Bernstein-Bézier weight-adjusted discontinuous Galerkin methods for wave propagation in heterogeneous media

Kaihang Guo[a,*], Jesse Chan[a]

[a]*Department of Computational and Applied Mathematics, Rice University, 6100 Main St, Houston, TX 77005, United States*

Abstract

This paper presents an efficient discontinuous Galerkin method to simulate wave propagation in heterogeneous media with sub-cell variations. This method is based on a weight-adjusted discontinuous Galerkin method (WADG), which achieves high order accuracy for arbitrary heterogeneous media [1]. However, the computational cost of WADG grows rapidly with the order of approximation. In this work, we propose a Bernstein-Bézier weight-adjusted discontinuous Galerkin method (BBWADG) to address this cost. By approximating sub-cell heterogeneities by a fixed degree polynomial, the main steps of WADG can be expressed as polynomial multiplication and L^2 projection, which we carry out using fast Bernstein algorithms. The proposed approach reduces the overall computational complexity from $O(N^{2d})$ to $O(N^{d+1})$ in d dimensions. Numerical experiments illustrate the accuracy of the proposed approach, and computational experiments for a GPU implementation of BBWADG verify that this theoretical complexity is achieved in practice.

Keywords: discontinuous Galerkin, Bernstein, high order, heterogeneous media, GPU

1. Introduction

Efficient and accurate simulations of wave propagation are central to applications in seismology, where heterogeneities arise from the presence of different geological structures in the subsurface. Accurate and efficient numerical methods for wave problems are becoming more and more important as the demand for solutions of large-

*Corresponding author: Email: kaihang.guo@rice.edu; Tel.: +1-281-702-8829;

scale problems increases. This paper presents an efficient discontinuous Galerkin (DG) method for wave equations in heterogeneous media with sub-cell variations. DG methods combine advantages of the finite volume method and the finite element method, which providing high order accuracy and addressing complex geometries through the use of unstructured meshes. These methods are straightforward to parallelize and can be accelerated by taking advantage of high performance architectures such as Graphics Processing Units (GPUs) [2].

High order methods are especially attractive for wave propagation problems. The simulation of wave propagation is observed to be more robust to grid distortion at high orders than at low orders [3, 4], and numerical dispersion and dissipation errors are small for high order approximations [5]. The goal of this work is to address two issues related to high order DG methods for wave propagation: computational cost at high orders and accurate resolution of media with sub-cell heterogeneities. Nodal DG methods, which are popular implementations of DG for wave propagation problems [6], have a high computational complexity with respect to the order of approximation. We aim to reduce this computational complexity using Bernstein polynomials [7].

Bernstein polynomials have been previously utilized by Ainsworth at el. [8] and Kirby [9] to reduce computational costs associated with high order continuous finite element methods on simplices. More recent work has exploited properties of Bernstein polynomials for DG methods. For example, Kirby introduced a fast algorithm in [10] to invert the local mass matrix in DG schemes by exploiting a recursive block structure present under a Bernstein basis.

Chan and Warburton later introduced a Bernstein-Bézier discontinuous Galerkin (BBDG) method based on the "strong" DG formulation [11]. In contrast to the approach of Kirby [10], the use of the "strong" formulation avoids explicitly introducing a mass matrix inverse, and instead formulates the DG formulation in terms of differentiation and lifting matrices. BBDG exploits the facts that, in d dimensions, the derivative and the lift matrices can be recast as a combination of sparse matrices. By exploiting this structure, the right-hand side of BBDG can be evaluated in $O(N^d)$ operations per element. In comparison, the dense linear algebra of nodal DG methods generally results in a computational complexity of $O(N^{2d})$ per element.

A separate challenge in the simulation of wave propagation is the approximation of media heterogeneities. High order finite difference methods are widely used [12] in practice, but face challenges for complex geometries and non-smooth media [13]. The spectral element method (SEM) [14] provides one alternative to explicit high order finite difference methods. SEM produces a diagonal global mass matrix, making it well-suited for explicit time-stepping, and can accommodate both complex geometries (through unstructured meshes) and discontinuous media. However, SEM is restricted to quadrilateral and hexahedral meshes, which are less geometrically flexible than tetrahedral meshes. Several modifications have been proposed to extend SEM to triangular and tetrahedral meshes, but they require non-standard approximation spaces and do not support arbitrarily high order approximations [15].

An alternative to triangular and tetrahedral SEM are high order DG methods. High order DG methods can accommodate unstructured triangular and tetrahedral meshes, and naturally result in a block-diagonal global mass matrix, making them amenable to explicit time-stepping schemes and complex geometries. However, in most DG implementations for heterogeneous media, the discretization is based on the assumption that wavespeed is piecewise constant over each element [16]. Fewer DG methods address the case when wavespeed varies within an element. Castro et al. [17] addressed sub-element variations in wavespeed by recasting the wave equation as a new linear hyperbolic PDE with variable coefficients and source terms, which are non-zero in the presence of sub-element variations in wavespeed. However, this method introduces additional source terms and stiffness matrices with variable coefficients, resulting in a more complex formulation. Additionally, semi-discrete energy stability is not guaranteed.

Mercerat and Glinsky [18] proposed instead replacing the mass matrix by a weighted mass matrix, where the wavespeed acts as a weight function. The weighted mass matrix is obtained by introducing a set of quadrature points for the material approximation and computing integrals for entries of the mass matrix through quadrature rules. This modification does not require new stiffness matrices or source terms, and can be shown to be energy stable and high order accurate. However, because the wavespeed varies from element to element, each local weighted mass matrix is different. Thus, one needs to

store inverses of weighted mass matrices over each element for time-explicit schemes, which significantly increases storage costs. Because GPUs have limited memory, these high storage costs restrict the problem sizes that can be run on a single GPU. Moreover, increased storage costs lead to more data movement, which is becoming increasingly expensive compared to the cost of floating point operations [19].

To address these storage costs, we utilize a weight-adjusted approximation of the weighted mass matrix, whose inverse can be applied in a low-storage manner. The idea of a weight-adjusted approximation to a weighted mass matrix was first introduced as "reverse numerical integration" in [20], though it was not analyzed in detail. The idea was independently reintroduced and analyzed by Chan et al. in [1]. The key idea is to approximate the weighted L^2 inner product using an equivalent weight-adjusted inner product, which produces provably high order accurate and energy stable DG methods with low storage requirements. Since WADG only modifies the local mass matrix, it maintains much of the structure of DG methods and is able to reuse existing DG implementations.

The main computational step of WADG is the computation of a quadrature-based polynomial L^2 projection. However, the implementation of the quadrature-based L^2 projection in WADG requires $O(N^{2d})$ operations, while complexity of BBDG is only $O(N^d)$. Hence, combining BBDG with WADG would result in the cost of the quadrature-based L^2 projection dominating the implementation at high polynomial degrees. The goal of this work is to reduce the computational complexity of WADG at high orders of approximation, which we do using Bernstein bases. We develop an efficient algorithm to implement the polynomial L^2 projection in terms of Bernstein coefficients, which leads to a Bernstein-Bézier WADG (BBWADG) method. The main idea is to decompose the projection operator into a combination of degree elevation operators. Due to the sparsity of the one-degree elevation matrices, the L^2 projection can be applied in $O(N^{d+1})$ operations, reducing the complexity of right-hand evaluation from $O(N^6)$ to $O(N^4)$ in three dimensions.

The paper is organized as follows. In Section 3, we review the weight-adjusted DG discretization of the first order acoustic and elastic wave equations in heterogeneous media. Section 4 introduces a Bernstein-Bézier DG method and its fast implementa-

tion. In Section 5, we propose a Bernstein-Bézier weight-adjusted DG method, based on an algorithm to efficiently apply the polynomial L^2 projection under Bernstein bases. Section 6 presents numerical validation and verification.

2. Mathematical notation

In this paper, we focus on wave problems in three dimensions since BBWADG can reduce the computational complexity by two orders. In contrast, only one order of complexity can be reduced in two dimensions.

We assume the physical domain Ω is well approximated by a triangulation Ω_h consisting of K non-overlapping elements D^k. The reference tetrahedron is defined as follows

$$\widehat{D} = \{(r,s,t) \geq -1; r+s+t \leq -1\}.$$

We assume that each element D^k is the image of the reference element $\widehat{D}$ under an affine mapping $\mathbf{\Phi}^k$

$$\boldsymbol{x} = \mathbf{\Phi}^k \widehat{\boldsymbol{x}}, \qquad \boldsymbol{x} \in D^k, \;\; \widehat{\boldsymbol{x}} \in \widehat{D},$$

where $\boldsymbol{x} = (x,y,z)$ are physical coordinates on the kth element and $\widehat{\boldsymbol{x}} = (r,s,t)$ are coordinates on the reference element. Over each element D^k, we define the approximation space $V_h\left(D^k\right)$ as

$$V_h\left(D^k\right) = \mathbf{\Phi}^k \circ V_h\left(\widehat{D}\right),$$

where $V_h\left(\widehat{D}\right)$ is a polynomial approximation space of degree N on the reference element. For the reference tetrahedron, $V_h\left(\widehat{D}\right)$ is defined as follows

$$V_h\left(\widehat{D}\right) = P^N\left(\widehat{D}\right) = \left\{r^i s^j t^k, \;\; 0 \leq i+j+k \leq N\right\}.$$

In three dimensions, Bernstein polynomials on a tetrahedron are expressed using barycentric coordinates. The barycentric coordinates for the reference tetrahedron are given as

$$\lambda_0 = -\frac{(1+r+s+t)}{2}, \quad \lambda_1 = \frac{(1+r)}{2}, \quad \lambda_2 = \frac{(1+s)}{2}, \quad \lambda_3 = \frac{(1+t)}{2}.$$

The Nth degree Bernstein basis is simply as a scaling of the barycentric monomials

$$B^N_{ijkl} = C^N_{ijkl} \lambda_0^i \lambda_1^j \lambda_2^k \lambda_3^l, \qquad C^N_{ijkl} = \frac{N!}{i!j!k!l!}, \qquad i+j+k+l = N,$$

which forms a nonnegative partition of unity. For simplicity, we introduce the multi-index $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_d)$ to denote the tuple of barycentric indices (i, j, k, l). We define the order of a multi-index as

$$|\boldsymbol{\alpha}| := \sum_{i=0}^{d} \alpha_i.$$

We take $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$ to mean that $\alpha_j \leq \beta_j$, $\forall j = 0, \ldots, d$.

3. Weight-adjusted Discontinuous Galerkin methods

The following sections introduce weight-adjusted DG discretizations of acoustic and elastic wave equations.

