Development of a Performance Portable Compressible Flow Solver on Structured Curvilinear Grids

Andrew Bodling
Research Scientist
Science and Technology Corporation
Moffett Field, CA, USA

George Zagaris
Research Scientist
Air Force Research Laboratory, Wright-Patterson AFB, OH, USA

Dylan Jude
Research Scientist
Science and Technology Corporation
Moffett Field, CA, USA

Jayanarayanan Sitaraman
U.S. Army Combat Capabilities Development Command
Aviation & Missile Center, Moffett Field, CA, 94035, USA

Shirzad Hosseinverdi
Research Scientist
Science and Technology Corporation
Moffett Field, CA, USA

A structured curvilinear grid based implicit compressible Navier-Stokes flow solver (G3D) is developed using a performance portable framework. The accuracy of the flow solver is verified by comparing it against peer simulations and experimental data for canonical test cases, including inviscid transonic flow around a NACA0012 airfoil and both steady and unsteady flows around a cylinder. Standard structured grid reconstructions schemes such as 3rd order (MUSCL) and 5th order (WENO5) are implemented, tested and verified. The performance portable code executes seamlessly on both CPUs and GPUs, using the same binary executable with run-time selection of the execution device. Execution on 1 A100 GPU is shown to yield a 37x speedup over execution on one CPU socket using up to 64 OpenMP threads. However, OpenMP scaling is found to be unsatisfactory, with execution time nearly plateauing beyond 16 threads. The execution time per time step and total convergence time are compared with other GPU-enabled flow-solvers. These comparisons are performed for a 18 million 3-D grid for the transonic ONERA M6 wing. Performance results obtained show favorable trends and show good agreement with those obtained with other GPU based structured and unstructured codes.

I. Introduction

The current trend in High Performance Computing (HPC) platforms has shifted to large scale use of accelerators such as Graphical Processing Units (GPUs). Computing paradigms have traditionally relied on distributed computing platforms consisting of Central Processing Units (CPUs). Domain decomposition using the Message Passing Interface (MPI) formed the backbone of most high performance computing codes. In contrast, GPUs support fine-grain parallelism, where a large number of threads are utilized in place of single-threaded (or small number of threads) execution. A single GPU thread is not as performant as an equivalent CPU thread, but the very large number of GPU threads possible

compensates and make the GPUs much more attractive for high performance computing simulations. Recent studies [1] have demonstrated that GPUs can obtain equivalently accurate simulations using 20% less power compared to the CPU resources required to obtain the equivalent performance.

In addition, GPU accelerators from different vendors with differing architectures may be used. Each vendor recommends using vendor specific programming language to achieve maximum performance on their devices. Implementation using specific programming models often leads to vendor-locks and difficulty in porting the same code base to another architecture. Furthermore, for application developers who are mostly domain scientists, maintaining different code bases for each architecture is difficult. In some cases, where there are large code bases, it is simply intractable. An attractive alternative in this context is the usage of performance portable frameworks, where a single-source multi-physics code can be constructed such that it executes seamlessly on underlying platforms, insulating application developers from underlying architecture and programming models. The U.S Department of Energy has spent significant resources developing performance portable frameworks such as RAJA [2], Kokkos [3, 4] and OCCA [5]. However, it is well known that the usage of these performance portable frameworks can have overheads in execution and may require further tuning of implementation to achieve similar performance as vendor-specific implementations.

Solution to Navier-Stokes equations in an Eulerian reference frame requires grids for discretization of the governing partial differential equations. The first computational fluid dynamics (CFD) techniques for Navier-Stokes equations utilized structured curvilinear grids and were applied to airfoil cross sections typical to wings, rotor blades, turbine blades etc. The approach for using overset structured curvilinear grids have since remained popular and are one of most efficient and accurate (both attributable to the structure of grids) methods for solutions to the Navier-Stokes equations. For example NASA OVERFLOW [6] and CFL3D [7] codes have pioneered this approach and are being widely used currently owing to their efficiency. Structured curvilinear grids have been attractive because of their smoothness and ability for facilitating high-resolution line-based discretization schemes and associated linear-solver algorithms. On the other hand, generating structured curvilinear grids for complex geometry require fair amount of manual effort. In this context, the HPCMP CREATE A/V Helios [8] is a multi-solver framework that provides the flexibility of combining easily generatable unstructured/strand meshes for complex geometry with simpler geometry that are meshed using structured curvilinear grids. The various solvers are connected using a domain connectivity module that appropriately cuts, masks and creates interpolation patterns between all participant mesh systems. Currently, Helios supports semi-structured strand [9] based solvers, unstructured flow solvers [1, 10, 11] and structured grid flow solvers [6, 12] for resolving near-wall flows. Away from the wall high-order Cartesian grid based solvers [13] are used to provide accurate tracking for wakes and vortices. In previous work, the Helios team has created performance portable implementations of the unstructured near-body solver [1, 14], the overset domain connectivity solver [15] and the off-body Cartesian solver [13]. To reiterate, structured curvilinear grid and strand grid based solution techniques are generally more efficient and accurate compared to the unstructured grid counterparts. These advantages in accuracy and efficiency come from using line-based methods for flow field gradients and linear solvers that leverage the grid structure. However, it is important to ensure that performance portable techniques for implementing structured curvilinear solvers can maintain the efficiency and accuracy of solvers implemented with vendor-specific methods.

Given this context, this paper has two objectives. The first objective is to explore, develop, test and validate an implicit, structured grid, flow solver using Mint [16], which provides a performance portable mesh data and execution model. The second objective is to show successful execution and accuracy on both CPUs and GPUs with performance comparisons with equivalent simulations from other flow solvers. The overarching goal is to explore and demonstrate a new performance programming paradigm that can be sustainable for future development of large code bases.

II. Numerical Methodology

G3D was designed based off of the CUDA implementation of the GPU-Accelerated Rotor Flow Field (GARFIELD) solver [12]. GARFIELD solves a discretization of the Reynolds-Averaged Navier Stokes (RANS) equations with a Spalart-Allmaras (SA) turbulence model on structured, curvilinear grids. GARFIELD uses a mix of finite-volume and finite-difference methods, with all fluid quantities stored at the center of grid cells. The main solver routines from GARFIELD were used with the CUDA implementation replaced with Mint. G3D solves the full Navier-Stokes equations, which written in strong conservation form are,

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x} + \frac{\partial \mathbf{G}_i}{\partial y} + \frac{\partial \mathbf{H}_i}{\partial z} = \frac{\partial \mathbf{F}_v}{\partial x} + \frac{\partial \mathbf{G}_v}{\partial y} + \frac{\partial \mathbf{H}_v}{\partial z} + \mathbf{S} \tag{1}$$

where $\mathbf{Q}$ is the vector of conserved variables,

$$\mathbf{Q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{pmatrix} \tag{2}$$

with ρ, u, v, w and e denoting the local flow density, three components of velocity and total energy, respectivly. The inviscid fluxes $\mathbf{F}_i$, $\mathbf{G}_i$, $\mathbf{H}_i$ are,

$$\mathbf{F}_i = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E+p)u \end{pmatrix}, \quad \mathbf{G}_i = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ (E+p)v \end{pmatrix}, \quad \mathbf{H}_i = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ (E+p)w \end{pmatrix} \tag{3}$$

where p is the local pressure. The viscous fluxes $\mathbf{F}_v$, $\mathbf{G}_v$, $\mathbf{H}_v$ are,

$$\mathbf{F}_v = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} - q_x \end{pmatrix}, \quad \mathbf{G}_v = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} - q_y \end{pmatrix}, \quad \mathbf{H}_v = \begin{pmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} - q_z \end{pmatrix} \tag{4}$$

.

To compute the inviscid flux terms $\mathbf{F}_i$, $\mathbf{G}_i$, and $\mathbf{H}_i$, the flux at the faces must be found. This is performed in two steps. The fluxes are functions of the conservative or primitive variables. The first step is to reconstruct the value of the flow variables at the face in a way that allows for discontinuities. Three different reconstruction schemes have been implemented into G3D; these schemes are first order (FO) interpolation, 3rd order accurate Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) [18] and the 5th order accurate Weighted Essentially Non Oscillatory (WENO) scheme [19].

After reconstruction, values on the left and right sides of the interface are combined into a single flux using the Roe Riemann solver [20]. Finally, the conserved variable is updated using the computed numerical fluxes. In G3D, the conserved variable is updated either with the explicit Low-storage 3-stage Runge Kutta method or the Diagonalized Alternating Direction Implicit [21] time integration method.

