Seventh International Conference on
Computational Fluid Dynamics (ICCFD7),
Big Island, Hawaii, July 9-13, 2012

ICCFD7-4205

An Implicit Hermite WENO Reconstruction-Based Discontinuous Galerkin Method on Tetrahedral Grids

Yidong Xia[*] and Robert Nourgaliev[**]
Corresponding author: yxia2@ncsu.edu

[*] North Carolina State University, Raleigh, NC 27695, USA
[**] Idaho National Laboratory, Idaho Falls, ID 83415 USA.

Abstract: An Implicit Reconstructed Discontinuous Galerkin method, IRDG(P_1P_2), is presented for solving the compressible Euler equations on tetrahedral grids. In this method, a quadratic polynomial (P_2) solution is first reconstructed using a least-squares method from the underlying linear polynomial (P_1) DG solution. By taking advantage of the derivatives in the DG formulation, the stencils used in the reconstruction involve only von Neumann neighborhood (adjacent face-neighboring cells) and thus are compact and consistent with the underlying DG method. The final P_2 solution is then obtained using a WENO reconstruction, which is necessary to ensure stability of the RDG(P_1P_2) method. A matrix-free GMRES (generalized minimum residual) algorithm is presented to solve the approximate system of linear equations arising from Newton linearization. The LU-SGS (lower-upper symmetric Gauss-Seidel) preconditioner is applied with both the simplified and approximate Jacobian matrices. The numerical experiments on a variety of flow problems demonstrate that the developed IRDG(P_1P_2) method is able to obtain a speedup of at least two orders of magnitude than its explicit counterpart, maintain the linear stability, and achieve the designed third order of accuracy: one order of accuracy higher than the underlying second-order DG(P_1) method without significant increase in computing costs and storage requirements. It is also found that a well approximated Jacobian matrix is essential for the IRDG method to achieve fast converging speed and maintain robustness on large-scale problems.

Keywords: Discontinuous Galerkin, Reconstruction Method, WENO, Compressible Flows, Implicit Method, Tetrahedral Grids.

1 Introduction

The discontinuous Galerkin methods [1-25] combine two advantageous features commonly associated to finite element and finite volume methods. As in classical finite element methods, accuracy is obtained by means of high-order polynomial approximation within an element rather than by wide stencils as in the case of finite volume methods. The physics of wave propagation is, however, accounted for by solving the Riemann problems that arise from the discontinuous representation of the solution at element interfaces. In this respect, the methods are therefore similar to finite volume methods. In contrast to the enormous advances in the theoretical and numerical analysis of the DG methods, the development of a viable, attractive, competitive, and ultimately superior DG method over the more mature and well-established second order methods is relatively an untouched area. This is mainly due to the fact that the DG methods have a number of weaknesses that have yet to be addressed, before they can be robustly used to flow problems of practical interest in a complex configuration environment. In particular, there are three most challenging and unresolved issues in the DG methods: a) how to efficiently discretize diffusion terms required for the Navier-Stokes equations, b) how to effectively control spurious oscillations in the presence of strong discontinuities, and c) how to develop efficient time integration schemes for time accurate and steady-state solutions. Indeed, compared to the finite element methods and finite volume methods, the DG methods require solutions of systems of equations with more unknowns for the same grids. Consequently, these methods have been recognized as expensive in terms of both computational costs and storage requirements.

Dumbser et al. [18-20] have originally introduced a new family of reconstructed DG (RDG) methods, termed P_nP_m schemes, where P_n indicates that a piecewise polynomial of degree of n is used to represent a DG solution, and P_m represents a reconstructed polynomial solution of degree of m ($m \geq n$) that is used to compute the fluxes. The beauty of P_nP_m schemes is that they provide a unified formulation for both finite volume and DG methods, and contain both classical finite volume and standard DG methods as two special cases of PnPm schemes, and thus allow for a direct efficiency comparison.

Obviously, the construction of an accurate and efficient reconstruction operator is crucial to the success of the PnPm schemes. In Dumbser's work, this is achieved using a so-called in-cell recovery. The resultant over-determined system is then solved using a least-squares method that guarantees exact conservation, not only of the cell averages but also of all higher order moments in the reconstructed cell itself, such as slopes and curvatures. However, this conservative least-squares recovery approach is computationally expensive, as it involves both recovery of a polynomial solution of higher order and least-squares solution of the resulting over-determined system. Furthermore, the recovery might be problematic for a boundary cell, where the number of the face-neighboring cells might be not enough to provide the necessary information to recover a polynomial solution of a desired order.

Fortunately, recovery is not the only way to obtain a polynomial solution of higher order from the underlying discontinuous Galerkin solutions. Rather, reconstruction widely used in the FV methods provides an alternative, probably a better choice to obtain a higher-order polynomial representation. Luo et al. develop a reconstructed discontinuous Galerkin method using a Taylor basis [26-28] for the solution of the compressible Euler and Navier-Stokes equations on arbitrary grids, where a higher order polynomial solution is reconstructed by use of a strong interpolation, requiring point values and derivatives to be interpolated on the face-neighboring cells. The resulting over-determined linear system of equations is then solved in the least-squares sense. This reconstruction scheme only involves the von Neumann neighborhood, and thus is compact, simple, robust, and flexible. The reconstruction scheme guarantees exact conservation, not only of the cell averages but also of their slopes due to a judicious choice of our Taylor basis.

However, the attempt to naturally extend the RDG method to solve 3D Euler equations on tetrahedral grids is not successful. Like the second order cell-centered finite volume methods, i.e., RDG(P_0P_1), the resultant RDG(P_1P_2) methods are unstable. Although RDG(P_0P_1) methods are in general stable in 2D and on Cartesian or structured grids in 3D, they suffer from the so-called linear instability on unstructured tetrahedral grids, when the reconstruction stencils only involve von Neumann neighborhood, i.e., adjacent face-neighboring cells [29]. Unfortunately, the RDG(P_1P_2) method exhibits the same linear instability, which can be overcome by using extended stencils. However, this is achieved at the expense of sacrificing the compactness of the underlying DG methods. Furthermore, these linear reconstruction-based DG methods will suffer from non-physical oscillations in the vicinity of strong discontinuities for the compressible Euler equations. Alternatively, ENO, WENO, and HWENO can be used to reconstruct a higher-order polynomial solution, which can not only enhance the order of accuracy of the underlying DG method but also achieve both linear and non-linear stability. This type of hybrid HWENO+DG schemes has been developed on 1D and 2D structured grids by Balsara et al. [30], where the HWENO reconstruction is relatively simple and straightforward.

On the other hand, early efforts in the development of temporal discretization methods for RDG methods in 3D focused on explicit schemes [26-28]. Usually, explicit temporal discretizations such as multistage Runge–Kutta schemes are used to drive the solution to steady state. Acceleration techniques such as local time-stepping and implicit residual smoothing have also been combined in this context. In general, explicit schemes and their boundary conditions are easy to implement, vectorize and parallelize, and require only limited memory storage. However, for large-scale problems and especially for the solution of the Navier–Stokes equations, the rate of convergence slows down dramatically, resulting in inefficient solution techniques. In order to speed up convergence, an implicit temporal discretization for RDG methods is required.

In general, implicit methods require the solution of a linear system of equations arising from the linearization of a fully implicit scheme at each time step or iteration. The most widely used methods to solve a linear system on tetrahedron grids are iterative solution methods and approximate factorization methods. Significant efforts have been made to develop efficient iterative solution methods. These range from Gauss–Seidel to Krylov subspace methods that use a wide variety of preconditioners (see, e.g., Stoufflet [31], Batina [32], Venkatakrishnan et al. [33], Knight [34], Whitaker [35], Luo et al. [36], and Barth et al. [37]). The most successful and effective iterative method is to use the Krylov subspace methods [38] such as GMRES and BICGSTAB with an ILU (incomplete lower–upper) factorization preconditioner.

The objective of the effort discussed is to develop an Implicit Rconstructed Discontinuous Galerkin method, IRDG(P_1P_2), based on a Hermite WENO reconstruction using a Taylor basis [13] for the solution of the compressible Euler equations on unstructured tetrahedral grids. This HWENO-based IRDG method is designed not only to reduce the high computing costs of the DGM and improve the converging speed, but also to avoid spurious oscillations in the vicinity of strong discontinuities, thus effectively overcoming the two shortcomings of the DG methods and ensuring the stability of the RDG method. A matrix-free GMRES algorithm with an LU-SGS preconditioner is used to solve a system of linear equations, arising from an approximate linearization of an implicit temporal discretization at each time step. The developed IRDG method is used to compute a variety of flow problems on tetrahedral grids to demonstrate its accuracy, efficiency, and robustness. The remainder of this paper is organized as follows. The governing equations are listed in Section 2. The underlying RDG method is presented in Section 3. The implicit algorithm is presented in Section 4. Extensive numerical experiments are reported in Section 5. Concluding remarks are given in Section 6.

2 Governing Equations

The Euler equations governing unsteady compressible inviscid flows can be expressed as

$$\frac{\partial \mathbf{U}(x,t)}{\partial t}+\frac{\partial \mathbf{F}_k(\mathbf{U}(x,t))}{\partial x_k}=0 \tag{2.1}$$

where the summation convention has been used. The conservative variable vector **U**, and inviscid flux vector **F** are defined by

$$\mathbf{U}=\begin{pmatrix}\rho\\ \rho u_i\\ \rho e\end{pmatrix} \qquad \mathbf{F}_j=\begin{pmatrix}\rho u_j\\ \rho u_i u_j+p\delta_{ij}\\ u_j(\rho e+p)\end{pmatrix} \tag{2.2}$$

Here ρ, p, and e denote the density, pressure, and specific total energy of the fluid, respectively, and u_i is the velocity of the flow in the coordinate direction x_i. The pressure can be computed from the equation of state

$$p=(\gamma-1)\rho\left(e-\frac{1}{2}u_j u_i\right) \tag{2.3}$$

which is valid for perfect gas, where γ is the ratio of the specific heats.

3 Reconstructed Discontinuous Galerkin Method

The governing equation (2.1) is discretized using a discontinuous Galerkin finite element formulation. To formulate the discontinuous Galerkin method, we first introduce the following weak formulation, which is obtained by multiplying the above conservation law by a test function **W**, integrating over the domain Ω, and then performing an integration by parts,

$$\int_{\Omega} \frac{\partial \mathbf{U}}{\partial t} \mathbf{W} d\Omega + \int_{\Gamma} \mathbf{F}_k \mathbf{n}_k d\Gamma - \int_{\Omega} \mathbf{F}_k \frac{\partial \mathbf{W}}{\partial x_k} d\Omega = 0, \qquad \forall \mathbf{W} \in V \tag{3.1}$$

where Γ(=∂Ω) denotes the boundary of Ω, and $\mathbf{n}_j$ the unit outward normal vector to the boundary. We assume that the domain Ω is subdivided into a collection of non-overlapping tetrahedral elements Ω_e in 3D. We introduce the following broken Sobolev space V_h^p

$$V_h^p = \{ v_h \in [L_2(\Omega)]^m : v_h|_{\Omega_e} \in [V_p^m] \; \forall \Omega_e \in \Omega \} \tag{3.2}$$

which consists of discontinuous vector-values polynomial functions of degree *p*, and where *m* is the dimension of the unknown vector and

$$V_p^m = \text{span}\left\{ \prod x_i^{\alpha_i} : 0 \le \alpha_i \le p, 0 \le i \le d \right\} \tag{3.3}$$

where α denotes a multi-index and *d* is the dimension of space. Then, we can obtain the following semi-discrete form by applying weak formulation on each element Ω_e, find $\mathbf{U}_h \in V_h^p$ such as

$$\frac{d}{dt} \int_{\Omega_e} \mathbf{U}_h \mathbf{W}_h d\Omega + \int_{\Gamma_e} \mathbf{F}_k(\mathbf{U}_h) \mathbf{n}_k \mathbf{W}_h d\Gamma - \int_{\Omega_e} \mathbf{F}_k(\mathbf{U}_h) \frac{\partial \mathbf{W}_h}{\partial x_k} d\Omega = 0, \quad \forall \mathbf{W}_h \in V_h^p \tag{3.4}$$

where $\mathbf{U}_h$ and $\mathbf{W}_h$ represent the finite element approximations to the analytical solution **U** and the test function **W** respectively, and they are approximated by a piecewise polynomial function of degrees p, which are discontinuous between the cell interfaces. Assume that *B* is the basis of polynomial function of degrees *p*, this is then equivalent to the following system of *N* equations,

$$\frac{d}{dt} \int_{\Omega_e} \mathbf{U}_h B_i d\Omega + \int_{\Gamma_e} \mathbf{F}_k(\mathbf{U}_h) \mathbf{n}_k B_i d\Gamma - \int_{\Omega_e} \mathbf{F}_k(\mathbf{U}_h) \frac{\partial B_i}{\partial x_k} d\Omega = 0, \quad 1 \ll i \ll N \tag{3.5}$$

where *N* is the dimension of the polynomial space. Since the numerical solution $\mathbf{U}_h$ is discontinuous between element interfaces, the interface fluxes are not uniquely defined. The flux function $\mathbf{F}_k(\mathbf{U}_h)\mathbf{n}_k$ appearing in the second terms of Eq. (3.5) is replaced by a numerical Riemann flux function $\mathbf{H}_k(\mathbf{U}_h^L, \mathbf{U}_h^R, \mathbf{n}_k)$ where $\mathbf{U}_h^L$ and $\mathbf{U}_h^R$ are the conservative state vector at the left and right side of the element boundary. This scheme is called discontinuous Galerkin method of degree P, or in short notation DG(P) method. Note that the DG formulations are very similar to finite volume schemes, especially in their use of numerical fluxes. Indeed, the classical first-order cell-centered finite volume scheme exactly corresponds to the DG(P_0) method, i.e., to the DG method using a piecewise constant polynomial. Consequently, the DG(P_k) methods with k>0 can be regarded as a natural generalization of finite volume methods to higher order methods. By simply increasing the degree P of the polynomials, the DG methods of corresponding higher order are obtained.

