OpenACC-based GPU Acceleration of a p-multigrid Discontinuous Galerkin Method for Compressible Flows on 3D Unstructured Grids

Jialin Lou [*], Yidong Xia [†], Lixiang Luo [‡], Hong Luo [§] and Jack Edwards [¶‡]

Department of Mechanical and Aerospace Engineering

North Carolina State University, Raleigh, NC 27695-7910, United States

and

Frank Mueller [‖]

Department of Computer Science

North Carolina State University, Raleigh, NC 27695-8206, United States

A GPU accelerated p-multigrid discontinuous Galerkin method is developed based on the OpenACC directives for the solution of the compressible flows on 3-D unstructured grids. The present design of GPU computing based on OpenACC is typically aimed to upgrade a legacy CFD solver with the capability of GPU computing without significant intrusion or algorithm alteration to the existing CPU code. In the present method, the p-multigrid technique proves to be a cost-effective approach for convergence acceleration, since the problem size would be seriously restricted if an implicit DG method with the use of the higher-order Jacobian matrix involved, provided with the fact that the GPU memory is still far from abundant for high-order methods even on a top-rank one. Specifically, a multi-stage explicit time stepping scheme is used for advancing the higher-order approximation in time, with the first-order implicit backward Euler scheme is applied to accelerate the lower-order approximation. A variety of inviscid and viscous flow problems are presented to verify and validate the developed solver. In each case, a strong scaling test is conducted on an NVIDIA Tesla K20c GPU to assess the performance of the GPU code, and in addition, the speedup factors are presented by performing a comparative study with the equivalent CPU code running on a computed node that consists of two eight-core AMD Opteron-6128 CPUs. The numerical results indicate that the resulting p-multigrid discontinuous Galerkin method is a competitive and accurate method for GPU computing.

I. Introduction

Computational Fluid Dynamics (CFD), a branch of mechanics that applies numerical way to solve fluid dynamics problem, has been one of the most significant applications on supercomputers. However, the capabilities of the traditional CPU(central processing unit) based parallel computing may not meet the needs of solving complex simulation problems. Fortunately, the General-Purpose Graphics Processing Unit (GPGPU[26]) technology offers a new opportunity to significantly accelerate CFD simulations by offloading compute-intensive portions of the application to the GPU, while the remainder of the computer program still runs on the CPU, which also make it expected to be a major compute unit in the near future.

NVIDIA's CUDA application programming interface (API) and CUDA-enabled accelerators are regarded as a popular parallel programming model and platform in GPGPU technology. Therefore, many researchers

[*]Graduate Student, AIAA Student Member. Corresponding Author: jlou3@ncsu.edu

[†]Postdoctoral Research Associate, AIAA Member.

[‡]Postdoctoral Research Associate.

[§]Professor, AIAA Associate Fellow.

[¶]Professor, AIAA Associate Fellow.

[‖]Professor

have put effort in the investigation and development GPU-accelerated CFD solvers with the CUDA technology[1,5,6,9,10,11,15,17,18,25,27,30] . As a matter of fact, the numerical methods range from the finite difference methods (FDMs), spectral difference methods (SDMs), finite volume methods (FVMs), discontinuous Galerkin methods (DGMs) to Lattice Boltzmann method (LBMs). For instance, Elsen *et al.*[13] reported a 3D high-order FDM solver for large calculation on multi-block structured grids; Klöckner *et al.*[20] developed a 3D unstructured high-order nodal DGM solver for the Maxwell's equations; Corrigan *et al.*[12] proposed a 3D FVM solver for compressible inviscid flows on unstructured tetrahedral grids; Zimmerman *et al.*[37] presented an SDM solver for the Navier-Stokes equations on unstructured hexahedral grids.

In order to adopt the CUDA technology to develop GPU-accelerated CFD solver, people would either need to start a brand new code design or extend an existing CPU code to GPU platform. The former one is often the case of fundamental study of a numerical model on GPU computing while the latter one requires applying NVIDIA CUDA model to a legacy CFD code, which is not an easy job since the developer has to define an explicit layout of the threads on the GPU (numbers of blocks, numbers of threads) for each kernel function.[19] For either production level or research level, people would prefer to maintain multi-platform compatibility at a minimum extra cost in time and effort. Nevertheless, adopting CUDA might spell almost a brand new design and long-term project, and a constraint to the CUDA-enabled devices, thus to lose the code portability on other platforms. Therefore, some alternatives come into play including OpenCL:[29] the currently dominant open GPGPU programming model (but dropped from further discussion since it does not support Fortran); and OpenACC:[31] a new open parallel programming standard based on directives.

For people who have experience with OpenMP, they would find OpenACC is much like OpenMP. What developers need to do is simply annotate their code to identify the areas that should be accelerated by wrapping with the OpenACC directives and some runtime library routines, instead of taking the huge effort to change the original algorithms as to accommodate the code to a specific GPU architecture and compiler. In that case, people benefit not only from easy implementation of the directives but also the freedom to compile the very same code and conduct computations on either CPU or GPU from different vendors, e.g., NVIDIA, AMD and Intel accelerators. Nevertheless, as for some cutting-edge features, OpenACC still lags behind CUDA due to vendors' distribution plan (note that Nvidia is among the OpenACC's main supporters). But still, OpenACC manages to offer a promising approach to minimize the effort to extend the existing legacy CFD codes while maintaining multi-platform and multi-vendor compatibility, and thus to become an attractive parallel programming model.

