AIAA JOURNAL
Vol. 46, No. 3, March 2008

Fast *p*-Multigrid Discontinuous Galerkin Method for Compressible Flows at All Speeds

Hong Luo*
North Carolina State University, Raleigh, North Carolina 27695
Joseph D. Baum†
Science Applications International Corporation, McLean, Virginia 22102
and
Rainald Löhner‡
George Mason University, Fairfax, Virginia 22030

DOI: 10.2514/1.28314

A *p*-multigrid (where *p* is the polynomial degree) discontinuous Galerkin method is presented for the solution of the compressible Euler equations on unstructured grids. The method operates on a sequence of solution approximations of different polynomial orders. A distinct feature of this *p*-multigrid method is the application of an explicit smoother on the higher-level approximations ($p > 0$) and an implicit smoother on the lowest-level approximation ($p = 0$), resulting in a fast (and low) storage method that can be efficiently used to accelerate the convergence to a steady-state solution. Furthermore, this *p*-multigrid method can be naturally applied to compute the flows with discontinuities, in which a monotonic limiting procedure is usually required for discontinuous Galerkin methods. An accurate representation of the boundary normals based on the definition of the geometries is used for imposing slip boundary conditions for curved geometries [Krivodonova, L., and Berger, M., "High-Order Implementation of Solid Wall Boundary Conditions in Curved Geometries," *Journal of Computational Physics*, Vol. 211, No. 2, 2006, pp. 492–512; and Luo, H., Baum, J. D., and Löhner, R., "On the Computation of Steady-State Compressible Flows Using a Discontinuous Galerkin Method," *International Journal for Numerical Methods in Engineering*, Vol. 73, No. 5, 2008, pp. 597–623]. A variety of compressible flow problems for a wide range of flow conditions from low Mach number to supersonic in both two-dimensional and three-dimensional configurations are computed to demonstrate the performance of this *p*-multigrid method. The numerical results obtained strongly indicate the order-independent property of this *p*-multigrid method and demonstrate that this method is orders-of-magnitude faster than its explicit counterpart. The performance comparison with a finite volume method shows that using this *p*-multigrid method, the discontinuous Galerkin method, provides a viable, attractive, competitive, and probably even superior alternative to the finite volume method for computing compressible flows at all speeds.

Nomenclature

B	=	basis function
C_p	=	pressure coefficient
D	=	diagonal matrices
d	=	number of spatial dimension
e	=	specific total energy
$\mathbf{F}$	=	inviscid flux vector
$\mathbf{H}$	=	numerical flux function vector
$\mathbf{I}$	=	state restriction operator
$\tilde{\mathbf{I}}$	=	residual restriction operator
$\mathbf{J}$	=	state prolongation operator
L	=	strict lower matrices
M	=	mass matrix
p	=	pressure
$\mathbf{R}$	=	residual value vector
S	=	entropy
t	=	time
U	=	strict upper matrices
$\mathbf{U}$	=	conservative variable vector
u_i	=	velocity
x_i	=	spatial coordinates
γ	=	ratio of the specific heats
ϵ	=	entropy production
ρ	=	density

Subscripts

h	=	finite element approximations
P	=	polynomials of degree p
∞	=	freestream quantity

Superscripts

g	=	ghost state
n	=	solution at the time level n
P	=	polynomials of degree p

Presented as Paper 0110 at the 44th AIAA Aerospace Sciences Meeting and Exhibits, Reno, NV, 9–12 January 2006; received 12 October 2006; revision received 24 September 2007; accepted for publication 6 November 2007. Copyright © 2007 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 0001-1452/08 $10.00 in correspondence with the CCC.

*Associate Professor, Department of Mechanical and Aerospace Engineering. Senior Member AIAA.
†Director, Center for Applied Computational Sciences. Associate Fellow AIAA.
‡Professor, Department of Computational and Data Sciences. Member AIAA.

I. Introduction

THE discontinuous Galerkin methods [1–13] (DGM) have recently become popular for the solution of systems of conservation laws. Nowadays, they are widely used in computational fluid dynamics, computational acoustics, and computational electromagnetics. The discontinuous Galerkin methods combine two advantageous features commonly associated with 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 accounted for, however, 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. The discontinuous Galerkin methods have many features:

1) The methods are well suited for complex geometries because they can be applied on unstructured grids. In addition, the methods can also handle nonconforming elements, in which the grids are allowed to have hanging nodes.

2) The methods are highly parallelizable, because they are compact and each element is independent. Because the elements are discontinuous and the interelement communications are minimal, domain decomposition can be efficiently employed. The compactness also allows for structured and simplified coding for the methods.

3) They can easily handle adaptive strategies, because refining or coarsening a grid can be achieved without considering the continuity restriction commonly associated with the conforming elements. The methods allow easy implementation of hp refinement: for example, the order of accuracy, or shape, can vary from element to element.

4) They have several useful mathematical properties with respect to conservation, stability, and convergence.

However, these methods have their own weaknesses. Compared with finite element methods and finite volume methods, the discontinuous Galerkin methods require the solutions of systems of equations with more unknowns for the same grids. Consequently, these methods are recognized as expensive in terms of both computational cost and storage requirement.

Most efforts in the development of the discontinuous Galerkin methods are primarily focused on the spatial discretization. The temporal discretization methods have lagged far behind. Usually, explicit temporal discretizations such as multistage total variation diminishing (TVD) Runge–Kutta schemes [1,6–11] are used to advance the solution in time. In general, explicit schemes and their boundary conditions are easy to implement, vectorize, and parallelize, and they require only limited memory storage. However, for large-scale simulations and, especially, for high-order solutions, the rate of convergence slows down dramatically, resulting in inefficient solution techniques to steady-state solutions. To speed up convergence, a multigrid strategy or an implicit temporal discretization 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. Recently, significant efforts have been made to develop efficient implicit solution methods for discontinuous Galerkin methods [12,13]. Unfortunately, the drawback is that they require a considerable amount of memory to store the Jacobian matrix, which may be prohibitive for large-scale problems and high-order solutions. Even in the implementation of so-called matrix-free implicit methods [13], in which only a block diagonal matrix is required to store, the memory requirements can still be extremely demanding. The block diagonal matrix requires a storage of

$$(\text{neqns} \times \text{ndegr}) \times (\text{neqns} \times \text{ndegr}) \times \text{nelem}$$

where neqns is the number of components in the solution vector (four for 2D and five for 3D Euler equations), ndegr is the degrees of freedom for the polynomial (3 for P1, 6 for P2, and 10 for P3 for the triangle element in 2D; 4 for P1, 10 for P2, and 20 for P3 for the tetrahedral element in 3D), and nelem is the number of elements for the grid. For example, for a fourth-order (cubic polynomial finite element approximation P3) discontinuous Galerkin method in 3D, the storage of this block diagonal matrix alone requires 10,000 words per element!

Multigrid methods offer an alternative and efficient approach to steady-state solutions. Similar to classical geometric multigrid methods in which the discrete equations are recursively solved on successively coarse grids of the domain, p-multigrid method is an iterative scheme in which systems of equations arising from compact, high-order, finite element discretization, such as spectral-hp or discontinuous Galerkin formulation, are solved by recursively iterating on solution approximations of different polynomial orders. For example, to solve equations derived using a polynomial approximation order of 2, the solution can be iterated on at an approximation order of $p = 2$, 1, and 0. The p component of this algorithm was introduced by Rönquist and Patera [14] and analyzed by Maday and Munoz [15] for a one-dimensional Galerkin spectral element discretization of the Laplace equation. Bassi and Rebay [16] presented their work on the p-multigrid method for the Euler equations. Helenfrook et al. [17] examined the performance of p-multigrid for the Laplace equation and the convection equation in two dimensions. Fidkowski et al. [18] used a p-multigrid method for solving the compressible Navier–Stokes equations.

The present authors developed a fast low-storage p-multigrid method for discontinuous Galerkin methods to solve the compressible Euler equations on unstructured grids [19]. Unlike the traditional p-multigrid methods in which the same time-integration scheme is used on all approximation levels, this p-multigrid method uses a multistage Runge–Kutta scheme as the iterative smoother on the higher-level approximations and a matrix-free implicit SGS method as the iterative smoother on the lowest-level approximation. As a result, this p-multigrid method has three remarkable features:

1) The method has low memory requirements. The implicit smoothing is only used on the lowest-level $P0$, for which the storage requirement is not as demanding as on the higher level.

2) The method has a natural extension to flows with discontinuities such as shock waves and contact discontinuities. Slope limiters required to eliminate spurious oscillations of high-order approximations in the vicinity of discontinuities can be easily implemented as a postprocessing filter (smoothing) in an explicit method, but not in an implicit method.

3) The method has no startup issue. It is well known that it is necessary to adaptively update the time step or, more precisely, to gradually increase the Courant–Friedrichs–Lewy (CFL) number to maintain stability and maximize the efficiency of the implicit methods. This procedure can be both mesh- and problem-dependent and may become difficult for higher-order approximations. Recent improvements and advances in the development of the p-multigrid method are presented in this work. Unlike our previous work [19], in which only the two-level V-cycle p-multigrid method was used to drive the iterations for both second- and third-order discontinuous Galerskin (DG) solutions, the successive $p - 1$ discretization is chosen as the coarse approximation, which was found to be more efficient and robust than the two-level V-cycle for the third-order DG method. An accurate representation of boundary normals based on the geometric definition is used for imposing slip boundary conditions for curved geometries. This avoids the use of higher-order geometrical approximation, which not only increases computational expense but also complicates implementation of boundary conditions. Furthermore, the p-multigrid is extended to compute flows with discontinuities using slope limiters that can be readily applied and implemented, due to the explicit smoother used for the high-order approximations used in this p-multigrid. A variety of compressible flow problems for a wide range of flow conditions from low Mach number to supersonic in both 2D and 3D are computed to demonstrate the performance of this p-multigrid. The numerical results obtained strongly indicate the order-independent property of this p-multigrid method and demonstrate that this method is orders-of-magnitude faster than its explicit counterpart. The comparison between this p-multigrid DG method and a finite volume method shows strong evidence that with the progress made in the development of the p-multigrid method, the DG method provides a viable, attractive, competitive, and probably even superior alternative to the finite volume method for computing compressible flows at all speeds.

II. Governing Equations

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

$$\frac{\partial \mathbf{U}(\mathbf{x}, t)}{\partial t} + \frac{\partial \mathbf{F}_j(\mathbf{U}(\mathbf{x}, t))}{\partial x_j} = 0 \tag{1}$$

where the conservative state vector $\mathbf{U}$ and the inviscid flux vectors $\mathbf{F}$ are defined by

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + p\delta_{ij} \\ u_j(\rho e + p) \end{pmatrix} \tag{2}$$

where the summation convention was used and 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. This set of equations is completed by the addition of the equation of state

$$p = (\gamma - 1)\rho(e - \tfrac{1}{2}u_j u_j) \tag{3}$$

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

III. Discontinuous Galerkin Method

A. Discontinuous Galerkin Spatial Discretization

To formulate the discontinuous Galerkin method, we first introduce the following weak formulation of Eq. (1), which is obtained by multiplying Eq. (1) by a test function $\mathbf{W}$, integrating over the domain Ω, and performing an integration by parts:

$$\int_\Omega \frac{\partial \mathbf{U}}{\partial t}\mathbf{W}\,\mathrm{d}\Omega + \int_\Gamma \mathbf{F}_j \mathbf{n}_j \mathbf{W}\,\mathrm{d}\Gamma - \int_\Omega \mathbf{F}_j \frac{\partial \mathbf{W}}{\partial x_j}\,\mathrm{d}\Omega = 0 \quad \forall\ \mathbf{W} \tag{4}$$

where $\Gamma(=\partial\Omega)$ denotes the boundary of Ω, and $\mathbf{n}_j$ the unit outward normal vector to the boundary.

Assuming that Ω_h is a classical triangulation of Ω, where the domain Ω is subdivided into a collection of nonoverlapping elements Ω_e, triangles in 2D, and tetrahedra in 3D, the following semidiscrete form of Eq. (4) is obtained by applying Eq. (4) on each element Ω_e:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega_e} \mathbf{U}_h \mathbf{W}_h\,\mathrm{d}\Omega + \int_{\Gamma_e} \mathbf{F}_j(\mathbf{U}_h)\mathbf{n}_j \mathbf{W}_h\,\mathrm{d}\Gamma - \int_{\Omega_e} \mathbf{F}_j(\mathbf{U}_h)\frac{\partial \mathbf{W}_h}{\partial x_j}\,\mathrm{d}\Omega = 0 \quad \forall\ \mathbf{W}_h \tag{5}$$

where $\Gamma_e(=\partial\Omega_e)$ denotes the boundary of Ω_e, and $\mathbf{U}_h$ and $\mathbf{W}_h$ represent the finite element approximations to the analytical solution $\mathbf{U}$ and the test function $\mathbf{W}$, respectively. Assume that the approximate solution and test function to be piecewise polynomials in each element, then $\mathbf{U}_h$ and $\mathbf{W}_h$ can be expressed as

$$\mathbf{U}_h(\mathbf{x}, t) = \sum_{m=1}^{N} \mathbf{U}_m(t) B_m^p(\mathbf{x}), \qquad \mathbf{W}_h(\mathbf{x}) = \sum_{m=1}^{N} \mathbf{W}_m B_m^p(\mathbf{x}) \tag{6}$$

where $B_m^p(\mathbf{x})$, $1 \le m \le N$ is the basis function of the polynomials of degree p. 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, as follows:

$$N = \frac{(p+1)(p+2)\cdots(p+d)}{d!} \quad \text{for } d = 1, 2, 3 \tag{7}$$

Equation (5) must be satisfied for any test function $\mathbf{W}_h$. Because B_n^p is the basis for $\mathbf{W}_h$, Eq. (5) is therefore equivalent to the following system of N equations:

$$\frac{\mathrm{d}\mathbf{U}_m}{\mathrm{d}t}\int_{\Omega_e} B_m^p B_n^p\,\mathrm{d}\Omega + \int_{\Gamma_e} \mathbf{F}_j(\mathbf{U}_h)\mathbf{n}_j B_n^p\,\mathrm{d}\Gamma - \int_{\Omega_e} \mathbf{F}_j(\mathbf{U}_h)\frac{\partial B_n^p}{\partial x_j}\,\mathrm{d}\Omega = 0 \qquad 1 \le n \le N \tag{8}$$

where $\mathbf{U}_h$ is replaced with Eq. (6). Because the numerical solution $\mathbf{U}_h$ is discontinuous between element interfaces, the interface fluxes are not uniquely defined. The flux function $\mathbf{F}_j(\mathbf{U}_h)\mathbf{n}_j$ appearing in the second term of Eq. (8) is replaced by a numerical Riemann flux function $\mathbf{H}(\mathbf{U}_h^L, \mathbf{U}_h^R, \mathbf{n})$, where $\mathbf{U}_h^L$ and $\mathbf{U}_h^R$ are the conservative state vector at the left and right sides of the element boundary. To guarantee consistency and conservation, $\mathbf{H}(\mathbf{U}^L, \mathbf{U}^R, \mathbf{n})$ is required to satisfy

$$\mathbf{H}(\mathbf{U}, \mathbf{U}, \mathbf{n}) = \mathbf{F}_j(\mathbf{U})\mathbf{n}_j, \qquad \mathbf{H}(\mathbf{U}, \mathbf{V}, \mathbf{n}) = -\mathbf{H}(\mathbf{V}, \mathbf{U}, \mathbf{n}) \tag{9}$$

This scheme is called the discontinuous Galerkin method of degree p, or, in short notation, the DG(p) method. Note that discontinuous Galerkin 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(P0) method (i.e., to the discontinuous Galerkin method using a piecewise constant polynomial). Consequently, the DG(p) methods with $p > 0$ can be regarded as a "natural" generalization of finite volume methods to higher-order methods. By simply increasing the degree p of the polynomials, DG methods of corresponding higher orders are obtained.

