A PARALLEL DISCONTINUOUS GALERKIN CODE FOR COMPRESSIBLE FLUID FLOWS ON UNSTRUCTURED GRIDS

Amjad Ali[1], Hong Luo[1], Khalid Saifullah Syed[2], Muhammad Ishaq[2]

ABSTRACT

A parallel discontinuous Galerkin code based on a Taylor series basis for the compressible Euler equations on unstructured meshes is presented. The parallel numerical method is capable to efficiently simulate the flow physics of the problems considered, including subsonic and transonic compressible inviscid fluid flows around two well known benchmark airfoils. The parallel discontinuous Galerkin method is presented for distributed memory parallel computing architectures, specifically considered cost effective compute clusters, composed of mass-market-commodity-off-the-shelf (M^2COTS) computer hardware components interconnected via Ethernet. The parallelization is based on computational domain partitioning, making use of the well-known graph and mesh partitioning package METIS. Single Program Multiple Data (SPMD) message passing parallel programming model has been employed, by making use of the de-facto industry standard Message Passing Interface (MPI) library. Favorable parallelization characteristics of the discontinuous Galerkin scheme have been exploited for hiding the communications behind the computations. The parallel performance of the developed code on an Ethernet based cluster, in terms of speedup is demonstrated.

Keywords: *Parallel, Discontinuous Galerkin Method, Compressible, Euler equations*

INTRODUCTION

Computational Fluid Dynamics (CFD) is apprehended as an effective tool for performing numerical simulations in applied sciences and engineering for problems related to fluid flow phenomena. CFD serves in reducing, and some times completely eliminating, a large number of expensive laboratory or physical experiments while designing and developing a new product. Most CFD problems of practical interest involve one or more partial differential equations governing some fluid flow phenomena subject to some assumed or measured boundary conditions, with the physical geometry being approximated by structured/unstructured grids. These problems require the use of some numerical methods for their solution. Often such a numerical method starts with converting the partial differential equations of the problem into a discretized form using a discretization method, usually a Finite Volume Method (FVM), Finite Element Method (FEM), Finite Difference Method (FDM) or a Hybrid FEM/FVM/FDM method like Discontinuous Galerkin (DG) Method. The discretized form is integrated in time explicitly or implicitly for obtaining a steady state solution.

The discontinuous Galerkin (DG) methods are emerging as a new class of methods in the field of numerical solution of partial differential equations representing conservation laws[1-13]. These methods are gaining popularity in solving the problems of fluid dynamics[11-19], acoustics[6,20], electromagnetism[4], etc. These methods enjoy the advantage of Finite Element Methods of high order accuracy and that of Finite Volume Methods of conserving field variables[1,4,13]. Therefore they have been successfully employed in the simulation of wave phenomena by solving Riemann problems at element interfaces arising from discontinuous nature of the solution at the interfaces. Growing popularity of these methods owes to its attractive features including: 1) These methods have good conservation, stability, and convergence properties; 2) Order of approximation may be extended as desired; 3) They are applicable on arbitrary grids (conforming/non-conforming/hybrid), thus having the ability of dealing with complex rather arbitrarily shaped geometries; 4) Since the polynomial approximation is taken independently on each element with minimal inter-element communication, they can be parallelized easily; 5) adaptive strategies using *hp*-refinement (i.e.,

[1] *North Carolina State University, Raleigh, NC, 27695, USA.*

[2] *Centre for Advanced Studies in Pure and Applied Mathematics, Bahauddin Zakariya University, Multan 60800, Pakistan.*

refining the grid by decreasing element sizes, *h*-refinement, and increasing order of approximating polynomials, *p*-refinement) can easily be integrated into these methods because of the removal of continuity restriction at element interfaces. In spite of all these features and the fact that there have been considerable developments in theoretical and numerical analysis of these methods, there are certain limitations in these methods that need to be addressed to make them a competent choice for fluid flow problems of practical and industrial interest. These limitations include efficient discretization of diffusion terms, effective control of blow up because of strong discontinuities, and development of efficient and accurate time integrators. Another main hurdle in adopting these methods for practical engineering applications is the lack of efficient solvers because of its high CPU and memory requirements[13].

Recently, a new DG formulation based on a Taylor series basis has been demonstrated to successfully approximate the solution of compressible inviscid fluid flow equations on arbitrary grids[13]. Unlike the traditional DG methods, where either standard Lagrange finite element or hierarchical node-based basis functions are used to represent numerical polynomial solutions in each element, this DG method represents the numerical polynomial solutions using a Taylor series expansion at the centroid of the cell. This way, the new formulation has several distinct, desirable, and attractive features including:

1) It has the ability to be implemented on arbitrary grids because of same polynomial solution for any shape of elements.

2) In it the hierarchic nature of the basis function makes it easy to implement *p*-refinement and *p*-multigrid strategies.

3) In it the availability of cell-averaged variables and their derivates facilitates implementation of very efficient limiters for eliminating the oscillations in the solution around the discontinuities.

4) In it the decoupling of cell-averaged variable equations from their derivates facilitates development of fast low storage implicit schemes.

These features can help in overcoming some of the disadvantages of the DG method.

Along with developing efficient numerical algorithms for a CFD application, an important need is to implement the complete application in some high performance computing (HPC) environment. In fact, a key factor for enormous growth in CFD applicability has been the rapid developments in HPC capabilities during the past two decades. The most commonly used and recognized form of obtaining a HPC based solution is to perform parallelization of the application, so that a number of processing elements (or simply processors) work together on different parts of the problem or domain to reduce the overall time required to solve the problem. From hardware perspective, HPC may be realized on a variety of distributed memory architectures, shared memory architectures, and hybrid architectures. From software perspective, HPC rely on utilizing efficient compilers, fast mathematical kernels and sophisticated parallelization libraries for an underlying architecture. Further, tuning the application with reference to an underlying architecture may also result in very fast solutions.

In this work an efficient parallel CFD solver is presented in which the discontinuous Galerkin method based on a Taylor basis[13] is used as the discretization method for the Euler equations governing compressible fluid flow phenomena. The explicit time integration solution is obtained using a Range-Kutta (RK) method for up to third order of approximation (P2 approximation). For parallelization of the discontinuous Galerkin method, domain decomposition approach with SPMD (Single Program Multiple Data) message-passing programming model is employed. For this, Message Passing Interface (MPI) library is used. MPI has emerged as a de-facto standard for portable and scalable parallel programming for distributed memory parallel architectures. Several free and commercial MPI implementations are available. These implementations include both general and system/vendor specific. PC Clusters, sometimes referred to as Beowulf clusters, offer a very cost effective solution for distributed memory, high performance parallel computing. They are established using commodity off-the-shelf (COTS) hardware components and interconnected via Ethernet switch[21,22]. In this paper, the accuracy and performance of the developed MPI based parallel solution have been demonstrated on such a cluster. For very fast inter-processor communication a specialized network technology, like Infiniband or Myrinet, may also

be deployed. The rest of this paper is structured as follows: The governing equations are described in Section II. The numerical method involving DG method based on a Taylor basis[13] is described in Section III. Parallelization of the method is described in Section IV. Numerical results are presented in Section V. The conclusions are given in Section VI.

GOVERNING EQUATIONS

The unsteady compressible flow of inviscid fluid is governed by a system of partial differential equations known as Euler equations. For a two dimensional space, the conservation form of this system in vector notation is given by,

$$\frac{\partial U(x,t)}{\partial t}+\frac{\partial F_1(U(x,t))}{\partial x_1}+\frac{\partial F_2(U(x,t))}{\partial x_2}=0,$$
$$\forall x=(x_1,x_2)\in\Omega, t\in(0,+\infty$$

Here Ω is a bounded connected domain in the two dimensional real space, $\mathbf{R}^2=\{(x_1,x_2)|x_1,x_2\in(-\infty,+\infty)\}$. The conservative variable vector $\mathbf{U}$, and the flux vectors $\mathbf{F}_1$ and $\mathbf{F}_2$ are given by,

$$U=\begin{pmatrix}\rho\\ \rho u_1\\ \rho u_2\\ \rho e\end{pmatrix}, F_1=\begin{pmatrix}\rho u_1\\ \rho_1^2+p\\ \rho u_1u_2\\ u_1(\rho e+p)\end{pmatrix}, F_2=\begin{pmatrix}\rho u_2\\ \rho u_1u_2\\ \rho_2^2+p\\ u_2(\rho e+p)\end{pmatrix}.$$

Here ρ, e and p denote the density, specific total energy and pressure of the fluid, respectively. u_1 and u_2 are the velocity components of the flow in the coordinate directions. In order to obtain closed form of the system, the equation of state, $p=(\gamma-1)\rho(e-\frac{1}{2}(u_1^2+u_2^2))$, is added into the system. Here γ is the ratio of the specific heats. This equation of state is valid for perfect gas.