To solve the Reynolds-Averaged Navier-Stokes (RANS) equations, the turbulent eddy viscosity μ_{turb} must be found. The Spalart-Allmaras Reynolds-Averaged Navier Stokes (SA-RANS) turbulence model [22] is used to approximate the value of the turbulent eddy viscosity μ_{turb} using transport properties. The model uses empirical constants based on experimental results for turbulence over a flat plate. The laminar kinematic viscosity, $\nu = \frac{\mu}{\rho}$, is used along with the model variable $\tilde{\nu}$:

$$\frac{D\tilde{\nu}}{Dt} = c_{b1}\tilde{S}\tilde{\nu} - c_{w1}f_w\left(\frac{\tilde{\nu}}{d}\right)^2 + \frac{1}{\sigma}[\nabla\cdot((\nu+\tilde{\nu})\nabla\tilde{\nu}) + c_{b2}(\nabla\tilde{\nu})^2] \tag{5}$$

where the turbulent eddy kinematic viscosity is

$$\nu_t = \tilde{\nu}f_{v1}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu} \tag{6}$$

and other variables are

$$\tilde{S} = S + \frac{\tilde{\nu}}{\kappa^2 d^2}, \quad f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}}, \quad f_w = g\left(\frac{1+c_{w3}^6}{g^6 + c_{w3}^6}\right)^{1/6}, \quad g = r + c_{w2}(r^6 - r), \quad r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \tag{7}$$

and constants are defined as

$$c_{b1} = 0.135, \quad c_{b2} = 0.622, \quad \sigma = \frac{2}{3}, \quad \kappa = 0.41, \quad c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}, \quad c_{w2} = 0.3, \quad c_{w3} = 2, \quad c_{v1} = 7.1 \tag{8}$$

Throughout the manuscript, the verification and validation of the inviscid and viscous flux terms as well as the SA-RANS turbulence model, will be demonstrated.

III. GPU Implementation

G3D uses Mint [16], which provides the fine-grain execution model that underpins the development of the kernels for the Finite Volume discretization. Mint may use either the Kokkos [3, 4] or RAJA [2] performance portability libraries for kernel launch and execution on the GPU using CUDA, or, using any of the other backends that are available through RAJA or Kokkos. At compile-time, the performance portability library to use is selected.

Mint's fine-grain parallel execution model serves as an intermediate layer that provides an application-oriented and physics-focused API that insulates computational scientists and engineers from the programming details and nuances for performance portability. In addition, Mint does not restrict the application to a particular software stack. This makes the code flexible and prevents developers from needing to change the code in the future as technologies emerge and programming languages evolve. The parallel execution model consists of the following main components:

1) Memory Management & Memory Spaces
2) Execution Spaces
3) Data Parallel Primitive Operations, and
4) Mesh Traversal Operators

Memory Management is facilitated by leveraging the Umpire [23] open-source library, which provides functionality to conveniently manage memory resources on heterogeneous platforms. Some of the memory operations in Mint are allocate, deallocate, copy, move, etc. These methods can optionally take a handle to the target memory space, e.g., host CPU memory or global device memory on the GPU and delegate the request to the Umpire library.

An Execution Space is a required template argument for the data-parallel primitive operations and mesh traversal operator routines. How and where the kernel is executed is specified with the execution space. This programming idiom facilitates the implementation of kernels that are born parallel and portable and enables the development of a single-source, performance portable code. Depending on which performance portability library is used, the top-level execution space that Mint defines is mapped to a corresponding RAJA execution policy or Kokkos execution space accordingly. Then, the appropriate construct, corresponding to the execution pattern, e.g., `kokkos::parallel_for`, `RAJA::forall`, etc., is used to launch and execute the kernel.

Mint also has a set of data-parallel primitive operations such as sort, scan, atomics, etc. These are routines that are templated on execution space and provide the key ingredients for developing data-parallel algorithms. The data-parallel primitives serve as the foundation for the implementation of Mesh Traversal Operators. The Mesh Traversal Operators consist of a set of routines that follow the following guidelines:

- The name of the routine starts with `for_all_` prefix.
- The suffix in the name of the routine indicates the mesh entity that is traversed or iterated. For example, `for_all_nodes`, iterates over the nodes of the mesh, while, `for_all_cells`, iterates over the mesh cells. For G3D, only the generic `for_all` routine was used, which essentially creates a nested for-loop based off of the specified minimum and maximum range of indices.
- The routines typically accept three primary arguments:
    1) A string corresponding to the desired name for the kernel being launched (can be used by profiling applications).
    2) A pointer to the mesh object (not used for G3D).
    3) The kernel, defined in-line with a C++ lambda capture.
- All routines are templated on execution space, which is provided as the first template argument.
- An optional second template argument may be provided, using the `xargs::*` idiom, to indicate any extra arguments or information that should be supplied to the kernel when it's launched.

As an illustration of the usage of Mint's execution model, the code sample in Fig. 1 shows an example of a kernel that adds the flux to the residual. The kernel loops over the range of indices from `j = num_ghost_cells to jmaxloop-1`, `k = num_ghost_cells to kmaxloop-1` and `l = num_ghost_cells to lmaxloop-1`. In general,

a one, two and three dimensional nested for-loop can be used. The `mint::ExecutionSpace exec` argument specifies the execution space for the kernel. This argument is selected at runtime and specifies whether the for-loop is run on the GPU or CPU. This allows a single source code to be used for running on both CPUs and GPUs. Inside the kernel, the residual array `res` is updated based on the `j+1` and `j` flux values (or `k+1` and `k` or `l+1` and `l`), which is captured by the lambda capture, denoted by the `MINT_LAMBDA`. The kernel assumes that the captured array pointers all point to a memory space that is accessible in the given execution context.

The Mesh Traversal Operators, in conjunction with the data-parallel primitives, memory management API and execution space definitions are used to compose the kernels for the physics algorithms. This approach enables development of a single-source, performance portable code.

```
void StructuredSolution::AddFluxToRhs(int dir,double *flux){
  int jstride = jstride_;
  int kstride = kstride_;
  int lstride = lstride_;
  int num_cells = num_cells_;
  mint::index_t num_ghost_cells = num_ghost_cells_;

  double *res = res_;

  mint::index_t jmaxloop = jtot_ - num_ghost_cells;
  mint::index_t kmaxloop = ktot_ - num_ghost_cells;
  mint::index_t lmaxloop = ltot_ - num_ghost_cells;

  mint::for_all< 3 >(
    "add flux to rhs", exec,
    {num_ghost_cells,num_ghost_cells,num_ghost_cells},
    {jmaxloop,kmaxloop,lmaxloop},
    MINT_LAMBDA(mint::index_t j,mint::index_t k, mint::index_t l) {

      int idx = j*jstride + k*kstride + l*lstride;
      int idx_p1 = (idx+jstride)*(dir==1) + (idx+kstride)*(dir==2) + (idx+lstride)*(dir==3);

      res[idx            ] +=  flux[idx_p1            ] - flux[idx            ];
      res[idx+num_cells  ] +=  flux[idx_p1+num_cells  ] - flux[idx+num_cells  ];
      res[idx+num_cells*2] +=  flux[idx_p1+num_cells*2] - flux[idx+num_cells*2];
      res[idx+num_cells*3] +=  flux[idx_p1+num_cells*3] - flux[idx+num_cells*3];
      res[idx+num_cells*4] +=  flux[idx_p1+num_cells*4] - flux[idx+num_cells*4];
    }
  );
}
```

Fig. 1 Sample kernel for adding flux to residual

IV. Results

The results presented herein include accuracy verifications for 4 different canonical test cases.

1) Steady, transonic 2-D inviscid airfoil
2) Uniform steady/unsteady flow past a 2-D circular cylinder
3) Steady, 2-D, NACA 0012 with SA-RANS turbulence modeling
4) Steady, ONERA M6 with SA-RANS turbulence modeling

The other solvers used in this work to compare accuracy and efficiency are:

1) GARFIELD [12], a structured grid curvilinear flow solver developed using vendor-specific (NVIDIA Compute Unified Device Architecture (CUDA)) programming paradigm

2) RAPIDUS [1], an unstructured cell-centered solver developed using the performance portable frameworks MINT [16] and OCCA [5]
3) NASA OVERFLOW [17] 2.5`beta_rc3` that utilizes OpenACC and NVIDIA CUDA for GPU execution.

A. Steady, Inviscid Flux Validation, NACA 0012

The Euler equations with the various reconstruction methods were validated by applying them to an inviscid transonic (Mach number $M_\infty = 0.8$, angle of attack $\alpha = 1.25°$) NACA 0012 airfoil and comparing the results to the calculations from Jameson *et al.* [24]. The NACA 0012 structured O-grid used for the validations is shown in Fig. 2. The grid has 187 and 105 points in the circumferential and wall normal directions, respectively. G3D and GARFIELD were run with the MUSCL and WENO scheme and the DADI time integration method. G3D and GARFIELD will be used with the DADI time integration method throughout the entire paper. The predicted pressure coefficient C_P distribution for the various reconstruction methods compared to the calculations from Jameson is shown in Fig. 3. The C_P distribution agrees well among all the predictions. Figure 4 shows the force comparisons between G3D (solid lines) and GARFIELD (dashed lines) while the Jameson prediction is in black dashed lines/markers. G3D and GARFIELD predict identical results, as expected. Also, both solvers agree well with the predictions from Jameson.

