owing to its high throughput/cost. Writing codes that run on a GPU require an intermediate low level interface (a GPU front end) that can transfer data between the CPU and GPU, and perform the required computation on the GPU. NVIDIA's CUDA architecture[3] is one such interface that supports native C/C++ language constructs. Similar to the MPI standard, where commands are concurrently executed on various processors, the CUDA programming model relies on *kernels* that execute on multiple threads. *Kernels* are similar to standard programming language functions, except for the manner in which these functions are invoked from the main program. One can write *kernels* for each arithmetic expression, or wrap a set of expressions into one *kernel*. A single call to a *kernel* will automatically spawn as many as processes the user wants, provided the number of processes is within the limits of the GPU. The last couple of years has seen a steep growth in the use of GPUs to accelerate CFD codes.[4–8] Corringan et al.[9] developed a Python based script to automatically port the flow solver *FEFLO* to the GPU. A good overview of GPU based methods including performance/memory issues for fluid flow problems can also be found in Corringan et al.[9] In this paper, we discuss the development of an Incompressible Navier-Stokes solver (GPUINS) on unstructured hybrid and overset grids which is aimed at understanding the flow physics behind flapping wings. For moving body problems, the grid motion is easily handled using overset grid methodologies.[10–12] Moreover for rigid moving bodies, the grids surrounding individual bodies need not be regenerated which makes this approach very suitable for the present problem. An example of an overset grid used in the present paper is shown in figure(1). Presently, the solver works on

*Postdoctoral Research Associate, Department of Mechanical Engineering, University of Wyoming
†Assistant Professor, Department of Mechanical Engineering, University of Wyoming
‡Professor, Department of Mechanical Engineering, University of Wyoming

fully unstructured or unstructured hybrid moving grids in two-dimensions and unstructured and non-moving overset grids in three-dimensions. For three-dimensional cases, the domain connectivity between individual component grids is handled using the approach described in Soni et al.[13] This paper is organized as follows: In Section II, the computational methodology for solving the incompressible Navier-Stokes equations is discussed in detail; In Section III, the GPU implementation of the various functions is discussed and in Section IV, sample test cases are investigated to test the GPU performance and accuracy of the numerical method.

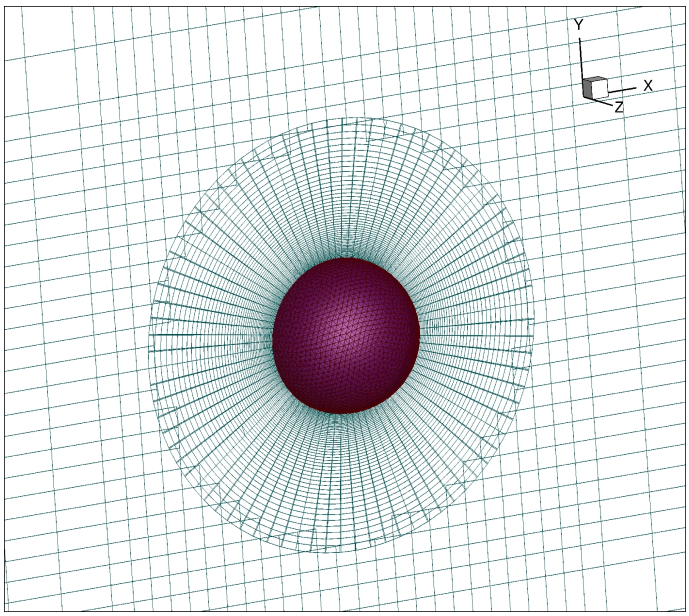


Figure 1. Overset Grid Surrounding a Sphere

II. Computational Modeling

The equations governing unsteady incompressible flow with moving grid terms are given by:

$$\frac{\partial U}{\partial t} + (U - U_G) \cdot \nabla U = -\nabla p + \nu \nabla^2 U \tag{1}$$

$$\nabla \cdot U = 0 \tag{2}$$

where U is the velocity vector, p is the pressure normalized by density, and U_G is a vector of grid speeds. Solving Eq.(1) coupled with the divergence constraint Eq.(2) is not straightforward, and has resulted in a lot of confusion in the literature. Reference is made to Gresho,[14] Brown et al.,[15] Minion and Brown[16] for a comprehensive discussion on various solution methods to the incompressible Navier-Stokes equations. In the present paper, we use the Pressure-Poisson formulation (PPE) of Henshaw et al.,[17] where the divergence constraint Eq.(2) is replaced by a Pressure-Poisson equation by taking the divergence of the momentum equation. However, the manner in which this divergence is taken is different in the present approach and is discussed subsequently. We list two forms of the PPE, the full conservative PPE,

$$\nabla \cdot (\nabla p) = -\nabla \cdot ((U - U_G) \cdot \nabla U) + \nabla \cdot (-\nu \nabla \times \nabla \times U) \tag{3}$$

and the simplified PPE (SPPE),

$$\nabla^2 p = -\left[\left(\frac{\partial u}{\partial x} \right)^2 + 2 \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} + \left(\frac{\partial v}{\partial y} \right)^2 \right] \tag{4}$$

The grid speed and viscous terms do not appear in Eq.(4) as a result of the divergence free condition. It is essential to note at this point that the SPPE has been derived from the conservative PPE through *exact* differentiation. In the above equation, the Laplacian $\nabla^2 U$ has been replaced by the cross product of vorticity $-\nabla \times \nabla \times U$ using the relation $\nabla^2 U = -\nabla \times \nabla \times U + \nabla (\nabla \cdot U)$, and by setting the divergence to zero.

II.A. Boundary Conditions and the Incompressibility Constraint

Boundary conditions for both velocity and pressure are straightforward if consistent discretization is adopted. For the velocity, a Dirichlet boundary condition on the wall which corresponds to the local velocity of the body, and a free-stream velocity at the far-field can be specified. However, for the pressure, the boundary conditions are derived from the governing equations themselves. The pressure gradients at the boundary (derived from Eq.(1) in two-dimensions) are given by

$$\frac{\partial p}{\partial x} = -(u-u_g)\frac{\partial u}{\partial x} - (v-v_g)\frac{\partial u}{\partial y} - \nu\left(\nabla\times\nabla\times U\right)_x - \dot{u_g} \tag{5}$$

$$\frac{\partial p}{\partial y} = -(u-u_g)\frac{\partial v}{\partial x} - (v-v_g)\frac{\partial v}{\partial y} - \nu\left(\nabla\times\nabla\times U\right)_y - \dot{v_g} \tag{6}$$

$$\tag{7}$$

Since pressure gradients are specified on all boundaries in the domain, the Poisson equation Eq.(3) has infinite number of solutions provided the following discrete compatibility condition Eq.(8),(9) is satisfied corresponding to the two forms of the PPE. Otherwise it has no solution.

$$\int_{d\Omega}\nabla_h p.ndS = -\int_{\Omega}\nabla\cdot\left((U-U_G)\cdot\nabla_h U\right) + \nabla\cdot\left(-\nu\nabla_h\times\nabla_h\times U\right)dV \tag{8}$$

$$\int_{d\Omega}\nabla_h p.ndS = -\int_{\Omega}\left(\left[\left(\frac{\delta u}{\delta x}\right)^2 + 2\frac{\delta u}{\delta y}\frac{\delta v}{\delta x} + \left(\frac{\delta v}{\delta y}\right)^2\right]\right)dV \tag{9}$$

where the suffix h, and the operator δ are used to represent the discrete form. Abdallah[18] demonstrates numerically, how the discrete compatibility condition is violated on non-staggered grids when the pressure Laplacian is derived in an inconsistent form on Cartesian grids. The inconsistency in his case was due to the fact that the R.H.S of the PPE was conservative, but the Laplacian was non-conservative. We demonstrate in this work that by making both the Laplacian and the R.H.S conservative, the compatibility constraint is satisfied exactly. Henshaw et al.[17] uses the SPPE with a non-conservative Laplacian and adds an extra equation for pressure, that sets the mean pressure to be zero on the domain. This results in the compatibility constraint to be satisfied approximately. The SPPE can also be forced to satisfy the compatibility constraint by fixing the solution at an arbitrary location. In this case, the compatibility constraint is satisfied as the linear solver iteration proceeds. Since this is a finite volume method, the Laplacian is always conservative, and when the conservative PPE is used (conservative R.H.S), the compatibility constraint is satisfied automatically for every linear solver iteration, and converges faster than the SPPE. The satisfaction of the discrete compatibility constraint is proved as follows:

For simplicity, consider the discrete, steady, inviscid form of the momentum equation Eq.(1). Addition of unsteady and viscous terms does alter the proof.

$$u\frac{\delta u}{\delta x} + v\frac{\delta u}{\delta y} = -\frac{\delta p}{\delta x} \tag{10}$$

$$u\frac{\delta v}{\delta x} + v\frac{\delta v}{\delta y} = -\frac{\delta p}{\delta y} \tag{11}$$

$$\tag{12}$$

If the gradients are estimated up to q^{th} order accuracy, then by Taylor's expansion, the following can be assumed.

$$\frac{\delta p}{\delta x} \approx \frac{\partial p}{\partial x} + Ch^q\frac{\partial^{q+1}p}{\partial x^{q+1}} \tag{13}$$

$$\frac{\delta p}{\delta y} \approx \frac{\partial p}{\partial y} + Ch^q\frac{\partial^{q+1}p}{\partial y^{q+1}} \tag{14}$$

$$\tag{15}$$

where the constant C depends on the grid. The PPE is formed by taking the *approximate* divergence of the momentum equations as follows:

$$-\frac{\delta}{\delta x}\left(\frac{\delta p}{\delta x}\right) - \frac{\delta}{\delta y}\left(\frac{\delta p}{\delta y}\right) = \frac{\delta}{\delta x}\left(u\frac{\delta u}{\delta x} + v\frac{\delta u}{\delta y}\right) + \frac{\delta}{\delta y}\left(u\frac{\delta v}{\delta x} + v\frac{\delta v}{\delta y}\right) \tag{16}$$

Integrating the L.H.S of Eq.(16) over the domain,

$$\int_\Omega -\frac{\delta}{\delta x}\left(\frac{\delta p}{\delta x}\right)-\frac{\delta}{\delta y}\left(\frac{\delta p}{\delta y}\right)dV=\int_\Omega -\frac{\partial}{\partial x}\left(\frac{\delta p}{\delta x}\right)-Ch^q\frac{\partial^{q+1}}{\partial x^{q+1}}\left(\frac{\delta p}{\delta x}\right)-\frac{\partial}{\partial y}\left(\frac{\delta p}{\delta y}\right)-Ch^q\frac{\partial^{q+1}}{\partial y^{q+1}}\left(\frac{\delta p}{\delta y}\right)dV \quad (17)$$