The objective of the effort discussed in the present work is to develop a GPU accelerated,p-multigrid discontinuous Galerkin method, for the solution of the compressible Navier-Stokes equations on 3D unstructured grids. This work is based on a class of reconstruction-based RDG(PnPm) methods,[23,24,32,34,35,36] which are recently developed in order to improve the overall performance of the underlying standard DG(Pn) methods without significant extra costs in terms of computing time and storage requirement. Due to the fact that OpenACC could offer multi-platform/complier support with minor effort to code directives, it has been employed to partially upgrade a legacy Navier-Stokes solver with the GPU computing capacity. Part of the solution modules in a well verified and validated RDG flow solver have already been upgraded with the capability of single-GPU computing based on the OpenACC directives in our prior work.[33] As for p-multigrid discontinuous Galerkin method,[14,16,21,22] it would allow higher approximation level would use explicit time integration while the lower approximation level applies implicit scheme, so that to accelerate the convergence to the steady state. In addition, a face renumbering and grouping algorithm is used to eliminate "race condition" in face-based flux calculation that takes place on GPU vectorization. The developed method is used to compute the compressible flows for a variety of test problems on unstructured grids. Speed-up factors that achieved by comparing the computing time of the OpenACC program on an NVIDIA Tesla K20c GPU and that of the equivalent MPI program on one single core and full sixteen cores of an AMD Opteron-6128 CPU indicate that p-multigrid discontinuous Galerkin method is a cost-effective high-order scheme on OpenACC-based GPU platform.

The outline of the rest of this abstract is organized as follows. The governing equations are briefly introduced in Section II. In Section III, the discontinuous Galerkin spatial discretization is described. In Section IV, the idea of p-multigrid method is given. In Section V, the keynotes of porting a p-multigrid discontinuous Galerkin flow solver to GPU based on the OpenACC directives is discussed in detail. In Section VI, a series of numerical test cases are presented. Finally the concluding remarks and plan of future work are given in Section VII.

II. Governing Equations

The Navier-Stokes equations governing the unsteady compressible viscous flows can be expressed as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_k(\mathbf{U})}{\partial x_k} = 0 \tag{1}$$

where the summation convention has been used. The conservative variable vector $\mathbf{U}$ and advective flux vector $\mathbf{F}$, are defined by

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

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

$$p = (\gamma - 1)\rho\left(e - \frac{1}{2}u_i u_i\right) \tag{3}$$

which is valid for perfect gas. The ratio of the specific heats γ is assumed to be constant and equal to 1.4 for air or diatomic perfect gas.

III. Discontinuous Galerkin Spatial Discretization

The governing equations in Eq. 1 can be discretized using a discontinuous Galerkin finite element formulation. We assume that the domain Ω is subdivided into a collection of non-overlapping arbitrary elements Ω_e in 3D, and then introduce the following broken Sobolev space V_h^p

$$V_h^p = \left\{ v_h \in \left[L^2(\Omega)\right]^m : v_h|_{\Omega_e} \in \left[V_p^m\right] \forall \Omega_e \in \Omega \right\} \tag{4}$$

which consists of discontinuous vector polynomial functions of degree p, and where m is the dimension of the unknown vector and V_p is the space of all polynomials of degree $\leq p$. To formulate the discontinuous Galerkin method, we introduce the following weak formulation, which is obtained by multiplying Eq. 1 by a test function $\mathbf{W}_h$, integrating over an element Ω_e, and then performing an integration by parts: find $\mathbf{U}_h \in V_h^p$ such as

$$\frac{d}{dt}\int_{\Omega_e} \mathbf{U}_h \mathbf{W}_h d\Omega + \int_{\Gamma_e} \mathbf{F}_k \mathbf{n}_k \mathbf{W}_h d\Gamma - \int_{\Omega_e} \mathbf{F}_k \frac{\partial \mathbf{W}_h}{\partial x_k} d\Omega = 0, \quad \forall \mathbf{W}_h \in V_h^p \tag{5}$$

where $\mathbf{U}_h$ and $\mathbf{W}_h$ are represented by piecewise polynomial functions of degrees p, which are discontinuous between the cell interfaces, and $\mathbf{n}_k$ the unit outward normal vector to the Γ_e: the boundary of Ω_e. Assume that B_i is the basis of polynomial function of degrees p, this is then equivalent to the following system of N equations,

$$\frac{d}{dt}\int_{\Omega_e} \mathbf{U}_h B_i d\Omega + \int_{\Gamma_e} \mathbf{F}_k \mathbf{n}_k B_i d\Gamma - \int_{\Omega_e} \mathbf{F}_k \frac{\partial B_i}{\partial x_k} d\Omega = 0, \quad 1 \leq i \leq N \tag{6}$$

where N is the dimension of the polynomial space. Since the numerical solution $\mathbf{U}_h$ is discontinuous between element interfaces, the interface fluxes are not uniquely defined. This scheme is called discontinuous Galerkin method of degree p, or in short notation DG(p) method. By simply increasing the degree p of the polynomials, the DG methods of corresponding higher order are obtained. In the present work, the HLLC scheme[4] is used for evaluating the inviscid fluxes.

By moving the second and third terms to the right-hand-side (r.h.s.) in Eq. 6, we will arrive at a system of ordinary differential equations (ODEs) in time, which can be written in semi-discrete form as

$$\boldsymbol{M}\frac{d\mathbf{U}}{dt} = \mathbf{R}(\mathbf{U}) \tag{7}$$

where $\boldsymbol{M}$ is the mass matrix and $\mathbf{R}$ is the residual vector. The present work employs a p-multigrid method, which uses explicit RDG(P1P2), a third-order, WENO reconstructed scheme to improve the overall performance of the underlying second-order DG(P1) method, as high order approximation and implicit DG(P0) as low order approximation, to accelerate the convergence rate.

