High-order entropy stable discontinuous Galerkin methods for the shallow water equations: curved triangular meshes and GPU acceleration

Xinhui Wu[1], Ethan J. Kubatko[2], and Jesse Chan[1]

[1]Department of Computational and Applied Mathematics, Rice University
[2]Department of Civil, Environmental and Geodetic Engineering, The Ohio State University

October 16, 2020

Abstract

We present a high-order entropy stable discontinuous Galerkin (ESDG) method for the two dimensional shallow water equations (SWE) on curved triangular meshes. The presented scheme preserves a semi-discrete entropy inequality and remains well-balanced for continuous bathymetry profiles. We provide numerical experiments which confirm the high-order accuracy and theoretical properties of the scheme, and compare the presented scheme to an entropy stable scheme based on simplicial summation-by-parts (SBP) finite difference operators. Finally, we report the computational performance of an implementation on Graphics Processing Units (GPUs) and provide comparisons to existing GPU-accelerated implementations of high-order DG methods on quadrilateral meshes.

1 Introduction

The aim of this paper is to present and compare two high-order entropy stable discontinuous Galerkin (DG) schemes. The first is a modal DG formulation of the shallow water equations (SWE), for which the volume and surface quadrature rules can be chosen arbitrarily [1]. The second scheme uses triangular summation-by-parts finite difference operators whose construction is based on carefully chosen quadrature rules which satisfy certain accuracy conditions and contain boundary points [2]. We also implemented both schemes on GPUs for computational acceleration. The analysis and optimization of the multi-threading OCCA code will be discussed.

The shallow water equations (SWE) are a popular mathematical model for fluid flows in rivers, lakes and coastal regions, where the horizontal scales are much greater than the vertical ones. The shallow water equations in 2D are [3]

$$h_t + (hu)_x + (hv)_y = 0, \tag{1.1}$$

$$(hu)_t + (hu^2 + gh^2/2)_x + (huv)_y = -ghb_x, \tag{1.2}$$

$$(hv)_t + (huv)_x + (hv^2 + gh^2/2)_y = -ghb_x. \tag{1.3}$$

The height of the water is denoted by $h = h(x, y, t)$, as measured from the bottom, and should be positive $h > 0$. The velocity in the x direction is denoted by $u = u(x, y, t)$ and the velocity in the y direction is denoted by $v = v(x, y, t)$. The gravitational constant is denoted by g. The bathymetry height is denoted by $b = b(x, y)$, and is assumed constant over time. The subscript $(.)_t$ denotes the time derivative, while the subscripts $(.)_x$ or $(.)_y$ denote directional derivatives along x and y axis, respectively. We also define the height of water free surface $H = h + b$.

The SWE are derived from the Navier-Stokes equations, which describe the motion of incompressible fluids such as water. The Navier-Stokes equations are themselves derived from the equations for conservation of mass and linear momentum. By specifying boundary conditions for the Navier-Stokes equations for a water column and integrating over the depth of the column, one obtains the SWE system [4]. One of the most prominent practical applications of the SWE is the numerical prediction of storm surges under extreme weather conditions like hurricanes near coastal regions [5, 6].

In addition to being accurate and efficient, numerical methods for the SWE should also preserve certain solutions. Of particular importance is the preservation of steady state solutions, also known as the "lake at rest" condition [3]:

$$H = \text{constant}, \ u = v = 0. \tag{1.4}$$

A numerical scheme that preserves this steady state is said to be well-balanced [7, 8]. Schemes that are not well-balanced can generate spurious waves in the presence of varying bottom topographies. A good numerical method for SWE should capture both steady states and their small perturbations so as to avoid the generation of spurious waves [9]. Accomplishing this discretely can be challenging, especially for discontinuous bottom topographies, where special discretizations of the source terms are required [9, 10, 11].

A natural strategy to achieve greater accuracy in numerical simulations is to use higher order methods. This leads to low dissipation errors and long-time accuracy, which are important in simulations of waves. Numerical schemes for the SWE should also address issues unique to the SWE equations, such as well-balancedness and wetting and drying. Finally, since the SWE are non-linear, the solution may become discontinuous even if the initial condition is smooth. For many numerical methods, stability is a challenge when solving non-linear PDEs, especially in the presence of discontinuous solutions.

Common numerical methods including the finite difference method, the finite volume method and the finite element method have been applied to the SWE system. This paper focuses on the DG method, which combines advantages of finite element and finite volume methods. DG methods provide a natural path to high-order accuracy and accommodate complex geometries through unstructured meshes. Furthermore, DG methods are simple to parallelize and can take advantage of acceleration using GPU.

Early methods for the SWE were typically low-order accurate and utilized structured grids, which are less geometrically flexible. High-order DG methods address these shortcomings, but introduce issues of stability. For example, higher order polynomials tend to oscillate in larger magnitude near a discontinuous shock and can result in blow-up of the solution. However, recent work on entropy-stable high-order DG methods [9, 1, 12] provide a way to address such instabilities. Entropy stable DG methods can also be extended to curved meshes, which can be necessary when dealing with complex geometries.

Traditional entropy stable DG formulations have relied on specific finite difference summation-by-parts (SBP) operators, which are constructed using carefully designed quadrature rules which

contain boundary points while satisfying certain accuracy conditions. High-order entropy stable DG schemes were more recently extended to "modal" formulations, which allow for arbitrary pairings of basis functions and volume/surface quadrature rules. Entropy stable and well-balanced modal DG formulations were introduced for the SWE on Cartesian quadrilateral meshes in [12]. Our goal is to extend these results to curved triangular meshes while ensuring the satisfaction of key properties such as well-balancedness. We examine the numerical behavior of this scheme, and analyze the computational performance of the method with special attention to GPU optimization.

The new contributions in this paper as follows: in addition to extending entropy stable DG methods for the SWE to curved triangular meshes, we provide a connection between traditional SBP operators and hybridized SBP operators used in "modal" entropy stable DG formulations. We also analyze the efficiency of GPU implementations of entropy stable DG methods on triangular meshes. In [3], it was shown that GPU implementations of entropy stable methods on quadrilateral meshes do not introduce additional computational cost compared as traditional collocation-type DG schemes. We demonstrate in this work that, while GPU acceleration does provide significant speedups, the cost of entropy stable DG schemes remains higher than the cost of traditional DG schemes on triangular meshes.

This paper begins with a review of entropy stable modal DG schemes for the two dimensional shallow water equations in Section 2. In Section 3, we briefly discuss the SBP formulation and its link with the modal DG formulation. We present numerical results which validate theoretical properties of our schemes in Section 4. Section 5 provides a description of our GPU implementation and optimization details of the OCCA code. We conclude in Section 6 with a summary of results.

2 DG method on the 2D shallow water equations

2.1 Mathematical assumption and notations

We first introduce some underlying mathematical assumptions and notations for our DG method. For consistency, we reuse notation from [1], with slight modifications to provide a cleaner discrete formulation. We denote the triangular reference element by $\hat{D}$ with boundary $\partial\hat{D}$. The vertices of the reference triangle are $(-1,-1)$, $(-1,1)$ and $(1,-1)$. We use $\hat{n}_i$ to represent the ith component of the outward normal vector scaled by the face Jacobian on the boundary of the reference element. The space of polynomials up to degree N on the reference element is defined as

$$P^N(\hat{D}) = \{\hat{x}^i\hat{y}^j, \quad (\hat{x},\hat{y}) \in \hat{D}, \quad 0 \le i+j \le N\}. \tag{2.1}$$

Finally, we denote the dimension of the P^N as $N_p = \dim(\mathrm{P^N}(\hat{\mathrm{D}}))$.

We wish to build foundations for a discrete matrix-vector formulation of our DG method. We assume the solution on the reference element $u(\boldsymbol{x}) \in P^N(\hat{D})$ such that it can be represented in some polynomial basis $\{\phi_i\}_{i=1}^{N_p}$ of degree up to N, as:

$$u(\boldsymbol{x}) = \sum_{i=1}^{N_p} u_i\phi_i(\hat{\boldsymbol{x}}), \qquad P^N(\hat{D}) = \mathrm{span}\{\phi_i(\hat{\boldsymbol{x}})\}_{i=1}^{N_p}. \tag{2.2}$$

We denote the number of volume and surface quadrature nodes by N_q and N_q^f respectively. Moreover, we assume the volume quadrature rule $\{(\boldsymbol{x}_i, w_i)\}_{i=1}^{N_q}$ exactly integrates polynomials of degree

at least $(2N-1)$ on the reference element $\hat{D}$ and that the surface quadrature $\{(\boldsymbol{x}_i^f, w_i^f)\}_{i=1}^{N_q^f}$ integrates polynomials of degree at least $2N$ on the faces of $\hat{D}$.

Let $\boldsymbol{W}$ denote the diagonal $N_q \times N_q$ matrix whose entries are $\boldsymbol{W}_{ii} = w_i$, where $w_i > 0$ corresponds to volume quadrature weight. We also define the diagonal matrix $\boldsymbol{W}_f = \text{diag}(w_i^f)$ for the surface quadrature weights. We then define the volume and surface quadrature interpolation matrices $\boldsymbol{V}_q$ and $\boldsymbol{V}_f$ as:

$$(\boldsymbol{V}_q)_{ij} = \phi_j(\hat{\boldsymbol{x}}_i), \quad 1 \le j \le N_p, \quad 1 \le i \le N_q, \tag{2.3}$$

$$(\boldsymbol{V}_f)_{ij} = \phi_j(\hat{\boldsymbol{x}}_i^f), \quad 1 \le j \le N_p, \quad 1 \le i \le N_q^f, \tag{2.4}$$

Let $f(\boldsymbol{x})$ denote some polynomial in the basis ϕ_j with coefficients $\boldsymbol{f}$. The matrix $\boldsymbol{V}_q$ maps coefficients $\boldsymbol{f}$ to evaluations of $f(\boldsymbol{x})$ at volume quadrature points and, similarly, the matrix $\boldsymbol{V}_f$ interpolates f to surface quadrature points. For example:

$$\boldsymbol{f}_q = \boldsymbol{V}_q \boldsymbol{f}, \qquad (\boldsymbol{f}_q)_i = f(\hat{\boldsymbol{x}}_i), \qquad 1 \le i \le N_q. \tag{2.5}$$

We now define $\boldsymbol{D}^i$ as the differentiation matrix with respect to the ith coordinate. We may denote the matrices $\boldsymbol{D}^1, \boldsymbol{D}^2$ as $\boldsymbol{D}^x$ and $\boldsymbol{D}^y$ in two-dimensional case. $\boldsymbol{D}^i$ is defined implicitly with:

$$u(\boldsymbol{x}) = \sum_{i=1}^{N_p} u_i \phi_i(\hat{\boldsymbol{x}}), \qquad \frac{\partial u}{\partial \hat{x}_i} = \sum_{j=1}^{N_p} (\boldsymbol{D}^i \boldsymbol{u})_j \phi_j(\hat{\boldsymbol{x}}). \tag{2.6}$$

$\boldsymbol{D}^i$ maps the basis coefficients of some polynomial $\boldsymbol{u} \in P^N$ to coefficients of its ith directional derivative with respect to the reference coordinate $\boldsymbol{x}_i$.