We now can use the divergence theorem in the continuous form using Eq.(17) as the discrete derivatives have been expressed in terms of their corresponding continuous ones. This gives,

$$-\int_{d\Omega}\nabla_h p.ndS-Ch^q\int_{d\Omega}\frac{\partial^q}{\partial x^q}\left(\frac{\delta p}{\delta x}\right)n_x dS+\frac{\partial^q}{\partial y^q}\left(\frac{\delta p}{\delta y}\right)n_y dS \quad (18)$$

Now if the conservative PPE is used, the R.H.S of Eq.(16) is not differentiated, but used as it is in the volume form. Eq.(16) is evaluated at the node as follows:

$$\int_\Omega \frac{\delta}{\delta x}\left(u\frac{\delta u}{\delta x}+v\frac{\delta u}{\delta y}\right)+\frac{\delta}{\delta y}\left(u\frac{\delta v}{\delta x}+v\frac{\delta v}{\delta y}\right)dV \quad (19)$$

Expressing in terms of exact derivatives and using the divergence theorem for the truncation error terms, we obtain

$$\int_\Omega \frac{\partial}{\partial x}\left(u\frac{\delta u}{\delta x}+v\frac{\delta u}{\delta y}\right)+\frac{\partial}{\partial y}\left(u\frac{\delta v}{\delta x}+v\frac{\delta v}{\delta y}\right)dV+Ch^q\int_{d\Omega}\frac{\partial^q}{\partial x^q}\left(u\frac{\delta u}{\delta x}+v\frac{\delta u}{\delta y}\right)n_x dS+\frac{\partial^q}{\partial y^q}\left(u\frac{\delta v}{\delta x}+v\frac{\delta v}{\delta y}\right)n_y dS \quad (20)$$

Using Eq.(10),(11) in Eq.(20), we obtain,

$$\int_\Omega \frac{\partial}{\partial x}\left(u\frac{\delta u}{\delta x}+v\frac{\delta u}{\delta y}\right)+\frac{\partial}{\partial y}\left(u\frac{\delta v}{\delta x}+v\frac{\delta v}{\delta y}\right)dV-Ch^q\int_{d\Omega}\frac{\partial^q}{\partial x^q}\left(\frac{\delta p}{\delta x}\right)n_x dS+\frac{\partial^q}{\partial y^q}\left(\frac{\delta p}{\delta y}\right)n_y dS \quad (21)$$

using L.H.S=R.H.S, we obtain,

$$-\int_{d\Omega}\nabla_h p.ndS=\int_\Omega \frac{\partial}{\partial x}\left(u\frac{\delta u}{\delta x}+v\frac{\delta u}{\delta y}\right)+\frac{\partial}{\partial y}\left(u\frac{\delta v}{\delta x}+v\frac{\delta v}{\delta y}\right)dV \quad (22)$$

or

$$-\int_{d\Omega}\nabla_h p.ndS=\int_\Omega \nabla\cdot\left(U\cdot\nabla_h U\right)dV \quad (23)$$

which is the discrete compatibility condition equivalent to Eq.(8). We can see that the discrete compatibility condition is satisfied to any order q provided the conservative form of PPE is used. This aspect is also demonstrated numerically later in the paper. Note that the integration over the volume of the complete domain, and integration over boundary of the domain in Eq.(23) is continuous. We can replace them by discrete quantities as follows.

$$-\sum_{d\Omega}\nabla_h p.n\Delta S=\sum_{\Omega}\nabla_h\cdot\left(U\cdot\nabla_h U\right)\Delta V \quad (24)$$

By virtue of the finite volume method, and its conservative properties, we can decompose Eq.(24) as follows

$$-\sum_i\sum_{d\Omega_{ik}}\nabla_h P.n\Delta S_{ik}=\sum_i\sum_{\Omega_i}\nabla_h\cdot\left(U\cdot\nabla_h U\right)\Delta V_i \quad (25)$$

For node-centered schemes, i represents the index of the dual cell constructed around node i, ΔV_i the volume of the dual cell, and $n\Delta S_{ik}$ the weighted outward normal on a dual cell face k from node i. The above equation can be cast into a set of linear equations for each node i as follows

$$\sum_i Ap_i=\sum_i B_i \quad (26)$$

For cell-centered schemes, i would represent cell indices. In the context of iterative solvers such as the Conjugate Gradient method (CG), or the Bi-Conjugate Stabilized method (BiCGSTAB), the quantity B_i-Ap_i represents the residual for any arbitrary p_i. Thus, Eq.(26) which is the compatibility constraint is just the sum of residuals on all nodes in the domain. Mathematically,

$$\sum_i(B_i-Ap_i)=\sum_i R_i=0 \quad (27)$$

Thus when the discrete compatibility constraint is satisfied exactly, the sum of the residuals across all nodes must vanish.

II.B. Finite Volume Discretization and Time Stepping

The governing equations are discretized on a node-centered grid and cell-centered grid for two- and three-dimensional computations respectively as shown in figure(2). On a node-centered grid, a dual cell is constructed surrounding every node, and the conservation equations are discretized on this dual cell. For the puporses of discretization, Eq.(1) is written in conservative form for each node i as follows:

$$V_i \frac{\partial U_i}{\partial t} + \sum_k F.ndS_k = \nu \sum_k \nabla U.ndS_k \tag{28}$$

where k represents the dual face index, and F, the non-linear terms inclusive of the pressure. Over any dual face k, the non-linear and viscous fluxes are computed as follows:

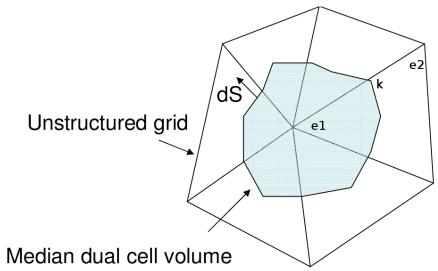
$$F_k = \frac{1}{2}\left(F_{e1} + F_{e2}\right) \tag{29}$$

$$\nabla U_k = \nabla \bar{U_k} - \left(\nabla \bar{U_k} \cdot \delta_{12} - \frac{U_{e1} - U_{e2}}{|x_{e1} - x_{e2}|}\right)\delta_{12} \tag{30}$$

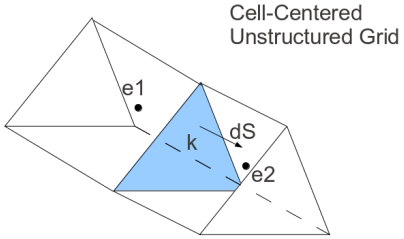
where

$$\delta_{12} = \frac{x_{e1} - x_{e2}}{|x_{e1} - x_{e2}|} \tag{31}$$

$\nabla \bar{U_k}$ represents the average of the gradients at nodes e_1 and e_2. The gradients at any node i are computed using Green-Gauss theorem. Similarly, on a cell-centered grid, the quantities e_1, e_2 represent the cell-centers of neighboring cells that share a face k. Note that there is no implicit upwinding for the convective terms, and the additional terms appearing in Eq.(30) are used to damp the high frequency modes occurring due to a central scheme.[19] Without this term, the solution will exhibit odd-even type of oscillations.



(a)



(b)

Figure 2. Discretization of the Governing Equations on (a) Node-Centered Grid and (b) Cell-Centered Grid

For temporal discretization, we use a second order Predictor-Corrector method, with the non-linear terms treated explicitly, and the viscous terms implicitly. If we denote the non-linear term by H_E and the viscous(linear) term by H_I, the predictor step is given by

$$\frac{U^p - U^n}{\Delta t} = \frac{3}{2}H_E^n - \frac{1}{2}H_E^{n-1} + \frac{1}{2}\left(H_I^p + H_I^n\right) \tag{32}$$

Since H_I is linear, we can cast it into a form AU where A is a linear operator. Rearranging the above equation gives

$$\left(U^p - \nu\frac{\Delta t}{2}AU^p\right) = \frac{\Delta t}{2}\left(3H_E^n - H_E^{n-1} + \nu AU^n\right) + U^n \tag{33}$$

or

$$BU^p = \frac{\Delta t}{2}\left(3H_E^n - H_E^{n-1} + \nu AU^n\right) + U^n \tag{34}$$

where B is a new linear operator. Note that the volume term V_i has been absorbed into the time step. The pressure is now solved using Eq.(3) using the predicted values of the velocity. Using the predicted values of pressure and velocity, the corrector step is executed as follows:

$$\frac{U^{n+1} - U^n}{\Delta t} = \frac{1}{2}H_E^p + \frac{1}{2}H_E^n + \frac{1}{2}\left(H_I^p + H_I^{n+1}\right) \tag{35}$$

Rearranging and using the linear operator B defined above, we arrive at

$$BU^{n+1} = \frac{\Delta t}{2}\left(H_E^n + H_E^p + \nu AU^n\right) + U^n \tag{36}$$

In Eq.(34) and (36), the cross product of vorticity $-\nabla \times \nabla \times U$ is used instead of the term AU^n. The pressure equation is again solved using corrected values of velocity. Eq.(34),(36) and the pressure equation Eq.(3) represent a set of linear equations which are solved using the BiCGSTAB[1] algorithm. It is to be noted that while the discrete divergence of the momentum equation was taken, the term $\frac{\partial}{\partial t}\left(\nabla \cdot U\right)$ is deliberately left out. Although this assumption appears to be acceptable in the continuous sense, the divergence is in fact non-zero in the discrete sense, and the absence of this term will result in the growth of divergence in the solution as the iteration proceeds. Abdallah[20] discretizes this divergence term using a backward difference approximation. Henshaw[17] avoids this term, but adds a damping coefficient $\approx C\nabla \cdot U$ to the pressure equation. In the present work we neglect this term, but project the velocities on to the space of divergence free velocity fields using Hodge decomposition after the predictor step. This amounts to solving an extra Poisson equation, but arrests the growth of divergence over a period of time. Similar decomposition is performed to make the initial condition divergence free.