3.1. Acoustic wave equation

We consider a first order velocity-pressure formulation of the acoustic wave equation given as

$$\begin{aligned} \frac{1}{c^2}\frac{\partial p}{\partial \tau} + \nabla \cdot \boldsymbol{u} &= 0, \\ \frac{\partial \boldsymbol{u}}{\partial \tau} + \nabla p &= 0, \end{aligned} \tag{1}$$

where p is the acoustic pressure, $\boldsymbol{u} \in \mathbb{R}^d$ is the vector of velocities in each coordinate direction and c is the wavespeed. We assume that (1) is posed over time $\tau \in [0, T)$ on the physical domain Ω with boundary $\partial\Omega$, and the wavespeed is bounded by

$$0 < c_{\min} \leq c(\boldsymbol{x}) \leq c_{\max} < \infty.$$

We define the jump across element interfaces as

$$[\![p]\!] = p^+ - p, \qquad [\![\boldsymbol{u}]\!] = \boldsymbol{u}^+ - \boldsymbol{u},$$

where $p^+, \boldsymbol{u}^+$ and $p, \boldsymbol{u}$ are the neighboring and local traces of the solution over each interface, respectively. The average across an element interface is denoted by

$$\{\{p\}\} = \frac{1}{2}\left(p^+ + p\right), \qquad \{\{\boldsymbol{u}\}\} = \frac{1}{2}\left(\boldsymbol{u}^+ + \boldsymbol{u}\right).$$

We discretize the acoustic wave equation (1) in space using a strong formulation and choose penalty fluxes as

$$\boldsymbol{u}^* = \{\{\boldsymbol{u}\}\} - \frac{\tau_p}{2}[\![p]\!]\boldsymbol{n}, \qquad p^* = \{\{p\}\} - \frac{\tau_u}{2}[\![\boldsymbol{u}]\!]\cdot\boldsymbol{n},$$

where $\boldsymbol{n}$ is the outward unit normal vector on D^k. The corresponding semi-discrete formulation is given as follows

$$\begin{aligned}
\int_{D^k} \frac{1}{c^2}\frac{\partial p_h^k}{\partial \tau}\phi d\boldsymbol{x} &= -\int_{D^k} \nabla\cdot\boldsymbol{u}_h^k \phi d\boldsymbol{x} + \int_{\partial D^k}\frac{1}{2}\left(\tau_p[\![p_h^k]\!] - \boldsymbol{n}\cdot[\![\boldsymbol{u}_h^k]\!]\right)\phi d\boldsymbol{x}, \\
\int_{D^k} \frac{\partial\left(\boldsymbol{u}_h^k\right)_i}{\partial\tau}\psi_i d\boldsymbol{x} &= -\int_{D^k}\frac{\partial p_h^k}{\partial \boldsymbol{x}_i}\psi_i d\boldsymbol{x} + \int_{\partial D^k}\frac{1}{2}\left(\tau_u[\![\boldsymbol{u}_h^k]\!]\cdot\boldsymbol{n} - [\![p_h^k]\!]\right)\psi_i \boldsymbol{n}_i d\boldsymbol{x},
\end{aligned} \tag{2}$$

where ϕ, ψ are test functions and $\tau_p, \tau_u \geq 0$ are penalty parameters.

We define the mass matrix $\boldsymbol{M}$ and the face mass matrix $\boldsymbol{M}_f$ on $\widehat{D}$ as

$$(\boldsymbol{M})_{ij} = \int_{\widehat{D}} \phi_i(\widehat{\boldsymbol{x}})\phi_j(\widehat{\boldsymbol{x}})d\widehat{\boldsymbol{x}}, \qquad \left(\boldsymbol{M}_f\right)_{ij} = \int_{f_{\widehat{D}}} \phi_i(\widehat{\boldsymbol{x}})\phi_j(\widehat{\boldsymbol{x}})d\widehat{\boldsymbol{x}}.$$

where $f_{\widehat{D}}$ is a face of the reference element $\widehat{D}$ and $\{\phi_i\}_{i=1}^{N_p}$ is an Nth degree polynomial basis on $\widehat{D}$. Through an affine mapping $\boldsymbol{\Phi}^k$, we can map the local operators on D^k to the reference operators

$$\boldsymbol{M}^k = J^k\boldsymbol{M}, \qquad \boldsymbol{M}_f^k = J_f^k\boldsymbol{M}_f,$$

where J^k is the determinant of the volume Jacobian and J_f^k is the determinant of the face Jacobian for f. Similarly, the weighted mass matrix $\boldsymbol{M}_w^k$ on D^k are given by

$$\left(\boldsymbol{M}_w^k\right)_{ij} = J^k\int_{\widehat{D}} w\left(\boldsymbol{\Phi}^k\widehat{\boldsymbol{x}}\right)\phi_i\left(\widehat{\boldsymbol{x}}\right)\phi_j\left(\widehat{\boldsymbol{x}}\right)d\widehat{\boldsymbol{x}}.$$

The stiffness matrix on D^k with respect to x is defined as

$$\left(\boldsymbol{S}_x^k\right)_{ij} = \int_{D^k} \varphi_i \frac{\partial \varphi_j}{\partial x} d\boldsymbol{x},$$

and $\boldsymbol{S}_y^k, \boldsymbol{S}_z^k$ are defined similarly with respect to y and z. Through chain rule, we can express stiffness matrices on D^k in terms of the reference stiffness matrices $\boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{S}_3$ with respect to reference coordinates r, s and t, respectively. Then, the semi-discrete formulation (2) can be written as

$$\begin{aligned}
\boldsymbol{M}_{1/c^2}^k \frac{d\boldsymbol{p}}{d\tau} &= -J^k \sum_{i=1}^{d} \left(\boldsymbol{G}_{i1}^k \boldsymbol{S}_1 + \boldsymbol{G}_{i2}^k \boldsymbol{S}_2 + \boldsymbol{G}_{i3}^k \boldsymbol{S}_3\right) \boldsymbol{U}_i + \sum_{f=1}^{N_{\text{faces}}} J_f^k \boldsymbol{M}_f F_p, \\
J^k \boldsymbol{M} \frac{d\boldsymbol{U}_i}{d\tau} &= -J^k \left(\boldsymbol{G}_{i1}^k \boldsymbol{S}_1 + \boldsymbol{G}_{i2}^k \boldsymbol{S}_2 + \boldsymbol{G}_{i3}^k \boldsymbol{S}_3\right) \boldsymbol{p} + \sum_{f=1}^{N_{\text{faces}}} J_f^k \boldsymbol{n}_i \boldsymbol{M}_f F_u,
\end{aligned} \tag{3}$$

where $\boldsymbol{p}$ and $\boldsymbol{U}_i$ are degrees of freedom for p and $\boldsymbol{u}_i$, and $\boldsymbol{G}^k$ is the matrix of geometric factors r_x, s_x, t_x, etc. The flux terms F_p, F_u are defined such that

$$\begin{aligned}
\left(\boldsymbol{M}_f F_p(\boldsymbol{p}, \boldsymbol{p}^+, \boldsymbol{U}, \boldsymbol{U}^+)\right)_j &= \int_{f_{\widehat{D}}} \frac{1}{2} \left(\tau_p [\![p]\!] - \boldsymbol{n} \cdot [\![\boldsymbol{u}]\!]\right) \phi_j d\widehat{\boldsymbol{x}}, \\
\left(\boldsymbol{n}_i \boldsymbol{M}_f F_u(\boldsymbol{p}, \boldsymbol{p}^+, \boldsymbol{U}, \boldsymbol{U}^+)\right)_j &= \int_{f_{\widehat{D}}} \frac{1}{2} \left(\tau_u [\![\boldsymbol{u}]\!] - [\![p]\!]\right) \psi_j \boldsymbol{n}_i d\widehat{\boldsymbol{x}}.
\end{aligned}$$

Inverting $\boldsymbol{M}_{1/c^2}^k$ and $\boldsymbol{M}$ in (3) produces a system of ODEs that can be solved by time-explicit methods.

When the wavespeed c^2 is approximated by a constant over each element, $\boldsymbol{M}_{1/c^2}^k = \frac{J^k}{c^2} \boldsymbol{M}$, and $\left(\boldsymbol{M}_{1/c^2}^k\right)^{-1} = \frac{c^2}{J^k} \boldsymbol{M}^{-1}$. Thus, to apply $\left(\boldsymbol{M}_{1/c^2}^k\right)^{-1}$, we need only store values of J^k, c^2 over each element and a single reference mass matrix inverse $\boldsymbol{M}^{-1}$ over the entire mesh.[1]. However, inverses of weighted mass matrices are distinct from element to element if the wavespeed possesses sub-element variations. Typical implementations precompute and store these weighted mass matrix inverese, which significantly increases the storage cost of high order DG schemes.

[1] In practice, the reference inverse mass matrix is incorporated into the definition of differentiation and lifting matrices on the reference element.

To address this issue, a weight-adjusted discontinuous Galerkin (WADG) is proposed in [21, 1], which is energy stable and high order accurate for sufficiently regular weight functions. WADG approximates the weighted mass matrix by a weight-adjusted approximation $\widetilde{\boldsymbol{M}}_w^k$ given as

$$\boldsymbol{M}_w^k \approx \widetilde{\boldsymbol{M}}_w^k = \boldsymbol{M}^k \left(\boldsymbol{M}_{1/w}^k\right)^{-1} \boldsymbol{M}^k.$$

Plugging above expression into (3), we obtain the semi-discrete WADG discretization of (1) as follows

$$\begin{aligned}
\frac{d\boldsymbol{p}}{d\tau} &= -\left(\boldsymbol{M}^k\right)^{-1} \boldsymbol{M}_{c^2}^k \left(\sum_{i=1}^{d}\sum_{j=1}^{d} \boldsymbol{G}_{ij}^k \boldsymbol{D}_j \boldsymbol{U}_i + \sum_{f=1}^{N_{\text{faces}}} \frac{J_f^k}{J^k} \boldsymbol{L}^f F_p\right), \\
\frac{d\boldsymbol{U}_i}{d\tau} &= -\left(\boldsymbol{G}_{i1}^k \boldsymbol{D}_1 + \boldsymbol{G}_{i2}^k \boldsymbol{D}_2 + \boldsymbol{G}_{i3}^k \boldsymbol{D}_3\right) \boldsymbol{p} + \sum_{f=1}^{N_{\text{faces}}} \frac{J_f^k}{J^k} \boldsymbol{n}_i \boldsymbol{L}^f F_u,
\end{aligned} \tag{4}$$

where $\boldsymbol{D}_i = \boldsymbol{M}^{-1}\boldsymbol{S}_i$ are derivative operators with respect to reference coordinates r,s,t, $\boldsymbol{L}^f = \boldsymbol{M}^{-1}\boldsymbol{M}_f$ are lift operators over faces.

3.2. Elastic wave equation

The weight-adjusted approach can be extended to matrix-valued weights, which appear in symmetrized first order velocity-stress formulations of the elastic wave equation [22]. Let ρ be the density and $\boldsymbol{C}$ be the symmetric matrix form of constitutive tensor relating stress and strain. The first-order elastic wave equations are given by

$$\begin{aligned}
\rho \frac{\partial \boldsymbol{v}}{\partial \tau} &= \sum_{i=1}^{d} \boldsymbol{A}_i^T \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{x}_i}, \\
\boldsymbol{C}^{-1} \frac{\partial \boldsymbol{\sigma}}{\partial \tau} &= \sum_{i=1}^{d} \boldsymbol{A}_i \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}_i},
\end{aligned} \tag{5}$$

where $\boldsymbol{v}$ is the velocity and $\boldsymbol{\sigma}$ is a vector consisting of unique entries of the symmetric stress tensor. The matrices $\boldsymbol{A}_i$ are given as

$$\boldsymbol{A}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad \boldsymbol{A}_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \boldsymbol{A}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

For isotropic media, $\boldsymbol{C}$ is given by

$$\boldsymbol{C} = \begin{pmatrix} 2\mu+\lambda & \lambda & \lambda & & \\ \lambda & 2\mu+\lambda & \lambda & & \\ \lambda & \lambda & 2\mu+\lambda & & \\ & & & \mu \boldsymbol{I}^{3\times 3} & \end{pmatrix},$$

where μ, λ are Lamé parameters. We note that $\boldsymbol{A}_i$ are spatially constant independently of media heterogeneities.

We can construct a semi-discrete DG scheme for elasticity analogous to the formulation (2) for the acoustic wave equation

$$\left(\rho \frac{\partial \boldsymbol{v}}{\partial \tau}, \boldsymbol{w}\right)_{L^2(D^k)} = \left(\sum_{i=1}^{d} \boldsymbol{A}_i^T \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{x}_i}, \boldsymbol{w}\right)_{L^2(D^k)} + \left\langle \frac{1}{2}\boldsymbol{A}_n^T [\![\boldsymbol{\sigma}]\!] + \frac{\tau_v}{2}\boldsymbol{A}_n^T \boldsymbol{A}_n [\![\boldsymbol{v}]\!], \boldsymbol{w} \right\rangle_{L^2(\partial D^k)},$$
$$\left(\boldsymbol{C}^{-1} \frac{\partial \boldsymbol{\sigma}}{\partial \tau}, \boldsymbol{q}\right)_{L^2(D^k)} = \left(\sum_{i=1}^{d} \boldsymbol{A}_i \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}_i}, \boldsymbol{q}\right)_{L^2(D^k)} + \left\langle \frac{1}{2}\boldsymbol{A}_n [\![\boldsymbol{v}]\!] + \frac{\tau_\sigma}{2}\boldsymbol{A}_n \boldsymbol{A}_n^T [\![\boldsymbol{\sigma}]\!], \boldsymbol{q} \right\rangle_{L^2(\partial D^k)},$$

where $(\cdot,\cdot)_{L(D^k)}$ and $\langle \cdot,\cdot \rangle_{L(D^k)}$ denote the L^2 inner product on D^k and ∂D^k, respectively.