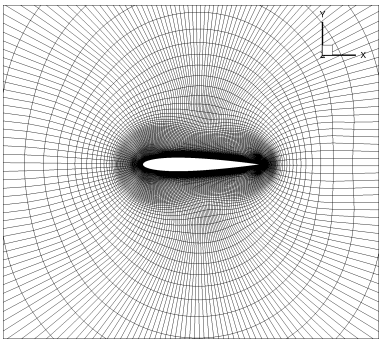


Fig. 2 NACA 0012 structured O-grid

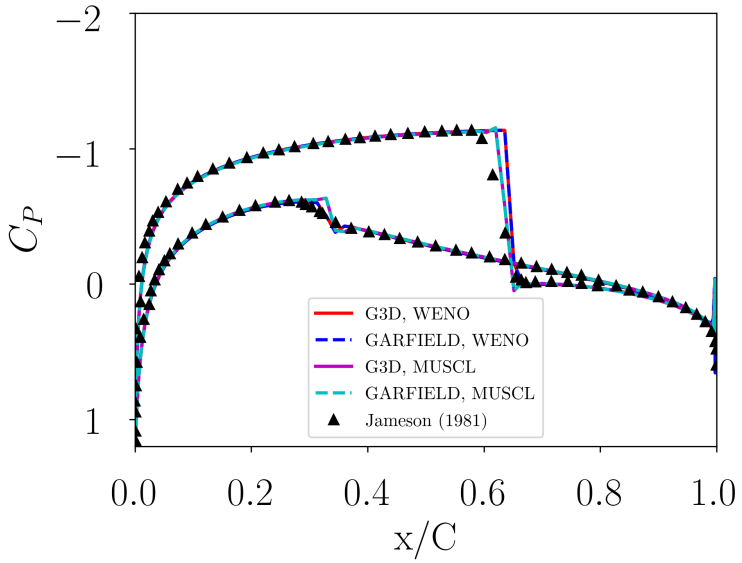
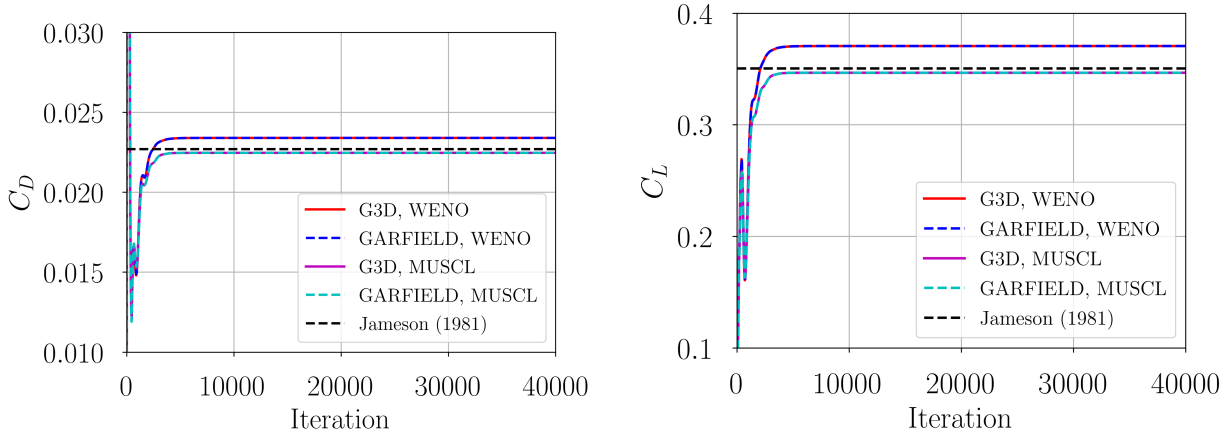


Fig. 3 Comparison of C_P distribution predicted by G3D and GARFIELD.

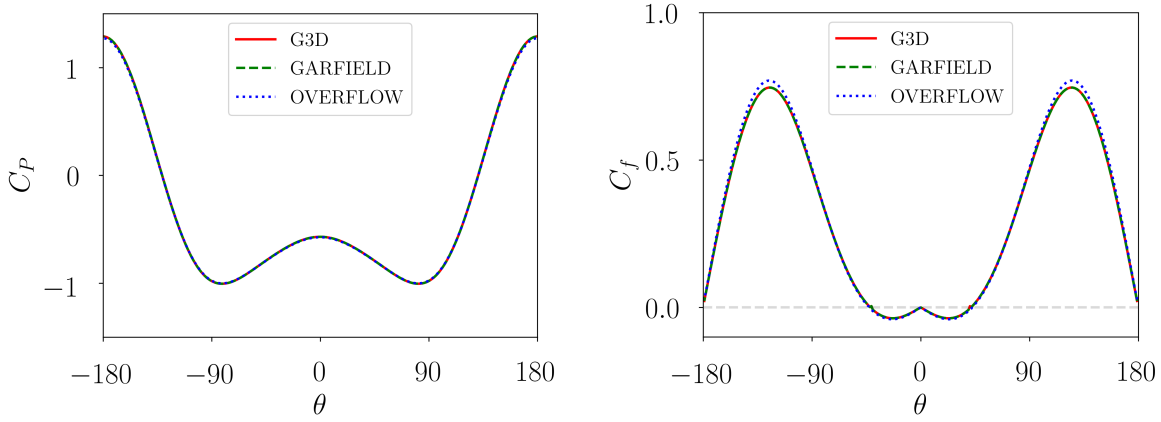


(a) C_D (b) C_L

Fig. 4 Comparison of C_D (a) and C_L (b) predicted by G3D and GARFIELD.

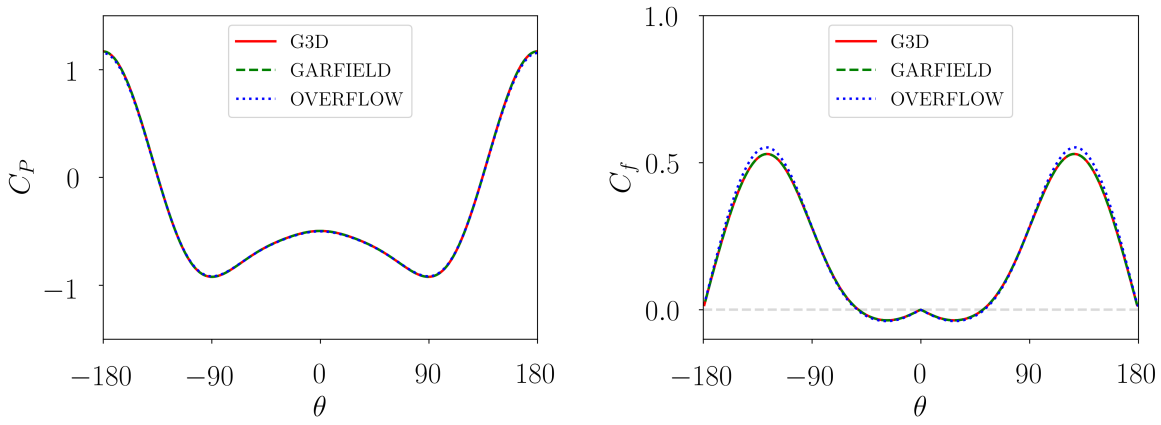
B. Steady, Viscous Flux Validation, 2-D Cylinder

To validate the viscous flux routines, G3D was tested with uniform steady flow past a circular cylinder at $Re = 20$ and 40, unsteady flow at $Re = 100$ and 200, and impulsively started flow at a relatively large $Re = 1000$. Here, the Reynolds number, Re, is defined based on the cylinder diameter, D. The outer boundary, r_{max}, is located $25D$ from the center of the cylinder for $Re \leq 200$ and $r_{max} = 10D$ for $Re = 1000$. A structured O-mesh with the same grid resolution, 257×257, is used for all cases. The grid points are exponentially displaced from the wall in radial direction with the minimum wall-normal distance of $0.01D$ for $Re \leq 200$ and $0.005D$ for $Re = 1000$. For all cases, the freestream Mach number is set to $M_\infty = 0.25$. Results from G3D, GARFIELD, and OVERFLOW are compared in Figs. 5 and 6 for the steady flow regime for Reynolds numbers $Re = 20$ and $Re = 40$. With G3D and GARFIELD the MUSCL scheme is used while with OVERFLOW, the 5th order central scheme is used. Currently only the central scheme is implemented in the GPU version of OVERFLOW, so this scheme will be used throughout the entire paper, except for when published OVERFLOW predictions are used. Unless specified otherwise, the 5th order central scheme is used in OVERFLOW. In addition, the ARC3D diagonalized Beam-Warming scalar pentadiagonal scheme is used with OVERFLOW, which will also be used throughout the entire paper. Comparing the C_P and C_f distribution around the cylinder for the three difffernent flow solvers, there are minimal differences observed. Figure 7 compares the length of the standing eddy behind the cylinder l, separation angle θ, drag coefficient C_D and lift coefficient C_L for OVERFLOW, G3D, GARFIELD, and RAPIDUS. The various quantities agree well among all the flow solvers.



(a) C_P (b) C_f

Fig. 5 C_P (a) and C_f (b) distribution around the cylinder for $Re = 20$. G3D and GARFIELD are computed using the MUSCL scheme.



(a) C_P (b) C_f

Fig. 6 C_P (a) and C_f (b) distribution around the cylinder for $Re = 40$. G3D and GARFIELD are computed using the MUSCL scheme.

