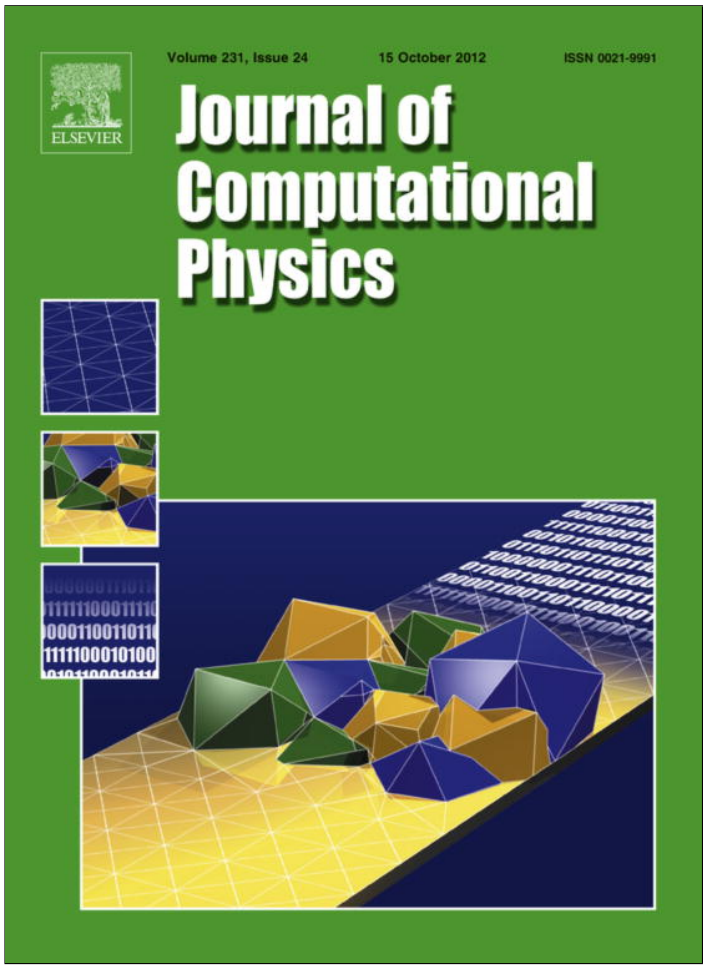
2012

# Continuous and Discontinuous Galerkin Methods for a Scalable 3D Nonhydrostatic Atmospheric Model: limited-area mode



Journal of Computational Physics 231 (2012) 7988–8008

Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

# Continuous and discontinuous Galerkin methods for a scalable three-dimensional nonhydrostatic atmospheric model: Limited-area mode

James F. Kelly, Francis X. Giraldo *

*Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA, United States*

ARTICLE INFO

*Article history:*
Received 15 April 2011
Received in revised form 10 March 2012
Accepted 30 April 2012
Available online 26 May 2012

*Keywords:*
Compressible flow
Element-based Galerkin method
Euler
Navier–Stokes
Parallelization

ABSTRACT

This paper describes a unified, element based Galerkin (EBG) framework for a three-dimensional, nonhydrostatic model for the atmosphere. In general, EBG methods possess high-order accuracy, geometric flexibility, excellent dispersion properties and good scalability. Our nonhydrostatic model, based on the compressible Euler equations, is appropriate for both limited-area and global atmospheric simulations. Both a continuous Galerkin (CG), or spectral element, and discontinuous Galerkin (DG) model are considered using hexahedral elements. The formulation is suitable for both global and limited-area atmospheric modeling, although we restrict our attention to 3D limited-area phenomena in this study; global atmospheric simulations will be presented in a follow-up paper. Domain decomposition and communication algorithms used by both our CG and DG models are presented. The communication volume and exchange algorithms for CG and DG are compared and contrasted. Numerical verification of the model was performed using two test cases: flow past a 3D mountain and buoyant convection of a bubble in a neutral atmosphere; these tests indicate that both CG and DG can simulate the necessary physics of dry atmospheric dynamics. Scalability of both methods is shown up to 8192 CPU cores, with near ideal scaling for DG up to 32,768 cores.



## 1. Introduction

As the resolution of numerical weather prediction (NWP) models increase, nonhydrostatic effects become relevant. Almost all current limited-area models use a nonhydrostatic core, while global models are currently transitioning from the hydrostatic to the nonhydrostatic regime. A host of challenging, non-trivial numerical problems arise when one enters the nonhydrostatic regime: these include (1) choosing the appropriate equation set to ensure efficiency, accuracy, and conservation, (2) effectively resolving multi-scale flow features, that may require adaptive mesh refinement (AMR), (3) developing efficient time-integrators and/or "soundproof" [24,31] equation sets to confront the fast acoustic and gravity waves, and (4) developing scalable parallel codes for shared, distributed, and hybrid architectures based on items (1)–(3).

Virtually all current nonhydrostatic NWP models are based on a combination of finite-difference discretization in space and either split-explicit or semi-implicit (i.e., implicit-explicit or IMEX) discretization in time. Examples include WRF [25] (NCAR), Lokal Modell [33] (DWD), COAMPS [20] (US Navy), and UM [4] (UK Met Office). Although finite difference schemes are very efficient, they may suffer from several problems, including: dispersion error (due to low-order approximations), geometric inflexibility, and lack of scalability to large (e.g. tens of thousands) of processors. To overcome some of the limitations of finite-difference approximations, several emerging NWP models use finite-volume spatial discretization, such

* Corresponding author.
*E-mail addresses:* giraldo@nrlmry.navy.mil, fxgirald@nps.edu (F.X. Giraldo).

0021-9991/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcp.2012.04.042

as MPAS [40] and MCORE [41], which employ polynomial reconstruction of the inter-element fluxes. In order to achieve high-order accuracy, these finite-volume based methods use a larger halo, which may impede scalability at high processor counts. Other approaches within a finite volume framework include the flux-based characteristic semi-Lagrangian (FBCSL) method [30], which uses the method of characteristics to enable larger time-steps that may present an obstacle to scalability.

Alternatively, element-based Galerkin (EBG) methods have recently been proposed for several next generation nonhydrostatic NWP models [15,16,32], as well as for hydrostatic models using both continuous Galerkin (CG) [8,7] and discontinuous Galerkin (DG) [29] formulations. We note that the hydrostatic models are only valid for horizontal resolutions coarser than 10 km and for this reason are not viable options for mesoscale (regional) modeling. EBG methods possess several desirable attributes for future NWP nonhydrostatic models, such as high-order accuracy, geometric flexibility, whereby the solver is completely independent of the grid, excellent dispersion properties [27], and minimal communication overhead within a parallel implementation. High-order accuracy, which allows fine-scale atmospheric flow features to be resolved, is achieved by representing the prognostic variables via a polynomial basis function expansion within each element. Geometric flexibility, which is inherent to all EBGs (both low and high order), is advantageous since any terrain-following coordinate may be used within the same solver; also, both static and dynamic adaptivity may be retro-fitted to the existing solver.

Finally, in a distributed memory environment (e.g. cluster), low communication overhead is critical for achieving linear scalability to hundreds of thousands of processor cores. In our previous work, high-order accuracy and geometric flexibility were addressed within explicit [15] and IMEX [16] frameworks for 2D ($x$–$z$ slices) problems using a serial implementation. However, parallel implementation was not explicitly addressed. The purpose of the present work is to extend the work begun in [15,32,16] to realistic three-dimensional (3D) domains using a parallel, MPI-based implementation.

The present work is guided by a need for highly scalable models in distributed memory environments. In the past five years, clock speeds of processors have remained stagnant; to achieve increased floating-point performance, chip manufacturers have developed multiple core processors that allow many threads to execute in parallel. In tandem, high performance computing (HPC) has evolved towards clusters with processors counts exceeding 100,000. As we approach the exaflop era, core counts are expected to approach 1,000,000. Therefore, next-generation NWP models must be based upon scalable numerical methods that allow arbitrarily large processor counts with minimal communication overhead. EBG methods, both CG and DG, have proved effective in this respect in modeling biological flow [18] (DG), the hydrostatic atmosphere [12,17] (CG), incompressible flow [21] (low-order CG), and geodynamical problems [42] (DG).

This paper presents a scalable, 3D nonhydrostatic atmospheric model based on a unified element based Galerkin (EBG) method targeted toward distributed memory architectures; to our knowledge, these are the first 3D continuous Galerkin (spectral element) and discontinuous Galerkin models for a nonhydrostatic atmosphere. We are developing a unified dynamical core appropriate for both mesoscale and global simulations; this nonhydrostatic unified model of the atmosphere we call NUMA. The current implementation uses tensor products of Lagrange polynomials in a hexahedral grid for maximum computational efficiency; however, more flexible grids based on either tetrahedral or triangular prisms may be incorporated into future versions. The remainder of this paper is structured as follows. In Section 2, we formulate the nonhydrostatic compressible Euler equations (set 2NC and 2C from [16]), which constitute the governing equations of our dynamical core. To ensure a well-balanced method, these nonhydrostatic equations are solved about a hydrostatic base state. In Section 3, we present both the continuous and discontinuous Galerkin discretization, along with the explicit time-integrator, boundary conditions, and artificial diffusion. Section 4, which forms the core of the paper, outlines the parallelization algorithm for both the CG and DG methods. The communication volume and memory access patterns of both CG and DG are compared in this section. Numerical results for a 3D linear hydrostatic mountain and a 3D rising thermal bubble are shown in Section 5 along with the results of the scalability experiments. Perfect scalability is demonstrated to 8192 processor cores for both the CG and DG methods, with near ideal scaling for DG up to 32,768 cores.

## 2. Governing equations

We consider the fully compressible, nonhydrostatic Euler equations in both conservative and non-conservative form. These equations (sets 2NC and 2C), which are valid for spatial resolutions finer than 10 km, have previously been considered in [15] for 2D limited-area atmospheric flows. Set 2NC is considered within a continuous Galerkin (CG) framework, whereas set 2C is considered within a discontinuous Galerkin (DG) framework.

### 2.1. Non-conservative form (set 2NC)

Of the five equation sets considered in [16], set 2NC proved to be both computationally efficient and provides acceptable mass and energy conservation properties; in addition, replacing the advection operator in the momentum equation allows for formal conservation of energy up to time truncation error. Both diabatic forcing and the effects of moisture are neglected; in other words, we consider a *dry* dynamical core without sub-grid scale turbulence closure. In the present study, we consider three-dimensional flow in Cartesian coordinates ($x$–$y$–$z$) subject to gravitational and Coriolis forces, yielding

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} + \frac{1}{\rho}\nabla P + g\hat{\mathbf{k}} + \mathbf{f}\times\mathbf{u} = 0 \tag{1b}$$

$$\frac{\partial \theta}{\partial t} + \mathbf{u}\cdot\nabla\theta = 0 \tag{1c}$$

with the prognostic variables defined as $(\rho, \mathbf{u}^{\mathcal{T}}, \theta)$, where $\rho$ is density, $\mathbf{u} = (u, v, w)^{\mathcal{T}}$ is velocity, and $\theta$ is potential temperature; the superscript $\mathcal{T}$ denotes the transpose operator. In addition, $P$ is pressure, $g$ is the gravitational constant, $\mathbf{f} = 2\Omega\hat{\mathbf{k}}$ is the Coriolis parameter (with $\Omega$ the angular frequency of the earth), and $\hat{\mathbf{k}}$ is the unit vector in the $z$ direction. Eq. (1a) enforces mass conservation, Eq. (1b) enforces conservation of momentum, and Eq. (1c) enforces conservation of entropy. To close the system of conservation laws given by Eq. (1), a thermodynamic equation of state is required. We use the ideal gas law given by

$$P = P_A\left(\frac{\rho R\theta}{P_A}\right)^{\gamma} \tag{2}$$

where $P_A$ is the atmospheric pressure at ground, $R = c_p - c_v$ is the ideal gas constant, and $\gamma \equiv \frac{c_p}{c_v} \approx 1.4$ is the ratio of specific heats. In three dimensions, Eqs. (1) and (2) constitute a closed system of nonlinear partial differential equations.

To facilitate the solution of the compressible Euler equations and maintain numerical stability, we split the density, pressure, and potential temperature about their mean hydrostatic values:

$$\rho(x,z,y,t) = \rho_0(z) + \rho'(x,y,z,t) \tag{3a}$$

$$\theta(x,y,z,t) = \theta_0(z) + \theta'(x,y,z,t) \tag{3b}$$

$$P(x,y,z,t) = P_0(z) + P'(x,y,z,t) \tag{3c}$$

where $\rho_0,\ \theta_0$, and $P_0$ are the hydrostatic reference states. For all the test cases considered in this paper, the reference states are functions of the vertical coordinate $z$; however, more sophisticated test cases require reference states that are functions of $x,\ y$, and $z$. Inserting Eq. (3) into Eq. (1) and applying the hydrostatic balance

$$\frac{dP_0}{dz} = -\rho_0 g \tag{4}$$

yields the system

$$\frac{\partial \rho'}{\partial t} + \mathbf{u}\cdot\nabla\rho' + \mathbf{u}\cdot\nabla\rho_0 + (\rho' + \rho_0)\nabla\cdot\mathbf{u} = 0 \tag{5a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} + \frac{1}{\rho' + \rho_0}\nabla P' + \frac{\rho'}{\rho' + \rho_0}g\hat{\mathbf{k}} + \mathbf{f}\times\mathbf{u} = 0 \tag{5b}$$

$$\frac{\partial \theta'}{\partial t} + \mathbf{u}\cdot\nabla\theta' + \mathbf{u}\cdot\nabla\theta_0 = 0. \tag{5c}$$

Defining a solution vector $\mathbf{q} = (\rho', \mathbf{u}^{\mathcal{T}}, \theta')^{\mathcal{T}}$, allows us to write Eq. (5) in the condensed form

$$\frac{\partial \mathbf{q}}{\partial t} = S^{2\mathrm{NC}}(\mathbf{q}) \tag{6}$$

where $S^{2\mathrm{NC}}(\mathbf{q})$ is a nonlinear, first-order differential operator.