In the present work, the Riemann flux function is approximated using the Harten, Lax, and van Leer with contact restoration (HLLC) approximate Riemann solver [20], which has been successfully used to compute compressible viscous and turbulent flows on both structured grids [21] and unstructured grids [22]. This HLLC scheme is found to have the following properties: 1) exact preservation of isolated contact and shear waves, 2) positivity-preserving of scalar quantity, and 3) enforcement of entropy condition. In addition, the implementation of the HLLC Riemann solver is easier and the computational cost is lower than with other available Riemann solvers.

The standard Lagrange finite element shape functions are used to represent numerical polynomial solutions in each element. The domain and boundary integrals in Eq. (8) are calculated using $2p$- and $(2p + 1)$-order-accurate Gauss quadrature formulas, respectively. The number of quadrature points necessary for a given order depends on the quadrature rule used. In the case of linear, quadratic, and cubic shape functions, the domain integrals are evaluated using 3, 6, and 12 points, respectively, and the boundary integrals are evaluated using 2, 3, and 4 points, respectively, for 2D. In 3D, integration over the elements for P1 and P2 approximation is performed using 4 and 11 quadrature points, respectively, and integration over the element boundaries for P0, P1, and P2 is performed using 1, 4, and 7 quadrature points, respectively.

By assembling all the elemental contributions, a system of ordinary differential equations governing the evolution in time of the discrete solution can be written as

$$M\frac{\mathrm{d}\mathbf{U}}{\mathrm{d}t} = \mathbf{R}(\mathbf{U}) \tag{10}$$

where M denotes the mass matrix, $\mathbf{U}$ is the global vector of the degrees of freedom, and $\mathbf{R}(\mathbf{U})$ is the residual vector. Because the shape functions $B^p|_{\Omega_e}$ are nonzero within element Ω_e only, the mass matrix M has a block diagonal structure that couples the N degrees of freedom of each component of the unknown vector only within Ω_e. As a result, the inverse of the mass matrix M can be easily computed by hand, considering one element at a time in advance.

B. Time Integration

The semidiscrete system can be integrated in time using explicit methods. For example, the following explicit, three-stage, third-order, TVD Runge–Kutta scheme [1,7]

$$\mathbf{U}^{(1)} = \mathbf{U}^n + \Delta t M^{-1}\mathbf{R}(\mathbf{U}^n) \tag{11}$$

$$\mathbf{U}^{(2)} = \tfrac{3}{4}\mathbf{U}^n + \tfrac{1}{4}[\mathbf{U}^{(1)} + \Delta t M^{-1}\mathbf{R}(\mathbf{U}^{(1)})] \tag{12}$$

$$\mathbf{U}^{n+1} = \tfrac{1}{3}\mathbf{U}^n + \tfrac{2}{3}[\mathbf{U}^{(2)} + \Delta t M^{-1}\mathbf{R}(\mathbf{U}^{(2)})] \tag{13}$$

is widely used to advance the solution in time. This method is linearly stable for a CFL number less than or equal to $1/(2p + 1)$. The inefficiency of the explicit method due to this rather restrictive CFL condition motivates us to develop the p-multigrid method to

accelerate the convergence of the Euler equations to a steady-state solution. Convergence is accelerated using local time stepping for steady-state solutions. The local time step Δt on each element Ω_e is determined by the following relation:

$$\Delta t = \frac{\Omega_e \mathrm{CFL}}{(2p+1)\int_{\Gamma_e} \max(|\, u_i^l n_i \,| + c^l, |\, u_i^r n_i \,| + c^r)\, \mathrm{d}\Gamma} \tag{14}$$

where superscripts l and r represent the left and right sides of the element boundary, respectively, and c is the speed of sound.

C. Curved-Wall Boundary Conditions

It is recognized that DG methods are far more sensitive to errors arising at curved boundaries than those obtained with finite volume methods of the same order of accuracy. Bassi and Rebay [8] showed that an accurate boundary representation is necessary to maintain the formal order of the DG methods and to avoid spurious production of entropy on the boundary. A common solution to this problem is to use higher-order geometrical approximation. Unfortunately, curved-element meshes are associated with extra computational expenses. Curved elements need to be mapped onto the computational straight-sided element by nonlinear mapping. To account for the nonconstant Jacobian and the variation of the metric, a higher number of quadrature points are required to compute volume and boundary integrals. In a novel approach suggested by Krivodonova and Berger [23], the elements adjacent to the solid-wall boundaries remain straight-sided elements. However, an accurate representation of the boundary normals is used to define a ghost state at the quadrature points. Let the interior density, pressure, and velocity vector be ρ, p, and u_i. Then the flow variables at the ghost state g are computed with

$$\rho^g = \rho \tag{15}$$

$$p^g = p \tag{16}$$

$$u_i^g = u_i - 2(u_i n_i) n_i \tag{17}$$

where n_i is the unit normal to the physical boundary, not the straight-sided element face unit normal. Once the ghost state values are determined at integration points, numerical fluxes in Eq. (5) are computed as usual by solving the Riemann problem. It has been numerically shown that this approach does not compromise the formal order of the DG method [23]. Using only straight-sided elements instead of boundary-fitted elements represents a huge simplification of the code implementation and tremendous saving in both storage requirements and computing costs. In our implementation, the normals at the boundary integration points are computed using the local true surface normal based on the analytically defined boundary geometries, which are handily available in our geometry definition file.

D. Monotonicity Limiter

High-order numerical schemes produce spurious oscillations in the vicinity of discontinuities, which can lead to numerical instabilities and unbounded solutions. Even though the fluxes at the interelement boundaries are computed using an appropriate Riemann flux function, spurious oscillations may still be generated by the discontinuous Galerkin finite element methods near discontinuities when polynomials of higher degree are used ($p > 0$). First-order schemes DG(P0) are the only approaches that maintain a monotonic solution at discontinuities. Unfortunately, numerical solutions obtained by these schemes exhibit too much dissipation. Indeed, most research efforts for DG methods are focused on developing and designing appropriate stabilization methods that can produce a solution with neither excessive diffusion nor spurious oscillations and that do not adversely affect the formal order of accuracy of the DG methods. Generally speaking, there are two strategies for resolving this problem: a discontinuity-capturing method and an appropriate flux/slope-limiting method. The former explicitly adds some form of nonlinear dissipation terms to the discontinuous Galerkin discretization. The main disadvantage of this approach is that it requires some user-defined parameters that can be both mesh- and problem-dependent. The latter is designed to restrict or suppress oscillations near discontinuities through a nonlinear procedure based on comparing elemental solution features such as slopes or curvatures with those of neighboring elements.

Classical techniques of flux limiting are not directly applicable for high-order DG methods because of the presence of volume terms in the formulation. Therefore, a slope limiter is not integrated in the computation of the residual, but effectively acts as a postprocessing filter. Note that such a limiting procedure can be easily integrated in an explicit method, but not into an implicit method. Many slope limiters used in the finite volume methods can be applied or modified to meet the needs of the DGM. Following Barth and Jesperson [24], slopes are limited so that the solution at the quadrature points $\mathbf{x}_j$ $(j = 1, 2, \ldots, K_{\Gamma_i})$ in an element Ω_i, $\mathbf{U}_i(\mathbf{x}_j)$ is in the range spanned by the neighboring solution averages:

$$\bar{\mathbf{U}}_i^{\min} \le \mathbf{U}_i(\mathbf{x}_j) \le \bar{\mathbf{U}}_i^{\max} \tag{18}$$

where $\bar{\mathbf{U}}_i^{\min}$, and $\bar{\mathbf{U}}_i^{\max}$ are the minimum and maximum element-averaged solution on the elements sharing faces with Ω_i. If Eq. (18) is violated for any quadrature points, then it is assumed that the element is close to a discontinuity, and the solution at this element Ω_i is locally modified (limited) as

$$\mathbf{U}_i(\mathbf{x}) = \bar{\mathbf{U}}_i + \alpha \nabla \mathbf{U}_i \cdot (\mathbf{x} - \mathbf{x}_i) \quad \forall\ \mathbf{x} \in \Omega_i \tag{19}$$

where $\bar{\mathbf{U}}_i$ is the cell-averaged solution at the element Ω_i, $\mathbf{x}_i$ is the position vector of the centroid of Ω_i, and

$$\alpha = \min_{1 \le j \le K_{\Gamma_i}} \alpha_j \tag{20}$$

$$\alpha_j = \begin{cases} \min\left(1, \frac{\bar{\mathbf{U}}_i^{\max} - \bar{\mathbf{U}}_i}{\mathbf{U}_i(\mathbf{x}_j) - \bar{\mathbf{U}}_i}\right) & \text{if } \mathbf{U}_i(\mathbf{x}_j) - \bar{\mathbf{U}}_i > 0; \\ \min\left(1, \frac{\bar{\mathbf{U}}_i^{\min} - \bar{\mathbf{U}}_i}{\mathbf{U}_i(\mathbf{x}_j) - \bar{\mathbf{U}}_i}\right) & \text{if } \mathbf{U}_i(\mathbf{x}_j) - \bar{\mathbf{U}}_i < 0; \\ 1 & \text{otherwise} \end{cases} \tag{21}$$

It has been numerically demonstrated [25] that just as DGMs are more sensitive to the treatment and implementation of slip boundary conditions at curved boundaries than those obtained with finite volume methods of the same order of accuracy, DGMs are also much more sensitive to the treatment and implementation of the slope limiters than their finite volume (FV) counterparts. Slope limiters frequently identify regions near smooth extrema as requiring limiting, and this typically results in a reduction of the optimal high-order convergence rate. For aerodynamic applications, the active limiters close to the smooth extrema, such as a leading edge of an airfoil, will pollute the solution in the flowfield and ultimately destroy the high-order-accuracy solution. To address this concern, the limiters are applied only where they are really needed. This is accomplished using the so-called discontinuity detector [26], which is helpful to distinguish regions in which solutions are smooth and discontinuous. Then the limiting is only used near discontinuities, and high-order accuracy can be preserved in smooth regions. The detailed implementation of this procedure can be found in [25,26]. Note that construction of an accurate, efficient, and robust limiter remains one of the issues and challenges for the DG methods.

IV. *p*-Multigrid Method

Nowadays, geometric multigrid methods (referred to as *h*-multigrid from now on) are routinely used to accelerate the convergence of the Euler and Navier–Stokes equations to a steady state on unstructured grids. It is well established that *h*-multigrid acceleration can drastically reduce the computational costs. In standard *h*-multigrid methods, solutions on spatially coarse grids are

used to correct solutions on the fine grid. The p-multigrid method is a natural extension of h-multigrid methods to high-order finite element formulation, such as spectral-hp or discontinuous Galerkin methods, in which systems of equations are solved by recursively iterating on solution approximations of different polynomial orders. For example, to solve equations derived using a polynomial approximation order of 3, the solution can be iterated on at an approximation order of $p = 2$, 1, and 0. The basic idea of a p-multigrid method is to perform time steps on the lower-order approximation levels to calculate corrections to a solution on a higher-order approximation level. Unlike our previous work [19], in which only two-level V-cycle p-multigrid method was used to drive the iterations for both DG(P2) and DG(P1) methods, the successive $p-1$ discretization is chosen as the coarse approximation. This was found to be more efficient and robust than the two-level V-cycle for the DG(P2) method. Specifically, this p-multigrid method for the DG (P2) method consists of the following steps at each p-multigrid cycle:

1) Perform a time step at the highest approximation order P2, which yields the initial solution $\mathbf{U}_{\mathrm{P2}}^{n+1}$.

2) Restrict the solution and residual vectors from P2 to one lower-level approximation P1:

$$\mathbf{U}_{\mathrm{P1}} = \mathbf{I}_{\mathrm{P2}}^{\mathrm{P1}}\mathbf{U}_{\mathrm{P2}}^{n+1}, \qquad \mathbf{R}_{\mathrm{P1}} = \tilde{\mathbf{I}}_{\mathrm{P2}}^{\mathrm{P1}}\mathbf{R}(\mathbf{U}_{\mathrm{P2}}^{n+1}) \tag{22}$$

3) Compute the force terms on the lower-level approximation P1:

$$\mathbf{F}_{\mathrm{P1}} = \mathbf{R}_{\mathrm{P1}} - \mathbf{R}(\mathbf{U}_{\mathrm{P1}}) \tag{23}$$

4) Perform a time step at the lower-level approximation P1 for which the residual is given by

$$\mathbf{R} = \mathbf{R}(\mathbf{U}_{\mathrm{P1}}) + \mathbf{F}_{\mathrm{P1}} \tag{24}$$

which yields the solution at the lower-level $\mathbf{U}_{\mathrm{P1}}^{n+1}$.

5) Restrict the solution and residual vectors from P1 to one lower-level approximation P0:

$$\mathbf{U}_{\mathrm{P0}} = \mathbf{I}_{\mathrm{P1}}^{\mathrm{P0}}\mathbf{U}_{\mathrm{P1}}^{n+1}, \qquad \mathbf{R}_{\mathrm{P0}} = \tilde{\mathbf{I}}_{\mathrm{P1}}^{\mathrm{P0}}(\mathbf{R}(\mathbf{U}_{\mathrm{P1}}^{n+1}) + \mathbf{F}_{\mathrm{P1}}) \tag{25}$$

6) Compute the force terms on the lower-level approximation P0:

$$\mathbf{F}_{P0} = \mathbf{R}_{P0} - \mathbf{R}(\mathbf{U}_{P0}) \tag{26}$$

7) Perform a time step at the lower-level approximation P0, where the residual is given by

$$\mathbf{R} = \mathbf{R}(\mathbf{U}_{\mathrm{P0}}) + \mathbf{F}_{\mathrm{P0}} \tag{27}$$

which yields the solution at the lower-level $\mathbf{U}_{\mathrm{P0}}^{n+1}$.