$Re = 20$					
	l	θ_1	θ_2	C_L	C_D
G3D	0.931	-40.76	40.76	-1.33E-10	2.026
GARFIELD	0.931	-40.76	40.76	-1.33E-10	2.026
OVERFLOW	0.930	-42.53	42.53	1.45E-18	2.04
RAPIDUS	0.94	-43.5	43.5	1.00E-14	2.08

$Re = 40$					
	l	θ_1	θ_2	C_L	C_D
G3D	2.324	-51.92	51.92	-5.10E-11	1.519
GARFIELD	2.320	-51.92	51.92	-5.07E-11	1.519
OVERFLOW	2.318	-52.58	52.58	1.37E-17	1.53
RAPIDUS	2.310	-53.5	53.5	1.00E-14	1.56

Fig. 7 Comparison of the length of the standing eddy behind the cylinder l, separation angle θ, drag coefficient C_D and lift coefficient C_L for OVERFLOW, G3D, GARFIELD, and RAPIDUS.

C. Unsteady, Viscous Flux Validation, 2-D Cylinder

Table 1 shows the predicted Strouhal number St, drag coefficients C_D (mean and amplitude), and lift coefficient C_L (amplitude) for unsteady flow past a circular cylinder with $Re = 100$ and 200. G3D and GARFIELD, using MUSCL and WENO, are compared to RAPIDUS, OVERFLOW and the literature. Good agreement is observed among all the solvers and literature.

Next, the impulsively started flow is considered for a circular cylinder at $Re = 1000$. At an early stage the flow will develop symmetrically about $H = 0$ and the wake will still be laminar and two-dimensional. The transient solution computed up to a non-dimensional time, $T = tu_\infty/D$, of 6 is illustrated in Fig. 8 using contour lines of the spanwise vorticity. For this comparison, the MUSCL scheme is used with G3D and GARFIELD (WENO has similar results). All flow solvers agree well throughout time. The surface vorticity distribution at various time instances is shown in Fig. 9; figure 9 (a) compares RAPIDUS versus Qian and Vezza and Fig. 9 (b) compares G3D versus Qian and Vezza. Lines correspond to results from RAPIDUS and G3D and lines with markers are from Qian and Vezza [25]. Both G3D and RAPIDUS agree well with the predictions from Qian and Vezza [25].

	Re = 100			Re = 200		
	St	C_D	C_L	St	C_D	C_L
G3D, MUSCL	**0.160**	**1.35 ± 0.009**	**±0.32**	**0.192**	**1.36 ± 0.044**	**±0.68**
GARFIELD, MUSCL	0.160	1.35 ± 0.009	±0.32	0.192	1.36 ± 0.044	±0.68
G3D, WENO	**0.160**	**1.35 ± 0.015**	**±0.32**	**0.192**	**1.36 ± 0.048**	**±0.68**
GARFIELD, WENO	0.160	1.35 ± 0.015	±0.32	0.192	1.36 ± 0.046	±0.68
OVERFLOW, 5th Order	0.160	1.36 ± 0.009	±0.33	0.192	1.37 ± 0.045	±0.69
RAPIDUS	0.166	1.38 ± 0.008	±0.34	0.196	1.39 ± 0.049	±0.70
Russell and Wang	0.169	1.38 ± 0.007	±0.30	0.195	1.29 ± 0.022	±0.50
Linnick and Fasel	0.166	1.34 ± 0.009	±0.33	0.197	1.34 ± 0.044	±0.69
Liu et al.	0.164	1.35 ± 0.012	±0.34	0.192	1.31 ± 0.049	±0.69
Ding et al.	0.164	1.33 ± 0.008	±0.28	0.196	1.33 ± 0.045	±0.60
Hosseinverdi and Fasel	0.166	1.32 ± 0.007	±0.24	0.195	1.33 ± 0.044	±0.63

Table 1 Unsteady flow past a circular cylinder for Re = 100 & Re = 200: Strouhal number St, drag coefficients C_D (mean and amplitude), and lift coefficient C_L (amplitude).

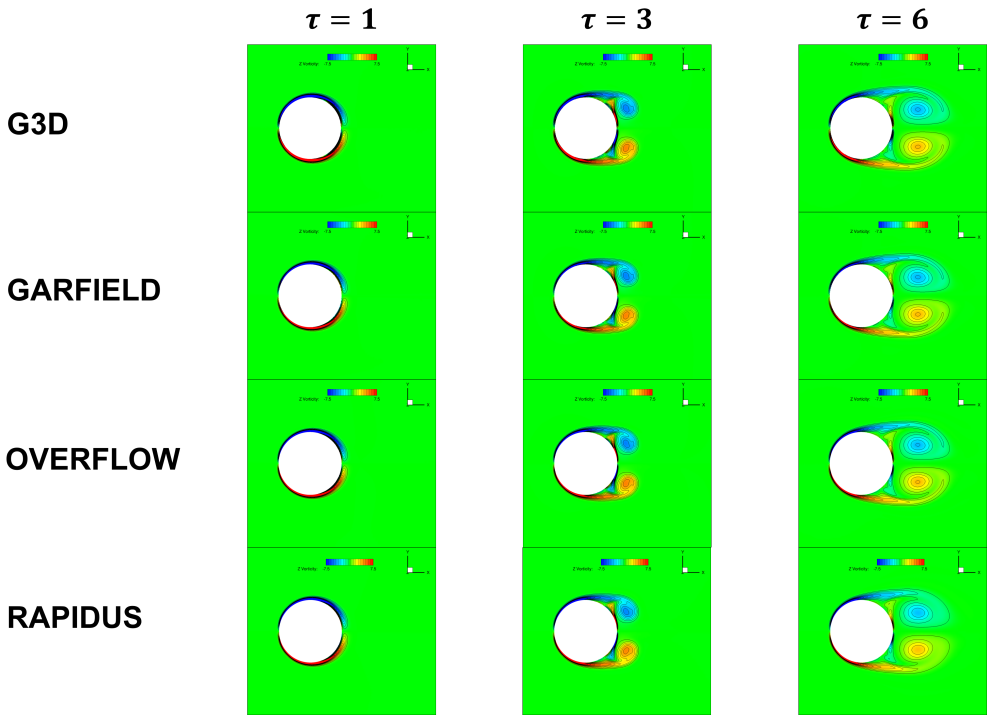
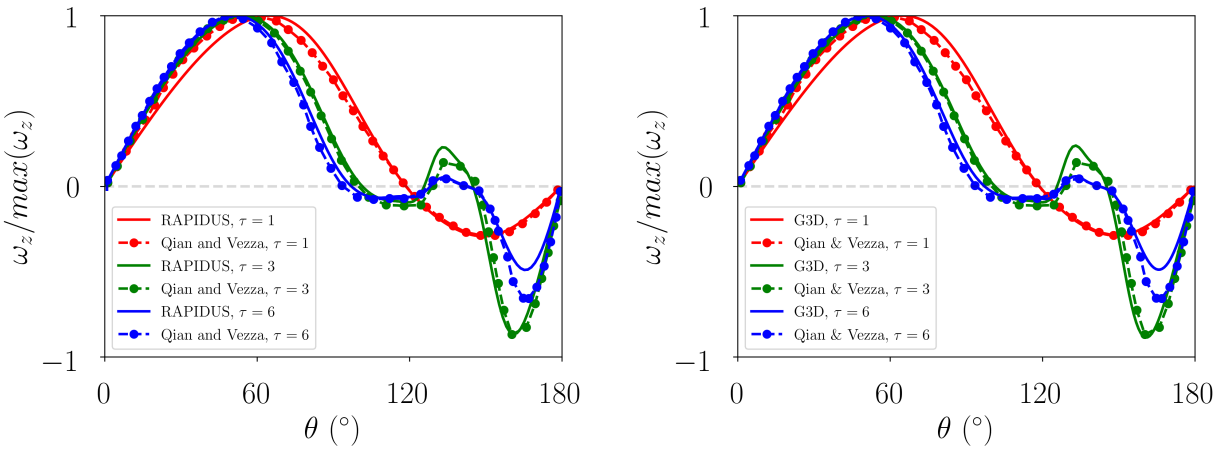


Fig. 8 Contour lines of spanwise vorticity for impulsively started flow past a circular cylinder at Re = 1000. G3D and GARFIELD (with MUSCL scheme) are compared to OVERFLOW and RAPIDUS.