THE NUMERICAL METHOD

Discontinuous Galerkin Formulation

The discontinuous spatial discretization of the Euler equations based on a Taylor series basis is the same as defined in[13]. To formulate the DG method used, first the weak formulation is obtained. From this point onward summation notation is used to express the equations. For obtaining the weak formulation the governing equation is multiplied by a test function $\mathbf{W}$, integrated over the domain Ω, and then integration by parts is performed. This yields the following:

$$\int_\Omega \frac{\partial U}{\partial t}Wd\Omega+\int_\Gamma F_jn_jd\Gamma-\int_\Omega F_j\frac{\partial W}{\partial x_j}d\Omega=0,\ \forall W\in V$$

where $\Gamma(=\partial\Omega)$ is the boundary of the domain Ω, and n_j the unit outward normal vector to the boundary. The domain Ω is subdivided into a collection of non-overlapping elements Ω_e. These elements or cells can be triangles, quadrilaterals, polygons, or their combinations for two dimensional domains. The space of test functions is taken to be the one consisting of discontinuous m-component vector-valued polynomials of degree p, i.e.,

$$V_p^h=\left\{v_h\in[L_2(\Omega)]^m : v_h\big|_{\Omega e}\in[V_p^m]\forall\Omega_\varepsilon\in\Omega\right\}$$

This can also be expressed as

$$V_p^h=span\left\{\prod_{j=1}^{2}x_j^{\alpha j}:0\le\alpha_j\le p,0\le j\le 2\right\},$$

where α denotes a multi-index. Application of the above weak formulation on each element Ω_e yields the following equivalent problem in semi-discrete form:

Find $U_h \in V^h_p$ such that

$$\begin{cases}\dfrac{d}{dt}\displaystyle\int_{\Omega e}\frac{\partial U_h}{\partial t}W_hd\Omega+\int_{\Gamma e}F_j(U_h)n_jW_hd\Gamma-\\ \displaystyle\int_{\Omega e}F_j(U_h)\frac{\partial W_h}{\partial x_j}d\Omega=0,\quad\forall W_h\in V_h^p.\end{cases}$$

Here $\mathbf{U}_h$ and $\mathbf{W}_h$ are piecewise polynomial approximations to the analytical solution $\mathbf{U}$ and the test function $\mathbf{W}$, respectively, from the space (V_p^h), defined above. These approximations are discontinuous across the element interfaces. Assume that $\boldsymbol{B}_i$ denotes the ith basis of the polynomial space, V_p^h of dimension N (say), then solving the above problem is equi-

valent to solving the following system of N equations:

$$\frac{d}{dt}\int_{\Omega e} U_h B_i d\Omega + \int_{\Gamma e} F_j(U_h) n_j B_i d\Gamma - \int_{\Omega e} F_j(U_h)\frac{\partial B_i}{\partial x_j} d\Omega = 0, \quad 1 \le i \le N.$$

The basis functions $\boldsymbol{B}_j$ are constructed by Taylor series expansion of the solution $\mathbf{U}_h$ at the centroid of an element. A quadratic approximation of $\mathbf{U}_h$, for example, can be expressed as:

$$U_h = U_c + \frac{\partial U}{\partial x}\Big|_c (x - x_c)\frac{\partial U}{\partial y}\Big|_c (y - y_c) + \frac{\partial^2 U}{\partial x^2}\Big|_c \frac{(x-x_c)^2}{2} + \frac{\partial^2 U}{\partial y^2}\Big|_c \frac{(y-y_c)^2}{2} + \frac{\partial^2 U}{\partial x \partial y^2}\Big|_c (x-x_c)(y-y_c),$$

Integrating the above equation over the element and then subtracting the resulting equation from the above equation, the following cell-averaged representation of $\mathbf{U}_h$ is obtained:

$$U_h = \bar{U} + \frac{\partial U}{\partial x}\Big|_c (x - x_c) + \frac{\partial U}{\partial y}\Big|_c (y - y_c) + \frac{\partial^2 U}{\partial y^2}\Big|_c \left(\frac{(x-x_c)^2}{2} - \frac{1}{\Omega_c}\int_{\Omega c}\frac{(x-x_c)^2}{2} d\Omega\right) + \frac{\partial^2 U}{\partial y^2}\Big|_c \left(\frac{(y-y_c)^2}{2} - \frac{1}{\Omega}\Big|_c \frac{(y-y_c)^2}{2} d\Omega\right) + \frac{\partial U}{\partial x \partial y}\Big|_c \left((x-x_c)(y-y_c) - \frac{1}{\Omega_c}\int_{\Omega c}(x-x_c)(y-y_c) d\Omega\right)$$

Here $\bar{U}$ is the cell-averaged value of $\mathbf{U}$. The unknown coefficients in the above representation are the cell-averaged conserved variables and their all first and second order partial derivatives at the cell-centroid. Therefore this representation of $\mathbf{U}_h$ is independent of element shapes, because the values are cell-averaged at the cell-centroid (instead of the values at the elemental nodes which are different for triangles and rectangles), as shown in Figure 1. The basis functions are the following six polynomials:

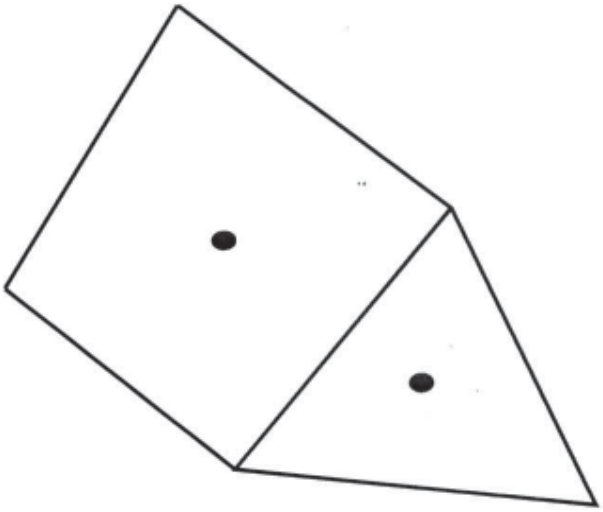

Figure 1: Representation of polynomial solutions using a Taylor series expansion at the cell centroid, hence, it is independent of the shape of element.

$$B_1 = 1$$
$$B_2 = (x - x_c)$$
$$B_3 = (y - y_c)$$
$$B_4 = \frac{(x-x_c)^2}{2} - \frac{1}{\Omega_c}\int_{\Omega c}\frac{(x-x_c)^2}{2} d\Omega$$
$$B_5 = \frac{(y-y_c)^2}{2} - \frac{1}{\Omega_c}\int_{\Omega c}\frac{(y-y_c)^2}{2} d\Omega$$
$$B_6 = (x-x_c)(y-y_c) - \frac{1}{\Omega}\int_{\Omega c}(x-x_c)(y-y_c) d\Omega.$$

Use of these basis functions in the DG formulation yields the following differential algebraic system of equations:

$$\frac{d}{dt}\int_{\Omega e}\bar{U} d\Omega + \int_{\Gamma e} F_j(U_h) n_j d\Gamma = 0, \quad i = 1$$

$$M_{5\times 5}\frac{d}{dt}\left(\frac{\partial U}{\partial x}\Big|_c \frac{\partial U}{\partial y}\Big|_c \frac{\partial^2 U}{\partial x^2}\Big|_c \frac{\partial^2 U}{\partial y^2}\Big|_c \frac{\partial^2 U}{\partial x \partial y}\Big|_c\right)^T + R_{5\times 1} = 0.$$

Here $M_{ji} = \int_{\Omega c} B_i B_j d\Omega, 2 \le i, j \le 6$, and $R_i = \int_{\Gamma e} F_j(U_h) n_j B_i d\Gamma - \int_{\Omega e} F_j(U_h)\frac{\partial B_i}{\partial x_j} d\Omega = 0, \ 2 \le i \le 6.$

Note that the basis functions have been constructed in such a way that

$$\int_{\Omega e} B_1 B_j d\Omega = 0, \ 2 \le j \le 6,$$

which results into decoupling of unknown $\bar{U}$ from all the unknown partial derivatives. For implementing the above computational procedure, normalization of the basis functions is carried out that improves the conditioning of the system matrix. The normalized basis functions are,

$$\hat{B}_1 = 1$$

$$\hat{B}_2 = \frac{(x - x_c)}{\delta x}$$

$$\hat{B}_3 = \frac{(y - y_c)}{\delta y}$$

$$\hat{B}_4 = \frac{(x - x_c)^2}{2\delta x^2} - \frac{1}{\Omega_e}\int_{\Omega e} \frac{(x - x_c)^2}{2\delta x^2} d\Omega$$

$$\hat{B}_5 = \frac{(y - y_c)^2}{2\delta y^2} - \frac{1}{\Omega_e}\int_{\Omega e} \frac{(y - y_c)^2}{2\delta y^2} d\Omega$$

$$\hat{B}_6 = \frac{(x - x_c)(y - y_c)}{\delta x \partial y} - \frac{1}{\Omega_e}\int_{\Omega e} \frac{(x - x_c)(y - y_c)}{\delta x \partial y} d\Omega.$$

Here $\delta x = 0.5\ (x_{max}\text{-}x_{min})$, and $\delta y = 0.5\ (y_{max}\text{-}y_{min})$, where x_{max} and y_{max}, x_{min}, and y_{min} denote the maximum and minimum coordinates in the cell W_e in x-direction and y-direction, respectively. Thus quadratic approximation of $\mathbf{U}_h$ becomes,

$$U_h - \bar{U} + \frac{\partial U}{\partial x}\Big|_c \delta x \hat{B}_2 + \frac{\partial U}{\partial x}\Big|_c \delta x \hat{B}_3 + \frac{\partial^2 U}{\partial x^2}$$
$$\Big|_c \delta x \hat{B}_4 + \frac{\partial^2 U}{\partial y^2}\Big|_c \delta x \hat{B}_5 + \frac{\partial^2 U}{\partial x \partial y^2}\Big|_c \delta x \delta y \hat{B}_6.$$

These normalized basis functions produce entries of mass matrix **M** with magnitude not greater than 1. This helps in removing the stiffness of the system matrix for higher order DG approximations[13].