8) Prolongate the correction $\mathbf{C}_{\mathrm{P0}}$ from the lowest-level P0 to update the higher-level solution $\mathbf{U}_{\mathrm{P1}}^{n+1}$:

$$\mathbf{C}_{\mathrm{P0}} = \mathbf{U}_{\mathrm{P0}}^{n+1} - \mathbf{U}_{\mathrm{P0}}, \qquad \tilde{\mathbf{U}}_{\mathrm{P1}}^{n+1} = \mathbf{U}_{\mathrm{P1}}^{n+1} + \mathbf{J}_{\mathrm{P0}}^{\mathrm{P1}}\mathbf{C}_{\mathrm{P0}} \tag{28}$$

9) Prolongate the correction $\mathbf{C}_{\mathrm{P1}}$ back from the level P1 to update the higher-level solution $\mathbf{U}_{\mathrm{P2}}^{n+1}$:

$$\mathbf{C}_{\mathrm{P1}} = \tilde{\mathbf{U}}_{\mathrm{P1}}^{n+1} - \mathbf{U}_{\mathrm{P1}}, \qquad \tilde{\mathbf{U}}_{\mathrm{P2}}^{n+1} = \mathbf{U}_{\mathrm{P2}}^{n+1} + \mathbf{J}_{\mathrm{P1}}^{\mathrm{P2}}\mathbf{C}_{\mathrm{P1}} \tag{29}$$

The preceding single p-multigrid cycle will produce a more accurate solution at the higher level, starting from an initial solution at the same level, where $\mathbf{I}$ is the state restriction operator, $\mathbf{J}$ is the state prolongation operator, and $\tilde{\mathbf{I}}$ is the residual restriction operator and is not necessarily the same as the state restriction operator. The definition of these operators can be introduced in a standard manner using the basis of the finite element approximation spaces. Specifically,

$$\mathbf{I}_{\ p}^{q} = (M^{q})^{-1}M^{qp} \tag{30}$$

$$\mathbf{J}_{\ q}^{p} = (M^{p})^{-1}M^{pq} \tag{31}$$

and

$$\tilde{\mathbf{I}}_{\ p}^{q} = M^{qp}(M^{p})^{-1} \tag{32}$$

where

$$M_{i,j}^{p} = \int_{\Omega_e} B_i^p B_j^p \, \mathrm{d}\Omega \tag{33}$$

$$M_{i,j}^{pq} = \int_{\Omega_e} B_i^p B_j^q \, \mathrm{d}\Omega \tag{34}$$

Note that the prolongation and restriction operators between orders are local to the element. As a result, they can be easily computed by hand, considering one element at a time in advance.

In general, the same time-integration scheme is applied to advance the solution on all levels P2, P1, and P0. Implicit time-integration schemes such as element Jacobian and element line Jacobian methods have been used as smoothers for p-multigrid for solving the compressible Navier–Stokes equations [18]. Unfortunately, they require prohibitively large memory and computing costs for a Jacobian matrix, rendering them impractical, if not impossible, for large-scale problems and, especially, for high-order solutions. Furthermore, the implementation of slope limiters for DG methods in any implicit schemes is problematic and difficult, limiting them to only smooth flows without shocks or discontinuities.

On the other hand, when the multistage TVD Runge–Kutta explicit scheme is used as an iterative smoother, the performance of the resulting p-multigrid method is quite disappointing [19], in contrast to the success that h-multigrid methods enjoyed to accelerate the convergence of the Euler equations using the multistage Runge–Kutta explicit scheme as an iterative smoother. This is mainly due to the fact that explicit schemes are inefficient to reduce lower-frequency errors on the lowest level, though they are fairly efficient at eliminating high-frequency error modes in the solution (i.e., local error). By transferring the discrete equations to a coarse approximation level, once the high-frequency error modes on the fine approximation level have been eliminated, the lower-frequency modes from the fine approximation level now appear as higher-frequency modes on the coarse approximation level in the p-multigrid scheme and are effectively handled by the explicit scheme on this approximation level. This observation motivates us to use an explicit smoother on the higher approximation levels P1 and P2 and an implicit smoother on the coarsest level P0. Implicit smoothers have better convergence properties and are far more effective at eliminating the lowest-frequency errors, and yet the storage requirements and computational costs for the Jacobian matrix on the coarsest level P0 are relatively small. Note that the DG(P0) method, corresponding to zero-order basis functions, degenerates to the classical first-order cell-centered finite volume scheme, so that the fairly mature implicit methods developed over the last decades can be readily used as the implicit iterative smoother. Using Euler implicit time integration, the spatially discretized Euler equations on the P0 level can be linearized in time and written as

$$\left(\frac{V}{\Delta t}\mathbf{I} - \frac{\partial \mathbf{R}_{P0}}{\partial \mathbf{U}_{P0}}\right)\Delta \mathbf{U}_{P0} = \mathbf{R}_{P0} \tag{35}$$

where V is the element volume, and $\mathbf{R}_{\mathrm{P0}}$ is the right-hand-side residual. The subscript P0 will be omitted from here on. Equation (35) represents a large system of linear simultaneous algebraic equations that needs to be solved. In this study, the symmetric Gauss–Seidel (SGS) method [27] is used to solve the linear system (35). The SGS method with k iterations, called SGS(k) hereafter, can be written as

Initialization:

$$\Delta \mathbf{U}^{0} = 0 \tag{36}$$

Forward Gauss–Seidel iteration:

$$(D + L)\Delta \mathbf{U}^{k+\frac{1}{2}} + U\Delta \mathbf{U}^{k} = \mathbf{R} \tag{37}$$

Backward Gauss–Seidel iteration:

$$(D+U)\Delta\mathbf{U}^{k+1} + L\Delta\mathbf{U}^{k+\frac{1}{2}} = \mathbf{R} \tag{38}$$

where U, L, and D represent strict upper, strict lower, and diagonal matrices, respectively, and k is the number of iterations for the SGS method. The main advantage of the SGS method is that it does not require any additional storage beyond that of the matrix itself. Note that if only one iteration is used in the SGS method and the initial guess is set to zero, the resulting method is nothing but the so-called lower/upper-SGS method [28,29], which can be written as follows:

Lower (forward) sweep:

$$(D+L)\Delta\mathbf{U}^{\star} = \mathbf{R} \tag{39}$$

Upper (backward) sweep:

$$(D+U)\Delta\mathbf{U}^{\star} = D\Delta\mathbf{U}^{\star} \tag{40}$$

It is clear that the preceding algorithms primarily involve the Jacobian-matrix-solution incremental vector product. Such an operation can be approximately replaced by computing increments of the flux vector product. Such a Jacobian-matrix-solution incremental vector can then be approximately replaced by computing increments of the flux vector. This is achieved by using the Lax–Friedrich flux function (scalar dissipation) to derive the left-hand-side matrix. The detailed matrix-free approach can be found in [27,28]. The most remarkable achievement of this approximation is that there is no need to store the upper and lower matrices U and L, which substantially reduces the memory requirements. The only matrix needed to store is the diagonal matrix D, which requires a memory of neqns × neqns × nelem, where neqns is the number of components in the solution vector (four for 2D and five for 3D Euler equations), and nelem is the number of elements for the grid. The storage of diagonal matrix only requires 16 words per element in 2D and 25 words per element in 3D. Compared with the memory requirement of about 100 words per element for the explicit DG(P1) method and 250 words per element for the explicit DG(P2) method in 2D, the additional storage requirement for this p-multigrid method is not significant at all. Furthermore, it is easy and straightforward to use this p-multigrid method to solve flow problems with discontinuities, because the slope or flux limiters for higher-order DG methods are easy to implement in the explicit smoother and no limiter is required for the implicit smoother on the coarsest level P0, where DG(P0) degenerates to the classical first-order cell-centered finite volume scheme.

V. Computational Results

All of the computations are performed on a Dell Precision M70 laptop computer (2.13-GHz Pentium M CPU with 2-GB memory) using a Suse 9.3 Linux operating system. The relative L_2 norm of the density residual is taken as a criterion to test convergence history. The explicit time-integration method for DG methods uses the explicit, three-stage, third-order, TVD, Runge–Kutta time-stepping scheme described in Sec. III.B with a local time-stepping technique, termed as TVDRKDG from now on, which also serves as the smoother on high-order approximations for the p-multigrid method. The implicit smoother on the coarsest level P0 used for the p-multigrid method is the matrix-free SGS method with five iterations, presented in Sec. IV. All computations are initiated with uniform flows and are carried out using a CFL number of 2 for the explicit method and 20 for the implicit method (used as iterative smoother in the p-multigrid method), unless stated otherwise. The p-multigrid method is compared with the explicit method for some test cases, to demonstrate its efficiency. A well-tested finite volume code [22,28] is used as a reference to compare the accuracy and performance of the DG method for some test cases.

A. Subsonic Flow Past a Circular Cylinder

The first example is a well-known test case: subsonic flow past a circular cylinder at a Mach number of $M_\infty = 0.38$. This test case is chosen to verify the implementation of boundary conditions for curved geometries for DG methods, assess the order of accuracy of the discontinuous Galerkin method, and numerically compare accuracy and performance between DG and FV methods. Figure 1 shows four successively refined o-type grids having 16 × 5, 32 × 9, 64 × 17, and 128 × 33 points. The first number refers to the number of points in the circular direction, and the second number designates the number of concentric circles in the mesh. The radius of the cylinder is $r_1 = 0.5$, the domain is bounded by $r_{33} = 20$, and the radii of concentric circles for 128 × 33 mesh are set up as

$$r_i = r_1\left(1 + \frac{2\pi}{128}\sum_{j=0}^{i-1}\alpha^j\right) \qquad i = 2, \ldots, 33$$

where $\alpha = 1.1580372$. The coarser grids are generated by successively unrefining the finest mesh.

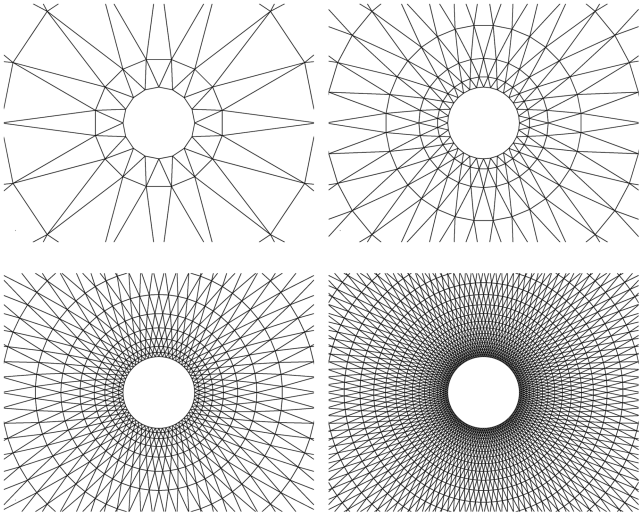

Fig. 1 Sequences of four successively globally refined meshes for computing subsonic flow past a circular cylinder: 16 × 5, 32 × 9, 64 × 17, 128 × 33.

Table 1 Subsonic circular cylinder test case: FV(P1) is $\mathcal{O}(h^2)$

Mesh	No. DOFs	L^2-error	Order
16 × 5	80	2.37148E-01	——
32 × 9	288	7.76551E-02	1.595
64 × 17	1088	1.36962E-02	2.551
128 × 33	4224	3.54568E-03	1.951

Table 2 Subsonic circular cylinder test case: DG(P1) is $\mathcal{O}(h^2)$

Mesh	No. DOFs	L^2-error	Order
16 × 5	360	5.68722E-02	——
32 × 9	1536	1.07103E-02	2.443
64 × 17	6144	1.67302E-03	2.688
128 × 33	24,576	2.34369E-04	2.838

Table 3 Subsonic circular cylinder test case: DG(P2) is $\mathcal{O}(h^3)$

Mesh	No. DOFs	L^2-error	Order
16 × 5	768	8.40814E-03	——
32 × 9	3072	5.26017E-04	4.055
64 × 17	12,288	4.48952E-05	3.563
128 × 33	49,152	4.16294E-06	3.434

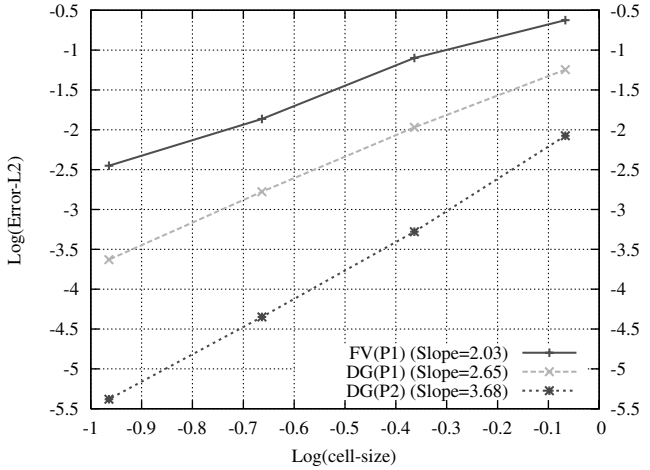

Fig. 2 Accuracy summary for subsonic flow past a circular cylinder for a second-order finite volume method, second-order discontinuous Galerkin method, and a third-order discontinuous Galerkin method.

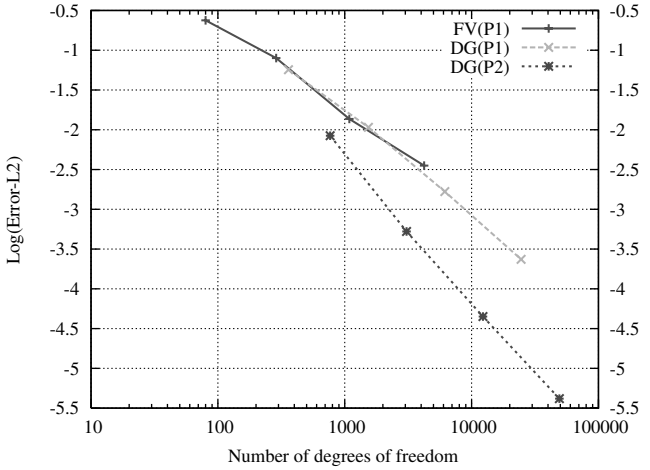

Fig. 3 L^2-error of numerical solutions against the number of degrees of freedom for subsonic flow past a circular cylinder by the FV(P1), DG (P1), and DG(P2) methods.

Numerical solutions to this problem are computed using FV(P1), DG(P1), and DG(P2) methods on these four grids to obtain quantitative measurement of the order of accuracy and discretization errors. The detailed results of this test case are presented in Tables 1–3. They show the mesh size, the number of degrees of freedom, the L^2-error of the solutions, and the order of convergence. In this case, the following entropy production ϵ defined as

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

is served as the error measurement, where S is the entropy. Note that the entropy production serves as a good criterion to measure accuracy of the numerical solutions, because the flow under consideration is isentropic. Figure 2 provides the details of the spatial accuracy of each method for this numerical experiment. Figure 3 shows the L^2-error of the FV(P1), DG(P1), and DG(P2) methods plotted against the number of degrees of freedom. The results obtained by DG methods (similar to those found in the literature [8], in which isoparametric elements are used to represent the curved boundaries) indicate that the discontinuous Galerkin method exhibits a full $\mathcal{O}(h^{p+1})$ order of convergence on smooth solutions.

The results also indicate that our simplified implementation of boundary conditions for curved geometries offers the same convergence rates as the high-order isoparametric finite element approximation approach [8]. Figure 3 illustrates that a higher-order

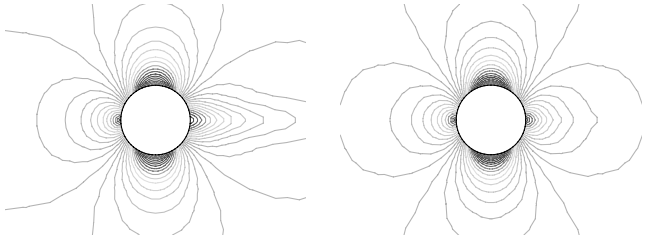

Fig. 4 Computed Mach number contours obtained by the finite volume method (left) and DG(P1) method (right) on 64 × 17 mesh for subsonic flow past a circular cylinder at $M_\infty = 0.38$.