D. Steady, Turbulence Modeling Validation, 2-D NACA 0012

In this section we present turbulence modeling validation for G3D using the LaRC Turbulence Modeling Resource (TMR) NACA0012 Airfoil validation case. This case has mach number $M_\infty = 0.15$, Reynolds number $Re = 6 \times 10^6$ based on a chord of 1 and three angle of attacks $\alpha = 0°, 10°$, and $15°$. The C-grid used for the study, which was taken from the TMR website, is shown in Fig. 10. The second-finest grid from the TMR website is used consisting of 897 nodes in the circumferential direction and 257 nodes in the normal direction. Figure 11 shows a comparison of pressure coefficient C_P and skin friction C_f (top surface) for G3D, GARFIELD, OVERFLOW and the experimental data from

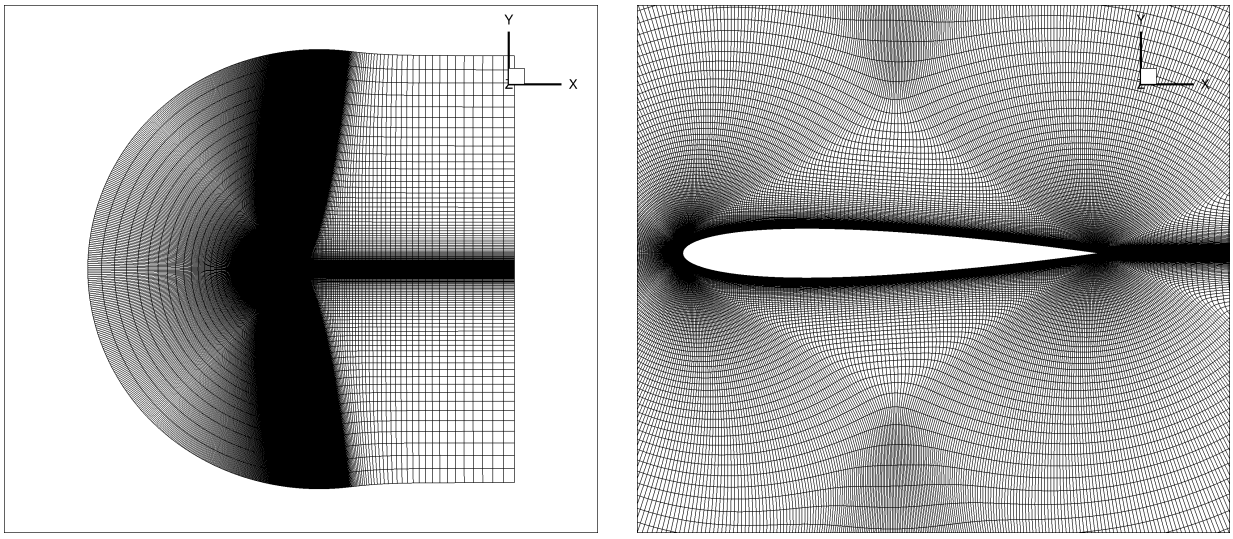


(a) RAPIDUS vs Qian and Vezza

(b) G3D vs Qian and Vezza

Fig. 9 Surface vorticity distribution at various time instances for the impulsively started flow around cylinder at Re = 1000. Figure (a) compares RAPIDUS versus Qian and Vezza and figure (b) compares G3D versus Qian and Vezza. Lines correspond to results from RAPIDUS and G3D and lines with markers are from Qian and Vezza [25].

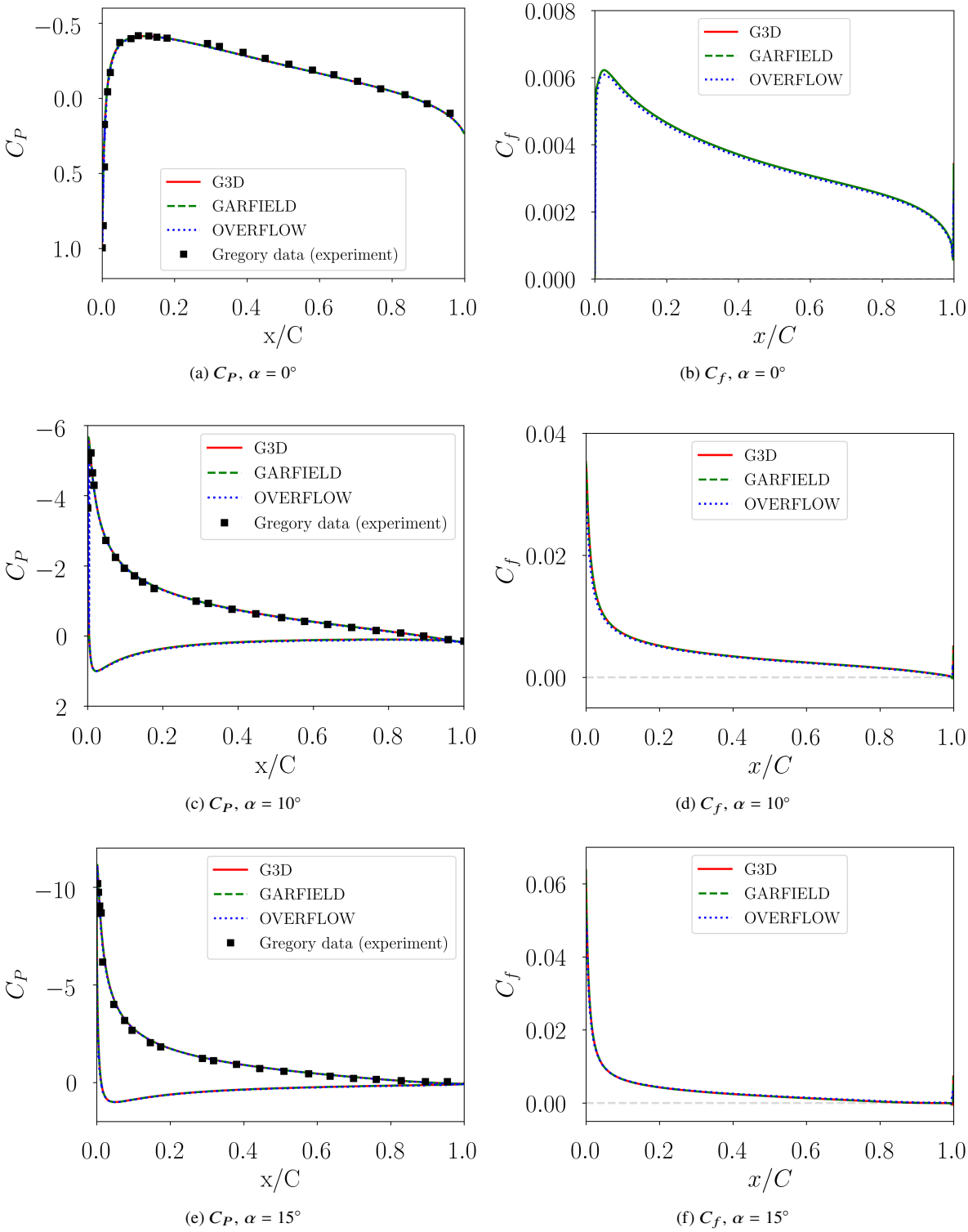
Gregory [26]. G3D and GARFIELD is used with the MUSCL scheme. OVERFLOW predictions are from Jespersen et al. [27] and use a third-order Roe differencing scheme for the right-hand side and the scalar pentadiagonal solver on the left-hand side. G3D and GARFIELD use dirchlet boundary conditions in the farfield while OVERFLOW uses farfield Riemann boundary conditions. An identical grid is used for all three flow solvers. Comparing the results, good agreement is observed among all the flow solvers and the measured data. Table 2 compares the lift C_L and drag C_D coefficients for the three flow solvers. The force predictions for all flow solvers are in close agreement.



(a) Far-view of C-Grid

(b) Close-up view of C-grid

Fig. 10 Farview (a) and close-up view of C-grid mesh.



(a) C_P, $\alpha = 0°$

(b) C_f, $\alpha = 0°$

(c) C_P, $\alpha = 10°$

(d) C_f, $\alpha = 10°$

(e) C_P, $\alpha = 15°$

(f) C_f, $\alpha = 15°$

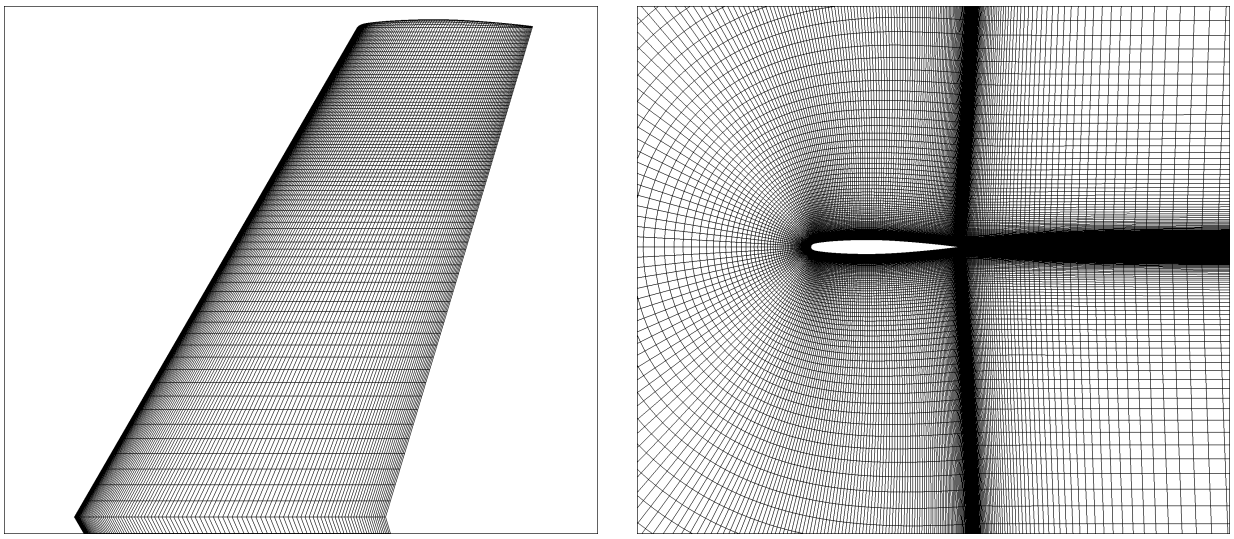
Fig. 11 Comparison of the pressure coefficient C_P and skin friction coefficient C_f (top surface) at $Re = 6 \times 10^6$ and $M = 0.15$ is performed for G3D, GARFIELD, OVERFLOW, and experimental data from Gregory [26]. A NACA 0012 airfoil with an SA turbulence model is used. The angles of attack (α) are set to $0°$ (a+b), $10°$ (c+d), and $15°$ (e+f). G3D and GARFIELD use the MUSCL scheme.