### 2.2. Conservative form (set 2C)

Eq. (1) may be written in flux, or conservative form

$$\frac{\partial \rho}{\partial t} + \nabla\cdot\mathbf{U} = 0 \tag{7a}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla\cdot\left(\frac{\mathbf{U}\otimes\mathbf{U}}{\rho} + P\mathbf{I}_3\right) + \rho g\hat{\mathbf{k}} + \mathbf{f}\times\mathbf{U} = 0 \tag{7b}$$

$$\frac{\partial \Theta}{\partial t} + \nabla\cdot\left(\frac{\Theta\mathbf{U}}{\rho}\right) = 0 \tag{7c}$$

complemented by the equation of state

$$P = P_A\left(\frac{R\Theta}{P_A}\right)^{\gamma} \tag{8}$$

with the prognostic variables defined as $(\rho, \mathbf{U}^T, \Theta)^T$, where $\rho$ is density, $\mathbf{U} = \rho\mathbf{u}$ is momentum, $\mathbf{u} = (u, v, w)^T$ is velocity, and $\Theta = \rho\theta$ is density potential temperature. In addition, the operator $\otimes$ denotes the tensor product and $\mathbf{I}_3$ is the rank-3 identity matrix. The variables in these equations are split in a similar fashion to Eq. (1) except that we now split the density potential temperature as follows

$$\Theta(x, y, z, t) = \Theta_0(z) + \Theta'(x, y, z, t) \tag{9}$$

Applying the hydrostatic decomposition to Eq. (7) yields

$$\frac{\partial \rho'}{\partial t} + \nabla \cdot \mathbf{U} = 0 \tag{10a}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \left( \frac{\mathbf{U} \otimes \mathbf{U}}{\rho} + P' \mathbf{I}_3 \right) + \rho' g \hat{\mathbf{k}} + \mathbf{f} \times \mathbf{U} = 0 \tag{10b}$$

$$\frac{\partial \Theta'}{\partial t} + \nabla \cdot \left( \frac{\Theta \mathbf{U}}{\rho} \right) = 0 \tag{10c}$$

which may be written in vector form as

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = S(\mathbf{q}) \tag{11}$$

where $\mathbf{F}(\mathbf{q})$ is the flux tensor and the gravitational and Coriolis terms are incorporated into $S(\mathbf{q})$. Finally, we can write this equation in the condensed form

$$\frac{\partial \mathbf{q}}{\partial t} = S^{2C}(\mathbf{q}) \tag{12}$$

## 3. Numerical methods

In this section, we briefly discuss the numerical discretization used by NUMA which includes: the spatial discretization of Eq. (1) using a continuous Galerkin (CG) method and Eq. (7) using a discontinuous Galerkin (DG) method (Section 3.1); the explicit time-integrator used to evolve the solution forward in time (Section 3.2); the necessary boundary conditions required for limited-area problems (Section 3.3); and the artificial diffusion used to control overshoots (Section 3.4).

### 3.1. Spatial discretization

Let us now describe the approximation of the continuous spatial operators using both CG and DG. We begin with a description of the decomposition of the global spatial domain into elements, the basis functions defined within these elements, and element-wise integrals that are common to both CG and DG.

#### 3.1.1. Basis functions, elements, and integrals

Element-based Galerkin methods such as the continuous Galerkin and discontinuous Galerkin methods require the decomposition of the global domain $\Omega \subset \mathcal{R}^3$ into $N_e$ non-overlapping elements $\Omega_e$ via

$$\Omega = \bigcup_{e=1}^{N_e} \Omega_e \tag{13}$$

In the current formulation of NUMA, we let $\Omega_e$ be hexahedra, which provides simple grid generation and efficient (fast) evaluation of the necessary differentiation and integration operators. We note, however, that $\Omega_e$ may be replaced by tetrahedral, pyramidal elements, or triangular prisms; while this can be done for both CG and DG, we envision doing this only for the DG version since DG does not require global matrices. Assuming a purely explicit scheme, CG would require the inversion of a sparse global mass matrix which becomes prohibitive in three-dimensions for very large problems (this is the case because no collocated set of interpolation and integration points, that results in a diagonal mass matrix, exists for triangular prisms or tetrahedra). We also note that tetrahedral and pyramidal elements would produce an unstructured grid in the vertical, which would impede the incorporation of column-based microphysics and physical parameterizations. Finally, the tensor product structure may be exploited with hexahedral elements, reducing the complexity of a 3D method based on $N$-th order polynomials from $\mathcal{O}(N^6)$ to $\mathcal{O}(N^4)$, thereby increasing the efficiency of EBG methods by one to three orders of magnitude for typical values of $N \in [3, 10]$.

Letting the unit cube $(\xi, \eta, \zeta) \in E = [-1, 1]^3$ be the reference hexahedral element, a transformation $\mathcal{F}_e : \Omega_e \to E$ mapping physical space to computational space is defined for each element, yielding $(x, y, z) = \mathcal{F}_e(\xi, \eta, \zeta)$ where the associated Jacobian of $\mathcal{F}_e$ is denoted by $J_e$.

Within each element $\Omega_e$, a finite-dimensional approximation $\mathbf{q}_N$ is formed by expanding $\mathbf{q}(\mathbf{x},t)$ in basis functions $\psi_j(\mathbf{x})$ such that

$$\mathbf{q}_N^{(e)}(\mathbf{x},t) = \sum_{j=1}^{M_N} \psi_j(\mathbf{x}) \mathbf{q}_j^{(e)}(t) \tag{14}$$

where $M_N = (N+1)^3$ is the number of nodes per element, $N$ is the order of the basis functions, and the superscript $(e)$ denotes element-wise (local) values. For basis functions, we construct tensor products of Lagrange polynomials given by

$$\psi_i(\mathbf{x}) = h_\alpha(\xi) \otimes h_\beta(\eta) \otimes h_\gamma(\zeta) \tag{15}$$

where $h_\alpha(\xi)$ is the Lagrange polynomial associated with the Legendre–Gauss–Lobatto (LGL) points $\xi_i$ and $(\xi,\eta,\zeta)$ are functions of the physical variable $\mathbf{x}$. These LGL points satisfy

$$(1-\xi^2) P_N'(\xi) = 0 \tag{16}$$

where $P_N(\xi)$ is the $N$-th order Legendre polynomial. Hence, we are utilizing nodal basis functions, as opposed to modal basis functions (e.g. Legendre polynomials). We can now use all of this machinery defined to construct discrete approximations of the continuous spatial derivatives. For example, the derivative of $\mathbf{q}$ can be obtained by differentiating equation (14) as follows

$$\nabla \mathbf{q}_N^{(e)}(\mathbf{x},t) = \sum_{j=1}^{M_N} \nabla \psi_j(\mathbf{x}) \mathbf{q}_j^{(e)}(t) \tag{17}$$

which, after multiplying by the test (basis) function $\psi$ and integrating within an element yields

$$\int_{\Omega_e} \psi_i \sum_{j=1}^{M_N} \nabla \psi_j(\mathbf{x}) \mathbf{q}_j^{(e)}(t) d\Omega_e \tag{18}$$

where we shall use Nth-degree LGL integration (numerical quadrature) points for approximating the integral. We note that aliasing errors arise due to an insufficient number of integration points in Eq. (18). To remove these aliasing errors, a modal low-pass filter is used [2].

### 3.1.2. Continuous Galerkin

For CG we shall use Eq. (1) as our governing equations. Using Eq. (14) to approximate $\mathbf{q}_N$, multiplying by a test function and integrating as in Eq. (18) yields the element-wise definition of the problem

$$\int_{\Omega_e} \psi_i \frac{\partial \mathbf{q}_N}{\partial t} d\Omega_e = \int_{\Omega_e} \psi_i S^{2\mathrm{NC}}(\mathbf{q}_N) d\Omega_e \tag{19}$$

and applying the global assembly, or direct stiffness summation (DSS) operator required by all CG methods yields the weak formulation for CG: find $\mathbf{q}_N \in \mathcal{V}_N^{\mathrm{CG}}$ such that

$$\int_{\Omega} \psi_I \frac{\partial \mathbf{q}_N}{\partial t} d\Omega = \int_{\Omega} \psi_I S^{2\mathrm{NC}}(\mathbf{q}_N) d\Omega \quad \forall \psi \in \mathcal{V}_N^{\mathrm{CG}} \tag{20}$$

where

$$\mathcal{V}_N^{\mathrm{CG}} = \left\{ \psi \in H^1(\Omega) | \psi \in P^N(I), \quad e = 1, \ldots, N_e \right\} \tag{21}$$

and $P^N$ denotes the space of all polynomials of degree $N$ in the interval $I = [-1,1]$. Notice that the requirement $\psi \in H^1(\Omega)$ implies $V_N^{\mathrm{CG}} \subset C^0(\Omega)$. The DSS operator $\bigwedge_{e=1}^{N_e}$ forms global matrices from the element (local) matrices. For example, for the local mass matrix defined as

$$M_{ij}^{(e)} = \int_{\Omega_e} \psi_i \psi_j d\Omega_e \tag{22}$$

the DSS operator produces the global mass matrix

$$M_{IJ} = \bigwedge_{e=1}^{N_e} M_{ij}^{(e)} \equiv \bigwedge_{e=1}^{N_e} \int_{\Omega_e} \psi_i \psi_j d\Omega_e \tag{23}$$

where the DSS sums the contribution of all the elements $e = 1, \ldots, N_e$ and all interpolation points within the elements $i = 1, \ldots, M_N$ and stores them in the global grid points $I = 1, \ldots, N_P$ as follows $(i,e) \to I$. Extracting the global mass matrix from Eq. (19) and writing the operator $S$ as a vector, results in the following compact matrix–vector form

$$\frac{d\mathbf{q}_I}{dt} = M_{IJ}^{-1} S_J^{\mathrm{CG}}(\mathbf{q}) \tag{24}$$

where we now use the total derivative with respect to time to emphasize that $\mathbf{q}_I = \mathbf{q}_I(t)$ represents the expansion coefficients that are only functions of time. In addition, note that the mass matrix $M_{IJ} = \int_\Omega \psi_I \psi_J d\Omega$ is diagonal because the interpolation and integration points have been deliberately chosen to be co-located; this approach incurs negligible error for $N \geqslant 4$ [14]. Denoting the right-hand side (RHS) of Eq. (24) by $R_I(\mathbf{q})$, Eq. (24) is expressed as

$$\frac{d\mathbf{q}_I}{dt} = \bigwedge_{e=1}^{N_e} R_i^{(e)}\left(\mathbf{q}_i^{(e)}\right) \tag{25}$$

Note that the DSS operator maps local, element-wise coordinates $(i,e)$ to global coordinates $I$. Eq. (25) forms the core of the spectral element method, allowing local, element-wise information $\mathbf{q}_i^{(e)}$ to propagate to adjacent elements via the DSS operator.

### 3.1.3. Discontinuous Galerkin

For DG we use Eq. (7) because the DG method requires the equations to be in conservation form. We have also developed a CG code with set 2C but we use set 2NC for CG because it represents the optimal equation set for our needs as described in [16].