With the matrix $\boldsymbol{V}_q$, we can now introduce the element mass matrix whose entries are the evaluations of inner products of different basis functions with quadrature points:

$$\boldsymbol{M} = \boldsymbol{V}_q^T \boldsymbol{W} \boldsymbol{V}_q, \quad \boldsymbol{M}_{ij} = \sum_{k=1}^{N_q} w_k \phi_j(\hat{\boldsymbol{x}}_k) \phi_i(\hat{\boldsymbol{x}}_k) \approx \int_{\hat{D}} \phi_j \phi_i d\hat{\boldsymbol{x}} = (\phi_j, \phi_i)_{\hat{D}}. \tag{2.7}$$

We also define a L^2 projection operator $\Pi_N : L^2(\hat{D}) \to P^N(\hat{D})$ such that

$$(\Pi_N f, v)_{\hat{D}} = (f, v)_{\hat{D}}, \qquad \forall v \in P^N(\hat{D}). \tag{2.8}$$

When integrals within the L^2 projection are computed with quadrature, the discrete quadrature-based L^2 projection of a function $f(x)$ can be expressed as following:

$$\boldsymbol{M}\boldsymbol{u} = \boldsymbol{V}_q^T \boldsymbol{W} \boldsymbol{f}, \qquad \boldsymbol{f}_i = f(\hat{\boldsymbol{x}}_i), \qquad 1 \le i \le N_q, \tag{2.9}$$

where $\boldsymbol{u}$ is the vector of coefficients of the quadrature-based L^2 projection of the function values of $\boldsymbol{f}$. We can define the quadrature-based L^2 projection matrix $\boldsymbol{P}_q$, by inverting the mass matrix:

$$\boldsymbol{P}_q = \boldsymbol{M}^{-1} \boldsymbol{V}_q^T \boldsymbol{W}. \tag{2.10}$$

The matrix $\boldsymbol{P}_q$ maps a function in terms of its evaluations at quadrature points to its coefficients of the L^2 projection in the basis $\phi_i(\hat{\boldsymbol{x}})$. Notice that since $\boldsymbol{M} = \boldsymbol{V}_q^T \boldsymbol{W} \boldsymbol{V}_q$, we have

$$\boldsymbol{P}_q \boldsymbol{V}_q = \boldsymbol{M}^{-1} \boldsymbol{V}_q^T \boldsymbol{W} \boldsymbol{V}_q = \boldsymbol{I}. \tag{2.11}$$

This implies that when we apply $\boldsymbol{P}_q$ to the evaluations of polynomial function at volume quadrature points, we recover the coefficients of the polynomial in the basis $\phi_i(\hat{\boldsymbol{x}})$.

2.2 Discrete formulation in 2D

With the tools defined in the previous sections, we can now derive discretization matrices which will be used in our discrete DG formulation.

In d-dimensions, define the following matrices

$$\hat{\boldsymbol{Q}}^i = \boldsymbol{M}\boldsymbol{D}^i, \qquad \boldsymbol{B}^i = \boldsymbol{W}_f \text{diag}\left(\hat{\boldsymbol{n}}_i\right), \qquad i = 1, ..., d. \tag{2.12}$$

With the above definitions, we have that

$$\hat{\boldsymbol{Q}}^i + (\hat{\boldsymbol{Q}}^i)^T = \boldsymbol{V}_f^T \boldsymbol{B}^i \boldsymbol{V}_f. \tag{2.13}$$

By combining the projection matrix $\boldsymbol{P}_q$ with the matrix $\hat{\boldsymbol{Q}}^i$, we can construct a nodal differentiation operator at quadrature points [1]:

$$\boldsymbol{Q}^i = \boldsymbol{P}_q^T \hat{\boldsymbol{Q}}^i \boldsymbol{P}_q. \tag{2.14}$$

We also define the the matrix $\boldsymbol{E}$, which extrapolates volume quadrature nodes to surface quadrature nodes, as

$$\boldsymbol{E} = \boldsymbol{V}_f \boldsymbol{P}_q. \tag{2.15}$$

Then we have the following generalized SBP property:

$$\boldsymbol{Q}^i + (\boldsymbol{Q}^i)^T = \boldsymbol{E}^T \boldsymbol{B}^i \boldsymbol{E}. \tag{2.16}$$

Similarly, for convenience, we define $\boldsymbol{V}_h$ as

$$\boldsymbol{V}_h = \begin{bmatrix} \boldsymbol{V}_q \\ \boldsymbol{V}_f \end{bmatrix}. \tag{2.17}$$

2.3 Hybridized SBP operators

Entropy stable formulations for nonlinear conservation laws can be constructed using the generalized SBP operators introduced in the previous section. However, these schemes introduce numerical flux terms which couple together all degrees of freedom on neighboring elements [13].

To avoid this, we introduce the hybridized operator $\boldsymbol{Q}_h^i$, which is given explicitly as

$$\boldsymbol{Q}_h^i = \frac{1}{2} \begin{bmatrix} \boldsymbol{Q}^i - (\boldsymbol{Q}^i)^T & \boldsymbol{E}^T \boldsymbol{B}^i \\ -\boldsymbol{B}^i \boldsymbol{E} & \boldsymbol{B}^i \end{bmatrix}. \tag{2.18}$$

This operator is designed to be applied to vectors of solution values at both volume and surface quadrature nodes and mimics the structure of boundary terms used in hybridized DG methods [14]. When used in a DG formulation, it allows one to construct entropy stable formulations using more standard DG numerical fluxes.

We have the following theorem:

Theorem 2.1. $\boldsymbol{Q}_h^i$ satisfies the $SBP - like$ property [1]:

$$\boldsymbol{Q}_h^i + \left(\boldsymbol{Q}_h^i\right)^T = \boldsymbol{B}_h^i, \qquad \boldsymbol{B}_h^i = \begin{bmatrix} \boldsymbol{0} & \\ & \boldsymbol{B}^i \end{bmatrix}, \tag{2.19}$$

and $\boldsymbol{Q}_h^i \boldsymbol{1} = 0$, where $\boldsymbol{1}$ is the vector of all ones.

2.4 SWE entropy and entropy variables

In this section, we introduce the entropy function and associated entropy variables for the SWE. Entropy stability [15] is the extension of L^2 (energy) stability for linear hyperbolic PDEs to nonlinear conservation laws. To provide a statement of entropy stability, we first need to define a convex entropy function $S(\boldsymbol{u})$. Solutions to nonlinear conservation laws are typically non-unique. To determine unique solutions, we require that solutions satisfy an entropy inequality.

The entropy function for the SWE is the total energy of the system [10, 9]:

$$S(\boldsymbol{u}) = \frac{1}{2}h(u^2 + v^2) + \frac{1}{2}gh^2 + ghb. \tag{2.20}$$

We also define the entropy variable $\boldsymbol{v} = S'(\boldsymbol{u})$. The convexity of the entropy function guarantees that the mapping between $\boldsymbol{u}$ and $\boldsymbol{v}$ is invertible. The entropy variables for the SWE are given explicitly as:

$$v_1 = \frac{\partial S}{\partial h} = g(h+b) - \frac{1}{2}u^2 - \frac{1}{2}v^2, \tag{2.21}$$

$$v_2 = \frac{\partial S}{\partial (hu)} = u, \tag{2.22}$$

$$v_3 = \frac{\partial S}{\partial (hv)} = v. \tag{2.23}$$

It can be shown as in [16] that there exists an entropy flux function $F(\boldsymbol{u})$ and entropy potential $\psi(\boldsymbol{u})$ such that

$$\boldsymbol{v}(\boldsymbol{u})^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} = \frac{\partial F(\boldsymbol{u})^T}{\partial \boldsymbol{u}}, \quad \psi(\boldsymbol{u}) = \boldsymbol{v}(\boldsymbol{u})^T \boldsymbol{f}(\boldsymbol{u}) - F(\boldsymbol{u}), \quad \psi'(\boldsymbol{u}) = \boldsymbol{f}(\boldsymbol{u}). \tag{2.24}$$

Assuming for simplicity that bathymetry is constant and that the domain Ω is periodic, an entropy equality can be derived for smooth solutions $\boldsymbol{u}$ by multiplying the SWE by $\boldsymbol{v}^T$ and integrating over the domain. Then, using the chain rule and definition of the entropy flux, we have the following statement of entropy conservation

$$\int_\Omega \frac{\partial S(\boldsymbol{u})}{\partial t} = \boldsymbol{0}. \tag{2.25}$$

For viscosity solutions, it can be shown that (2.25) becomes an entropy inequality.

Our goal is to reproduce this statement of entropy conservation discretely. The resulting entropy conservative formulation can then be used to construct entropy stable formulations by adding appropriate entropy dissipation terms.

2.5 Entropy conservation and flux differencing

In this section, we introduce numerical fluxes for SWE and describe an entropy conservation discrete formulation [17, 18, 2, 19]. To construct the entropy stable scheme in higher dimensions, we require entropy conservative fluxes as defined in [20]:

Definition 2.1. Let $\boldsymbol{f}_S^i(\boldsymbol{u}_L, \boldsymbol{u}_R)$ be a bivariate function which is symmetric and consistent with the flux function $\boldsymbol{f}^i(\boldsymbol{u})$, for $i = 1, ..., d$

$$\boldsymbol{f}_S^i(\boldsymbol{u}, \boldsymbol{u}) = \boldsymbol{f}^i(\boldsymbol{u}), \qquad \boldsymbol{f}_S^i(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{f}_S^i(\boldsymbol{v}, \boldsymbol{u}). \tag{2.26}$$

The numerical flux $\boldsymbol{f}_S^i(\boldsymbol{u}_L, \boldsymbol{u}_R)$ is entropy conservative if, for entropy variable $\boldsymbol{v}_L = \boldsymbol{v}(\boldsymbol{u}_L)$, $\boldsymbol{v}_R = \boldsymbol{v}(\boldsymbol{u}_R)$

$$(\boldsymbol{v}_L - \boldsymbol{v}_R)^T \boldsymbol{f}_S^i(\boldsymbol{u}_L, \boldsymbol{u}_R) = \psi_L^i - \psi_R^i, \tag{2.27}$$

$$\psi_L^i = \psi^i(\boldsymbol{v}(\boldsymbol{u}_L)), \quad \psi_R^i = \psi^i(\boldsymbol{v}(\boldsymbol{u}_R)). \tag{2.28}$$

The flux $\boldsymbol{f}_S^i$ can be used to construct entropy conservative and entropy stable finite volume methods. Entropy stable finite volume schemes were generalized in [21] to arbitrary high order.

This numerical flux is also used for the construction of discretely entropy stable DG schemes using an approach referred to as flux differencing [17, 22, 18, 2]. Flux differencing was first used to systematically recover entropy stable split formulations in [19], but is applicable to a broader range entropy stable formulations. Entropy stable DG schemes also couple elements together using using the same entropy conservative flux $\boldsymbol{f}_S^i(\boldsymbol{u}_L, \boldsymbol{u}_R)$ as an interface flux [22, 18, 2]. Entropy stable schemes are typically constructed by first constructing an entropy conservative scheme, then adding entropy dissipation through appropriate penalization terms at element interfaces. These additional penalization terms convert schemes which satisfy a global entropy equality into schemes which satisfy a global entropy inequality.

Using flux differencing from [1, 23], we can replace the term $\boldsymbol{f}^i(\boldsymbol{u}(x))$ with the term $2\boldsymbol{f}_S^i(\boldsymbol{u}(x), \boldsymbol{u}(x))$. Then we define a flux matrix $\boldsymbol{F}^i$ as the evaluations of $\boldsymbol{f}_S^i(\boldsymbol{u}(x), \boldsymbol{u}(y))$ at quadrature points:

$$(\boldsymbol{F}^i)_{jk} = \boldsymbol{f}_S^i(\boldsymbol{u}(\hat{x}_j), \boldsymbol{u}(\hat{x}_k)), \qquad 1 \leq j, k \leq N_q. \tag{2.29}$$

The term $2(\boldsymbol{Q} \circ \boldsymbol{F})\boldsymbol{1}$ approximates $\int \frac{\partial \boldsymbol{f}^i(\boldsymbol{u}(x))}{\partial x}$, and the key idea in entropy stable DG formulations is to replace $\boldsymbol{Q}\boldsymbol{f}(\boldsymbol{u})$ with $2(\boldsymbol{Q} \circ \boldsymbol{F})\boldsymbol{1}$, where $\boldsymbol{Q} \circ \boldsymbol{F}$ denotes the Hadamard product between $\boldsymbol{Q}$ and $\boldsymbol{F}$.