II.C. Implementing Boundary Conditions

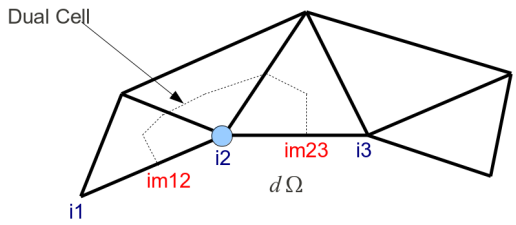
II.C.1. Two-Dimensions

Boundary conditions are needed for both velocity and pressure for solving the Poisson equations Eq.(34),(36), and (3). Since two-dimensional computations are carried out on a node-centered grid, we construct a dual cell surrounding the boundary point as shown in figure(3a). For a Dirichlet boundary condition, the governing equation at the boundary point is replaced by

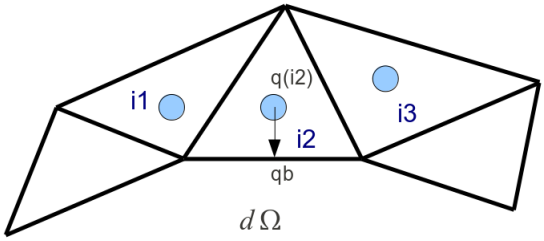
$$IU = U_b \tag{37}$$

where I is an identity matrix, and U_b is a vector of boundary values. This Dirichlet boundary condition (D.B.C) is usually applied for the velocity components. For a Neumann boundary condition (N.B.C) for the pressure equation, the governing equation is applied as follows: If B denotes the R.H.S of the Poisson equation, Eq.(3), then the finite volume discretization at point $i2$ is given by

$$\sum_k \nabla p_k \cdot n dS = B_{i2} \tag{38}$$



(a)



(b)

Figure 3. Discretization of Boundary Conditions on a (a) Node-Centered Grid and (b) Cell-Centered Grid

where k denotes the dual cell face index. The above equation can be expanded using Eqns.(29)-(31) as:

$$\sum_k \left[\frac{1}{2}\left(\nabla p_{i2} + \nabla p_{ik}\right) - \left[\frac{1}{2}\left(\nabla p_{i2} + \nabla p_{ik}\right)\cdot\delta_{ik} - \frac{p_{i2} - p_{ik}}{|x_{i2} - x_{ik}|}\right]\delta_{ik}\right]\cdot ndS = B_{i2} \tag{39}$$

where the suffix ik denotes the neighboring node along the face k to node $i2$. Re-arranging Eq.(39), we obtain

$$\sum_k \left[\frac{1}{2}\nabla p_{ik} - \left[\frac{1}{2}\nabla p_{ik}\cdot\delta_{ik} - \frac{p_{i2} - p_{ik}}{|x_{i2} - x_{ik}|}\right]\delta_{ik}\right]\cdot ndS = B_{i2} - \sum_k \left[\frac{1}{2}\nabla p_{i2} - \left[\frac{1}{2}\nabla p_{i2}\cdot\delta_{ik}\right]\delta_{ik}\right]\cdot ndS \tag{40}$$

or

$$A^{'}p = B' \tag{41}$$

The L.H.S of the above equation contain the unknowns p, and the R.H.S, the N.B.C ∇p_{i2}. It is imperative that the equations be represented in this form, else the iterative solver will fail to converge. When the face k is perpendicular to the physical boundary (dual face intersecting the boundary at point im12 or im23 in figure(3a)), the gradient at $im12$ or $im23$ is computed using a simple average and moved to the R.H.S.

II.C.2. Three-Dimensions

Since in three-dimensions, a cell-centered approach is used, a different formulation applies for the boundary conditions. As it will be seen, applying a N.B.C is quite straightforward than applying a D.B.C. For a D.B.C, following figure(3b), for any quantity q, we project the value at the cell-center $q(i2)$ on to the boundary q_b using a first order Taylor expansion.

$$q_b = q_{i2} + \Delta x_{ib}\frac{\partial q}{\partial x}_{i2} + \Delta y_{ib}\frac{\partial q}{\partial y}_{i2} + \Delta z_{ib}\frac{\partial q}{\partial z}_{i2} \tag{42}$$

Using Green-Gauss theorem, we express the gradients at the cell-center $i2$ as a function of the boundary value q_b as:

$$\begin{aligned} q_b = q_{i2} + \frac{1}{V_{i2}} \left[\Delta x_{ib} \sum_i q_i n_x dS + \Delta y_{ib} \sum_i q_i n_y dS + \Delta z_{ib} \sum_i q_i n_z dS \right] \\ + \frac{1}{V_{i2}} \left[\Delta x_{ib} q_b n_x dS + \Delta y_{ib} q_b n_y dS + \Delta z_{ib} q_b n_z dS \right] \end{aligned} \tag{43}$$

where the index i represents internal faces and V_{i2} the volume of the cell. Re-arranging, we obtain,

$$q_{i2} + \frac{1}{V_{i2}} \Delta \mathbf{x} \cdot \sum_i q \mathbf{n} dS = q_b \left(1 - \frac{1}{V_{i2}} \Delta \mathbf{x} \cdot \mathbf{n} dS \right) \tag{44}$$

Eq.(44) represents the governing equation for a cell whose boundary face is Dirichlet. Implementing a N.B.C is similar to that of Eq.(40)

II.D. Computing Aerodynamic Forces

The forces acting on the body are computed using pressure p and shear stress tensor $\tau = \nu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$ evaluated at the cell-center of the boundary cell:

$$\mathbf{F} = \int_{d\Omega} (p - \tau) \, \mathbf{n} dS \tag{45}$$

where $\mathbf{n}$ represents the normal pointing into the solid. As the gradients of the velocity field are computed to evaluate the R.H.S of the pressure Poisson equation, the same gradients are reused in this step to compute the aerodynamic forces once the pressure field is available.

II.E. Hyperviscosity and Smallest Scale Dissipation

Since there is no implicit dissipation in the above algorithm, the method becomes non-robust and unstable for small viscosities and coarse grids,[16] i.e. when the flow is under-resolved. Addition of a small amount of artificial viscosity such that the smallest scales are resolved within the given mesh produces stable solutions. It is imperative to note that the flow is still under-resolved under these circumstances for the given viscosity, but the numerical method is stable. Spurious oscillations might result in these circumstances in the current implementation as well as for other robust methods such as the Pseudospectral method or the Godunov Projection method.[16] Based on the result of Henshaw et al.,[21] the smallest scale h_{min} is proportional to the velocity gradient as follows:

$$h_{min} \approx \sqrt{\frac{\nu}{\nabla U}} \tag{46}$$

Based on this fact, we add artificial dissipation of the form

$$\nu_a = (c_1 + c_2 \nabla U) \, h^2 \tag{47}$$

where c_1, and c_2 are constants, and $h = \sqrt{V_i}$ and $V_i^{1/3}$ for two- and three-dimensions respectively. V_i is the volume of the dual cell in two-dimensions and the cell volume in three-dimensions. The above artificial dissipation is added only to the momentum equations in the predictor Eq.(34) and corrector Eq.(36) steps and not to the PPE. The modified operator B is defined as follows:

$$B = I - \left(\frac{\Delta t}{2} \nu + \Delta t \nu_a \right) A \tag{48}$$

A similar implementation of the above method on structured grids can be found in Henshaw et al.[22]

II.F. Overset Grid Connectivity for Three-Dimensional Problems

The GPU based overset grid assembly tool in three-dimensions is an extension of the implicit hole cutting algorithm outlined in Soni et al.[13] for two-dimensions. Given a set of meshes that overlap, the aim of the assembly tool would be to find suitable donor-recipient relationships and interpolation strategies. At the end of the algorithm, each cell would be classified as either (a) a discretization point (b) a fringe point or (c) a hole/unused point. If the point is a discretization point, it would also act as a potential donor for fringe points of neighboring grids. The field variables at these fringe points would be interpolated from donor cells. For the problems considered in this paper, the field values at fringe points are estimated using a first order Taylor expansion about the donor point. Figure(4) shows a section of the overset grid surrounding a sphere. The grid near the sphere consists of prisms which overlap with a Cartesian hex mesh. In the current framework, the determination of donors, fringe, and hole points are performed in an offline mode where a separate GPU program computes these parameters, hence the current capability is limited to non-moving grids.
All of the procedures required for overset grid assembly have been implemented on the GPU. Briefly, computation of the overset grid assembly can be split into three separate procedures.

1. *Preprocessing:* In the preprocessing step, meta-data structures are constructed for each grid to aid with point containment search (also called donor search). We utilize a structured auxiliary mesh for facilitating the divide-and-conquer sequence desired in donor search. Details of this methodology have been documented in our previous works,[13].[10]

2. *Donor search:* Donor search, which entails finding the cell that encompasses a given point, forms the back bone of the overset grid assembly. We utilize a gradient search approach (also termed stencil-walk in literature), which locates the actual donor cell by marching across potential donor cells based on face-edge intersections - where face is any face belonging to a given cell and edge is a line segment that connects the given point and cell center of the initial potential donor cell. Initial donor cells are located with the aid of structured auxiliary mesh generated in the first step.

3. *Donor/Fringe/Hole classification*: We utilize an implicit overset grid assembly procedure, i.e. donor are fringes are characterized based on their resolution. The main theme driving this process is to establish an overset grid assembly where the highest resolution mesh is used to solve at any location in space. Lower resolution meshes that overlap the same spatial location will interpolate from this high resolution mesh. To achieve this, we search for all grid points of any mesh for potential donor in all other meshes. Once potential donor are located, they are accepted or rejected based on the resolution constraint. In addition, we also perform checks to make sure that no donor cells are fringes and vice-versa. Minimum hole cutting, i.e removal of points inside solid boundaries is performed using a simple bounding box check - where the bounding box encompasses all the wall boundary nodes. The philosophy behind this approach is simple - any point that did not find a donor cell and is encompassed by the wall node bounding box is inside the solid body.

In Figures 5, 6 and 7 we demonstrate the improved performance of the GPU based overset grid assembly compared to corresponding CPU calculations. Figures 5 and 6 show the wall clock times and code-speedup achieved for the donor search algorithm. A purely prizmatic near-body mesh with a million points is used as the donor mesh and increasingly larger (from 1000 to 10 million) number of random points are searched in this mesh for containment. It is very encouraging to note that the GPU computations outperform the CPU calculations by a significant amount with a peak speedup of 60 obtained for large number of points. In Figure 7 the wall clock times required for the complete overset grid assembly is shown for two sets of grids system for the sphere geometry - a coarse grid system composed of 270,000 cells and a fine grid system composed of 2.7 million cells. GPU calculations show consistent speedup for both preprocessing and connectivity (which includes donor search and classification). Preprocessing is more viable for parallelism and hence accelerated by two orders of magnitude, while connectivity procedure (which include several forks and redirections in kernels) are accelerated by a factor of 30. It is also worth noting that the speedup was not as significant for two dimensional calculations (about 6 as observed in Soni et al.[13]) compared to the dramatic improvement observed in this work. It appears that dimensionality is a boon rather than a curse when it comes to utilizing GPU architectures for overset grid assembly.