In the traditional DG method, numerical polynomial solutions $\mathbf{U}_h$ in each element are expressed using either standard Lagrange finite element or hierarchical node-based basis as following

$$\mathbf{U}_h = \sum_{i=1}^{\mathrm{N}} \mathbf{U}_i(t) B_i(x) \tag{3.6}$$

where B_i is the finite element basis function. As a result, the unknowns to be solved are the variables at the nodes $\mathbf{U}_i$. On each cell, a system of $N\times N$ has to be solved, where polynomial solutions are dependent on the shape of elements. For example, for a linear polynomial approximation in 3D, a linear polynomial is used for tetrahedral elements and the unknowns to be solved are the variables at the four. However, numerical polynomial solutions $\mathbf{U}$ can be expressed in other forms as well. In the present work, the numerical polynomial solutions are represented using a Taylor series expansion at the center of the cell. For example, if we do a Taylor series expansion at the cell centroid, the quadratic polynomial solutions can be expressed as follows

$$\begin{aligned}\mathbf{U}_h &= \mathbf{U}_c + \frac{\partial \mathbf{U}}{\partial x}|_c(x-x_c) + \frac{\partial \mathbf{U}}{\partial y}|_c(y-y_c) + \frac{\partial \mathbf{U}}{\partial z}|_c(z-z_c) \\ &+ \frac{\partial^2 \mathbf{U}}{\partial x^2}|_c\frac{(x-x_c)^2}{2} + \frac{\partial^2 \mathbf{U}}{\partial y^2}|_c\frac{(y-y_c)^2}{2} + \frac{\partial^2 \mathbf{U}}{\partial z^2}|_c\frac{(z-z_c)^2}{2} \\ &+ \frac{\partial^2 \mathbf{U}}{\partial x\partial y}|_c(x-x_c)(y-y_c) + \frac{\partial^2 \mathbf{U}}{\partial x\partial z}|_c(x-x_c)(z-z_c) + \frac{\partial^2 \mathbf{U}}{\partial y\partial z}|_c(y-y_c)(z-z_c)\end{aligned} \tag{3.7}$$

which can be further expressed as cell-averaged values and their derivatives at the center of the cell

$$\begin{aligned}\mathbf{U}_h &= \tilde{\mathbf{U}} + \frac{\partial \mathbf{U}}{\partial x}|_c B_2 + \frac{\partial \mathbf{U}}{\partial y}|_c B_3 + \frac{\partial \mathbf{U}}{\partial z}|_c B_4 \\ &+ \frac{\partial^2 \mathbf{U}}{\partial x^2}|_c B_5 + \frac{\partial^2 \mathbf{U}}{\partial y^2}|_c B_6 + \frac{\partial^2 \mathbf{U}}{\partial z^2}|_c B_7 + \frac{\partial^2 \mathbf{U}}{\partial x\partial y}|_c B_8 + \frac{\partial^2 \mathbf{U}}{\partial x\partial z}|_c B_9 + \frac{\partial^2 \mathbf{U}}{\partial y\partial z}|_c B_{10}\end{aligned} \tag{3.8}$$

where $\tilde{\mathbf{U}}$ is the mean value of $\mathbf{U}$ in this cell and the ten basis functions are as below. The unknowns to be solved in this formulation are the cell-averaged variables and their derivatives at the center of the cells. The dimension of the polynomial space is ten and the ten basis functions are

$$\begin{aligned}&B_1 = 1\,, B_2 = x - x_c\,, B_3 = y - y_c\,, B_4 = z - z_c \\ &B_5 = \frac{{B_2}^2}{2} - \frac{1}{\Omega_e}\int_{\Omega_e}\frac{{B_2}^2}{2}d\Omega\,, B_6 = \frac{{B_3}^2}{2} - \frac{1}{\Omega_e}\int_{\Omega_e}\frac{{B_3}^2}{2}d\Omega\,, B_7 = \frac{{B_4}^2}{2} - \frac{1}{\Omega_e}\int_{\Omega_e}\frac{{B_4}^2}{2}d\Omega \\ &B_8 = B_2B_3 - \frac{1}{\Omega_e}\int_{\Omega_e}B_2B_3d\Omega\,, B_9 = B_2B_4 - \frac{1}{\Omega_e}\int_{\Omega_e}B_2B_4d\Omega\,, B_{10} = B_3B_4 - \frac{1}{\Omega_e}\int_{\Omega_e}B_3B_4d\Omega\end{aligned} \tag{3.9}$$

The discontinuous Galerkin formulation then leads to the following ten equations

$$\begin{aligned}&\frac{d}{dt}\int_{\Omega_e}\tilde{\mathbf{U}}d\Omega + \int_{\Gamma_e}\mathbf{F}_k(\mathbf{U}_h)\mathbf{n}_k d\Gamma = 0, \quad i = 1 \\ &\boldsymbol{M}_{9\times 9}\frac{d}{dt}\left(\frac{\partial \mathbf{U}}{\partial x}|_c \;\; \frac{\partial \mathbf{U}}{\partial y}|_c \;\; \frac{\partial \mathbf{U}}{\partial z}|_c \;\; \frac{\partial^2 \mathbf{U}}{\partial x^2}|_c \;\; \frac{\partial^2 \mathbf{U}}{\partial y^2}|_c \;\; \frac{\partial^2 \mathbf{U}}{\partial z^2}|_c \;\; \frac{\partial^2 \mathbf{U}}{\partial x\partial y}|_c \;\; \frac{\partial^2 \mathbf{U}}{\partial x\partial z}|_c \;\; \frac{\partial^2 \mathbf{U}}{\partial y\partial z}|_c\right)^T + \mathbf{R}_{9\times 1} = 0\end{aligned} \tag{3.10}$$

Note that in this formulation, equations for the cell-averaged variables are decoupled from equations for their derivatives due to the judicial choice of the basis functions and the fact that

$$\int_{\Omega_e} B_1B_i d\Omega = 0, \qquad 2 \le i \le 10 \tag{3.11}$$

In the implementation of this DG method, the basis functions are actually normalized in order to improve the conditioning of the system matrix (3.5) as follows:

$$B_1 = 1\,, B_2 = \frac{x - x_c}{\Delta x}\,, B_3 = \frac{y - y_c}{\Delta y}\,, B_4 = \frac{z - z_c}{\Delta z}$$

$$B_5 = \frac{{B_2}^2}{2} - \frac{1}{\Omega_e}\int_{\Omega_e} \frac{{B_2}^2}{2} d\Omega\,, B_6 = \frac{{B_3}^2}{2} - \frac{1}{\Omega_e}\int_{\Omega_e} \frac{{B_3}^2}{2} d\Omega\,, B_7 = \frac{{B_4}^2}{2} - \frac{1}{\Omega_e}\int_{\Omega_e} \frac{{B_4}^2}{2} d\Omega \tag{3.12}$$

$$B_8 = B_2 B_3 - \frac{1}{\Omega_e}\int_{\Omega_e} B_2 B_3 d\Omega\,, B_9 = B_2 B_4 - \frac{1}{\Omega_e}\int_{\Omega_e} B_2 B_4 d\Omega\,, B_{10} = B_3 B_4 - \frac{1}{\Omega_e}\int_{\Omega_e} B_3 B_4 d\Omega$$

where $\Delta x = 0.5(x_{max} - x_{min})$, $\Delta y = 0.5(y_{max} - y_{mn})$, and $\Delta z = 0.5(z_{max} - z_{min})$ and x_{max} , y_{max}, z_{max} and x_{min}, y_{min}, z_{min} are the maximum and minimum coordinates in the cell Ω_e in *x*-, *y*- and *z*-directions, respectively. A quadratic polynomial solution can then be rewritten as

$$\mathbf{U}_h = \widetilde{\mathbf{U}} + \frac{\partial \mathbf{U}}{\partial x}|_c \Delta x B_2 + \frac{\partial \mathbf{U}}{\partial y}|_c \Delta y B_3 + \frac{\partial \mathbf{U}}{\partial z}|_c \Delta z B_4 + \frac{\partial^2 \mathbf{U}}{\partial x^2}|_c \Delta x^2 B_5 + \frac{\partial^2 \mathbf{U}}{\partial y^2}|_c \Delta y^2 B_6 + \frac{\partial^2 \mathbf{U}}{\partial z^2}|_c \Delta z^2 B_7 + \frac{\partial^2 \mathbf{U}}{\partial x \partial y}|_c \Delta x \Delta y B_8 + \frac{\partial^2 \mathbf{U}}{\partial x \partial z}|_c \Delta x \Delta z B_9 + \frac{\partial^2 \mathbf{U}}{\partial y \partial z}|_c \Delta y \Delta z B_{10} \tag{3.13}$$

The above normalization is especially important to alleviate the stiffness of the system matrix for higher-order DG approximations.

This formulation allows us to clearly see the similarities and differences between the DG and FV methods. In fact, the discretized governing equations for the cell-averaged variables and the assumption of polynomial solutions on each cell are exactly the same for both methods. The only difference between them is the way how they obtain high-order (>1) polynomial solutions. In the finite volume methods, the polynomial solution of degrees *p* are reconstructed using information from the cell-averaged values of the flow variables, which can be obtained using either TVD/MUSCL or ENO/WENO reconstruction schemes. Unfortunately, the multi-dimensional MUSCL approach suffers from two shortcomings in the context of unstructured grids: 1) Uncertainty and arbitrariness in choosing the stencils and methods to compute the gradients in the case of linear reconstruction; This explains why a nominally second-order finite volume scheme is hardly able to deliver a formal solution of the second order accuracy in practice for unstructured grids. The situation becomes even more evident, severe, and profound, when a highly stretched tetrahedral grid is used in the boundary layers. Many studies, as reported by many researchers [39-41] have demonstrated that it is difficult to obtain a second-order accurate flux reconstruction on highly stretched tetrahedral grids and that for the discretization of inviscid fluxes, the classic 1D-based upwind schemes using median-dual finite volume approximation suffer from excessive numerical diffusion due to such skewing. 2) Extended stencils required for the reconstruction of higher-order (>1st) polynomial solutions. This is exactly the reason why the current finite-volume methods using the TVD/MUSCL reconstruction are not practical at higher order and have remained second-order on unstructured grids. When the ENO/WENO reconstruction schemes are used for the construction of a polynomial of degree *p* on unstructured grids, the dimension of the polynomial space *N=N (p,d)* depends on the degree of the polynomials of the expansion *p*, and the number of spatial dimensions *d*. One must have three, six, and ten cells in 2D and four, ten, and twenty cells in 3D for the construction of a linear, quadratic, cubic Lagrange polynomial, respectively. Undoubtedly, it is an overwhelmingly challenging, if not practically impossible, task to judiciously choose a set of admissible and proper stencils that have such a large number of cells on unstructured grids especially for higher order polynomials and higher dimensions. This explains why the application of higher-order ENO/WENO methods hardly exists on unstructured grids, in spite of their tremendous success on structured grids and their superior performance over the MUSCL/TVD methods. Unlike the FV methods, where the derivatives are reconstructed using cell average values of the neighboring cells, the DG method computes the derivatives in a manner similar to the mean variables. This is compact, rigorous, and elegant mathematically in contrast with arbitrariness characterizing the reconstruction schemes with respect how to compute the derivatives and how to choose the stencils used in the FV methods. It is our believe that this is one of the main reasons why the second order DG methods are more accurate than the FV methods using either TVD/MUSCL or ENO/WENO reconstruction schemes and are less dependent on the mesh regularity, which has been demonstrated numerically [13]. Furthermore, the higher order DG methods can be easily constructed by simply increasing the degree *p* of the polynomials locally, in contrast to the finite volume methods which use the extended stencils to achieve higher order of accuracy.

However, in comparison with reconstructed FV methods, the DG methods have a significant drawback in that they require more degrees of freedom, an additional domain integration, and more Gauss quadrature points for the boundary integration, and therefore more computational costs and storage requirements. On one hand, the reconstruction methods that FV methods use to achieve higher-order accuracy are relatively inexpensive but less accurate and robust. One the other hand, DG methods that can be viewed as a different way to extend a FV method to higher orders are accurate and robust but costly. It is only natural and tempting to combine the efficiency of the reconstruction methods and the accuracy of the DG methods. This idea was originally introduced by Dumbser et al in the frame of P_nP_m scheme [18-20], where Pn indicates that a piecewise polynomial of degree of n is used to represent a DG solution, and Pm represents a reconstructed polynomial solution of degree of m (m ≥ n) that is used to compute the fluxes and source terms. The beauty of P_nP_m schemes is that they provide a unified formulation for both finite volume and DG methods, and contain both classical finite volume and standard DG methods as two special cases of P_nP_m schemes, and thus allow for a direct efficiency comparison. When n=0, i.e. a piecewise constant polynomial is used to represent a numerical solution, P0Pm is nothing but classical high order finite volume schemes, where a polynomial solution of degree m (m ≥1) is reconstructed from a piecewise constant solution. When m=n, the reconstruction reduces to the identity operator, and PnPm scheme yields a standard DG method. Clearly, an accurate and efficient reconstruction is the key ingredient in extending the underlying DG method to higher order accuracy. Although our discussion in this work is mainly focused on the linear DG method, its extension to higher order DG methods is straightforward. In the case of DG(P_1) method, a linear polynomial solution $\mathbf{U}_i$ in any cell *i* is

$$\mathbf{U}_i = \widetilde{\mathbf{U}}_i + \frac{\partial \mathbf{U}}{\partial x}|_i \Delta x_i B_2 + \frac{\partial \mathbf{U}}{\partial y}|_i \Delta y_i B_3 + \frac{\partial \mathbf{U}}{\partial z}|_i \Delta z_i B_4 \tag{3.14}$$