2.6 Entropy projection

We seek a degree N polynomial approximation of the conservative variables $\boldsymbol{u}(x, t)$ with coefficients $\boldsymbol{u}_h(t)$ such that

$$\boldsymbol{u}_N(\hat{\boldsymbol{x}}, t) = \sum_{i=1}^{N_p} (\boldsymbol{u}_h(t))_i \phi_i(\hat{\boldsymbol{x}}), \qquad (\boldsymbol{u}_h(t))_i \in \mathbb{R}^n. \tag{2.30}$$

Because $\boldsymbol{u}_h$ consists of vectors of coefficients for each scalar component of $\boldsymbol{u}_N(\hat{\boldsymbol{x}}, t)$, we should understand the discretization matrices as being applied to vectors like $\boldsymbol{u}_h$ in a Kronecker product sense. For example, $\boldsymbol{A}\boldsymbol{u}_h$ should be interpreted as applying $\boldsymbol{A}$ to each component of $\boldsymbol{u}_h$.

Reproducing conservation of entropy discretely faces an additional challenge. Since the SWE system is non-linear, entropy variables are not contained in the approximation space, and in general, $\boldsymbol{v}(\boldsymbol{u}) \notin P^N$ even if $\boldsymbol{u} \in P^N$. For shallow water, if one assumes $h, hu \in P^N$, then $v_2(h, hu) = u = \frac{hu}{h}$ is rational and non-polynomial.

Unfortunately, for DG methods, the test space contains only piecewise polynomial functions. To circumvent this issue, we introduce $\boldsymbol{v}_h$ as the L^2 projection of the entropy variables and $\tilde{\boldsymbol{u}}$ as the evaluations of the conservative variables in terms of the L^2 projected entropy variables

$$\boldsymbol{u}_q = \boldsymbol{V}_q \boldsymbol{u}_h, \qquad \boldsymbol{v}_q = \boldsymbol{v}(\boldsymbol{u}_q), \qquad \boldsymbol{v}_h = \boldsymbol{P}_q \boldsymbol{v}_q, \tag{2.31}$$

$$\tilde{\boldsymbol{v}} = \begin{bmatrix} \tilde{\boldsymbol{v}}_q \\ \tilde{\boldsymbol{v}}_f \end{bmatrix} = \begin{bmatrix} \boldsymbol{V}_q \\ \boldsymbol{V}_f \end{bmatrix} \boldsymbol{v}_h, \qquad \tilde{\boldsymbol{u}} = \begin{bmatrix} \tilde{\boldsymbol{u}}_q \\ \tilde{\boldsymbol{u}}_f \end{bmatrix} = \boldsymbol{u}(\tilde{\boldsymbol{v}}). \tag{2.32}$$

Here $\boldsymbol{u}_q$ and $\boldsymbol{v}_q$ denote the conservative variables and entropy variables evaluated at the volume quadrature points. The vector $\tilde{\boldsymbol{v}}$ denotes the evaluations of the L^2 projection of the entropy variables at both volume and surface quadrature points, while $\tilde{\boldsymbol{u}}$ denotes the evaluations of the conservative variables in terms of the projected entropy variables $\boldsymbol{u}(\Pi_N \boldsymbol{v})$.

2.7 Curved triangular meshes

We now extend the construction of hybridized SBP operators to curved triangular meshes, where each element can be represented as a curvilinear mapping Φ^k of the reference element $\hat{D}$ [1, 24]. Because we map the reference triangle to curved triangular elements, the geometric factors are not constant anymore, which can impact the stability of our DG formulation. We want to ensure the entropy stability on curved triangular meshes.

We introduce the following definitions associated with Jacobians of the mapping Φ^k:

- J^k denotes the determinant of the Jacobian of Φ^k.
- $\boldsymbol{J}_f^k$ denotes the vector of J_f^k at surface quadrature points.
- $\hat{\boldsymbol{J}}_f$ denotes the vector contains $\hat{J}_f$, the face Jocabian factor of the mapping from faces of the reference elements to the reference face. We assume $\hat{J}_f$ is pre-multiplied into the surface quadrature weights.

We introduce the "split" form of derivative to preserve entropy stability[19, 1, 24]

$$\frac{\partial u}{\partial x_i} = \frac{1}{2} \sum_j \left(\frac{\partial \hat{x}_j}{\partial x_i} \frac{\partial u}{\partial \hat{x}_j} + \frac{\partial}{\partial \hat{x}_j} \left(u \frac{\partial \hat{x}_j}{\partial x_i} \right) \right). \tag{2.33}$$

Let $\boldsymbol{G}_{ij}^k = \frac{\partial \hat{x}_j}{\partial x_i}$ be the vector of geometric factors evaluated at the quadrature points on element D^k. We define the physical SBP operator on D^k as:

$$\boldsymbol{Q}_h^{i,k} = \frac{1}{2} \sum_{j=1}^{2} \text{diag}\left(\boldsymbol{G}_{ij}^k \right) \hat{\boldsymbol{Q}}_h^{j,k} + \hat{\boldsymbol{Q}}_h^{j,k} \text{diag}\left(\boldsymbol{G}_{i,j}^k \right), \tag{2.34}$$

It can be shown that $\boldsymbol{Q}_h^{i,k}$ satisfies the following SBP property on D^k

$$\boldsymbol{Q}_h^{i,k} + \left(\boldsymbol{Q}_h^{i,k} \right)^T = \begin{bmatrix} \boldsymbol{0} & \\ & \boldsymbol{B}^{i,k} \end{bmatrix}, \qquad \boldsymbol{B}^{i,k} = \boldsymbol{W}_f \text{diag}\left(\boldsymbol{J}_f^k / \hat{\boldsymbol{J}}_f \circ \boldsymbol{n}_i^k \right),$$

where $\boldsymbol{J}_f^k/\hat{\boldsymbol{J}}_f$ denotes element-wise division between the vectors containing the face Jacobians and the reference face Jacobians, and $\boldsymbol{n}_i^k$ is a vector of the outward unit normals on D^k.

Finally, we define the mass matrix for the element D^k

$$\boldsymbol{M}_h^k = \boldsymbol{V}_q^T \boldsymbol{W}^k \boldsymbol{V}_q = \boldsymbol{V}_q^T \boldsymbol{W} J^k \boldsymbol{V}_q, \tag{2.35}$$

where the L^2 mass matrix is now weighted by a non-constant Jacobian J^k.

2.8 Entropy conservative flux for SWE

In this section, we finalize our discrete DG formulation for the SWE. We first present entropy conservative (EC) fluxes for the 2D SWE [15, 22, 7, 3]

$$\boldsymbol{f}_S^x(\boldsymbol{u}_L, \boldsymbol{u}_R) = \begin{bmatrix} \{\{hu\}\} \\ \{\{hu\}\}\{\{u\}\} + g\{\{h\}\}^2 - \frac{1}{2}g\{\{h^2\}\} \\ \{\{hu\}\}\{\{v\}\} \end{bmatrix}, \tag{2.36}$$

$$\boldsymbol{f}_S^y(\boldsymbol{u}_L, \boldsymbol{u}_R) = \begin{bmatrix} \{\{hv\}\} \\ \{\{hv\}\}\{\{u\}\} \\ \{\{hv\}\}\{\{v\}\} + g\{\{h\}\}^2 - \frac{1}{2}g\{\{h^2\}\} \end{bmatrix}. \tag{2.37}$$

A discrete entropy conservative formulation of the SWE is then given as follows on an element D^k:

$$\boldsymbol{M}_h^k \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + \sum_{i=x,y} \begin{bmatrix} \boldsymbol{V}_q \\ \boldsymbol{V}_f \end{bmatrix}^T \left(2\boldsymbol{Q}_h^{i,k} \circ \boldsymbol{F}^i\right)\boldsymbol{1} + \boldsymbol{V}_f^T \boldsymbol{B}^{i,k}\left(\boldsymbol{f}_S^i(\tilde{\boldsymbol{u}}^+, \tilde{\boldsymbol{u}}) - \boldsymbol{f}^i(\tilde{\boldsymbol{u}}_f)\right) = \boldsymbol{S}, \tag{2.38}$$

$$(\boldsymbol{F}^i)_{j,k} = \boldsymbol{f}_S^i(\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_j), \qquad 1 \le j,k \le N_q + N_q^f,$$

where $\boldsymbol{S}$ is the source term

$$\boldsymbol{S} = -g \begin{bmatrix} \boldsymbol{0} \\ \mathrm{diag}(\boldsymbol{h})\boldsymbol{Q}^x \boldsymbol{b} \\ \mathrm{diag}(\boldsymbol{h})\boldsymbol{Q}^y \boldsymbol{b} \end{bmatrix}.$$

Recall that $\tilde{\boldsymbol{u}}$ are the "entropy-projected" conservative variables introduced in the previous section. This formulation was shown to be entropy conservative in [25, 26].

We note that, in our implementation, we precompute the inverses of the element mass matrices, i.e. $(\boldsymbol{M}_h^k)^{-1}$ for all k, and store them on the GPU. This can be avoid using weight-adjusted mass matrix inverses [25], which we will investigate in future work.

Now we present a proof of entropy conservation for our discrete DG formulation of the SWE. A proof of entropy conservation for general nonlinear conservation laws was given in [1], but did not account for bathymetric source terms. This proof extends this theory while accounting for the presence of varying bathymetry.

Theorem 2.2. Let $\boldsymbol{f}_s$ be an entropy conservative flux from Definition 2.1. Then assuming continuity in time, the semi-discrete formulation (2.38) is entropy conservative and well-balanced for $b \in P^N$.

Proof. First, we divide the entropy variable $\boldsymbol{v}$ into two parts

$$\boldsymbol{v} = \boldsymbol{v}_0 + \boldsymbol{v}_b, \qquad \boldsymbol{v}_0 = \begin{bmatrix} gh - \frac{1}{2}(u^2+v^2) \\ u \\ v \end{bmatrix}, \qquad \boldsymbol{v}_b = \begin{bmatrix} gb \\ 0 \\ 0 \end{bmatrix}, \tag{2.39}$$

where $\boldsymbol{v}_0$ are terms in the entropy variables corresponding to zero bathymetry and $\boldsymbol{v}_b$ are terms corresponding to the contribution from variable bathymetry.

Without loss of generality, we assume the solution and b are constant along the y direction such that only derivatives in the x-direction remain. The general proof can be treated by repeating this procedure in each coordinate direction.

Without y-derivatives, the formulation (2.38) reduces to 1D formulation

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + 2\left(\boldsymbol{Q}_h^{x,k} \circ \boldsymbol{F}_S^x\right)\mathbf{1} + \boldsymbol{B}^{x,k}\left(\boldsymbol{f}_S^x\left(\tilde{\boldsymbol{u}}_f^+, \tilde{\boldsymbol{u}}_f\right) - \boldsymbol{f}^x(\tilde{\boldsymbol{u}}_f)\right) = \mathbf{0}.$$

From here, we drop superscripts, subscripts, tildes, and $(\cdot)_h^{x,k}$ on all solution variables to simplify notation.

Notice that the term $2\left(\boldsymbol{Q} \circ \boldsymbol{F}_S\right)\mathbf{1}$ is independent of the bathymetry b. For $b = 0$, multiplying the discrete formulation (2.38) by $\boldsymbol{v}_0^T$ yields that

$$\boldsymbol{v}_0^T\left(\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + 2\left(\boldsymbol{Q} \circ \boldsymbol{F}_S\right)\mathbf{1} + \boldsymbol{B}\left(\boldsymbol{f}_S - \boldsymbol{f}(\boldsymbol{u})\right)\right) = \boldsymbol{v}_0^T\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} = 0. \tag{2.40}$$

The proof is the same as the one given in [1] (Theorem 2).