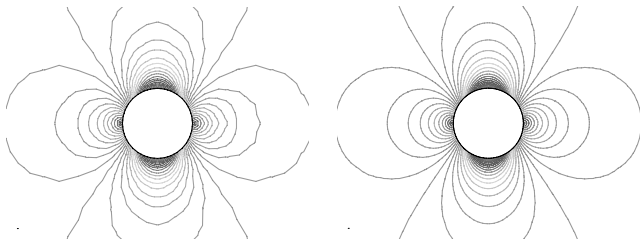

Fig. 5 Computed Mach number contours obtained by the DG(P2) method (left) on 32 × 9 mesh and DG(P2) method (right) on 64 × 17 mesh for subsonic flow past a circular cylinder at $M_\infty = 0.38$.

DG method requires significantly reduced degrees of freedom than a lower-order DG method to achieve the same accuracy. In fact, the DG(P2) solution on a given coarse mesh is actually more accurate than the DG(P1) solution on a double-resolution mesh. The results obtained by DG(P1) appear to be much more accurate than those obtained by its finite volume counterpart FV(P1) on the same mesh, as shown on the Mach number contours in the flowfield in Figs. 4 and 5.

These observations become especially apparent in Fig. 6, in which one compares the pressure coefficient and entropy production on the surface of cylinder obtained by DG(P1), DG(P2), and FV(P1) on the 64 × 17 mesh and DG(P2) on the 32 × 9 mesh. Figure 7 displays a comparison of convergence histories versus time steps for different meshes using the TVDRKDG(P1) method (left) and different orders of the TVDRKDG method on the 32 × 9 mesh (right). Convergence of the explicit TVDRKDG method deteriorates drastically when the number of degrees of freedom and, especially, the polynomial order increase. To demonstrate the importance and effect of the implicit SGS solutions at the P0 level on the convergence of the p-multigrid method, Fig. 8 shows a comparison of convergence histories versus time steps, respectively, for the P1, P2, and P3 element approximations using the p-multigrid method on the 32 × 9 mesh, with 5 (left) and 20 (right) iterations for the matrix-free SGS method on the P0 level.

Figures 9–11 display a comparison of convergence histories versus time steps and CPU time for the P1, P2, and P3 element approximations using SGS(1), SGS(5), and SGS(20), respectively, on the 32 × 9 mesh. Clearly, a lack of convergence at the P0 level using one iteration for the SGS method is not a good choice for the performance of the p-multigrid method. The full convergence at the P0 level using 20 iterations for the SGS method tends to achieve the order independence of this p-multigrid method at the expense of computing time. The best convergence in terms of CPU time is obtained using five iterations for the SGS method at the P0 level. In this case, the p-multigrid method converges in nearly the same number of time steps for the P1, P2, and P3 solutions, demonstrating the near-order independence of this p-multigrid. Finally, Fig. 12 shows a comparison of convergence histories versus time steps for DG(P2) solutions on different meshes using the p-multigrid method. Although the present p-multigrid method is unable to obtain the mesh-independent solutions, it does not deteriorate as drastically as its explicit counterpart.

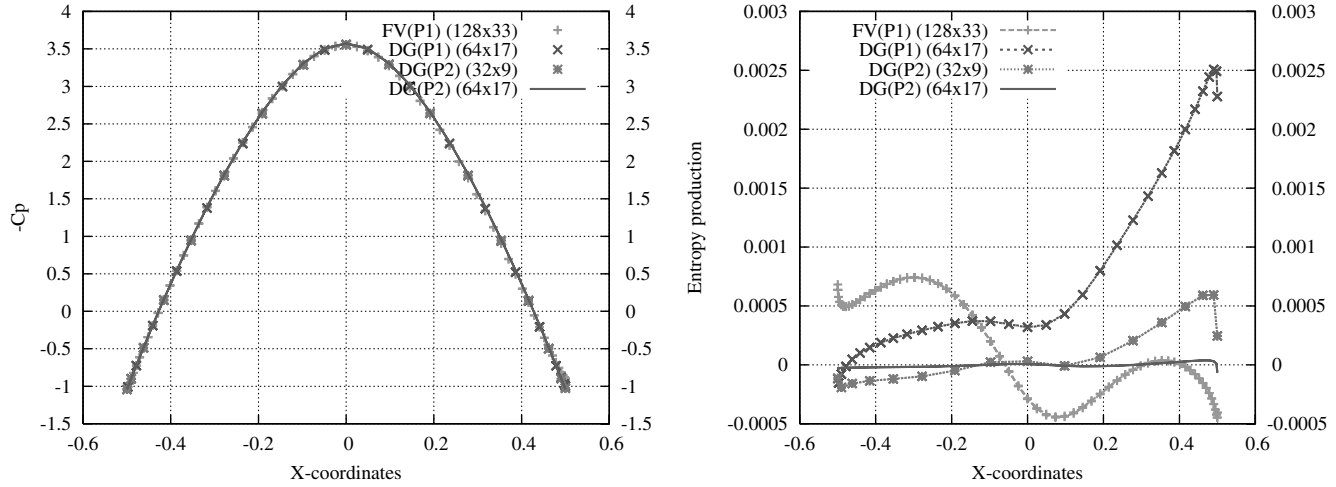

Fig. 6 Comparison of computed pressure coefficient (left) and entropy production (right) on the lower surface obtained by the FV(P1) on 128 × 33 mesh, DG(P1) on 64 × 17 mesh, and DG(P2) on 32 × 9 mesh for subsonic flow past a circular cylinder at $M_\infty = 0.38$.

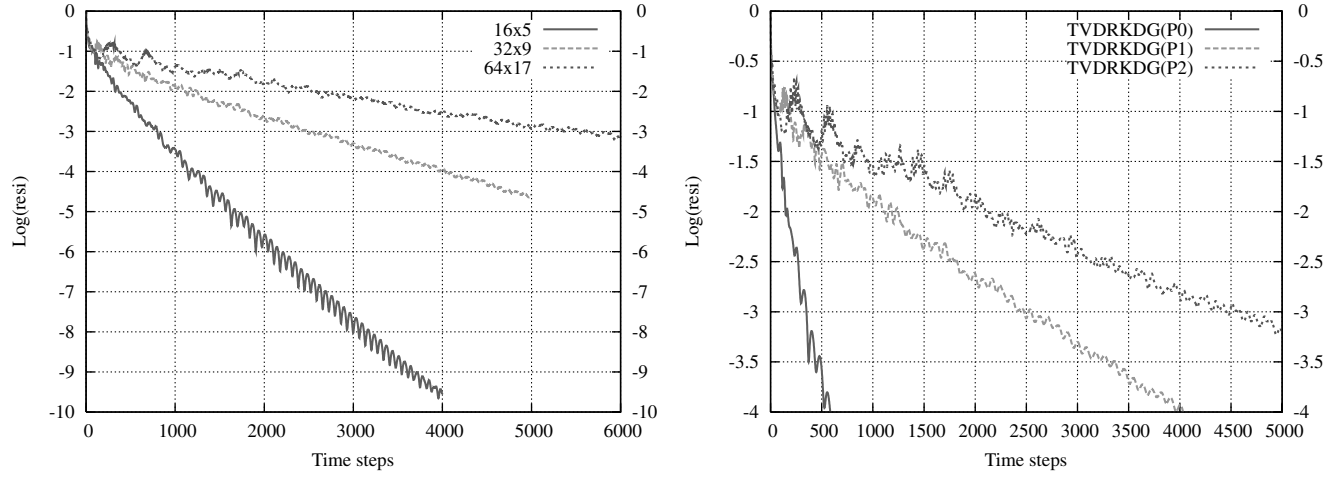

Fig. 7 Comparison of convergence history versus time steps for TVDRK(P1) solutions on different meshes (left) and for different orders of TVDRK solutions on 32 × 9 mesh (right) for subsonic flow past a circular cylinder.

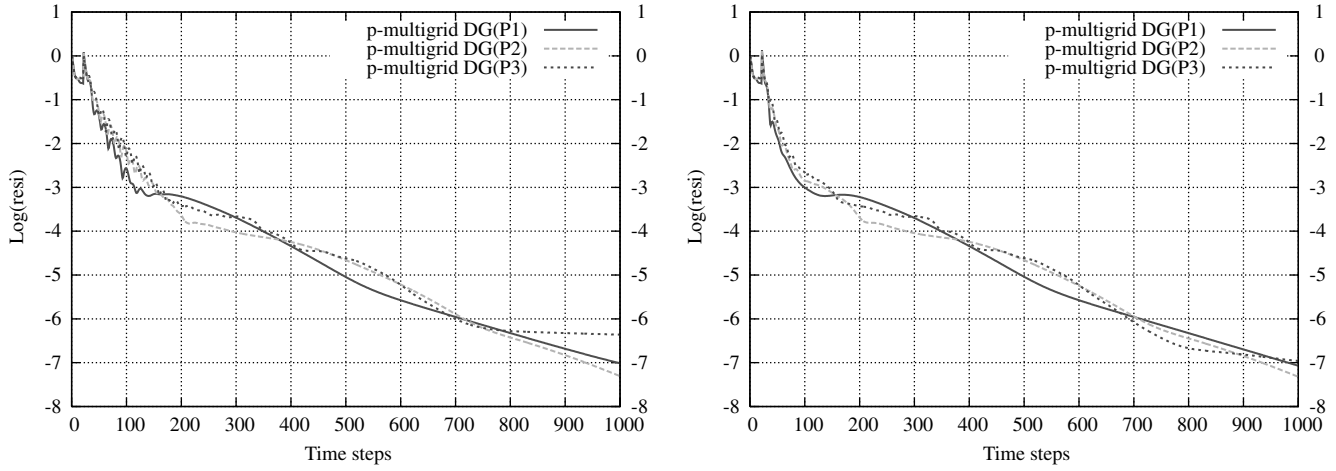

Fig. 8 Comparison of convergence history versus time steps for the *p*-multigrid method using 5 (left) and 20 (right) iterations for the matrix-free SGS method at the P0 level on 32 × 9 mesh for subsonic flow past a circular cylinder.

B. Low-Mach-Number Flow Past a Circular Cylinder

This test case is chosen to demonstrate the accuracy of the DG method at low Mach numbers. The computation is performed for low-speed flows past a half-circular cylinder at a Mach number of $M_\infty = 0.01$ using both the DG(P1) and FV(P1) methods on the mesh shown in Fig. 13, in which the comparison of the velocity distributions on the lower surface obtained by these two methods and those of the analytic solution for incompressible flows is presented as

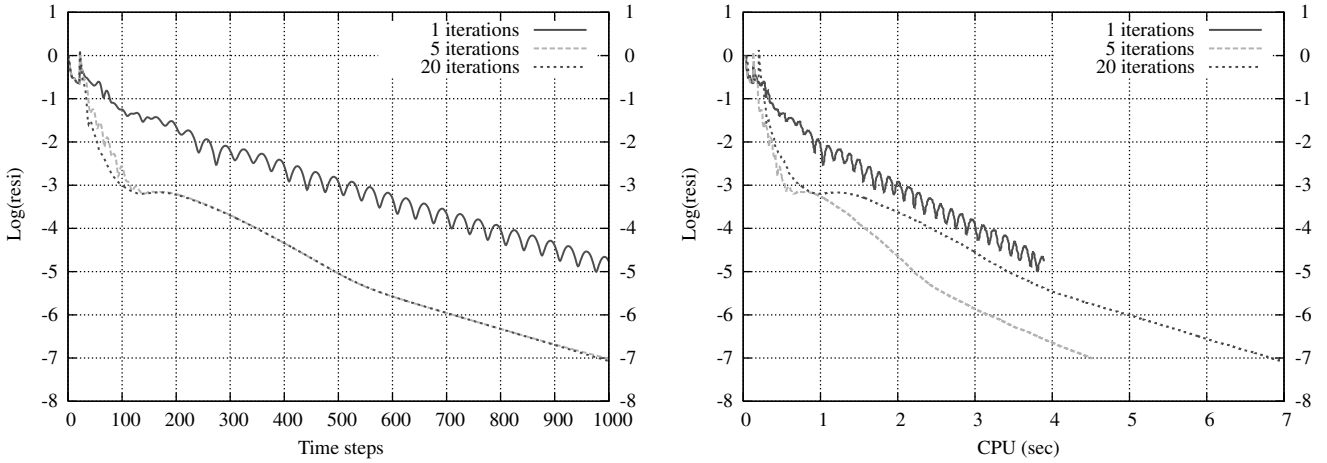

Fig. 9 Comparison of convergence history versus time steps (left) and CPU time (right) for the *p*-multigrid method using different iterations for the SGS method on the P0 level for the P1 solution on 32 × 9 mesh for subsonic flow past a circular cylinder.

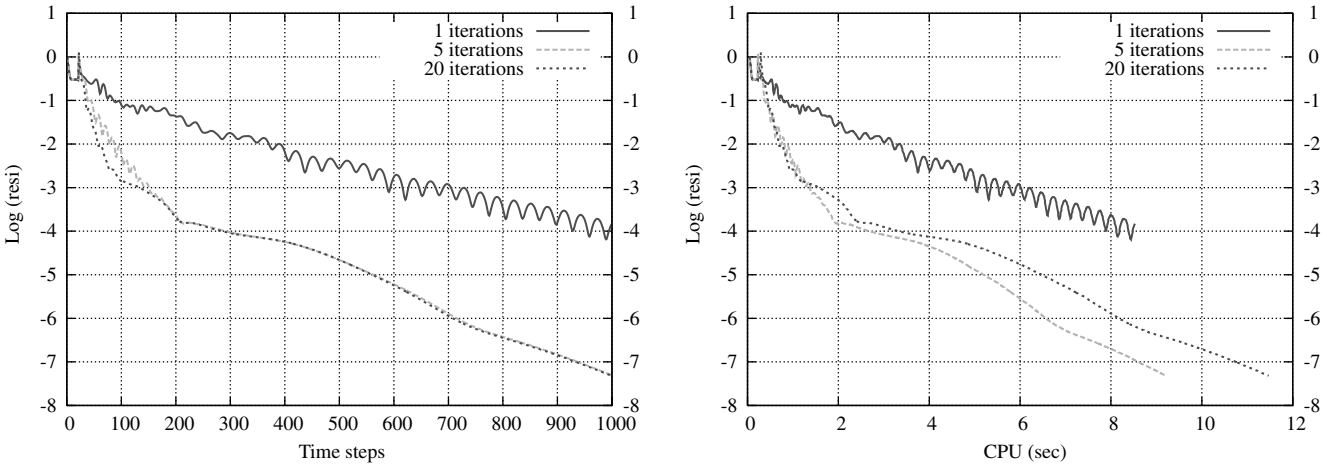

Fig. 10 Comparison of convergence history versus time steps (left) and CPU time (right) for the *p*-multigrid method using different iterations for the SGS method on the P0 level for the P2 solution on 32 × 9 mesh for subsonic flow past a circular cylinder.