Beginning with Eq. (11), using the basis function expansion, Eq. (14), multiplying by a test function, and integrating element-wise yields

$$\int_{\Omega_e} \psi_i \frac{\partial \mathbf{q}_N^{(e)}}{\partial t} d\Omega_e + \int_{\Omega_e} \psi_i \nabla \cdot \mathbf{F}^{(e)}(\mathbf{q}_N) d\Omega_e = \int_{\Omega_e} \psi_i S^{(e)}(\mathbf{q}_N) d\Omega_e \tag{26}$$

Integrating the divergence of the flux by parts yields the weak formulation for DG: find $\mathbf{q}_N^{(e)} \in \mathcal{V}_N^{\mathrm{DG}}$

$$\int_{\Omega_e} \psi_i \frac{\partial \mathbf{q}_N^{(e)}}{\partial t} d\Omega_e + \sum_{k=1}^{N_{faces}} \int_{\Gamma_e} \psi_i \hat{\mathbf{n}}^{(e,k)} \cdot \mathbf{F}^{(e,k)}(\mathbf{q}_N) d\Gamma_e - \int_{\Omega_e} \nabla \psi_i \cdot \mathbf{F}^{(e)}(\mathbf{q}_N) d\Omega_e = \int_{\Omega_e} \psi_i S^{(e)}(\mathbf{q}_N) d\Omega_e \quad \forall \psi \in \mathcal{V}_N^{DG} \tag{27}$$

where the DG finite-dimensional space is defined as

$$\mathcal{V}_N^{\mathrm{DG}} = \left\{ \psi \in L^2(\Omega) | \psi \in P^N(I), \quad e = 1, \ldots, N_e \right\} \tag{28}$$

Notice that, contrary to Eq. (21), there is no global continuity requirement, so that $\mathcal{V}_N^{\mathrm{DG}} \not\subset C^0(\Omega)$. This is possible because in Eq. (27) differentiability is required separately within each element $\Omega_e$, and not within the entire domain $\Omega$. The coupling between neighboring elements is then recovered through the numerical flux (or Riemann solver) $\mathbf{F}^{(e,k)}$, which is required to be a single valued function on the inter-element boundaries.

We use the Rusanov flux defined as

$$\mathbf{F}^{(e,k)} = \frac{1}{2}\left[\mathbf{F}^{(e)} + \mathbf{F}^{(k)} - \hat{\mathbf{n}}^{(e,k)} c_{max}\left(\mathbf{q}^{(k)} - \mathbf{q}^{(e)}\right)\right] \tag{29}$$

where $\hat{\mathbf{n}}^{(e,k)}$ is the unit normal vector pointing from the element $e$ to the neighboring element $k$ and $c_{max}$ is the maximum wave speed of the Euler equations (i.e., maximum eigenvalue of the Jacobian matrix) which turns out to be $c_{max} = |\hat{\mathbf{n}} \cdot \mathbf{u}| + a$ where $a$ denotes the speed of sound. Note that other types of numerical fluxes are possible including multi-dimensional Riemann solvers (e.g. [10]). We have initially chosen this simple flux function because it vastly simplifies the parallelization strategy. Using a one-dimensional flux, as we have, reduces the element communication to edge neighbors but restricts the maximum allowable time-step. Conversely, using a multi-dimensional Riemann solver will increase the maximum allowable time-step but will also increase the size of the communication stencil. The vices and virtues of this approach need to be studied in detail and we reserve this for future work.

## 3.2. Explicit RK methods

We use the strong stability preserving (SSP) Runge–Kutta third-order, five stage time-integrator (RK35) proposed in [38]. This time-integrator is stable for (acoustic) Courant numbers of 1.3 or less when using a continuous Galerkin discretization, whereas for DG, the time-integrator is stable for (acoustic) Courant numbers of 0.85 or less.

We emphasize that this explicit RK method will not be used in an operational setting due to the stringent CFL requirement; rather, we are developing IMEX methods which allow much larger time-steps. Nevertheless, this particular time integrator was chosen for this study for the following reasons: (1) to provide high-order accuracy in time to complement the high-order spatial discretization, (2) to minimize the amount of numerical dissipation, (3) to allow a relatively larger time-step (with respect to other explicit methods), and (4) to provide stable solutions for both CG and DG methods. Criteria (1) and (2) are necessary for resolving fine-scale flow features present in many atmospheric phenomena, whereas criterion (3) is necessary for model efficiency, and criterion (4) is required in order to facilitate the comparison of CG and DG. In a previous study involving nonhydrostatic modeling [6], all available third-order multi-stage methods in addition to an assortment of multi-step methods were compared and RK35 was found to be the most efficient. In addition, RK35, unlike leapfrog, does not have a computational mode and therefore does not require a time filter (DG also would not work with leapfrog due

to the eigenvalues for DG not residing along the imaginary axis as they do for CG). Note, however, that within a parallel setting, each RK stage requires parallel communication, thus making this choice of time-integrator potentially expensive. However, this extra expense is tolerated in order to maintain high-order accuracy in time. It should also be mentioned that we have chosen an explicit method in order to focus the discussion of CG and DG with respect to scalability. The optimal scalability will be achieved by an explicit method. A study on the scalability of IMEX methods requires a discussion of a number of topics including: implicit time-integrators, the choice of iterative solver, and specific preconditioning strategies; all of which we reserve for future work.

### 3.3. Boundary conditions

Limited-area atmospheric models, such as mesoscale codes, typically require two types of boundary conditions: no-flux boundary conditions (NFBCs), that represent impenetrable objects (e.g., the ground) and non-reflecting boundary conditions (NRBCs), that represent an infinite domain (e.g. the top of the atmosphere) by allowing waves to propagate out of the computational domain without generating spurious reflections. In this section, we outline the implementation of NFBCs and NRBCs for both CG and DG. For additional details, see [15,16].

#### 3.3.1. No-flux boundary conditions

All of our test cases use NFBCs on the bottom boundary, while some test cases (such as the rising thermal bubble) use NFBCs on other boundaries as well. For CG, we enforce

$$\hat{\mathbf{n}} \cdot \mathbf{u} = 0 \tag{30}$$

for all points on the boundary $\Gamma$, where $\hat{\mathbf{n}}$ is the outward pointing unit normal vector on $\Gamma$ defined as $\hat{\mathbf{n}} = (n_x, n_y, n_z)^T$. In order to apply the NFBC to the prognostic vector $\mathbf{q}$, we augment $\hat{\mathbf{n}}$ to $R^5$ via $\hat{\mathbf{n}} = (0, \hat{\mathbf{n}}^T, 0)^T$, yielding $\hat{\mathbf{n}} \cdot \mathbf{q} = 0$. To apply these boundary conditions in the *strong* sense, we construct a 3 by 3 orthogonal projector $P$ defined as follows:

$$P = \begin{pmatrix} 1 - n_x^2 & -n_x n_y & -n_x n_z \\ -n_y n_x & 1 - n_y^2 & -n_y n_z \\ -n_z n_x & -n_z n_y & 1 - n_z^2 \end{pmatrix} \tag{31}$$

This matrix is constructed during the initialization phase and applied to the RHS operator after each time-step.

On the other hand, for DG the NFBCs are imposed via the numerical flux as follows. On boundary edges where NFBCs are to be imposed, a ghost-cell is constructed on the other side of the wall that has a velocity vector the negative of the interior element. In other words,

$$\mathbf{U}^{(k)} = -\mathbf{U}^{(e)}$$

where $k$ denotes the ghost-cell positioned on the other side of a no-flux boundary; note that this ensures that the no-flux condition $\hat{\mathbf{n}} \cdot \mathbf{U}$ is satisfied. The scalar variables are copied directly such that $\rho^{(k)} = \rho^{(e)}$, etc.

#### 3.3.2. Non-reflecting boundary conditions

In an operational mesoscale NWP model, the four lateral and the top boundaries should properly represent an open domain. That is, waves should smoothly exit the domain without reflection; in addition, information from outside the domain should be allowed to enter the domain of interest. Mathematically modeling this behavior is non-trivial and has attracted the attention of researchers in many disciplines [5,19,28]. In our model, we use a simple, albeit, effective absorbing sponge layer method. The computational domain is surrounded by a layer with Newtonian relaxation coefficients $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$ such that $\alpha = 1$ and $\beta = 0$ in the domain of interest, while $\alpha \to 0$ and $\beta \to 1$ as the boundary is approached. Specifically, for the top boundary, we choose

$$\beta = \left( \frac{z - z_s}{z_t - z_s} \right)^4 \tag{32}$$

where $z_t$ is the vertical height of the domain and $z_s$ is the bottom of the sponge layer. Similar functions are used for the lateral boundaries of the domain. Once the sponge layer is constructed, the numerical solution $\tilde{\mathbf{q}}$ given by the RHS operator is relaxed to some known solution at the boundary $\mathbf{q}_b$ via

$$\mathbf{q} = \alpha(\mathbf{x})\tilde{\mathbf{q}} + \beta(\mathbf{x})\mathbf{q}_b \tag{33}$$

For problems under consideration in this paper, $\mathbf{q}_b$ is a far-field condition (e.g. known wind velocity and zero perturbation of scalars).

### 3.4. Artificial diffusion

To add a certain level of numerical dissipation we have implemented second order artificial diffusion operators, although hyper-viscosity is also possible. For Eq. (6) we add the following right-hand-side operator

$$\begin{pmatrix} 0 \\ \nu\nabla\cdot(\nabla\mathbf{u}) \\ \nu\nabla\cdot(\nabla\theta') \end{pmatrix}$$

and for Eq. (12) we add the operator

$$\begin{pmatrix} 0 \\ \nu\nabla\cdot(\rho\nabla\mathbf{u}) \\ \nu\nabla\cdot(\rho\nabla\theta') \end{pmatrix}$$

Note that the artificial diffusion operator applied to the potential temperature equation must act only on the potential temperature perturbation, $\theta'$, since applying it to the full potential temperature variable would smoothen the background reference field and destroy the hydrostatic balance. For CG the implementation of the diffusion operator follows the usual strategy through integration by parts. For DG, we use the local discontinuous Galerkin method as described in [15].

## 4. Parallel implementations of EBG methods

NUMA is an MPI-based code targeted toward distributed memory architectures (e.g. clusters). In this section, we discuss the parallel implementation of NUMA, including: a description of the domain-decomposition (Section 4.1), communication strategies for both CG and DG (Sections 4.2 and 4.3), a comparison of these communication volumes (Section 4.4 discussion of the memory access of these EBG methods (Section 4.5).

### 4.1. Domain decomposition

We decompose $\Omega$ into $N_p$ processor elements (PE) $\Omega_p$ that consist of local elements $\Omega_{e'}^{(p)}$. Mathematically, we rewrite Eq. (13) as

$$\Omega = \bigcup_{p=1}^{N_p} \bigcup_{e'=1}^{N_e^{(p)}} \Omega_{e'}^{(p)} \tag{34}$$

where $N_e^{(p)}$ is the number of local elements residing on PE $p$. Since the DSS operator for CG requires global elements $e$ and the flux integrals for DG require global faces $f$, we must also construct local to global mappings for both elements $e = LGE^{(p)}(e')$ and faces $f = LGF^{(p)}(f')$ that map local elements $e'$ and faces $f'$ on processor $p$ to global elements $e$ and faces $f$ residing on the global domain $\Omega$. For CG, the inverse global–local mapping is also required, which may be constructed via a hash table or implicitly using a binary search.

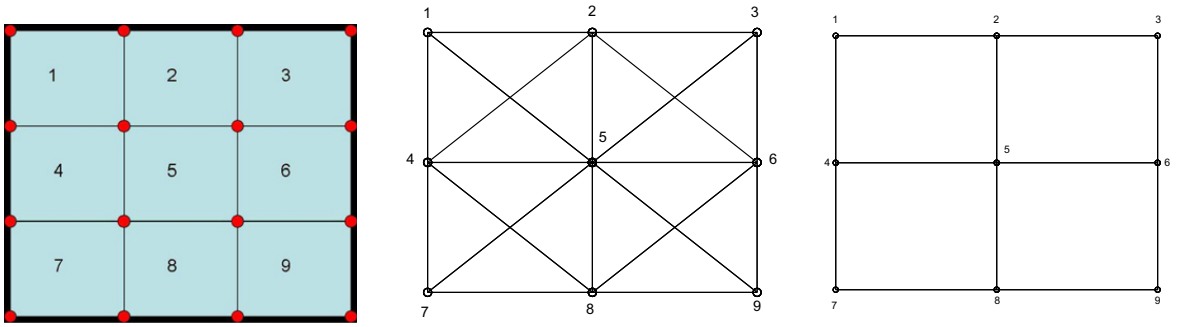
A guiding principle in the construction of NUMA is to maintain independence between grid generation and the CG/DG solver; hence, domain decomposition should be as general as possible and not be constrained by the choice of grid. Therefore, we have implemented a domain decomposition strategy based on the widely used METIS graph partitioning library [22]. METIS requires an adjacency graph where the $N_e$ vertices are elements $\Omega_e$ and the "edges" denote the connectivity between elements. Since the DSS operator requires information from elements that share nodes, the "edges" include all forms of geometric connectivity (faces, edges, and vertices). For maximum flexibility, we construct a weighted adjacency graph $G' = (V, E)$ with adjacency matrix $A'$ of size $N_e$ by $N_e$ defined as

$$a'_{ij} = \begin{cases} v & \text{if } i \text{ and } j \text{ are vertex neighbors} \\ e & \text{if } i \text{ and } j \text{ are edge neighbors} \\ 1 & \text{if } i \text{ and } j \text{ are faces neighbors} \end{cases} \tag{35}$$

where $v$ and $e$ are arbitrary weights. Although not explored in this paper, the values of $v$ and $e$ may be chosen to construct a machine-optimal weighted adjacency matrix. Once $A'$ is constructed the adjacency matrix $A$ for the graph $G = (V, F)$, where $F$ are geometrical faces, is simply $a_{ij} = 1$ if $a'_{ij} = 1$ and $a_{ij} = 0$ otherwise. Two example connectivity graphs for a 2D grid are shown in Fig. 1, showing both edge and vertex connectivity. The associated 9 by 9 adjacency matrix has nodes with degree ranging from 3 (for the corner nodes) to 8 (for the central node). In contrast, the flux integrals only require face adjacency information; hence, our DG code only uses the standard adjacency matrix $A$. Also shown in Fig. 1 is the adjacency matrix associated with DG, which only shows edge (face) adjacency. These adjacency matrices, along with the number of processors $N_p$ are then passed to METIS, which returns a partition mapping global elements to processors. This mapping is then used to construct the local to global mappings $LG$ necessary for global assembly in Eq. (25), as well as processor-element communication data structures.