It remains to show that entropy conservation still holds if b is not constant. Multiplying (2.38) by $\boldsymbol{v}^T$ then yields

$$\begin{aligned}&\boldsymbol{v}^T\left(\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + 2\left(\boldsymbol{Q} \circ \boldsymbol{F}_S\right)\mathbf{1} + \boldsymbol{B}\left(\boldsymbol{f}_S - \boldsymbol{f}(\boldsymbol{u})\right)\right)\\ =&\boldsymbol{v}_0^T\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + \boldsymbol{v}_b^T\left(\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + 2\left(\boldsymbol{Q} \circ \boldsymbol{F}_S\right)\mathbf{1} + \boldsymbol{B}\left(\boldsymbol{f}_S - \boldsymbol{f}(\boldsymbol{u})\right)\right) = \boldsymbol{v}^T\boldsymbol{s}.\end{aligned} \tag{2.41}$$

We wish to show that this implies

$$\mathbf{1}^T\boldsymbol{W}\frac{\mathrm{d}S(\boldsymbol{u})}{\mathrm{dt}} = 0.$$

The time derivative terms $\boldsymbol{v}_b^T\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}}$ and $\boldsymbol{v}_0^T\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}}$ from (2.41) combine to the form the term $\boldsymbol{v}^T\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} = \mathbf{1}^T\boldsymbol{W}\frac{\partial S}{\partial t}$ for varying bathymetry (the proof can be found in [1], and utilizes properties of the L^2 projection). What remains to show is then that

$$\boldsymbol{v}_b^T\left(2\left(\boldsymbol{Q} \circ \boldsymbol{F}_S\right)\mathbf{1} + \boldsymbol{B}\left(\boldsymbol{f}_S - \boldsymbol{f}(\boldsymbol{u})\right)\right) - \boldsymbol{v}^T\boldsymbol{s} = 0.$$

Since the second and the third components of $\boldsymbol{v}_b$ are 0, the first expression corresponds to testing the mass conservation equation in the DG formulation with gb. Substituting the flux from (2.36), this yields

$$\begin{aligned}&\boldsymbol{v}_b^T 2\left(\boldsymbol{Q} \circ \boldsymbol{F}_S\right)\mathbf{1} + \boldsymbol{v}_b^T\boldsymbol{B}\left(\boldsymbol{f}_S - \boldsymbol{f}(\boldsymbol{u})\right) - (\boldsymbol{v}_b^T + \boldsymbol{v}_0^T)\boldsymbol{s}\\ =&g\boldsymbol{b}^T\boldsymbol{Q}(\boldsymbol{h}\boldsymbol{u}) + g(\boldsymbol{h}\boldsymbol{u})^T\boldsymbol{Q}\boldsymbol{b} + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}[[\boldsymbol{h}\boldsymbol{u}]].\end{aligned} \tag{2.42}$$

where we have expanded out the interface term $\boldsymbol{v}_b^T\boldsymbol{B}\left(\boldsymbol{f}_S - \boldsymbol{f}\right)$ to yield an expression involving the jump $[[.]]$ of a quantity across the element boundary.

Using the SBP property yields

$$\begin{aligned} &g\boldsymbol{b}^T\boldsymbol{Q}(\boldsymbol{hu}) + g(\boldsymbol{hu})^T\boldsymbol{Q}\boldsymbol{b} + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}[[\boldsymbol{hu}]] \\ =&g\boldsymbol{b}^T\boldsymbol{Q}(\boldsymbol{hu}) + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}[[\boldsymbol{hu}]] + g\boldsymbol{b}^T(\boldsymbol{Q})^T(\boldsymbol{hu}) \\ =&g\boldsymbol{b}^T\boldsymbol{B}(\boldsymbol{hu}) + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}[[\boldsymbol{hu}]]. \end{aligned} \tag{2.43}$$

Consider two neighboring elements D^k and D^{k+}. Then, on each element, the boundary terms are

$$\begin{aligned} D^k: \quad & g\boldsymbol{b}^T\boldsymbol{B}^x(\boldsymbol{hu}) + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}^x((\boldsymbol{hu})^+ - (\boldsymbol{hu})), \\ D^{k+}: \quad & g\boldsymbol{b}^T\boldsymbol{B}^{x+}(\boldsymbol{hu})^+ + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}^{x+}((\boldsymbol{hu}) - (\boldsymbol{hu})^+) \\ & = -g\boldsymbol{b}^T\boldsymbol{B}^x(\boldsymbol{hu})^+ - \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}^x((\boldsymbol{hu})^+ - (\boldsymbol{hu})). \end{aligned} \tag{2.44}$$

Since outward normals are equal and opposite across an interface, $B^{x,+} = -B^x$ and the boundary terms (2.43) cancel out when summed up over two neighboring elements. This implies that

$$\sum_{\partial D^k} g\boldsymbol{b}^T\boldsymbol{B}^x(\boldsymbol{hu}) + \frac{1}{2}g\boldsymbol{b}^T\boldsymbol{B}^x[[\boldsymbol{hu}]] = 0. \tag{2.45}$$

Combining the results from the $b = 0$ part, we obtain the entropy conservative property.

To show this formulation is well-balanced, we consider the steady state solution

$$h(x,t) = c - b(x), \qquad u(x,t) = 0, \tag{2.46}$$

where c is a constant total water height and b is continuous. We immediately obtain well-balancedness for the first equation involving h because $hu = 0$. To see $u(x,t) = 0$ is preserved, we substitute the nonzero terms of flux from (2.37) into formulation (2.38)

$$2g\left(\boldsymbol{Q}\circ\left(\{\{\boldsymbol{h}\}\}^2 - \frac{\{\{\boldsymbol{h}^2\}\}}{2}\right)\right)\mathbf{1} + g\boldsymbol{B}\left(\{\{\boldsymbol{h}\}\}^2 - \frac{\{\{\boldsymbol{h}^2\}\}}{2} - \frac{\boldsymbol{h}^2}{2}\right) \tag{2.47}$$

$$=2g\left(\boldsymbol{Q}\circ\frac{\boldsymbol{h}_i\boldsymbol{h}_j}{2}\right)\mathbf{1} + g\boldsymbol{B}\mathbf{0} \tag{2.48}$$

$$=g\left(\boldsymbol{Q}\circ(\boldsymbol{h}_i\boldsymbol{h}_j)\right)\mathbf{1} \tag{2.49}$$

$$=g\sum_j \boldsymbol{Q}_{ij}\boldsymbol{h}_i\boldsymbol{h}_j, \qquad i = 1, ..., N_q. \tag{2.50}$$

Since $\boldsymbol{Q}$ differentiate polynomial exactly up to degree N, $\boldsymbol{Q}\mathbf{1} = \mathbf{0}$ and $\boldsymbol{Q}\boldsymbol{c} = \mathbf{0}$ for any constant vector $\boldsymbol{c}$. We also have the source term

$$-g\cdot\operatorname{diag}(\boldsymbol{h})\boldsymbol{Q}\boldsymbol{b} \tag{2.51}$$

$$=-g\cdot\operatorname{diag}(\boldsymbol{h})\boldsymbol{Q}(\boldsymbol{c}-\boldsymbol{h}) \tag{2.52}$$

$$=g\cdot\operatorname{diag}(\boldsymbol{h})\boldsymbol{Q}\boldsymbol{h} \tag{2.53}$$

$$=g\boldsymbol{h}_i\sum_j \boldsymbol{Q}_{ij}\boldsymbol{h}_j \tag{2.54}$$

$$=g\sum_j \boldsymbol{Q}_{ij}\boldsymbol{h}_i\boldsymbol{h}_j, \qquad i = 1, ..., N_q. \tag{2.55}$$

Therefore the volume and surface contributions cancel out with the source term to achieve well-balancedness. □

Thus, our DG formulation is entropy conservative. The fact that the entropy function for the SWE is the total energy of the system also provides a connection between the mathematical stability of our numerical scheme and physical principles.

To construct an entropy stable scheme, we add entropy dissipative interface penalization terms to the entropy conservative formulation. In this work, we utilize local Lax-Friedrichs penalization [9, 2] applied to the entropy-projected conservative variables. We note that this choice of penalization also preserves the well-balanced property: for a lake-at-rest condition with zero velocity, the entropy projected conservative variables are the same as the original conservative variables. Lax-Friedrichs penalization adds a scaling of the jumps of the conservative variables, and since the jumps of the conservative variables vanish for continuous water height and bathymetry, the entropy stable scheme reduces to the well-balanced entropy conservative scheme for the lake-at-rest condition.

Finally, while we have restricted ourselves to continuous bathymetry for this paper, it is possible to construct well-balanced and entropy stable schemes for discontinuous bathymetry by adding additional interface terms [10, 9].

2.9 Imposing reflective (wall) boundary conditions

Proofs of entropy stability have involved periodic boundary conditions. However, one can also show entropy conservation or entropy stability under reflective wall boundary conditions. To impose reflective "wall" boundary conditions in an entropy conservative or entropy stable fashion, we follow procedures outlined in [9, 2]. For surfaces that correspond to the wall, we set $u_{\boldsymbol{n}}^+ = -u_{\boldsymbol{n}}$ and $h^+ = h$, where $(.)^+$ denotes the exterior value of the solution used in a numerical flux (e.g., the value of the solution on a neighboring element for an interior interface), and the subscript $(.)_{\boldsymbol{n}}$ denotes the normal component of the vector with respect to the wall.

3 Entropy stable formulations on curved triangular meshes

In this section, we present a second entropy stable DG method for the SWE based on summation-by-parts (SBP) operators [2, 27]. SBP operators are nodal finite-difference matrices which mimic integration by parts. This property is crucial in constructing entropy stable discretizations of non-linear conservation laws.

The SBP operators used in entropy stable shceme usually include a diagonal mass matrix and differentiation matrices, which are designed to approximate spatial derivatives up to a specified order of accuracy. They are constructed algebraically given a set of quadrature nodes with positive weights and some specific level of accuracy [2, 28]. The SBP property is typically sufficient to prove stability for linear PDEs under periodic boundary conditions and appropriate coupling terms [29].

Entropy stable numerical schemes can be constructed using either hybridized SBP operators or traditional SBP operators. Hybridized SBP operators provide a numerical connection to polynomial DG methods and allow for flexibility in the choice of quadrature rules. In contrast, traditional SBP operators do not correspond to any polynomial DG discretizations. This is because SBP operators identify nodal values as individual degrees of freedom, and the number of points is larger than the dimension of underlying polynomial space. As a result, traditional SBP operators cannot be derived from standard DG variational formulations.

SBP operators minic the structure of mass-lumped under-integrated DG discretizations, which can result in higher aliasing errors. However, entropy stable discretizations using traditional SBP operators are more efficient, as they typically involve fewer operations and can skip the entropy projection step discussed in Section 2. We will compare the performances of entropy stable methods under both hybridized and traditional SBP operators in Section 4.

3.1 SBP quadrature rules

We require an SBP-quadrature rule to have the following properties:

- The surface quadrature points are identically distributed on each face (which enables straightforward coupling between elements.
- The quadrature weights are positive.
- The volume quadrature rule is exact for polynomials up to degree $2N-1$.
- The volume quadrature rule also contains boundary points which form a separate surface quadrature rule.
- The surface quadrature rule is exact for degree $2N$ polynomials on each face.

In our numerical experiments, we consider two sets of 2D SBP quadrature points. The first uses 1D Gauss-Legendre quadrature on the edges while the second uses 1D Gauss-Lobatto quadrature on edges. They are shown in the Figure 1 and 2 respectively. The Gauss-Legendre SBP quadrature rules were introduced in [2], and the Gauss-Lobatto SBP quadrature rules were computed specifically for this paper.