The presence of $\boldsymbol{C}^{-1}$ on the left-hand side produces a matrix-valued mass matrix

$\boldsymbol{M}_{\boldsymbol{C}^{-1}}$ involving the constitutive stress tensor $\boldsymbol{C}$

$$\boldsymbol{M}_{\boldsymbol{C}^{-1}} = \begin{bmatrix} \boldsymbol{M}_{\boldsymbol{C}_{11}^{-1}} & \dots & \boldsymbol{M}_{\boldsymbol{C}_{1d}^{-1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{M}_{\boldsymbol{C}_{d1}^{-1}} & \dots & \boldsymbol{M}_{\boldsymbol{C}_{dd}^{-1}} \end{bmatrix},$$

where $\boldsymbol{C}_{ij}^{-1}$ denotes the ijth entry of $\boldsymbol{C}^{-1}$ and $\boldsymbol{M}_{\boldsymbol{C}_{ij}^{-1}}$ denotes the scalar weighted mass matrix with weight $\boldsymbol{C}_{ij}^{-1}$. The matrix $\boldsymbol{M}_{\boldsymbol{C}^{-1}}$ can be understood as the matrix-weighted analogue of the scalar wavespeed-weighted mass matrix $\boldsymbol{M}_{1/c^2}$ which appeared for the acoustic wave equation.

The inverse of $\boldsymbol{M}_{\boldsymbol{C}^{-1}}$ can be approximated by the inverse of a matrix-weighted weight-adjusted mass matrix

$$\boldsymbol{M}_{\boldsymbol{C}^{-1}}^{-1} \approx \left(\boldsymbol{I} \otimes \boldsymbol{M}^{-1}\right) \boldsymbol{M}_{\boldsymbol{C}} \left(\boldsymbol{I} \otimes \boldsymbol{M}^{-1}\right),$$

where $\otimes$ denotes the Kronecker product. We note that this approximation can be applied in terms of scalar weight-adjusted mass matrix inverses. Incorporating this approximation yields the following WADG scheme for the elastic wave equations (5)

$$\begin{aligned} \frac{\partial \boldsymbol{V}}{\partial \tau} &= \left(\boldsymbol{I} \otimes \left(\boldsymbol{M}^k\right)^{-1}\right) \boldsymbol{M}_{\rho^{-1}\boldsymbol{I}}^k \left(\sum_{i=1}^{d} \sum_{j=1}^{d} \boldsymbol{G}_{ij}^k \left(\boldsymbol{A}_i^T \otimes \boldsymbol{D}_j\right) \boldsymbol{\Sigma} + \frac{\boldsymbol{J}_f^k}{\boldsymbol{J}^k} \sum_{f=1}^{N_{\text{faces}}} \left(\boldsymbol{I} \otimes \boldsymbol{L}^f\right) \boldsymbol{F}_v \right), \\ \frac{\partial \boldsymbol{\Sigma}}{\partial \tau} &= \left(\boldsymbol{I} \otimes \left(\boldsymbol{M}^k\right)^{-1}\right) \boldsymbol{M}_{\boldsymbol{C}}^k \left(\sum_{i=1}^{d} \sum_{j=1}^{d} \boldsymbol{G}_{ij}^k \left(\boldsymbol{A}_i \otimes \boldsymbol{D}_j\right) \boldsymbol{V} + \frac{\boldsymbol{J}_f^k}{\boldsymbol{J}^k} \sum_{f=1}^{N_{\text{faces}}} \left(\boldsymbol{I} \otimes \boldsymbol{L}^f\right) \boldsymbol{F}_\sigma \right), \end{aligned} \tag{6}$$

where $\boldsymbol{V}, \boldsymbol{\Sigma}$ are constructed by concatenating $\boldsymbol{\Sigma}_i, \boldsymbol{V}_i$ into single vectors, respectively, and $\boldsymbol{F}_v, \boldsymbol{F}_\sigma$ are vectors representing the velocity and stress numerical fluxes. We note that this formulation is energy stable and high order accurate for elastic wave propagation in either isotropic or aniostropic heterogeneous media [21].

3.3. Quadrature-based implementation

In practice, weight-adjusted mass matrrix inverses are applied in a matrix-free fashion using sufficiently accurate quadrature rules. We follow [1] and use simplicial

quadratures which are exact for polynomials of degree $2N+1$ [23]. Let $\widehat{\boldsymbol{x}}_i, \widehat{\boldsymbol{w}}_i$ denote the quadrature points and weights on the reference element. We define the interpolation matrix $\boldsymbol{V}_q$ as

$$(\boldsymbol{V}_q)_{ij} = \phi_j(\widehat{\boldsymbol{x}}_i),$$

whose columns consist of values of basis functions at quadrature points. On each element D^k, we have

$$\boldsymbol{M}^k = J^k \boldsymbol{M} = J^k \boldsymbol{V}_q^T \operatorname{diag}(\widehat{\boldsymbol{w}}) \boldsymbol{V}_q, \quad \boldsymbol{M}_{c^2}^k = J^k \boldsymbol{V}_q^T \operatorname{diag}(\boldsymbol{d}) \boldsymbol{V}_q, \quad \boldsymbol{d}_i = \frac{\widehat{\boldsymbol{w}}_i}{c^2(\boldsymbol{\Phi}^k \widehat{\boldsymbol{x}}_i)}$$

where $\boldsymbol{\Phi}^k \widehat{\boldsymbol{x}}_i$ are quadrature points on D^k and $c^2\left(\boldsymbol{\Phi}^k \widehat{\boldsymbol{x}}\right)$ denote the values of the wavespeed at quadrature points. Evaluating the right hand side of (4) and (6) requires applying the product of an unweighted mass matrix and weighted mass matrix, such as $\left(\boldsymbol{M}^k\right)^{-1} \boldsymbol{M}_{c^2}^k$. This can be done using quadrature-based matrices as follows

$$\left(\boldsymbol{M}^k\right)^{-1} \boldsymbol{M}_{c^2}^k = \boldsymbol{P}_q \operatorname{diag}\left(\frac{1}{c^2(\boldsymbol{\Phi}^k \widehat{\boldsymbol{x}})}\right) \boldsymbol{V}_q, \tag{7}$$

where $\boldsymbol{P}_q = \boldsymbol{M}^{-1} \boldsymbol{V}_q^T \operatorname{diag}(\widehat{\boldsymbol{w}})$ is a quadrature-based polynomial L^2 projection operator on the reference element. Moreover, since $\boldsymbol{P}_q, \boldsymbol{V}_q$ are reference operators, the implementation of (7) requires only $O\left(N^d\right)$ storage for values of the wavespeed $c^2\left(\boldsymbol{\Phi}^k \widehat{\boldsymbol{x}}\right)$ at quadrature points for each element. In contrast, storing full weighted mass matrix inverses or factorizations requires $O\left(N^{2d}\right)$ storage on each element. For example, in three dimensions, the number of quadrature points on one element, scales with $O(N_p) = O(N^3)$, while size of the weighted mass matrix inverse is $O(N_p) \times O(N_p)$, implying an $O(N^6)$ storage requirement.

4. Bernstein-Bézier DG methods

In this section, we review how to use Bernstein-Bézier polynomial bases to construct efficient high order DG methods. For nodal DG methods, the numerical fluxes can be computed in terms of the difference between degrees of freedom at face nodes on two neighboring elements. This is also true of the Bernstein basis, since Bernstein

polynomials share a geometrical decomposition with vertex, edge, face and interior nodes in the sense that edge basis functions vanish at vertices, face basis functions vanish at vertices and edges, and interior basis functions vanish at vertices, edges, and faces [7]. Hence, the value of a Bernstein polynomial on one face is determined by basis functions associated with that face only, and the jumps of polynomial solutions under Bernstein bases across element interfaces can be computed similarly using node-to-node connectivity maps and degrees of freedom corresponding to face points on two neighboring elements.

Evaluating the DG formulation (4) requires applying derivative and lift operators. These steps can be accelerated using properties of Bernstein polynomials. Let $\boldsymbol{D}^i$ be the Bernstein derivative operator with respect to ith barycentric coordinate. Differentiation matrices with respect to reference coordinates can be expressed as a linear combination of barycentric differentiation matrices $\boldsymbol{D}^i$. It can be shown that each row of $\boldsymbol{D}^i$ has at most $d+1$ non-zeros in d dimensions [8, 9], such that the sparse application of barycentric Bernstein differentiation matrices requires only $O(N^d)$ operations. In contrast, nodal derivative operators are generally dense matrices of size $N_p \times N_p$, which require $O(N^{2d})$ operations to apply.

For a Bernstein lift operator $\boldsymbol{L}^f$, it was observed in [11] that $\boldsymbol{L}^f$ can be factorized as

$$\boldsymbol{L}^f = \boldsymbol{E}_L^f \boldsymbol{L}_0,$$

where $\boldsymbol{E}_L^f$ is the face reduction matrix and $\boldsymbol{L}_0$ is a sparse $N_p^f \times N_p^f$ matrix, where N_p^f is the number of degrees of freedom in the Nth degree polynomial space on a single face. Moreover, $\boldsymbol{L}_0$ has no more than seven nonzeros per row (independent of N).[2] The fixed bandwidth of the matrix $\boldsymbol{L}_0$ implies that it can be applied in $O(N^{d-1})$ operations. The face reduction operator $\boldsymbol{E}_L^f$ can be further expanded as product of one-degree reduction operators. Application of $\boldsymbol{E}_L^f$ requires applying N triangular one-degree reduction operators, each of which costs $O(N^{d-1})$ to apply. Hence, the total cost of the implementation of the lift matrix $\boldsymbol{L}^f$ is $O(N^d)$ in d dimensions. In contrast, the lift matrices

[2] Explicit expressions for $\boldsymbol{E}_L^f$ and $\boldsymbol{L}_0$ can be found in [11].

under a nodal basis have size $N_p \times N_p^f$ and cost $O(N^{2d-1})$ to apply.

To summarize, the overall cost of evaluating the DG right-hand side is $O(N^d)$ per element in d dimensions under a Bernstein basis. Since the number of degreees of freedom grows as $O(N^d)$, this complexity is optimal.

5. A fast implementation of weight-adjusted DG methods

While the evaluation of the BBDG right-hand side requires only $O(N^d)$ operations per element, this is true only if media is homogeneous (piecewise constant) over each element. Sub-element heterogeneities can be incorporated using WADG and numerical quadrature as discusssed in Section 3.3. However, because quadrature-based WADG involves dense matrix-vector products, the cost generally scales as $O(N^{2d})$ in d dimensions. Thus, naively utilizing WADG to address sub-cell heterogeneities results in a computational complexity of $O(N^{2d})$ per element, which will dominate the $O(N^d)$ complexity of BBDG and negate any gains in computational efficiency.

To address this, we propose a Bernstein-Bézier weight-adjusted discontinuous Galerkin (BBWADG) method based on a polynomial approximation of media heterogeneities. We first note that the evaluation of the DG right-hand side yields a polynomial of degree N. Let $u(x)$ denote this N polynomial, and let $\boldsymbol{u}$ denote its coefficients in some basis. WADG involves applying (7) to $\boldsymbol{u}$ to compute

$$\boldsymbol{P}_q \text{diag}\left(\frac{1}{c^2\left(\boldsymbol{\Phi}^k \widehat{\boldsymbol{x}}\right)}\right) \boldsymbol{V}_q \boldsymbol{u}.$$

Since $\boldsymbol{P}_q$ is a quadrature-based discretization of the L^2 projection operator, this is simply a quadrature-based L^2 projection of u/c^2 onto polynomials of degree N.