Using this underlying linear polynomial DG solution in the neighboring cells, one can reconstruct a quadratic polynomial solution $\mathbf{U}_i^R$ as follows:

$$\mathbf{U}_i^R = \widetilde{\mathbf{U}}_i^R + \frac{\partial \mathbf{U}^R}{\partial x}|_i \Delta x_i B_2 + \frac{\partial \mathbf{U}^R}{\partial y}|_i \Delta y_i B_3 + \frac{\partial \mathbf{U}^R}{\partial z}|_i \Delta z_i B_4 + \frac{\partial^2 \mathbf{U}^R}{\partial x^2}|_i \Delta {x_i}^2 B_5 \tag{3.15}$$
$$+ \frac{\partial^2 \mathbf{U}^R}{\partial y^2}|_i \Delta {y_i}^2 B_6 + \frac{\partial^2 \mathbf{U}^R}{\partial z^2}|_i \Delta {z_i}^2 B_7 + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial y}|_i \Delta x_i \Delta y_i B_8 + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial z}|_i \Delta x_i \Delta z_i B_9 + \frac{\partial^2 \mathbf{U}^R}{\partial y \partial z}|_i \Delta y_i \Delta z_i B_{10}$$

In order to maintain the compactness of the DG methods, the reconstruction is required to only involve Von Neumann neighborhood, i.e., the adjacent cells that share a face with the cell *i* under consideration. There are ten degrees of freedom, and therefore ten unknowns to be determined. However, the first four unknowns can be trivially obtained, by requiring that the reconstruction scheme has to be conservative, a fundamental requirement, and the values of the reconstructed first derivatives are equal to the ones of the first derivatives of the underlying DG solution at the centroid *i*. Due to the judicious choice of Taylor basis in our DG formulation, these four degrees of freedom (cell average and slopes) simply coincide with the ones from the underlying DG solution, i.e.,

$$\widetilde{\mathbf{U}}_i^R = \widetilde{\mathbf{U}}_i \qquad \frac{\partial \mathbf{U}^R}{\partial x}|_i = \frac{\partial \mathbf{U}}{\partial x}|_i \qquad \frac{\partial \mathbf{U}^R}{\partial y}|_i = \frac{\partial \mathbf{U}}{\partial y}|_i \qquad \frac{\partial \mathbf{U}^R}{\partial z}|_i = \frac{\partial \mathbf{U}}{\partial z}|_i \tag{3.16}$$

The remaining six degrees of freedom can be determined by requiring that the reconstructed solution and its first derivatives are equal to the underlying DG solution and its first derivatives for all the adjacent face neighboring cells. Consider a neighboring cell *j*, one requires

$$
\begin{aligned}
\mathbf{U}_j \;&= \widetilde{\mathbf{U}}_i^R + \frac{\partial \mathbf{U}^R}{\partial x}\Big|_i \Delta x_i B_2 + \frac{\partial \mathbf{U}^R}{\partial y}\Big|_i \Delta y_i B_3 + \frac{\partial \mathbf{U}^R}{\partial z}\Big|_i \Delta z_i B_4 + \frac{\partial^2 \mathbf{U}^R}{\partial x^2}\Big|_i {\Delta x_i}^2 B_5 + \frac{\partial^2 \mathbf{U}^R}{\partial y^2}\Big|_i {\Delta y_i}^2 B_6 \\
&+ \frac{\partial^2 \mathbf{U}^R}{\partial z^2}\Big|_i {\Delta z_i}^2 B_7 + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial y}\Big|_i \Delta x_i \Delta y_i B_8 + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial z}\Big|_i \Delta x_i \Delta z_i B_9 + \frac{\partial^2 \mathbf{U}^R}{\partial y \partial z}\Big|_i \Delta y_i \Delta z_i B_{10} \\
\frac{\partial \mathbf{U}}{\partial x}\Big|_j &= \frac{\partial \mathbf{U}^R}{\partial x}\Big|_i \Delta x_i \frac{1}{\Delta x_i} + \frac{\partial^2 \mathbf{U}^R}{\partial x^2}\Big|_i {\Delta x_i}^2 \frac{B_2}{\Delta x_i} + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial y}\Big|_i \Delta x_i \Delta y_i \frac{B_3}{\Delta x_i} + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial z}\Big|_i \Delta x_i \Delta z_i \frac{B_4}{\Delta x_i} \qquad (3.17) \\
\frac{\partial \mathbf{U}}{\partial y}\Big|_j &= \frac{\partial \mathbf{U}^R}{\partial y}\Big|_i \Delta y_i \frac{1}{\Delta y_i} + \frac{\partial^2 \mathbf{U}^R}{\partial y^2}\Big|_i {\Delta y_i}^2 \frac{B_3}{\Delta y_i} + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial y}\Big|_i \Delta x_i \Delta y_i \frac{B_2}{\Delta y_i} + \frac{\partial^2 \mathbf{U}^R}{\partial y \partial z}\Big|_i \Delta y_i \Delta z_i \frac{B_4}{\Delta y_i} \\
\frac{\partial \mathbf{U}}{\partial z}\Big|_j &= \frac{\partial \mathbf{U}^R}{\partial z}\Big|_i \Delta z_i \frac{1}{\Delta z_i} + \frac{\partial^2 \mathbf{U}^R}{\partial z^2}\Big|_i {\Delta z_i}^2 \frac{B_4}{\Delta z_i} + \frac{\partial^2 \mathbf{U}^R}{\partial x \partial z}\Big|_i \Delta x_i \Delta z_i \frac{B_2}{\Delta z_i} + \frac{\partial^2 \mathbf{U}^R}{\partial y \partial z}\Big|_i \Delta y_i \Delta z_i \frac{B_3}{\Delta z_i}
\end{aligned}
$$

where the basis function *B* are evaluated at the center of cell *j*, i.e., *B*=*B*(x_j, y_j). Its matrix form can be written as

$$
\begin{pmatrix}
B_5^j & B_6^j & B_7^j & B_8^j & B_9^j & B_{10}^j \\
B_2^j & 0 & 0 & B_3^j & B_4^j & 0 \\
0 & B_3^j & 0 & B_2^j & 0 & B_4^j \\
0 & 0 & B_4^j & 0 & B_2^j & B_3^j
\end{pmatrix}
\begin{pmatrix}
\mathbf{U}_{xxi}^R \\ \mathbf{U}_{yyi}^R \\ \mathbf{U}_{zzi}^R \\ \mathbf{U}_{xyi}^R \\ \mathbf{U}_{xzi}^R \\ \mathbf{U}_{yzi}^R
\end{pmatrix}
=
\begin{pmatrix}
\mathbf{R}_1^j \\ \mathbf{R}_2^j \\ \mathbf{R}_3^j \\ \mathbf{R}_4^j
\end{pmatrix}
\qquad (3.18)
$$

where **R** is used to represent the right-hand-side for simplicity. Similar equations could be written for all cells connected to the cell *i* with a common face, which leads to a non-square matrix. The number of face-neighboring cells for a tetrahedral and hexahedral cell is four and six, respectively. As a result, the size of the resulting non-square matrix is 16×6 and 24×6, respectively. This over-determined linear system of 16 or 24 equations for 6 unknowns can be solved in the least-squares sense. In the present work, it is solved using a normal equation approach, which, by pre-multiplying through by matrix transpose, yields a symmetric linear system of equations 6×6. The linear system of 6×6 can be then trivially solved to obtain the second derivatives of the reconstructed quadratic polynomial solution.

This linear reconstruction-based RDG(P_1P_2) method is able to achieve the designed third order of accuracy and significantly improve the accuracy of the underlying second-order DG method for solving the 2D compressible Euler equations on arbitrary grids [42-46]. However, when used to solve the 3D compressible Euler equations on tetrahedral grids, this RDG method suffers from the so-called linear instability, that is also observed in the second-order cell-centered finite volume methods, i.e., RDG(P_0P_1) [29]. This linear instability is attributed to the fact that the reconstruction stencils only involve the von Neumann neighborhood, i.e., adjacent face-neighboring cells [29]. The linear stability can be achieved using extended stencils, which will unfortunately sacrifice the compactness of the underlying DG methods. Also, such a linear reconstruction-based DG method will suffer from non-linear instability, leading to non-physical oscillations in the vicinity of strong discontinuities for the compressible Euler equations. Alternatively, ENO, WENO, or HWENO can be used to reconstruct a higher-order polynomial solution, which can not only enhance the order of accuracy of the underlying DG method but also achieve both linear and non-linear stability. In the present work, the reconstructed quadratic polynomial based on the Hermite WENO on cell *i* are a convex combination of the least-squares reconstructed second derivatives at the cell itself and its four face-neighboring cells

$$\frac{\partial^2 \mathbf{U}}{\partial x_i \partial x_j}|_i = \sum_{k=1}^{5} w_k \frac{\partial^2 \mathbf{U}}{\partial x_i \partial x_j}|_k \tag{3.19}$$

where the weights w_k are computed as

$$w_k = \frac{(\varepsilon + o_k)^{-\gamma}}{\sum_{i=1}^{5} (\varepsilon + o_i)^{-\gamma}} \tag{3.20}$$

where ε is a small positive number used to avoid division by zero, o_k the oscillation indicator for the reconstructed second order polynomials, and γ an integer parameter to control how fast the non-linear weights decay for non-smooth stencils. The oscillation indicator is defined as

$$o_k = \left[\int_{\Omega_i} \left(\frac{\partial^2 \mathbf{U}}{\partial x_i \partial x_j}|_k \right)^2 d\Omega \right]^2 \tag{3.21}$$

Note that the least-squares reconstructed polynomial at the cell itself serves as the central stencil and the least-squares reconstructed polynomials on its four face-neighboring cells act as biased stencils in this WENO reconstruction. This reconstructed quadratic polynomial solution is then used to compute the domain and boundary integrals of the underlying DG(P_1) method in Eq. (3.5). The resulting RDG(P_1P_2) method, is expected to have third order of accuracy at a moderate increase of computing costs in comparison to the underlying DG(P_1) method. The extra costs are mainly due to the least-squares reconstruction, which is relatively cheap in comparison to the evaluation of fluxes, and an extra Gauss quadrature point, which is required to calculate both domain and boundary integrals. In comparison to DG(P_2), this represents a significant saving in terms of flux evaluations. Furthermore, the number of degree of freedom is greatly reduced, which leads to a significant reduction in memory requirements, and from which implicit methods will benefit tremendously. The cost analysis for the RDG(P_1P_1) (DG(P_1)), RDG(P_1P_2) and RDG(P_2P_2) (DG(P_2)) is summarized in Table 1, where the memory requirement for storing only the implicit diagonal matrix is given as well, and which grows quadratically with the order of the DG methods. The storage requirement for the implicit DG methods is extremely demanding, especially for higher-order DG methods.

Table 1 Cost analysis for different RDG(P_mP_n) methods on tetrahedral grids

	RDG(P_1P_1)	RDG(P_1P_2)	RDG(P_2P_2)
Number of quadrature points for boundary integrals	3	4	7
Number of quadrature points for domain integrals	4	5	11
Reconstruction	No	Yes	No
Order of accuracy	$O(h^2)$	$O(h^3)$	$O(h^3)$
Storage for implicit diagonal matrix	400 word per element	400	2500

4 Implicit Time Discretization

The spatially discretized governing equations must be integrated in time to obtain a steady-state solution. In the previous work [42], the Runge-Kutta time-stepping method was used to compute the time integration with a slow converging speed. In order to efficiently converge the solution, implicit time integration methods must be used for the reconstructed discontinuous Galerkin methods. Eq. (3.5) can be written be a semi-discrete form as

$$\boldsymbol{V}_i \frac{\partial \mathbf{U}_i}{\partial t} = \mathbf{R}_i \tag{4.1}$$

where $\boldsymbol{V}_i$ is the mass matrix and of the cell i, and $\mathbf{R}_i$ is the right hand side residual and equals zero for a steady-state solution. Using the Euler implicit time integration, Eq. (4.1) can be further written in discrete form as

$$\boldsymbol{V}_i \frac{\Delta \mathbf{U}_i^n}{\Delta t} = \mathbf{R}_i^{n+1} \tag{4.2}$$

where Δt is the time increment and $\Delta \mathbf{U}^n$ is the difference of conservative variables between time level n and n+1, i.e.,

$$\Delta \mathbf{U}^n = \mathbf{U}^{n+1} - \mathbf{U}^n \tag{4.3}$$

Eq. (4.2) can be linearized in time as

$$\boldsymbol{V}_i \frac{\Delta \mathbf{U}_i^n}{\Delta t} = \mathbf{R}_i^n + \frac{\partial \mathbf{R}_i^n}{\partial \mathbf{U}} \Delta \mathbf{U}_i \tag{4.4}$$

Writing the equation for all cells leads to the delta form of the backward Euler scheme

$$\boldsymbol{A} \Delta \mathbf{U} = \mathbf{R} \tag{4.5}$$

where

$$\boldsymbol{A} = \frac{\boldsymbol{V}}{\Delta t} \mathbf{I} - \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}} \tag{4.6}$$

Note that as Δt tends to infinity, the scheme reduces to the standard Newton's method for solving a system of nonlinear equations. It's known that the Newton's method has a quadratic convergence property. The term $\partial \mathbf{R}/\partial \mathbf{U}$ represents symbolically the Jacobian matrix. It involves the linearization of both inviscid and viscous flux vectors. To obtain the quadratic convergence of Newton's method, the linearization of the numerical flux function must be virtually exact. However, explicit formation of the Jacobian matrix resulting from the exact linearization of the third order numerical flux functions using the reconstructed unknown variables for inviscid fluxes requires excessive storage and is extremely expensive. An approximated second-order representation of the numerical fluxes is linearized. In each cell, the conservative variables are evaluated at every Gauss quadrature point with all 10 degrees of freedom (including the reconstructed 6 degrees), the same with DG(P_2), while the matrix $\boldsymbol{A}$ maintains a DG(P_1) structure ((4 $\times$ *neqns*) $\times$ (4 $\times$ *neqns*)), where *neqns* (=5 in 3D) is the number of the unknown variables. Although the number of time steps may increase, the memory requirement and cost per time step is significantly reduced: it takes less CPU time and fewer Gauss quadrature points than implicit DG(P_2) methods to compute the Jacobian matrix, thus reduces computational cost to solve the resulting linear system.