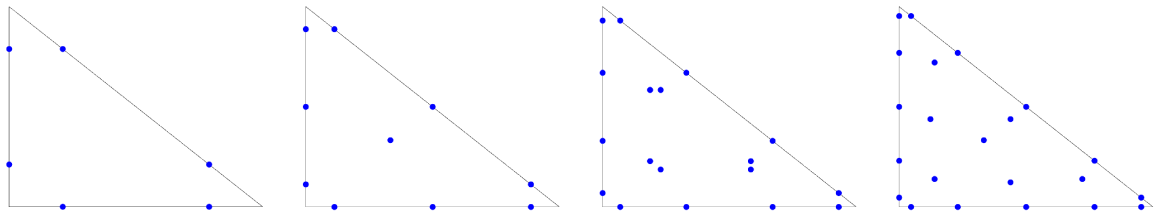

Figure 1: Gauss-Legendre quadrature for $N = 1, 2, 3, 4$

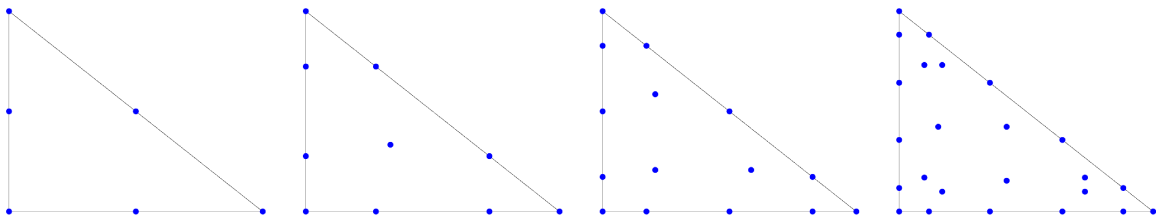

Figure 2: Gauss-Lobatto quadrature for $N = 1, 2, 3, 4$

3.2 Construction of traditional SBP operators

We now describe a process for constructing traditional SBP operators. We assume we are given a degree $2N-1$ quadrature rule for the polynomial space $P^N(\hat{D})$, with N_q nodes, $\{\boldsymbol{x_i}\}_{i=1}^{N_q}$, and positive weights $\{w_i\}_{i=1}^{N_q}$. The nodal values of the function $u(x)$ on the quadrature points is denoted by:

$$\boldsymbol{u} = [u(\boldsymbol{x}_1), ..., u(\boldsymbol{x}_{N_q})]^T. \tag{3.1}$$

The SBP mass matrix is defined as the a diagonal matrix with quadrature weights on the diagonal.

$$\boldsymbol{M}_{\text{SBP}} = \text{diag}\left([w_1, ..., w_{N_q}]\right). \tag{3.2}$$

We define $\boldsymbol{D}^x$ and $\boldsymbol{D}^y$ to be the nodal differentiation matrices associated with the x and y derivatives. We also define $\boldsymbol{B}^x$ and $\boldsymbol{B}^y$ to be diagonal surface matrices, whose entries are surface quadrature weights scaled by the x and y components of the outward normal vector.

We have the following definition [2].

Definition 3.1. Consider the diagonal mass matrix consisting of quadrature weights

$$\boldsymbol{M}_{\text{SBP}} = \text{diag}\left([w_1, ..., w_{N_q}]\right). \tag{3.3}$$

A 2D operator $\boldsymbol{Q}^i_{\text{SBP}}$ is said to have SBP property if for $i = x, y$, the following properties holds.

- Let $\boldsymbol{D}^i = \boldsymbol{M}^{-1}_{\text{SBP}}\boldsymbol{Q}^i_{\text{SBP}}$. Then $(\boldsymbol{D}^i\boldsymbol{u})_j = \frac{\partial \boldsymbol{u}}{\partial x_i}\Big|_{x=x_j}$ for any $\boldsymbol{u} \in P^N(\hat{D})$.
- $\boldsymbol{Q}^i_{\text{SBP}} + (\boldsymbol{Q}^i_{\text{SBP}})^T = \boldsymbol{B}^i$.

While it is not immediately apparent, one can derive traditional SBP operators from hybridized SBP operators. We introduce the "selection" matrix $\boldsymbol{I}_f$ of size $N_f \times N_q$. $\boldsymbol{I}_f$ is a generalized permutation matrix which extracts the surface nodes from the list of all quadrature nodes. For example, suppose that the ith node in the list of all quadrature nodes is on the surface of the reference domain. Suppose that this node corrsponds to the jth node in the list of surface quadrature nodes. Then, the (i, j) entry of $\boldsymbol{I}_f$ is one. To summarize, $\boldsymbol{I}_f$ selects out the surface quadrature nodes from the set of SBP quadrature nodes and reorders them for computation.

Theorem 3.1. Define $\boldsymbol{Q}^i_{\text{SBP}}$ as:

$$\boldsymbol{Q}^i_{\text{SBP}} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}^T \boldsymbol{Q}^i_h \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}, \tag{3.4}$$

where $\boldsymbol{Q}^i_h$ is defined in Section 2.3. Then, $\boldsymbol{Q}^i_{\text{SBP}}$ a multi-dimensional SBP operator, which is explicitly defined as.

Proof. In order to show that the matrix $\boldsymbol{Q}^i_{\text{SBP}}$ is consistent with definition 4.1 in [2], we first need to show that the difference matrix $\boldsymbol{M}^{-1}_{\text{SBP}}\boldsymbol{Q}^i_{\text{SBP}}$ is exact for any polynomial $u(\boldsymbol{x}) \in P^N(\hat{D})$. Let

$\boldsymbol{u}$ denote the modal coefficient of the polynomial and $\boldsymbol{u}_q = \boldsymbol{V}_q\boldsymbol{u}$ denote its nodal value at the quadrature points. We then have

$$\boldsymbol{Q}^i_{\text{SBP}} = \begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}^T \boldsymbol{Q}^i_h \begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix} \tag{3.5}$$

$$= \begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}^T \frac{1}{2}\begin{bmatrix}\boldsymbol{Q}^i - (\boldsymbol{Q}^i)^T & \boldsymbol{E}^T\boldsymbol{B}^i\\ -\boldsymbol{B}^i\boldsymbol{E} & \boldsymbol{B}^i\end{bmatrix}\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}. \tag{3.6}$$

We use the fact that $\boldsymbol{Q}^i + (\boldsymbol{Q}^i)^T = \boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E}$ from Section 2 to substitute $(\boldsymbol{Q}^i)^T = \boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E} - \boldsymbol{Q}^i$ into $\boldsymbol{Q}^i - (\boldsymbol{Q}^i)^T$. We have

$$\boldsymbol{Q}^i - (\boldsymbol{Q}^i)^T = \boldsymbol{Q}^i - (\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E} - \boldsymbol{Q}^i) \tag{3.7}$$

$$= 2\boldsymbol{Q}^i - \boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E}. \tag{3.8}$$

Multiply the $\frac{1}{2}$ into the matrix, we have:

$$\boldsymbol{Q}^i_{\text{SBP}} = \begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}^T \begin{bmatrix}\boldsymbol{Q}^i - \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E} & \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\\ -\frac{1}{2}\boldsymbol{B}^i\boldsymbol{E} & \frac{1}{2}\boldsymbol{B}^i\end{bmatrix}\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}. \tag{3.9}$$

So we can write the difference matrix as:

$$\boldsymbol{D}^i_{\text{SBP}} = \boldsymbol{M}^{-1}_{\text{SBP}}\boldsymbol{Q}^i_{\text{SBP}} \tag{3.10}$$

$$= \boldsymbol{M}^{-1}_{\text{SBP}}\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}^T \begin{bmatrix}\boldsymbol{Q}^i - \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E} & \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\\ -\frac{1}{2}\boldsymbol{B}^i\boldsymbol{E} & \frac{1}{2}\boldsymbol{B}^i\end{bmatrix}\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}. \tag{3.11}$$

If we apply $\boldsymbol{D}^i_{\text{SBP}}$ at $\boldsymbol{u}_q$, we have

$$\boldsymbol{D}^i_{\text{SBP}}\boldsymbol{u}_q = \boldsymbol{M}^{-1}_{\text{SBP}}\left[\boldsymbol{Q}^i - \tfrac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E} - \tfrac{1}{2}\boldsymbol{I}_f^T\boldsymbol{B}^i\boldsymbol{E},\quad \tfrac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i + \tfrac{1}{2}\boldsymbol{I}_f^T\boldsymbol{B}^i\right]\begin{bmatrix}\boldsymbol{u}_q\\ \boldsymbol{u}_f\end{bmatrix}, \tag{3.12}$$

since $\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{I}_f\end{bmatrix}$ maps the vector $\boldsymbol{u}_q$ to the vector of values at both volume and surface quadrature points. Notice that we have $\boldsymbol{E} = \boldsymbol{V}_f\boldsymbol{P}_q$ and $\boldsymbol{P}_q\boldsymbol{V}_q = \boldsymbol{I}$ from 2. Then, $\boldsymbol{E}\boldsymbol{u}_q = \boldsymbol{V}_f\boldsymbol{P}_q\boldsymbol{V}_q\boldsymbol{u} = \boldsymbol{V}_f\boldsymbol{u} = \boldsymbol{u}_f$

$$\begin{aligned}\boldsymbol{D}^i_{\text{SBP}}\boldsymbol{u}_q &= \boldsymbol{M}^{-1}_{\text{SBP}}\left(\boldsymbol{Q}^i\boldsymbol{u}_q - \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{E}\boldsymbol{u}_q - \frac{1}{2}\boldsymbol{I}_f^T\boldsymbol{B}^i\boldsymbol{E}\boldsymbol{u}_q + \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{u}_f + \frac{1}{2}\boldsymbol{I}_f^T\boldsymbol{B}^i\boldsymbol{u}_f\right)\\ &= \boldsymbol{M}^{-1}_{\text{SBP}}\left(\boldsymbol{Q}^i\boldsymbol{u}_q - \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{u}_f - \frac{1}{2}\boldsymbol{I}_f^T\boldsymbol{B}^i\boldsymbol{u}_f + \frac{1}{2}\boldsymbol{E}^T\boldsymbol{B}^i\boldsymbol{u}_f + \frac{1}{2}\boldsymbol{I}_f^T\boldsymbol{B}^i\boldsymbol{u}_f\right).\end{aligned} \tag{3.13}$$

We simplify the above formulation and substitute in $\boldsymbol{Q}^i = \boldsymbol{P}_q^T\hat{\boldsymbol{Q}}^i\boldsymbol{P}_q$, $\boldsymbol{P}_q = \boldsymbol{M}^{-1}\boldsymbol{V}_q^T\boldsymbol{W}$ and $\boldsymbol{M}^{-1}_{\text{SBP}} = \boldsymbol{W}$. Since $\boldsymbol{M}$ is symmetric, $\boldsymbol{M}^{-1} = (\boldsymbol{M}^{-1})^T$. Then using $\boldsymbol{P}_q\boldsymbol{V}_q = \boldsymbol{I}$,

$$\boldsymbol{D}^i_{\text{SBP}}\boldsymbol{u}_q = \boldsymbol{M}^{-1}_{\text{SBP}}\boldsymbol{Q}^i\boldsymbol{V}_q\boldsymbol{u} \tag{3.14}$$

$$= \boldsymbol{W}^{-1}\boldsymbol{P}_q^T\hat{\boldsymbol{Q}}^i\boldsymbol{P}_q\boldsymbol{V}_q\boldsymbol{u} \tag{3.15}$$

$$= \boldsymbol{W}^{-1}\boldsymbol{W}\boldsymbol{V}_q(\boldsymbol{M}^{-1})^T\boldsymbol{M}\boldsymbol{D}^i(\boldsymbol{P}_q\boldsymbol{V}_q)\boldsymbol{u} \tag{3.16}$$

$$= \boldsymbol{V}_q\boldsymbol{D}^i\boldsymbol{u}. \tag{3.17}$$

Recall that the differentiation matrix $\boldsymbol{D}^i$ is exact for polynomials $\boldsymbol{u} \in P^N(\hat{D})$. Since $\boldsymbol{V}_q$ is a degree N interpolation matrix of the derivative at quadrature points, we conclude that $\boldsymbol{D}^i_{\text{SBP}}$ exactly differentiate $\boldsymbol{u}_q$, where $\boldsymbol{u}_q$ is the vector of values of a polynomial $\boldsymbol{u} \in P^N(\hat{D})$ at quadrature points.