Suppose now that $1/c^2$ is a polynomial of degree M. Then, the main steps of WADG are equivalent to computing u/c^2, which is a polynomial of degree $M+N$, and projecting this polynomial onto degree N polynomials. These two steps correspond to polynomial multiplication and polynomial L^2 projection, both of which can be performed efficiently under Bernstein bases. The resulting algorithms require $O(N^{d+1})$ operations per element in d dimensions.

In practice, we construct a polynomial approximation of $1/c^2$ using a quadrature-based L^2 projection of the true wavespeed. Since the wavespeed does not generally change during a simulation, this approximation can be computed and stored once in a pre-processing step so that it does not affect the computational cost of the solver.

The remainder of this section describes efficient algorithms for computing the polynomial multiplication and polynomial L^2 projection of two Bernstein polynomials. This section is separated into four parts: in Section 5.1, we explain how to compute the product of two Bernstein polynomials as a higher degree Bernstein polynomial. We introduce Bernstein degree elevation matrices in Section 5.2, which are then used in Section 5.3 to construct a representation of the polynomial L^2 projection matrix which can be evaluated in $O(N^{d+1})$ operations. Finally, we present a GPU-accelerated algorithm of the Bernstein polynomial L^2 projection in Section 5.5.

5.1. Bernstein polynomial multiplication

Efficient algorithms exist for the multiplication of two Bernstein polynomials based on discrete convolutions [24]. We describe a sparse matrix-based implementation here, which is simpler to implement on GPUs.

Let $B^N_{\boldsymbol{\alpha}}$ and $B^M_{\boldsymbol{\beta}}$ be any two Bernstein basis functions of degree N and M respectively. Their product is

$$B^N_{\boldsymbol{\alpha}} B^M_{\boldsymbol{\beta}} = \frac{\binom{\boldsymbol{\alpha}+\boldsymbol{\beta}}{\boldsymbol{\alpha}}}{\binom{N+M}{N}} B^{N+M}_{\boldsymbol{\alpha}+\boldsymbol{\beta}},$$

which is a Bernstein basis function of degree $N+M$ up to a scaling. This observation can be used to efficiently compute the product of two Bernstein polynomials. Let $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ be two Bernstein polynomials of degree N and M respectively with representations

$$f(\boldsymbol{x}) = \sum_{|\boldsymbol{\alpha}|=N} a_{\boldsymbol{\alpha}} B^N_{\boldsymbol{\alpha}}(\boldsymbol{x}), \qquad g(\boldsymbol{x}) = \sum_{|\boldsymbol{\beta}|=M} b_{\boldsymbol{\beta}} B^M_{\boldsymbol{\beta}}(\boldsymbol{x}). \tag{8}$$

Then, $h(\boldsymbol{x}) = f(\boldsymbol{x})g(\boldsymbol{x})$ is a Bernstein polynomial of degree $N+M$.

We first illustrate polynomial multiplication for the $M = 1$ case, such that $g(\boldsymbol{x})$ is a

linear polynomial. Let $\boldsymbol{e}_j$ denote the canonical vector such that $g(\boldsymbol{x}) = \sum_{j=0}^{d} b_j B^1_{\boldsymbol{e}_j}(\boldsymbol{x})$. Then, the product of f, g is

$$\begin{aligned} h(\boldsymbol{x}) &= \sum_{j=0}^{d} \sum_{|\boldsymbol{\alpha}|=N} a_{\boldsymbol{\alpha}} b_j B^N_{\boldsymbol{\alpha}}(\boldsymbol{x}) B^1_{\boldsymbol{e}_j}(\boldsymbol{x}) \\ &= \sum_{j=0}^{d} \sum_{|\boldsymbol{\alpha}|=N} a_{\boldsymbol{\alpha}} b_j \frac{\alpha_j + 1}{N+1} B^{N+1}_{\boldsymbol{\alpha}+\boldsymbol{e}_j}(\boldsymbol{x}). \end{aligned} \tag{9}$$

Let $\boldsymbol{\gamma}$ be a multi-index and $c_{\boldsymbol{\gamma}}$ denote the coefficient of $B^{N+1}_{\boldsymbol{\gamma}}$ in the expression for h in (9). Then $c_{\boldsymbol{\gamma}}$ can be computed as

$$c_{\boldsymbol{\gamma}} = \sum_{j=0}^{d} a_{\boldsymbol{\gamma}-\boldsymbol{e}_j} b_j \frac{\gamma_j}{N+1}, \tag{10}$$

where we set the coefficient to be zero if the corresponding multi-index $\boldsymbol{\gamma} - \boldsymbol{e}_j$ has negative components. Hence, for the case $M = 1$, the Bernstein coefficients of $h(\boldsymbol{x})$ can be expressed as a linear combination of at most $d+1$ products of coefficients for $f(\boldsymbol{x})$ and coefficients for $g(\boldsymbol{x})$. This, in turn, can be efficiently computed using sparse matrix operations, as illustrated in Fig. 1.

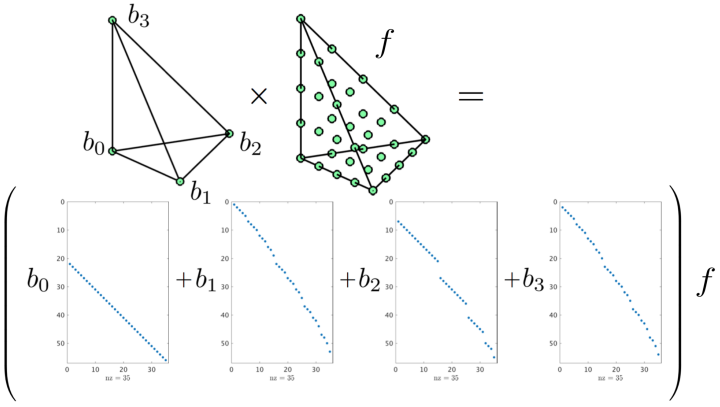


Figure 1: Visualization of Bernstein polynomial multiplication for $M = 1$

We now consider the more general case of arbitrrary M. We are interested in computing the product $h(\boldsymbol{x}) = f(\boldsymbol{x})g(\boldsymbol{x})$, where $f(\boldsymbol{x}) \in P^N$ and $g(\boldsymbol{x}) \in P^M$. We have the

following

$$h(\boldsymbol{x}) = \sum_{|\boldsymbol{\beta}|=M} \sum_{|\boldsymbol{\alpha}|=N} a_{\boldsymbol{\alpha}} b_{\boldsymbol{\beta}} B^N_{\boldsymbol{\alpha}}(\boldsymbol{x}) B^M_{\boldsymbol{\beta}}(\boldsymbol{x})$$
$$= \sum_{|\boldsymbol{\beta}|=M} \sum_{|\boldsymbol{\alpha}|=N} a_{\boldsymbol{\alpha}} b_{\boldsymbol{\beta}} \frac{\binom{\boldsymbol{\alpha}+\boldsymbol{\beta}}{\boldsymbol{\alpha}}}{\binom{N+M}{N}} B^{N+M}_{\boldsymbol{\alpha}+\boldsymbol{\beta}}(\boldsymbol{x}).$$

Hence, the coefficient $c_{\boldsymbol{\gamma}}$ of $B^{N+M}_{\boldsymbol{\gamma}}$ in $h(\boldsymbol{x})$ can be computed as

$$c_{\boldsymbol{\gamma}} = \sum_{|\boldsymbol{\beta}|=M} a_{\boldsymbol{\gamma}-\boldsymbol{\beta}} b_{\boldsymbol{\beta}} \frac{\binom{\boldsymbol{\gamma}}{\boldsymbol{\beta}}}{\binom{N+M}{N}}. \tag{11}$$

As in (10), the coefficient $c_{\boldsymbol{\gamma}}$ is set to be zero if the corresponding multi-index $\boldsymbol{\gamma}-\boldsymbol{\beta}$ has negative components. Hence, $c_{\boldsymbol{\gamma}}$ can be written as a combination of at most M_p products of coefficients from f and h, where M_p is the dimension of the Mth degree polynomial space. As in the $M = 1$ case, the multiplication of two arbitrary Bernstein polynomials can be implemented efficiently using sparse matrix multiplications.

We can also determine the computational complexity of Bernstein polynomial multiplication from the expression (11) for the product of two Bernstein polynomials. We summarize this in the following theorem:

Theorem 5.1. *The multiplication of two Bernstein polynomials of degree N and M can be performed in* $O\left((MN)^d\right)$ *operations. For fixed M, polynomial multiplication requires* $O(N^d)$ *operations.*

5.2. Bernstein degree elevation operators

In this section, we introduce degree elevation matrices, which are used within algorithms for polynomial L^2 projection in Section 5.3.. Degree elevation refers to the representation of a lower degree polynomial in a high degree polynomial basis. It can be shown that the d-dimensional Bernstein polynomial of degree $N-1$ can be expressed as a linear combination of no more than $d+1$ Bernstein polynomials of degree N [10]. For example, a basis function $B^{N-1}_{\boldsymbol{\alpha}}$ can be written as

$$B^{N-1}_{\boldsymbol{\alpha}} = \sum_{j=0}^{d} \frac{\alpha_j + 1}{N} B^N_{\boldsymbol{\alpha}+\boldsymbol{e}_j}, \tag{12}$$

where $\boldsymbol{e}_j$ is the jth canonical vector [9]. This property can be used to construct degree elevation matrices under the Bernstein basis. Let $\boldsymbol{E}_{N-i}^{N}$ denote the degree elevation operator, which evaluates a polynomial of degree $N-i$ as a degree N polynomials on a triangle. From (12), we know that the one-degree elevation matrix $\boldsymbol{E}_{N-1}^{N}$ is sparse, and only contains at most $d+1$ non-zero entries per row independently of the degree N.

Let $\boldsymbol{\alpha}$ denote the multi-index for the row corresponding to the basis function $B_{\boldsymbol{\alpha}}^{N-1}$. Then, the non-zero values and column indices $\boldsymbol{\beta}$ of $\boldsymbol{E}_{N-1}^{N}$ are

$$\left(\boldsymbol{E}_{N-1}^{N}\right)_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \frac{\alpha_j + 1}{N}, \quad \boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{e}_j, \quad j = 1, \ldots, d.$$

The degree elevation matrix $\boldsymbol{E}_{N-i}^{N}$ between arbitrary degrees can be expressed as the product of one-degree elevation matrices

$$\boldsymbol{E}_{N-i}^{N} = \boldsymbol{E}_{N-1}^{N} \boldsymbol{E}_{N-2}^{N-1} \cdots \boldsymbol{E}_{N-i}^{N-i+1}. \tag{13}$$

We also refer to the transpose of the degree elevation operator $\left(\boldsymbol{E}_{N-1}^{N}\right)^T$ as the degree reduction operator.

5.3. Bernstein polynomial L^2 projection

Recall that the two steps of BBWADG are polynomial multiplication and polynomial L^2 projection. The first step was discussed in Section 5.1, and we discuss the second step in this section. We introduce an efficient method of computing the L^2 projection of a polynomial to a lower degree polynomial under a Bernstein basis. This approach is based on a representation of the polynomial projection matrix in terms of sparse one-degree elevation matrices.

The representation of the polynomial L^2 projection matrix using degree elevation matrices is based on two observations. The first observation is that the polynomial L^2 projection operator is rectangular diagonal under a modal (orthogonal) basis. These modal basis functions [25, 26, 27, 28] are hierarchical and L^2 orthogonal, such that

$$\left(L_{\boldsymbol{\gamma}}, L_{\boldsymbol{\sigma}}\right) = \begin{cases} 1, & \boldsymbol{\gamma} = \boldsymbol{\sigma}, \\ 0, & \text{otherwise}, \end{cases}, \qquad L_{\boldsymbol{\gamma}} \in P^{|\boldsymbol{\gamma}|},$$

where γ and σ are d-dimensional multi-indices. For simplicity, we assume the hierarchical modal basis functions are arranged in ascending order with respect to $|\gamma|$.