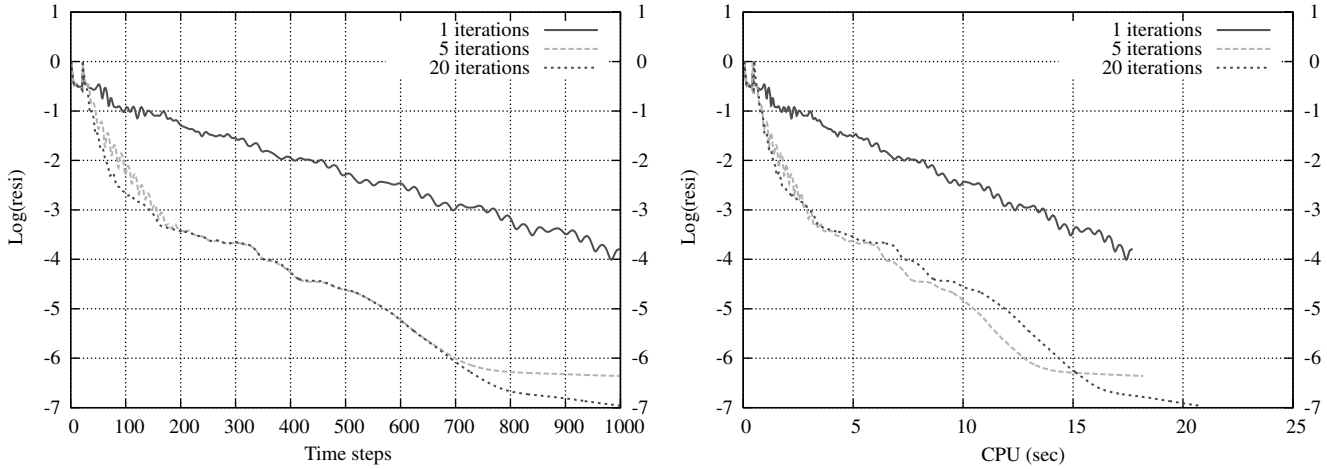

Fig. 11 Comparison of convergence history versus time steps (left) and CPU time (right) for the *p*-multigrid method using different iterations for the SGS method on the P0 level for the P3 solution on 32 × 9 mesh for subsonic flow past a circular cylinder.

well. The mesh consists of 2981 elements, 1564 grid points, and 145 boundary points. The computed Mach number contours in the flowfield obtained using the FV(P1) and DG(P1) methods are shown in Fig. 14. The computed pressure contours in the flowfield obtained using FV(P1) and DG(P1) methods are presented in Fig. 15. The results illustrate dramatic deterioration of FV(P1) solution, whereas DG(P1) is able to produce an accurate solution. Both methods use the same HLLC scheme for computing numerical flux function.

C. Transonic Flow Past a NACA0012 Airfoil

The third example is the transonic flow past a NACA0012 airfoil at a Mach number of 0.80 and an angle of attack of 1.25 deg,

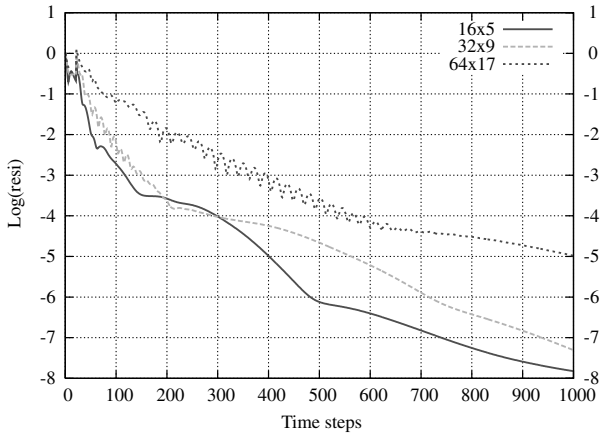

Fig. 12 Comparison of convergence history versus time steps for the *p*-multigrid P2 solutions on different meshes for subsonic flow past a circular cylinder.

characterized by the existence of a strong shock on the upper surface and a weak shock on the lower surface. This test case is chosen primarily to test the accuracy of the DG(P2) method and the performance of the *p*-multigrid method for transonic flows. Two grids of similar quality are generated, as shown in Fig. 16. The coarse mesh has 1999 elements, 1048 grid points, and 97 boundary points, and the fine mesh consists of 8006 elements, 4102 grid points, and 198 boundary points. Figure 17 shows the computed pressure contours in the flowfield obtained using DG(P2) computation on the coarse mesh and FV(P1) computation on the fine mesh. The computed Mach number contours in the flowfield obtained using DG (P2) computation on the coarse mesh and FV(P1) computation on the fine mesh are shown in Fig. 18. Figure 19 compares the Mach number and entropy production on the surface of the airfoil obtained by the DG(P2) computation on the coarse mesh and FV(P1) computations on the fine mesh, respectively. The entropy production results clearly show that the DG(P2) solution on a coarse mesh is much more accurate than FV(P1) solution on a double-sized mesh. Figure 20 displays a comparison of convergence histories versus time steps and CPU time obtained by these two methods. Although the implicit finite volume solution converges faster than the *p*-multigrid discontinuous Galerkin solution for this test case, the *p*-multigrid DG method should be regarded as highly competitive, because the implicit FV method in terms of computing cost, due to the more accurate solution it provides.

D. Supersonic Inlet Flow

This example is the supersonic flow entering a generic inlet configuration that is typical of scramjet engines. This test case is chosen to test the ability of higher-order DG method for accurately computing supersonic flows. The configuration is taken from [6], and the prescribed Mach number at the inlet is $M_\infty = 3$. The mesh used in the computation, which contains 7993 elements, 4276 points, and 559 boundary points, is depicted in Fig. 21. Because of the special

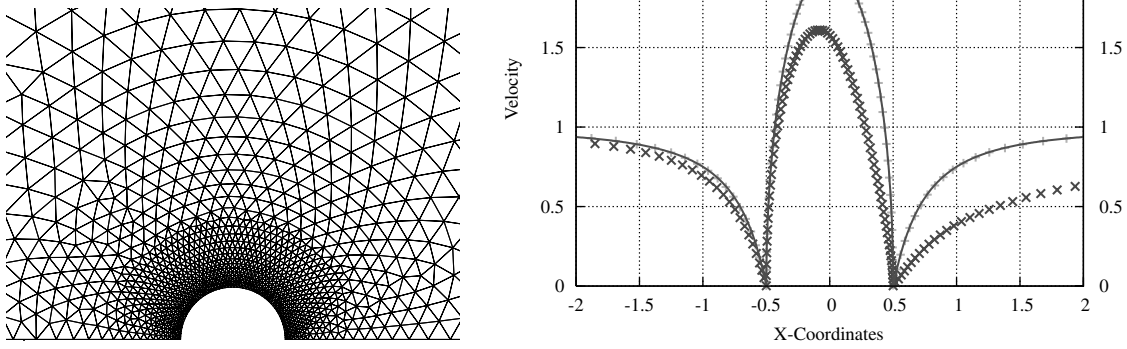

Fig. 13 Left: Unstructured grid (nelem = 2981, npoin = 1564, nboun = 145) used for computing low-Mach-number flow past a half-circular cylinder. Right: Comparison of computed velocity distributions on the lower surface obtained by the FV(P1) and DG(P1) methods with the incompressible solution for low-Mach-number flow past a circular cylinder at $M_\infty = 0.01$.

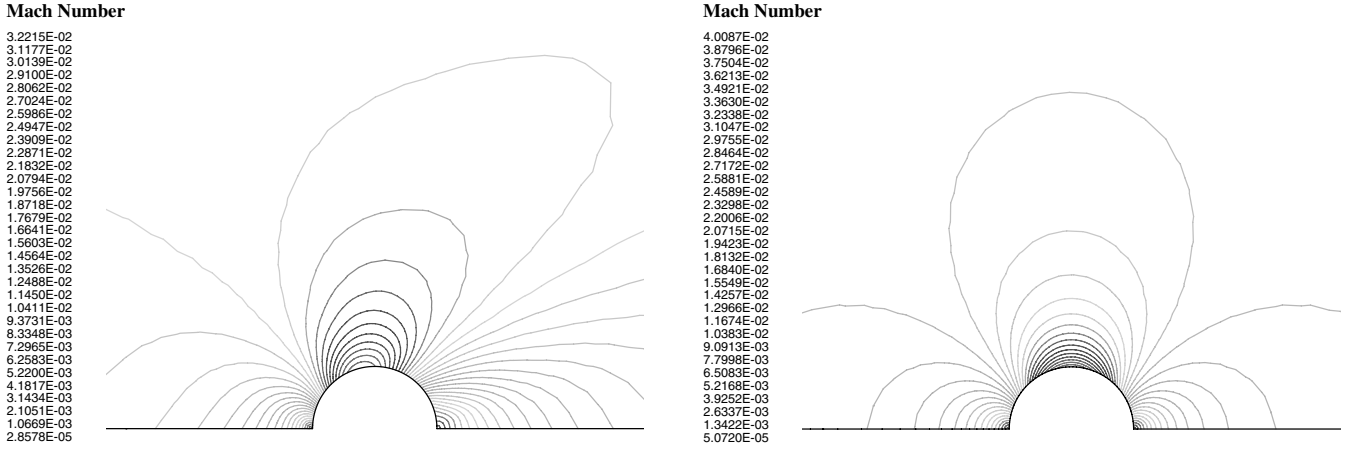

Fig. 14 Computed Mach number contours obtained by the FV(P1) (left) and DG(P1) (right) methods for low-Mach-number flow past a circular cylinder at $M_\infty = 0.01$.

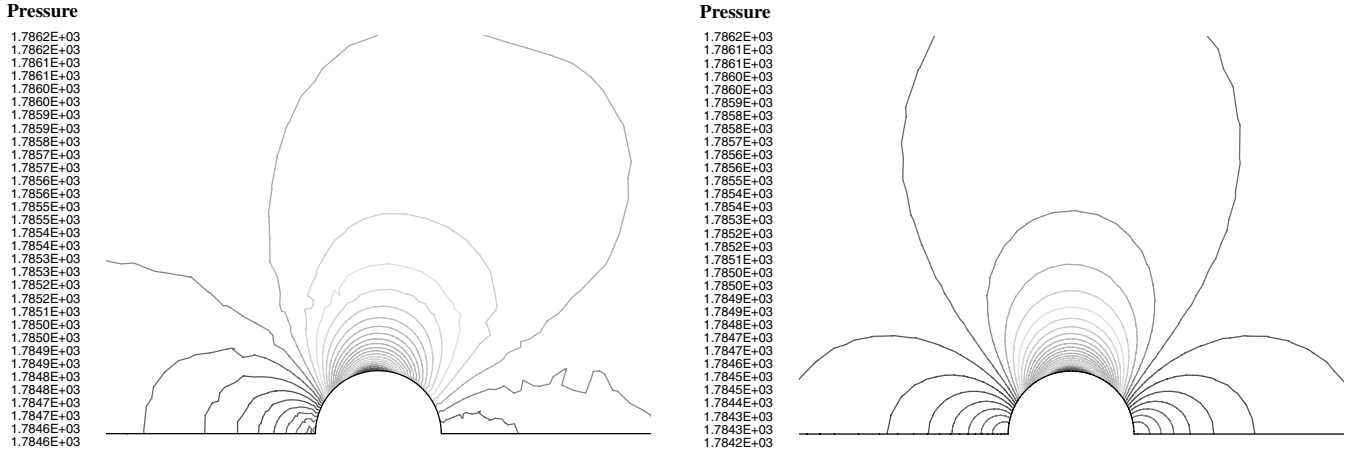

Fig. 15 Computed pressure contours obtained by the FV(P1) (left) and DG(P1) (right) methods for low-Mach-number flow past a circular cylinder at $M_\infty = 0.01$.

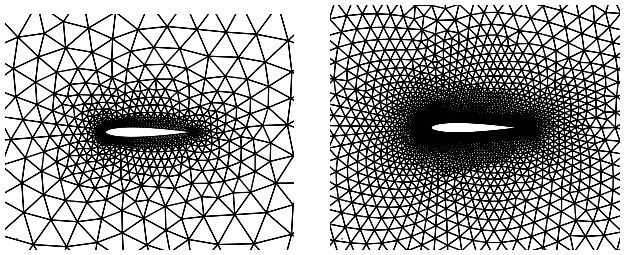

Fig. 16 Coarse (left, nelem = 1999, npoin = 1048, nboun = 97) and fine (right, nelem = 8006, npoin = 4102, nboun = 198) unstructured mesh used for computing transonic flow past a NACA0012 airfoil.

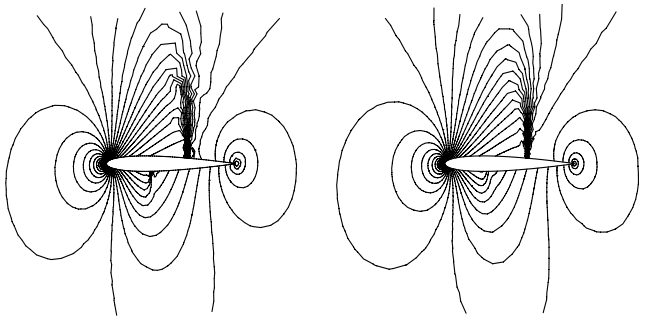

Fig. 17 Computed pressure contours in the flowfield by DG(P2) method on the coarse mesh (left) and FV(P1) method on the fine mesh (right) for transonic flow past a NACA0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

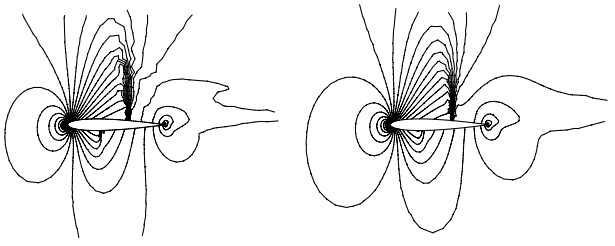

Fig. 18 Computed Mach number contours in the flowfield by the DG (P2) method on the coarse mesh (left) and FV(P1) method on the fine mesh (right) for transonic flow past a NACA0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

configuration inside the inlet, very complex flow features appear. The computed Mach number contours in the flowfield obtained using FV(P1), DG(P1), and DG(P2) computations are shown in Figs. 22–24, respectively. Although both FV(P1) and DG(P1) methods are able to produce similar flow features, DG(P1) produces a more accurate solution than its second-order counterpart, FV(P1). As expected, DG(P2) delivers the best solution witnessed by the sharp resolution of shock waves and slip line.

E. Subsonic Flow Past a Sphere

An example is presented for 3D subsonic flow past a sphere at a Mach number of $M_\infty = 0.5$. This test case is chosen to verify the implementation of boundary conditions for curved geometries for DG methods and to assess the order of accuracy of the discontinuous Galerkin method and the performance of p-multigrid for 3D configurations. Figure 25 shows three successively refined unstructured grids having 2174, 17,140, and 137,028 elements and the computed Mach number contours in the flowfield obtained by DG(P2) on the coarse mesh, DG(P1) on the medium mesh, and DG (P0) on the fine mesh. Note that only a quarter of the configuration is modeled, due 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 a smoothing procedure after dividing a tetrahedron into eight smaller tetrahedra. Numerical solutions to this problem are computed using DG(P0), DG(P1), and DG(P2) methods on these three grids to obtain quantitative measurement of the order of accuracy and discretization errors. As in the 2D case, the entropy production serves as the error measurement. Figure 26 provides spatial accuracy details of each method for this numerical experiment.