This DG method is capable of accurately and efficiently computing numerical solution of the compressible inviscid flow equations on arbitrary grids[13]. In order to impose the solid wall boundary conditions on curved boundaries, an accurate representation of the boundary normals is required. A linear approximation of curved edges on the solid wall boundary leads to numerical instability and loss in the order of accuracy of DG methods. For removing these drawbacks the approach given in[18] is followed. In this approach, although the curved edges at the physical boundary are approximated by straight edges, the normals to these edges at the quadrature points are computed accurately using true geometry of the boundary. Further details about its implementation can be found in[18]. In this work, HLLC (Harten- Lax and Van Leer-Contact) approximate Riemann solver[23] is used to approximate the Riemann flux function. HLLC based solvers have been effectively used to compute compressible viscous and turbulent flows on both structured and unstructured grids[24,25]. HLLC Riemann solver is easier to implement, and the computational cost is lower than that of other available Riemann solvers[23-26].

In this work, Gauss quadrature formulas have been used to evaluate the domain and boundary integrals. The number of quadrature points used is chosen to integrate exactly polynomials of order of $2p$ on the reference element. In the case of triangular elements, the domain integrals are evaluated using three points for linear, six points for quadratic, and twelve points for cubic shape functions. In the case of rectangular elements, the domain integrals are evaluated using four points for linear, nine points for quadratic, and sixteen points for cubic shape functions. The boundary integrals are evaluated using two points for linear, three points for quadratic, and four points for cubic shape functions in 2D. By assembling together all the elemental contributions, a system of ordinary differential equations governing the evolution in time of the discrete solution is obtained, which can be written as

$$M \frac{dU}{dt} = R(U).$$

Here **M** represents the mass matrix, **U** represents the global vector of the degrees of freedom, and **R**(**U**) represents the right hand side residual vector. Since the shape functions $\boldsymbol{B}_i$ are nonzero within element W_e only, the mass matrix **M** has a block diagonal struc-

ture that couples the N degrees of freedom of each component of the unknown vector only within W_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. It is stable for any mesh to perform the inversion of the mass matrix[13].

Explicit Time Integration

The semi-discrete system can be integrated in time using explicit methods. In this work, the following explicit three stage third-order TVD Runge–Kutta method[1,2] is used to march the solution in time:

$$U^{(1)} \quad U^n \quad tM^{\ 1}R(U^n)$$

$$U^{(2)} \quad \frac{3}{4}U^n \quad \frac{1}{4}U^{(1)} \quad tM^{\ 1}R(U^{(1)})$$

$$U^{(n\ 1)} \quad \frac{1}{3}U^n \quad \frac{2}{3}U^{(2)} \quad tM^{\ 1}R(U^{(2)}) \ .$$

This scheme is stable (linearly) for a Courant number less than or equal to $1/(2p+1)$.

PARALLELIZATION

As the discontinuous Galerkin method is highly parallelizable, the main enabling mechanism for its suitability for implementation on parallel computers is the *local* nature of DG discretization, allowing for the formulation of very compact numerical schemes. This is due to the fact that the solution representation in an element is purposefully kept independent of the solutions in other elements, so that the inter-element communication is needed to be carried out only with adjacent elements (those sharing a common face). Thus for parallel implementation of the discontinuous Galerkin method the computational domain is partitioned among the available processors and only the information at the partition boundary faces, i.e., faces having its *left* and *right* elements on different processors, is needed to be exchanged between the corresponding neighboring processors. For such parallel solutions distributed memory architectures with message passing programming paradigm are natural to consider. That's why it is quite common that the numerical simulations in CFD are accomplished on clusters of PC components because of their cost effectiveness and easy availability as the distributed memory architectures with message passing programming paradigm. The clusters compliment rather than compete with the more sophisticated parallel computing architectures. Emergence of multi-core CPUs offers another parallel computing platform, even within a PC or laptop. Today dual-core, quad-core and six-core CPUs are commonly available and they may efficiently run up to two, four and six processes respectively, of a parallel program. Therefore the multi socket PCs can run up to 24 processes, although the amount of performance (speedup) gained by the use of a multi-core CPU is strongly dependent on the algorithm and implementation. For CFD applications the memory bandwidth is turned out to be the performance determining factor[27], setting an upper bound on speedup, especially on many-core systems.

In general, performance of a parallel program in a distributed memory environment with a given data size depends on many factors including,

- CPUs, their number and speed
- memory capacity and bandwidth, caching effects
- communication network (its data transfer rate and latency)
- communication to computation ratio.

Performance of a parallel program may be measured by computing certain performance metrics, for example, *execution time*, *relative speedup*, and *relative efficiency*. In this work, the 'relative speedup' and 'relative efficiency' are simply called as 'speedup' and 'efficiency' respectively. Execution time is the elapsed wall clock time from the start of execution of first process of a parallel program to the end of execution of its last process. It includes both computation and communication time. Relative speedup, S_p of a parallel program is the ratio of elapsed time, t_1, taken by one processor to solve the problem to the elapsed time, t_p, taken by p processors to solve the problem, i.e., $S_p = t_1/t_p$. The relative efficiency, E is defined as $E = S_p/p$. Generally, in practice, speedup remains less than p and efficiency lies between 0 and 1. In an ideal case, $t_p = t_1/p$ so that $S_p = p$ and $E = 1$. Sometimes in practice $S_p > p$ is observed, which is called as super-linear speedup. It is mainly due to the cache efficiency with smaller data sizes on the p processors as compare to serial single processor case. An important feature of a parallel program is its *scalability*. Scalability of a parallel program is a measure of its ability that its efficiency is maintained at a certain

level if both the parallel processing resources and problem size are increased in proportion to each other[28]. Scalability with respect to speedup, for the test cases is presented in the next section. The scalability analysis performed indicates that how the performance metric, speedup, of the parallel code varies with the increase in number of processors for a fixed problem size.

Domain Decomposition

For a good scalability of a domain decomposition based parallel SPMD solution, a well load-balanced and communication-efficient partitioning is quite necessary[29]. Poor load balances often result in idle time at the synchronization points, and a communication-inefficient partitioning results in larger data sizes to be communicated among the processes; hence increasing the communication burden. Although the two objectives are not perfectly compatible, a sufficiently well load-balanced and communication-efficient domain partitioning could be obtained by METIS[30] software package. For this purpose its standalone program **partdmesh** has been used to partition the unstructured mesh in the preprocessing step. **partdmesh** performs the partitioning with an objective of minimizing the total communication volume (i.e., to minimize the interface degrees of freedom). This program first converts the unstructured mesh into a dual graph, in which the vertices represent the grid elements. Then the dual graph is partitioned, by the *k*-metis program in METIS, into *k* parts of approximately the same size using the multilevel *k*-way partitioning algorithm[31].

The Parallel Algorithm

Most computational work of the discontinuous Galerkin scheme, like any explicit scheme for CFD, is performed within the subroutine that computes the residual **R** in the right hand side (including the numerical fluxes). The parallel performance of such a scheme is bounded either by memory bandwidth or by the number of basic operations that can be performed in a single clock cycle[29].

The parallel implementation is obtained without compromising the serial algorithm. Moreover the MPI communications may be overlapped with the computations hence significantly reduce the overhead incurred due to parallelization. This is commonly achieved by initiating non-blocking sending and receiving operations so that the computations remain continue meanwhile the communication is completed. Such a scenario is usually referred to as hiding communication behind computation. This is easier to achieve in explicit time marching schemes[32]. Following is the algorithm used to compute the right hand side (RHS) residual (including flux computations) on each processor having a subdomain after domain partitioning. The word "local" refers to what is owned by 'this' processor with respect to its subdomain:

INITIATE *Non-blocking Communication to RECEIVE data from neighboring processors*

INITIATE *Non-blocking Communication to SEND data to neighboring processors*

START LOOP 1 *(over the local faces lying on the grid Boundary)*

Compute RHS from Boundary face contributions, using local data

END LOOP 1

START LOOP 2 *(over the local elements)*

Compute RHS from domain contributions, using local data

END LOOP 2

START LOOP 3 *(over the local internal faces)*

IF *(both left and right elements are local)* **THEN**

Compute RHS from Internal face contributions using local data

ENDIF

END LOOP 3

WAIT for the completion of non blocking communications initiated above

START LOOP 4 (over the local faces having one side element non-local)

Compute RHS from Internal face contributions using local data and the received data

END LOOP 4

Depending upon the sophistication of MPI implementation used, the communication may already be completed before reaching the WAIT point, for the problems with sufficiently large data sizes, i.e., those with higher computation-to-communication ratio, at each processor. In the explicit scheme the above algorithm is called in every stage of the RK method.

NUMERICAL RESULTS

The numerical results, in this work, are computed on a Linux cluster having hundreds of nodes interconnected via gigabit Ethernet (1000 baseT). The cluster had a heterogeneous architecture with a mix of nodes, each having 2 dualcore or quadcore CPUs of varying speeds and/or memory subsystem characteristics. It is worth to mention that the cluster is shared among hundreds of users at the same time so even if some nodes runs processes of only a specific job request, the network bandwidth cannot be guaranteed to be dedicated solely for that job. That's why quite varying execution timings may be observed with the same problem size on different occasions. For obtaining some justified parallel performance measures of the code on such a heterogeneous and shared cluster, special care has been taken to compute the result on nodes with similar architecture and CPUs for a particular test case with specific problem size. For example for a problem *A* with data size *K*, all the results from 1 to 16 processors are obtained on the similar nodes, preferably within a single blade server chassis. The performance measurements of the parallel code with the test cases selected have mostly been performed on 4-nodes such that each node has two Xeon 5520 quadcore processors (i.e., 8 cores on a node). The software setup used includes 64-bit compilers and MPICH implementation of MPI library. All the floating point operations are performed in double precision. Moreover one-to-one mapping of MPI processes and physical CPU cores is assumed as the job scheduling software on cluster assumes so. Thus the words 'process', 'processor' and 'CPU core' are used to refer to the same entity.