IV. p-Multigrid Method

Nowadays, geometric multigrid methods are very popular among CFD filed to accelerate the convergence of Euler and Navier-Stokes equations to a steady state on unstructured grids. Analysis of geometric multigrid indicates mesh that independent results are possible, which leads to drastically reduction of the computational cost. p-Multigrid method is a natural extension of geometric multigrid methods to high-order finite element formulation, such as spectral-hp or discontinuous Galerkin methods, where systems of equations are solved by recursively iterating on solution approximations of different polynomial order.

The basic idea of a p-multigrid method is to perform time steps on the lower order approximation levels to obtain the corrections to a solution on a higher order approximation level. Here in the present work, we would only consider RDG(P1P2) method. A two level V-cycle p-multigrid method has been used to drive the iterations. More specifically, this two level p-multigrid method consists of the following steps at each p-multigrid cycle:

(1) Perform a time-step at the high approximation order, that is RDG(P1P2), which yields the initial solution $\mathbf{U}^{n+1}_{P_1P_2}$.

(2) Transfer the flow solution and residual to the low approximation level, that is, DG(P0). This can be readily obtained using the shape function as

$$\mathbf{U}_{P_0}(\Omega_e) = \sum_{i=1}^{N} \mathbf{U}^{n+1}_{P_1P_2} B_i(\mathbf{x}_c) \tag{8}$$

$$\mathbf{R}_{P_0}(\Omega_e) = \sum_{i=1}^{N} \mathbf{R}^{n+1}_{P_1P_2} B_i(\mathbf{x}_c) \tag{9}$$

where $\mathbf{x}_c$ is the coordinates of the center of element Ω_e.

(3) Compute the force terms on the lower approximation level

$$\mathbf{F}_{P_0} = \mathbf{R}_{P_0} - \mathbf{R}(\mathbf{U}_{P_0}) \tag{10}$$

(4) Perform a time-step at the lower approximation level where the residual is given by

$$\mathbf{R} = \mathbf{R}(\mathrm{U}_{P_0}) + \mathbf{F}_{P_0} \tag{11}$$

which yields the solution at the lower level $\mathbf{U}^{n+1}_{P_0}$.

(5) Interpolate the correction $\mathbf{C}_{P_0}$ back from the lower level to update the higher level solution

$$\mathbf{C}_{P_0} = \mathbf{U}^{n+1}_{P_0} - \mathbf{U}_{P_0} \tag{12}$$

$$\widetilde{\mathbf{U}}^{n+1}_{P_1P_2} = \mathbf{U}^{n+1}_{P_1P_2} + \mathbf{C}_{P_0} \tag{13}$$

As for time integration, p-multigrid method allows different schemes for different level approximation. In the present work, we would employ three-stage TVD-Runta explicit method due to the consideration of storage limitation. As a matter of fact, the storage limitation is one of the most important consideration when it comes to GPU computing. As for the lower level approximation, DG(P0), where the storage requirement is not as demanding as in the higher level, first order backward implicit Euler time integration is utilized.

V. OpenACC Implementation

The computation-intensive portion of this reconstructed discontinuous Galerkin method is a time marching loop which repeatedly computes the time derivatives of the conservative variable vector as shown in Eq. 7. In the present work, the conservative variable vector in high approximation level is updated using the explicit, three-stage, TVD Runge-Kutta time stepping scheme[7,8] (TVDRK3) within each time iteration while the lower approximation level uses first order backward Euler implicit time integration to compute the correction. To enable GPU computing, all the required arrays are first allocated on the CPU memory and initialized before the computation enters the main loop. These arrays are then copied to the GPU memory, most of which will not need to be copied back the CPU memory. In fact, the data copy between the CPUs

Table 1: Workflow for the main loop over the time loop.

```
!! loop over time steps
DO itime = 1, ntime

  <ACC> Predict time-step size on higher level

  !! loop over TVDRK stages for RDG(P1P2)
  DO istage = 1, nstage

    <ACC>  P1P2 least-squares reconstruction

    <ACC>  R.H.S. residual from diffusion

    <ACC> WENO reconstruction

    <ACC> residual from convection

    <ACC>  update solution vector

  ENDDO

  !!p-multigrid
  <ACC> Transfer P1->P0

  <ACC> Predict time-step size on lower level

  <ACC> Compute R.H.S. vector R

  <ACC> Compute L.H.S. matirx  A=M/dt-dR/dU

  <ACC> Solve the linear system Au=R
      !!Linear solver + Preconditioner

      ! Option 1: GMRES +  LU-SGS
      ! Option 2: LU-SGS + Jacobi
      ! Option 3: Jacobi
      ! Option 4: ...

   <ACC> Transfer P0->P1 and update

ENDDO
```

and GPUs, usually considered to be one of the major overheads in GPU computing, needs to be minimized in order to improve the efficiency. The workflow of time iterations is outlined in Table 1.

For higher approximation level, the most expensive workload when computing the time derivatives of solutions $d\mathbf{U}/dt$ includes both the reconstruction of the second derivatives and the accumulation of the right hand side residual vector in Eq. 7. Thus these procedures need to be properly ported to acceleration kernels by using the OpenACC parallel construct directives. In fact , the way to add OpenACC directives in a legacy code is very similar to that of OpenMP. The example shown in Table 2 demonstrates the parallelization of a loop over the elements for collecting contribution to the residual vector `rhsel(1:Ndegr,1:Netot,1:Nelem)`, where `Ndegr` is the number of degree of the approximation polynomial (= 1 for P0, 3 for P1 and 6 for P2 in 2D; = 1 for P0, 4 for P1 and 10 for P2 in 3D), `Netot` the number of governing equations of the perfect gas (= 4 in 2D, 5 in 3D), `Nelem` the number of elements, and `Ngp` the number of Gauss quadrature points over an element. Both the OpenMP and OpenACC parallel construct directives can be applied to a readily vectorizable loop like in Table 2 without the need to modify the original code.