Figure 27 shows the L^2-error of the DG(P0), DG(P1), and DG(P2) methods plotted against the number of degrees of freedom. The results obtained by the DG method, perhaps not as impressive as those shown in the 2D study (likely due the 3D grid quality), do indicate that the discontinuous Galerkin method exhibits a $\mathcal{O}(h^{p+1})$ order of convergence on smooth solutions. Results also show that our simplified implementation of boundary conditions for curved geometries well conserves the formal order of the DG method. In addition, the higher-order DG method requires a more significantly reduced number of degrees of freedom than the lower-order DG method to achieve the same accuracy. In fact, the DG(P2) solution on a given coarse mesh is actually more accurate than the DG(P1) solution on a double-refined mesh. The advantage of the higher-order method is again demonstrated for 3D configurations.

Figure 28 displays a comparison of convergence histories versus time steps on the coarse, medium, and fine grids using the TVDRKDG(P0) method, and Fig. 29 displays a comparison of convergence histories versus time steps, obtained using different orders of the TVDRKDG method on the coarse mesh. One can clearly see that the convergence of the explicit TVDRKDG method

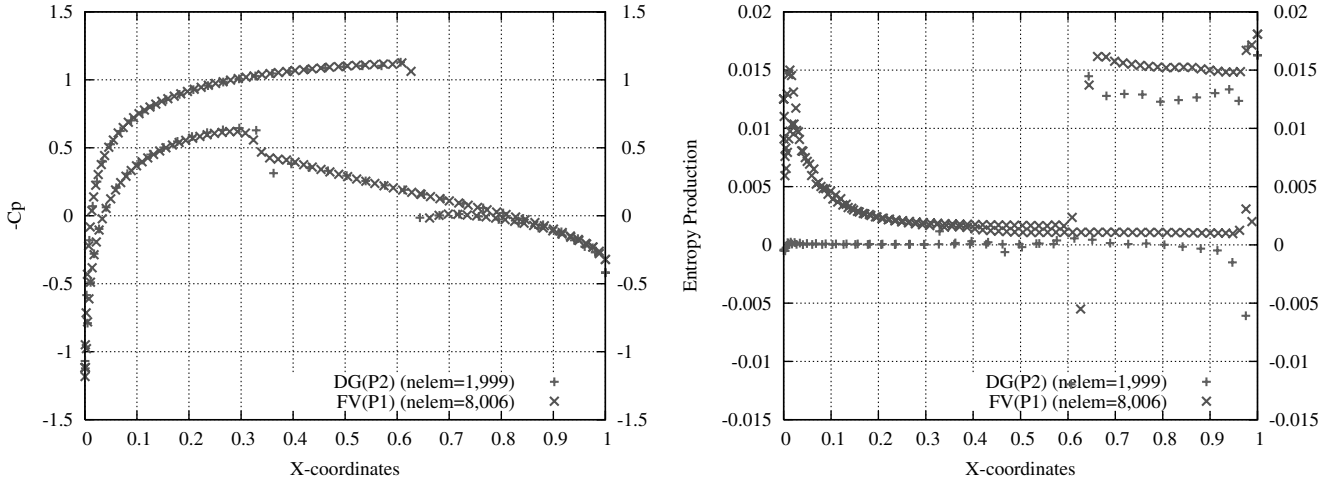

Fig. 19 Comparison of computed pressure coefficient (left) and entropy production (right) distributions on the surface of airfoil obtained by the FV(P1) and DG(P2) computations for transonic flow past a NACA0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

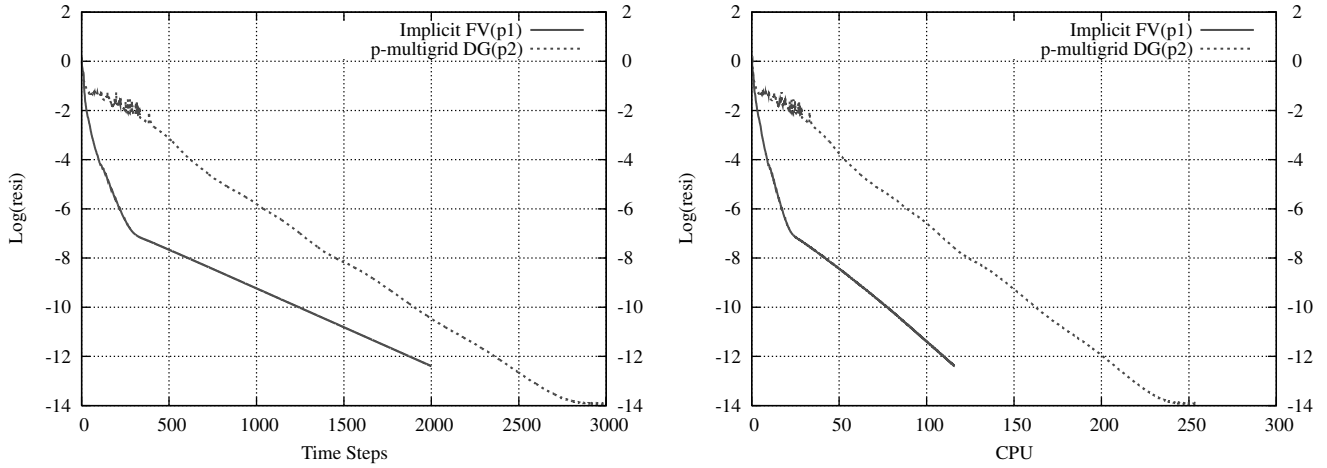

Fig. 20 Comparison of convergence history versus time steps (left) and CPU time (right) between the FV(P1) and p-multigrid DG(P2) methods for transonic flow past a NACA0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

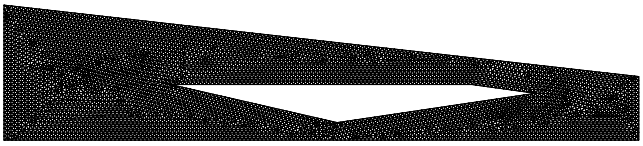

Fig. 21 Unstructured mesh used for computing supersonic flow inside an inlet (nelem = 7993, npoin = 4276, nboun = 559).

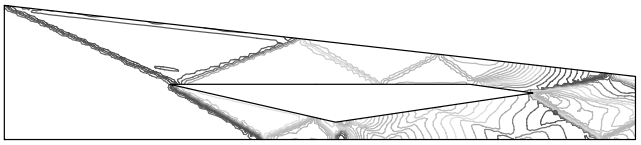

Fig. 24 Computed Mach number contours by the DG(P2) method for supersonic flow inside an inlet at $M_\infty = 3$.

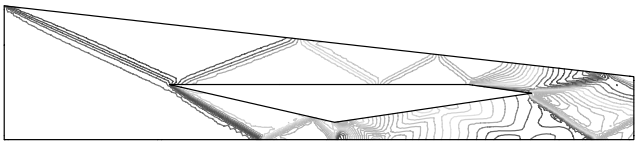

Fig. 22 Computed Mach number contours by the FV(P1) method for supersonic flow inside an inlet at $M_\infty = 3$.

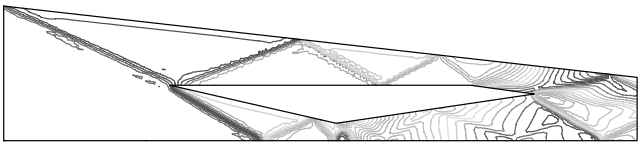

Fig. 23 Computed Mach number contours by the DG(P1) method for supersonic flow inside an inlet at $M_\infty = 3$.

deteriorates drastically when the number of elements and, especially, the polynomial order increase.

Figures 30–32 show a comparison of convergence histories versus time steps among the P0, P1, and P2 computations using the p-multigrid method on the coarse, medium, and fine grids, respectively. The p-multigrid method obtains the convergence in nearly the same number of time steps for both P1 and P2 solutions for all three grids, demonstrating the order independence of this p-multigrid.

Figure 33 displays a comparison of convergence histories versus time steps and CPU time for the P1 element approximation between the TVDRKDG and p-multigrid methods on the medium mesh, and Fig. 34 displays the same comparison for P2 element computations. The p-multigrid method is about 20 times faster for P1 computation and about 100 times faster for P2 computation than its explicit TVDRK counterpart.

Figure 35 displays a comparison of convergence histories versus time steps and CPU time for DG(P1) solutions on the fine mesh and DG(P2) solutions on the medium mesh. The DG(P2) solution on the

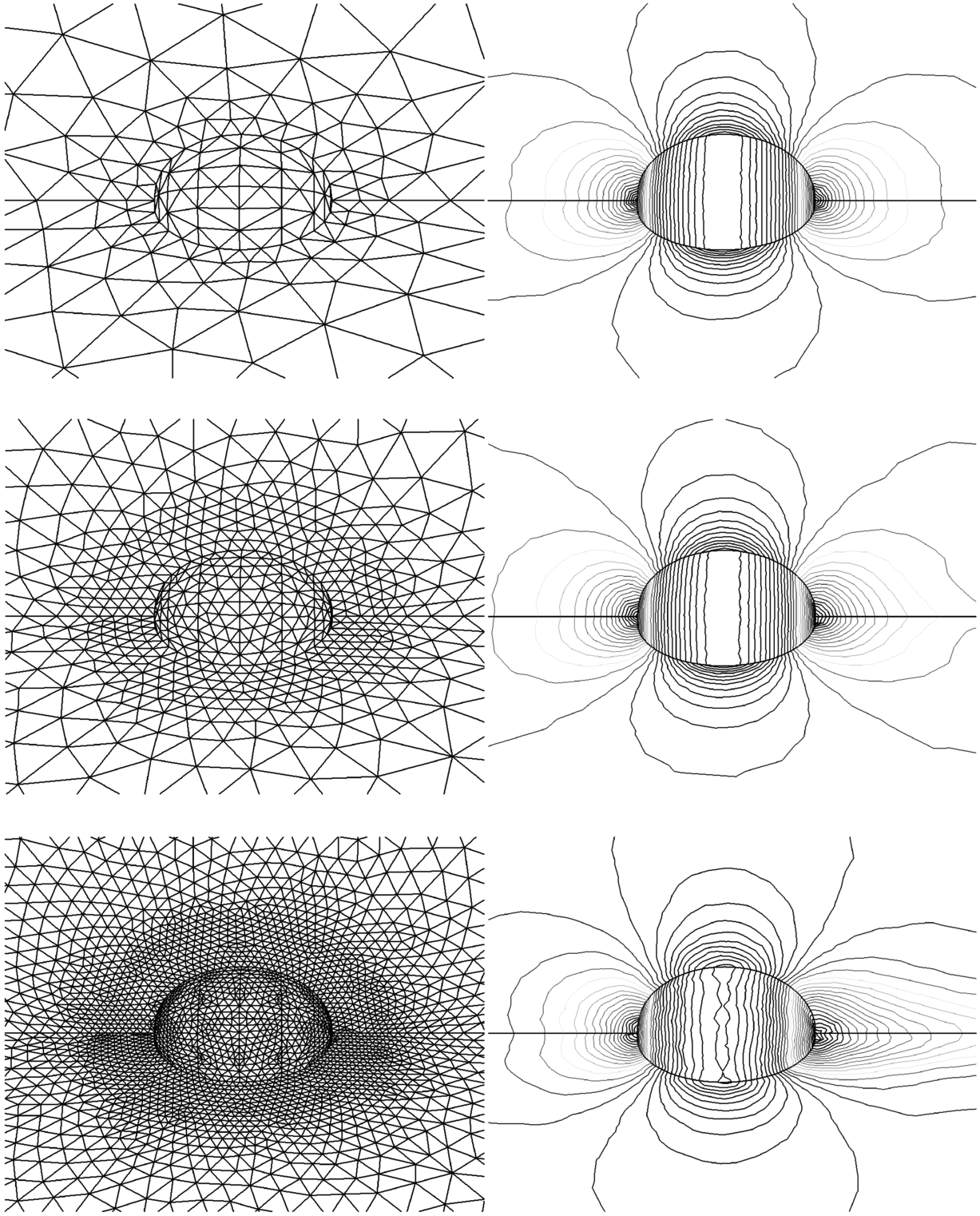

Fig. 25 Sequences of three globally refined unstructured surface meshes used for computing subsonic flow past a sphere (left) and computed Mach number contours obtained by DG(P2) (right) on the coarse mesh (top), by DG(P1) on the medium mesh (middle), and DG(P0) on the fine mesh (bottom) for computing subsonic flow past a sphere $M_\infty = 0.5$.

medium mesh converges almost 10 times faster than the DG(P1) solution on the fine mesh. Keep in mind that the former actually yields a more accurate solution than the latter, clearly indicating the advantages of using higher-order approximations.

F. Subsonic Flow in a Channel with a Circular Bump on the Lower Wall

This is the well-known Ni's test case: a subsonic flow in a channel with a 10%-thick circular bump on the bottom. This test case is chosen to test the performance of the p-multigrid for computing internal flows. The length of the channel is 3, its height is 1, and its width is 0.5. The computation is performed at an inlet Mach number of 0.5. This is a three-dimensional simulation of a two-dimensional flow problem. The mesh, which contains 16,266 elements, 3650 grid points, and 1665 boundary points, is depicted in Fig. 36. The computed Mach number contours obtained by the DG(P0), DG(P1), and DG(P2) solutions in the flowfield are shown in Figs. 37–39, respectively. Figure 40 shows a comparison of convergence histories

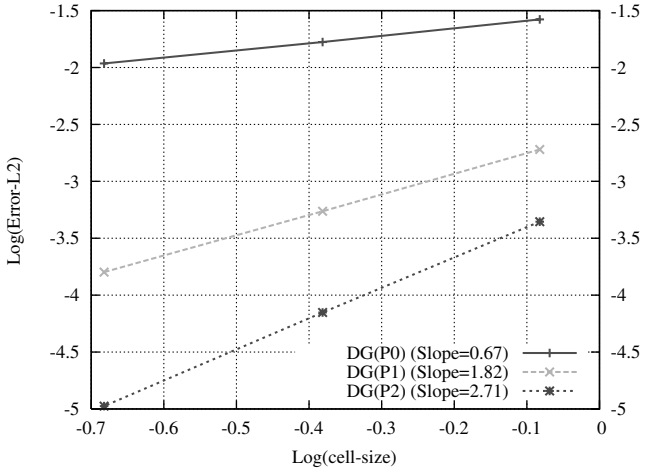

Fig. 26 Accuracy summary for subsonic flow past a sphere for DG(P0), DG(P1), and DG(P2) computations.