It has already been verified that a formal order of convergence rate of the DG method based on a Taylor basis can be achieved for the following test cases on hybrid grids[13]. Further the superior accuracy of second order DG method using a Taylor basis, over the second order finite volume method has already been demonstrated[13]. Note that for all the following problems the slip boundary conditions are used for solid walls and the characteristic boundary conditions are imposed on the far-field boundaries. The detailed implementations of the boundary conditions can be found in the Reference[18].

Subsonic flow past a circular cylinder

This well-known test case involves the solution of inviscid subsonic flow past a circular cylinder at a Mach number of 0.38. The numerical solutions of this problem are computed using the parallel DG(P0), DG(P1), and DG(P2) methods (i.e., the discontinuous Galerkin methods of first, second and third order of accuracy, respectively) on a triangular element mesh consisting of 8192 elements, 4224 grid points, and 256 boundary points. A sketch of the complete physical domain showing its size (in units of length), along with the triangular mesh is given in Figure 2. Partitioning of this mesh using METIS for 16 processes is depicted in Figure 3. The resulting contour plots for the three approximation orders are compared for the respective serial and parallel codes in Figure 4(a-c)

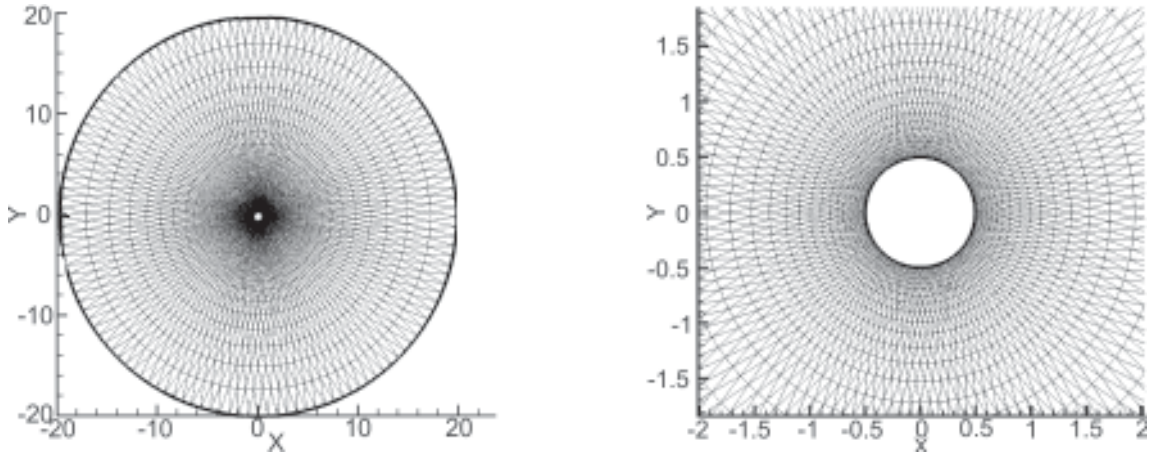

Figure 2: *Subsonic flow past a circular cylinder problem*; Triangular mesh used to obtain the solution (central region focused in the right one). The governing equations are dimensionless; there could be any unit of measurement. Here the computational domain extends to 40 times the chord length of the cylinder.

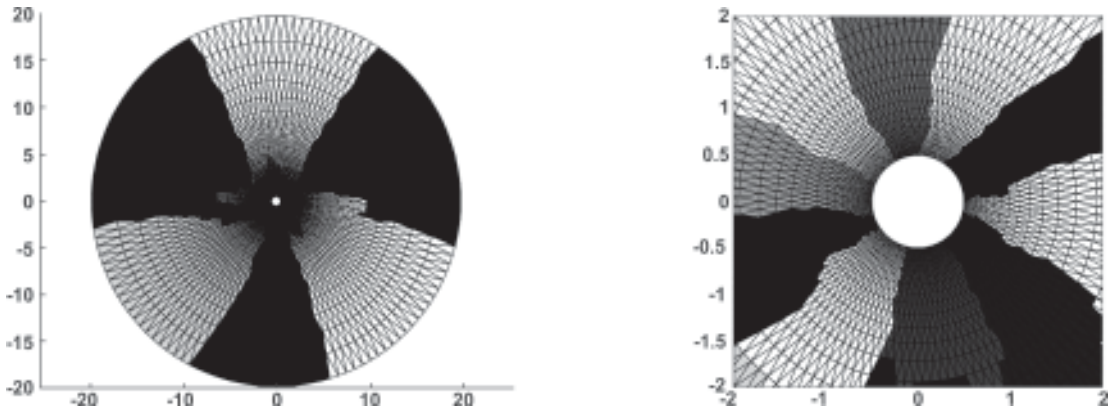

Figure 3: *Subsonic flow past a circular cylinder problem*; Triangular mesh partitioned for 16 processes (central region focused in the right one).

and Figure 5. The results computed, using the parallel DG code, are in full agreement with those obtained with the respective serial DG code[13]. Hence, the convergence and accuracy is not compromised at all in our parallel version of the DG method.

Recall that a 4-node-cluster setup of dual Xeon-5520 processors (having 32 CPU-cores in total) is used for the measurement of parallel performance of the code. For demonstration of parallel scalability, a relatively larger mesh of just over 45 thousand elements is considered. The larger mesh is used so that a reasonable work load would be allocated to each of the parallel processes. Parallel scalability up to 16 processes on the 4 nodes of the Ethernet based cluster with the DG(P1) method are shown in Figure 6. A speedup of 7.52 is observed when the code is executed on 8 processors/cores (i.e., 7.52/(8´100) = 94% per processor efficiency). The obtained speedup value of 7.52 with 8 processes may be regarded as near-linear (or near-ideal) speedup as it is close the linear (or ideal) speedup value of 8. Further a good speedup of 12.15 on 16 processors/cores is observed (i.e., about 76% per processor efficiency). Generally the speedup decreases with the increase in number of processes, because more parallel processes involve extra network congestion and other parallel overheads. Moreover, a reason for the slightly over-dropped speedup for 12 and 16 processors cases is the "network interface congestion". In the case of 4 processes, only one process is mapped on each node of the 4-node cluster. Note that, in Figure 6, a speedup value of greater than 4 is observed on 4 processors. This reflects over 100% per processor efficiency, and is called as super-linear speedup. The super-linear speedup might be observed for certain test cases, usually due to the cache effects, because the distribution of the chunks of the problem data among the processes on different nodes causes largest percentage of local data to fit into the available cache. This decreases the cache-miss ratio and results in better performance. On the other hand, for more than 4 processes, more than 1 process is mapped per node. Thus, in the case of 16 processes, 4 processes are mapped on each node. More than one processes mapped on a node strive for their turn to communicate through the network interface of that node. Thus the network interface bandwidth is shared among a number of processes. This causes "network interface congestion" and, hence, resulting in the drop of parallel efficiency.

Transonic flow past a NACA0012 airfoil

This well-known test case involves solution of inviscid transonic flow past a NACA0012 airfoil at a Mach number of 0.8, and an angle of attack 1.25°. There exists a strong shock on the upper surface and a weak shock on the lower surface of the airfoil. The numerical solutions of this problem are computed using the parallel DG method on a triangular element mesh consisting of 1999 elements, 1048 grid points, and 97 boundary points. A sketch of the complete physical domain showing its size (in units of length), along with the triangular mesh is given in Figure 7. The mesh decomposed into 16 parts using METIS is shown in Figure 8. The computed pressure and Mach number contours using the parallel DG(P2) method with 16 processes and the respective serial code are compared in Figure 9 and Figure 10 respectively. Again there is complete agreement between the results