As for DG(P0) level, one would need to compute R.H.S. and L.H.S. and thus to apply the linear solver like GMRES[2,28] to solve the linear system. And here the preconditioning matrix P is the approximation of L.H.S. matrix A. However, for unstructured grid, P at least need to be the block-diagonal of A for efficiently preconditioning. For computing the inverse of the preconditioning matrix P, one could only use direct method since the iterative methods are not able to invert the non-diagonal-dominant matrix, which is indeed the common case for unstructured grid. Due to the fact that GMRES involves too many local arrays, which would bring extra data transfer between the host and device, we would not consider porting GMRES algorithm onto the GPU platform for now. As for LU-SGS solver, we would face the memory contention issue for unstructured grids. An elemental based renumbering algorithm would be required to properly thread the LU-SGS onto GPU platform. And since DG(P0) has small degrees of freedom, we would port the simplest Jacobi solver to GPU platform to solve the linear system in lower level approximation for now.

Table 2: An example of loop over the elements.

```
!! OpenMP for CPUs:

!! loop over the elements
!$omp parallel
!$omp do
do ie = 1, Nelem

  !! loop over the Gauss quadrature points
  do ig = 1, Ngp

    !! contribution to this element
    rhsel(*,*,ie) = rhsel(*,*,ie) + flux

  enddo

enddo
!$omp end parallel
```

```
!! OpenACC for GPUs:

!! loop over the elements
!$acc parallel
!$acc loop
do ie = 1, Nelem

  !! loop over the Gauss quadrature points
  do ig = 1, Ngp

    !! contribution to this element
    rhsel(*,*,ie) = rhsel(*,*,ie) + flux

  enddo

enddo
!$acc end parallel
```

However due to the unstructured grid topology, the attempt to directly wrap a loop over the dual-edges for collecting contribution to the residual vector with either the OpenMP or OpenACC directives can lead to the so-called "race condition", that is, multiple writes to the same elemental residual vector, and thus result in the incorrect values. The "race condition" can be eliminated with a moderate amount of work by adopting a mature algorithm of face renumbering and grouping. This algorithm is designed to divide all the faces into a number of groups by ensuring that any two faces that belong to a common element never fall in the same group, so that the face loop in each group can be vectorized without "race condition". An example is shown in Table 3, where an extra do-construct that loops over these groups is nested on top of the original loop over the internal faces, and executed sequentially. The inner do-construct that loops over the internal faces is vectorized without the "race condition" issue.

Table 3: An example of loop over the edges.

```
!! OpenMP for CPUs (without race condition):

!! loop over the groups
Nfac1 = Njfac
do ipass = 1, Npass_ift
  Nfac0 = Nfac1 + 1
  Nfac1 = fpass_ift(ipass)

  !! loop over the edges
  !$omp parallel
  !$omp do
  do ifa = Nfac0, Nfac1

    !! left  element
    iel = intfac(1,ifa)

    !! right element
    ier = intfac(2,ifa)

    !! loop over Gauss quadrature points
    do ig = 1, Ngp

      !! contribution to the left element
      rhsel(*,*,iel) = rhsel(*,*,iel) - flux

      !! contribution to the right element
      rhsel(*,*,ier) = rhsel(*,*,ier) + flux

    enddo

  enddo
  !$omp end parallel

enddo
```

```
!! OpenACC for GPUs (without race condition):

!! loop over the groups
Nfac1 = Njfac
do ipass = 1, Npass_ift
  Nfac0 = Nfac1 + 1
  Nfac1 = fpass_ift(ipass)

  !! loop over the edges
  !$acc parallel
  !$acc do
  do ifa = Nfac0, Nfac1

    !! left  element
    iel = intfac(1,ifa)

    !! right element
    ier = intfac(2,ifa)

    !! loop over Gauss quadrature points
    do ig = 1, Ngp

      !! contribution to the left element
      rhsel(*,*,iel) = rhsel(*,*,iel) - flux

      !! contribution to the right element
      rhsel(*,*,ier) = rhsel(*,*,ier) + flux

    enddo

  enddo
  !$acc end parallel

enddo
```

In fact, this kind of algorithm is widely used for vectorized computing on unstructured grids with OpenMP. The implementation details can be found in an abundance of literature. The number of groups

for each subdomain grid is usually between 6 and 8 according to a wide range of test cases, indicating some overheads in repeatedly launching and terminating the OpenACC acceleration kernels for the loop over the face groups. This kind of overheads is typically associated to GPU computing, but not for the code if parallelized by OpenMP for CPU computing. Nevertheless, the most favorable feature in this design approach is that it allows the original CPU code to be recovered when the OpenACC directives are dismissed in the pre-processing stage of compilation. Therefore, the use of this face renumbering and grouping algorithm will result in a unified source code for both the CPU and GPU computing on unstructured grids.

In a word, the face renumbering and grouping method can suit well in the present work, for its simplicity and portability to quickly adapt into the original source code without any major change in the legacy programming structures. It is applied for the face integrals as well as some other procedures that require the loop over faces like P1P2 least-squares reconstruction and evaluation of the local time-step sizes.