### 4.2. Parallelization: CG

We first consider the parallelization of CG. The global DSS operator in Eq. (25) requires inter-processor communication due to the $C^0$ requirement at processor element boundaries. A global DSS is required in two parts of the code: construction




(a) EBG Grid (b) Generalized Adjacency Graph (CG) (c) Adjacency Graph (DG)

**Fig. 1.** Example 2D grid (left), the CG associated adjacency graph (center), and DG adjacency graph (right). Since CG methods use nodal communication, the generalized adjacency graph includes both edge and vertex connectivity, with a maximum degree of eight. For 3D, structured, hexahedral grids, the maximum degree is 26. In contrast, typical DG methods use face-based communication and hence only require the adjacency graph shown in the right panel. For 3D hexahedral grids the maximum degree for DG is only 6.

of the mass matrix and construction of the right-hand side (RHS). To perform this communication, we first construct the boundary nodes of each processor (excluding the physical boundary where BCs are applied). We denote the boundary of processor element $i$ by $\partial\Omega_i$. In order to communicate between processor elements, we construct two data structures *send* and *recv*. Consider a particular processor $i$ and neighboring processor $j$. The *send* data structure contains the local nodes on processor $i$ that are sent to each neighboring processor $j$, while *recv* contains the local nodes on processor $j$ that must be sent to $i$. In order to construct *send*, we use the method shown in Algorithm 1. The corresponding *recv* structure contains the same grid points as *send*, although they may be ordered differently.

**Algorithm 1:** Construction of MPI_ISEND/MPI_IRECV communication data structures for both CG and DG

**for all** NBHs $j$ of $i$ **do**

MPI_ISEND $\partial\Omega_i^G \leftarrow LGE^{(i)}(\partial\Omega_i)$ to proc $j$

MPI_IRECV $\partial\Omega_j^G \leftarrow LGE^{(j)}(\partial\Omega_j)$ to proc $i$

$B \leftarrow \partial\Omega_i^G \cap \partial\Omega_j^G$

$send(j) \leftarrow \left[LGE^{(i)}\right]^{-1}(B)$

**end for**

Once these data structures have been constructed, the global mass matrix and RHS operators may be constructed via a two step process. The global mass matrix $M_{IJ}$ and RHS operator in Eq. (25) are decomposed as

$$M_{IJ} = \bigwedge_{e=1}^{N_e} M_{ij}^{(e)} = \bigwedge_{n_p=1}^{N_p} \bigwedge_{e'=1}^{N_e^{(p)}} M_{ij}^{(e')} \quad \text{and} \tag{36a}$$

$$R_I = \bigwedge_{e=1}^{N_e} R_i^{(e)} = \bigwedge_{n_p=1}^{N_p} \bigwedge_{e'=1}^{N_e^{(p)}} R_i^{(e')} \tag{36b}$$

Hence, the global DSS is decomposed into a local, on-processor DSS and a global DSS, that requires inter-processor communication. In this way, global continuity between elements is preserved during the construction of each RHS. Note that the communication stencil is simply the boundary of each processor element $\partial\Omega^{(i)}$ and is independent of the polynomial order $N$; that is, unlike high-order finite difference or finite volume methods, the spectral-element method is *halo-free*. We note however, that the message size grows with the surface area of the processor element as $N^2$. The global DSS procedure may be summarized as follows:

**Algorithm 2:** The direct stiffness summation (DSS) operation in CG

1. Perform a local DSS on processor $i$.
2. Exchange boundary points between processors $i$ and all processor neighbors $j$ using (asynchronous) *isend* and *irecv* MPI commands.
3. Perform a global DSS using the boundary data received from neighbors $j$.

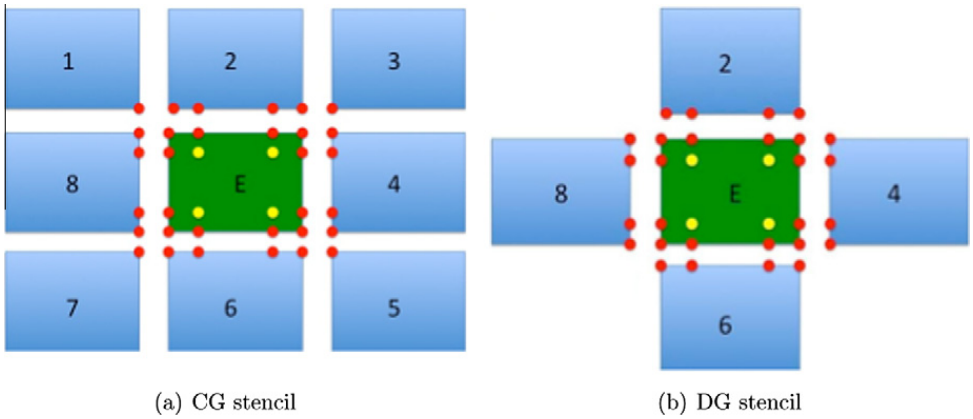



**Fig. 2.** Computational stencils for (a) CG and (b) DG in a 2D setting, where each processor element owns one element. In the left panel (CG), the element $E$ (green) requires nodal information from its 8 nodal neighbors in order to construct the DSS operator on the boundary (red dots). In the right panel (DG), the same element $E$ only requires nodal information from its 4 face neighbors in order to construct flux integrals. In both methods, the interior nodes (yellow dots) do not need to be communicated to adjacent processors, resulting in a *halo-free* algorithm. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The global DSS operation is represented schematically in the left panel of Fig. 2 in a 2D setting. To simplify the discussion, each processor is assumed to own one element. In order to construct the RHS operator on the boundary (red dots), the element $E$ (green) requires nodal information for its 8 nodal neighbors. These neighbors include both edge neighbors (2, 4, 6, and 8) and vertex neighbors (1, 3, 5, and 7). In a 3D setting, a hexahedral element may have (a maximum) of 6 face neighbors, 12 edge neighbors, and 8 vertex neighbors, for a total of 26 nodal neighbors. Note that for tetrahedral grids, the number of vertex neighbors may be much greater. To reduce this communication cost, METIS is used to reduce the total number of vertex neighbors; in a typical 3D partition, a processor element lying away from a physical boundary has 16 or fewer nodal neighbors; hence, the METIS-based decomposition further reduces communication cost relative to a naive geometric domain decomposition. In particular, we use the routine `METIS_PartGraphRecursive`, which reduces the edge-cut of the graph $G$.

EBG methods possess purely local communication stencils. Referring to the left panel of Fig. 2, the interior nodes (yellow dots) do not need to be communicated to adjacent processors, resulting in a *halo*-free algorithm. Thus unlike high-order finite difference and finite volume schemes, spectral elements may achieve spectral accuracy without sacrificing their local character.

### 4.3. Parallelization: DG

Parallelization of DG is much more straightforward than CG. Since there are no global mass matrices, all that is required is a way to communicate boundary data between processor elements. In addition to a local/global element-mapping, an additional local/global face structure is also required. Faces on each processor element (excluding boundary faces) are classified into two types: intra-processor and inter-processor. The necessary send and receive data structures are then constructed by calculating the intersection of face boundaries on each PE as described in a manner similar to Algorithm 1. Unlike CG, however, the DG algorithm requires only the set of faces $f$ which need to be sent to adjacent processors and not the actual grid points. Once these maps and data structures are constructed, the flux operators are evaluated using the following algorithm, which employs interleaved computation/communication:

**Algorithm 3:** Interleaved computation of the flux operator in DG

1. Send/Receive boundary data via non-blocking MPI Send/Receive. While waiting for boundary data do:
2. Compute volume integrals.
3. Compute intra-processor flux integrals.
4. `MPI_Wait()`
5. Compute inter-processor flux integrals.
6. Multiply by element-wise inverse mass matrix.

The communication stencil is illustrated geometrically in the right panel of Fig. 2. Like CG, DG algorithms do not require internal nodes to be communicated to adjacent processors and is hence *halo-less*. Moreover, since all communication is based on fluxes, only elements which share adjacent faces need to communicate. This compact stencil has a crucial impact on both the communication volume and memory access patterns of DG when compared to CG. These issues are explored in the next two sections.

We note that the focus of this study is a comparison of CG and DG from an algorithmic point of view and that the parallelization algorithms outlined in this and the previous section are not necessarily optimal. In particular, we have not implemented sophisticated scheduling between the individual MPI processes using `MPI_Waitany` or `Mpi_Waitsome`, nor have we exploited topology aware communication. We are currently addressing these implementation details in our codes. Therefore, it must be understood that both methods can indeed be further optimized; however, the results we present represent the simplest approaches for constructing parallelization strategies.

### 4.4. Communication volume

A simple model [9] for the communication time (or cost) of a single MPI process is

$$t_c = t_a(\alpha + \beta m) \tag{37}$$

where $t_a$ is the time to do a floating-point operation, $\alpha$ is proportional to the latency (i.e., message startup cost), and $\beta$ is the asymptotic transfer rate (i.e., inverse bandwidth). Typically, the latency cost $\alpha$ is larger than bandwidth cost $\beta$ by one to three orders of magnitude, depending on the architecture of the machine. To characterize the trade-off between bandwidth and latency, the ratio $\beta/\alpha$ provides a useful metric; a typical value is $\beta/\alpha = 0.01$. Using the model given by Eq. (37) the communication overhead of both DG and CG may be quantitatively analyzed.

Consider a hexahedral processor element (PE) with $N_e$ elements discretized with order $N$ polynomials. Geometrically, the lengths of the face, edge, and vertex messages are approximated as

$$m_f = n_{var} N_e^{2/3} (N+1)^2 \tag{38a}$$

$$m_e = n_{var} N_e^{1/3} (N+1) \tag{38b}$$

$$m_v = n_{var} \tag{38c}$$

where $n_{var} = 5$ is the number of state variables, $N_e^{2/3}$ approximates the number of faces while $N_e^{1/3}$ the number of edges.

For DG, since the inter-processor fluxes only require face communication, and since each PE has an average of six face neighbors, the total communication cost at each stage of the RK time-integrator is

$$C_{\text{DG}} = t_a \alpha \left( 6 + 6 \frac{\beta}{\alpha} n_{var} N_e^{2/3} (N+1)^2 \right). \tag{39}$$

Asymptotically, we see that the communication cost scales as $\mathcal{O}\left(N_e^{2/3} N^2\right)$. In contrast, the computation volume, which is dominated by the construction of volume integrals, scales as $\mathcal{O}\left(N_e N^4\right)$; hence, for large $N$ and/or large $N_e$, the ratio of communication to computation $\mathcal{O}\left(N_e^{-1/3} N^{-2}\right)$ tends to zero, thereby ensuring weak scaling. In other words, the total wallclock time remains constant if the number of PEs grows at the same rate as the number of elements.

In contrast, for CG the construction of the DSS operator requires edge and vertex communications in addition to face communication. Since for a hexahedral PE, there are 12 edges and 8 vertices, the communication cost is

$$\begin{aligned} C_{\text{CG}} &= 6 t_a (\alpha + \beta m_f) + 12 t_a (\alpha + \beta m_e) + 8 t_a (\alpha + \beta m_v) \\ &= t_a \alpha \left( 26 + 6 \frac{\beta}{\alpha} n_{var} N_e^{2/3} (N+1)^2 + 8 \frac{\beta}{\alpha} n_{var} N_e^{1/3} (N+1) + 8 \frac{\beta}{\alpha} n_{var} \right) \end{aligned} \tag{40}$$

As with DG, the communication cost scales as $\mathcal{O}\left(N_e^{2/3} N^2\right)$ with a similar estimate for computation; hence, in an asymptotic sense, both CG and DG have the same communication footprints. Geometrically, this can be understood since the surface area of the faces (required by both CG and DG) of a PE grow faster than the edges and vertices (that are required only by CG). These asymptotic estimates are not significant for practical choices of $N$ and $N_e$, e.g., for $N \in [4, 16]$ and small values of $N_e$ the intermediate behavior of Eqs. (39) and (40) become quite important. We note briefly that the communication footprint of CG methods may be reduced to a footprint similar to DG using the $d$-directional exchange algorithm proposed in [11]; however, implementation of this algorithm is non-trivial and limited to tensor product meshes, and is therefore not explored in this study.