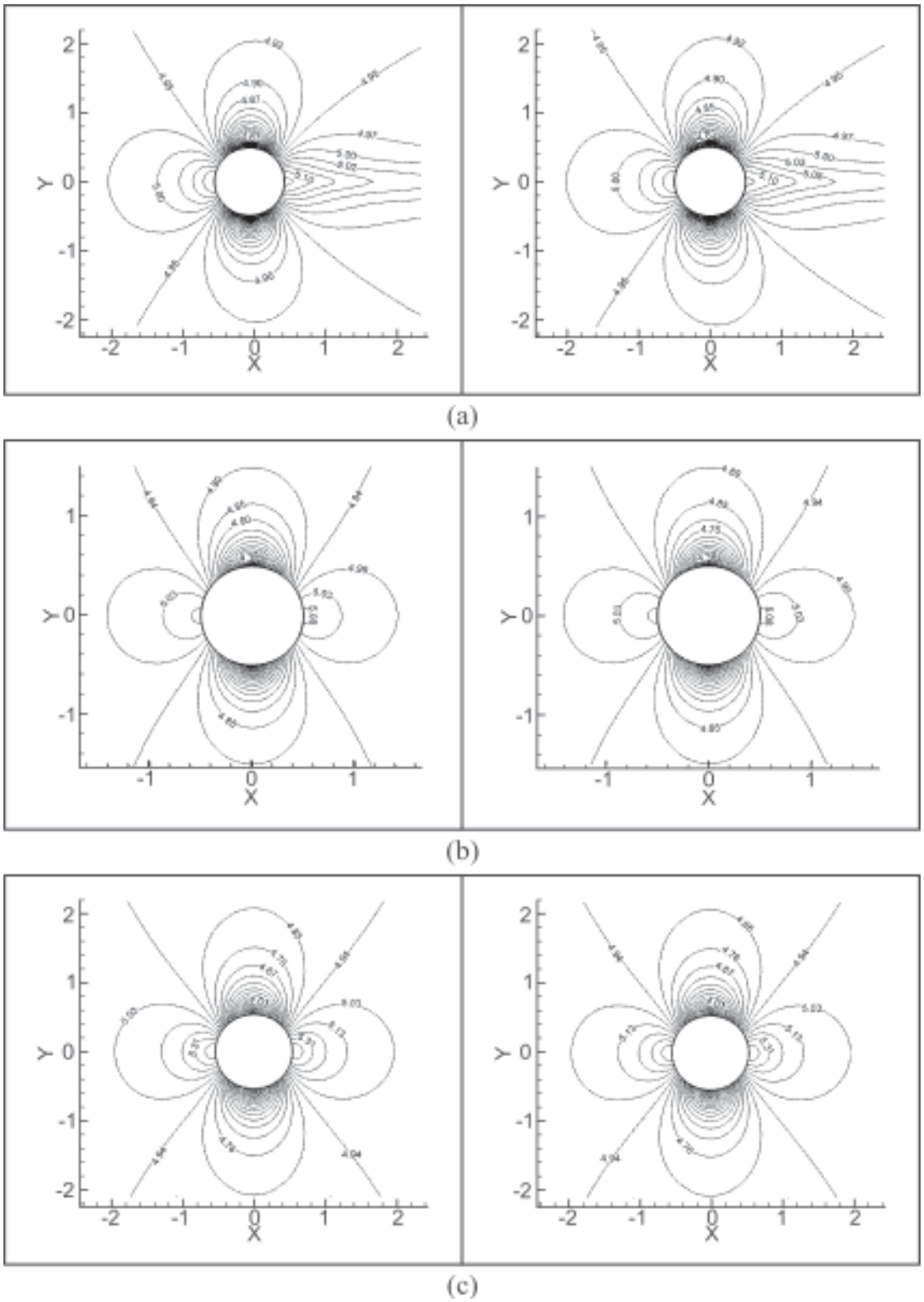

Figure 4: *Subsonic flow past a circular cylinder problem*; Comparison of pressure contours obtained using the parallel DG method (left) and the serial DG method (right), (a) P0 approximation case, (b) P1 approximation case, (c) P2 approximation case. The corresponding profiles are identical. The contour levels vary between a minimum value 4.3 and a maximum value 5.1 in (a), a minimum value 4.0 and a maximum value 5.0 in (b), and a minimum value 3.2 and a maximum value 5.4 in (c).

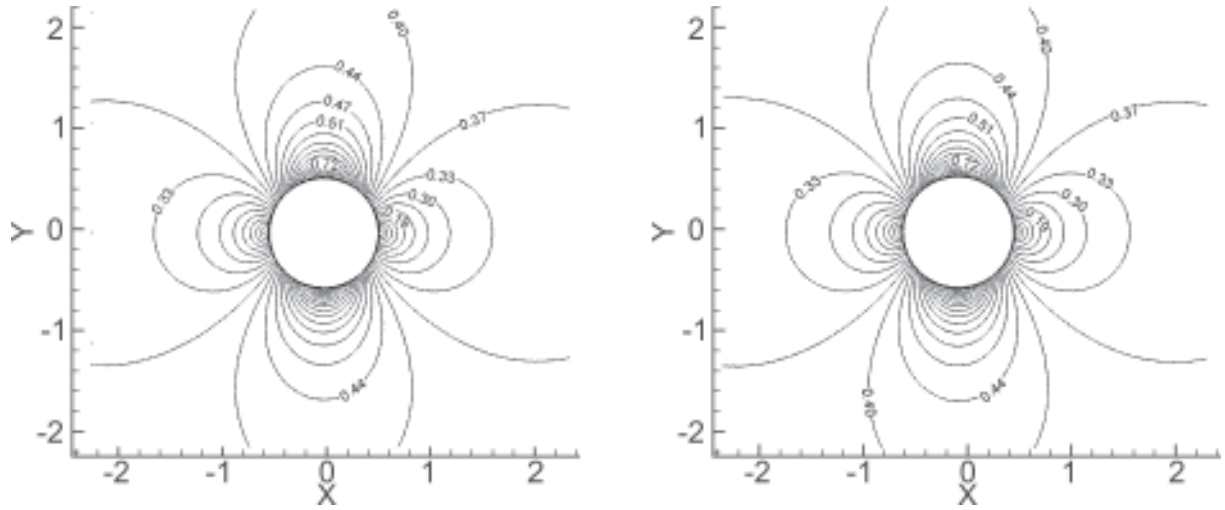

Figure 5: *Subsonic flow past a circular cylinder problem*; Comparison of Mach number contours obtained using the parallel DG method (left) and the serial DG method (right) with P2 approximation. Both plots are identical. The contour levels vary between a minimum value 0.05 and a maximum value 0.9.

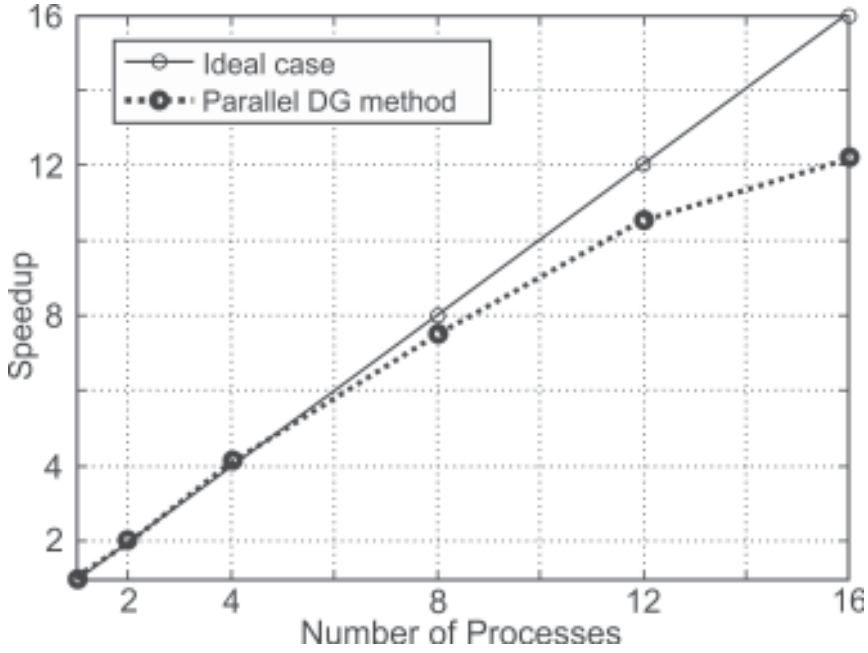

Figure 6: *Subsonic flow past a circular cylinder problem*; Scalability: speedup versus number of processes.

obtained using the parallel and serial version of the DG method.

For demonstration of parallel scalability, a relatively larger mesh of just over 22 thousand elements is considered. Parallel scalability up to 16 processes on the 4-nodes of the Ethernet based cluster with the DG(P1) method are shown in Figure 11. A near linear (or near ideal) speedup value of 7.6 is observed with 8 processes (i.e., over 95% per processor efficiency). A good speedup of 11.7 on 16 processors is observed. The reasons for super-linear speedup with 4 processes on the 4-nodes and for the drop in speedup on 16 processors are the same as described earlier in section 5.1.

Subsonic flow past a three-element airfoil

In this test case a subsonic flow past three-element airfoil is presented. The results are computed at a Mach number of 0.2, and an angle of attack 16^{o}. The numerical solutions of this problem are computed

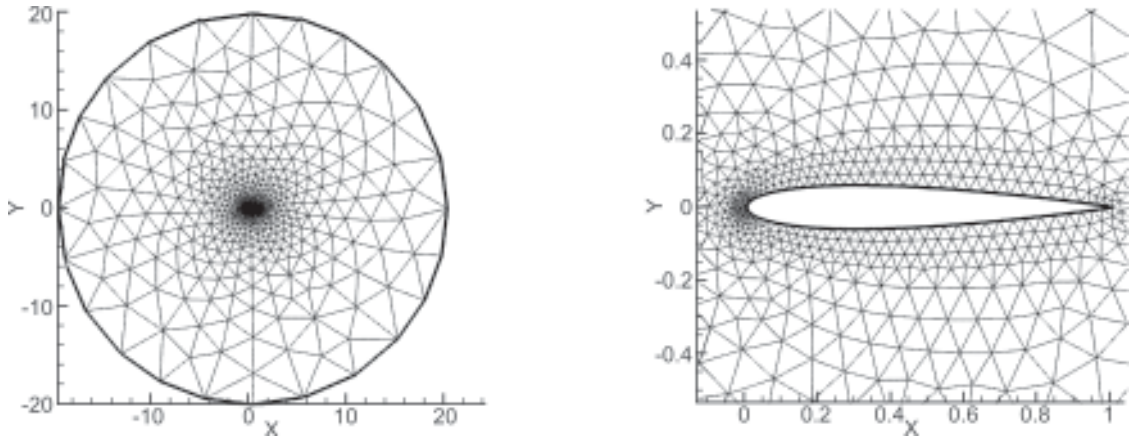

Figure 7: *Transonic flow past NACA0012 airfoil problem*; Triangular mesh used to obtain the solution (central region focused in the right one). The governing equations are dimensionless; there could be any unit of measurement. Here the computational domain extends to 40 times the chord length of the airfoil.