The following flux function of Lax-Friedrichs scheme, which was previously used to obtain the left-hand-side (LHS) Jacobian matrix for the implicit finite volume methods, has showed its high efficiency for both 2D and 3D compressible flow problems [36].

$$\mathbf{R}_i(\mathbf{U}_i, \mathbf{U}_i, \mathbf{n}_{ij}) = \frac{1}{2}\left(\mathbf{F}(\mathbf{U}_i, \mathbf{n}_{ij}) + \mathbf{F}(\mathbf{U}_j, \mathbf{n}_{ij})\right) - \frac{1}{2}|\lambda_{ij}|(\mathbf{U}_j - \mathbf{U}_i) \tag{4.7}$$

where

$$\lambda_{ij} = |\mathbf{V}_{ij} \cdot \mathbf{n}_{ij}| + C_{ij} \tag{4.8}$$

where $\mathbf{V}_{ij}$ is the velocity vector and C_{ij} is the speed of sound. The linearization of the flux function yields

$$\frac{\partial \mathbf{R}_i}{\partial \mathbf{U}_i} = \frac{1}{2}\left(J(\mathbf{U}_i) + |\lambda_{ij}|\mathbf{I}\right) \tag{4.9}$$

$$\frac{\partial \mathbf{R}_i}{\partial \mathbf{U}_j} = \frac{1}{2}\left(J(\mathbf{U}_j) - |\lambda_{ij}|\mathbf{I}\right) \tag{4.10}$$

where $J = \partial\mathbf{F}/\partial\mathbf{U}$ represents the Jacobian of the inviscid flux vector. However, the extension of this simplified flux linearization to either the implicit DG(P_1) or implicit RDG(P_1P_2) methods on tetrahedral grids suffers from *CFL* constraints and leads to inefficient converging speed. Therefore the performance of implicit time integration would be greatly impaired if this version of Jacobian representation is applied.

Alternatively, $\partial\mathbf{F}/\partial\mathbf{U}$ can be formulated according to the method proposed by Batten et.al [47], in which the average-state Jacobians with frozen acoustic wave-speed are considered. This method can guarantee an uncompromised converging speed and the robustness is preserved. The implicit form of the HLLC flux is then given by

$$\mathbf{F}_{HLLC}^{n+1} = \begin{cases} \mathbf{F}_i^n + \dfrac{\partial \mathbf{F}_i}{\partial \mathbf{U}_i}\Delta\mathbf{U}_i & if\ S_i^n > 0 \\ (\mathbf{F}_i^*)^n + \dfrac{\partial \mathbf{F}_i^*}{\partial \mathbf{U}_i}\Delta\mathbf{U}_i + \dfrac{\partial \mathbf{F}_i^*}{\partial \mathbf{U}_j}\Delta\mathbf{U}_j & if\ S_i^n \le 0 < S_M^n \\ (\mathbf{F}_j^*)^n + \dfrac{\partial \mathbf{F}_j^*}{\partial \mathbf{U}_i}\Delta\mathbf{U}_i + \dfrac{\partial \mathbf{F}_j^*}{\partial \mathbf{U}_j}\Delta\mathbf{U}_j & if\ S_M^n \le 0 < S_j^n \\ \mathbf{F}_j^n + \dfrac{\partial \mathbf{F}_j}{\partial \mathbf{U}_j}\Delta\mathbf{U}_j & if\ S_j^n < 0 \end{cases} \tag{4.11}$$

where

$$S_M = \frac{\rho_j q_j (S_j - q_j) - \rho_i q_i (S_i - q_i) + p_i - p_j}{\rho_j (S_j - q_j) - \rho_i (S_i - q_i)} \tag{4.12}$$

where $q_i = \mathbf{V}_i \cdot \mathbf{n}_{ij}$ and $q_j = \mathbf{V}_j \cdot \mathbf{n}_{ij}$ and

$$S_i = \min[\lambda_1(\mathbf{U}_i), \lambda_1(U^{Roe})] \tag{4.13}$$

$$S_j = \max[\lambda_m(U^{Roe}), \lambda_m(\boldsymbol{U}_j)] \tag{4.14}$$

with $\lambda_1(U^{Roe})$ and $\lambda_m(U^{Roe})$ being the smallest and largest eigenvalues of the Roe matrix [48]. Details of $\partial\mathbf{F}_i^*/\partial\mathbf{U}_i$ and $\partial\mathbf{F}_i^*/\partial\mathbf{U}_j$ can be found in Reference 47. By doing so, the quadratic convergence of the Newton's method can no longer be achieved because of the inexact representation of LHS in Eq. (4.5).

In this study, the linear system at each time step is solved approximately by using a preconditioned matrix-free GMRES solver. Instead of calculating and storing the full matrix, the matrix-free GMRES algorithm only requires the result of matrix-vector products, which can be approximated as

$$\frac{\partial \mathbf{R}(\mathbf{U})}{\partial \mathbf{U}}\Delta\mathbf{U} = \frac{\mathbf{R}(\mathbf{U} + \varepsilon \cdot \Delta\mathbf{U}) - \mathbf{R}(\mathbf{U})}{\varepsilon} \tag{4.15}$$

where $\mathbf{R}(\mathbf{U}+ \varepsilon \cdot\Delta\mathbf{U})$ is the residual for the governing equations computed using perturbed state quantities and ε is a scalar. The parameter ε can be computed as proposed by Pernice and Walker [49]

$$\varepsilon = \epsilon \frac{\sqrt{1 + ||\mathbf{U}||_{L^2}}}{||\Delta\mathbf{U}||_{L^2}} \tag{4.16}$$

where the parameter ϵ is a user-defined input and the choice of this parameter is between round-off and truncation error. In this study $\varepsilon = 10^{-7}$ works without problem for all the computations. The GMRES process requires one flux evaluation per time step and one flux evaluation per inner GMRES iteration.

Since the speed of convergence of an iterative algorithm for a linear system depends on the condition number of the matrix $\boldsymbol{A}$, preconditioning is used to alter the spectrum and therefore accelerate the convergence speed of iterative algorithm. The preconditioning technique involves solving an equivalent preconditioned linear system,

$$\widetilde{\boldsymbol{A}}\Delta\widetilde{\mathbf{U}} = \widetilde{\mathbf{R}} \tag{4.17}$$

instead of the original system in Eq. (4.5), hoping that $\widetilde{\boldsymbol{A}}$ is well conditioned. Left preconditioning involves pre-multiplying the linear system with the inverse of a preconditioning matrix $\boldsymbol{P}$

$$\boldsymbol{P}^{-1}\boldsymbol{A}\Delta\mathbf{U} = \boldsymbol{P}^{-1}\mathbf{R} \tag{4.18}$$

The best preconditioning matrix for $\boldsymbol{A}$ would cluster as many eigenvalues as possible at unity. Obviously, the optimal choice of $\boldsymbol{P}$ is $\boldsymbol{A}$. Preconditioning will be cost-effective only if the additional computational work incurred for each sub-iteration, is compensated for by a reduction in the total number of iterations to convergence. In this way, the total cost of solving the overall nonlinear system is reduced.

Using an edge-based data structure, the matrix $\boldsymbol{A}$ is stored in upper, lower, and diagonal forms, which can be expressed as

$$\boldsymbol{U} = \frac{\partial \mathbf{R}_i(\mathbf{U}_i, \mathbf{U}_j, \boldsymbol{n}_{ij})}{\partial \mathbf{U}_j} \qquad \boldsymbol{L} = -\frac{\partial \mathbf{R}_i(\mathbf{U}_i, \mathbf{U}_j, \boldsymbol{n}_{ij})}{\partial \mathbf{U}_i} \qquad \boldsymbol{D} = \frac{\boldsymbol{V}}{\Delta t}\mathbf{I} + \sum_j \frac{\partial \mathbf{R}_i(\mathbf{U}_i, \mathbf{U}_j, \boldsymbol{n}_{ij})}{\partial \mathbf{U}_i} \tag{4.19}$$

where $\boldsymbol{U}$, $\boldsymbol{L}$, and $\boldsymbol{D}$ represent the strict upper matrix, the strict lower matrix, and the diagonal matrix, respectively. Both upper and lower matrices require a storage of *nedge* $\times$ (*ndegr* $\times$ *neqns*)^2 and the diagonal matrix needs a storage of *nelem* $\times$ (*ndegr* $\times$ *neqns*)^2, where *nedge* is the number of faces, *ndegr* (= 4 for RDG(P_1P_2) in 3D) is the number of degree of freedom, and *nelem* is the number of cells. Therefore the current scheme for the preconditioner is not completely matrix free. In the present work, the LU-SGS method is used as a preconditioner [29] , i.e.,

$$\boldsymbol{P} = (\boldsymbol{D} + \boldsymbol{L})\boldsymbol{D}^{-1}(\boldsymbol{D} + \boldsymbol{U}) \tag{4.20}$$

The structure of the restarted preconditioned matrix-free GMRES algorithm is described below to solve the linear system of equations at each time step.

```
For $l$ = 1, $m$ Do                                   $m$ restarted iterations
    $\mathbf{v}_0 = \mathbf{R} - \boldsymbol{A}\Delta\mathbf{U}_0$                      initial residual
    $\mathbf{r}_0 := \boldsymbol{P}^{-1}\mathbf{v}_0$                          preconditioning step
    $\beta := \|\mathbf{r}_0\|_2$                              initial residual norm
    $\mathbf{v}_1 := \mathbf{r}_0/\beta$                             define initial Krylov
    For $j$ = 1, $k$ Do                               inner iterations
        $\mathbf{y}_j := (\mathbf{R}(\mathbf{U}+\varepsilon\cdot\Delta\mathbf{U})-\mathbf{R}(\mathbf{U}))/\varepsilon$    matrix-vector product
        $\mathbf{w}_j = \boldsymbol{P}^{-1}\mathbf{y}_j$                       preconditioning step
        For $i$ = 1, $j$ Do                           Gram-Schmidt step
            $h_{i,j} := (\mathbf{w}_j,\mathbf{v}_i)$                  …
            $\mathbf{w}_j = \mathbf{w}_j - h_{i,j}\mathbf{v}_i$                ...
        End Do
        $h_{j+1,j} := \|\mathbf{w}_j\|_2$                     ...
        $\mathbf{v}_{j+1} := \mathbf{w}_j / \mathbf{h}_{j+1,j}$                  define Krylov vector
    End Do
    $\mathbf{z} := \min_{\hat{z}}\|\beta\mathbf{e}_1 - H\hat{\mathbf{z}}\|_2$             solve least squares
    $\Delta\mathbf{U} := \Delta\mathbf{U}_0 + \sum_{i=1}^{m}\mathbf{v}_i\mathbf{z}_i$              approximate solution
    if $\|\beta\mathbf{e}_1 - H\hat{\mathbf{z}}\|_2 \ll \epsilon$ exit          convergence check
    $\Delta\mathbf{U}_0 := \Delta\mathbf{U}$                            restart
End Do
```

The primary storage is dictated by the LU-SGS preconditioner, which requires the upper, lower and diagonal matrix to be stored for every non-zero element in the global matrix $\widetilde{\boldsymbol{A}}$ on the left hand side of Eq. (4.17).When evaluating the matrix-vector product by Eq. (4.15), the linearization of the fluxes requires a storage of $nelem \times ndegr \times neqns$. The need for additional storage associated with the GMRES algorithm is an array of size $(k+2) \times nelem \times (ndegr \times neqns)$, where k is the number of search directions. Since the preconditioned matrix-free GMRES algorithm is completely separated from the flux computation procedure, the memory which is used to compute the fluxes can also be shared in the procedure of solving the linear system.

5 Computational Results

The developed implicit Hermite WENO reconstruction-based discontinuous Galerkin method has been used to solve a variety of the compressible flow problems on tetrahedral grids. All of the computations are performed on a Dell Precision T7400 personal workstation computer (2.98 GHz Xeon CPU with 18GBytes memory) using Red Hat 5 Linux operating system. The developed implicit method is used to compete with the explicit three-stage Runge-Kutta time-stepping method for all test cases in order to demonstrate its superior efficiency. The average-state HLLC full Jacobians (Full LHS) and diagonal Jacobians (Diagonal LHS), and Lax-Friedrichs Jacobians (Simplified LHS) are used respectively as the preconditioning matrix for the implicit method in order to verify the effect of accuracy of approximate Jacobian matrix on converging speed. All computations are initiated with uniform flows and are carried out using a *CFL* number of 1 for the explicit method, and 10 for the implicit method for the first 20 steps. The setting of *CFL* number is not considered an emphasis in this paper. A few examples are presented in this section to demonstrate that the developed IRDG(P_1P_2) method is able to maintain the linear stability, achieve the designed third order of accuracy, and significantly improve the accuracy of the underlying second-order DG(P_1) method without significant increase in computing costs and storage requirements.