A typical ratio of bandwidth to latency on modern computers is $\beta/\alpha = 0.01$. Consider the case of one element per PE, $N_e = 1$, and $N = 4$. Then $C_{\text{CG}}/C_{\text{DG}} \approx 2.6$. Hence, CG has a significant communication overhead relative to DG in practice. To help explain this communication overhead, we plot the ratio of Eqs. (40) and (39) in Fig. 3 for three different values of $\beta/\alpha$ and a range of polynomial orders $N$. Several trends are evident from this graph. Firstly, for all bandwidth/latency ratios, the communication overhead is greatest for linear ($N = 1$) polynomials and decreases monotonically to one as $N \to \infty$. This behavior is expected from the asymptotic estimates made above. However, from a practical point of view, the machine architecture and MPI implementation, which is characterized by the ratio $\beta/\alpha$, determine the relative communication footprint of CG to DG. As the latency costs increase relative to bandwidth costs ($\beta/\alpha$ decreases), CG incurs greater communication rela-

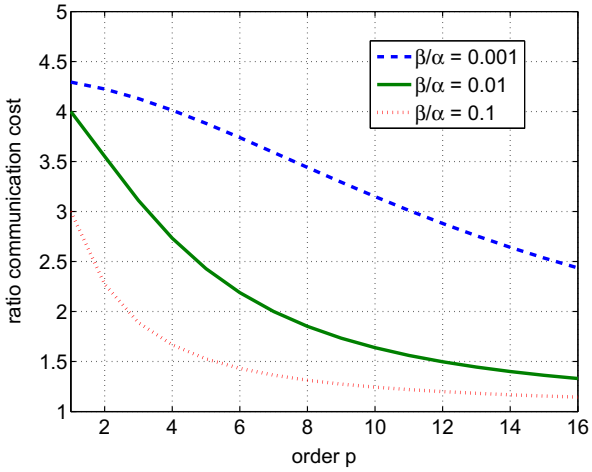



**Fig. 3.** The Ratio of communication costs $C_{CG}/C_{DG}$ plotted for transfer rate/latency ratios $\beta/\alpha = 0.1, 0.01$ and 0.001 for polynomial orders $N = 1$ to 16.

tive to DG. Since the general trend in HPC is toward greater bandwidth ($1/\beta$ is large and therefore $\beta$ is small) and greater latency (large $\alpha$), we expected DG to significantly outperform CG in terms of communication footprint as $\beta/\alpha$ decreases.

Note that for both CG and DG, the communication volume is $\mathcal{O}\left(N_e^{2/3}N^2\right)$ while the computation volume is $\mathcal{O}\left(N_eN^4\right)$. Hence, for large number of elements and (relatively) large orders $N$, the computation volume overwhelms the communication volume. Therefore, in the DG algorithm described in Algorithm 3, steps 2 and 3 typically require much more time than the communication in step 1. For this reason, DG is expected to scale to massive numbers of cores on distributed memory architectures.

### 4.5. Memory access

Fetching data from memory poses a serious bottleneck in any HPC application. The latency associated with reading data from main memory can adversely affect the efficiency and scaling of any HPC application. In general, EBG methods exhibit a high level of data locality since the computational work is organized into discrete elements. In this section, we explore the impact of data layout and access in CG and DG and how this data access affects performance.

Fig. 4 shows the grid numbering used by (a) CG and (b) DG for $N_e = 4$ and $N = 3$. Let us assume for the moment that CG uses a global indexing while DG must use a local indexing. This data layout has an impact on the amount of data that must be fetched from memory and stored in fast cache for an EBG method. In DG, the construction of volume integrals is element-local. Since the data for each element lies in a contiguous region of memory, all of the data for each element may be fetched from slow memory loaded into fast cache at the beginning of an element-wise operation. The only operation that requires access to a neighboring element's data is the construction of flux integrals. Hence, during the construction of a RHS, the number of cache misses is small.

For CG, the state vector is typically stored in a continuous block of memory, rather than in an element-wise fashion. Note that it is possible to use element-wise storage in CG using an additional mapping operator; however, in this analysis, we discuss only the standard finite element (global) numbering. Obviously, this data layout uses less memory than the element-wise DG layout; however, a penalty in performance may be incurred. To construct a RHS, first data must be fetched from the global state vector and stored in a local array. Cache misses may occur at all points on the boundary of the element, especially for large grids, where there is a large stride in indexing. Element-wise operations are then performed, followed by DSS, which requires updating the global state vector, thus resulting in additional cache misses. These cache misses are exacerbated in 3D, where a typical element requires data from 26 adjacent elements. Although these cache misses may be partially alleviated by grid renumbering, a CG method can never achieve the *natural* data locality of a DG method. Even if CG uses DG storage, the cache misses associated with the DSS operator cannot be circumvented; the DSS operator requires indirect memory access that requires striding through the memory in order to enforce $C^0$ continuity.

For optimal performance, the entire on-processor problem should fit in cache. In this situation, which only occurs for small test problems and/or large processor counts, no cache misses will happen. A rough estimate for the size of the on-processor problem is made for both DG and CG. To construct a RHS requires the following data structures: (1) a state vector, (2) reference vector, (3) RHS vector, (4) the inverse mass "matrix", and (5) nine metric terms and two Jacobian matrices (for both volume and flux integrals). The data structures in items (1)–(3) each have size $n_{var}N_e(N+1)^3$ with $n_{var} = 5$, whereas structures (4) and (5) have size $N_e(N+1)^3$, thus yielding $26N_e(N+1)^3$ real numbers. A typical 64-bit machine uses 8 bytes to store a real, yielding a problem size of $208N_e(N+1)^3$. The memory requirements for CG is slightly smaller, since data is stored using a global, rather than element-based index; however the magnitude is similar. On typical clusters (e.g., TACC's Ranger), each core (and hence each PE) has a cache with size 512 KB, implying that a single element can fit into cache provided $N \leqslant 12$. Using this estimate, approximately 20 continuous elements fit into cache for $N = 4$, while about 3 elements fit into cache for $N = 8$.

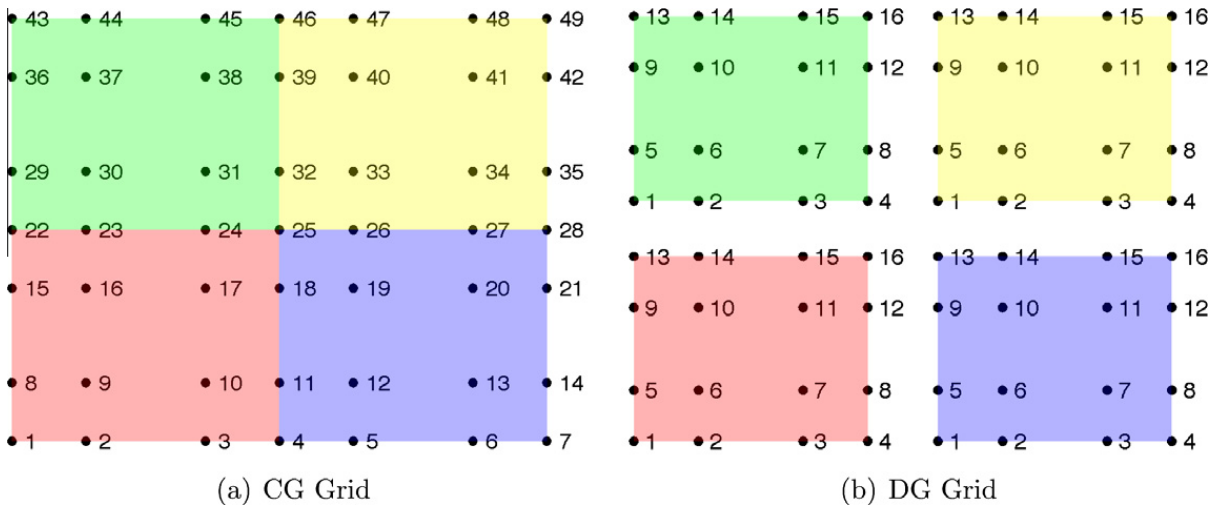



**Fig. 4.** Grid numbering for (a) CG with global indexing and (b) DG with local indexing for $N_e = 4$ and $N = 3$.

From these estimates, the number of cache misses for both CG and DG may be estimated. For DG, the number of cache misses may vary from 0 to 6. For a very large problem or a small number of PEs, a cache miss will occur during each flux construction, when data from an adjacent processor is accessed, yielding 6 misses. For intermediate sizes, a cache miss does not occur when fetching data from an element with continuous numbering, yielding 2 or 4 misses; finally, for very small problems, all the data is in cache, yielding no misses. For CG, the number of misses ranges from 0 to 52. For large problems, reading and writing any data from an adjacent element will incur a cache miss, yielding 2 * 26 = 52 misses. For smaller problems, this count may be reduced to 48 or fewer misses, depending on the configuration of the sub-domain. Excluding sub-domains small enough to fit into cache, the number of cache misses associated with CG will be many times larger than the misses associated with DG regardless of the numbering scheme used in CG (i.e., even if element-wise DG numbering is used with CG).

## 5. Results

### 5.1. Test cases

Although a standard set of 2D mesoscale test cases has been proposed [34] and later used within an element-based Galerkin framework [15], an analogous suite of 3D test cases has not yet been developed. Fortunately, a 3D dynamical core can be run in 2D mode by imposing symmetry in the y-dimension and periodic boundary conditions in the $y$-lateral boundaries. For initial verification of NUMA, we used the test cases proposed in [34], followed by full 3D test cases with no $y$-symmetry. In the following analysis, we consider two test cases: a 3D linear hydrostatic mountain and a 3D rising thermal bubble.

#### 5.1.1. 3D linear hydrostatic mountain

To test the 3D capabilities of NUMA and the NRBCs, we consider stratified flow past an isolated mountain as outlined in [35]. An initial horizontal flow $U = 20$ m/s goes past a mountain with orography given by

$$h(x,y) = \frac{h_0}{(x^2/a^2 + y^2/a^2 + 1)^{3/2}} \tag{41}$$

with mountain half-width $a = 10$ km and height $h_0 = 1$ m. The hydrostatic background is specified by a constant Brunt–Väisälä frequency $N_{bv} = g/\sqrt{c_p T_0}$ with ground temperature $T_0 = 250$ K. In other words, we consider an isothermal atmosphere. No flux boundary conditions are imposed on the bottom of the domain, while NRBCs are imposed on the four lateral boundaries and the top boundary. In order to verify our numerical results, several analytical results from [35,36] are used. First, a contour integral solution for the density perturbations $\rho'$ valid under a linear Boussinesq approximation is used. Also, the velocity perturbations parallel and perpendicular to the flow ($u'$ and $v'$) are known for observation points near the ground ($z = 0$) (see Eqs. (39) and (41) in [35]):

$$u'(x,y,0) = hN_{bv}\frac{x/a}{1 + x^2/a^2 + y^2/a^2} \tag{42a}$$

$$v'(x,y,0) = hN_{bv}\frac{y/a}{1 + x^2/a^2 + y^2/a^2} \tag{42b}$$

under the same approximation.

*5.1.2. 3D rising thermal bubble*

We consider a 3D buoyant thermal bubble rising in a neutrally stratified atmosphere [39], which is the 3D extension of the 2D thermal bubble originally considered in [37]. The hydrostatic potential temperature $\theta_0(z) = 300$ K (neutral atmosphere) is perturbed with a sphere of radius $r_c = 250$ m centered at $(x_c, y_c, z_c) = (500, 500, 260)$ m defined by a cosine taper given by

$$\theta' = A\left[1 + \cos\left(\frac{\pi r}{r_c}\right)\right] \tag{43}$$

with $r = \|\mathbf{x} - \mathbf{x}_c\|_2$ where $\|\cdot\|_2$ denotes the 2-norm, and $A = \frac{1}{2}$ is a constant. Unlike the test case used in [39], our problem has a $C^1$ initial condition, thus mitigating unphysical oscillations at the interface of the bubble. The domain is defined as $(x, y, z) \in [0, 1000]^3$ m. No-flux boundary conditions are applied on all six boundaries.

*5.2. Numerical verification*

The numerical verification proceeds in three steps. In phase one, we ran the code in pseudo-2D mode using 1 element and periodic boundary conditions in the $y$-direction. The numerical results for the standard mesoscale suite [34] are directly compared to the results of our existing 2D model [15]. In phase two, we considered the linear isolated mountain problem, which possesses an approximate analytical solution in the form of a contour integral. In phase three, we consider a three-dimensional buoyant convection problem. Although this problem does not have an analytic solution, its qualitative behavior is well understood.

*5.2.1. 3D linear hydrostatic mountain*

In order to test the 3D operators, orography, and non-reflecting boundary conditions, flow over a 3D isolated linear hydrostatic mountain is considered. This problem may be solved under the linear Boussinesq approximation via a contour integral technique as described in [36]. In addition, closed form expressions for both the down-stream and cross-stream velocity components may be used to judge the numerical solution but only in a qualitative sense (i.e., since we use the fully compressible equations then we cannot use the linear Boussinesq analytic solution to compute convergence rates). In our simulations, we used 20 elements in the $x$ and $y$ dimensions and 10 in the $z$ direction with eighth-order polynomials yielding effective resolutions of $\overline{\Delta x} = \overline{\Delta y} = 1.5$ km and $\overline{\Delta z} = 300$ m. The explicit time-integrator was run at the maximum allowable time-step for each method.