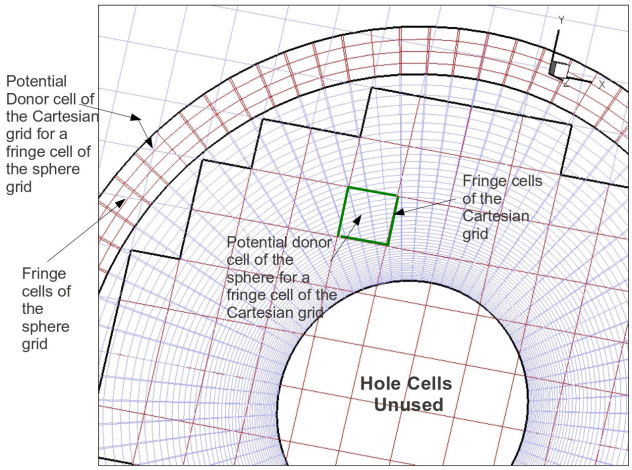


Figure 4. Overset Grid Surrounding a Sphere Showing Donor, Fringe and Hole Cells

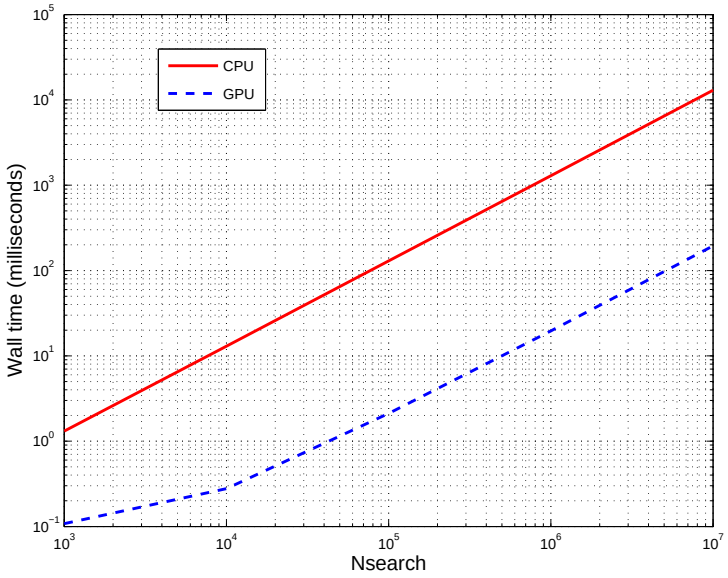


Figure 5. Wall clock times for CPU and GPU implementation of donor search algorithm for increasing number of search points ($Nsearch$).

III. GPU Implementation - The GPUINS Solver

III.A. Two-Dimensional Computations on Tesla C2050

Implementing the above algorithm on the GPU requires the user to write *kernels* similar to standard C/C++ functions. For example, kernels have to be written to compute the gradient, Laplacian, and reduction operations etc. To start with, all arrays are declared both on the GPU and the CPU. However only the arrays that reside on the GPU are used for computation. The basic building block of GPU programming using CUDA involves partitioning the domain into several blocks such that each block holds a certain number of threads. Each thread is assigned to an edge or a node depending on the type of loop that is involved(edge based or node based). For the present work, we have four different types of block and thread distributions.

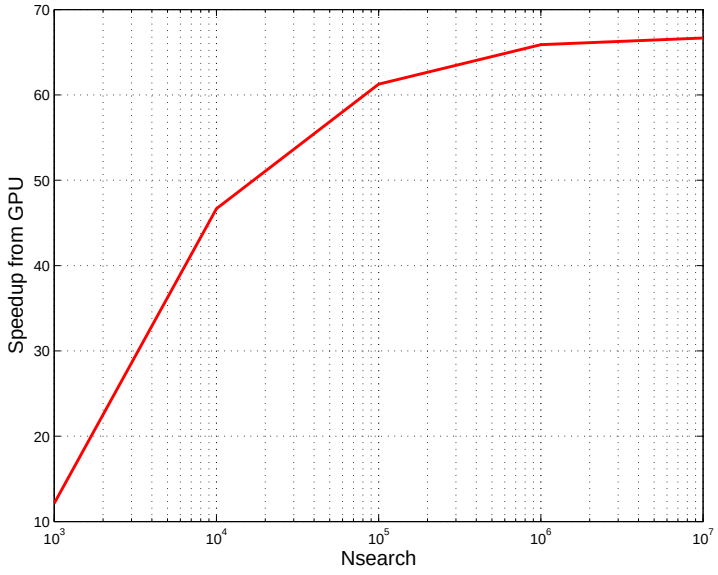


Figure 6. Speedup obtained using GPU for increasing number of search points $Nsearch$.

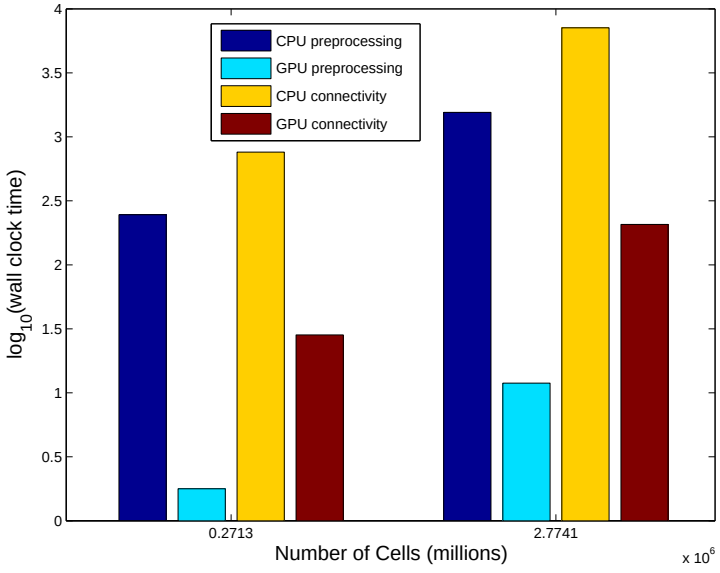


Figure 7. Wall clock times for CPU and GPU computations for various procedures used in overset grid assembly - results for two different grid systems for the same geometry, one coarse and another fine are shown.

1. Edge based, number of threads $\geq$ number of edges
2. Node based, number of threads $\geq$ number of nodes
3. Exact node based, number of threads = number of nodes (for reduction operation)
4. Boundary node based, number of threads $\geq$ number of boundary nodes

For item (1),(2) and (4), threads more than the number of edges, nodes and boundary nodes are spawned respectively, so that the threads per block equals 256. The threads per block must be a multiple of 32 (warp size) to avoid thread divergence issues which results in poor performance. For reduction operations however, the greatest common divisor of the number of nodes less than 512 (the maximum threads per block)is used to avoid erroneous addition of array values. All these characteristics are preset by the user and can be provided as an input file to the main solver. Although the GPU device supports a two-dimensional data layout, it is cumbersome to implement unstructured grid data in that form. Hence all thread and block distributions are one-dimensional. On the Tesla C2050 card, the maximum possible threads (nodes) that one can use using a one-dimensional layout is given by *maximum number of threads per block(1024)*$\times$*Maximum number of blocks in a direction(65536)*$\approx$67 Million nodes. However, one may not reach this limit due to the limitation in memory per GPU (3GB).

Having defined kernel properties, various kernels are written to handle setting up the initial conditions and solving the system of linear equations (Laplacian, gradient and reduction kernels). The following listing is a description of how the x-gradient is computed on the GPU on an unstructured grid using Green-Gauss theorem.

Listing 1. CUDA Kernel to Compute the Gradient of a Function

```
__global__ void GRADX(int N      , real* q    , real* ds1, real* ds3,
                      real* edge1, real* edge2,
                      real* dqx  , real* ivol , real* qeL, real* qeR)
{
  int tx = threadIdx.x; // Thread ID local to a block
  int bx = blockIdx.x;  // Block ID
  int bNx= blockDim.x;  // Total number of blocks
  int TID= tx + bx*bNx; // Thread ID in a 1D layout

  real xmid1, xmid2, vec11, vec21, vec12, vec22;

  if ( TID < N ) // Loop over Internal EDGES
   {
     int e1 = (int)edge1[TID];
     int e2 = (int)edge2[TID];

     real qe1 = qeL[TID];
     real qe2 = qeR[TID];
     real qm = 0.5*( qe1 + qe2 );

     real dqlx = qm*ds1[TID];
     real dqrx = qm*ds3[TID];

     real sum = dqlx + dqrx;

     real iv1 = ivol[e1]; // 1/volume(e1)
     real iv2 = ivol[e2]; // 1/volume(e2)
     atomicAdd(&dqx[e1],sum*iv1);
     atomicAdd(&dqx[e2],-sum*iv2);


   }
}
```

In the above listing, the gradient of a quantity q is sought. $ds1$, $ds3$ are weighted x-normals across the faces of a dual cell. e_1, e_2 are according to figure (2a). Using an edge based approach, this results in a race condition for a node which is shared by multiple edges. In the author's previous work,[13] this race condition is avoided by coloring various edges, and also by using a node based approach (codes run on Tesla C1060). However on the Tesla C2050 (Fermi), atomic operations are very efficient, and are found to be at least 1.2x faster than the coloring or node based approach. These atomic operations are used to sum up the contribution from various edges for a given node. For simple arithmetic operations, such as computing vorticity, divergence

or setting the R.H.S of the pressure Poisson equation, we use the $CU++ET$ framework[2] where kernels are automatically generated at compile time for any mathematical expression. As an example, the following expressions compute the divergence, vorticity, and the R.H.S of the PPE respectively using the CU++ET framework.

$$div(i) = ux(i) + vy(i) \tag{49}$$

$$vor(i) = vx(i) - uy(i) \tag{50}$$

$$rhs(i) = (divLapU(i) - dqx(i) - dqy(i)) * nodeVol(i) \tag{51}$$

i represents an index that runs through all node points and is declared as a *distributedArray*. It is worthy to note that, there are no explicit kernel calls by the user, and these statements are automatically translated to GPU kernels.

III.B. Three-Dimensional Computations on Tesla C2070

The thread and block distributions for the three-dimensional case have the following characteristics:

1. Face based, number of threads $\geq$ number of faces
2. Cell based, number of threads $\geq$ number of cells

The exact cell based approach is not used here as all reductions are performed using the *Thrust* library.[23] In this preliminary implementation, we follow a mixed cell and face based approach where the the gradients/Laplacian are computed as a two step procedure. In the first step, a kernel is called to compute the face-averaged field variables using a face based approach. In the subsequent step, another kernel is called to sum up the contribution of all faces for a given cell using a cell based approach. The disadvantage of this method is that each face value is read twice from global memory which might result in poor performance. If one were to use a face based approach completely, it is unsure at this moment whether the use of atomic operations for this purpose (similar to the two-dimensional implementation) will be beneficial as six atomic operations (2 per direction) per face are required (in contrast to 4 in two-dimensions) to compute all gradients. As the number of atomic instructions increase, the GPU performance degrades, hence using atomic operations for three-dimensional cases might not be beneficial. Similar to the two-dimensional implementation, a one-dimensional thread/block layout is used.

IV. Numerical Validation and GPU Performance Tests

IV.A. Two-Dimensional Computations

Several validation cases have been considered to establish confidence in the computed results using the GPUINS solver. First, we demonstrate the satisfaction of the compatibility constraint Eq.(8),(9) numerically, followed by accuracy test cases and some benchmark problems.