5.1 A Subsonic Flow through a Channel with a Smooth Bump

This test case is chosen to demonstrate the convergence accuracy and efficiency of the IRDG(P_1P_2) methods for internal flows. The problem under consideration is a subsonic flow inside a 3D channel with a smooth bump on the lower surface. The height, width, and length of the channel are 0.8, 0.8, and 3, respectively. The shape of the lower wall is defined by the function $0.0625exp(-25x^2)$ from x = -1.5 to x = 1.5. The inflow condition is prescribed at a Mach number of 0.5, and an angle of attack of 0^o. Fig. 1 shows the four successively refined tetrahedral grids used for the grid convergence study. The numbers of elements, points, and boundary points for the coarse, medium, fine, and finest grids are (889, 254, 171), (6986, 1555, 691), (55703, 10822, 2711), and (449522, 81567, 10999), respectively. The cell size is halved between consecutive grids. Numerical solutions to this problem are computed using the IRDG(P_1P_1) and IRDG(P_1P_2) methods on the first three grids to obtain a quantitative measurement of the order of accuracy and discretization errors. The following L^2-norm of the entropy production is used as the error measurement

$$\|\varepsilon\|_{L_2(\Omega)} = \sqrt{\int_{\Omega} \varepsilon^2 d\Omega}$$

where the entropy production ε defined as

$$\varepsilon = \frac{S - S_\infty}{S_\infty} = \frac{p}{p_\infty}(\frac{\rho_\infty}{\rho})^\gamma - 1$$

Note that the entropy production, where the entropy is defined as $S = p/\rho^\gamma$, is a very good criterion to measure accuracy of the numerical solutions, since the flow under consideration is isentropic. Fig. 2 illustrates the computed velocity contours in the flow field obtained by the RDG(P_1P_1) and RDG(P_1P_2) methods on the fine grid. Table 2 provides the details of the spatial convergence of the RDG methods for this numerical experiment. Consider the fact that this is a 3D simulation of a 2D problem, and unstructured tetrahedral grids are not symmetric by nature, thus causing error in the z-direction, the second-order RDG(P_1P_1) method can be considered to deliver the designed second order of accuracy. As expected, the RDG(P_1P_2) method offers a full $O(h^{p+2})$ order of the convergence, adding one order of accuracy to the underlying IRDG(P_1P_1) method, thus demonstrate that the IRDG(P_1P_2) method can significantly increase the accuracy of the underlying DG method, and therefore greatly decrease its computing costs.

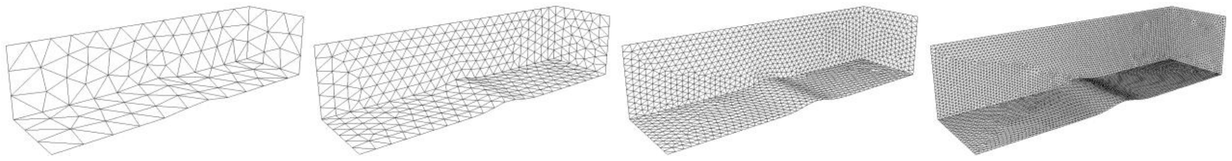


Figure 1. A sequence of four successively globally refined unstructured meshes used for computing subsonic flow in a channel with a smooth bump.

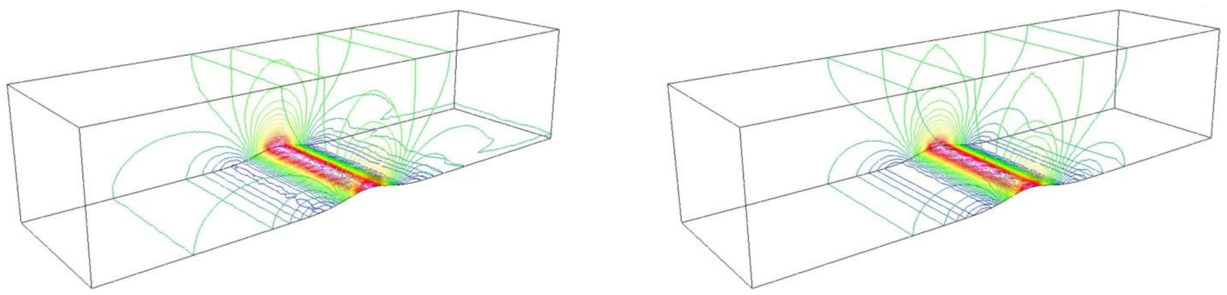


Figure 2. Computed velocity contours in the flow field obtained by the IRDG(P_1P_1) (left) and IRDG(P_1P_2) (right) on the fine mesh for a subsonic flow through a channel with a bump on the lower surface at M_∞=0.5.

Table 2 Convergence order of accuracy for RDG(P_1P_1) and RDG(P_1P_2) methods for a subsonic flow through a channel with a bump on the lower surface at M_∞=0.5.

Cell size	Log(L^2-error)		Order	
	IRDG(P_1P_1)	IRDG(P_1P_2)	IRDG(P_1P_1)	IRDG(P_1P_2)
8X	-2.61293	-2.67090		
4X	-3.13334	-3.56436	1.89	2.80
2X	-3.74301	-4.35115		

The *CFL* number for the implicit method is set to be 500 for all grids after the initial time steps. Fig. 3-5 shows a series of plots of logarithmic density residual versus time steps (left) and CPU time (right) for the explicit and implicit methods using RDG(P_1P_2) on different grids. In comparison, even the implicit method with only the diagonal part of Jacobians is three orders of magnitude faster than its explicit counterpart, indicating the advantages of using implicit time integration scheme under the context of the high-order reconstructed discontinuous Galerkin methods.

On the other side, comparison among the different approximated preconditioning matrix verifies that the converging speed of solution is remarkably affected by how the Jacobians are formulated. On the fine grid, the HLLC full Jacobian matrix leads in the best converging speed in terms CPU time, whereas the HLLC diagonal Jacobian matrix is about six times slower, and simplified full Jacobian matrix is two times slower. It is also found that only the solver with the HLLC full Jacobian matrix maintains stable convergence with *CFL* numbers like 10^4 or even larger, although the speedup does not increase obviously. In contrast, the solver with either of the other two Jacobians suffers instability when *CFL* number of 10^3. In another word, less approximated Jacobians lead to larger condition number, thus lead to more rigorous *CFL* constraints. Also see Fig. 6, the solver with HLLC full Jacobians is over three times faster than that with simplified full Jacobians on the finest grid, indicating the necessity of using well approximated Jacobians on highly refined grids. In fact, for the traditional finite volume methods in 2D and 3D, it's common practice to apply simplified Jacobians for implicit time integration without difficulty. In contrast, for high-order discontinuous Galerkin methods on tetrahedral grids, the efficiency of an implicit solver is closely linked to how accurately the numerical flux function is linearized.

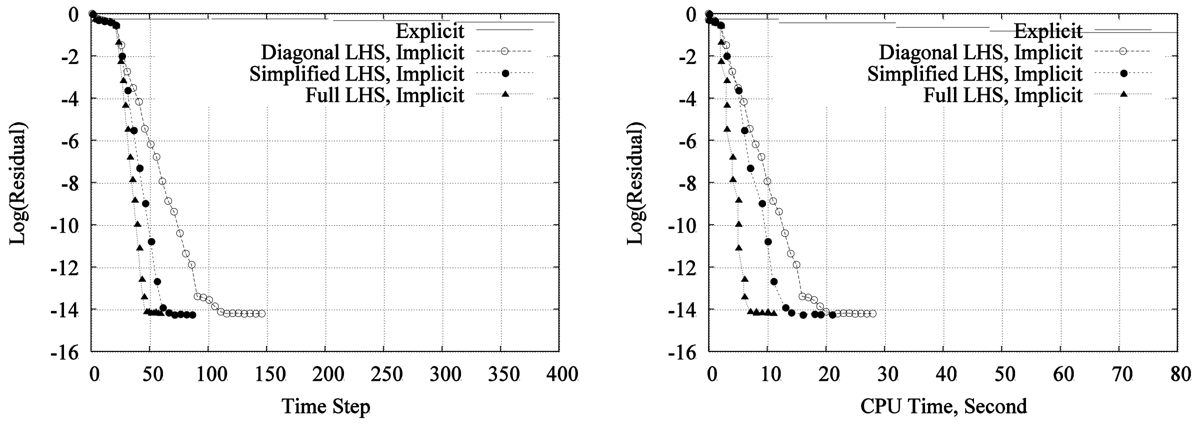


Figure 3. Logarithmic density residual versus time step (left) and CPU time (right) for RDG(P_1P_2) on the coarse mesh for subsonic flow through a channel with a bump on the lower surface M_∞=0.5.

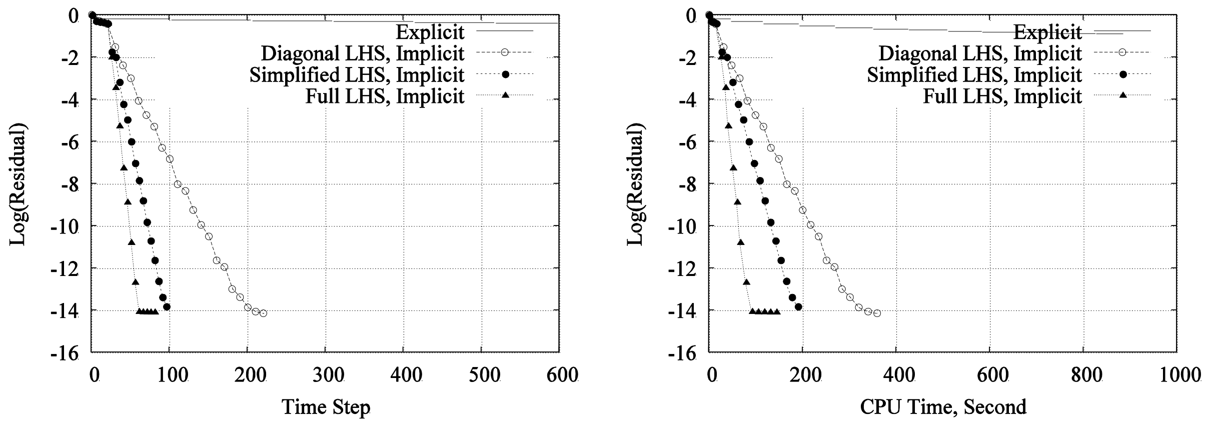


Figure 4. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the medium mesh for a subsonic flow through a channel with a bump on the lower surface at M_∞=0.5.

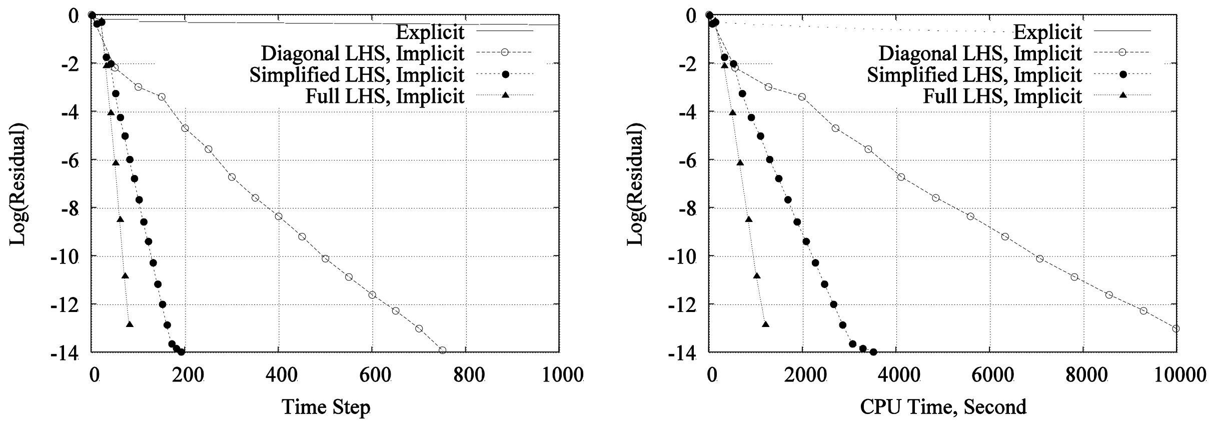


Figure 5. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the fine mesh for a subsonic flow through a channel with a bump on the lower surface at M_∞=0.5.

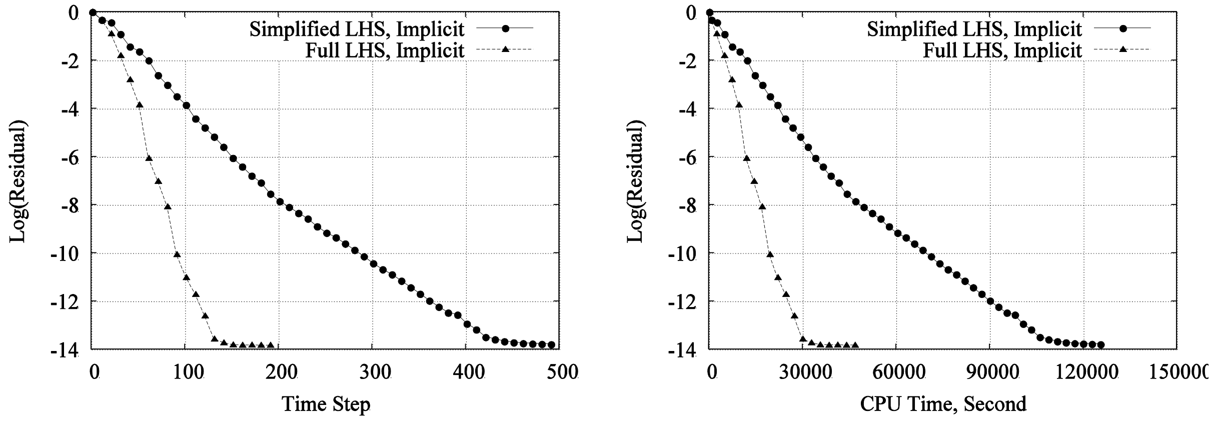


Figure 6. Logarithmic versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the finest mesh for a subsonic flow through a channel with a bump on the lower surface at M_∞=0.5.

5.2 A Subsonic Flow past a Sphere

In this test case, a subsonic flow past a sphere at a Mach number of M_∞=0.5 is chosen to assess if the developed IRDG method can achieve a formal order of convergence rate and high converging speed for external flows. A sequence of the four successively refined tetrahedral grids used in this grid convergence study is shown in Fig. 7. The numbers of elements, points, and boundary points for the coarse, medium, fine, and finest grids are (535, 167, 124), (2426, 598, 322), (16467, 3425, 1188), and (124706, 23462, 4538), respectively. The cell size is halved between consecutive meshes. Note that only a quarter of the configuration is modeled due to the symmetry of the problem, and that the number of elements on a successively refined mesh is not exactly eight times the coarse mesh's elements due to the nature of unstructured grid generation.