Fig. 5 compares the density perturbations $\rho$' of the analytic and numerical solutions. In panel (a), contours for $\rho$' (analytic are solid while NUMA-CG are dashed) are shown after 5 h of simulation time and compared to the contour integral solution; after 5 h the models have converged to steady-state. The results of NUMA-DG are similar to the dashed contours in Fig. 5(a). Panel (b) of this figure compares the numerical results produced by NUMA-CG, NUMA-DG, and Smith's analytical model along the line $x = y = 120,000$ m. Agreement is very good near the mountain, while the two solutions begin to deviate as $z$ increases due to the influence of the sponge. Agreement between NUMA and Smith's results may be brought into closer agreement by increasing the resolution of the model; at least this is the case in the 2D analog of the problem (see [15,16]). Recall that the "analytic" solution used here is for a Boussinesq model and therefore one should not expect to get exact agreement. Additional verification is performed by comparing the velocity on the surface of the mountain. The down-stream velocity perturbation $u'$

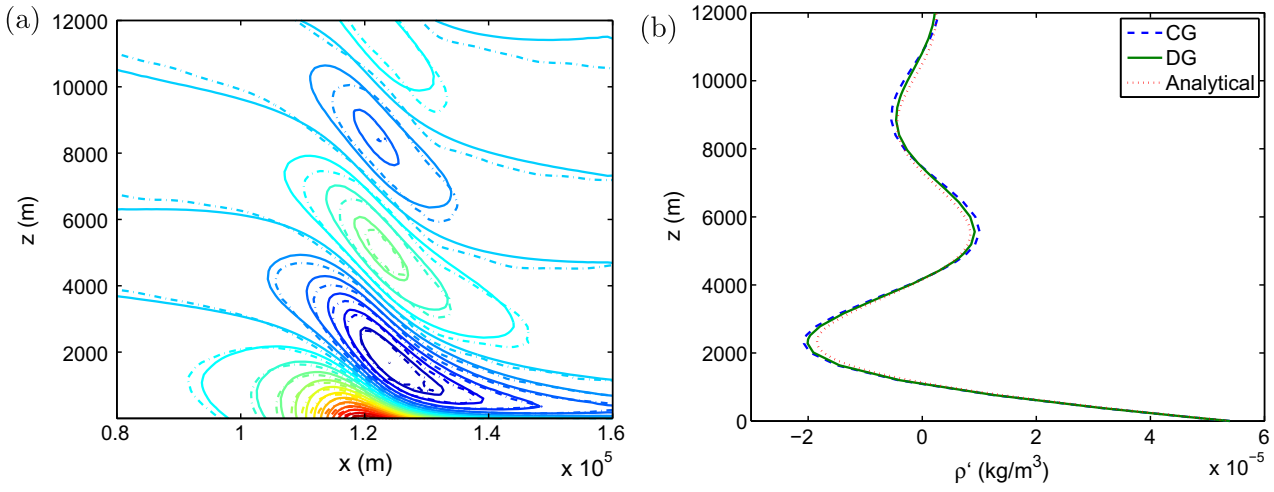



**Fig. 5.** Comparison of the density perturbations $\rho'$ for the isolated linear hydrostatic mountain using NUMA (solid line) and a contour integral solution (dashed line) [36]. In panel (a), the $\rho'$ contours in the plane $y = 120,000$ m are shown (the analytical model are solid while the CG model are dashed – the DG results are similar to those for CG and are not shown). In panel (b), $\rho'$ is shown as a function of $z$ using $x = y = 120,000$ m using numerical results produced by the CG and DG models, and Smith's analytical model.

and cross-stream velocity perturbation $v'$ at the ground for both the CG and DG models are compared with the analytical formulas in Eq. (42). Fig. 6 displays the results of this comparison after $t = 5$ h of integration. Agreement between the CG and DG models and the analytical formulas given by Eq. (42) is satisfactory, especially for the cross-stream velocity perturbation. We attribute the slight deviation of the down-stream velocity to the sponge layer in the model, which drives the total down-stream velocity to a constant velocity of 20 m/s at the boundary of the computational domain. This additional forcing term affects the quality of the solution near the sponge layer. In fact, for this particular case, the NRBCs dominate the solution; a similar behavior is seen in the 2D mountain cases in the NUMA2D results presented in [15].

#### 5.2.2. 3D rising thermal bubble

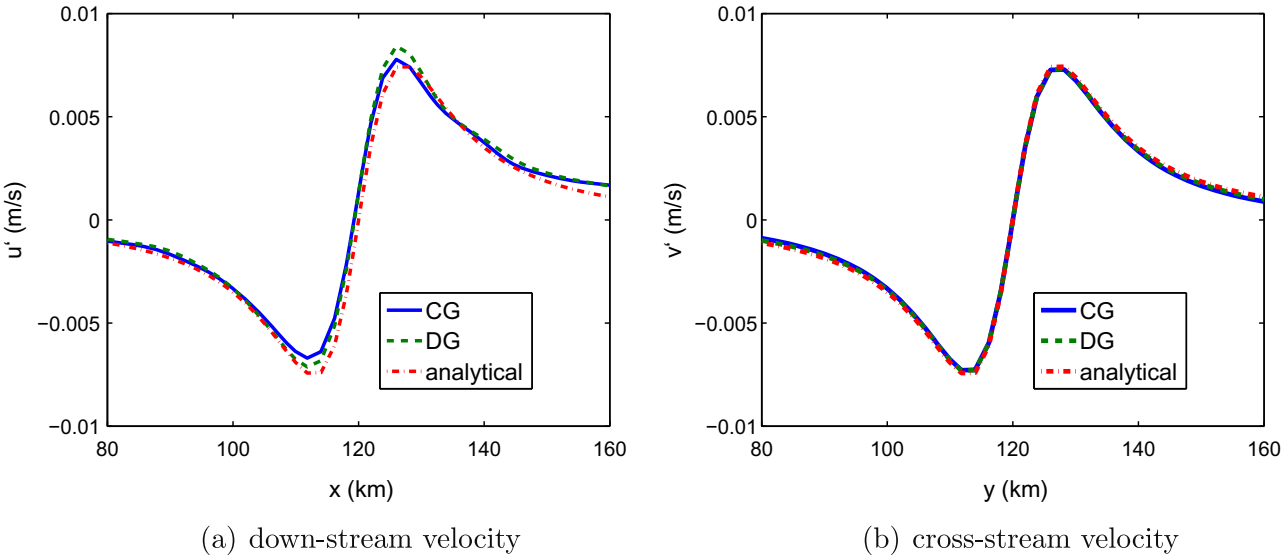
Finally, the results of the buoyant convection experiment are shown in Figs. 7 and 8. Numerical results using both NUMA-CG and NUMA-DG are shown. We used a total of $10^3$ elements with $N = 8$ polynomials, yielding an effective resolution of 12.5 m. Due to the coarse resolution of this run, a small amount of artificial diffusion $\nu = 0.5$ m$^2$/s was used to suppress grid noise. In the CG simulation, a time-step of $\Delta t = 0.01$ s was used, whereas DG required $\Delta t = 0.005$. A smooth potential temperature perturbation in an initially neutral atmosphere generates vertical updrafts and shear velocities, which cause the bubble to rise, deform, and transition to turbulence. Fig. 7 displays $x$-$z$ cross-sections of the potential temperature perturbation for $t = 0$, 200, and 400 s, while Fig. 8 displays a 1D cross-section along the line $x = y = 500$ m for both CG and DG. Note that the CG result displays significant Gibbs oscillations, while the Gibbs oscillations are less pronounced in the DG code; this behavior is expected, since DG is known to capture sharp gradients more effectively. The rising thermal bubble problem tests the models' ability to capture the effects of turbulence and turbulent convection. Although no analytical solution exists for this problem, the numerical results are physically plausible and resemble previous 3D bubble experiments in [39,1]. Similar results are seen for both the NUMA-CG and NUMA-DG codes.

For this test case, the relative conservation of both mass ($M$) and energy ($E$) for both CG and DG were calculated to be $M \approx 10^{-13}$ and $E \approx 10^{-7}$. In other words, both CG and DG conserve mass to a level approaching machine precision, whereas energy, which is not a conservation variable in either model, is not conserved. Hence, there are no particular disadvantages to using a non-conservative form of the equations within the CG model.

### 5.3. Parallel performance

Both NUMA-CG and NUMA-DG have been deployed on TACC's Ranger Sun Constellation cluster [3]. It is important to understand that there may exist more optimal parallel implementations and, therefore, our results are illustrative of two very specific choices for constructing parallel algorithms (as described in Section 4). However, the specific choices we have made in our parallel implementation are either standard or simple in that they represent the most obvious ways of constructing parallelization strategies for both the CG and DG methods.

Two test problems using the rising thermal bubble are executed using a fixed grid comprised of $32^3 = 32,768$ elements with both $N = 4$ and $N = 8$ polynomials. The $N = 4$ simulation has an effective resolution of 7.8 m in both the horizontal and vertical, while for $N = 8$ the effective resolution is 3.9 m. The maximum allowable stable time step was used for both the CG and DG models (note that the CG model uses a time-step twice as large as DG and the $N = 4$ simulations allow a time-step




(a) down-stream velocity (b) cross-stream velocity

**Fig. 6.** Comparison of the (a) down-stream velocity perturbation $u'$ and (b) cross-stream velocity perturbation $v'$ for the isolated mountain at ground level. Agreement between the CG and DG models and the analytical formulas given by Eq. (42) is satisfactory, especially for the cross-stream velocity perturbation.

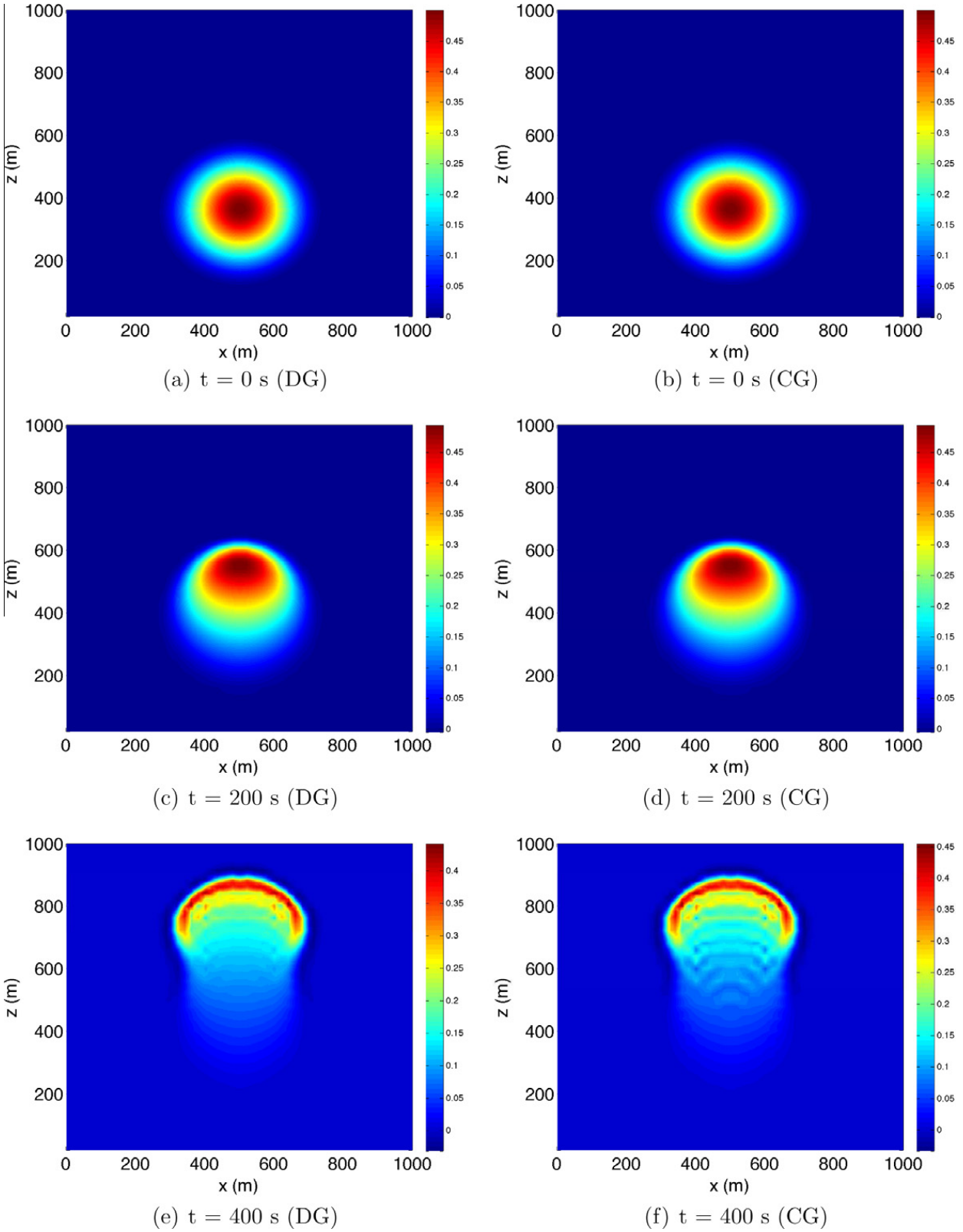



**Fig. 7.** Evolution of a 3D rising thermal bubble problem ($x-z$-slices of the potential temperature perturbation $\theta'$) in the $y = 500$ m plane for $t = 0$, 200, and 400 s using both NUMA-CG and NUMA-DG. A grid consisting of $N_e = 10^3$ elements with $N = 8$ polynomials is used.

twice as large as those for $N = 8$). Due to limitations on the available computational resources, only one run at each processor count was feasible. Also, due to the size of this problem, running the model in serial was impractical; therefore, the wallclock time for each method at 16 cores is assumed to be 16 times the serial wallclock time.