The second observation is that the outer product of the degree elevation matrix and its transpose is diagonal under a modal basis. We wish to represent the polynomial L^2 projection matrix as a linear combination of these outer products. We recall some results from [11], which will be used in this proof.

Lemma 5.2 (Lemma A.2 in [11]). *Suppose $p \in P^N(\widehat{D})$. Let $\boldsymbol{T}$ be the transformation matrix mapping model coefficients to Bernstein coefficients such that*

$$p = \sum_{|\gamma| \leq N} c_\gamma^L L_\gamma = \sum_{|\alpha|=N} c_\alpha^B B_\alpha^N, \quad \boldsymbol{c}^B = \boldsymbol{T}\boldsymbol{c}^L,$$

where L_γ, B_α^N are modal and Bernstein polynomials, respectively. Define $\widetilde{\boldsymbol{D}}$ as

$$\widetilde{\boldsymbol{D}} = \boldsymbol{T}_{N-i}^{-1} \left(\boldsymbol{E}_{N-i}^N\right)^T \boldsymbol{T}_N$$

Suppose $0 \leq k \leq N$, and let $\lambda_k^N, \lambda_k^{N-i}$ be the distinct eigenvalues of $\boldsymbol{M}_N$ and $\boldsymbol{M}_{N-i}$, respectively. The entries of $\widetilde{\boldsymbol{D}}$ are

$$\widetilde{\boldsymbol{D}}_{\nu,\gamma} = \begin{cases} \lambda_{|\gamma|}^{N-i} / \lambda_{|\gamma|}^N, & \nu = \gamma, \\ 0, & \text{otherwise,} \end{cases} \qquad \widetilde{\boldsymbol{D}} \in \mathbb{R}^{(N-i)_p, N_p}$$

where $N_p, (N-i)_p$ are the dimensions of the space of polynomials of total degree N and $N-i$, respectively.

Corollary 1 (Corollary A.3 in [11]). *Under a transformation to a modal basis, $\boldsymbol{E}_{N-i}^N (\boldsymbol{E}_{N-i}^N)^T$ is diagonal, with entries*

$$\left(\boldsymbol{T}_N^{-1} \boldsymbol{E}_{N-i}^N \left(\boldsymbol{E}_{N-i}^N\right)^T \boldsymbol{T}_N\right)_{\gamma,\gamma} = \begin{cases} 0, & |\gamma| > (N-i), \\ \lambda_{|\gamma|}^{N-i} / \lambda_{|\gamma|}^N, & |\gamma| \leq (N-i). \end{cases}$$

A straightforward extension of Corollary 1 gives the following corollary:

Corollary 2. *Under a transformation to a modal basis, $\left\{\boldsymbol{E}_{N-i}^N (\boldsymbol{E}_{N-i}^N)^T\right\}_{i=0}^N$ is a basis*

for any $\boldsymbol{D}$ *such that*

$$\boldsymbol{D} = \begin{pmatrix} d_0 & & & \\ & d_1\boldsymbol{I}_1 & & \\ & & \ddots & \\ & & & d_N\boldsymbol{I}_N \end{pmatrix},$$

where $\boldsymbol{I}_i$ *is the identity matrix of dimension* $(i_p-(i-1)_p)\times(i_p-(i-1)_p)$.

Let $\boldsymbol{P}_N^{N+M}$ denote the Bernstein polynomial L^2 projection operator from the polynomial space of degree $N+M$ to the polynomial space of degree N. By transforming to a modal basis, we observe that the projection operator should be a diagonal rectangular matrix with diagonal entries equal to one, i.e.,

$$\boldsymbol{T}_N^{-1}\left(\boldsymbol{P}_N^{N+M}\right)\boldsymbol{T}_{N+M} = \left(\begin{array}{c|c} \boldsymbol{I} & \boldsymbol{0} \end{array}\right)$$

where $\boldsymbol{T}_N, \boldsymbol{T}_{N+M}$ are basis transformation matrices between Bernstein and modal bases of degree N and $N+M$ respectively. Based on this observation, we have the following theorem:

Theorem 5.3. *There exist* c_j, $0 \le j \le N$, *such that*

$$\boldsymbol{P}_N^{N+M} = \sum_{j=0}^{N} c_j \boldsymbol{E}_{N-j}^{N}\left(\boldsymbol{E}_{N-j}^{N}\right)^T\left(\boldsymbol{E}_N^{N+M}\right)^T. \tag{14}$$

Proof. From Lemma 5.2, we know that

$$\boldsymbol{T}_N^{-1}\left(\boldsymbol{E}_N^{N+M}\right)^T\boldsymbol{T}_{N+M} = \left(\begin{array}{cccc|ccc} \frac{\lambda_0^N}{\lambda_0^{N+M}} & & & & 0 & \cdots & 0 \\ & \frac{\lambda_1^N}{\lambda_1^{N+M}}\boldsymbol{I}_1 & & & \vdots & & \vdots \\ & & \ddots & & \vdots & & \vdots \\ & & & \frac{\lambda_N^N}{\lambda_N^{N+M}}\boldsymbol{I}_N & 0 & \cdots & 0 \end{array}\right),$$

which is a rectangular diagonal diagonal matrix. By Corollary 2, there exist c_j, $0 \le$

	c_0	c_1	c_2	c_3	c_4	c_5
$N=2,\ M=1$	0.6667	-0.0667				
$N=2,\ M=2$	1.0000	-0.3810	0.0238			
$N=3,\ M=1$	1.6000	-0.8000	0.1333	-0.0048		
$N=3,\ M=2$	1.8182	-1.2121	0.2273	-0.0087		
$N=4,\ M=1$	2.0833	-1.5152	0.4545	-0.0505	0.0013	
$N=4,\ M=2$	2.8846	-2.7972	0.9441	-0.1119	0.0029	
$N=5,\ M=1$	2.5714	-2.4725	1.0989	-0.2248	0.0180	-0.0003
$N=5,\ M=2$	4.2000	-5.3846	2.6923	-0.5874	0.0490	-0.0009

Table 1: Coefficients c_j for the Bernstein polynomial projection matrix $\boldsymbol{P}_N^{M+N}$ for different choices of degree N and M.

$j \leq N$, such that

$$\sum_{j=0}^{N} c_j \boldsymbol{T}_N^{-1} \boldsymbol{E}_{N-j}^{N} \left(\boldsymbol{E}_{N-j}^{N}\right)^T \boldsymbol{T}_N = \begin{pmatrix} \frac{\lambda_0^{N+M}}{\lambda_0^N} & & & \\ & \frac{\lambda_1^{N+M}}{\lambda_1^N} \boldsymbol{I}_1 & & \\ & & \ddots & \\ & & & \frac{\lambda_N^{N+M}}{\lambda_N^N} \boldsymbol{I}_N \end{pmatrix}.$$

Then, we obtain

$$\sum_{j=0}^{N} c_j \boldsymbol{T}_N^{-1} \boldsymbol{E}_{N-j}^{N} \left(\boldsymbol{E}_{N-j}^{N}\right)^T \left(\boldsymbol{E}_{N}^{N+M}\right)^T \boldsymbol{T}_{N+M} = \boldsymbol{T}_N^{-1} \left(\boldsymbol{P}_N^{N+M}\right) \boldsymbol{T}_{N+M}.$$

Multiplying $\boldsymbol{T}_N$ and $\boldsymbol{T}_{N+M}^{-1}$ from left and right hand side, respectively, gives

$$\boldsymbol{P}_N^{N+M} = \sum_{j=0}^{N} c_j \boldsymbol{E}_{N-j}^{N} \left(\boldsymbol{E}_{N-j}^{N}\right)^T \left(\boldsymbol{E}_{N}^{N+M}\right)^T. \tag{15}$$

□

In practice, these coefficient c_j can be computed by solving a linear system. Table 1 shows values of c_j for several combinations of degree N and M in three dimensions.

5.4. A note on fast mass matrix inversion

It should be noted that the approach described in Theorem 5.3 is in fact applicable to matrices beyond the polynomial projection matrix. For example, since the Bernstein

mass matrix is diagonal under a modal basis [10], the inverse Bernstein mass matrix can also be represented as a combination of degree elevation matrices. We start with an interesting observation in the proof of Lemma 5.2 (see [11]):

Lemma 5.4. *Let $\boldsymbol{M}_N$ be the Bernstein mass matrix of degree N. Under a transformation to a modal basis, the inverse $\boldsymbol{M}_N^{-1}$ is diagonal given by*

$$\boldsymbol{T}_N^{-1}\boldsymbol{M}_N^{-1}\boldsymbol{T}_N = \begin{pmatrix} \frac{1}{\lambda_0^N} & & & \\ & \frac{1}{\lambda_1^N}\boldsymbol{I}_1 & & \\ & & \ddots & \\ & & & \frac{1}{\lambda_N^N}\boldsymbol{I}_N \end{pmatrix}, \tag{16}$$

where λ_j^N is the jth distinct eigenvalue of $\boldsymbol{M}_N$.

Applying Corollary 2 to (16) directly, we obtain the following theorem:

Theorem 5.5. *There exist c_j, $0 \leq j \leq N$, such that, the inverse of Bernstein mass matrix can be written as*

$$\boldsymbol{M}_N^{-1} = \sum_{j=0}^{N} c_j \boldsymbol{E}_{N-j}^N \left(\boldsymbol{E}_{N-j}^N\right)^T. \tag{17}$$

Using (17), the inverse of a Bernstein mass matrix can be represented as a linear combination of sparse Bernstein degree elevation matrices. Thus, we can apply $\boldsymbol{M}_N^{-1}$ using the expression (20), which requires $O(N^4)$ operations in 3D. Since WADG requires only applications of $\boldsymbol{V}_q$ and $\boldsymbol{P}_q = \boldsymbol{M}^{-1}\boldsymbol{V}_q^T\boldsymbol{W}$, by combining fast mass matrix inversion with efficient $O(N^4)$ algorithms for evaluating Bernstein polynomials at quadrature points [8], it is possible to implement 3D quadrature-based WADG in $O(N^4)$ total operations.[3]

In light of these results, one may then ask why we bother with the strategy presented in Section 5, which involves both approximation of the weight function and specialized algorithms for polynomial multiplication and polynomial L^2 projection. The answer

[3]Fast Bernstein mass matrix inversion could also be performed using the algorithm described in [10]. However, as noted by Kirby, this approach is more involved and may be difficult to implement efficiently on GPUs.

lies in the nature of the coefficients c_j. We observe that, when representing the Bernstein mass matrix inverse using (17), the coefficients c_j are highly oscillatory with large positive and negative components (see Table 2), which can result in significant numerical roundoff in the application of $\boldsymbol{M}^{-1}$ using (17). In contrast, the coefficients used to represent the Bernstein polynomial projection matrix are much less oscillatory (see Table 1) and result in less roundoff error.

We can estimate sensitivity of (15) and (17) to roundoff by computing

$$\sum_{j=0}^{N} |c_j| \tag{18}$$

In the context of numerical quadrature with negative weights, the quantity (18) is referred to as the condition number of a quadrature rule [29]. For $N = 7$, the value of (18) is approximately 1.67×10^7 for $\boldsymbol{M}^{-1}$. In contrast, for $N = 7$, the value of (18) for $\boldsymbol{P}_N^{M+N}$ is approximately 14.53 for $M = 1$ and 41.35 for $M = 2$.