IV.A.1. Satisfaction of the Compatibility Constraint

We consider a driven cavity flow at $Re = 400$ on a square $0 \leq x \leq 1$, $0 \leq y \leq 1$ using 8300 nodes. Using Eq.(27), we compute the sum of all the pressure residuals using the two forms of the PPE Eq.(8),(9) during each BiCGSTAB iteration. Convergence is said to be obtained when the relative residual $\frac{||B - Ap||_2}{||B||_2}$ falls below a tolerance. Tables(1)-(3) show the convergence for the two types of PPE considered. It is seen that when the conservative form of the PPE is used, this residual is almost zero right from the initial iterate, and converges in 241 iterations. However when the SPPE is used, the residual is non-zero, and the l^2 norm of the relative residual diverges. However, when the solution at an arbitrary point is fixed, the SPPE also converges, but slowly as the iterate proceeds (≈2x the number of iterations compared to the earlier PPE).

IV.A.2. Convergence Studies

To assess the accuracy of the discretization procedure adopted, we consider traveling wave solutions to the incompressible Navier-Stokes equations[16] on a doubly periodic unit square. These solutions do not introduce

Table 1. Convergence of the Conservative PPE

Iteration	$\sum_i R_i$	Relative Residual
1	-2.03×10^{-16}	2.78
141	-9.76×10^{-17}	5.9×10^{-5}
241	-2.18×10^{-17}	3.16×10^{-9}

Table 2. Convergence of the Simplified PPE

Iteration	$\sum_i R_i$	Relative Residual
1	-7.7×10^{-2}	2.73
141	-7.7×10^{-2}	2.5×10^{-2}
241	-7.7×10^{-2}	3.9×10^{-2}
481	-7.7×10^{-2}	7.11×10^{8}

source terms into the existing governing equations.

$$u(x,y,t) = 1 + 2cos(2\pi(x-t))sin(2\pi(y-t))e^{-8\pi^2\nu t} \quad (52)$$

$$v(x,y,t) = 1 - 2sin(2\pi(x-t))cos(2\pi(y-t))e^{-8\pi^2\nu t} \quad (53)$$

$$p(x,y,t) = -(cos(4\pi(x-t)) + cos(4\pi(y-t))e^{-16\pi^2\nu t} \quad (54)$$

The initial conditions are prescribed based on the above set of equations by setting $t = 0$. A Dirichlet boundary condition for velocity based on the boundary values of the above equation is specified. Pressure is solved using the conservative PPE. A constant time step of $\Delta t = 10^{-6}$ is used for all grids ranging from 185 to 10600 nodes. Since the pressure obtained from BiCGSTAB is one of the infinite set of solutions to the Neumann problem, we choose to compare the computed pressure gradient with that of the exact gradient rather than comparing pressure. Figure(8) shows the convergence rates after 100 time steps along with a base 2^{nd} order curve for velocity, pressure and divergence. We find that velocity has global second order convergence, but the pressure gradient has second order convergence only in the interior nodes. This is due to the fact that at boundaries, a non-compact stencil is used to compute the gradient, which lowers the accuracy. However, in practical applications the mesh near the boundary will be very fine, hence the accuracy of the pressure gradient can be recovered. The order of convergence for divergence is between 1 and 2. We hope to improve this accuracy by using higher order discretization procedures. It is important to note that these results depend on the type of unstructured grid, and are valid only if the grid has regular equilateral triangles in most of the domain. For a completely irregular triangular mesh, it can be proved that the global order of accuracy of gradients is between 1 and 2.

IV.A.3. Flow in a Driven Cavity at Re = 100

The flow in a driven cavity is of considerable interest to the CFD community as the problem is very simple, has boundary conditions which are straightforward to implement, and the computation can be carried out very quickly. The appearance of primary and secondary vortices make it more appealing and many flow

Table 3. Convergence of the Simplified PPE When the Solution is Fixed at an Arbitrary Point

Iteration	$\sum_i R_i$	Relative Residual
1	-7.7×10^{-2}	2.73
141	-7.4×10^{-2}	4.1×10^{-2}
241	-5.9×10^{-2}	2.1×10^{-2}
361	-3.7×10^{-4}	4.3×10^{-4}
461	-5.8×10^{-9}	7.6×10^{-9}

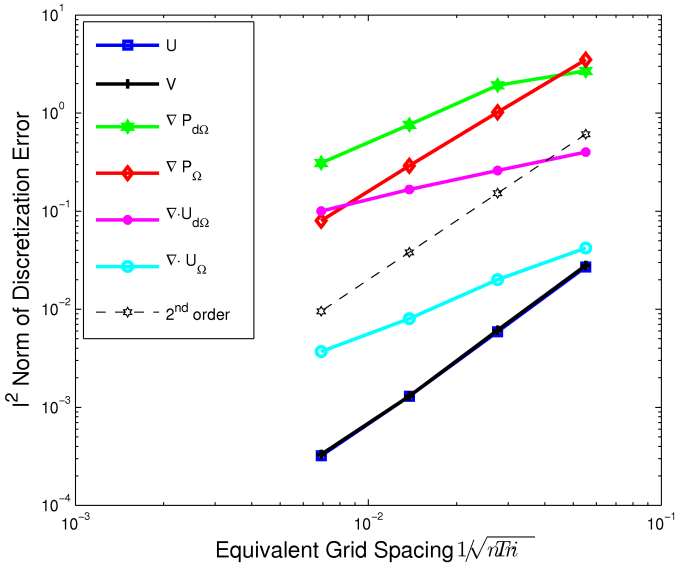


Figure 8. Convergence of the Traveling Wave Solution for Different Grids

solvers are tested to reproduce these flow features. Ghia's[24] test case is considered one of the standard benchmark cases to validate incompressible flow problems, as fine grids with a vorticity-stream function approach was used to compute the flow in a driven cavity for a wide range of Reynolds numbers. In the current implementation, we demonstrate results for Re=100 in comparison with the reference computation of Ghia et al.[24] For this case, no explicit artificial dissipation is needed as the Reynolds number is low. Figure(9) shows the triangular grid with 16384 elements, and figure(10) shows the streamlines after 20s of physical time. Both the primary and secondary vortices are captured very well. Figure (11) compares the x-velocity profile along the geometric center of the cavity ($x = 0.5$) with that of Ghia's computation. A very good match is obtained thereby establishing confidence in our implementation.

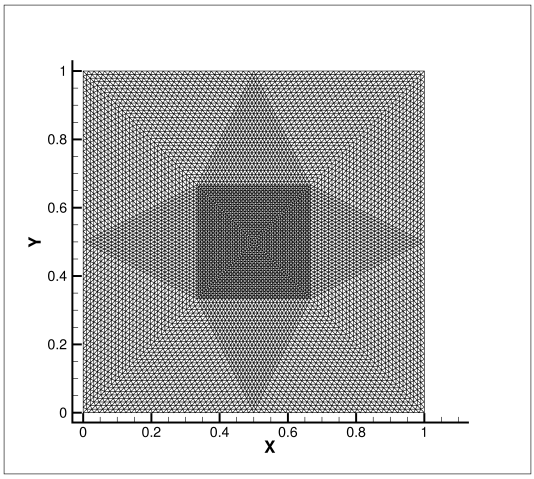


Figure 9. Unstructured Grid used for the Driven Cavity Flow Problem at Re=100

IV.A.4. Plunging Airfoil

Recently, a lot of effort is being put in to understanding the mechanics of thrust generation when an airfoil or a wing is made to execute simple periodic oscillations. Following the same lines, we consider two cases of

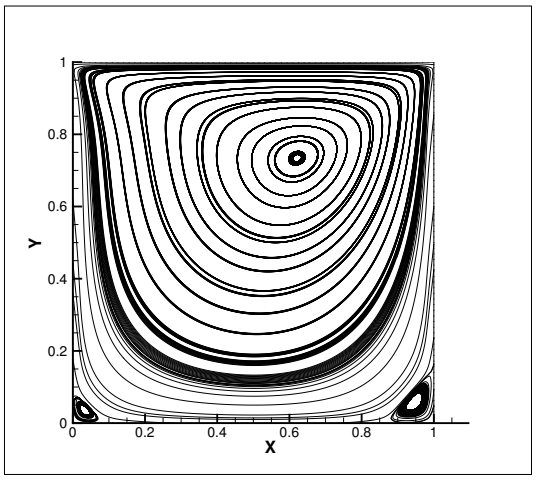


Figure 10. Streamlines for the Flow in a Driven Cavity at Re=100

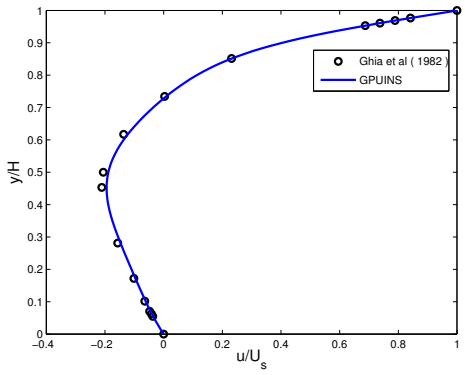


Figure 11. Comparison of the X-Velocity Profile along the Geometric Center of the Cavity at Re=100

a plunging NACA 0012, 0014 airfoil at Reynolds number 500 and 10000 respectively. For Re=500, current computations are compared with flow visualization results of Jones et al.[25], and for Re=10000, with the overset grid computations of Tuncer and Kaya[26]. A sinusoidal prescribed motion of the form $h = h_0 sin(\omega t)$ is prescribed for the airfoil. Table (4) shows various parameters for the two test cases. Since this is a moving body test case, usage of overset grids to handle complex motions is quite favorable. Since this part is under development, we choose to move the entire grid for the moment. As there is no deformation, there is no need to invoke the Geometric Conservation Law (GCL), and the grid motion is simplified to a greater extent. Also there is no requirement to update cell face normals as this is a pure translational motion. The grid velocities are passed to the velocity solver in both the predictor and corrector steps, and grid accelerations appear as boundary conditions to the Pressure Poisson equation Eq.(5),(6). The grid motion is then updated at the end of the time step. An unstructured hybrid grid with 47600(quad + tri) elements is generated such that the boundary between the two types of elements is conformal. A portion of the hybrid grid is shown in figure(12). Solutions are computed for six cycles of oscillation, and the corresponding vorticity contours are shown in figure(13) in comparison with the flow visualization results of Jones et al.[25] A comparison of the particle traces from the present computation along with the unsteady panel code[25] is also given in figure(14). A satisfactory comparison is obtained in terms of the wake deflection. A comparison of the aerodynamic forces (drag) with those of Tuncer and Kaya[26] shown in figure(15) also show good agreement.