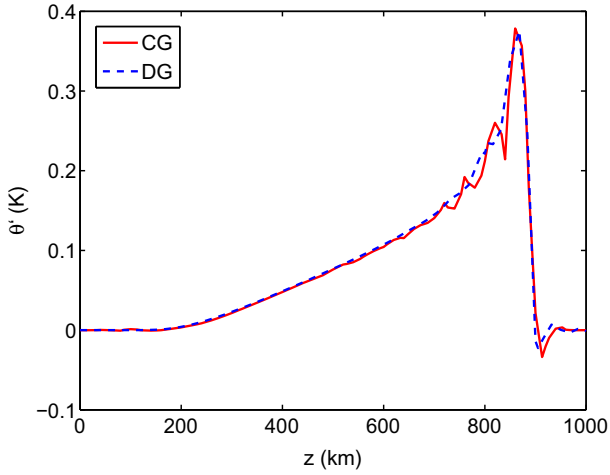



**Fig. 8.** Numerical results for rising thermal bubble problem along the line $x = y = 500$ m using both NUMA-CG and NUMA-DG.

*5.3.1. Results for 4th and 8th order polynomials*

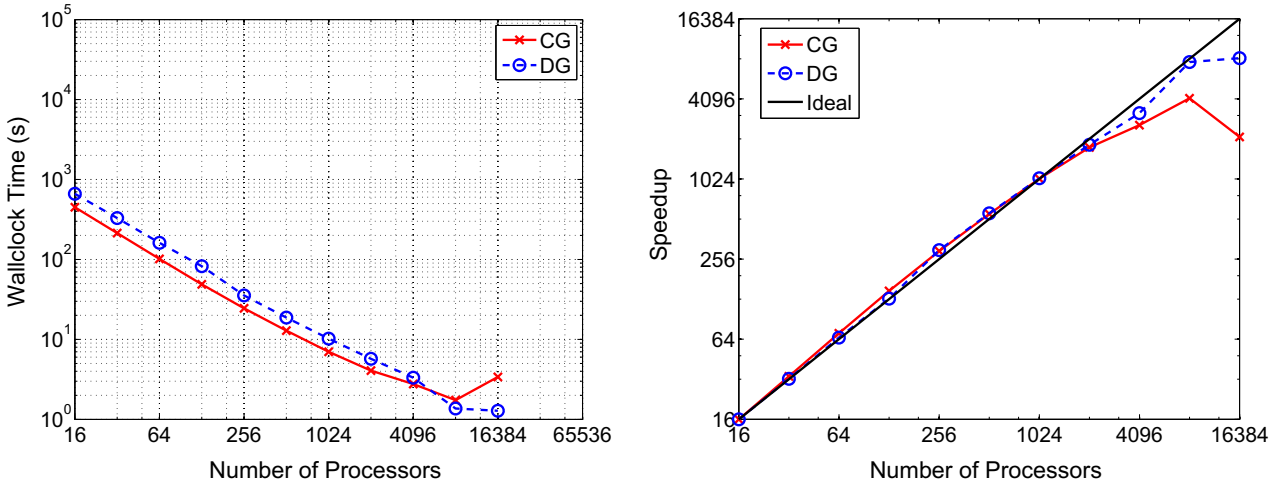
Fig. 9 displays both (a) the wallclock time and (b) speedup for the CG and DG methods using $N = 4$. In panel (b), the ideal speedup (perfect scalability) is also shown. Both methods exhibit nearly perfect linear scaling to 2048 cores. Beyond 2048 cores, however, the scaling of CG begins to decline while DG continues to scale ideally up to 8192 cores. At 16,384 cores, the DG scalability plateaus. Let us now see what happens when we increase the polynomial order.

Fig. 10 displays the same data for $N = 8$. Both CG and DG exhibit nearly perfect linear scaling to 512 cores. Between 512 and 2048, CG loses perfect scaling but recovers beyond 2048 cores. Note that DG exhibits perfect linear scaling all the way through to 32,768 processors which is the maximum number of processors that a $32^3$ element simulation can use. At this point, each PE contains only one element demonstrating the strong scalability of the method.

*5.3.2. Communication cost*

The difference in the scaling behavior between the two methods can be explained by examining the relative communication footprints of both CG and DG. In 3D, each CG processor element (PE) may have up to 26 neighbors, whereas DG has only 6; hence, NUMA-DG's communication footprint is more than 4 times as compact as NUMA-CG. In addition, DG has a more local memory access, since DG stores data on an element-by-element basis. Therefore, only the flux computations require non-local memory fetches; in contrast, CG uses a global index, and requires many more non-local memory accesses although this overhead can be eliminated if DG-type local indexing is used.

*Effect of polynomial order.* Comparing Figs. 9 and 10, we also see the dependence of scalability on the order of the method. The number of grid points for a high-order EBG method is $\mathcal{O}\left(N_e N^3\right)$, whereas the computation volume is $\mathcal{O}\left(N_e N^4\right)$; hence,




(a) wallclock time (b) speedup

**Fig. 9.** Scaling study performed using the NUMA-CG and NUMA-DG codes for the RTB problem using $32^3$ elements and $N = 4$ polynomials. The wallclock time refers to the total simulation time.

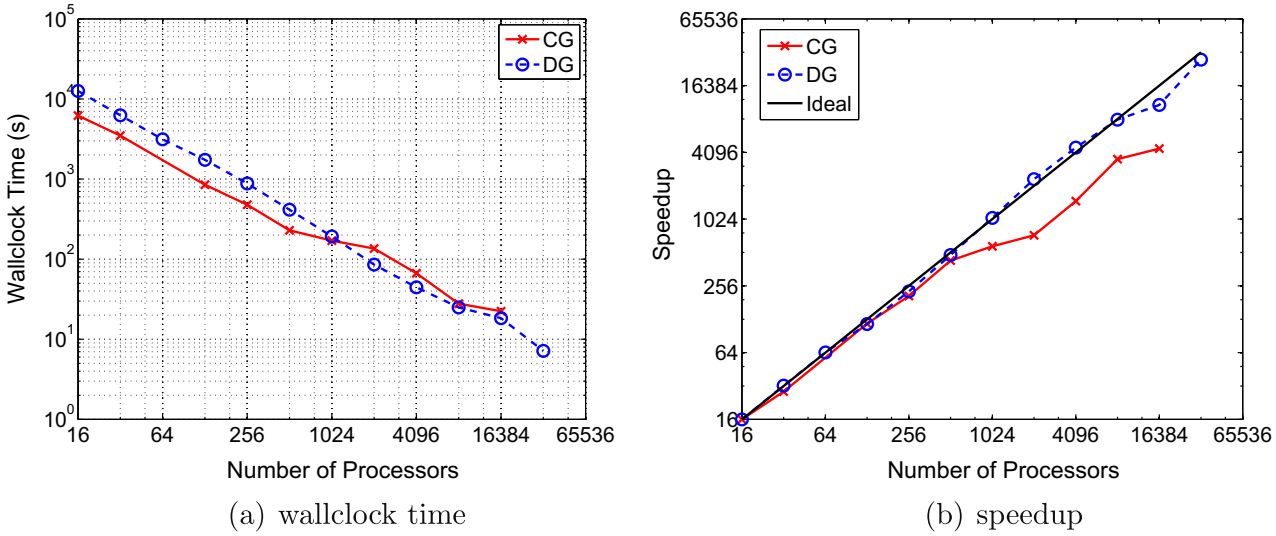



**Fig. 10.** Scaling study performed using the NUMA-CG and NUMA-DG codes for the RTB problem using $32^3$ elements and $N = 8$.

the computation density is $\mathcal{O}(N)$. In contrast, the communication volume, as described in Section 4.4, grows only as $\mathcal{O}\left(N_e^{2/3}N^2\right)$. Hence, as we increase the order $N$, the computation density grows as N whereas the communication density (ratio of communication volume to the number of grid points) decreases as $1/N$. From a computer architecture viewpoint, the *data locality* for both CG and DG increases with order, which is advantageous for both distributed memory implementations and shared-memory implementations using graphical processor units (GPUs) [26]. A high-order method, while more expensive at low processor counts, becomes more efficient as the processor count increases.

For example, at 16 PEs, DG requires 650 s of wallclock time at $N = 4$ and 15,000 s at $N = 8$ (both using $32^3$ elements), yielding a ratio of 23.7 for the wallclock times, i.e.,

$$R_{16} \equiv \left(\frac{WT_{N=8}}{WT_{N=4}}\right)_{NPROC=16} = 23.7$$

where $WT$ is the wallclock time. In contrast, at 8182 PEs, the same simulation requires 1.5 s at $N = 4$ and 25 s at $N = 8$, for a ratio of $R_{8192} = 16.6$; therefore going from 16 to 8192 processors has decreased the computational cost ratio between the 4th and 8th order simulations. We attribute this phenomena to the low communication volume and increased data locality (e.g., greater on-processor work per grid point) achieved by DG as order $N$ increases. Of course, the same arguments hold for CG although the curves are not as straight as they are for DG.

Note that for a fixed number of elements (in this case $32^3$ elements) and increasing the polynomial order from $N = 4$ to $N = 8$ increases the size of the problem by a factor of 32: the number of grid points increases by a factor of 8 and the number of time-steps by a factor of 4. Although the $N = 8$ simulation is 32 times larger than the $N = 4$ simulation, the additional cost of increasing the order $N$ decreases as we move to larger core counts. For example, the minimum wallclock time in Fig. 9 for $N = 4$ is 1.5 s (corresponding to 8192 processors) while the minimum wallclock time in Fig. 10 for $N = 8$ is 7 s (corresponding to 32,768 processors). In this case, we see that the cost ratio between 8th and 4th order is now 4.7 even though the ratio of the problem size is 32.

*Complexity analysis: a geometrical view*. Another way of interpreting the results in Figs. 9 and 10 involves quantifying the operation count that takes place on-processor versus the size of the message that has to be communicated across processors. Geometrically, this can be viewed as the ratio between the computational volume (volume of the PE) and communication footprint (surface area of the PE). To simplify the following discussion let us first define the computational volume on a specific hexahedral processor element (PE) as

$$V_{PE} = \mathcal{O}\left(N_e n_{var}(N+1)^4\right)$$

while the surface area of this HPE that defines the communication cost is

$$S_{PE} = \mathcal{O}\left(N_e^{2/3} n_{var}(N+1)^2\right)$$

where $N_e$ denotes the number of elements per PE, $n_{var}$ are the number of variables in our equations ($n_{var} = 5$ in our case), and $N$ denotes the order of the polynomial; these relations come from Eq. (38). Note that the term $N_e^{2/3}$ represents the number of faces of a PE (i.e., the convex hull of the PE). Using these two relations, we can now define the ratio

$$\mathcal{R}_{PE}^N \equiv \frac{V_{PE}}{S_{PE}} = \mathcal{O}\left(N_e[PE]^{1/3}(N+1)^2\right)$$

where $N_e[PE]$ denotes that (for a fixed grid) the number of elements per processor $N_e$ is a function of the number of processors used (PE); this ratio we now use to compare Figs. 9 and 10.

Fig. 9 shows that for $N = 4$ both CG and DG lose perfect scalability between 8192 and 16,384 processors. Using a grid with $32^3$ total elements we can determine that 8192 processors gives $N_e = 4$ and for 16384 processors yields $N_e = 2$. From these values we determine that $\mathcal{R}^4_{8192} = 39$ and $\mathcal{R}^4_{16384} = 31$. In contrast for $N = 8, \mathcal{R}^8_{8192} = 127$ and $\mathcal{R}^8_{16384} = 101$ and scalability is maintained (Fig. 10). This tells us that for the entire range of possible number of processors, for $N = 8$, the ratio of the on-processor work to inter-processor communication is $\geqslant 100$ which means that the communication is overwhelmed by the computation and we therefore see perfect scalability. The $N = 4$ results tell us that a factor of 40 or less is not sufficient to maintain scalability. Looking again at Fig. 9 we see that for $N = 4$ we maintain perfect linear scalability for both CG and DG at 2048 or fewer processors. For these values we determine that $\mathcal{R}^4_{2048} = 62$ so we can conjecture that the factor of on-processor work to inter-processor communication can be as low as 62 while still maintaining scalability.