We also investigated roundoff errors numerically by computing the difference between $\boldsymbol{M}^{-1}\boldsymbol{b} - \boldsymbol{e}$ (where $\boldsymbol{M}^{-1}$ is computed using backslash in Matlab) and the quantity

$$\sum_{j=0}^{N} c_j \boldsymbol{E}_{N-j}^{N} \left(\boldsymbol{E}_{N-j}^{N}\right)^T \boldsymbol{b} - \boldsymbol{e}.$$

Here, $\boldsymbol{e}$ is the vector of all ones and $\boldsymbol{b} = \boldsymbol{M}\boldsymbol{e}$. In the absence of roundoff errors, both quantities should be zero. However, for all N, the roundoff error in applying $\boldsymbol{M}^{-1}$ using (17) is larger than the roundoff error incurred when using Matlab's backslash directly. Since the Bernstein mass matrix $\boldsymbol{M}$ is already known to become highly ill-conditioned as N increases [8, 30], these numerical experiments suggest that evaluating $\boldsymbol{M}^{-1}$ using (17) is impractical for large N.

5.5. GPU algorithms

In this section, we describe GPU-accelerated algorithms for Bernstein polynomial multiplication and polynomial L^2 projection.

	c_0	c_1	c_2	c_3	c_4	c_5
$N=1$	15	-3				
$N=2$	157.5	-90	7.5			
$N=3$	1260	-1260	315	-15		
$N=4$	8662.5	-12600	5670	-840	26.25	
$N=5$	54054	-103950	69300	-18900	1890	-42

Table 2: Coefficient c_j for $\boldsymbol{M}^{-1}$ represented using (17) for different orders N.

5.5.1. Polynomial multiplication

For polynomial multiplication, we aim to compute Bernstein coefficients of of the product $h(\boldsymbol{x}) = f(\boldsymbol{x})g(\boldsymbol{x})$, where $f(\boldsymbol{x}), g(\boldsymbol{x})$ are Bernstein polynomials of degree N and degree M, respectively. From (11), we observe that each coefficient of $h(\boldsymbol{x})$ is a linear combination of at most M_p products of coefficients from f and g as follows

$$c_{\boldsymbol{\gamma}} = \sum_{|\boldsymbol{\beta}|=M} a_{\boldsymbol{\gamma}-\boldsymbol{\beta}} b_{\boldsymbol{\beta}} \frac{\binom{\boldsymbol{\gamma}}{\boldsymbol{\beta}}}{\binom{N+M}{N}} = \sum_{|\boldsymbol{\beta}|=M} a_{\boldsymbol{\gamma}-\boldsymbol{\beta}} b_{\boldsymbol{\beta}} \ell_{\boldsymbol{\beta}}, \tag{19}$$

where $a_{\boldsymbol{\gamma}-\boldsymbol{\beta}}$ and $b_{\boldsymbol{\beta}}$ are coefficients of f and g, respectively. In our implementation, we store the coefficients $\ell_{\boldsymbol{\beta}}$ in some sparse matrix, where the row and column indices correspond to the multi-indices $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, respectively. Each thread will load non-zero entries in a row of this matrix along with the corresponding coefficients b_j and $a_{\boldsymbol{\gamma}-\boldsymbol{e}_j}$, compute one of the coefficients $c_{\boldsymbol{\gamma}}$, and store the result into shared memory.

5.5.2. Polynomial L^2 projection

We now introduce an algorithm to evaluate the polynomial L^2 projection based on (15). Unfortunately, it is difficult to directly evaluate (15) in a low-complexity fashion. This is because the degree elevation matrices $\boldsymbol{E}_{N-j}^{N}$ transition from sparse to dense matrices as j increases. Instead, we evaluate (15) using an equivalent reformulation. By plugging (13) into (15), we can derive a "telescoping form" for $\boldsymbol{P}_N^{M+N}$ involving

sparse one-degree elevation matrices

$$\begin{aligned}\boldsymbol{P}_N^{N+M} &= \left(c_0\boldsymbol{I}+\boldsymbol{E}_{N-1}^{N}\left(c_1\boldsymbol{I}+\boldsymbol{E}_{N-2}^{N-1}(c_2\boldsymbol{I}+\cdots)\left(\boldsymbol{E}_{N-2}^{N-1}\right)^T\right)\left(\boldsymbol{E}_{N-1}^{N}\right)^T\right)\left(\boldsymbol{E}_{N}^{N+1}\right)^T\cdots\left(\boldsymbol{E}_{N+M-1}^{N+M}\right)^T \\ &= \widetilde{\boldsymbol{P}}_N\left(\boldsymbol{E}_{N}^{N+1}\right)^T\cdots\left(\boldsymbol{E}_{N+M-1}^{N+M}\right)^T\end{aligned} \tag{20}$$

where we have defined $\widetilde{\boldsymbol{P}}_N = \left(c_0\boldsymbol{I}+\boldsymbol{E}_{N-1}^{N}\left(c_1\boldsymbol{I}+\boldsymbol{E}_{N-2}^{N-1}(c_2\boldsymbol{I}+\cdots)\left(\boldsymbol{E}_{N-2}^{N-1}\right)^T\right)\left(\boldsymbol{E}_{N-1}^{N}\right)^T\right)$. We next provide an algorithm to efficiently evaluate this telescoping expression on GPUs.

The first step in applying $\boldsymbol{P}_N^{N+M}$ is to apply the product of degree reduction matrices $\left(\boldsymbol{E}_{N}^{N+1}\right)^T\cdots\left(\boldsymbol{E}_{N+M-1}^{N+M}\right)^T$. Since each of these matrices is sparse and requires $O(N^d)$ operations to apply, this step has an overall computational complexity of $O(N^d)$ for fixed M.

The next step applies $\widetilde{\boldsymbol{P}}_N$ to the degree-reduced result. We separate the application of $\widetilde{\boldsymbol{P}}_N$ into two parts. The first part applies the one-degree reduction matrices in a "downward" sweep, while the second applies the one-degree elevation matrices in an "upward" sweep (see Fig. 2 for an illustration). Both the application of degree elevation or reduction operators and accumulate results during each step simultaneously.

We briefly describe our GPU implementation used to apply $\widetilde{\boldsymbol{P}}_N$. Let $\boldsymbol{p}$ be some vector to which we will apply $\widetilde{\boldsymbol{P}}_N$. In the first step, we set $\boldsymbol{p}_s = \boldsymbol{p}$, then compute the product of $\left(\boldsymbol{E}_{i-1}^{i}\right)^T$ and the matrix-vector product $\boldsymbol{p}_s$ stored in shared memory. More specifically, each thread computes the dot product of a sparse row of $\left(\boldsymbol{E}_{i-1}^{i}\right)^T$ with the vector $\boldsymbol{p}_s$. The resulting output vector $\boldsymbol{q}_s$ will be stored in another shared memory array and transfered to $\boldsymbol{p}_s$ after all threads complete their computation. At the same time, $\boldsymbol{q}_s$ will be scaled by the constant c_γ in (19) and stored in thread-local register memory.

For the second part, we compute the product of $\boldsymbol{E}_{i-1}^{i}$ and the vector $\boldsymbol{p}_s$ in shared memory, and accumulate results with the values in register memory during each step. More specifically, each thread computes the dot product of a sparse row of $\boldsymbol{E}_{i-1}^{i}$ with $\boldsymbol{p}_s$, and the result will be added to the corresponding value in register memory. After the accumulation, the values in register memory will be transfered to $\boldsymbol{p}_s$ in shared memory,

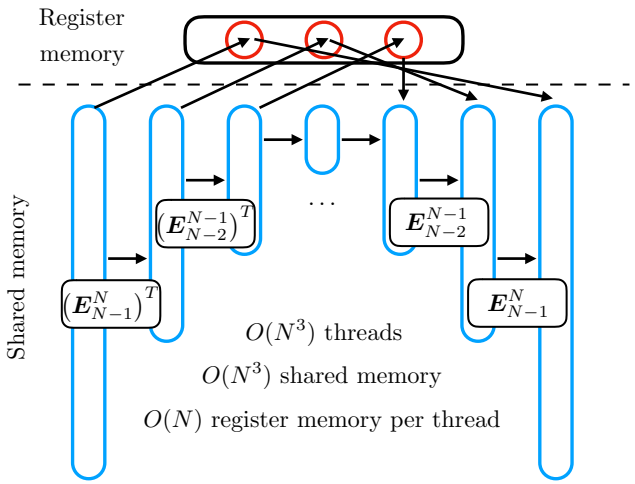


Figure 2: Illustration of GPU algorithm for the polynomial L^2 projection

which will be used in the next step.

In our algorithm, the multiplication of two Bernstein polynomials can be computed in $O(N^d)$ operations. For the polynomial L^2 projection, each application of $\boldsymbol{E}_{i-1}^{i}$ or $\left(\boldsymbol{E}_{i-1}^{i}\right)^T$ requires $O(N^d)$ operations. We need to apply N one-degree elevation operators and $N+M$ one-degree reduction operators, resulting in a total asymptotic complexity of $O(N^{d+1})$ for fixed M. This reduces the computational complexity of the projection step in WADG from $O(N^6)$ to $O(N^4)$ in three dimensions.

6. Numerical results

In this section, we examine the accuracy and performance of BBWADG. For clarity, we refer to WADG as the quadrature-based weight-adjusted discontinuous Galerkin method. This section is divided into four parts: in Section 6.1, we discuss accuracy of BBWADG using the method of manufactured solutions; In Section 6.2, we test BBWADG for wavespeed with different frequencies; in Section 6.3, we present runtime comparisons between BBWADG and WADG; in Section 6.4, we present results which quantify the computational efficiency of BBWADG.

6.1. Convergence for heterogeneous media

In this section, we investigate the convergence of BBWADG to manufactured solutions. In two dimensions, we assume that the pressure $p(x,y,\tau)$ is of the form

$$p(x,y,\tau) = \sin(\pi x)\sin(\pi y)\cos(\pi\tau). \tag{21}$$

We take the corresponding velocity vector as follows

$$\boldsymbol{u} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\cos(\pi x)\sin(\pi y)\sin(\pi\tau) \\ -\sin(\pi x)\cos(\pi y)\sin(\pi\tau) \end{pmatrix}.$$

Because this is not a solution of the acoustic wave equation in heterogeneous media, we utilize the method of manufactured solutions and add a source term $f(x,y,\tau)$ for which $p(x,y,\tau)$ is a solution. Plugging $p, \boldsymbol{u}$ into (1), we obtain the source term f

$$\begin{aligned} f(x,y,\tau) &= \frac{1}{c^2(x,y)}\frac{\partial p}{\partial \tau} + \nabla\cdot\boldsymbol{u} \\ &= -\frac{1}{c^2(x,y)}\pi\sin(\pi x)\sin(\pi y)\sin(\pi\tau) + 2\pi\sin(\pi x)\sin(\pi y)\sin(\pi\tau) \\ &= \left(2 - \frac{1}{c^2(x,y)}\right)\pi\sin(\pi x)\sin(\pi y)\sin(\pi\tau). \end{aligned}$$

Similarly, in three dimensions, we assume the pressure $p(x,y,z,\tau)$ satisfies

$$p(x,y,z,\tau) = \sin(\pi x)\sin(\pi y)\sin(\pi z)\cos(\pi\tau).$$

We can compute the corresponding velocity vector as follows

$$\boldsymbol{u} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} -\cos(\pi x)\sin(\pi y)\sin(\pi z)\sin(\pi\tau) \\ -\sin(\pi x)\cos(\pi y)\sin(\pi z)\sin(\pi\tau) \\ -\sin(\pi x)\sin(\pi y)\cos(\pi z)\sin(\pi\tau) \end{pmatrix}.$$

Plugging $p, \boldsymbol{u}$ into (1), we obtain the source term f

$$\begin{aligned} f(x,y,z,\tau) &= \frac{1}{c^2(x,y,z)}\frac{\partial p}{\partial \tau} + \nabla \cdot \boldsymbol{u} \\ &= \left(3 - \frac{1}{c^2(x,y,z)}\right)\pi \sin(\pi x)\sin(\pi y)\sin(\pi z)\sin(\pi \tau). \end{aligned}$$

In numerical experiments, we choose the wavespeed as

$$c^2(x,y,z) = 1 + \frac{1}{2}\sin(\pi x)\sin(\pi y)$$

for two dimensions and

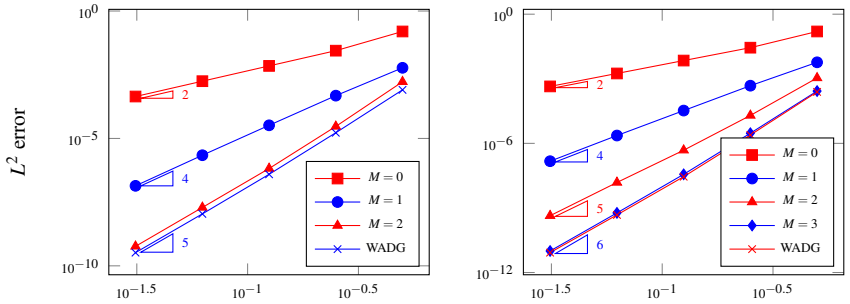
$$c^2(x,y,z) = 1 + \frac{1}{2}\sin(\pi x)\sin(\pi y)\sin(\pi z).$$

for three dimensions. In BBWADG, we project c^2 onto a polynomial space of degree M in L^2 sense.