Table 2 Comparison of lift C_L and drag C_D coefficient at $Re = 6 \times 10^6$ and $M = 0.15$ is performed for G3D, GARFIELD, and OVERFLOW. A NACA 0012 airfoil with an SA turbulence model is used.

	C_L			C_D		
	$\alpha = 0^\circ$	$\alpha = 10^\circ$	$\alpha = 15^\circ$	$\alpha = 0^\circ$	$\alpha = 10^\circ$	$\alpha = 15^\circ$
G3D	~ 0	1.0907	1.5436	0.00844	0.01253	0.02159
GARFIELD	~ 0	1.0898	1.5435	0.00841	0.01259	0.02160
OVERFLOW	~ 0	1.0990	1.5576	0.00838	0.01251	0.02149

E. Steady, 3-D Turbulence Modeling Validation, ONERA M6

The Onera-M6 wing experiments of Schmitt et. al. from 1979 [28] provide a popular validation data set for CFD. The swept aircraft wing was mounted in a wind tunnel and run at transonic conditions with a Mach number $M_\infty = 0.8395$, and angle of attack of $\alpha = 3.06^\circ$. The Reynolds number for this case is $Re = 14.6 \times 10^6$, which is based on the free-stream velocity and root chord. The structured surface mesh and side view (at the center of the wing) is shown in Fig. 12. For the purpose of this study, the wing is mirrored in the span-wise direction to double the problem size and avoid modeling the wind tunnel wall. The problem size is intentionally doubled to create a large domain suitable for a scalability study. The wing geometry, provided in the experimental report, is meshed using a C-grid topology. The j-coordinate is the airfoil wrap-around direction resulting in a wake cut periodic boundary before and after the trailing edge. The k-coordinate is in the airfoil normal direction, resulting in a no-slip boundary at one end and a dirchlet boundary at the end of the k-direction. The l-boundaries are "wake" boundaries where the mesh collapses to a plane. The non-dimensional viscous spacing at the wall corresponds to $y^+ = 1$ at $0.1c_{tip}$ and the volume mesh extends 50 root chords away from the wing. The structured mesh consists of a total of 18.2 million nodes, with 501 nodes in the wrap-around direction (275 around the wing), 223 nodes in the wall normal direction and 163 nodes along the span.



(a) ONERA M6 Surface Mesh

(b) Side View of ONERA M6 Mesh (at center of wing)

Fig. 12 View of ONERA M6 surface mesh and side-view (at center of wing).

Figure 13 shows contours of the surface pressure together with the contour plot of the divergence of velocity at 80% span. Shock waves are characterized by a rapid compression, which implies a sudden change in velocity. A high negative divergence region indicates where the fluid is experiencing a sudden compression. Thus, the shock wave appears as a sharp boundary or line in the contour plot where there is a steep gradient in divergence. The divergence of velocity contour plot clearly shows the double shock (two vertical black regions) that occurs at 80% span.

A quantitative comparison is provided in Fig. 14, where the section pressure coefficients, C_P, predicted by G3D,

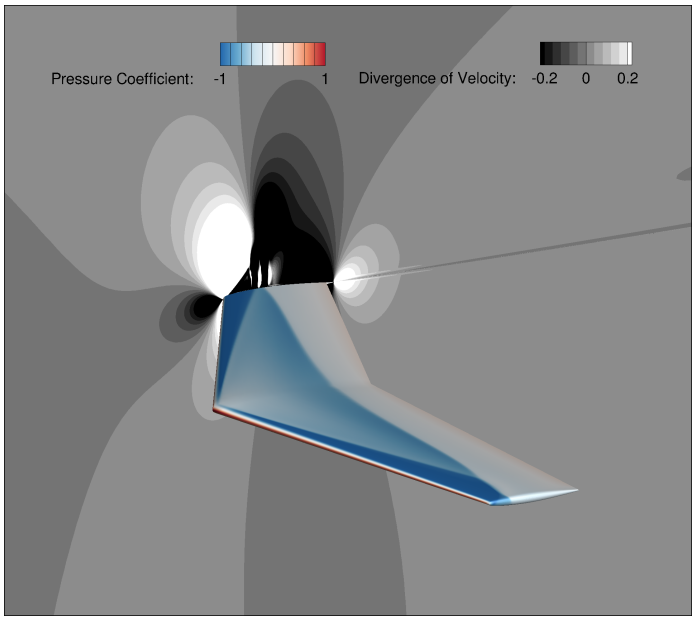


Fig. 13 Contours of surface pressure and divergence of velocity at 80% span. A steep gradient in divergence illustrates where the shock occurs.

GARFIELD, OVERFLOW and RAPIDUS are plotted alongside the experimental data. Six span-wise locations are selected ranging from 44% span to 99% span. The span percentage is defined such that 0% is at the center of the CFD wing mesh and 100% is at the wing tip. With G3D and GARFIELD, the 3rd order MUSCL scheme is used. With OVERFLOW, due to instability with higher-order schemes, a 2nd order central scheme was used. The results from the various flow solvers are nearly indistinguishable from each other, where the main difference is near the shock. With the 2nd-order scheme used in OVERFLOW, the shock is more smoothed than the other solvers. Comparing to the experimental data, all flow solvers agree well with the measured data except near the wing tip at 99% span. This is likely due to the underresolved mesh density at the tip.

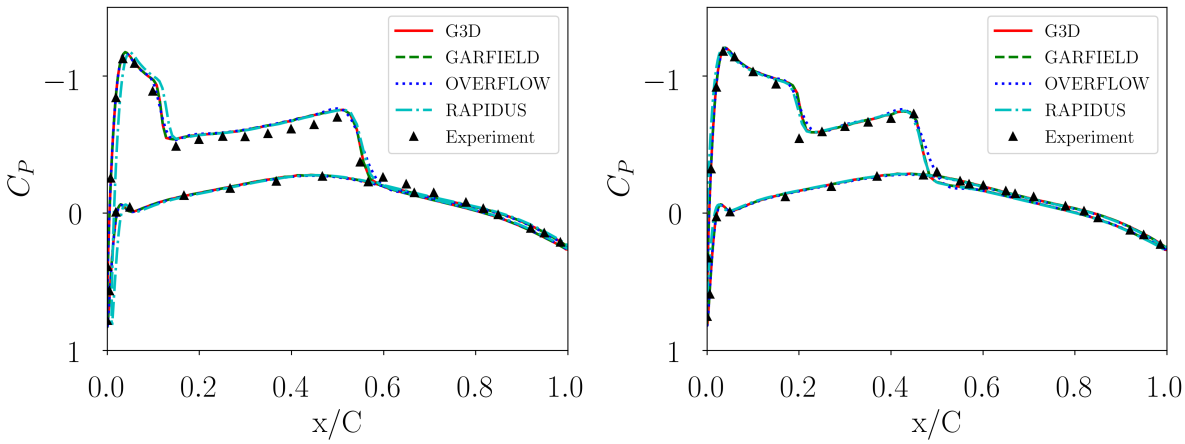
F. Convergence and Code Performance Comparisons of the ONERA M6

In this section, we provide a brief assessment of the preliminary performance of G3D. All computations discussed herein were conducted on the local compute cluster at Ames Research Center. The cluster is equipped with 4 NVIDIA A100-SXM4-80GB GPUs each with an AMD EPYC 7513 64 physical core processor. The execution time was evaluated for the ONERA M6 case. All wall times, measured in seconds, are averages derived from 5 runs, each consisting of 100 iterations. The mesh communication manager is currently under implementation in G3D, thus, the results presented here were obtained using a single GPU.