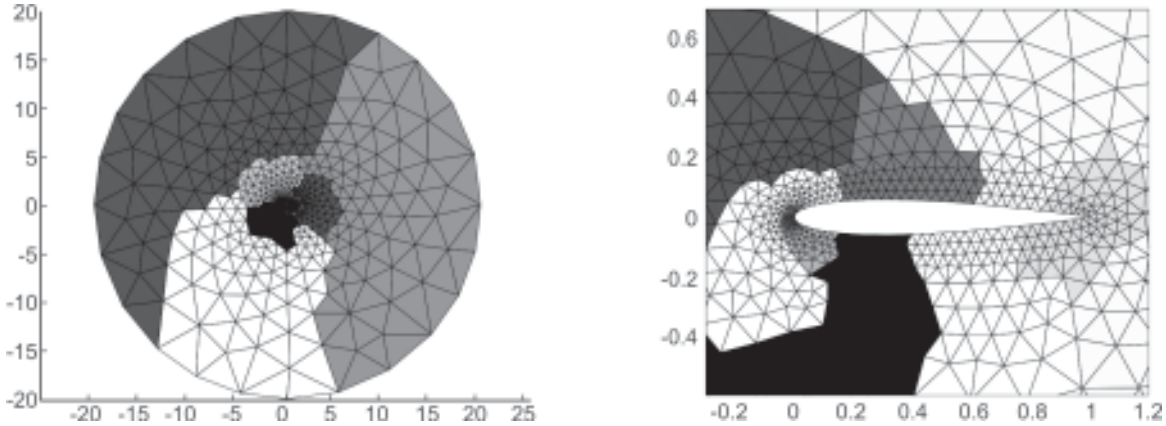

Figure 8: *Transonic flow past NACA0012 airfoil problem*; Triangular mesh partitioned for 16 processes (central region focused in the right one).

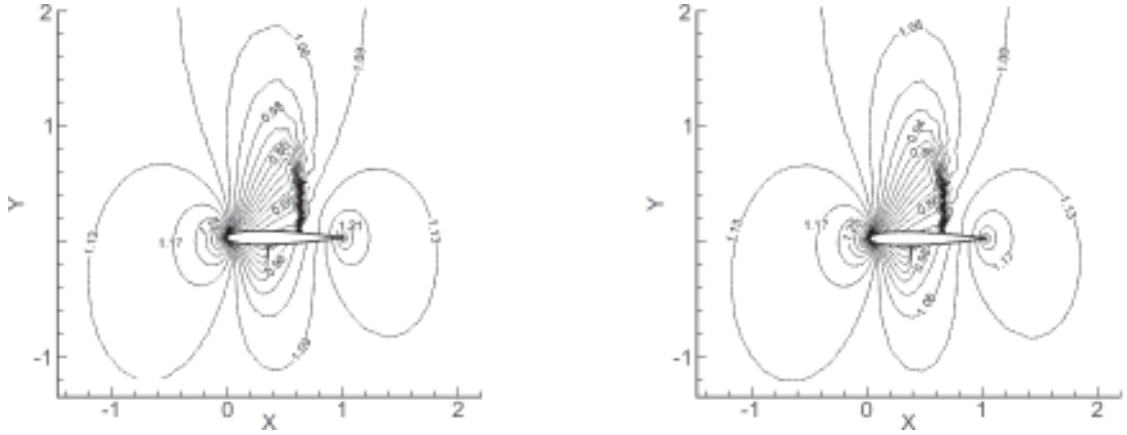

Figure 9: *Transonic flow past NACA0012 airfoil problem*; Comparison of pressure contours obtained using the parallel DG method (left) and the serial DG method (right) with P2 approximation. Both plots are identical. The contour levels vary between a minimum value 0.5 and a maximum value 1.65.

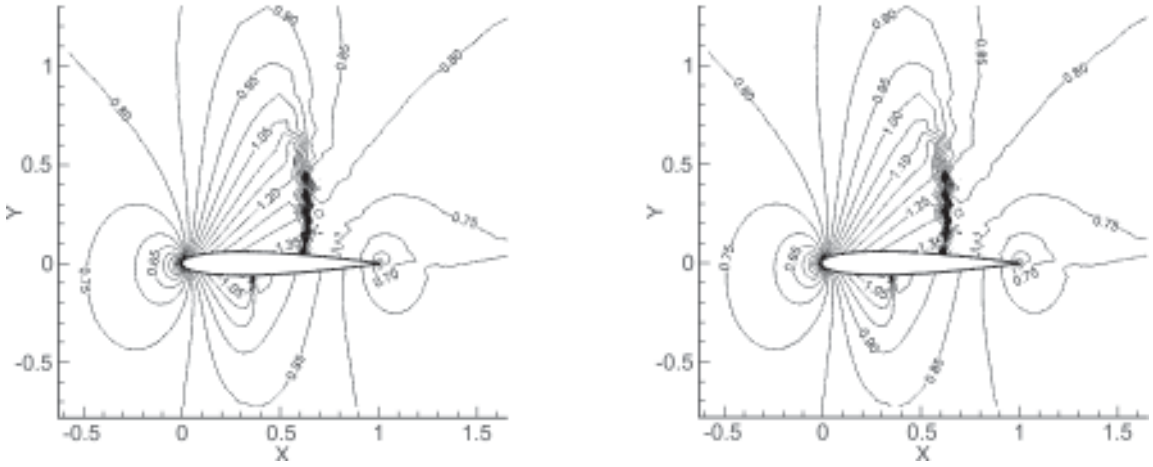

Figure 10: *Transonic flow past NACA0012 airfoil problem*; Comparison of Mach number contours obtained using the parallel DG method (left) and the serial DG method (right) with P2 approximation. Both plots are identical. The contour levels vary between a minimum value 0.05 and a maximum value 1.55.

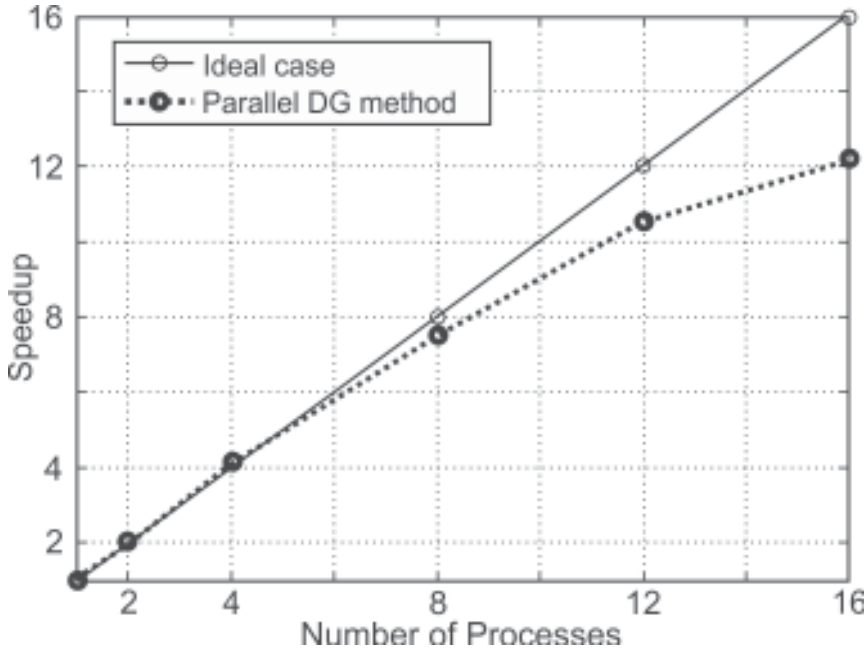

Figure 11: *Transonic flow past NACA0012 airfoil problem*; Scalability: speedup versus number of processes.

using the parallel DG method on a triangular element mesh consisting of 4768 elements, 2531 grid points, and 298 boundary points. A sketch of the complete physical domain showing its size (in units of length), along with the triangular mesh is given in Figure 12. The mesh decomposed into 8 parts using METIS is shown in Figure 13. The computed pressure and Mach number contours using the parallel DG(P2) method with 16 processes and the respective serial code are compared in Figure 14 and Figure 15, respectively. Again there is complete agreement between the results obtained by using the parallel and serial version of the DG method.

For the measurement of parallel performance of the code on this test case, the numerical solutions are computed on a triangular element mesh consisting of 19244 elements, 9922 grid points, and 604 boundary points. The parallel scalability up to 16 processes on the 4 nodes of the Ethernet based cluster with the

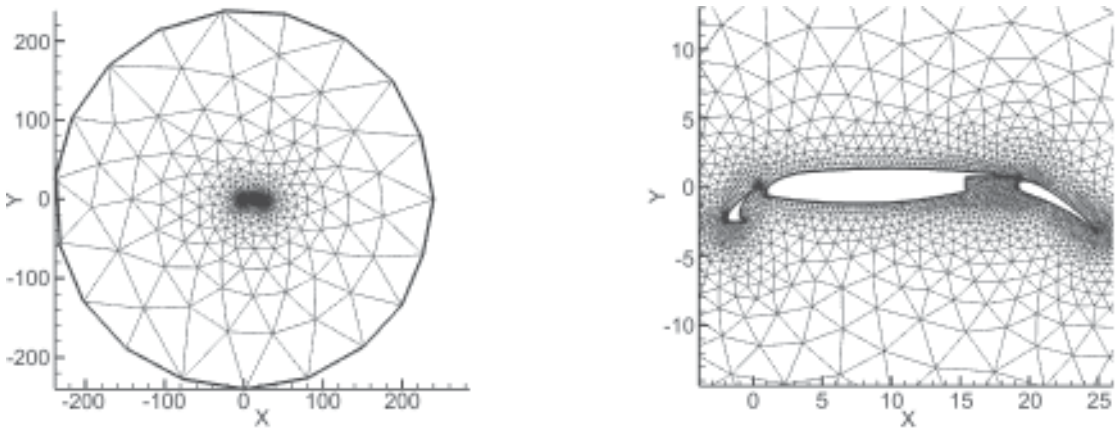

Figure 12: *Subsonic flow past a 3-element airfoil problem*; Triangular mesh used to obtain the solution (central region focused in the right one). The governing equations are dimensionless; there could be any unit of measurement.