Fig. 3 and Fig. 4 show the convergence of BBWADG and WADG to the manufactured solution under mesh refinement. The 3D uniform meshes used in our experiments are generated by GMSH [31]. From these plots, we observe that the convergence rate of BBWADG is $O(h^r)$, where $r = 2$ when $M = 0$ and $r = \min\{N+1, M+3\}$ when $M \geq 1$. We note that rates of convergence only observed when c^2 is approximated using the polynomial L^2 projection onto P^M. For other approximations (e.g. piecewise linear interpolation), the convergence rates are $O(h^M)$ in general.

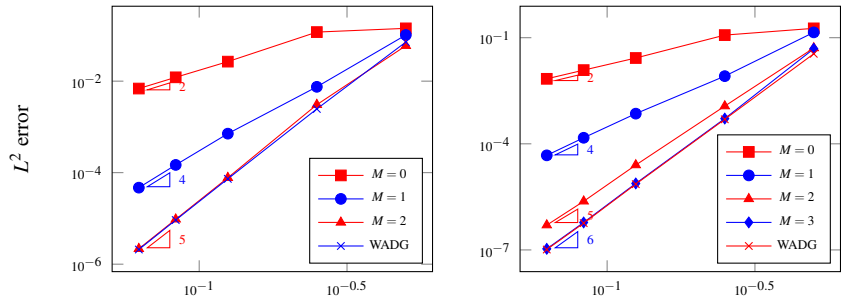
It should be noted that these rates of convergence are better than those suggested by an initial error analysis. It is straightforward to extend the error analysis of [1, 21] to accomodate approximations of $c^2 \in P^M$. However, this extension predicts that, when c^2 is approximated using L^2 projection onto degree M polynomials, the L^2 error should converge at a rate of $O(h^{M+1})$. This rate is observed only for $M = 0$, and the source of the discrepancy between the predicted and observed rates for $M > 0$ is presently unclear to the authors.

Increasing from $M = 0$ to $M = 1$ increases the observed rate of convergence by 2 orders. In contrast, increasing M further only increases the observed rate of conver-



(a) Convergence for $N = 4$ (b) Convergence for $N = 5$

Figure 3: Convergence under mesh refinement (2D)



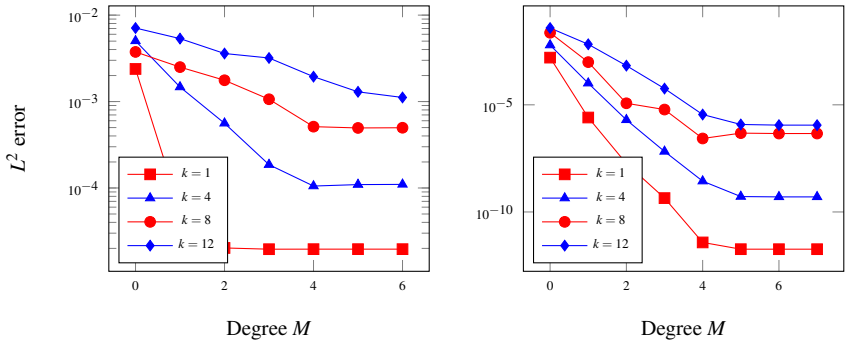
(a) Convergence for $N = 4$ (b) Convergence for $N = 5$

Figure 4: Convergence under mesh refinement (3D).

gence by one order for each degree past $M = 1$. For this reason, $M = 1$ may be an attractive choice for practical computations, since it provides a larger improvement in terms of rates of convergence relative to the increase in computational cost.

6.2. Wavespeed with different frequencies

Since the accuracy of the polynomial approximation of the wavespeed depends on M, we examine how the error depends on the approximability of c^2. We test BBWADG



(a) 2D convergence

(b) 3D convergence

Figure 5: Convergence of L^2 error when approximating wavespeeds given by (22).

using the following wavespeeds

$$\begin{aligned} c^2(x,y) &= 1 + \frac{1}{2}\sin(k\pi x)\sin(k\pi y) \qquad \text{(2D)}, \\ c^2(x,y,z) &= 1 + \frac{1}{2}\sin(k\pi x)\sin(k\pi y)\sin(k\pi z) \qquad \text{(3D)}, \end{aligned} \tag{22}$$

with different frequencies k. However, the manufactured solution remains the same independently of k.

This experiment is intended to show how the error depends on the approximability of the wavespeed. For higher k, c^2 is more oscillatory and harder to approximate; thus, we expect that the error should increase as k increases, despite the fact that the exact solution is independent of k.

We compute L^2 errors on a fixed mesh for various choices of k, choose $N = 7$ and a uniform mesh with $h = 0.0625$ for 2D experiments, and choose $N = 6$ and a uniform mesh with $h = 0.0833$ for 3D experiments. From Fig. 5, we observe that, for a fixed M, the accuracy of the method does indeed depend on the frequency of wavespeed: the lower frequency is (or the smaller k is), the smaller the error, despite the fact that the solution remains the same for all k.

6.3. Runtime comparisons

In this section, we present runtime comparisons between BBWADG and quadrature-based WADG for $M = 1$ and $M = 2$. In Section 5.5, we showed that the computational

complexity of BBWADG is $O(N^{d+1})$ for a fixed M. In this section, we will verify that this complexity is observed in practice, though the constant depends on M. All results are run on an Nvidia GTX 980 GPU, and the solvers are implemented in the Open Concurrent Compute Abstraction framework (OCCA) [32] for clarity and portability.

6.3.1. Computational implementation

A time-explicit DG scheme consists of the evaluation of the right hand side and the solution update. Its implementation is typically divided into three kernels.

- A volume kernel, which evaluates contributions to the right hand side resulting from volume terms in (4). Specifically, the volume kernel evaluates derivatives of local solutions over each element.
- A surface kernel, which evaluates numerical fluxes and contributions to the right hand side resulting from the surface terms in (4). More specifically, the surface kernel computes numerical fluxes and applies the lift matrix.
- An update kernel, which updates the solution in time. We use a low-storage 4th order Runge-Kutta method [33] in this thesis.

We adopt the same volume and surface kernels from [11]. BBWADG and WADG are implemented within the update kernel by modifying the right hand side computed in the volume and surface kernels.

6.3.2. Acoustic wave equations

In this experiment, we apply both BBWADG and WADG to the acoustic wave equation (1). Runtimes for the update kernels are given in Fig. 6. The case of $M = 1$ is denoted by BBWADG-1, while $M = 2$ is denoted by BBWADG-2.

	$N = 3$	$N = 4$	$N = 5$	$N = 6$	$N = 7$	$N = 8$
WADG	2.02e-8	4.91e-8	1.20e-7	2.19e-7	4.87e-7	5.25e-6
BBWADG-1	2.09e-8	3.32e-8	6.56e-8	8.54e-8	1.35e-7	1.65e-7
Speedup	0.9665	1.4789	1.8292	2.5644	3.6074	31.8182

Table 3: Achieved speedup for $M = 1$

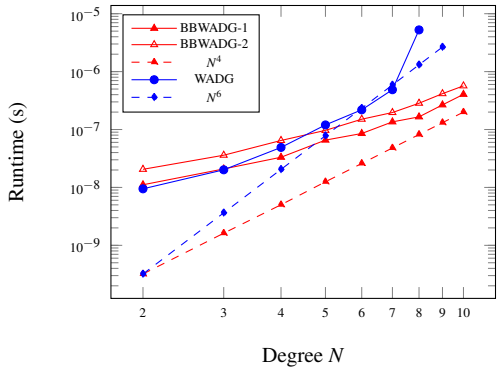


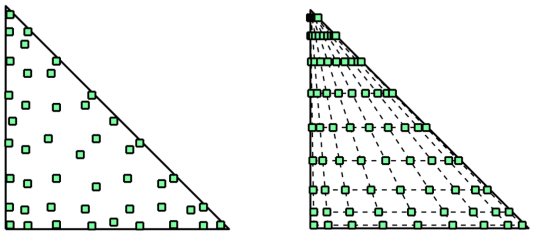
Figure 6: Per-element runtimes of update kernels using BBWADG and WADG on a mesh of 7854 elements (acoustic).

	$N=3$	$N=4$	$N=5$	$N=6$	$N=7$	$N=8$
WADG	2.02e-8	4.91e-8	1.20e-7	2.19e-7	4.87e-7	5.25e-6
BBWADG-2	3.60e-8	6.47e-8	9.67e-8	1.51e-7	1.97e-7	2.84e-7
Speedup	0.5611	0.7589	1.2409	1.4503	2.4721	18.4859

Table 4: Achieved speedup for $M = 2$

From Fig. 6, we observe that BBWADG is more expensive than WADG for low orders $(N < 4)$. However, runtime of the WADG update kernel increases more rapidly with N, displaying an asymptotic complexity of $O(N^6)$. On the other hand, the runtime of the BBWADG update kernel increases more slowly and displays a complexity of $O(N^4)$ as proven in Section 5.5.

Table 3 displays observed speedups of BBWADG over WADG. We find that for $N = 7$, the BBWADG update kernel for $M = 1$ achieves a 3.6 times speedup over the WADG update kernel. For $N = 8$, we observe an unexpected over 30 times speedup. However, we should note that this result is due to the use of different quadratures between $N = 7$ and $N = 8$. We choose a tetrahedral quadrature from Xiao and Gimbutas [23] which is exact for degree $2N+1$ polynomials for $N \leq 7$. For $N = 8$, this implies that the quadrature rule should be exact for polynomials of degree 17. However, the publicly available quadrature rules are only exact up to degree 15 polynomials. Because optimized quadrature points were not publicly available for $N > 7$, we switch to a collapsed coordinate quadrature [34] for $N > 7$ (see Fig. 7). Since the construction of



(a) $N = 7$ quadrature (b) $N = 8$ quadrature

Figure 7: Visualization of the quadrature points on the reference triangle

quadrature points is different, one should not compare results for degrees $N \leq 7$ with degrees $N > 7$.

Table 4 shows observed speedups for $M = 2$. We observe that the BBWADG update kernel for $M = 2$ is slower than WADG until $N = 5$. This is due to several reasons. First, increasing from $M = 1$ to $M = 2$ does not change the overall computational complexity with respect to N, but it does change the constant, which scales as $O(M^d)$. Secondly, for $M = 1$, since we know *a-priori* that the sparse matrices involved in polynomial multiplication contain only $d + 1 = 4$ nonzeros per row in 3D, we can store such matrices using `float4` and `int4` data structures, which have a slightly faster access time on GPUs [11]. This convenient storage structure is not available for $M = 2$.

6.3.3. Elastic wave equation

In this experiment, we compute runtimes for both BBWADG and WADG applied the elastic wave equations (5). Computational results for $M = 1$ and $M = 2$ are presented in Fig. 8.