We now show the summation-by-parts property. First we use the property that

$$\boldsymbol{Q}^i_h + (\boldsymbol{Q}^i_h)^T = \begin{bmatrix} \boldsymbol{0} & \\ & \boldsymbol{B}^i \end{bmatrix}, \tag{3.18}$$

to rewrite $\boldsymbol{Q}^i_{\text{SBP}}$ as

$$\begin{aligned} \boldsymbol{Q}^i_{\text{SBP}} &= \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}^T \left(\begin{bmatrix} \boldsymbol{0} & \\ & \boldsymbol{B}^i \end{bmatrix} - (\boldsymbol{Q}^i_h)^T \right) \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}, && (3.19) \\ &= \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}^T \begin{bmatrix} \boldsymbol{0} & \\ & \boldsymbol{B}^i \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix} - \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}^T (\boldsymbol{Q}^i_h)^T \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I}_f \end{bmatrix}, && (3.20) \\ &= \boldsymbol{I}_f^T \boldsymbol{B}^i \boldsymbol{I}_f - (\boldsymbol{Q}^i_{\text{SBP}})^T. && (3.21) \end{aligned}$$

We conclude that $\boldsymbol{Q}^i_{\text{SBP}} + (\boldsymbol{Q}^i_{\text{SBP}})^T = \boldsymbol{I}_f^T \boldsymbol{B}^i \boldsymbol{I}_f$. Since $\boldsymbol{I}_f$ is a matrix which selects the surface quadrature points, $\boldsymbol{I}_f^T \boldsymbol{B}^i \boldsymbol{I}_f$ is still diagonal with entries of $\boldsymbol{B}^i$ permuted. The permutation order depends on the order of the quadrature points listed in the implementation, but does not affect the summation-by-parts property. □

Given this equivalence, we can now construct an entropy conservative DG-SBP form:

$$\boldsymbol{M} \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + \sum_{i=x,y} \left(2\boldsymbol{Q}^i_{\text{SBP}} \circ \boldsymbol{F}^i \right) \boldsymbol{1} + \boldsymbol{I}_f^T \boldsymbol{B}^i \left(\boldsymbol{f}^i_S - \boldsymbol{f}^i \right) = \boldsymbol{S}. \tag{3.22}$$

The stability analysis of the SBP formulation follows from the results from [2]. The extension to curved elements can be found in [23].

4 Numerical results

In this section, we present some two dimensional numerical experiments and results to demonstrate the accuracy and stability of the entropy stable DG scheme. All experiments are run using an entropy stable scheme, which we construct by adding local Lax-Friedrichs penalization [9] to a baseline entropy conservative DG formulation.

The first experiment is a "lake-at-rest" condition to test the well-balancedness of our scheme. The second experiment is a translating vortex. This problem has an explicit analytic solution, which we use to investigate the convergence rate of our algorithm. The third experiment is a dam break simulation from [9]. The last experiment is a converging channel simulation [30].

All numerical experiments utilize the fourth order five-stage low-storage Runge-–Kutta method [31]. Following the derivation of stable timestep restrictions in [32], we define the timestep Δt to be

$$\Delta t = CFL \times \frac{h}{C_N}, \qquad C_N = \frac{(N+1)(N+2)}{2}, \tag{4.1}$$

where C_N is the degree dependent constant in the trace inequality space [33], and CFL is a user-defined constant. We use CFL = 0.125 for all experiments.

We test "lake-at-rest" and translating vortex problem on both affine and curved meshes. For the curved mesh, we construct the warping of the regular triangular mesh in the following way:

$$x = x + C_{curve} L_x \cos\left(\pi\left(x - x_0\right)/L_x\right) \cos\left(1.5\pi\left(y - y_0\right)/L_y\right),$$
$$y = y + C_{curve} L_y \sin\left(2\pi\left(x - x0\right)/L_x\right) \cos\left(pi\left(y - y0\right)/L_y\right),$$

where $(x_0, y_0) = (0, 0)$ is location of the center of the simulation, $(0, 0)$ in our case. L_x and L_y are the length of the domain in x and y direction respectively. C_{curve} is a user specified curving coefficient, which we use 0.1 in all the experiments in "lake-at-rest" and translating vortex problem. For the dam break and converging channel experiment, we construct a curved mesh which is boundary-fitted to the curved dam (a curved interior boundary) or the curved channel. More details are provided in the following sections.

4.1 Lake at rest

We first consider the "lake-at-rest" condition [7, 8, 34] on $[-1, 1] \times [-1, 1]$ and we set the boundary to be periodic. We set:

$$H = h + b = 2, \tag{4.2}$$
$$b(x, y) = 0.1 \sin(2\pi x) \cos(2\pi x) + 0.5. \tag{4.3}$$

For a degree N approximation, we first interpolate b using a continuous C^0 degree N polynomial, then we set $h = 2 - b$. In the Table 1 and Table 2, we observe the error for this setting is of the magnitude of round-off errors at each quadrature point for both affine and curved triangular meshes. All of the "lake-at-rest" experiments are run to a final time of $T = 1/2$.

N	$K = 256$	$K = 1024$	$K = 4096$	$K = 16384$
1	1.03E-13	1.54E-13	3.17E-13	7.27E-13
2	9.62E-13	4.61E-12	1.90E-11	7.88E-11
3	2.96E-12	7.71E-12	4.43E-11	1.58E-10
4	1.62E-11	6.41E-11	2.53E-10	1.03E-09

Table 1: L^2 error for the lake at rest problem on affine triangular meshes. N denotes the polynomial degree and K denotes the number of elements.

N	$K = 256$	$K = 1024$	$K = 4096$	$K = 16384$
1	1.06E-13	1.64E-13	2.88E-13	5.51E-13
2	1.07E-12	4.06E-12	1.67E-11	6.78E-11
3	2.99E-12	1.16E-11	4.55E-11	1.84E-10
4	1.17E-11	4.66E-11	1.90E-10	7.57E-10

Table 2: L^2 error for the lake at rest problem on curved triangular mesh. N denotes the polynomial degree and K denotes the number of elements.

We evaluate the L^2 error using a more accurate triangular quadrature rule exact for degree $2N+2$ polynomials. We notice that the error for this problem increases as we use higher order polynomial bases or finer meshes, and we attribute this to numerical round off.

4.2 Translating vortex

We now consider the vortex translation test. We set the domain to be $[-10,10]\times[-5,5]$ and the exact solution for the vortex at any time t is given by [35, 36]:

$$h = h_\infty - \frac{\beta^2}{32\pi^2} e^{-2(r^2-1)}, \tag{4.4}$$

$$u = u_\infty - \frac{\beta}{2\pi} e^{-2(r^2-1)} y_t, \tag{4.5}$$

$$v = v_\infty + \frac{\beta}{2\pi} e^{-2(r^2-1)} x_t, \tag{4.6}$$

$$b = 0, \tag{4.7}$$

where

$$x_t = x - x_c - u_\infty t, \tag{4.8}$$

$$y_t = y - y_c - v_\infty t, \tag{4.9}$$

$$r^2 = x^2 + y^2. \tag{4.10}$$

In this example, we set

$$h_\infty = 1, \quad \beta = 5, \quad g = 2 \quad \text{and} \quad (\mathrm{u}_\infty, \mathrm{v}_\infty) = (1,0). \tag{4.11}$$

Initially the vortex is located at $(x_c, y_c) = (0,0)$. In this setup, the vortex propagates to the right along the x-axis. The domain and problem setup are chosen such that periodic boundary conditions can be used without affecting accuracy.

We use both affine and curved meshes for this experiment. We also compare the L^2 error the SBP formulation using operators based on both Gauss-Legendre and Gauss-Lobatto quadrature nodes as we introduced in section 3. We calculate the L^2 error using the same method as in the "lake-at-rest" problem. The convergence results are presented in the Figure 5 and Figure 6:

Recall that the SBP-DG discretization does not correspond to a polynomial approximation space. Thus, to calculate the L^2 error for the SBP-DG discretization, we first project the final numerical solution to polynomials of degree N. We then use this projection to evaluate the L^2 error using a quadrature rule which is exact for at least degree $2N+2$ polynomials. The error for the DG method with hybridized SBP operators is computed using the same quadrature.

In Figure 5, we analyze the accuracy of the hybridized SBP scheme on both curved and affine meshes. We observe the same rate of convergence for both curved and affine meshes, but note that some accuracy is lost on the curved meshes. In Figure 6 and 7, we show the comparisons between two SBP methods with the hybridized SBP method. Figure 6 presents the Gauss-Legendre SBP nodes and Figure 7 presents the Gauss-Lobatto nodes compared against the hybridized SBP method. We observe that the use of traditional SBP operators, while more efficient, lose one order of accuracy compared to the use of hybridized SBP operators. The Gauss-Lobatto SBP operator seems to be slightly more accurate than Gauss-Lobatto SBP operator, but both converge at about the same rate.