Table 4. Computational Parameters for the Plunging Airfoil Case

	0012	0014
Reduced Frequency k, $\omega c/U_{\text{inf}}$	12.3	2.0
Plunge Amplitude h_0	0.12	0.4
Reynolds Number	500	10000
Small Scale Dissipation(c_1, c_2)	0.1	0.1
Number of Triangles	30000	32000

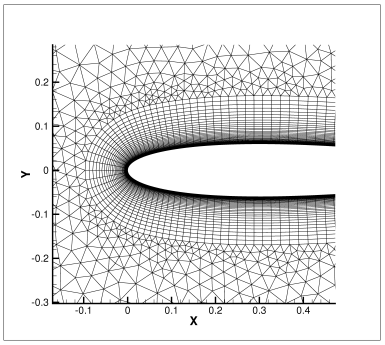


Figure 12. A Portion of the Unstructured Hybrid Grid Around a NACA 0012 Airfoil

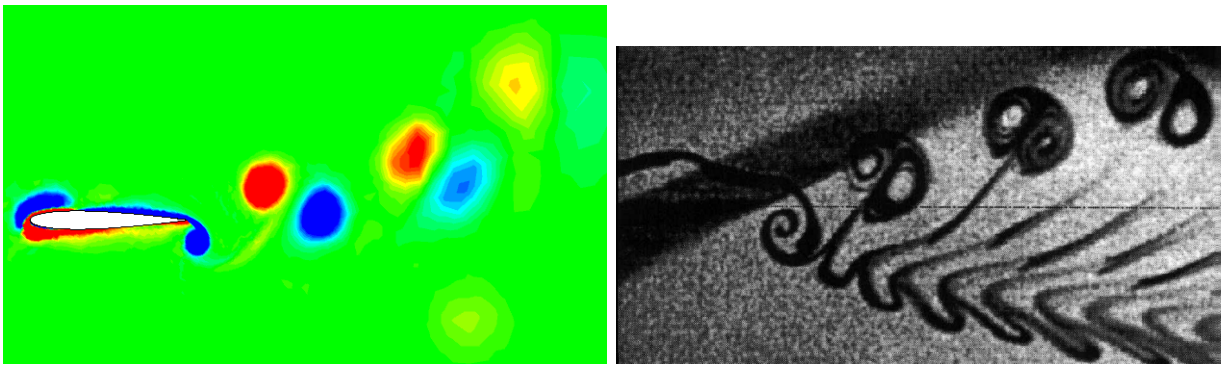
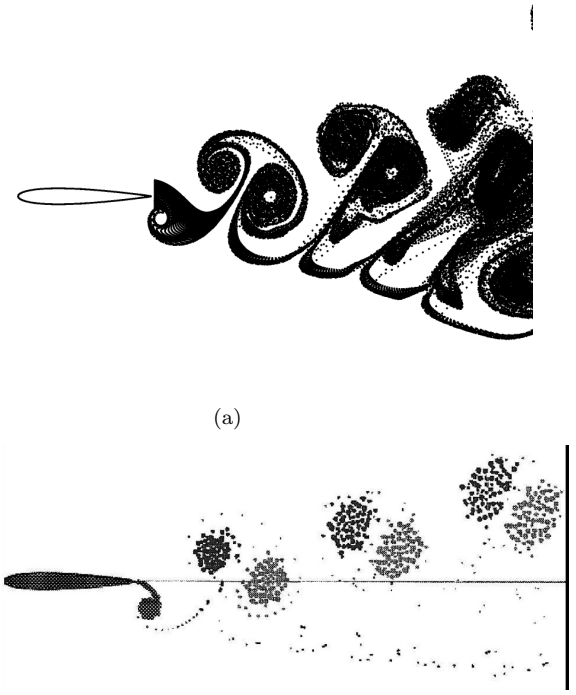


Figure 13. A Comparison of Flow Structures Behind a Plunging Airfoil between GPUINS Computation (left) and experiments from Jones et al.[25] (right)

IV.A.5. GPU Performance Results

In this section, we compare the performance of the GPU with the corresponding CPU version of the code. All two-dimensional computations have been performed on a Tesla C2050 (Fermi) graphics card using double precision. As the base code is written in the GPU framework, a CUDA to C++ backward compatibility tool *ugc* (*ugc* for **u**nified **G**PU **C**PU) has been developed using C++ string manipulations. Using this tool, both CUDA and C++ codes are identically the same. The *ugc* tool scans the CUDA code, and converts kernels to serial loops and writes a C++ version of the CUDA code, which is then compiled using *gcc*. A medium level of optimization $-O2$ is used for the serial code. A comparison of the CPU time spent per time step, for one iteration of the linear solver (velocity and pressure) during the predictor step on a grid with 1 million points for the driven cavity problem is given in figure(16). Each of the routines have been timed individually, and on an average a speed-up of 10 is obtained. We further decompose the timings for the pressure solver and list individual kernel performances such as the reduction operation (dot product), Laplacian, and gradient computation. Figure(17) lists these results. The number of calls to these functions during the first BiCGSTAB iteration is listed at the end of the graph. It is clearly seen that the reduction operation performs very poorly on the GPU. This is due to the fact that a very naive approach was followed where



(a)

(b)

Figure 14. A Comparison of Particle Traces Behind a Plunging Airfoil (a) GPUINS Computation (b) Unsteady Panel code from Jones et al.[25]

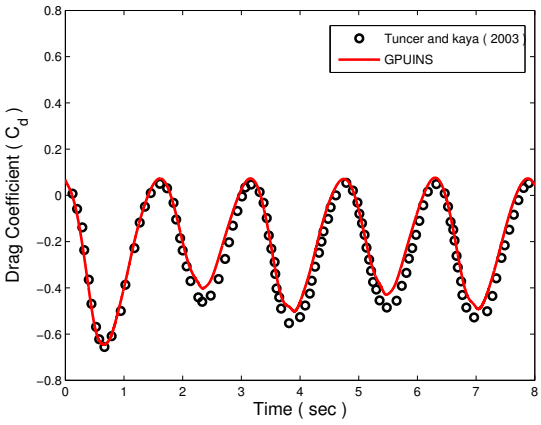


Figure 15. Time history of Drag Coefficient for a Plunging NACA0014 Airfoil at Re 10000

the first thread of each block would sum the respective data local to that block, followed by serial addition over the entire set of blocks. However, it will be seen in the next section that using the *Thrust*[23] library for three-dimensional computations, reductions are much faster due to a binary type reduction algorithm. Computing the Laplacian takes less time than the gradient as there are half the number of atomic operations for the Laplacian than that of the gradient. Further on, in figure(18), we compare the GPU performance for different grid sizes on two different types of grid numbering/arrangement. The number of edges for different grids are also mentioned in the figure (ranging from 0.03 to 1.3 million edges). Figure (19) shows the two types of grids. Although both are made up of triangular elements, grid (a) is constructed using Delaunay triangulation, and grid (b) by dividing a Cartesian grid into triangles. We call grid (b) to have natural

ordering of the nodes, and grid (a) to have quasi-random ordering. The results in figure(18) clearly indicate that random memory access patterns are highly detrimental to GPU performance.

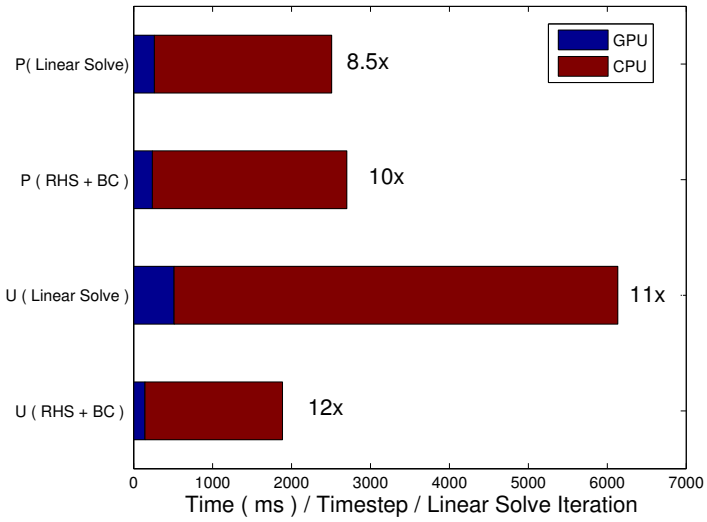


Figure 16. A Comparison of CPU time for Different Functions

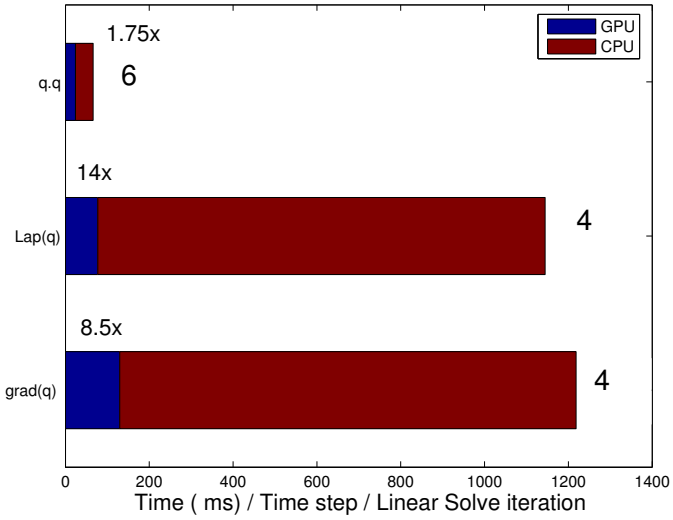


Figure 17. A Comparison of CPU time for Different Kernels

IV.B. Three-Dimensional Computations

IV.B.1. Single Grid

Results are presented for the flow past a sphere at Reynolds numbers 50, 75, and 100 on a Tesla C2070 card. The governing equations are discretized on an unstructured grid made up of prisms (256,000) using 5120 surface elements. At these Reynolds numbers, the wake behind the sphere is stationary, which makes the problem a suitable candidate for preliminary validation. For these Reynolds numbers, we compare the drag coefficient (C_d), polar separation angle (Θ_s), wake width (X_s/d) and the center of the wake $(X_c, Y_c)/d$ with experimental and numerical computations found in Sheard et al.[27] and Johnson and Patel[28] in table(5).