We observe that the runtime behaves similarly to the acoustic case. The runtime of the BBWADG update kernel increases roughly as $O(N^4)$ up to $N = 8$, with about a 2.2 times speedup achieved for $M = 1$ and $N = 7$. However, the runtime of BBWADG increases more rapidly than $O(N^4)$ for $N > 8$. We expect that this is due to GPU occupancy/memory effects.

The application of $\widetilde{\boldsymbol{P}}_N$ described in Section 5.5.2 requires storage of $(N+1)$ intermediate values per thread for each application of a scalar weight-adjusted inverse mass matrix. For the scalar acoustic wave equation, this additional storage is negligible, as

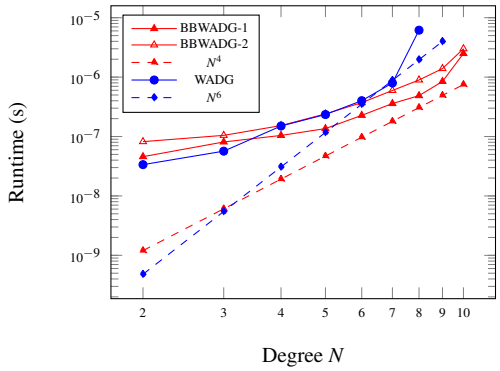


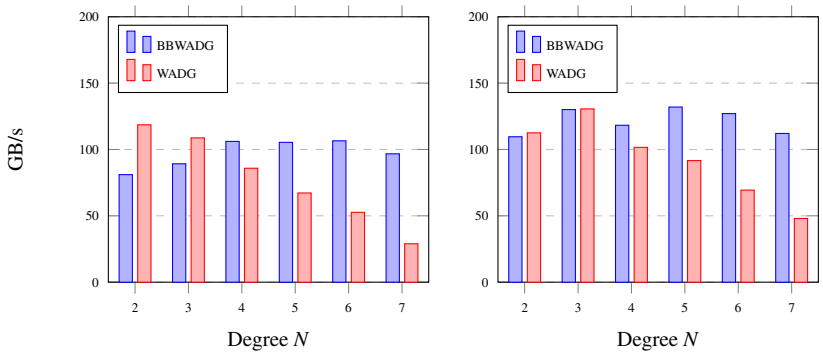
Figure 8: Per-element runtimes for update kernels using BBWADG and WADG on a mesh of 7854 elements (elastic).

only a single weight-adjusted inverse mass matrix is applied per element. However, for the elastic wave equation, we apply a matrix-weighted weight-adjusted inverse mass matrix, which is computed by applying multiple scalar weight-adjusted inverse mass matrices and combining the results. For elastic wave propagation in 3D, this increases the per-thread memory cost by a factor of 6 (corresponding to each of the six components of the elastic stress tensor), resulting in significant register pressure and reduced GPU occupancy..

This additional storage can be decreased by processing fewer components simultaneously; however, processing fewer components simultaneously also reduces data reuse and temporal locality. It is not immediately clear whether this approach will result in an overall lower runtime, and will be the subject of future investigation.

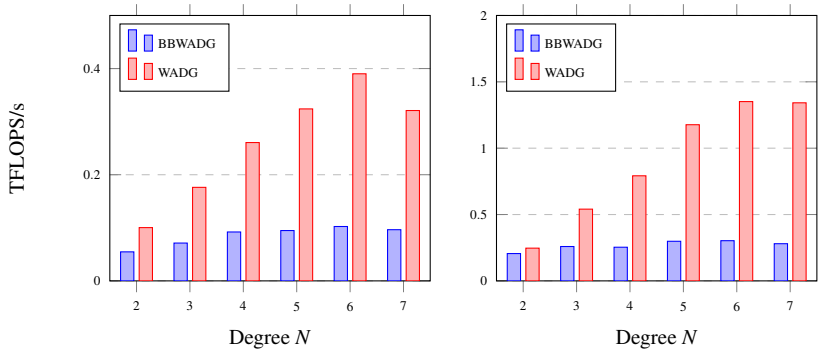
6.4. Performance analysis

In this section, we present computational results for BBWADG with $M = 1$ and WADG. Fig. 9 and Fig. 10 show the profiled computational performance and bandwidth of the BBWADG and WADG update kernels. From Fig. 9, we observe that the bandwidth of the WADG update kernel decreases steadily as N increases. In comparison, the BBWADG update kernel sustains a near-constant bandwidth as N increases. From Fig. 10, we can see that, for all N, the BBWADG update kernel achieves a lower computational performance compared to the WADG update kernel. These results are



(a) Acoustic wave equation (b) Elastic wave equation

Figure 9: Achieved bandwidth (GB/s) for update kernels using BBWADG and WADG.



(a) Acoustic wave equation (b) Elastic wave equation

Figure 10: Achieved computational performance (TFLOPS/s) for update kernels using BBWADG and WADG.

similar to those achieved for BBDG with piecewise constant wavespeeds [11].

7. Conclusion and future work

In this paper, we present a Bernstein-Bézier discontinuous Galerkin (BBWADG) method to simulate acoustic and elastic wave propagation in heterogeneous media based on a polynomial approximation of sub-cell heterogeneities and fast algorithms for Bernstein polynomial multiplication and L^2 projection. The resulting solver inherits the advantages of WADG (provable energy stability, high order accuracy) while

reducing the computational complexity of the update kernel from $O(N^{2d})$ to $O(N^{d+1})$ in d dimensions. Moreover, this implementation reuses the BBDG volume and surface kernels from [11], both of which can be applied in $O(N^d)$ operations. Thus, the total computational complexity of the BBWADG solver is $O(N^{d+1})$ per timestep for a fixed polynomial approximation of sub-cell media heterogeneities.

Due to its low computational complexity, BBWADG offers advantages in simulating wave propagation in heterogeneous media using higher order approximations. These properties make BBWADG promising for accurate and efficient simulation of large-scale wave propagation problems.

Acknowledgments

The authors acknowledge the support of the National Science Foundation under awards DMS-1719818 and DMS-1712639.

References

[1] J. Chan, R. J. Hewett, T. Warburton, Weight-adjusted discontinuous Galerkin methods: wave propagation in heterogeneous media, SIAM Journal on Scientific Computing 39 (2017) A2935–A2961.

[2] A. Klöckner, T. Warburton, J. Bridge, J. S. Hesthaven, Nodal discontinuous Galerkin methods on graphics processors, Journal of Computational Physics 228 (2009) 7863–7882.

[3] D. De Grazia, G. Mengaldo, D. Moxey, P. Vincent, S. Sherwin, Connections between the discontinuous Galerkin method and high-order flux reconstruction schemes, International journal for numerical methods in fluids 75 (2014) 860–877.

[4] F. Lörcher, G. Gassner, C.-D. Munz, An explicit discontinuous Galerkin scheme with local time-stepping for general unsteady diffusion equations, Journal of Computational Physics 227 (2008) 5649–5670.

[5] M. Ainsworth, Dispersive and dissipative behaviour of high order discontinuous Galerkin finite element methods, Journal of Computational Physics 198 (2004) 106–130.

[6] J. S. Hesthaven, T. Warburton, Nodal discontinuous Galerkin methods algorithms, analysis, and applications, Springer, 2008.

[7] R. T. Farouki, The Bernstein polynomial basis: A centennial retrospective, Computer Aided Geometric Design 29 (2012) 379–419.

[8] M. Ainsworth, G. Andriamaro, O. Davydov, Bernstein–Bézier finite elements of arbitrary order and optimal assembly procedures, SIAM Journal on Scientific Computing 33 (2011) 3087–3109.

[9] R. C. Kirby, Fast simplicial finite element algorithms using Bernstein polynomials, Numerische Mathematik 117 (2011) 631–652.

[10] R. C. Kirby, Fast inversion of the simplicial Bernstein mass matrix, Numerische Mathematik 135 (2017) 73–95.

[11] J. Chan, T. Warburton, GPU-Accelerated Bernstein–Bézier Discontinuous Galerkin Methods for Wave Problems, SIAM Journal on Scientific Computing 39 (2017) A628–A654.

[12] J. Virieux, H. Calandra, R.-E. Plessix, A review of the spectral, pseudo-spectral, finite-difference and finite-element modelling techniques for geophysical imaging, Geophysical Prospecting 59 (2011) 794–813.

[13] W. W. Symes, T. Vdovina, Interface error analysis for numerical wave propagation, Computational Geosciences 13 (2009) 363–371.

[14] D. Komatitsch, J.-P. Vilotte, The spectral element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures, Bulletin of the seismological society of America 88 (1998) 368–392.

[15] M. Chin-Joe-Kong, W. A. Mulder, M. Van Veldhuizen, Higher-order triangular and tetrahedral finite elements with mass lumping for solving the wave equation, Journal of Engineering Mathematics 35 (1999) 405–426.

[16] J. Chan, Z. Wang, A. Modave, J.-F. Remacle, T. Warburton, GPU-accelerated discontinuous Galerkin methods on hybrid meshes, Journal of Computational Physics 318 (2016) 142–168.

[17] C. E. Castro, M. Käser, G. B. Brietzke, Seismic waves in heterogeneous material: subcell resolution of the discontinuous Galerkin method, Geophysical Journal International 182 (2010) 250–264.

[18] E. D. Mercerat, N. Glinsky, A nodal high-order discontinuous Galerkin method for elastic wave propagation in arbitrary heterogeneous media, Geophysical Journal International 201 (2015) 1101–1118.

[19] V. Volkov, Understanding latency hiding on gpus, Ph.D. thesis, UC Berkeley, 2016.

[20] C. Koutschan, C. Lehrenfeld, J. Schöberl, Computer algebra meets finite elements: an efficient implementation for Maxwell's equations, in: Numerical and Symbolic Scientific Computing, Springer, 2012, pp. 105–121.

[21] J. Chan, Weight-adjusted discontinuous Galerkin methods: Matrix-valued weights and elastic wave propagation in heterogeneous media, International Journal for Numerical Methods in Engineering 113 (2018) 1779–1809.

[22] T. J. Hughes, J. E. Marsden, Classical elastodynamics as a linear symmetric hyperbolic system, Journal of Elasticity 8 (1978) 97–110.

[23] H. Xiao, Z. Gimbutas, A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions, Computers & mathematics with applications 59 (2010) 663–676.

[24] J. Sánchez-Reyes, Algebraic manipulation in the Bernstein form made simple via convolutions, Computer-Aided Design 35 (2003) 959–967.

[25] O. Owens, et al., Polynomial solutions of the cylindrical wave equation, Duke Mathematical Journal 23 (1956) 371–383.

[26] M. Dubiner, Spectral methods on triangles and other domains, Journal of Scientific Computing 6 (1991) 345–390.

[27] T. Koornwinder, Two-variable analogues of the classical orthogonal polynomials, in: Theory and application of special functions, Elsevier, 1975, pp. 435–495.

[28] J. Proriol, Sur une famille de polynômes à deux variables orthogonaux dans un triangle, Comptes Rendus Academic des Sciences Paris 245 (1957) 2459–2461.

[29] H. Antil, S. E. Field, F. Herrmann, R. H. Nochetto, M. Tiglio, Two-step greedy algorithm for reduced order quadratures, Journal of Scientific Computing 57 (2013) 604–637.

[30] J. Chan, T. Warburton, A Short Note on a Bernstein–Bezier Basis for the Pyramid, SIAM Journal on Scientific Computing 38 (2016) A2162–A2172.

[31] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities, International journal for numerical methods in engineering 79 (2009) 1309–1331.

[32] D. S. Medina, A. St-Cyr, T. Warburton, OCCA: A unified approach to multi-threading languages, arXiv preprint arXiv:1403.0968 (2014).

[33] M. H. Carpenter, C. A. Kennedy, Fourth-order 2N-storage Runge-Kutta schemes (1994).

[34] Karniadakis. G, S. J. Sherwin, Spectral/hp Element Methods for CFD, Oxford University Press, 1999.