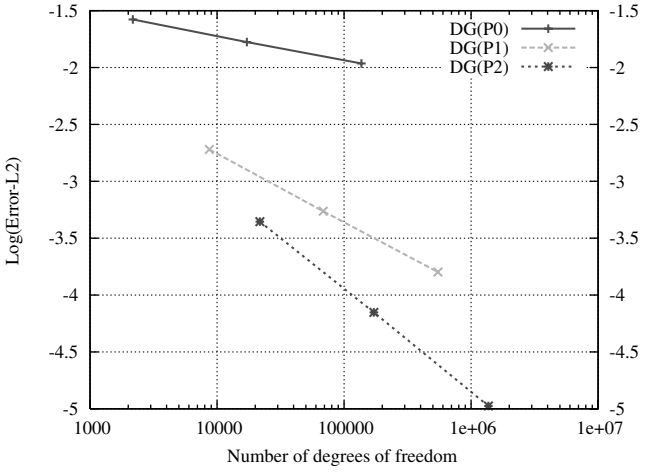

Fig. 27 L^2-error of numerical solutions against the number of degrees of freedom for subsonic flow past a sphere by the DG(P0), DG(P1), and DG(P2) methods.

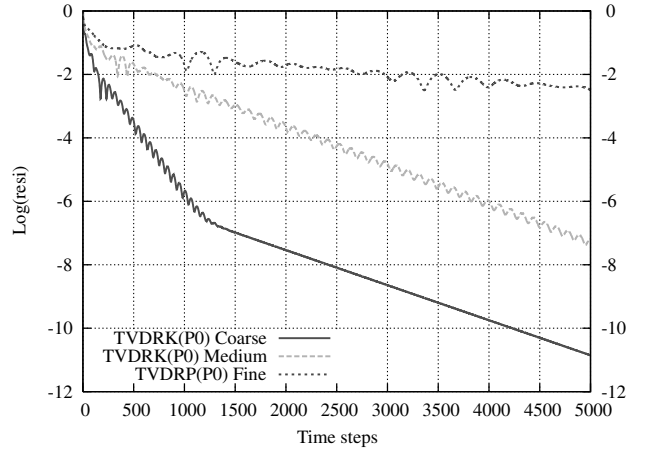

Fig. 28 Comparison of convergence history versus time steps for coarse, medium, and fine meshes using TVDRK method for subsonic flow past a sphere at $M_\infty = 0.5$.

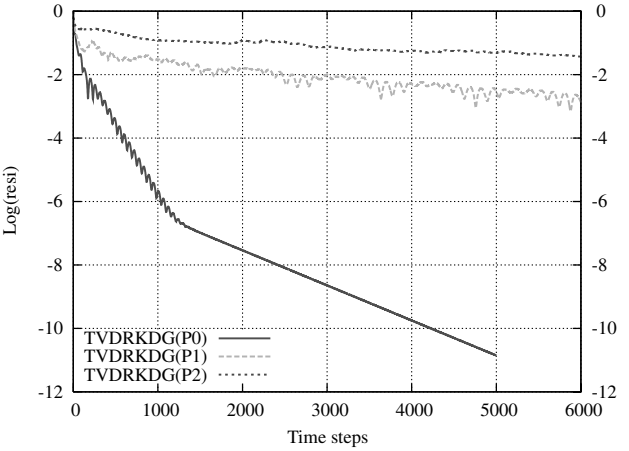

Fig. 29 Comparison of convergence history versus time steps for DG (P0), DG(P1), and DG(P2) on the coarse mesh using TVDRK method for subsonic flow past a sphere at $M_\infty = 0.5$.

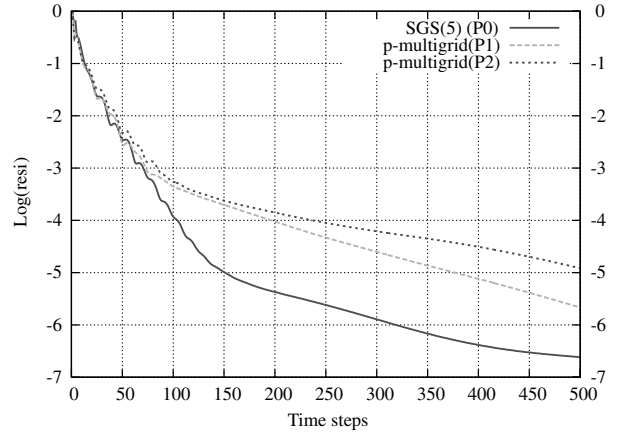

Fig. 30 Comparison of convergence history versus time steps for DG (P0), DG(P1), and DG(P2) on the coarse mesh using the *p*-multigrid method for subsonic flow past a sphere at $M_\infty = 0.5$.

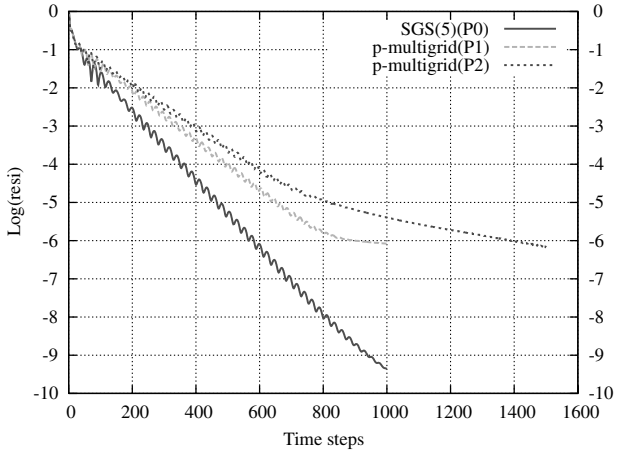

Fig. 31 Comparison of convergence history versus time steps for DG (P0), DG(P1), and DG(P2) on the medium mesh using the *p*-multigrid method for subsonic flow past a sphere at $M_\infty = 0.5$.

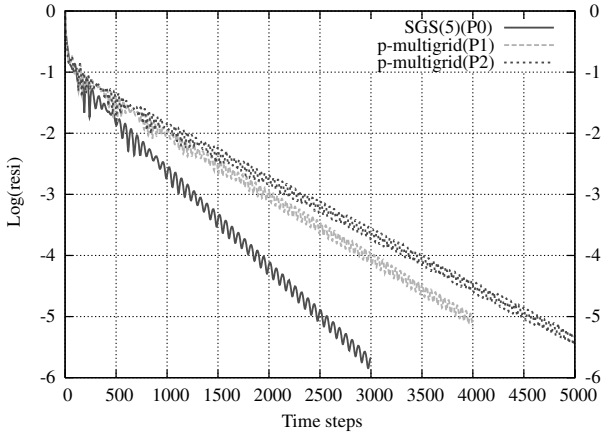

Fig. 32 Comparison of convergence history versus time steps for DG (P0), DG(P1), and DG(P2) on the fine mesh using the p-multigrid method for subsonic flow past a sphere at $M_\infty = 0.5$.

versus time steps among the P0, P1, and P2 computations using the p-multigrid method, in which one can see that the p-multigrid method obtains the convergence in nearly the same number of time steps for both P1 and P2 solutions, again demonstrating the order independence of this p-multigrid for this test case. Figures 41 and 42 display a comparison of convergence histories versus time steps and CPU time for P1 computation between TVDRKDG and p-multigrid methods. The p-multigrid method is more than 20 times faster than its explicit TVDRK counterpart for this test case. The excellent acceleration of the p-multigrid method is again demonstrated for this internal flow problem.

VI. Conclusions

The discontinuous Galerkin methods are recognized as expensive in terms of both computational costs and storage requirements. Indeed, this is true for second-order solutions. As an example, we compare the computing cost between cell-centered FV(P1) and DG (P1) methods for a tetrahedral mesh. We will only count the number of Riemann fluxes required to evaluate for each method, because it represents the most dominate CPU time-consuming operations. Let nelem be the number of elements (tetrahedra) and nbfac be the number of boundary faces. The number of total faces nface $=$ (4nelem $+$ nbfac)$/2$. FV(P1) must evaluate nface Riemann fluxes, whereas that number for DG(P1) is 4nface, because four quadrature points are used to compute the boundary integrals on each face. Even though DG(P1) yields a more accurate solution than FV (P1), it is still hard to justify three times more computation time. However, if we accept as a fact that DG(P2) on a given mesh (cell size of h) yields the same solution (actually, more accurate) as FV(P1) on a eight-times-larger mesh (cell size of $h/2$), the DG methods, regarded notoriously as expensive, make a comeback. When a tetrahedron is subdivided into eight tetrahedra, the number of total

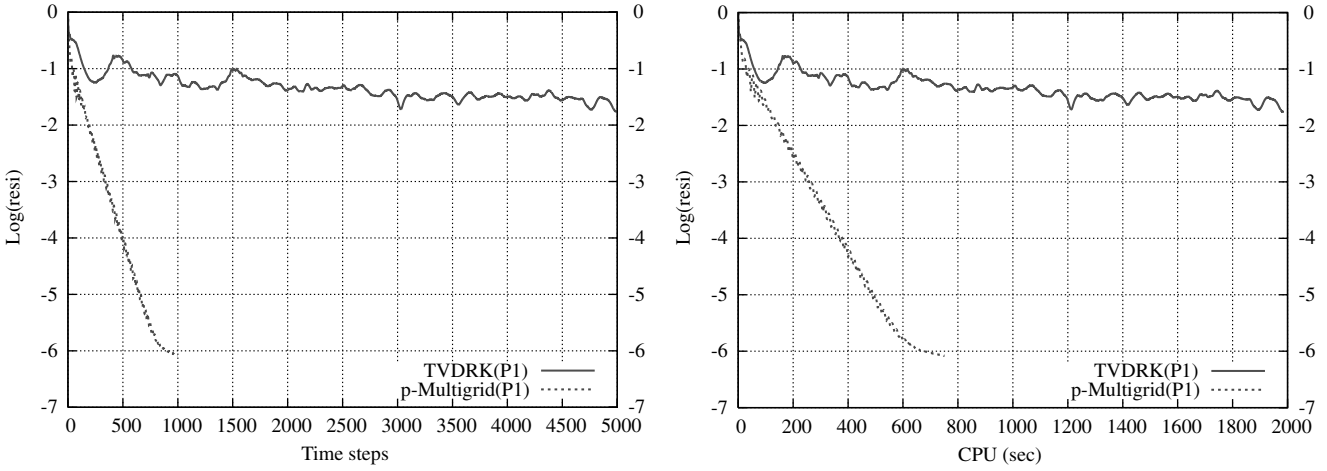

Fig. 33 Comparison of convergence history versus time steps (left) and CPU time (right) between TVBRK and p-multigrid methods for DG(P1) computation on the medium mesh for subsonic flow past a sphere at $M_\infty = 0.5$.

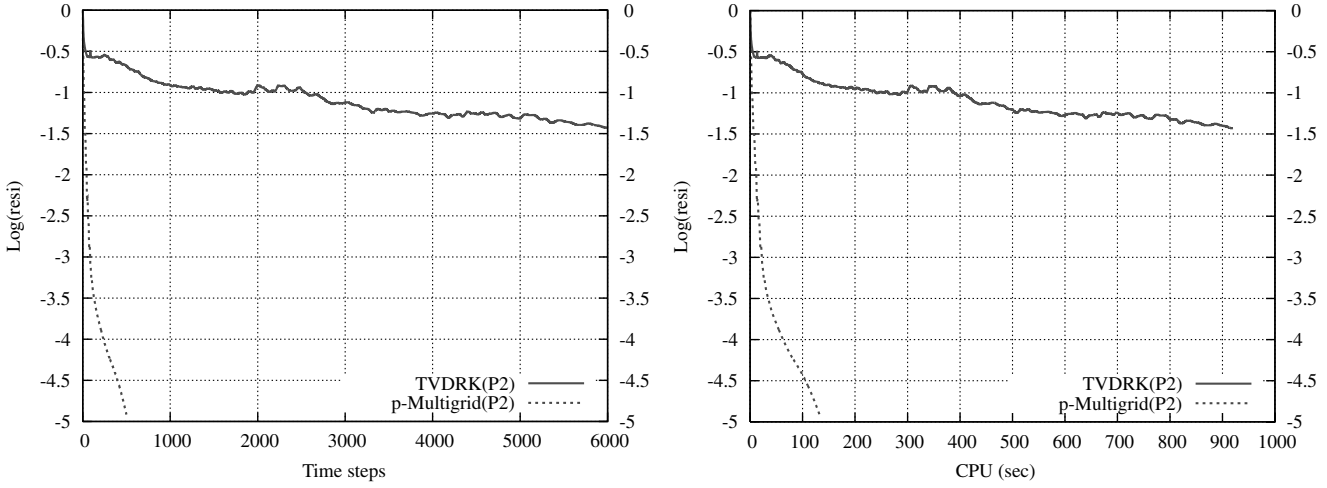

Fig. 34 Comparison of convergence history versus time steps (left) and CPU time (right) between TVBRK and p-multigrid methods using DG(P2) approximation on the coarse mesh for subsonic flow past a sphere at $M_\infty = 0.5$.

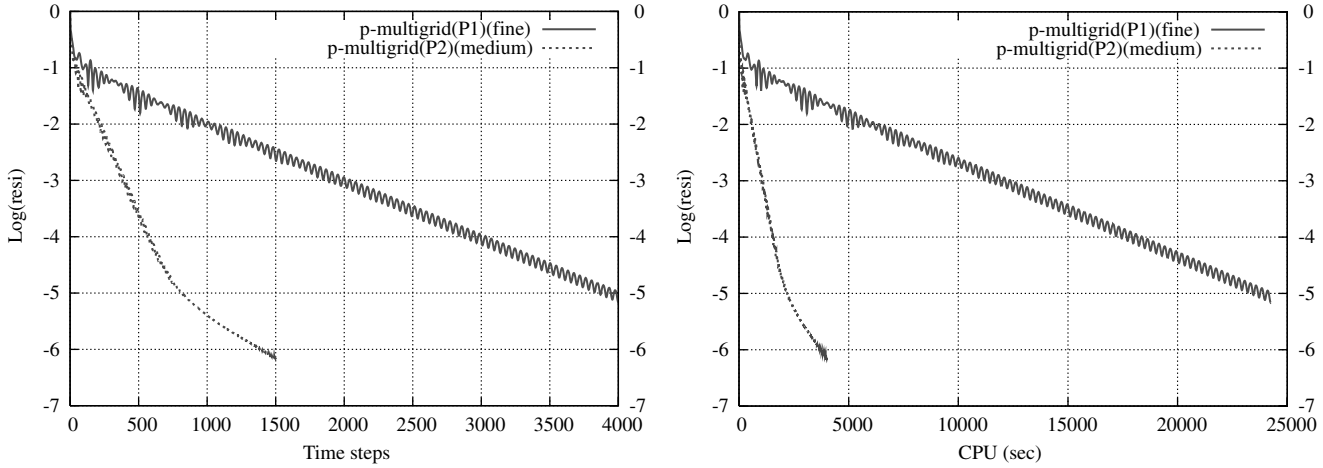

Fig. 35 Comparison of convergence history versus time steps (left) and CPU time (right) between DG(P1) approximation on the fine mesh and DG(P2) approximation on the medium mesh using the *p*-multigrid method for subsonic flow past a sphere at $M_\infty = 0.5$.