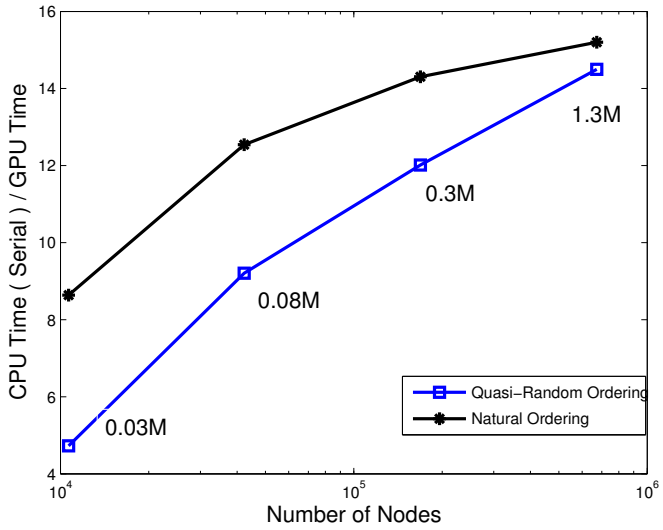
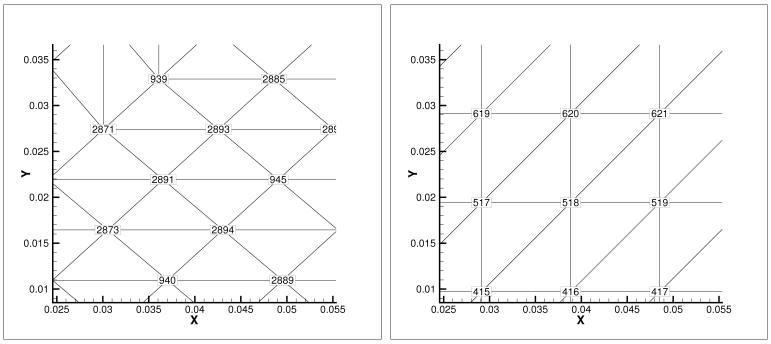


Figure 18. A Comparison of CPU time on Different Grids for Two Types of Grid Arrangements



(a) (b)

Figure 19. (a) Grid with Quasi-random Ordering (b) Grid with Natural Ordering

Reference is made to figure(20) for a description of wake characteristics. Figures(21)-(23) show the pressure contours interspersed with streamlines for different Reynolds numbers. A good correspondence is obtained for most of the wake characteristics giving us confidence in the numerical procedure. Following this comparison, the performance of the GPU is investigated for three different grid sizes having (1) 256,000 (2) 1,024,000 and (3) 2,048,000 cells respectively. The CPU time consumed per linear solver iteration for both the velocity and pressure is computed similar to the two-dimensional computations. A serial version of the GPU code is generated with the highest level of optimization $-O3$ for estimating GPU performance. Figure (24) shows the data corresponding to different grid sizes and different functions. Consistent with two-dimensional computations, the velocity solve results in a higher speed-up compared to the pressure solve due to the Neumann boundary condition for pressure. In general, for a Neumann boundary condition problem, access to the gradients at the boundary result in additional global memory reads which reduce the performance of the GPU. The speed-up figures in the three-dimensional case are marginally lower compared to the two-dimensional case due to a higher serial code optimization. We also compare the performance of individual kernels associated with the solution of the Poisson equation in figure (25). As the *Thrust* library is used for the reduction operations, the dot product operation on the GPU performs substantially faster than the previous implementation for the two-dimensional case. It is however not known exactly at this point why the performance of the Laplacian computation is poor compared to the two-dimensional case.

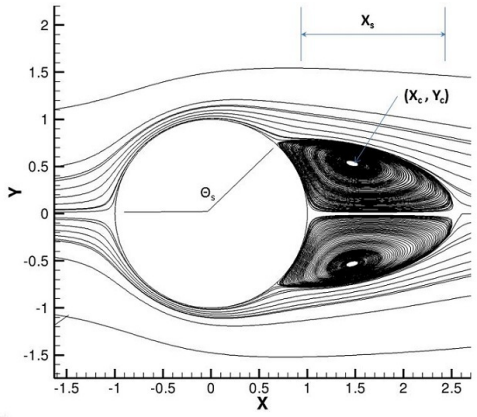


Figure 20. A Description of the Wake Characteristics for the Flow past a Sphere

Table 5. Comparison of Wake Characteristics for different Reynolds numbers with data adapted from References,[27,28]

Re	C_d		X_c/d		Y_c/d		X_s/d		Θ_s	
	Computation	Ref	Computation	Ref	Computation	Ref	Computation	Ref	Computation	Ref
50	1.65	1.62	0.647	0.64	0.215	0.2	0.451	0.4	140.1	139
75	1.31	1.3	0.732	0.7	0.266	0.26	0.78	0.65	132.3	134
100	1.12	1.1	0.811	0.76	0.295	0.3	1.15	1.2	128	129

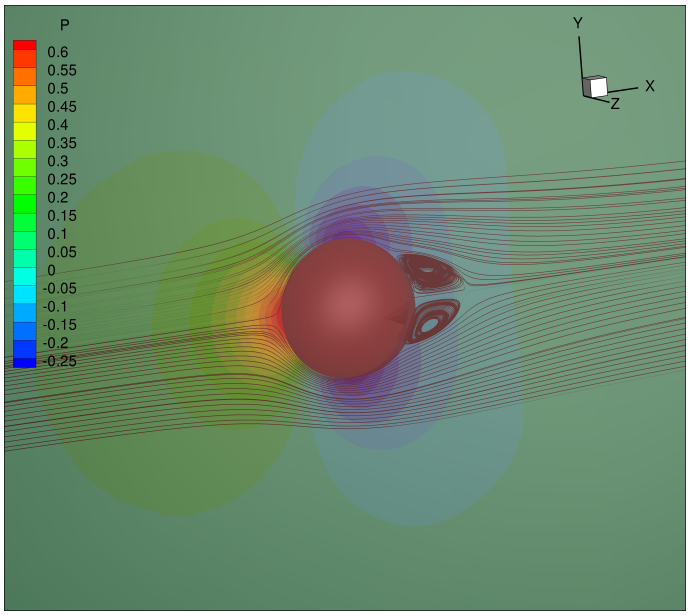


Figure 21. Pressure Contours and Streamlines for the Flow Past a Sphere at Re=50

IV.B.2. Overset Grids

In the overset grid implementation, the grid system consists of a near body unstructured grid overlapping a Cartesian grid over a finite region. Solution of the Poisson equation on overlapping grids is similar to the Schwarz alternating method[29] described below:

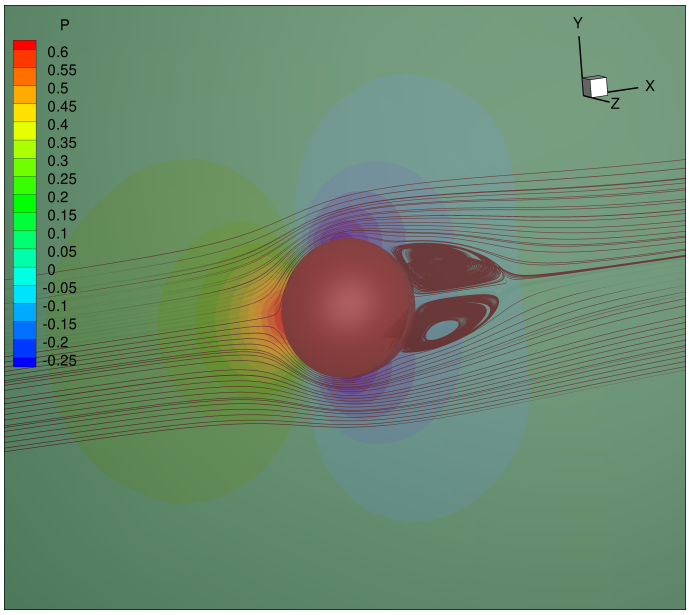


Figure 22. Pressure Contours and Streamlines for the Flow Past a Sphere at Re=75

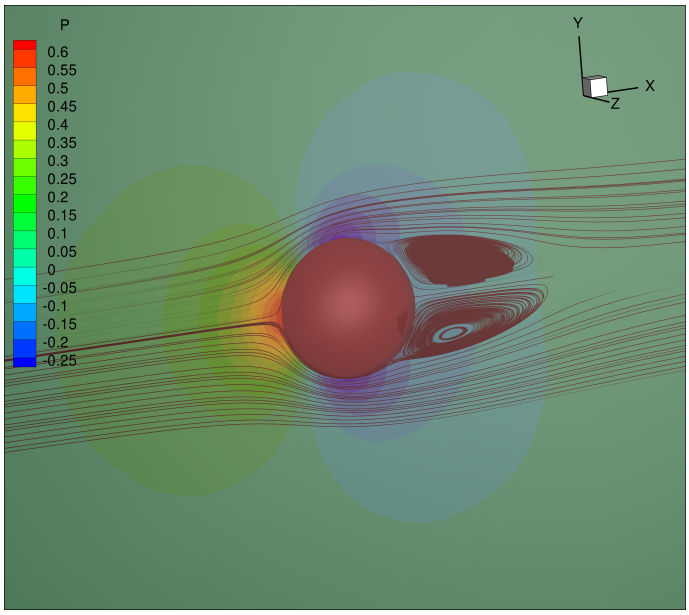


Figure 23. Pressure Contours and Streamlines for the Flow Past a Sphere at Re=100

Listing 2. Solution to the Poisson Equation on Overlapping Domains

```
do < Global Iterate >
 {
   ncon = 0
   for i = 1 to  Ng, Ng is the number of grids
    {
      fixFringePointsFromPotentialDonors();
      fixRHSAtFringePoints();
      ncon += solvePoisson();
    }
 } while ( ncon > Ng )
```

Here *ncon* is the number of iterations for convergence during each Poisson solution for each grid. For each global iterate, we solve the Poisson equation once on each component grids and exchange information at fringe points. If the global solution has converged, then the Poisson equation must converge during the first BiCGSTAB iterate thereby *ncon* takes a value of Ng. Results are presented for the flow past a sphere

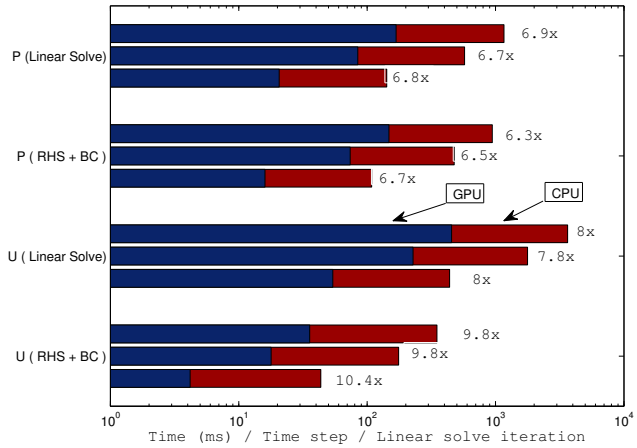


Figure 24. GPU Performance Compared to CPU Execution for Different Functions in Three-Dimensions