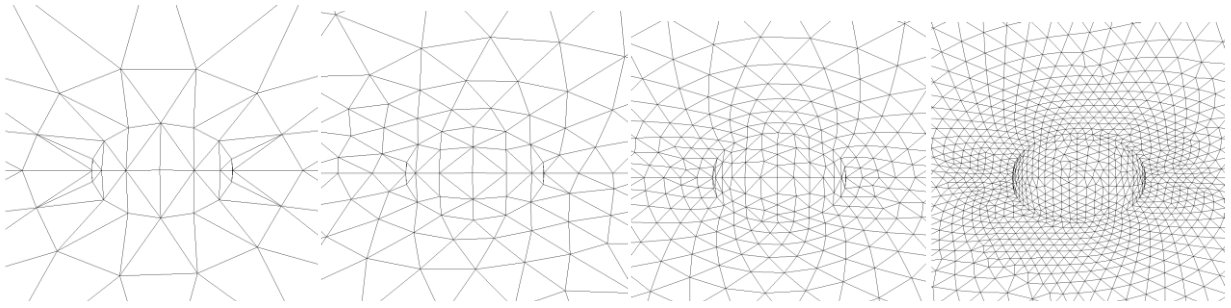


Figure 7. A series of four successively globally refined tetrahedral meshes for computing subsonic flow past a sphere at M_∞=0.5

The computations are conducted on the first three grids using the IRDG(P_1P_1) and IRDG(P_1P_2) methods to obtain a quantitative measurement of the order of accuracy and discretization errors. As in the previous case, the entropy production is used as the error measurement. Fig. 8 illustrates the computed velocity contours in the flow field obtained by the RDG(P_1P_1) and RDG(P_1P_2) methods on the fine grid. One can observe that the RDG(P_1P_2) solution is much more accurate than the RDG(P_1P_1) solution on the same fine grid. Table. 3 provides the details of the spatial convergence of the two IRDG methods for this numerical experiment. Both the IRDG(P_1P_1) and IRDG(P_1P_2) methods achieve higher than expected convergence rates, being 2.36 and 3.55 respectively, convincingly demonstrating the benefits of using a higher-order method. The *CFL* number is set to be 10^3 for all grids after the initial time steps. Fig. 9-11 shows a series of plots of logarithmic density residual versus time step (left) and CPU time (right) for the explicit and implicit RDG(P_1P_2) methods on the four grids. The implicit method is at least four orders of magnitude faster than its explicit counterpart in this test case, again indicating the superior advantages of using implicit time integration schemes. Among the three preconditioners, The HLLC full Jacobians still provide the best converging speed. However, the HLLC diagonal Jacobians are two times faster than the simplified full Jacobians on the fine grid, which is different than in the first test case. Again in this case, it is found that if a very large *CFL* number like 10^4 is used, only the solver with the HLLC full Jacobians is able to maintain stability, whereas with the other two Jacobians, the solver needs more initial time steps before a larger *CFL* number can be safely applied. See Fig. 12, the implicit solver with HLLC Jacobians is three times faster than that with a simplified form on the finest grid, indicating the better robustness and performance of using more accurately approximated Jacobians on highly refined grids.

Table 3 Orders of convergence rate for RDG(P_1P_1) and RDG(P_1P_2) methods for subsonic flow past a sphere at M_∞=0.5

Cell size	Log(L^2-error)		Mean order	
	IRDG(P_1P_1)	IRDG(P_1P_2)	IRDG(P_1P_1)	IRDG(P_1P_2)
8X	-1.74887	-1.97797		
4X	-2.30018	-2.88048	2.36	3.55
2X	-2.90949	-3.70373		

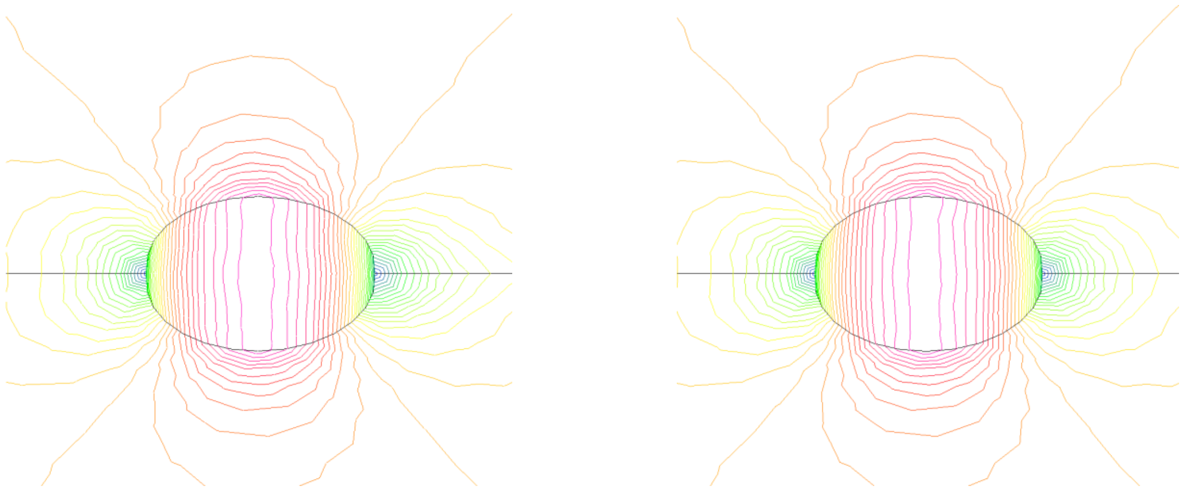


Figure 8. Computed velocity contours in the flow field obtained by the IRDG(P_1P_1) method on the fine grid (left), and IRDG(P_1P_2) on the fine grid (right) for subsonic flow past a sphere at M_∞=0.5.

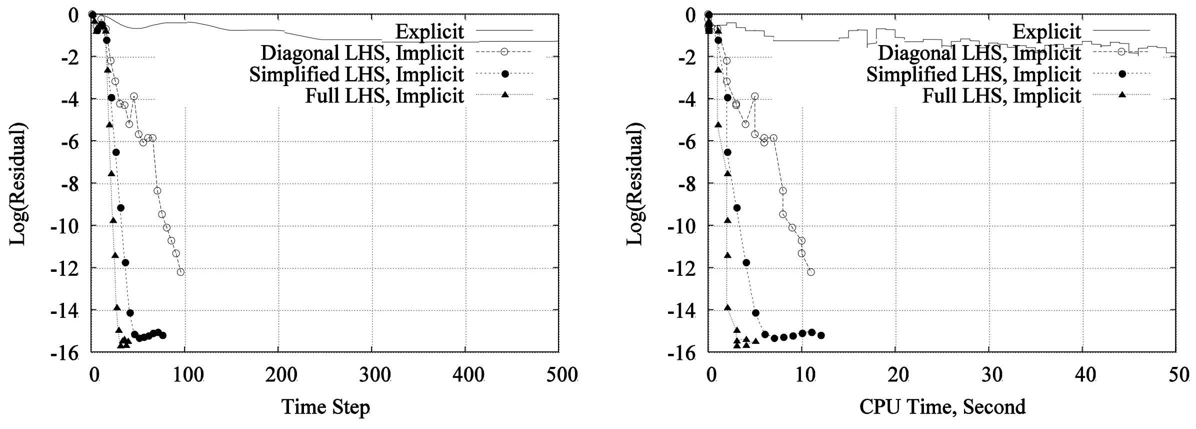


Figure 9. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the coarse mesh for subsonic flow past a sphere M_∞=0.5.

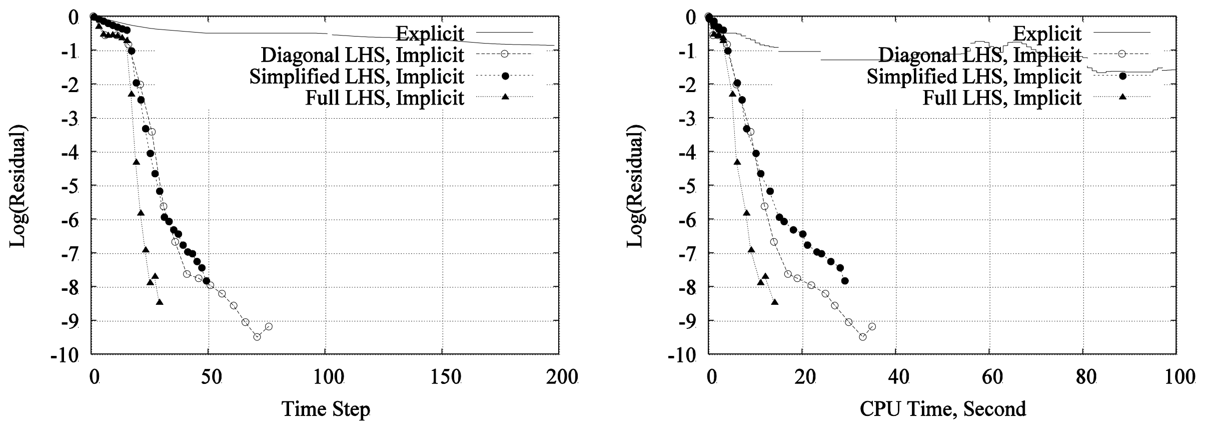


Figure 10. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the medium mesh for subsonic flow past a sphere M_∞=0.5.

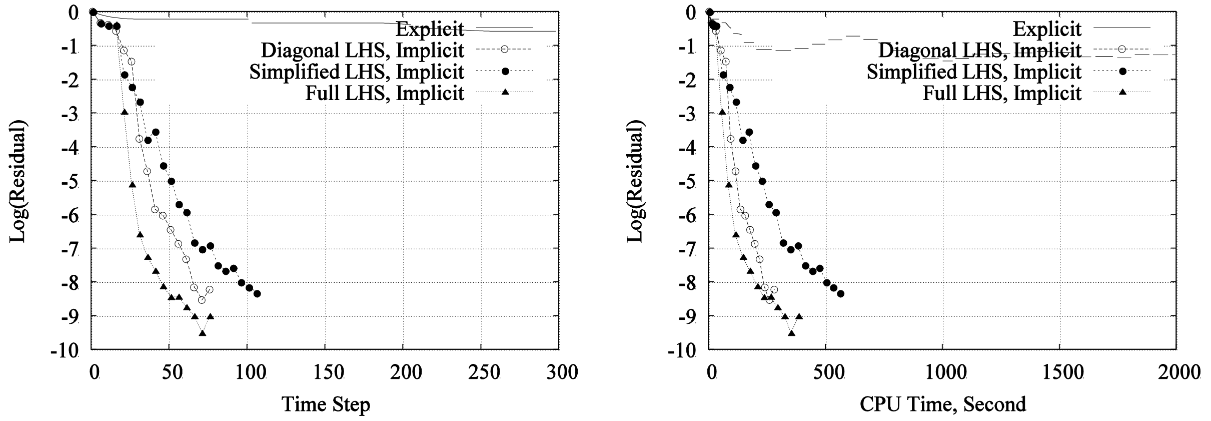


Figure 11. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the fine mesh for subsonic flow past a sphere M_∞=0.5.

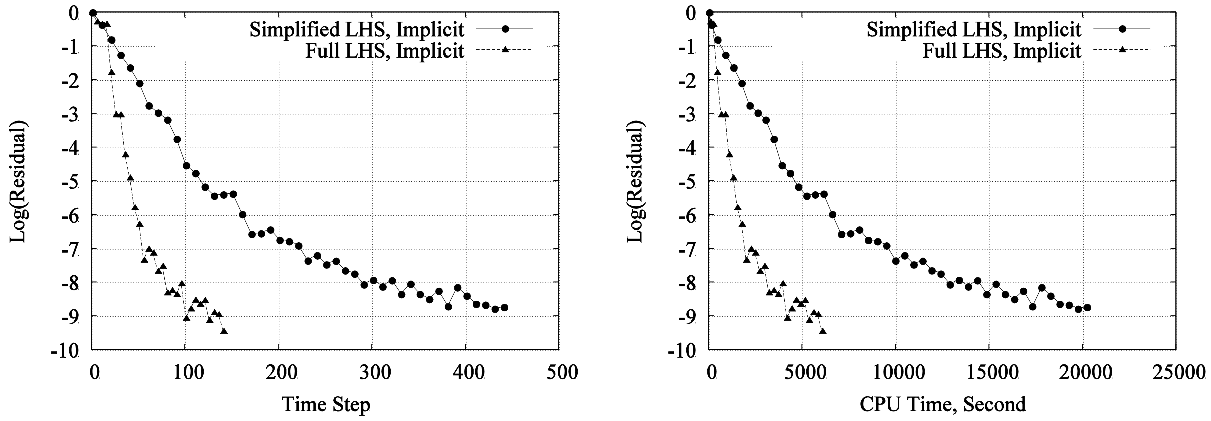


Figure 12. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) on the finest mesh for subsonic flow past a sphere M_∞=0.5.

5.3 Transonic Flow over the ONERA M6 Wing

A transonic flow over the ONERA M6 wing at a Mach number of M_∞=0.699 and an attack angle of α=3.06° is considered in this example. This case is chosen to demonstrate that the IRDG(P_1P_2) method is able to maintain the robustness of the underlying DG methods at the presence of weak discontinuities.

The DG method is not only linear stable but also has the ability to obtain a stable solution for weak discontinuities in spite of the over- and under-shots in the vicinity of shock waves.

The mesh used in this computation consists of 41,440 elements, 8,325 grid points, and 2,575 boundary points, as shown in Fig. 13. One can observe the coarseness of grids even in the vicinity of the leading edge.The computed pressure contours obtained by the RDG(P_1P_2) solution on the wing surface are shown in Fig. 14. Fig. 15 compares the pressure coefficient distributions at six span-wise locations on the wing surface between the numerical results and the experimental data. The pressure coefficients are computed at the two nodes of each triangle that intersect with the cut plane, and plotted by a straight line. This representation truly reflects the discontinuous nature of the DG solution. Since no limiters and Hermite WENO-reconstruction are used to eliminate the spurious oscillations for the underlying DG(P_1) method, the over- and under-shoots in the vicinity of the shock waves are clearly visible. Besides, the calculations show a good agreement with experiment data. This example clearly demonstrates that the IRDG(P_1P_2) is able to maintain the linear stability of the underlying DG method.