VI. Numerical examples

Performance of the developed GPU code based on OpenACC was measured on the North Carolina State University's research-oriented cluster `ARC`, which has 1728 CPU cores on 108 compute nodes integrated by Advanced HPC. All machines are 2-way SMPs with AMD Opteron 6128 (Magny Core) processors with 8 cores per socket (16 cores per node). The GPGPU card used in the present work is a NVIDIA Tesla K20c GPU containing 2496 multiprocessors. The performance of the equivalent MPI-based parallel CPU program was measured on an AMD Opteron 6128 CPU containing 16 cores. The source code was written in Fortran 90 and compiled with the PGI Accelerator with OpenACC (version 13.9) + OpenMPI (version 1.5.1) development suite. The unit time T_{unit} is calculated as

$$T_{\text{unit}} = \frac{T_{\text{run}}}{\texttt{Ntime} \times \texttt{Nelem}} \times 10^6 \quad \text{(microsecond)}$$

where T_{run} refers to the time recorded for completing the entire time marching loop with a given number of time steps `Ntime`, not including the start-up procedures, initial/end data translation, and solution file dumping.

A. Inviscid flow past a sphere

In the first test case, an inviscid subsonic flow past a sphere at a free-stream Mach number of $M_\infty = 0.5$ is considered in order to assess the performance of OpenACC-based GPU acceleration on implicit DG method. The explicit part can be found in authors' previous work.[33] A sequence of four successively refined tetrahedral grids are displayed in Figs. 1(a) – 1(d). And the surface pressure contours by explicit RDG(P1P2) are shown in Figs. 1(e) – 1(h). The cell size is halved between two consecutive grids. Note that only a quarter of the configuration is modeled due to symmetry of the problem.

The following L^2 norm of the entropy production is used as the error measurement for the steady-state inviscid flow problems:

$$\|\varepsilon\|_{L^2(\Omega)} = \sqrt{\int_\Omega \varepsilon^2 \, d\Omega} = \sqrt{\sum_{i=1}^{Nelem} \int_{\Omega_i} \varepsilon^2 \, d\Omega}$$

where the entropy production ε is defined as

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

Note that the entropy production, where the entropy is defined as $S = (p/\rho)^\gamma$, is a very good criterion to measure accuracy of the numerical solutions, since the flow under consideration is isentropic. The quantitative measurement of the discretization errors has been done in the authors' previous work, which indicates that explicit RDG(P1P2) method achieved a formal order of accuracy of convergence, convincingly demonstrating the benefits of using the RDG method over its underlying baseline DG method.

Secondly a strong scaling timing test is carried out on same sequence of four successively refined tetrahedral grids. The simulations are done by GPU-accelerated DG(P1) or RDG(P1P2) with p-multigrid method and original CPU code, so that we can see the efficiency of the GPU computing. The detailed timing measurements are presented in Table 4, showing the statistics of unit running time. Note that the reason we

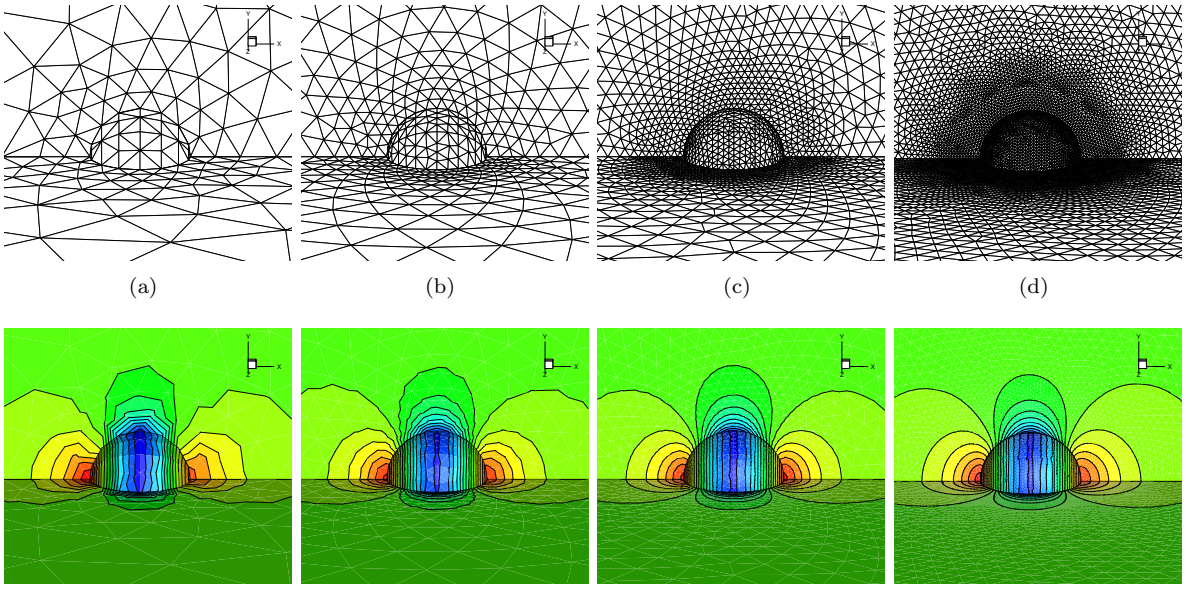


Figure 1: Subsonic flow past a sphere at a free-stream Mach number of $M_\infty = 0.5$: (a) – (d) Surface triangular meshes of the three successively refined tetrahedral grids; (e) – (h) Computed pressure contours obtained by p-multigrid RDG(P1P2) on the surface meshes.

do not present the data for the p-multigrid RDG(P1P2) for the most fine mesh is that it would exceed the K20c GPU card memory. From the results we can see that GPU performs better when it comes to larger `Nelem`, and can achieve higher speedup factor when we use DGP1.