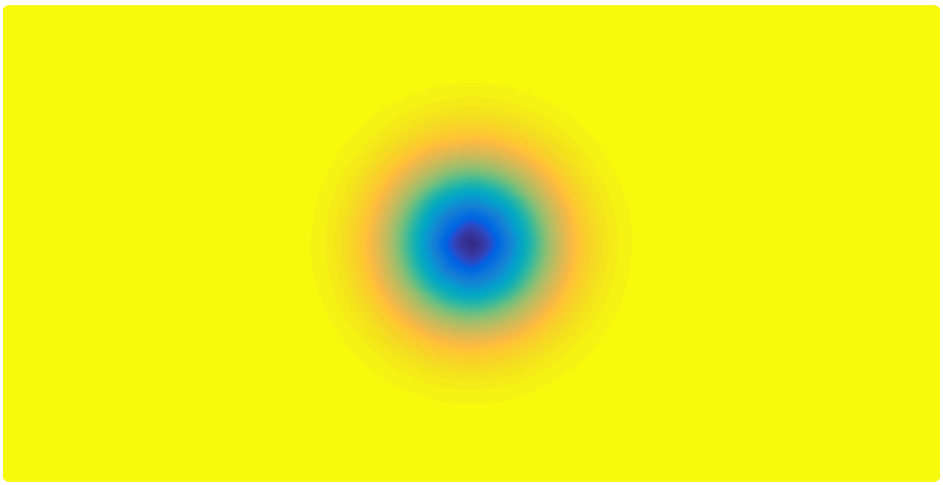

Figure 3: A translating vortex in 2D

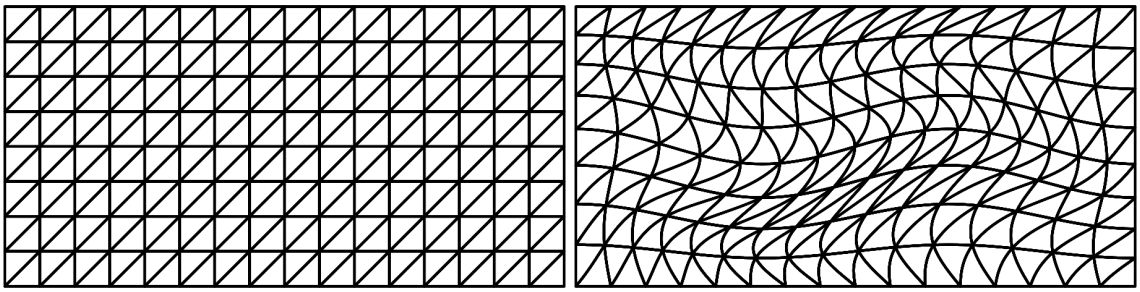

Figure 4: Meshes in 2D for translating vortex problem

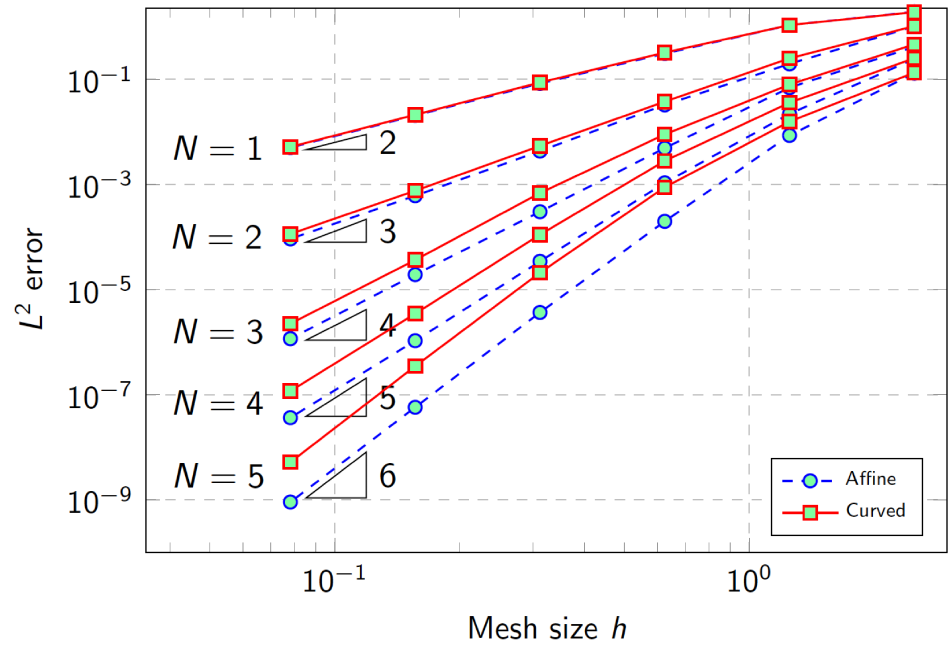

Figure 5: L_2 error for the translating vortex after 0.5 second on affine and curved meshes

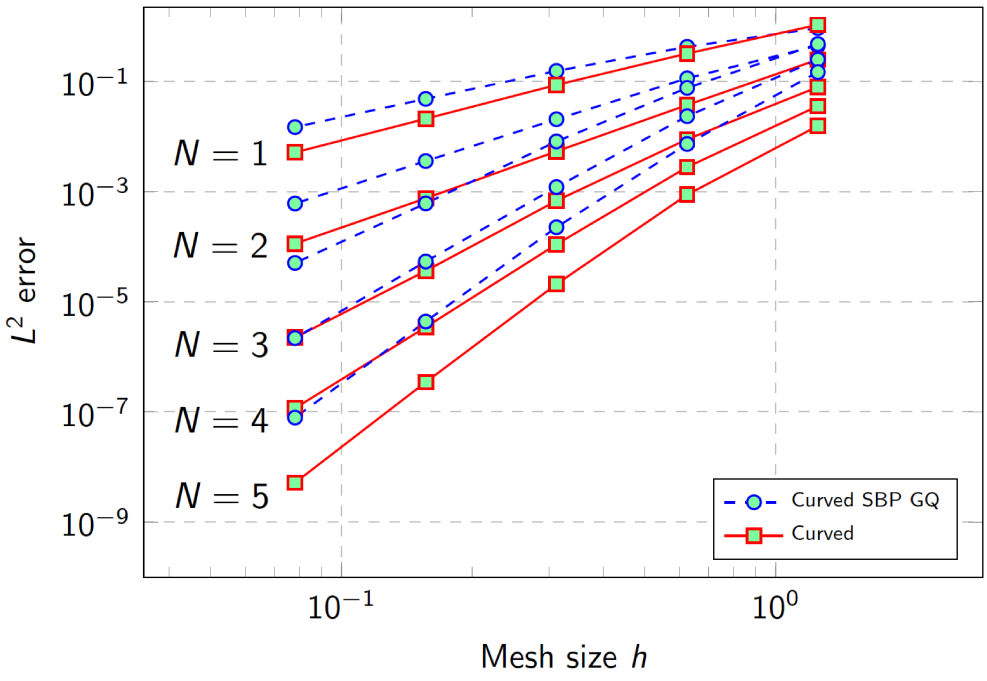

Figure 6: L_2 errors for the translating vortex after 0.5 seconds using hybridized DG and Gauss-Legendre SBP operator schemes on curved meshes.

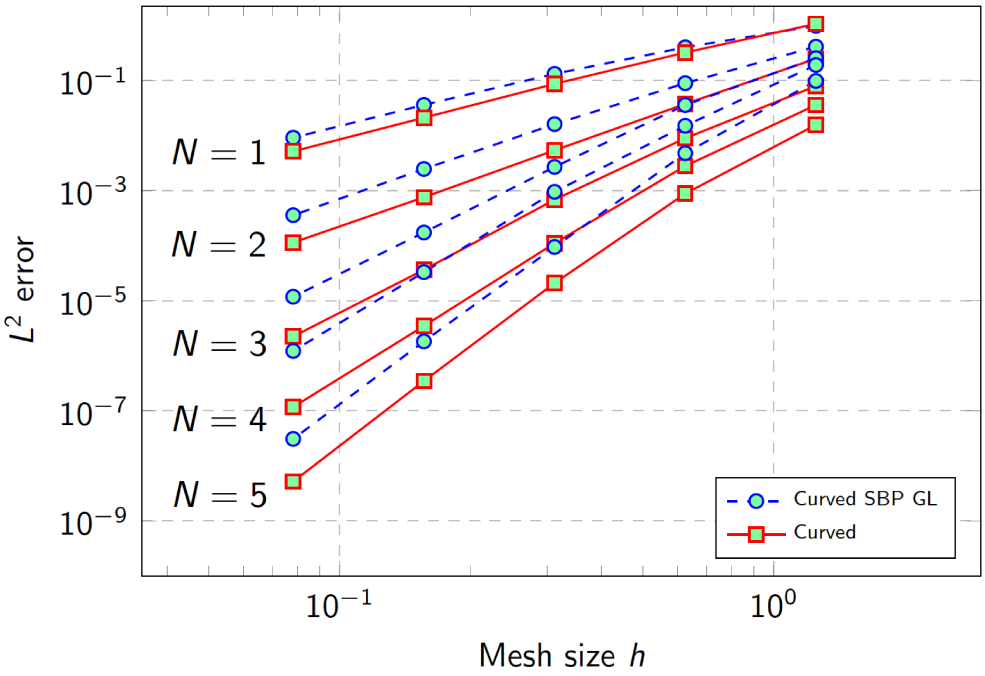

Figure 7: L_2 errors for the translating vortex after 0.5 seconds using hybridized DG and Gauss-Lobatto SBP operator schemes on curved meshes.

4.3 Dam break

This experiment is taken from [3, 37]. We utilize the same physical setting but use curved triangular meshes instead of curved quadrilateral meshes. The domain $[-10, 10]^2$ is discretized using a 20×20 grid of quadrilaterals, which are then split into triangular elements. We set $N = 3$ for the polynomial degree.

The dam is modeled by imposing reflective boundary conditions along the curve defined by the following function:

$$x = \frac{1}{25}y^2, \tag{4.12}$$

with a break between $y = -0.5$ and $y = 0.5$ to allow water to flow through, as shown in Figure 8. The dam is marked in red and fitted by the curved mesh.

We start with a constant water height on both sides of the dam with zero initial velocity. The bathymetry is set to $b = 0$ on both sides. We set the initial water height to be $h = 10$ on the left side of the dam and $h = 5$ on the right side as shown in Figure 8.

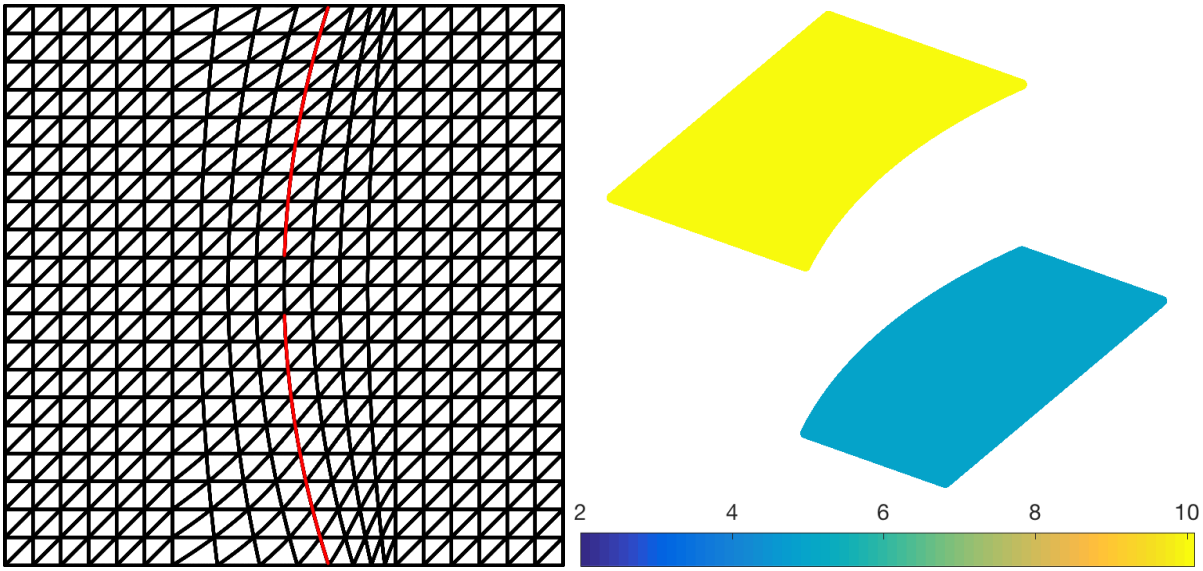

Figure 8: left: curved mesh for 2D dam break problem with the red curve denoting the broken dam. right: initial condition

We plot for the solution at several different times in Figures 9, 10 and 11.We observe that the water falling from the left side of the dam to the right produce a wave front in the lower half. This wave is discontinuous, so we observe some mesh dependent solution oscillations, but they are on a smaller scale and do not cause the solution to blow up. We have also tried simulations with polynomial degrees $N = 5$ and $N = 7$. Both of these simulations also remain stable throughout the duration (1.5s) of the run. The numerical solution of the parabolic dam break problem demonstrates that the entropy stable numerical remains robust on the presence of shock discontinuities.