#### 5.3.3. Memory access

The previous discussion has focused on the effects of load-balancing and communication on scaling. However, memory access must also be considered in order to achieve maximum parallel efficiency. In particular, we wish to fit the local problem into the fast cache on each compute core in order to minimize the number of fetches from *random access memory* (RAM), since each fetch may consume hundreds of clock cycles. To determine the core count at which the local problem fits into cache, consider that the approximate size of each state-vector for the global problem (in bytes) for DG, which uses an element-local storage scheme, is approximately $8n_{var}N_e^g(N+1)^3$, where the coefficient 8 is the size of a real number and $N_e^g$ is the global number of elements. Since we need to store both the state-vector and a RHS, the total size of the global problem for $N = 8, N_e^g = 32^3$ and $n_{var} = 5$ is approximately 2 GB. The size of the global problem for CG is slightly smaller due to a global storage scheme, although the difference is small for high polynomial orders. Each quad-core processor on Ranger has 6 MB of shared L3 cache and each core has 512 KB of dedicated L2 cache. Hence the total cache per core on Ranger is 2 MB. Assuming good load balancing, the local problem will fit into cache using 1000 cores. Performing the same calculation for $N = 4$, we see the local problem will fit into cache using approximately 160 cores. Therefore, we expect to see a super-linear speedup at these core counts, which is evident in Figs. 9 and 10 (see panels b) where the CG and DG simulations exceed the ideal scaling curve.

## 6. Conclusion and future work

### 6.1. Conclusion

In this paper, we have developed a nonhydrostatic model of the atmosphere (NUMA) based on both CG and DG discretizations in space and explicit discretization in time. We note that purely explicit time-discretization is not practical for operational purposes due to the stringent CFL restriction; therefore, we will present the results of our implicit-explicit time-integrators in future work. This paper has subjected NUMA to a couple of limited-area simulations, including orographic flow and buoyant convection problems. The results of these test problems are in agreement with either previous simulations, analytical results, or physical intuition.

The parallel performance study described in Section 5 suggests several directions for the future of EBG methods (in particular, DG methods). Firstly, good scalability was exhibited for both the CG and DG codes for $\mathcal{O}(10^4)$ cores, with near-linear strong scaling for the DG code beyond $\mathcal{O}(10^4)$. As described in Sections 4.4 and 4.5, we attribute these scalability results to a large computational volume relative to communication volume and high data locality, which yields a low-cache miss percentage. In particular, DG exhibits low communication volume and outstanding data locality, indicating that DG will be able to achieve ideal scaling up to $\mathcal{O}(10^5)$ cores with no modifications. Secondly, these experiments indicate that high order EBGs (e.g. $N = 8$ relative to $N = 4$) become more efficient at higher core counts. Hence, as we move towards the exascale epoch in which core counts of $\mathcal{O}(10^5)$ and $\mathcal{O}(10^6)$ are rapidly become available, these high order EBGs may become competitive with their established low-order counterparts (finite element and finite volume methods). For these reasons, high-order EBG methods are excellent candidates for next-generation NWP models.

### 6.2. Future work

In conjunction with the Naval Research Laboratory-Monterey, we are incorporating microphysical parameterizations into NUMA. Preliminary experiments using the Kessler scheme [23] have been conducted within a 2D, serial implementation [13]. Since physical parameterizations operate on columns of data independently of adjacent data, the problem is embarrassingly parallel *provided* the domain is decomposed in the horizontal only such that all $z$ values reside on-processor. To facilitate scaling on hybrid shared-distributed memory architectures (such as TACC's Ranger), hierarchical domain decomposition is desirable, whereby MPI communication based on the algorithm developed in the present paper is used in the horizontal and either OpenMP parallelization, appropriate for shared memory, or graphical processor units (GPUs) are employed for fine-grained parallelism in the vertical. For such problems where the domain decomposition is only performed in the $xy$ plane such that all $z$ values are on-processor both the CG and DG methods will always have a much larger computational volume versus

communication footprint that will allow both methods to scale perfectly for the maximum number of processors accommodated by a 2D domain decomposition. In future work, we will explore hybrid parallelization strategies, implicit-explicit (IMEX) time-integrators (along with iterative solvers and preconditioners), and adaptive mesh refinement. The goal of all of these topics is to further improve the efficiency of the NUMA model.

## Acknowledgments

The authors acknowledge Shiva Gopalakrishnan for his assistance in analyzing the bottlenecks of the MPI codes as well as running some of the simulations. In addition we thank Shiva Gopalakrishnan, Michal Kopera, and Les Carr for reading drafts of the paper and offering constructive comments. The authors also acknowledge XSEDE for providing resources on TACC's Ranger Sun Constellation cluster. We would also like to thank the people who run Ranger for their assistance. This work was funded by ONR Grant PE-0602435N.

## References

[1] N. Ahmad, J. Lindeman, A Godunov-type finite volume scheme for meso- and micro-scale flows in three dimensions, Pure Appl. Geophys. 165 (9) (2008) 1929–1939.
[2] J.P. Boyd, Chebyshev and Fourier Spectral Methods, Dover Publications Inc., Mineola, New York, 2001.
[3] Texas Advanced Computing Center, Ranger user guide, 2012. <http://www.tacc.utexas.edu/user-services/user-guides/ranger-user-guide>.
[4] T. Davies, M.J.P. Cullen, A.J. Malcolm, M.H. Mawson, A. Staniforth, A.A. White, N. Wood, A new dynamical core for the Met Office's global and regional modelling of the atmosphere, Quart. J. Roy. Meteorol. Soc. 131 (2005) 1759–1782.
[5] J.R. Dea, F.X. Giraldo, B. Neta, High-order non-reflecting boundary conditions for the linearized 2-D Euler equations: no mean flow case, Wave Motion 46 (2009) 210–220.
[6] T.J. DeLuca, Performance of hybrid Eulerian–Lagrangian semi-implicit time-integrators for nonhydrostatic mesoscale atmospheric modeling, Master's thesis, Naval Postgraduate School, 2008.
[7] J.M. Dennis, J. Edwards, K.J. Evans, O.N. Guba, P.H. Lauritzen, A. Mirin, A.St. Cyr, M. Taylor, P.H. Worley, CAM-SE: A scalable spectral element dynamical core for the community atmosphere model, Int. J. High Perform. Comput. Appl. 26 (1) (2012) 74–89.
[8] J.M. Dennis, M. Levy, R.D. Nair, H.M. Tufo, T. Voran. Towards and efficient and scalable discontinuous Galerkin atmospheric model, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing, Symposium, 2005, pp. 1–7.
[9] M.O. Deville, P.F. Fischer, E.H. Mund, High-Order Methods for Incompressible Fluid Flow, Cambridge University Press, 2002.
[10] M. Fey, Multidimensional upwinding. Part I. The method of transport for solving the Euler equations, J. Comput. Phys. 143 (1) (1998) 159–180.
[11] P.F. Fischer, A.T. Patera, Parallel spectral element solution of the Stokes problem, J. Comput. Phys. 92 (1991) 380–421.
[12] Aimé Fournier, Mark A. Taylor, Joseph J. Tribbia, The spectral element atmosphere model (SEAM): high-resolution parallel computation and localized resolution of regional dynamics, Mon. Weather Rev. 132 (3) (2004) 726–748.
[13] S. Gabersek, F.X. Giraldo, J.D. Doyle, Dry and moist idealized experiments with a two-dimensional spectral element model, Mon. Weather Rev. http://dx.doi.org/10.1175/MWR-D-11-00144.1, in press.
[14] F.X. Giraldo, The Lagrange–Galerkin spectral element method on unstructured quadrilateral grids, J. Comput. Phys. 147 (1) (1998) 114–146.
[15] F.X. Giraldo, M. Restelli, A study of spectral element and discontinuous Galerkin methods for the Navier–Stokes equations in nonhydrostatic mesoscale atmospheric modeling: equation sets and test cases, J. Comput. Phys. 227 (8) (2008) 3849–3877.
[16] F.X. Giraldo, M. Restelli, M. L äuter, Semi-implicit formulations of the Navier–Stokes equations: applications to non-hydrostatic atmospheric modeling, SIAM J. Sci. Comput. 32 (2010) 3394–3425.
[17] F.X. Giraldo, Thomas E. Rosmond, A scalable spectral element Eulerian atmospheric model (SEE-AM) for NWP: dynamical core tests, Mon. Weather Rev. 132 (1) (2004) 133–153.
[18] L. Grinberg, G.E. Karniadakis, A new domain decomposition method with overlapping patches for ultrascale simulations: application to biological flows, J. Comput. Phys. 229 (2010) 5541–5563.
[19] R.L. Higdon, Radiation boundary condtions for dispersive waves, SIAM J. Numer. Anal. 31 (1) (1994) 64–100.
[20] R.M. Hodur, The naval research laboratory's coupled ocean/atmosphere mesoscale prediction system (COAMPS), Mon. Weather Rev. 125 (7) (1997) 1414–1430.
[21] G. Houzeaux, M. Vázquez, R. Aubry, J.M. Cela, A massively parallel fractional step solver for incompressible flows, J. Comput. Phys. 228 (2009) 6316–6332.
[22] G. Karypis, V. Kuman, A fast and highly quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[23] E. Kessler, On the Distribution and Continuity of Water Substance in Atmospheric Circulation, AMS, 1969.
[24] R. Klein, U. Achatz, D. Bresch, O.M. Knio, P.K. Smolarkiewicz, Regime of validity of soundproof atmospheric flow models, J. Atmos. Sci. 67 (10) (2010) 3226–3237.
[25] J.B. Klemp, W.C. Skamarock, J. Dudhia, Conservative split-explicit time integration methods for the compressible nonhydrostatic equations, Mon. Weather Rev. 135 (8) (2007) 2897–2913.
[26] A. Klockner, T. Warburton, J. Bridge, J.S, Hesthaven, Nodal discontinuous galerkin methods on graphics processors, J. Comput. Phys. 228 (21) (2009) 7863–7882.
[27] Daniel Y. Le Roux, Dispersion relation analysis of the $p^{NC_1} - p^1$ finite-element pair in shallow-water models, SIAM J. Sci. Comput. 27 (2) (2005) 394–414.
[28] Joseph M. Lindquist, Beny Neta, Francis X. Giraldo, A spectral element solution of the Klein–Gordon equation with high-order treatment of time and non-reflecting boundary, Wave Motion 47 (5) (2010) 289–298.
[29] R.D. Nair, H.W. Choi, H.M. Tufo, Computational aspects of a scalable high-order discontinuous Galerkin atmospheric dynamical core, Comput. Fluids 30 (2009) 309–319.
[30] M.R. Norman, R.D. Nair, F.H.M. Semazzi, A low communication and large time step explicit finite-volume solver for non-hydrostatic atmospheric dynamics, J. Comput. Phys. 230 (2011) 1567–1584.
[31] J.M. Prusa, P.K. Smolarkiewicz, A.A. Wyszogrodzki, EULAG, a computational model for multiscale flows, Comput. Fluids 37 (2008) 1193–1207.
[32] M. Restelli, F.X. Giraldo, A conservative discontinuous galerkin semi-implicit formulation for the Navier–Stokes equations in nonhydrostatic mesoscale modeling, SIAM J. Sci. Comput. 31 (2009) 2231–2257.
[33] U. Schattler, G. Doms, J. Steppeler, Requirements and problems in parallel model development at DWD, Sci. Program. 8 (1) (2000) 13–22.
[34] W.C. Skamarock, J.D. Doyle, P. Clark, N. Wood, A standard test set for nonhydrostatic dynamical cores of NWP models, AMS NWP-WAF Conference Poster, p. P2.17, 2004.
[35] R.B. Smith, Linear theory of stratified hydrostatic flow past an isolated mountain, Tellus 32 (1980) 348–364.
[36] R.B. Smith, Linear theory of stratified flow past an isolated mountain in isoteric coordinates, J. Atmos. Sci. 45 (42) (1988) 3889–3896.
[37] P.K. Smolarkiewicz, J.A. Pudykiewicz, A class of semi-Lagrangian approximations for fluids, J. Atmos. Sci. 49 (22) (1992) 2082–2096.

[38] Raymond J. Spiteri, Steven J. Ruuth, A new class of optimal high-order strong-stability-preserving time discretization methods, SIAM J. Numer. Anal. 40 (2) (2002) 469–491.
[39] S.J. Thomas, J.P. Hacker, P.K. Smolarkiewicz, Spectral preconditioners for nonhydrostatic atmospheric models, Mon. Weather Rev. 131 (2003) 2464–2478.
[40] J. Thuburn, Todd D. Ringler, William C. Skamarock, Joseph B. Klemp, Numerical representation of geostrophic modes on arbitrarily structured c-grids, J. Comput. Phys. 228 (2009) 8321–8335.
[41] Paul A. Ullrich, C. Jablonowski, B. van Leer, High-order finite-volume methods for the shallow-water equations on the sphere, J. Comput. Phys. 229 (17) (2010) 6104–6134.
[42] L.C. Wilcox, G. Stadler, C. Burstedde, O. Ghattas, A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media, J. Comput. Phys. 229 (24) (2010) 9373–9396.