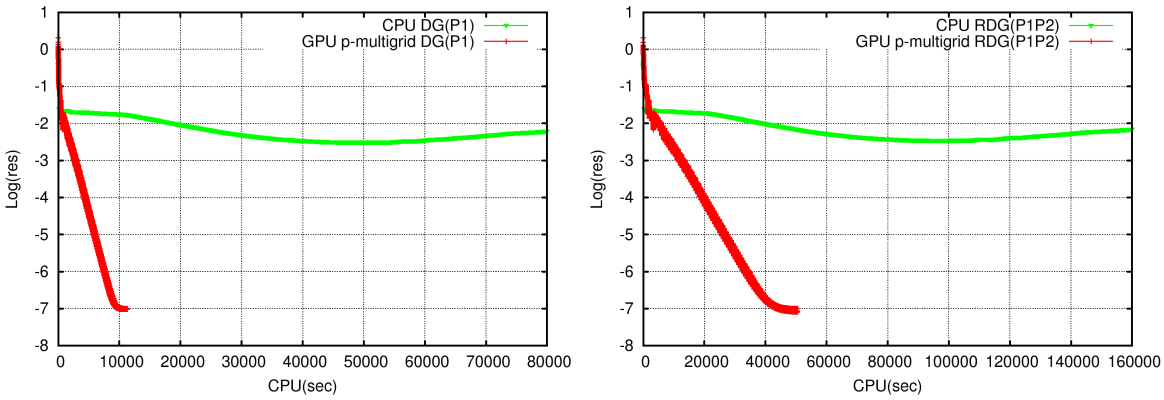
Table 4: Timing measurements of using p-multigrid DG methods for subsonic flow past a sphere.

	T_{unit} by p-multigrid DG(P1)			T_{unit} by p-multigrid RDG(P1P2)		
Nelem	GPU	CPU	Speedup	GPU	CPU	Speedup
2,426	12.53	16.31	1.30	25.93	39.95	1.54
16,467	5.15	19.40	3.77	16.36	49.28	3.01
124,706	3.01	27.15	9.00	13.50	53.24	3.94
966,497	2.02	21.37	10.58	-	-	-

To show the effect of the p-multigrid, we would show the convergence history of the next fine mesh, thus we can include RDG(P1P2) with p-multigrid on GPU platform without the memory issue. The results are shown on Figs. 2(a)-2(b). Clearly, the p-multigrid shows its effect on accelerating the convergence procedure. This case shows that the GPU accelerated p-multigrid DG method is effective and the speed up is promising.

B. Transonic Flow over a Boeing 747 Aircraft

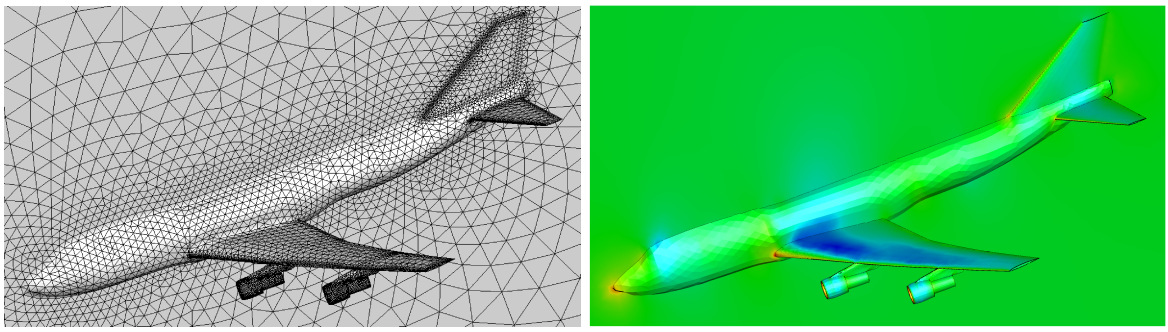
In the second test case, we choose a transonic flow pasting a Boeing 747 aircraft at a free stream Mach number of $M_\infty = 0.85$, and an angle of attack of $\alpha = 2°$. This case could test the ability of computing complex geometric configurations by a OpenACC-based GPU program. The configuration of Boeing 747 includes the fuselage, wing, horizontal and vertical tails, under-wing pylons, and flow-through engine nacelle. The grids we are using here are both tetrahedron grid with 253,577 and 1,025,170 grids respectively. Similarly, we only model half of the aircraft because of the symmetry of the problem. The grid is shown in Fig. 3(a). The



(a) (b)

Figure 2: Subsonic flow past a sphere at a free-stream Mach number of $M_\infty = 0.5$: (a) Comparison of convergence history versus CPU time between TVDRK and p-multigrid methods for DG(P1); (b) Comparison of convergence history versus CPU time between TVDRK and p-multigrid methods for RDG(P1P2)

computed Mach number contours obtained by p-multigrid DG(P1) solution in the flow field are shown in Figs.3(b). Again, the explicit part can be found in authors' previous work.[33]



(a) (b)

Figure 3: Transonic flow past a at a free-stream Mach number of $M_\infty = 0.85$ and a angle of attack of $\alpha = 2°$: (a) Unstructured mesh used for computation; (b) Computed pressure contours obtained by p-multigrid DG(P1)