Comparison of convergence for G3D, GARFIELD, OVERFLOW, and RAPIDUS is shown in Fig. 15. The L_2 norm of residual is normalized by the peak residual according to the equation below,

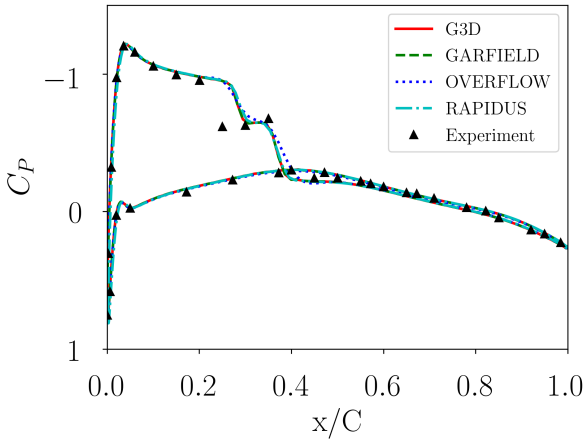
$$L_2 \text{ Norm} = \log\left(\frac{\text{Residual}}{\text{Peak Residual}}\right) \tag{9}$$

so that a value of 0 corresponds to the peak residual. A horizontal dashed line is added to the plot to indicate a sufficient level of convergence (4 orders of magnitude (OOM) residual drop). Comparing the residual of the flow solvers, RAPIDUS converges the fastest, followed by OVERFLOW and then G3D/GARFIELD. Figure 16 tabulates the time per iteration, the number of iterations to 4 OOM residual drop as well as the number of iterations to a converged drag result. The total wall time it takes to reach this number of iterations is also included. A converged drag result was defined using a moving 1000-sample window, where the drag was deemed converged once the data within this window had a standard deviation of less than 1E-4. Comparing the solvers, RAPIDUS takes the least amount of iterations to converge the residual and drag. Although it takes the most amount of time per iteration, it is still the fastest to reach 4 OOM residual drop. However, RAPIDUS has the largest wall time among all the solvers to reach a converged drag result. G3D and GARFIELD perform very similar, with GARFIELD about 9% faster, as expected for a code written entirely in raw

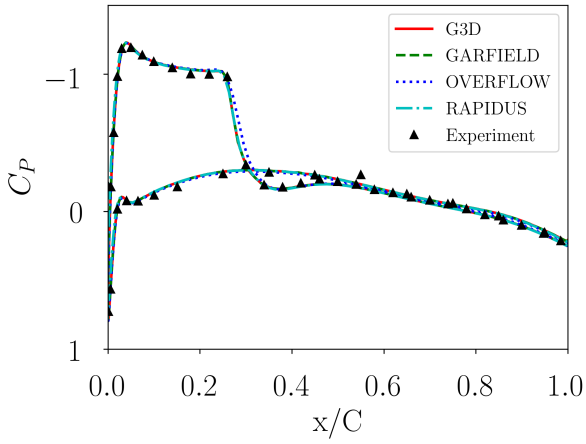


(a) 44% Span

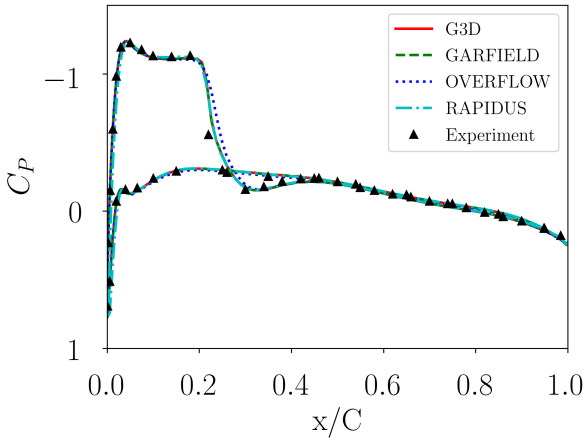
(b) 65% Span



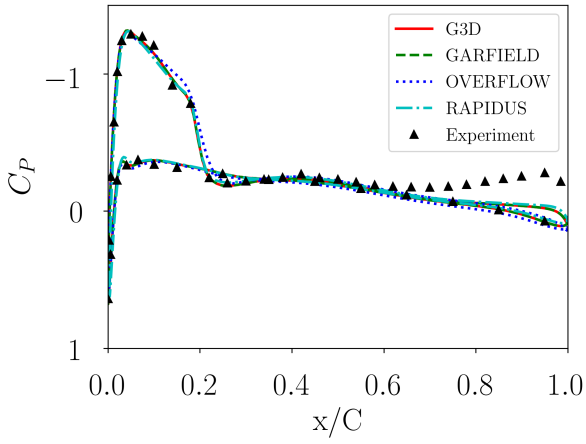
(c) 80% Span



(d) 90% Span



(e) 95% Span



(f) 99% Span

Fig. 14 Pressure coefficient solution at six wing sections comparing G3D, GARFIELD, OVERFLOW, RAPIDUS, and experimental results.

CUDA. The speed up is defined as,

$$\text{Speedup} = \frac{T_{\text{original}}}{T_{\text{optimized}}} \tag{10}$$

Comparing the time it takes to converge the drag in G3D and OVERFLOW, for this particular test case, G3D has a speed up of 1.6x versus the current GPU implementation of OVERFLOW. This speed up could be due partly to the different numerical schemes used for the left- and right-hand side; G3D uses DADI and MUSCL while OVERFLOW uses the pentadiagonal and central scheme. Comparing to OVERFLOW after the upwind and DADI schemes are implemented could help better understand this difference in performance.

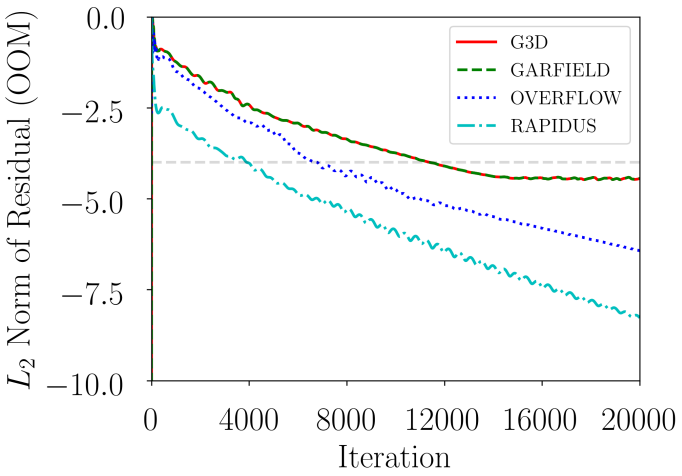


Fig. 15 Comparison of convergence for G3D, GARFIELD, OVERFLOW, and RAPIDUS. L_2 norm of residual is normalized by the peak residual.

	Average Time Per Iteration	Iterations to 4 OOM drop	Average Time to 4 OOM drop	Iterations to Drag Window σ < 1E-4	Avg. Time to Drag Window σ < 1E-4
G3D	0.136	11392	1547.3	6359	863.7
GARFIELD	0.123	11394	1405.9	6356	784.3
OVERFLOW	0.269	6790	1826.4	5200	1398.7
RAPIDUS	0.305	3902	1189.1	4882	1487.7

Fig. 16 Comparison of code performance for G3D, GARFIELD, OVERFLOW, and RAPIDUS. All codes are executed on a single NVIDIA A100 GPU. An averaging window of 1000 iterations was used with a standard deviation σ threshold of less than 1E-4. All wall times, measured in seconds, are averages derived from 5 runs, each consisting of 100 iterations.

For CPU execution, OpenMP directives were employed to enhance performance. Figure 17 illustrates the wall-clock execution time of G3D as a function of the number of OpenMP threads, alongside the GPU execution time. Above 16 OpenMP threads, the wall time flat-lines. Comparing 64 OpenMP threads versus 1 GPU, the GPU implementation is about 37x faster. The large speedup seen on GPUs is a result of poor scaling of OpenMP, which shows saturation of execution time above 16 threads for large grids. True performance portability requires domain decomposition and usage of MPI+X type execution paradigm. Note that OVERFLOW, GARFIELD and RAPIDUS are all multi-GPU and multi-CPU compatible and implement the MPI+X approach . Future work will be geared towards making G3D MPI compatible and performing detailed scaling studies on massively parallel systems.

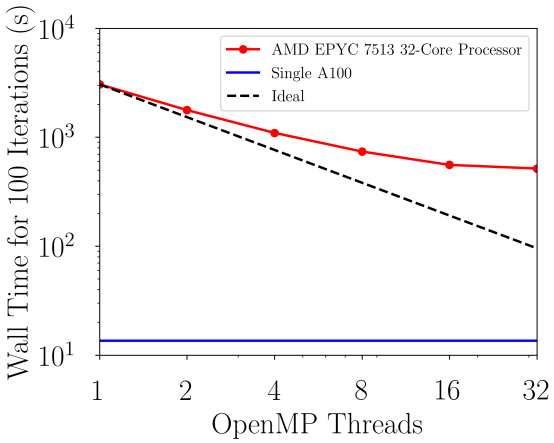


Fig. 17 G3D scaling study on OpenMP.