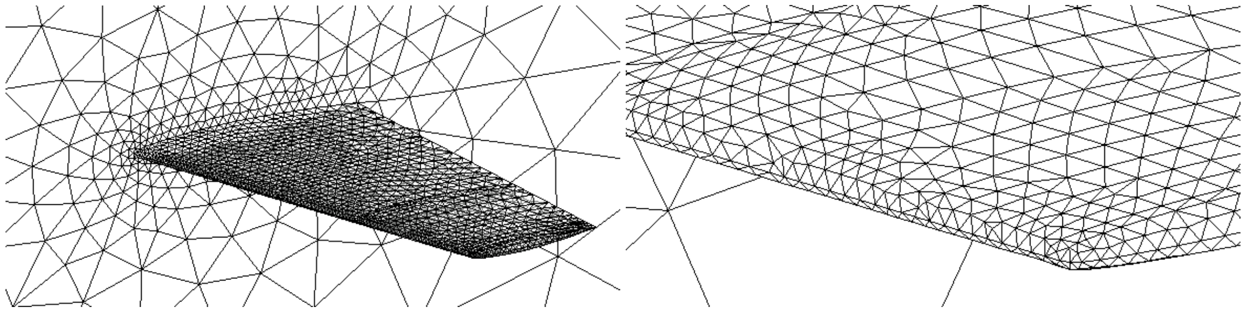


Figure 13. Left: unstructured mesh used for computing a transonic flow past an ONERA M6 wing (41,440 elements, 8,325 points, 2,575 boundary points). Right: Details around the leading edge.

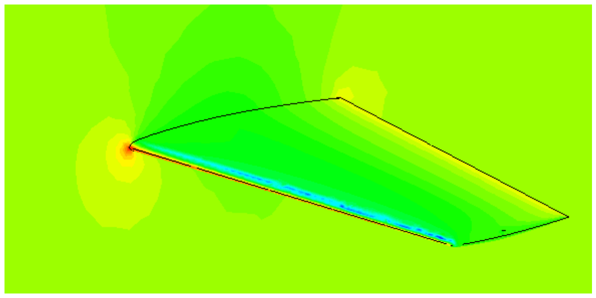
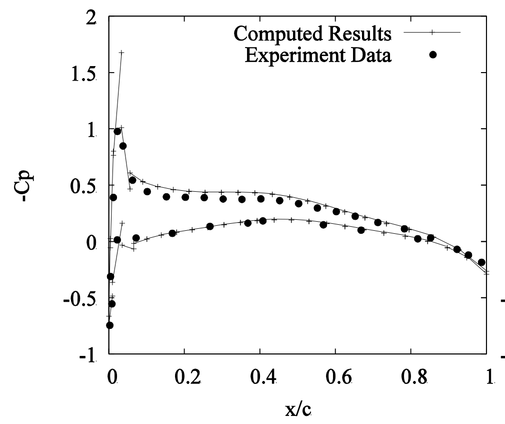
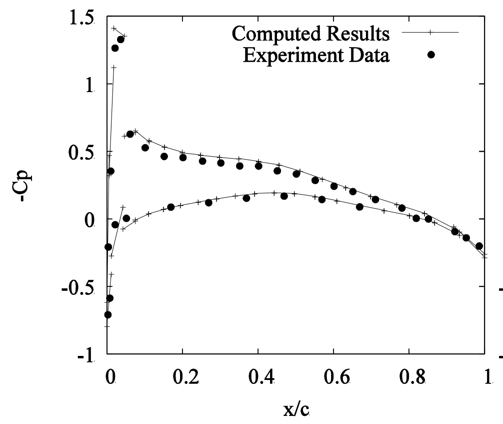


Figure 14. Computed pressure contours obtained by the RDG(P_1P_2) method for transonic flow past an ONERA M6 wing at M_∞=0.699, α=3.06°.

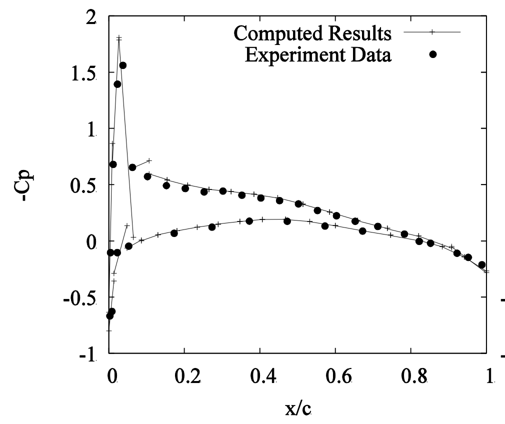
The plots of logarithmic density residual versus time steps and CPU time for RDG(P_1P_2) methods are shown in Fig. 16. The implicit solver with HLLC full Jacobians at a *CFL* number of 10^5, is more than 30 times faster than its explicit counterpart in terms of CPU time. It is found when the simplified Jacobians are used, stability will not be maintained if a *CFL* number of 10^5 is applied after the same initial steps. As a result, it needs 2.5 times more time steps and 3.5 times more CPU time for full convergence at a *CFL* number of 10^4 than the implicit solver with HLLC Jacobians, due to the fact that more inner iterations in the GMRES process are required for each time step.



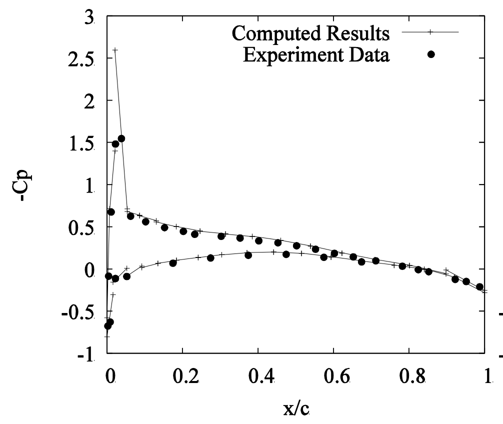
(a) η=0.20



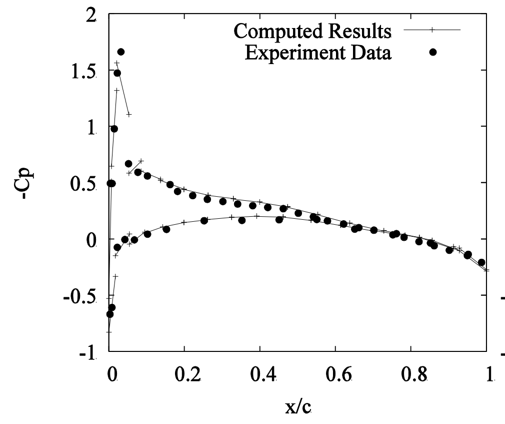
(b) η=0.44



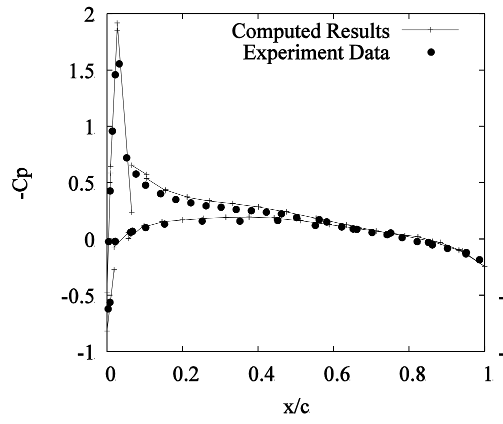
(c) η=0.65



(d) η=0.80



(e) η=0.90



(f) η=0.95

Figure 15. Pressure coefficient distributions for ONERA M6 wing at 6 span-wise locations, M_∞=0.699 α=3.06°.

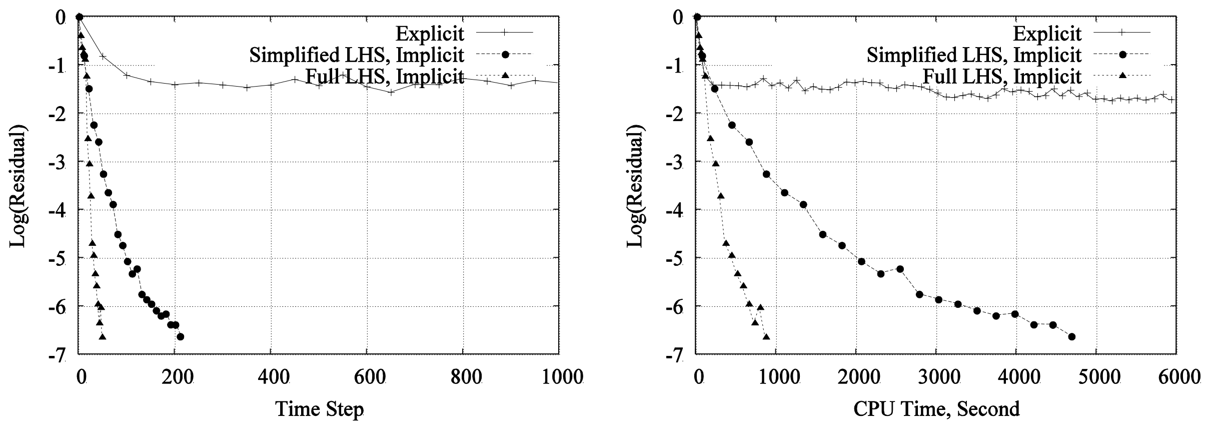
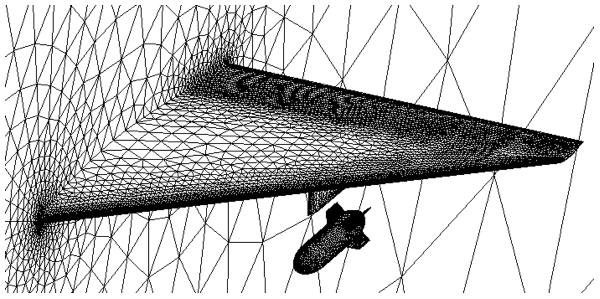
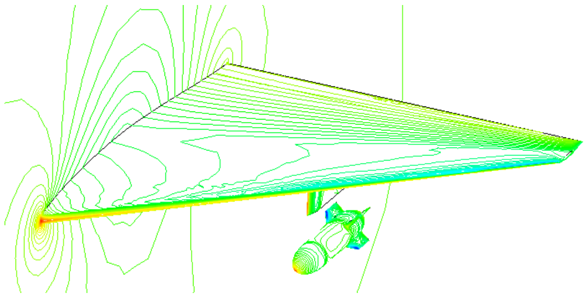


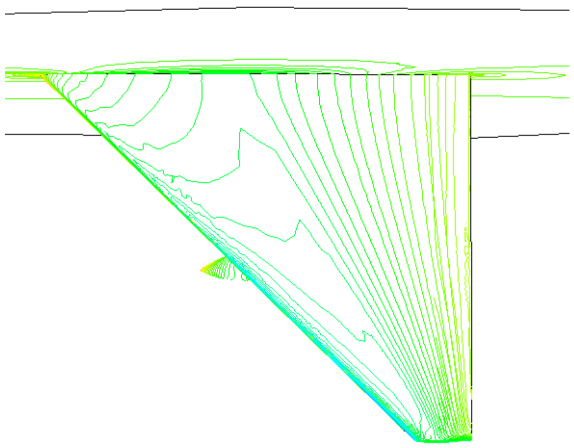
Figure 16. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) for transonic flow past ONERA M6 wing at M_∞=0.699, α=3.06°.

5.4 Subsonic Flow over the Wing/Pylon/Finned-Store Configuration

The fourth case is for the subsonic flow over the wing/pylon/finned-store configuration reported in Ref. [50] at M_∞=0.5 and α=3.06°. This test case is conducted to test the performance of the IRDG(P_1P_2) method for computing flows over complex geometric configurations. The configuration consists of a clipped delta wing with a 45° sweep comprised from a constant NACA 64010 symmetric airfoil section. The wing has a root chord of 15 inches, a semispan of 13 inches, and a taper ratio of 0.134. The pylon is located at the midspan station and has a cross-section characterized by a flat plate closed at the leading and trailing edges by a symmetrical ogive shape. The width of the pylon is 0.294 inches. The four fins on the store are defined by a constant NACA 0008 airfoil section with a leading-edge sweep of 45° and a truncated tip. The mesh used in the computation is shown in Fig.17(a). It contains 328,370 elements, 62,630 grid points, and 14,325 boundary points. Fig.17(b), 17(c) and 17(d) show the computed pressure contours from the front side view, on the upper and lower wing surface, respectively. Fig. 18 shows a comparison for logarithmic density residual versus time steps and CPU time between the explicit and implicit solvers for RDG(P_1P_2). Only the implicit solver with HLLC full Jacobians is tested in this case. The IRDG(P_1P_2) solver obtains a speedup of more than two orders of magnitude than the explicit solver. In fact, the explicit solver is already not a practical choice in this case. Furthermore, it is even impossible for explicit solvers to fully converge the flow at all for some more complex configurations. Therefore the developed IRDG(P_1P_2) method has again proved its superior performance and offers the feasibility in large-scale engineering applications.