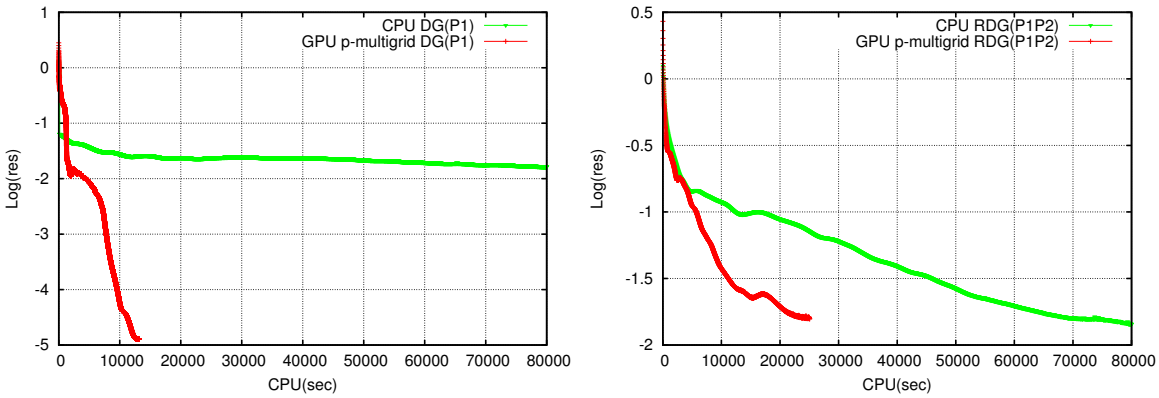
The strong scaling test is carried out on the same grids, which can be found in Table 5. Clearly, we can see that the GPU achieves higher speed up factor with p-multigrid DG(P1). Again, we do not have the data for p -multigrid RDG(P1P2) for the fine mesh due the memory limitation of the available GPU cards. Figs.4(a)-4(b) display a comparison of convergence histories versus CPU time between TVDRKDG on CPU and p-multigrid DG method on GPU platform. The p-multigrid method is much faster than its explicit TVDRK counterpart for this test case. The excellent acceleration of the p-multigrid method is again demonstrated for this flow problem.

VII. Conclusions and Outlook

A GPU accelerated, p-multigrid discontinuous Galerkin method has been developed based on the OpenACC directives for the solution of compressible flows on 3D unstructured grids. A remarkable design feature in the present scheme is that it requires minimum intrusion and algorithm alteration to an existing CPU code, and renders an efficient approach to upgrading a legacy solver with the GPU-computing capability without compromising its cross-platform portability and compatibility with the mainstream compilers. The

Table 5: Timing measurements of using p-multigrid DG methods for transonic flow past a Boeing 747 aircraft.

	T_{unit} by p-multigrid DG(P1)			T_{unit} by p-multigrid RDG(P1P2)		
Nelem	GPU	CPU	Speedup	GPU	CPU	Speedup
253,577	3.47	21.49	6.19	32.94	81.43	2.47
1,025,170	2.61	22.71	8.70	-	-	-



(a) (b)

Figure 4: Transonic flow past a Boeing 747 aircraft at a free-stream Mach number of $M_\infty = 0.85$ and a angle of attack of $\alpha = 2°$: (a) Comparison of convergence history versus CPU time between TVDRK and p-multigrid methods for DG(P1); (b) Comparison of convergence history versus CPU time between TVDRK and p-multigrid methods for RDG(P1P2)

p-multigrid method employes explicit three-stage Runge-Kutta RDG(P1P2) scheme as higher approximation level and first order backward implicit Euler DG(P0) as lower approximation level to accelerate the solver to converge to steady state. A face renumbering and grouping algorithm is used to eliminate memory contention of vectorized computing over the face loops on GPU platform. A series of inviscid flow problems have been presented for the verification the original CPU code for both explicit RDG(P1P2) and p-multigrid method. Further work should be focused on properly extending the current solver to viscous terms so that the scaling test could be taken on both inviscid and viscous flow problems. Additionally, porting the LU-SGS and finally GMRES + LU-SGS solver on the p-multigrid first could be good start to threading the fully implicit DG method on GPU platform.

Acknowledgments

The authors would like to acknowledge the support for this work provided by the Basic Research Initiative program of The Air Force Office of Scientific Research. Dr. F. Fariba and Dr. D. Smith serve as the technical monitors.

References

[1]V. G. Asouti, X. S. Trompoukis, I. C. Kampolis, and K. C. Giannakoglou. Unsteady cfd computations using vertex-centered finite volumes for unstructured grids on graphics processing units. *International Journal for Numerical Methods in Fluids*, 67(2):232–246, 2011.

[2]F. Bassi and S. Rebay. GMRES discontinuous Galerkin solution of the Compressible Navier-Stokes Equations. *Discontinuous Galerkin Methods, Theory, Computation, and Applications. Edited by B. Cockburn, G. E. Karniadakis, and C. W. Shu. Lecture Notes in Computational Science and Engineering*, 11:197–208, 2000.

[3]F. Bassi and S. Rebay. Discontinuous Galerkin Solution of the Reynolds-Averaged Navier-Stokes and κ-ω Turbulence Model Equations. *Computers & Fluids*, 34(4-5):507–540, 2005.

[4]P. Batten, M. A. Leschziner, and U. C. Goldberg. Average-State Jacobians and Implicit Methods for Compressible Viscous and Turbulent Flows. *Journal of Computational Physics*, 137(1):38–78, 1997.

[5]T. Brandvik and G. Pullan. Acceleration of a two-dimensional euler flow solver using commodity graphics hardware. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 221(12):1745–1748, 2007.

[6]T. Brandvik and G. Pullan. Acceleration of a 3d euler solver using commodity graphics hardware. In *46th AIAA aerospace sciences meeting and exhibit*, pages 2008–607, 2008.

[7]B. Cockburn, S. Hou, and C. W. Shu. TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for conservation laws IV: the Multidimensional Case. *Journal of Mathematical Physics*, 55:545–581, 1990.