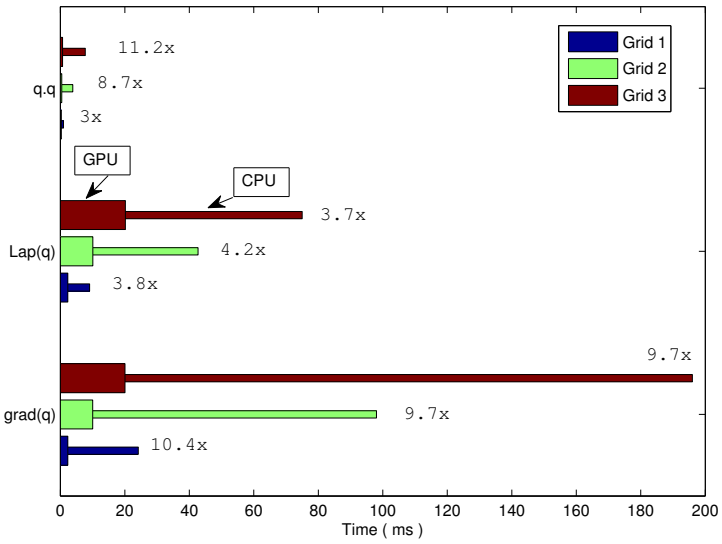
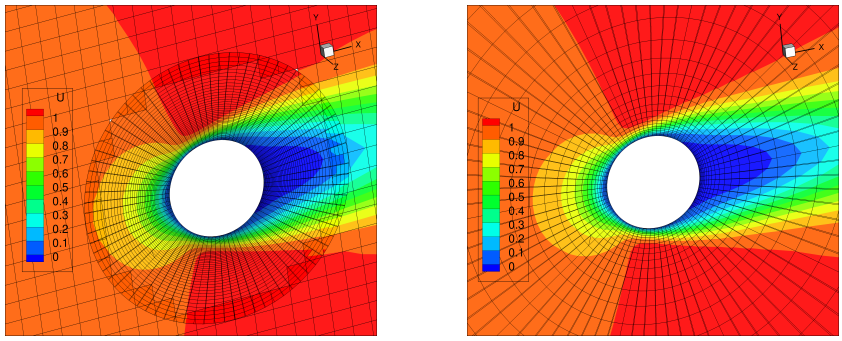


Figure 25. GPU Performance Compared to CPU Execution for Different Kernels in Three-Dimensions

at Re=75 for a grid size of 400,000 cells (256,000 on the sphere). Figure (26) shows the comparison of the x-velocity contours between an overset grid solution and single grid solution. A reasonable comparison is obtained for the grid configuration used. There are minor irregularities in the contours downstream of the sphere in the overlapping region due to a coarse Cartesian grid, which will disappear if the cell sizes commensurate between the Cartesian and the sphere grid in the overlap region. In figure(27), we compare the convergence rates of the drag coefficient between a single grid and an overset grid. It is seen that the overset methodology does not hamper the convergence rates to a large extent, and that both display similar behavior.

V. Conclusions

A GPU based incompressible Navier-Stokes solver has been developed and preliminary results are encouraging and compare well with available data in literature. For two-dimensional cases, only single grid



(a) (b)

Figure 26. Comparison of Velocity contours between the (a) Overset Grid Solution and the (b) Single Grid Solution

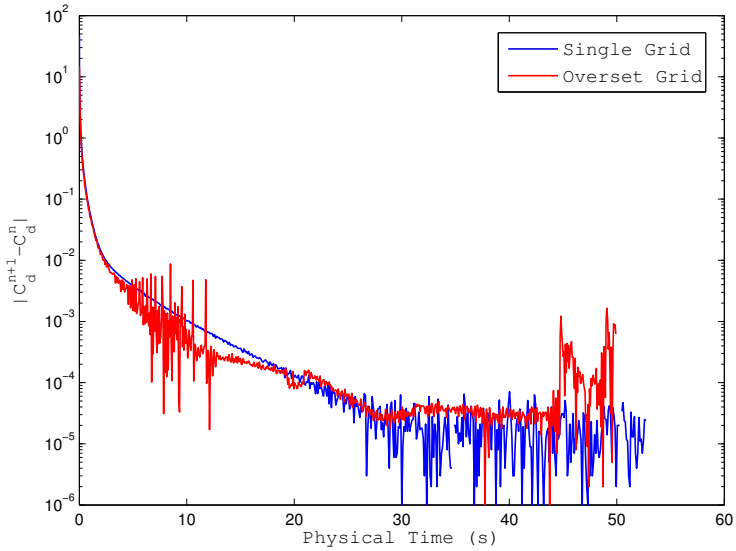


Figure 27. Comparison of Convergence Rates Between Overset Grids and Single Grid for the Flow Past a Sphere at $Re = 75$

computations were performed whereas for three-dimensional cases, both single and overset grid formulations were incorporated. For the overset grid cases, the donor-recipient relationships (domain connectivity) were computed offline before the main flow solver was executed hence this formulation was limited to non-moving cases. The single grid cases on the GPU showed an average speedup of 8x-10x corresponding to the serial code on one processor. It was also demonstrated that one may not be able to reap the full potential of GPUs if the memory access patterns are random, as in the case of unstructured grids. A significant improvement in the speed-up for the reduction operation was obtained using NVIDIA's *Thrust* library. Extension of the capabilities of the current solver to handle multiple moving bodies using overset grids will be sought in the near future. This would also include generating the domain connectivity information during run time within the flow solver. Improvements to the accuracy of the discretization procedure, and implementing a multi-GPU programming model using MPI will also be considered.

Acknowledgments

We gratefully acknowledge support from the Office of Naval Research under ONR Grant N00014-09-1-1060 and NVIDIA for providing their hardware.

References

[1]Van der Vorst, H. A., *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG For the Solution of Nonsymmetric Linear Systems*. SIAM Journal on Scientific and Statistical Computing, Vol. 13, pp.631-644, 1992.

[2]Chandar, D.D.J., Sitaraman, J., and Mavriplis, D., *CU++ET: An Object Oriented Tool for Accelerating Computational Fluid Dynamics Codes Using Graphical Processing Units*, 20^{th} AIAA Computational Fluid Dynamics Conference, Honolulu, HI, 2011.

[3]NVIDIA CUDA C programming Guide 3.1, *http : //developer.nvidia.com/object/cuda_3_1_downloads.html*

[4]Hagen, T.R., Lie, K-.,A and Natvig, J.R., *Solving the Euler Equations on Graphics Processing Units*, Lecture Notes in Computer Science, 3994, pp. 220-227, 2006.

[5]Elsen, E., LeGresley, P., and Darve, E., *Large Calculation of the Flow over a Hypersonic Vehicle using a GPU*, Journal of Computational Physics, Vol. 227, No. 24, pp. 10148-10161, 2008.

[6]Brandvik, T., and Pullan, G., *Acceleration of a 3D Euler Solver using Commodity Graphics Hardware*, 46^{th} AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2008-0607, Reno, NV, 2008.

[7]Cohen, J.M., and Molemaker, M.J., *A Fast Double Precision Code using CUDA* , Proceedings of Parallel CFD, Moffett Field, CA, 2009.

[8]Phillips, E.H., Zhang, Y., Davis, R.L., and Owens, J.D., *Rapid Aerodynamic Performance Prediction on a Cluster of Graphics Processing Units*, 47^{th} Aerospace Sciences Meeting and Exhibit, AIAA-2009-0565, Orlando, FL, 2009.

[9]Corringan, A., Camelli, F., Lohner, R., and Mut, F., *Semi-Automatic Porting of a Large-Scale Fortran CFD Code to GPUs*, International Journal for Numerical Methods in Fluids, Vol. 66, No.6, 2011.

[10]Sitaraman, J., Floros, M., Wissink, A., and Potsdam, M., *Parallel Domain Connectivity Algorithm for Unsteady Flow Computations Using Overlapping and Adaptive Grids*, Journal of Computational Physics, Vol. 229, No. 12, pp. 4703-4723, 2010.

[11]Tuncer, I.H., and Kaya, M., *Thrust Generation Caused by Flapping Airfoils in a Biplane Configuration*, Journal of Aircraft, Vol. 40, pp.509-515, 2003.

[12]Chandar, D.D.J., and Damodaran, M., *Computation of Unsteady Low Reynolds Number Free-Flight Aerodynamics of Flapping Wings*, Journal of Aircraft, Vol. 47, pp.141-150, 2010.

[13]Soni, K., Chandar, D.D.J., and Sitaraman, J., *Development of an Overset Grid Computational Fluid Dynamics Solver on Graphical Processing Units*, 49^{th} AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2011-1268, Orlando, FL.

[14]Gresho, P.M., *Some Current CFD Issues Relevant to the Incompressible Navier-Stokes Equations*, Computer Methods in Applied Mechanics and Engineering, Vol. 87, pp.201-252, 1991.

[15]Brown, D.L., Cortez, R., and Minion, M.L., *Accurate Projection Methods for the Incompressible Navier-Stokes Equations*, Journal of Computational Physics, Vol. 168, pp.464-499, 2001.

[16]Minion, M.L., and Brown, D.L., *Performance of Under-Resolved Two-Dimensional Incompressible Flow Simulations, II*, Journal of Computational Physics, Vol. 138, pp.734-765, 1997.

[17]Henshaw, W.D., *A Fourth Order Accurate Method for the Incompressible Navier-Stokes Equations on Overlapping Grids*, Journal of Computational Physics, Vol. 113, pp.13-25, 1994.

[18]Abdallah, S., *Numerical Solutions for the Pressure Poisson Equation with Neumann Boundary Conditions Using a Non-Staggered Grid, I*, Journal of Computational Physics, Vol. 70, pp.182-192, 1987.

[19]Crumpton, P.I., Moinier, P., and Giles, M.B., *An Unstructured Algorithm for High Reynolds Number Flows on Highly Stretched Grids*, Numerical Methods in Laminar and Turbulent Flow, pp.561-572, Pineridge Press, 1997.

[20]Abdallah, S., *Numerical Solutions for the Incompressible Navier-Stokes Equations in Primitive Variables Using a Non-Staggered Grid, II*, Journal of Computational Physics, Vol. 70, pp. 193-202, 1987.

[21]Henshaw, W.D., Kreiss, H.O., and Reyna, L.G., *On the Smallest Scale for the Incompressible Navier-Stokes Equations*, ICASE Report 88-8, 1988.

[22]Henshaw, W.D., *Cgins Reference Manual: An Overture Solver for the Incompressible Navier-Stokes Equations on Composite Overlapping Grids*, Lawrence Livermore National Laboratory Report LLNL-SM-455871, 2011.

[23]http://research.nvidia.com/news/thrust-cuda-library

[24]Ghia, U., Ghia, N., and Shin, C.T., *High-Re Solutions for Imcompressible Flow Using the Navier-Stokes Equations and a Multigrid Method*, Journal of Computational Physics, Vol. 28, pp.387-411, 1982.

[25]Jones, K.D., Dohring, C.M., and Platzer, M.F., *Experimental and Computational Investigation of the Knoller-Betz Effect*, AIAA Journal, Vol. 36, No. 7, pp.1240-1246, 1998.

[26]Tuncer, I.H., and Kaya, M., *Thrust Generation Caused by Flapping Airfoils in a Biplane Configuration*, Journal of Aircraft, Vol. 40, pp.509-515, 2003.

[27]Sheard, G.J., Hourigan, K., and Thompson, M.C., *Computation of the Drag Coefficients for Low-Reynolds-Number Flow Past Rings*, Journal of Fluid Mechanics, Vol. 526, pp.257-275, 2005.

[28]Johnson, T.A., and Patel, V.C., *Flow Past a Sphere up to a Reynolds Number of 300*, Journal of Fluid Mechanics, Vol. 378, pp.19-70, 1999.

[29]Schwartz, H.A. *ber einen Grenzbergang durch alternierendes Verfahren", Vierteljahrsschrift der Naturforschenden Gesellschaft in Zrich*, Vol.15, pp. 272286, 1870.