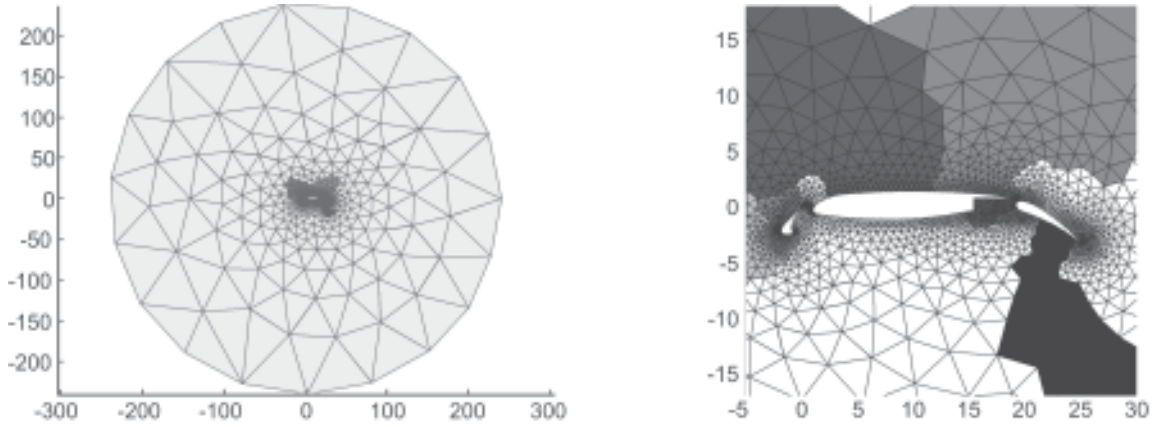

Figure 13: *Subsonic flow past a 3-element airfoil problem;* Triangular mesh partitioned for 16 processes (central region focused in the right one).

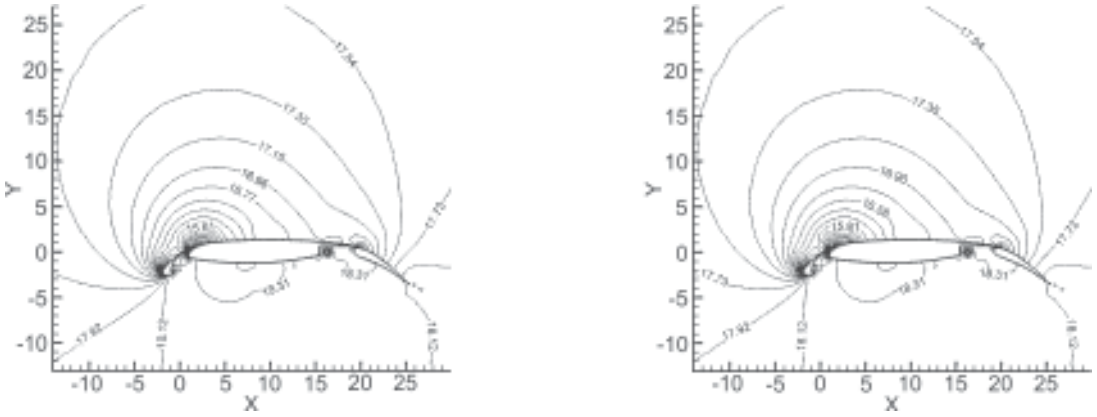

Figure 14: *Subsonic flow past a 3-element airfoil problem*; Comparison of pressure contours obtained using the parallel DG method (left) and the serial DG method (right) with P2 approximation. Both plots are identical. The contour levels vary between a minimum value 11.0 and a maximum value 18.5.

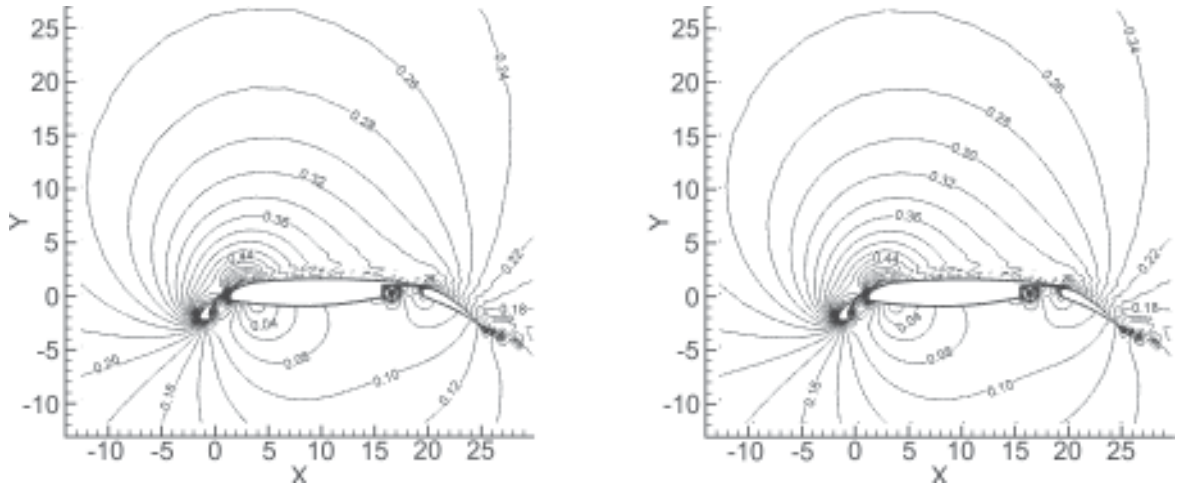

Figure 15: *Subsonic flow past a 3-element airfoil problem*; Comparison of Mach number contours obtained using the parallel DG method (left) and the serial DG method (right) with P2 approximation. Both plots are identical. The contour levels vary between a minimum value 0.02 and a maximum value 0.9.

DG(P1) method is shown in Figure 16. A super linear speedup is observed up to 8 processors (i.e., over 100% per processor efficiency). In this test case, the mesh size used is smaller than those used in the previous test cases. The reason for better performance than previous test cases might be that the relatively smaller data sizes allocated to each process result in fitting of larger part of the local data into the available cache. A good speedup of 12.63 on 16 processors is observed. The reason for drop in speedup on 16 processors is the same as described earlier in section 5.1.

At the last, the convergence behavior of the parallel DG and the serial DG codes is demonstrated to be identical. To test the convergence history, the plots of the logarithm of the relative L_2 norm of the density residual versus the number of time steps for the above three numerical examples are shown in Figure 17. The comparison shows that after any num-

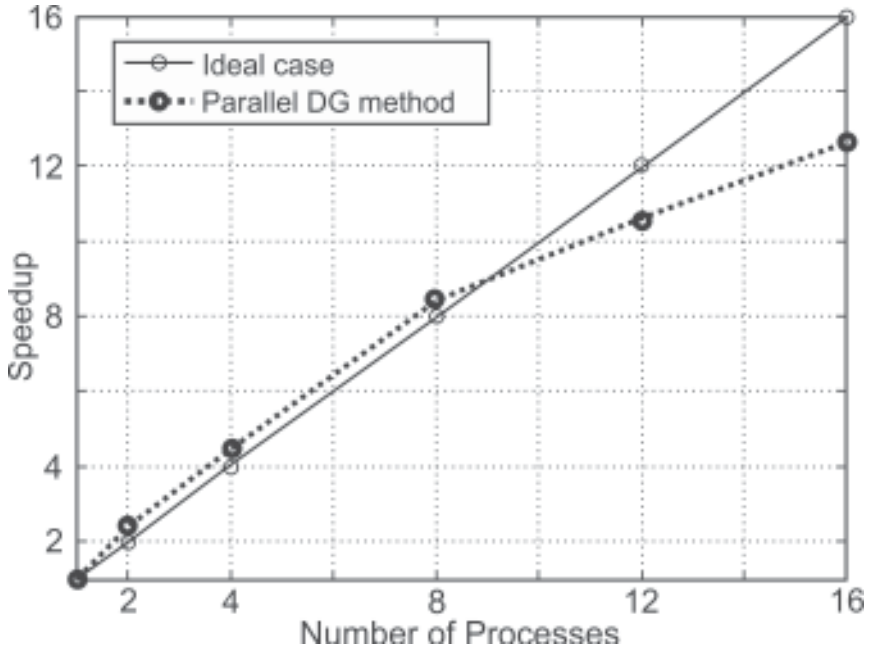

Figure 16: *Subsonic flow past a 3-element airfoil problem*; Scalability: speedup versus number of processes.