(a)


(b)


(c)

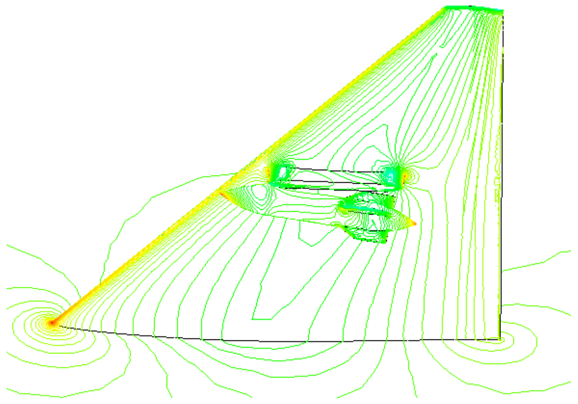
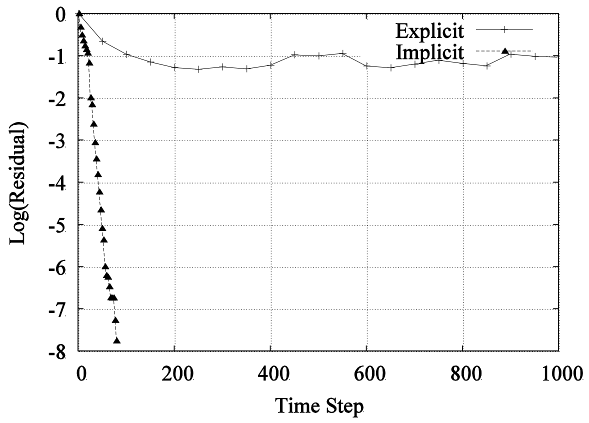

(d)

Figure 17. (a) Surface mesh for the wing/pylon/store (328,370 elements, 62,630 points, 14,325 boundary points). (b) Computed pressure contours from the front side view at M_∞=0.5, α=3.06°. (c) Computed pressure contours on upper surface at M_∞=0.5, α=3.06°. (d) Computed pressure contours on lower surface at M_∞=0.5, α=3.06°.



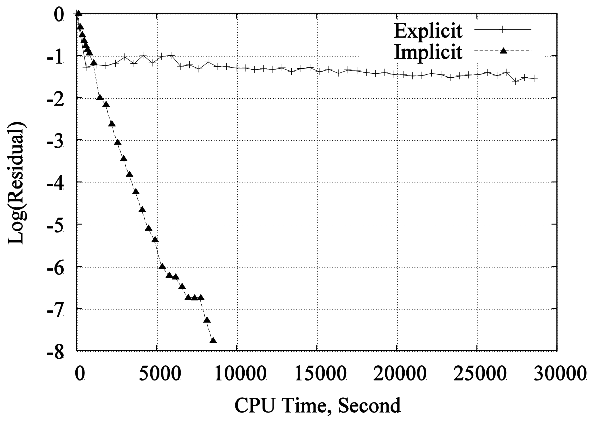


Figure 18. Logarithmic density residual versus time steps (left) and CPU time (right) for RDG(P_1P_2) for subsonic flow past wing/pylon/store at M_∞=0.5, α=3.06°.

5 Conclusions

An implicit Hermite WENO reconstruction-based discontinuous Galerkin method, IRDG(P_1P_2), has been presented to solve the compressible Euler equations on tetrahedral grids. An LU-SGS preconditioned matrix-free GMRES algorithm has been applied to solve an approximate system of linear equations arising from the Newton linearization. A variety of three-dimensional test cases have been conducted to demonstrate that this developed IRDG(P_1P_2) method is able to achieve a speedup of at least two orders of magnitude faster than its explicit counterpart, provided that a well approximated Jacobian matirx is necessarily implemented as the preconditioning matrix. The numerical experiments also indicate that this IRDG(P_1P_2) method can maintain linear stability in smooth flow and nonlinear stability at the presence of weak discontinuities, deliver the desired third order of accuracy: an order of accuracy higher than that of the underlying DG(P1) method, without significant increase in computing cost and memory requirements. The current development is focused on the extension of the IRDG method on tetrahedral grids for all speeds.

Acknowledgements

This manuscript has been authored by Battelle Energy Alliance, LLC under contract No. DE-AC07-05ID14517 (INL/CON-09-16255) with the U.S. Department of Energy. The United States Government retains and the published, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The authors would like to acknowledge the partial support for this work provided by DOE under Nuclear Engineering University Program.

Reference

[1] Reed, W.H. Reed and T.R. Hill.Triangular Mesh Methods for the Neutron Transport Equation. Los Alamos Scientific Laboratory Report, LA-UR-73-479, 1973.

[2] B. Cockburn, S. Hou, and C. W. Shu. TVD Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for conservation laws IV: the Multidimensional Case. Math. Comput., Vol. 55, pp. 545-581, 1990.

[3] B. Cockburn, and C. W. Shu. The Runge-Kutta Discontinuous Galerkin Method for conservation laws V: Multidimensional System. J. Comput. Phys., Vol. 141, pp. 199-224, 1998.

[4] B. Cockburn, G. Karniadakis, and C. W. Shu. *The Development of Discontinuous Galerkin Method, in Discontinuous Galerkin Methods, Theory, Computation, and Applications*, edited by B. Cockburn, G.E. Karniadakis, and C. W. Shu, Lecture Notes in Computational Science and Engineering, Springer-Verlag, New York, 2000, Vol. 11 pp. 5-50, 2000.

[5] F. Bassi and S. Rebay, High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations. J. Comput. Phys., Vol. 138, pp. 251-285, 1997.

[6] H. L. Atkins and C. W. Shu. Quadrature Free Implementation of Discontinuous Galerkin Method for Hyperbolic Equations. AIAA J., Vol. 36, No. 5, 1998.

[7] F. Bassi and S. Rebay. GMRES discontinuous Galerkin solution of the Compressible Navier-Stokes Equations, *Discontinuous Galerkin Methods, Theory, Computation, and Applications*, edited by B. Cockburn, G.E. Karniadakis, and C. W. Shu, Lecture Notes in Computational Science and Engineering, Springer-Verlag, New York, 2000, Vol. 11 pp. 197-208, 2000.

[8] T. C. Warburton, and G. E. Karniadakis. A Discontinuous Galerkin Method for the Viscous MHD Equations. J. Comput. Phys., Vol. 152, pp. 608-641, 1999.

[9] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Texts in Applied Mathematics, Vol. 56, 2008.

[10] P. Rasetarinera and M. Y. Hussaini. An Efficient Implicit Discontinuous Spectral Galerkin Method. J. Comput. Phys., Vol. 172, pp. 718-738, 2001.

[11] B. T. Helenbrook, D. Mavriplis, and H. L. Atkins. Analysis of p-Multigrid for Continuous and Discontinuous Finite Element Discretizations. AIAA Paper, 2003-3989, 2003.

[12] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal. p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations. J. Comput. Phys., Vol. 207, No. 1, pp. 92-113, 2005.

[13] H. Luo, J. D. Baum, and R. Löhner. A Discontinuous Galerkin Method Using Taylor Basis for Compressible Flows on Arbitrary Grids. J. Comput. Phys., Vol. 227, No 20, pp. 8875-8893, October 2008.

[14] H. Luo, J.D. Baum, and R. Löhner. On the Computation of Steady-State Compressible Flows Using a Discontinuous Galerkin Method. Int. J. Numer. Meth. Eng, Vol. 73, No. 5, pp. 597-623, 2008.

[15] H. Luo, J. D. Baum, and R. Löhner. A Hermite WENO-based Limiter for Discontinuous Galerkin Method on Unstructured Grids. J. Comput. Phys., Vol. 225, No. 1, pp. 686-713, 2007.

[16] H. Luo, J.D. Baum, and R. Löhner. A p-Multigrid Discontinuous Galerkin Method for the Euler Equations on Unstructured Grids. J. Comput. Phys., Vol. 211, No. 2, pp. 767-783, 2006.

[17] H. Luo, J.D. Baum, and R. Löhner. A Fast, p-Multigrid Discontinuous Galerkin Method for Compressible Flows at All Speed. AIAA J., Vol. 46, No. 3, pp.635-652, 2008.

[18] M. Dumbser, D.S. Balsara, E.F. Toro, C.D. Munz. A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes. J. Comput. Phys., 227:8209-8253, 2008.

[19] M. Dumbser, O. Zanotti. Very high order PNPM schemes on unstructured meshes for the resistive relativistic MHD equations. J. Comput. Phys., 228:6991-7006, 2009.

[20] M. Dumbser. Arbitrary High Order PNPM Schemes on Unstructured Meshes for the Compressible Navier-Stokes Equations. J. Computers & Fluids, 39: 60-76. 2010.

[21] F. Bassi and S. Rebay. A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier-Stokes Equations. J. Comput. Phys., Vol. 131, pp. 267-279, 1997.

[22] F. Bassi and S. Rebay. Discontinuous Galerkin Solution of the Reynolds-Averaged Navier-Stokes and k-ω Turbulence Model Equations. J. Comput. Phys., Vol. 34, pp. 507-540, 2005.

[23] B. Cockburn and C.W. Shu. The Local Discontinuous Galerkin Method for Time-dependent Convection-Diffusion System. SIAM, Journal of Numerical Analysis, Vo. 16, 2001.

[24] C. E. Baumann and J. T. Oden. A Discontinuous hp Finite Element Method for the Euler and Navier-Stokes Equations. Int. J. Numer. Meth. Fl., Vol. 31, 1999.

[25] J. Peraire and P. O. Persson. The Compact Discontinuous Galerkin Method for Elliptic Problems. SIAM Journal on Scientific Computing, 30: 1806-1824, 2008.

[26] H. Luo, L. Luo, R. Nourgaliev, and V. Mousseau. A Reconstructed Discontinuous Galerkin Method for the Compressible Euler Equations on Arbitrary Grids. AIAA-2009-3788, 2009.

[27] H. Luo, L. Luo, R. Nourgaliev, Mousseau, A, and N. Dinh. A Reconstructed Discontinuous Galerkin Method for the Compressible Navier-STokes Equations on Arbitrary Grids. J. Comput. Phys., Vol. 229, pp. 6961-6978, 2010.

[28] H. Luo, Luo, L., Ali, A., Norgaliev, R., and i, C. A Parallel, Reconstructed Discontinuous Galerkin Method for the Compressible Flows on Arbitrary Grids. Commun. Comput. Phys., Vo. 9, No.2, pp. 363-389, 2011.

[29] D. F. Haider, J.P. Croisille, and B. Courbet. Stability Analysis of the Cell Centered Finite-Volume MUSCL Method on Unstructured Grids. Numirische Mathematik, Vol. 113, No. 4 pp. 555-600, 2009.

[30] D. Balsara, C. Altmann, C.D. Munz and M. Dumbser. A sub-cell based indicator for troubled zones in RKDG schemes and a novel class of hybrid RKDG + HWENO schemes. J. Comput. Phys., Vol. **226**, pp. 586–620, 2007.

[31] B. Stoufflet. Implicit finite element methods for the Euler equations, in Numerical Methods for the Euler Equations of Fluid Dynamics, edited by F. Angrand. SIAM, Philadelphia, 1985.

[32] J. T. Batina. Implicit flux-split Euler schemes for unsteady aerodynamic analysis involving unstructured dynamic meshes. AIAA J., 29(11), 1991.

[33] V. Venkatakrishnan and D. J. Mavriplis. Implicit solvers for unstructured meshes. J. Comput. Phys., 105(83), 1993.

[34] D. D. Knight. A fully implicit Navier-Stokes Algorithm using an unstructured grid and flux difference splitting. AIAA Paper, 93-0875, 1993.

[35] D. L. Whitaker. Three-dimensional unstructured grid Euler computations using a fully-implicit, upwind method. AIAA Paper, 93-3337, 1993.

[36] H. Luo, J. D. Baum, and R. Löhner. A Fast, Matrix-free Implicit Method for compressible flows on Unstructured Grids. J. Comput. Phys., Vol. 146, No. 2, pp. 664-690, 1998.

[37] T.J. Barth and S. W. Linton, "An unstructured mesh Newton solver for compressible fluid flow and its parallel implementation. AIAA Paper, 95-0221, 1995

[38] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comp. 7(3), 89, 1998.

[39] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems. SIAM Journal on Numerical Analysis. Vol. 39, No. 5., pp. 1749-1779, 2002.

[40] G. Gassner, F. Lorcher, and C. D. Munz. A Contribution to the Construction of Diffusion Fluxes for Finite Volume and Discontinuous Galerkin Schemes. J. Comput. Phys., Vol. 224, No. 2, pp. 1049-1063, 2007.

[41] H. Liu and K. Xu. A Runge-Kutta Discontinuous Galerkin Method for Viscous Flow Equations. J. Comput. Phys., Vol. 224, No. 2, pp. 1223-1242, 2007.

[42] H. Luo and Y. Xia. A Class of Reconstructed Discontinuous Galerkin Methods in Computational Fluid Dynamics. International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Brazil, May, 2011.

[43] H. Luo, L. Luo, R. Nourgaliev, and V. Mousseau. A Reconstructed Discontinuous Galerkin Method for the Compressible Navier-Stokes Equations on Arbitrary Grids, AIAA-2010-0364, 2010.

[44] H. Luo, L. Luo, R. Norgaliev, V.A. Mousseau, and N. Dinh. A Reconstructed Discontinuous Galerkin Method for the Compressible Navier-Stokes Equations on Arbitrary Grids. J. Comput. Phys., Vol. 229, pp. 6961-6978, 2010.

[45] H. Luo, L. Luo, A. Ali, R. Norgaliev, and C. Cai. A Parallel, Reconstructed Discontinuous Galerkin Method for the Compressible Flows on Arbitrary Grids. Commun. Comput. Phys., Vo. 9, No. 2, pp. 363-389, 2011.

[46] Zhang L.P, Liu W., He L.X., Deng, X.G., Zhang H.X. A Class of Hybrid DG/FV Methods for Conservation Laws II: Two dimensional Cases, doi:10.1016/j.jcp.2011.03.032. J. Comput. Phys., 2011.

[47] P. Batten, M. A. Leschziner, and U.C. Goldberg. Average-State Jacobians and Implicit Methods for Compressible Viscous and Turbulent Flows. J. Comput. Phys., Vol. 137, pp. 38-78, 1997.

[48] P. L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. J. Comput. Phys., Vol. 43, Issue 2, pp 357-372, 1981.

[49] M. Pernice, HF. Walker. NITSOL: a Newton iterative solver for nonlinear systems. SIAM J Sci Stat Comput, 19:302-18, 1998.

[50] E. R. Illeim. CFD wing/pylon/finned store mutual interference wind tunnel experiment. AEDC-TSR-91-P4, Arnold Engineering Development Center, Arnold AFB, TN, Jan. 1991.