V. Conclusions

A performance portable implementation of an implicit compressible Reynolds-Averaged Navier-Stokes solver that uses structured curvilinear grids is developed in this work. Structured curvilinear grid based solver have always historically shown superior accuracy and efficiency compared to the unstructured grid counterparts on CPU based architectures. Vendor specific implementations of structured grid solvers have also shown similar superior performance compared to unstructured grid solvers on GPUs. In this context, the main theme of this work was to evaluate whether a performance portable (single code base executable on multiple platforms) implementation of an implicit structured grid based RANS solver can retain the same performance and accuracy advantages. Accuracy evaluations were conducted on 3 different geometries, which are steady/unsteady flows over cylinder, turbulent flow over NACA0012 airfoil and Transonic flow with shocks over ONERAM6 wing. All results showed good agreement with available test data and equivalent simulations from other codes.

Performance on GPU systems showed about 10% deficit when compared with vendor specific (CUDA) implementation that was used by the GARFIELD code. OVERFLOW uses an OpenACC based implementation on GPUs with a stronger linear solver at each iteration. Therefore, OVERFLOW is found to be slower on a per-iteration basis. However, when the time for achieving the same amount of convergence in residuals are compared, the time to solution required by OVERFLOW is within 20% of the performance portable and native CUDA implementations. As expected, the structured grid based computations are a factor of 3X faster on per time step basis when compared with unstructured simulations on the same grid. When convergence metrics are considered, the unstructured solver (RAPIDUS) appears faster since it required fewer number of iterations to achieve the same amount of residual drop. When time to solution is considered (measured using force convergence as a metric), the structured grid solvers are found to be 2X faster with only a 10% difference between native CUDA and the performance portable implementations.

An advantage of the performance portable software development is the ability to use the same exact executable on both CPUs and GPUs. The G3D solver is executed on a CPU system with OpenMP threading to verify its capability to run on both CPUs and GPUs. Scalability with OpenMP is found to be poor with the time-per-iteration saturating at 16 threads. A single A100 GPU is seen to outperform the best solution that can be used on a 64 core CPU node using OpenMP by a factor of 37x. The lack of OpenMP performance will be explored in the future. The longer term vision includes a fully partitioned MPI+X type of implementation of G3D, where X could be GPUs or OpenMP threading.

Overall, results from performance portable implementation of a structured grid flow solver showed promising trends. Furthermore, the implementations developed in this work for line-based solution reconstruction techniques and linear solvers provide a foundation for extending this technology to semi-structured strand meshes in the future.

VI. Acknowledgements

Material presented in this abstract is a product of the U.S. Army and the CREATE (Computational Research and Engineering for Acquisition Tools and Environments) element of the U.S. Department of Defense HPC Modernization Program. The authors also thank Dr. Charles Jackson from NASA for providing a development version of the OVERFLOW code.

References

[1] Hosseinverdi, S., Sitaraman, J., and Jude, D., "A Performance Portable Flow Solver for Rotorcraft Applications," *Transformative Vertical Flight 2024, 6th Decennial VFS Aeromachanics Specialist Conference*, 2024.

[2] Beckingsale, D. A., Burmark, J., Hornung, R., Jones, H., Killian, W., Kunen, A. J., Pearce, O., Robinson, P., Ryujin, B. S., and Scogland, T. R., "RAJA: Portable Performance for Large-Scale Scientific Applications," *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2019, pp. 71–81. https://doi.org/10.1109/P3HPC49587.2019.00012.

[3] Trott, C., Berger-Vergiat, L., Poliakoff, D., Rajamanickam, S., Lebrun-Grandie, D., Madsen, J., Al Awar, N., Gligoric, M., Shipman, G., and Womeldorff, G., "The Kokkos EcoSystem: Comprehensive Performance Portability for High Performance Computing," *Computing in Science and Engineering*, Vol. 23, No. 5, 2021, pp. 10–18. https://doi.org/10.1109/MCSE.2021.3098509.

[4] Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D., Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., and Wilke, J., "Kokkos 3: Programming Model Extensions for the Exascale Era," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 4, 2022, pp. 805–817. https://doi.org/10.1109/TPDS.2021.3097283.

[5] Medina, D. S., St-Cyr, A., and Warburton, T., "OCCA: A unified approach to multi-threading languages," *arXiv preprint arXiv:1403.0968*, 2014.

[6] Derlaga, J. M., Jackson, C. W., and Buning, P. G., "Recent progress in OVERFLOW convergence improvements," *AIAA Scitech 2020 Forum*, 2020, p. 1045.

[7] Bartels, R. E., Rumsey, C. L., and Biedron, R. T., "Cfl3d version 6.4-general usage and aeroelastic analysis," Tech. rep., 2006.

[8] Wissink, A. M., Staruk, W. J., Tran, S. A., Roget, B., Lakshminarayan, V. K., Sitaraman, J., and Jayaraman, B., "Overview of new capabilities in helios version 9.0," *AIAA Scitech 2019 Forum*, 2019, p. 0839.

[9] Roget, B., Sitaraman, J., Lakshminarayan, V., and Wissink, A., "Prismatic mesh generation using minimum distance fields," *Computers & Fluids*, Vol. 200, 2020, p. 104429.

[10] McDaniel, D. R., "A Summary of New and Emerging Features in HPCMP CREATETM-AV Kestrel," *AIAA Scitech 2021 Forum*, 2021, p. 0234.

[11] Anderson, W. K., Biedron, R. T., Carlson, J.-R., Derlaga, J. M., Diskin, B., Druyor Jr, C. T., Gnoffo, P. A., Hammond, D. P., Jacobson, K. E., Jones, W. T., et al., "FUN3D Manual: 14.0. 2," Tech. rep., 2023.

[12] Jude, D., "Advancing the multi-solver paradigm for overset CFD toward heterogeneous architectures," Ph.D. thesis, University of Maryland, 2019.

[13] Jude, D., Sitaraman, J., and Wissink, A. M., "An Octree-based, Cartesian CFD Solver for Helios on CPU and GPU Architectures." *AIAA Scitech 2021 Forum*, 2021, p. 0841.

[14] Hosseinverdi, S., Sitaraman, J., Zagaris, G., Lakshminarayan, V. K., and Holst, K. R., "Development of a Performance Portable Massively Parallel Unstructured Compressible Flow Solver," *AIAA AVIATION 2023 Forum*, 2023, p. 3427.

[15] Sitaraman, J., Hosseinverdi, S., Jude, D., Roget, B., Abras, J., Lakshminarayan, V., and Zagaris, G., "Progress in Massively Parallel and Performance Portable Overset Implementations in HPCMP CREATE A/V Helios," *AIAA SCITECH 2024 Forum*, 2024, p. 1940.

[16] Zagaris, G., Sitaraman, J., Holst, K., Tyson, W., Starr, R., McNally, R. P., Lamberson, S., and Bond, R., "Using Mint to Explore Performance Portability Strategies in HPCMP CREATE^{TM} Kestrel," https://ugm.hpc.mil/abstracts/2022/Zagaris.html, 2022.

[17] Jackson, C. W., Appelhans, D., Derlaga, J. M., and Buning, P. G., "GPU Implementation of the OVERFLOW CFD Code," *AIAA SCITECH 2024 Forum*, 2024, p. 0042.

[18] Koren, B., "A robust upwind discretization method for advection, diffusion and source terms," *Notes on Numerical Fluid Mechanics*, 1993, pp. 117–138.

[19] Jiang, G. S., and Shu, C. W., "Efficient Implementation of Weighted ENO Schemes," *Journal of Computational Physics*, Vol. 126, No. 1, 1996, pp. 202–228.

[20] Roe, P., "Approximate riemann solvers, paramter vectors, and difference schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372.

[21] Pulliam, T., and Chaussee, D., "A diagonal form of an implicit approximate-factorization algorithm," *Journal of Computational Physics*, Vol. 39, No. 2, 1981, pp. 347–363. https://doi.org/10.1016/0021-9991(81)90156-X.

[22] Spalart, P., and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," *30th aerospace sciences meeting and exhibit*, 1992, p. 439.

[23] Beckingsale, D. A., McFadden, M. J., Dahm, J. P. S., Pankajakshan, R., and Hornung, R. D., "Umpire: Application-focused management and coordination of complex hierarchical memory," *IBM Journal of Research and Development*, Vol. 64, No. 3/4, 2020, pp. 00:1–00:10. https://doi.org/10.1147/JRD.2019.2954403.

[24] Jameson, A., Schmidt, W., and Turkel, E., "Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes," *14th Fluid and Plasma Dynamics Conference*, American Institute of Aeronautics and Astronautics, 1981. https://doi.org/10.2514/6.1981-1259.

[25] Qian, L., and Vezza, M., "A vorticity-based method for incompressible unsteady viscous flows," *Journal of computational physics*, Vol. 172, No. 2, 2001, pp. 515–542.

[26] Gregory, N., and O'reilly, C., "Low-speed aerodynamic characteristics of NACA 0012 aerofoil section, including the effects of upper-surface roughness simulating hoar frost," 1970.

[27] Jespersen, D. C., Pulliam, T. H., and Childs, M. L., "Overflow turbulence modeling resource validation results," Tech. Rep. NAS-2016-01, NAS Technical Report, 2016. URL https://turbmodels.larc.nasa.gov/Papers/NAS_Technical_Report_NAS-2016-01.pdf.

[28] Schmitt, V., and Charpin, F., "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers AGARD-AR-138 B1-1," , 1979.