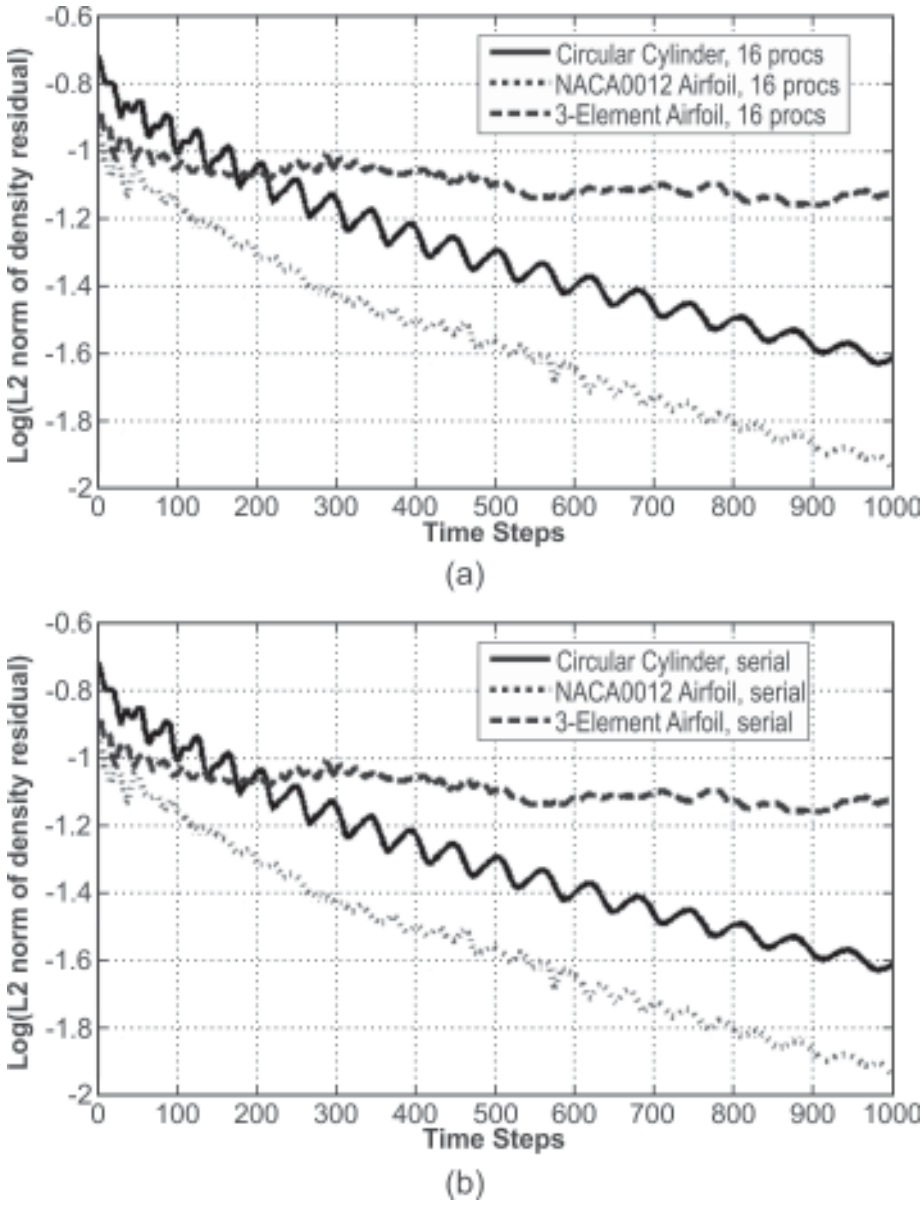

Figure 17: Comparison of convergence histories; Log(residual-norm) versus time steps, (a) for the parallel DG code, (b) for the serial DG code.

ber of time steps the convergence level achieved by the parallel DG code is the same as achieved by the serial DG code, hence the convergence of the serial DG algorithm is not compromised at all in the parallel algorithm.

CONCLUSION

In this work, a serial DG method (presented in Reference 13) is parallelized by designing and implementing a parallel algorithm/version of the method for distributed memory parallel computers using MPI library. The well known concept of hiding communication behind computations (i.e., overlapping of communication with computation) is also incorporated in the parallel algorithm for reducing the effects of parallel overheads. The "accuracy" and "convergence" of the parallel code are demonstrated to be identical to those exhibited by the serial DG method. Hence, the parallelization has been performed without compromising the serial algorithm. The "scalability" of the developed parallel algorithm is also demonstrated

on an Ethernet based cluster. Future work should include parallel performance profiling and tuning, extension of the parallel solution for implicit schemes and developing a multi-level grid acceleration provision, as well.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Gary W. Howell (Applications Scientist for High Performance Computing, NCSU) for his valuable support in using the henry2 cluster system at NCSU. The first author also thanks Higher Education Commission (HEC), Pakistan for providing travel grant.

REFERENCES

1. *Cockburn, B., Karniadakis, G.E., and Shu, C.W. (Eds.), 2000. Discontinuous Galerkin Methods, Theory, Computation, and Applications, Lecture Notes in Computational Science and Engineering, Springer-Verlag, New York, Vol. 11.*

2. *Cockburn, B., and Shu, C.W., 1998. The Runge-Kutta discontinuous Galerkin method for conservation laws V: multidimensional system. Journal of Computational Physics, Vol. 141, 199-224.*

3. *Arnold, D.N., Brezzi, F., Cockburn, B., and Marini, D., 2001. Unified Analysis of discontinuous Galerkin methods for elliptic problems. SIAM Journal on Numerical Analysis, Vol. 39 (5), 1749-1779.*

4. *Hesthaven, J.S., and Warburton, T., 2008. Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications. Texts in Applied Mathematics, Vol. 56, Springer, New York.*

5. *Peraire, J., and Persson, P.O., 2008. The compact discontinuous Galerkin method for elliptic problems. SIAM Journal on Scientific Computing, Vol. 30, 1806-1824.*

6. *Atkins, H.L., and Shu, C.W., 1998. Quadrature free implementation of discontinuous Galerkin method for hyperbolic equations. AIAA Journal, Vol. 36(5), 775-782.*

7. *Rasetarinera, P., and Hussaini, M.Y., 2001. An efficient implicit discontinuous spectral Galerkin method. Journal of Computational Physics, Vol. 172, 718-738.*

8. *Dumbser, M., Balsara, D.S., Toro, E.F., and Munz, C.D., 2008. A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes. Journal of Computational Physics, Vol. 227, 8209-8253.*

9. *Van-Leer, B., and Nomura, S., 2005. Discontinuous Galerkin method for diffusion. 43rd AIAA Aerospace Sciences Meeting, AIAA-2005-5108.*

10. *Cockburn, B., and Shu, C.W., 2001. The local discontinuous Galerkin method for time-dependent convection-diffusion system. SIAM Journal on Numerical Analysis, Vol. 16.*

11. *Bassi, F., and Rebay, S., 2005. Discontinuous Galerkin solution of the Reynolds-averaged Navier-Stokes and k-ù turbulence model equations. Journal of Computational Physics, Vol. 34, 507-540.*

12. *Luo, H., Ali, A., Nourgaliev, R., and Mousseau, V.A., 2010. Reconstructed discontinuous Galerkin method for the compressible flows on arbitrary grids. 48th AIAA Aerospace Sciences Meeting, AIAA-2010-0366.*

13. *Luo, H., Baum, J.D., and Löhner, R., 2008. A discontinuous Galerkin method using Taylor basis for the compressible flows on arbitrary grids. Journal of Computational Physics, Vol. 227(20), 8875-8893.*

14. *Baumann, C.E., and Oden, J.T., 1999. A discontinuous hp finite element method for the Euler and the Navier–Stokes equations. International Journal of Numerical Methods in Fluids, Vol. 31, 79-95.*

15. *Bassi, F., and Rebay, S., 1997. High-order accurate discontinuous finite element solution of the 2D Euler equations. Journal of Computational Physics, Vol. 138, 251-285.*

16. *Bassi, F., and Rebay, S., 1997. A high-order accurate discontinuous Galerkin finite element method for the numerical solution of the compressible Navier–Stokes equations. Journal of Computational Physics, 131, 267-279.*

17. *Fidkowski, K.J., Oliver, T.A., Lu, J., and Darmofal, D.L., 2005. p-Multigrid solution of high-order discontinuous Galerkin discretiza-*

tions of the compressible Navier–Stokes equations. Journal of Computational Physics, Vol. 207(1), 92-113.

18. *Luo, H., Baum, J.D., and Löhner, R., 2008. On the computation of steady-state compressible flows using a discontinuous Galerkin method. International Journal for Numerical Methods in Engineering, Vol. 73(5), 597-623.*

19. *Luo, H., Baum, J.D., and Löhner, R., 2008. A fast,* p*-multigrid discontinuous Galerkin method for compressible flows at all speeds. AIAA Journal, Vol. 46(3), 635-652.*

20. *Toulopoulos, I., and Ekaterinaris, J.A., 2006. High-order discontinuous Galerkin discritizations for computational aeroacoustics in complex domains. AIAA Journal, Vol. 44 (3), 502-511.*

21. *Sterling, T., Becker, D.J., and Savarese, D.F., 1999. How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters, The MIT Press, Cambridge, MA.*

22. *Lucke, R.W., 2004. Building Clustered Linux Systems, Prentice Hall PTR, Indianapolis, IN.*

23. *Toro, E.F., Spruce, M., and Speares, W., 1994. Restoration of the contact surface in the HLL-Riemann solver. Shock Waves, Vol. 4, 25-34.*

24. *Batten, P., Leschziner, M.A., and Goldberg, U.C., 1997. Average-state Jacobians and implicit methods for compressible viscous and turbulent flows. Journal of Computational Physics, Vol. 137, 38-78.*

25. *Luo, H., Baum, J.D., and Löhner, R., 2005. High-Reynolds number viscous flow computations using an unstructured-grid method. Journal of Aircraft, Vol. 42 (2), 483-492.*

26 *Luo, H., Baum, J.D., and Löhner, R., 2005. Extension of HLLC scheme for flows at all speeds. AIAA Journal, Vol. 43(6).*

27. *Gropp, W.D., Kaushik, D.K., Keyes, D.E., and Smith, B.F., 1999. Toward realistic performance bounds for implicit CFD codes. In Keyes, D.K., Ecer, A., Periaux, J., Satofuka, N., and Fox, P. (Eds.), Proceedings of Parallel CFD'99, Elsevier, 233-240.*

28. *Grama, A., Gupta, A., Karypis, G., and Kumar, V., 2003. Introduction to Parallel Computing, 2nd ed., Addison-Wesley, Boston, MA.*

29. *Gropp, W.D., Kaushik, D.K., Keyes, D.E., and Smith, B.F., 2001. High performance parallel implicit CFD. Parallel Computing, Vol. 27, 337-362.*

30. *Karypis, G., and Kumar, V., 1999. A fast and high quality scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, Vol. 20, 359-392.*

31. *Karypis, G., and Kumar, V., 1998. Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, Vol. 48(1), 96-129.*

32. *Baggag, A., Atkins, H., and Keyes, D., 1999. Parallel implementation of the discontinuous Galerkin method". NASA ICASE Technical Report No. 99-35.*