[8]B. Cockburn and C. W. Shu. The Runge-Kutta Discontinuous Galerkin Method for conservation laws V: Multidimensional System. *Journal of Computational Physics*, 141:199–224, 1998.

[9]J. Cohen and M. J. Molemaker. A fast double precision cfd code using cuda. *Parallel Computational Fluid Dynamics: Recent Advances and Future Directions*, pages 414–429, 2009.

[10]A. Corrigan, F. Camelli, R. Löhner, and F. Mut. Porting of an edge-based cfd solver to gpus. In *48th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, 2010.

[11]A. Corrigan, F. Camelli, R. Löhner, and F. Mut. Semi-automatic porting of a large-scale fortran cfd code to gpus. *International Journal for Numerical Methods in Fluids*, 69(2):314–331, 2012.

[12]A. Corrigan, F. Camelli, R. Löhner, and J. Wallin. Running unstructured grid-based CFD solvers on modern graphics hardware. *International Journal for Numerical Methods in Fluids*, 66(2):221–229, 2011.

[13]E. Elsen, P. LeGresley, and E. Darve. Large calculation of the flow over a hypersonic vehicle using a gpu. *Journal of Computational Physics*, 227(24):10148–10161, 2008.

[14]K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal. *p*-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible NavierStokes equations. *Journal of Computational Physics*, 207(1):92–113, 2005.

[15]D. Goddeke, S. HM Buijssen, H. Wobker, and S. Turek. Gpu acceleration of an unmodified parallel finite element navier-stokes solver. In *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, pages 12–21. IEEE, 2009.

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

[17]D. A. Jacobsen, J. C. Thibault, and I. Senocak. An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters. In *48th AIAA Aerospace Sciences Meeting and Exhibit*, volume 16, 2010.

[18]D. C. Jespersen. Acceleration of a cfd code with a gpu. *Scientific Programming*, 18(3):193–201, 2010.

[19]H. Jin, M. Kellogg, and P. Mehrotra. Using compiler directives for accelerating CFD applications on GPUs. In *OpenMP in a Heterogeneous World*, pages 154–168. Springer, 2012.

[20]A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven. Nodal discontinuous galerkin methods on graphics processors. *Journal of Computational Physics*, 228(21):7863–7882, 2009.

[21]H. Luo, J. D. Baum, and R. Löhner. A fast, p-multigrid discontinuous Galerkin method for compressible flows at all speeds. *AIAA Paper*, 110:2006, 2006.

[22]H. Luo, J. D. Baum, and R. Löhner. A *p*-Multigrid Discontinuous Galerkin Method for the Euler Equations on Unstructured Grids. *Journal of Computational Physics*, 211(2):767–783, 2006.

[23]H. Luo, Y. Xia, S. Li, and R. Nourgaliev. A Hermite WENO Reconstruction-Based Discontinuous Galerkin Method for the Euler Equations on Tetrahedral grids. *Journal of Computational Physics*, 231(16):5489–5503, 2012.

[24]H. Luo, Y. Xia, S. Spiegel, R. Nourgaliev, and Z. Jiang. A reconstructed discontinuous Galerkin method based on a hierarchical WENO reconstruction for compressible flows on tetrahedral grids. *Journal of Computational Physics*, 236:477–492, 2013.

[25]D. Michéa and D. Komatitsch. Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International*, 182(1):389–402, 2010.

[26]J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

[27]E. H. Phillips, Y. Zhang, R. L. Davis, and J. D. Owens. Rapid aerodynamic performance prediction on a cluster of graphics processing units. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting, number AIAA*, volume 565, 2009.

[28]Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[29]J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.

[30]J. C. Thibault and I. Senocak. Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting*, pages 2009–758, 2009.

[31]S. Wienke, P. Springer, C. Terboven, and D. Mey. Openaccfirst experiences with real-world applications. In *Euro-Par 2012 Parallel Processing*, pages 859–870. Springer, 2012.

[32]Y. Xia, M. Frisbey, H. Luo, and R. Nourgaliev. A WENO Reconstruction-Based Discontinuous Galerkin Method for Compressible Flows on Hybrid Grids. *AIAA Paper*, 2013-0516, 2013.

[33]Y. Xia, H. Luo, L. Luo, J. Edwards, J. Lou, and F. Mueller. OpenACC-based GPU Acceleration of a 3-D Unstructured Discontinuous Galerkin Method. *AIAA Paper*, 2014-1129, 2014. http://arc.aiaa.org/doi/abs/10.2514/6.2014-1129.

[34]Y. Xia, H. Luo, and R. Nourgaliev. An Implicit Reconstructed Discontinuous Galerkin Method Based on Automatic Differentiation for the Navier-Stokes Equations on Tetrahedron Grids. *AIAA Paper*, 2013-0687, 2013.

[35]Y. Xia, H. Luo, S. Spiegel, M. Frisbey, and R. Nourgaliev. A Parallel, Implicit Reconstruction-Based Hermite-WENO Discontinuous Galerkin Method for the Compressible Flows on 3D Arbitrary Grids. *AIAA Paper*, submitted, in process, 2013.

[36]Y. Xia and R. Nourgaliev. An Implicit Hermite WENO Reconstruction-Based Discontinuous Galerkin Method on Tetrahedral Grids. *7th International Conference on Computational Fluid Dynamics*, ICCFD7-4205, 2012.

[37]B. Zimmerman, Z. Wang, and M. Visbal. High-Order Spectral Difference: Verification and Acceleration using GPU Computing. 2013-2491, 2013.