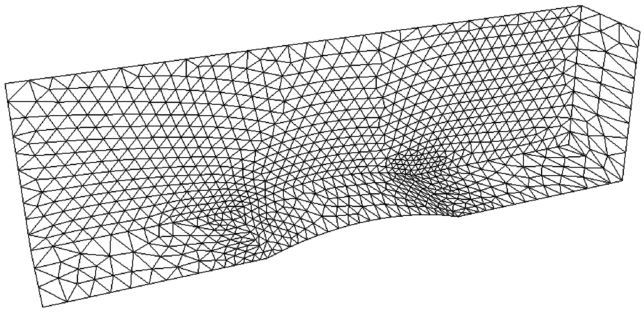

Fig. 36 Unstructured mesh used for computing a subsonic flow in a channel with a 10%-thick circular bump on the bottom (nelem = 16, 266, npoin = 3650, and nboun = 1665).

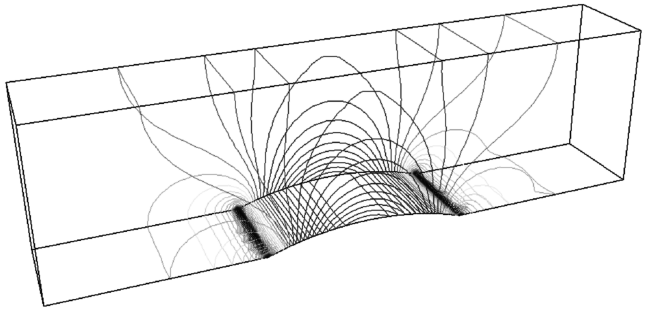

Fig. 39 Computed Mach number contours obtained by the DG(P2) methods for subsonic flow in a channel with a 10%-thick circular bump on the bottom at $M_\infty = 0.5$.

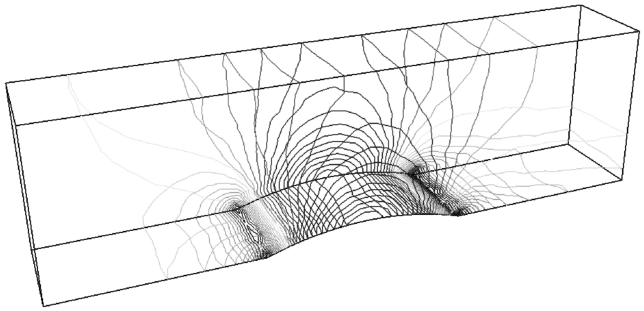

Fig. 37 Computed Mach number contours obtained by the DG(P0) methods for subsonic flow in a channel with a 10%-thick circular bump on the bottom at $M_\infty = 0.5$.

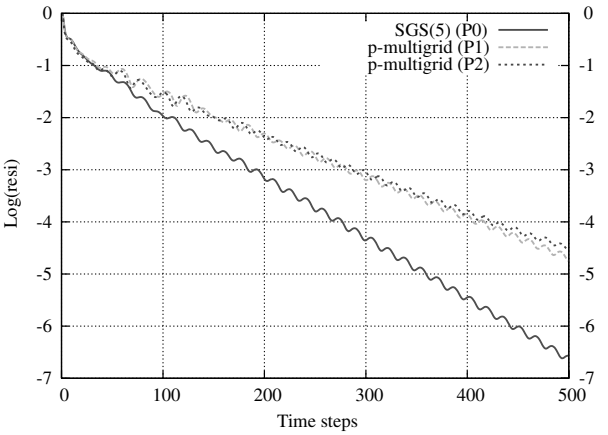

Fig. 40 Comparison of convergence history versus time steps for DG (P0), DG(P1), and DG(P2) using the *p*-multigrid method for subsonic flow in a channel with a 10%-thick circular bump on the bottom.

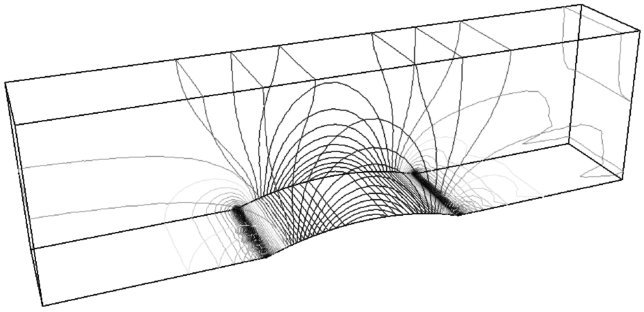

Fig. 38 Computed Mach number contours obtained by the DG(P1) methods for subsonic flow in a channel with a 10%-thick circular bump on the bottom at $M_\infty = 0.5$.

faces is 16nelem + 2nbfac, which is also the number of Riemann fluxes to be evaluated for FV(P1), whereas the number of Riemann fluxes to be evaluated for DG(P2) is 14nelem + 3.5nbfac, in which seven quadrature points are used in the interelement boundary faces, meaning that the DG(P2) method actually requires less computational effort than the FV(P1) method.

A *p*-multigrid algorithm for the discontinuous Galerkin finite element method was presented for solving the compressible Euler equations on unstructured grids. This *p*-multigrid algorithm uses an explicit smoother (TVDRK) on the higher-level approximations ($p > 0$) to significantly reduce the prohibitive memory requirements

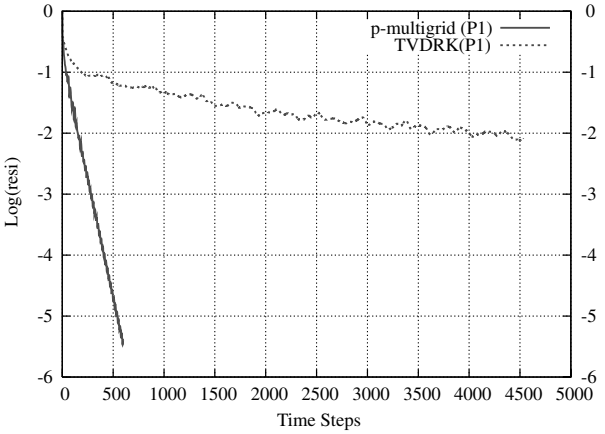

Fig. 41 Comparison of convergence history versus time steps between TVBRK and *p*-multigrid methods for DG(P1) computation for subsonic flow in a channel with a 10%-thick circular bump on the bottom.

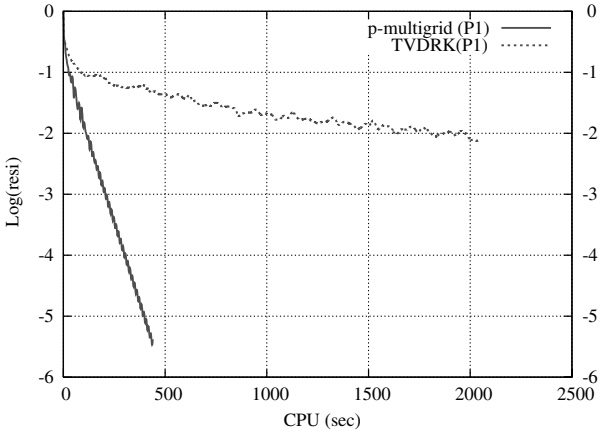

Fig. 42 Comparison of convergence history versus CPU time between TVBRK and *p*-multigrid methods for DG(P1) computation for subsonic flow in a channel with a 10%-thick circular bump on the bottom.

and computing costs for the Jacobian matrix commonly required for any implicit smoothers and uses an implicit smoother (SGS) on the lowest-level approximation ($p = 0$) to effectively eliminate lowest-frequency error modes. Thus, the p-multigrid method can be naturally applied to compute the flows with discontinuities, in which a monotonic limiting procedure is usually required for discontinuous Galerkin methods. An accurate representation of the boundary normals based on the analytically defined boundary geometries is used for imposing slip boundary conditions for curved geometries, which avoids the use of isoparametric elements to represent curved geometries and leads to a huge simplification of the boundary-condition implementation, as well as tremendous savings in both storage requirements and computing costs.

The p-multigrid was used to compute a variety of compressible flow problems for a wide range of flow conditions from low Mach number to supersonic in both 2D and 3D configurations. The numerical results obtained strongly indicate the order-independent property of this p-multigrid method. The overall performance of this novel p-multigrid is orders-of-magnitude better than its explicit counterpart without significant increase in memory. Using this acceleration method, the discontinuous Galerkin methods provide a viable, attractive, and competitive alternative to the traditional finite volume, finite element, and finite difference methods for computing compressible flows at all speeds, not only in terms of other advantages provided by DG methods, but also in terms of computing costs. Future work will explore the application of this method for the solution of the Navier–Stokes equations.

References

[1] Cockburn, B., Hou, S., and Shu, C. W., "TVD Runge–Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws 4: The Multidimensional Case," *Mathematics of Computation*, Vol. 54, 1990, pp. 545–581. doi:10.2307/2008501

[2] Lin, S. Y., and Chin, Y. S., "Discontinuous Galerkin Finite Element Method for Euler and Navier–Stokes Equations," *AIAA Journal*, Vol. 31, No. 11, 1993, pp. 2016–2023.

[3] Biswas, R., Devine, K. D., and Flaherty, J., "Parallel, Adaptive Finite Element Methods for Conservation Laws," *Applied Numerical Mathematics*, Vol. 14, Nos. 1–3, Apr. 1994, pp. 255–284. doi:10.1016/0168-9274(94)90029-9

[4] Bey, K. S., Oden, J. T., and Patra, A., "A Parallel hp-Adaptive Discontinuous Galerkin Method for Hyperbolic Conservation Laws," *Applied Numerical Mathematics*, Vol. 20, No. 4, 1996, pp. 321–336. doi:10.1016/0168-9274(95)00101-8

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

[6] Tu, S., and Aliabadi, S., "A Slope Limiting Procedure in Discontinuous Galerkin Finite Element Method for Gasdynamics Applications," *International Journal of Numerical Analysis and Modeling*, Vol. 2, No. 2, 2005, pp. 163–178.

[7] Cockburn, B., and Shu, C. W., "The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws 5: Multidimensional System," *Journal of Computational Physics*, Vol. 141, No. 2, 1998, pp. 199–224. doi:10.1006/jcph.1998.5892

[8] Bassi, F., and Rebay, S., "High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations," *Journal of Computational Physics*, Vol. 138, No. 2, 1997, pp. 251–285. doi:10.1006/jcph.1997.5454

[9] Bassi, F., and Rebay, S., "A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier–Stokes Equations," *Journal of Computational Physics*, Vol. 131, No. 2, 1997, pp. 267–279. doi:10.1006/jcph.1996.5572

[10] van der Vegt, J. J. W., and van der Ven, H., "Discontinuous Galerkin Finite Element Method with Anisotropic Local Grid Refinement for Inviscid Compressible Flows," *Journal of Computational Physics*, Vol. 141, No. 1, 1998, pp. 46–77. doi:10.1006/jcph.1998.5904

[11] Remaki, M., Habashi, W. G., Ait-Ali-Yahia, D., and Jay, A., "A 3D Discontinuous Galerkin Method for Multiple Pure Tone Noise Problems," AIAA Paper AIAA-2002-0229, 2002.

[12] Bassi, F., and Rebay, S., "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, Vol. 11, Lecture Notes in Computational Science and Engineering, Springer–Verlag, New York, 2000, pp. 197–208.

[13] Rasetarinera, P., and Hussaini, M. Y., "An Efficient Implicit Discontinuous Spectral Galerkin Method," *Journal of Computational Physics*, Vol. 172, No. 2, 2001, pp. 718–738. doi:10.1006/jcph.2001.6853

[14] Rönquist, E. M., and Patera, A. T., "Spectral Element Multigrid, Part 1: Formulation and Numerical Results," *Journal of Scientific Computing*, Vol. 2, No. 4, 1987, pp. 389–406. doi:10.1007/BF01061297

[15] Maday, Y., and Munoz, R., "Spectral Element Multigrid, Part 2: Theoretical Justification," Inst. of Control, Automatic and Systems Engineering, TR 88-73, 1988.

[16] Bassi, F., and Rebay, S., "Numerical Solution of the Euler Equations with a Multiorder Discontinuous Finite Element Method," *Computational Fluid Dynamics 2002*, Springer, New York, July 2002, pp. 15–19.

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

[18] Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L., "*p*-Multigrid Solution of High-Order Discontinuous Galerkin Discretizations of the Compressible Navier–Stokes Equations," *Journal of Computational Physics*, Vol. 207, No. 1, 2005, pp. 92–113. doi:10.1016/j.jcp.2005.01.005

[19] Luo, H., Baum, J. D., and Löhner, R., "A *p*-Multigrid Discontinuous Galerkin Method for the Euler Equations on Unstructured Grids," *Journal of Computational Physics*, Vol. 211, No. 2, 2006, pp. 767–783. doi:10.1016/j.jcp.2005.06.019

[20] Toro, E. F., Spruce, M., and Speares, W., "Restoration of the Contact Surface in the HLL-Riemann Solver," *Shock Waves*, Vol. 4, 1994, pp. 25–34. doi:10.1007/BF01414629

[21] Batten, P., Leschziner, M. A., and Goldberg, U. C., "Average-State Jacobians and Implicit Methods for Compressible Viscous and Turbulent Flows," *Journal of Computational Physics*, Vol. 137, No. 1, 1997, pp. 38–78. doi:10.1006/jcph.1997.5793

[22] Luo, H., Baum, J. D., and Löhner, R., "High-Reynolds Number Viscous Flow Computations Using an Unstructured-Grid Method," *Journal of Aircraft*, Vol. 42, No. 2, 2005, pp. 483–492.

[23] Krivodonova, L., and Berger, M., "High-Order Implementation of Solid Wall Boundary Conditions in Curved Geometries," *Journal of Computational Physics*, Vol. 211, No. 2, 2006, pp. 492–512. doi:10.1016/j.jcp.2005.05.029

[24] Barth, T., and Jesperson, D., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper AIAA-1989-0366, 1989.

[25] Luo, H., Baum, J. D., and Löhner, R., "On the Computation of Steady-State Compressible Flows Using a Discontinuous Galerkin Method," *International Journal for Numerical Methods in Engineering*, Vol. 73, No. 5, 2008, pp. 597–623.doi:10.1002/nme.2081

[26] Krivodonova, L., Xin, J., Remacle, J. F., Chevaugeon, N., and Flaherty, J. E., "Shock Detection and Limiting with Discontinuous Galerkin Methods for Hyperbolic Conservation Laws," *Applied Numerical Mathematics*, Vol. 48, Nos. 3–4, Apr. 2004, pp. 323–338. doi:10.1016/j.apnum.2003.11.002

[27] Luo, H., Sharov, D., Baum, J. D., and Löhner, R., "A Class of Matrix-Free Implicit Methods for Compressible Flows on Unstructured Grids," *Computational Fluid Dynamics 2000*, Springer, New York, July 2000, pp. 10–14.

[28] Luo, H., Baum, J. D., and Löhner, R., "A Fast, Matrix-Free Implicit Method for Compressible Flows on Unstructured Grids," *Journal of Computational Physics*, Vol. 146, No. 2, 1998, pp. 664–690. doi:10.1006/jcph.1998.6076

[29] Jameson, A., and Yoon, S., "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations," *AIAA Journal*, Vol. 25, No. 7, 1987, pp. 929–935.

Z. Wang
Associate Editor