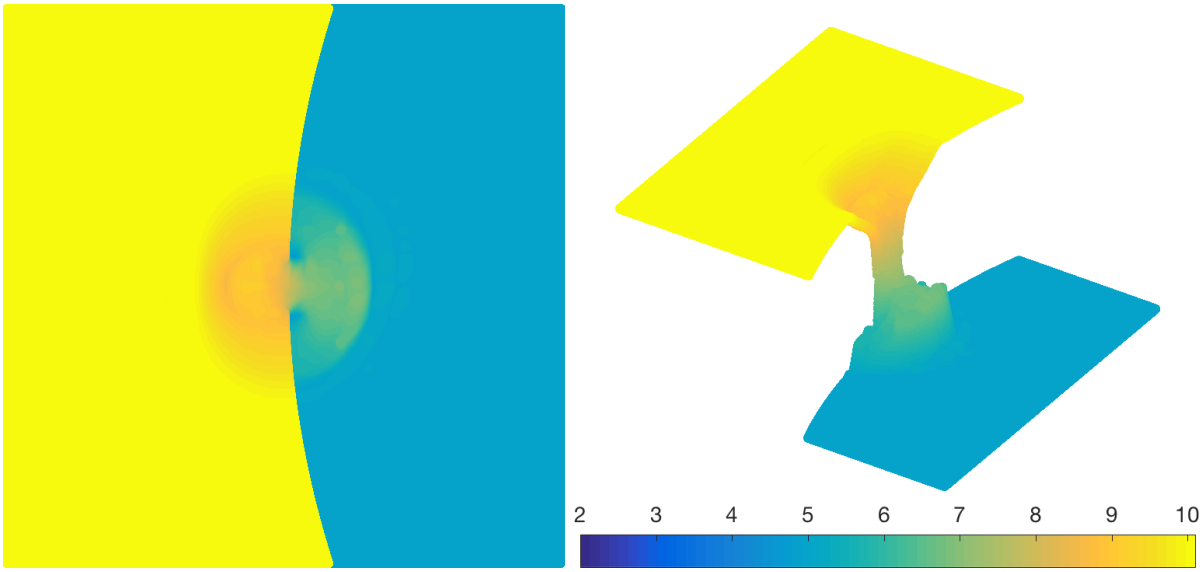

Figure 9: Top and side view of the dam breaking problem at T=0.5s

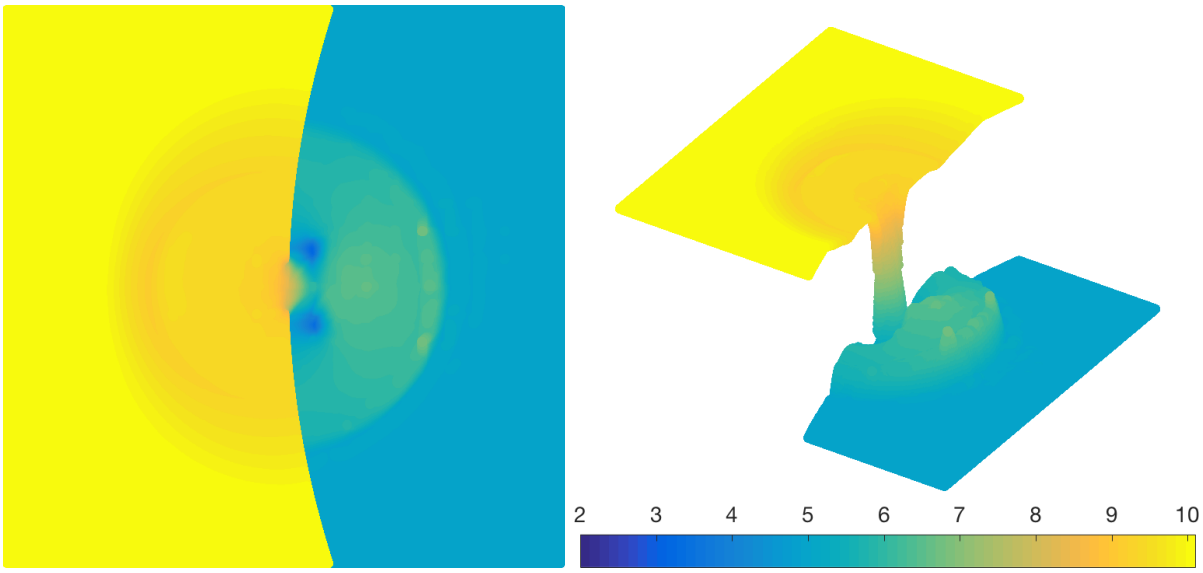

Figure 10: Top and side view of the dam breaking problem at T=1s

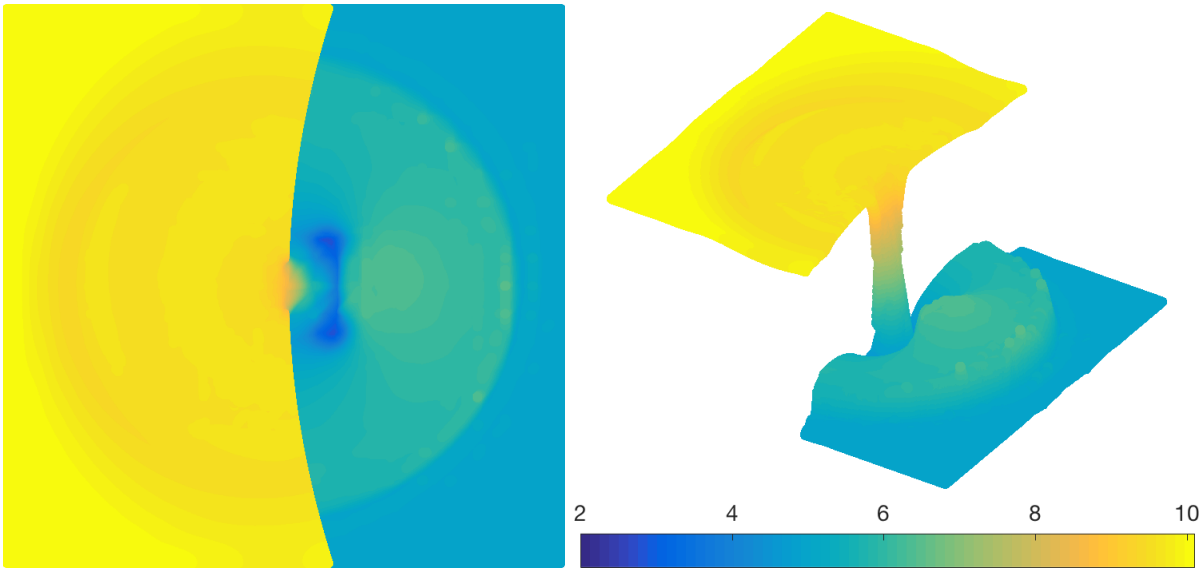

Figure 11: Top and side view of the dam breaking problem at T=1.5s

5 GPU optimization

The use of Graphic Processing Units (GPUs) in scientific computing has become well-established over the last 20 years. For example, the use of GPUs for accelerating the solution of PDE using finite difference methods is common in seismic applications [38]. GPUs have been widely used to accelerate explicit time-stepping schemes, which typically have high arithmetic intensity [32]. DG algorithms with explicit time integration are well suited to the parallel GPU architectures since most of the computational work is element local, and elements are only loosely coupled through shared interfaces. For each element, the computational intensity is also very high. In this section, we present the details of our GPU implementation and describe some computational optimization applied.

5.1 Infrastructure

We start by introducing our general coding infrastructure. The GPU implementation is written with C++ and OCCA code. OCCA is a unified approach to multi-threading languages, which compiles code written in the OCCA kernel language (OKL) at runtime for either CPU (Serial, OpenMP) or GPU (OpenCL, CUDA) architectures. We run all experiments on Google Cloud using a NVIDIA Tesla V100 GPU. We use double precision for all of our tests and the Tesla V100 is a device designed for scientific computations and better suited for double precision calculations than other general purpose GPUs.

We construct the mesh, quadrature nodes, and matrices on a CPU before moving on to the main time-stepping. We divide the iteration into four OCCA kernels: projection, volume, surface and update. The projection kernel performs the entropy projection. The volume kernel calculates

volume contributions while the surface kernel accumulates contributions from surface fluxes for each element. Finally, the update kernel applies the inverses of mass matrices (which are precomputed and stored on the GPU) on each element and performs time-integration.

In our entropy stable scheme, we apply flux differencing in the volume kernel, which computes the Hadamard product of differentiation and flux matrices over each element. The entries of the flux matrix are constructed on-the-fly by evaluating entropy conservative flux functions between each pair of nodal values of the solution. This operation involves significantly more computations than standard matrix vector multiplication, and is expected to be well suited to GPUs architectures. Despite this, the volume kernel is the most expensive kernel in our implementation.

5.2 Optimization

Following [3], we have applied some optimizations to our GPU implementation:

- Step 1: Utilizing Shared Memory.
  The first step in improving performance is to reduce the number of reads from GPU global memory. We load all solution values on an element into the shared memory, which avoids repeated access to slow global memory.

- Step 2: Declaring variables constant and pointers restricted.
  When a variable does not mutate during its lifespan, we declare the variable constant. We also use the restricted keyword on pointers to memory that is not referenced by other pointers. This allows the compiler to optimize performance. We also pre-compute all constants, (such as $\frac{1}{2}g$), before the start of the kernel to avoid repeated computation in each iteration.

- Step 3: Processing multiple elements per block.
  GPUs perform operations in synchronized groups of 32 threads, referred to as "warps". However, the natural number of threads used in each block is not typically a multiple of 32, which leads to idle threads and reduced performance on GPUs. In order to better utilize GPU resources and reduce the number of idle threads, we process multiple elements at the same time. In practice we manually optimize the block size to minimize run time of each kernel.

  This step, however, has its limitations. As the degree of the polynomial basis increases, the amount of shared memory used by each element also increases, but we cannot exceed the total GPU shared memory when combining multiple elements in to the same block.

- Step 4: Finally, we utilize an optimized implementation of the volume kernel based on matrix structure.
  The first three steps are common and can be applied to general GPU programming. Step 4 is more specific to the structure of the entropy stable DG formulation using hybridized SBP operators.

  We can reduce the computational cost of computing $(\boldsymbol{Q}_h^i \circ \boldsymbol{F})\mathbf{1}$, by avoiding computing the Hadamard product for any zero sub-blocks of the matrix $\boldsymbol{Q}_h^i$. Using the fact that $\boldsymbol{Q}_h^i = \boldsymbol{B}_h^i - (\boldsymbol{Q}_h^i)^T$, we can rewrite the DG formulation without projection as

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + \sum_{i=x,y} \left((\boldsymbol{Q}_h^i - (\boldsymbol{Q}_h^i)^T + \boldsymbol{B}_h^i) \circ \boldsymbol{F}^i \right) \mathbf{1} + \boldsymbol{B}_h^i \left(\boldsymbol{f}_S^i(\boldsymbol{u}^+, \boldsymbol{u}) - \boldsymbol{f}^i(\boldsymbol{u}) \right) = \boldsymbol{S}. \tag{5.1}$$

N	Baseline	Optimized	Improvement percentage
1	0.0368s	0.0221s	39.94%
2	0.0607s	0.0316s	47.94%
3	0.0947s	0.0515s	54.38%
4	0.1467s	0.0627s	57.26%
5	0.3202s	0.1585s	50.50%

Table 3: GPU runtime comparison between baseline and optimized version of implementation

Using the identity $(\boldsymbol{B}_h^i \circ \boldsymbol{F}^i)\mathbf{1} = \boldsymbol{B}_h^i \boldsymbol{f}^i(\boldsymbol{u})$, which follows from the consistency of the flux, we have

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + \sum_{i=x,y} \left((\boldsymbol{Q}^i - (\boldsymbol{Q}^i)^T) \circ \boldsymbol{F}_i\right)\mathbf{1} + \boldsymbol{B}^i \left(\boldsymbol{f}_S^i(\boldsymbol{u}^+, \boldsymbol{u})\right) = \boldsymbol{S}. \tag{5.2}$$

We define $\boldsymbol{Q}_{h,skew}^i = \boldsymbol{Q}_h^i - (\boldsymbol{Q}_h^i)^T$. Then, assembling all the pieces and combining the projection step, we can rewrite our DG scheme out as:

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{dt}} + \sum_{i=x,y} \begin{bmatrix} \boldsymbol{V}_q \\ \boldsymbol{V}_f \end{bmatrix}^T \left(2\boldsymbol{Q}_{h,skew}^i \circ \boldsymbol{F}^i\right)\mathbf{1} + \boldsymbol{V}_f^T \boldsymbol{B}_h^i \boldsymbol{f}_S^i = \boldsymbol{S}. \tag{5.3}$$

We observe that the matrix $\boldsymbol{Q}_{h,skew}^i$ has block structure where the lower right block is all zeros as shown in Figure 12. We split the original OCCA kernel that computes the Hadamard product of $\boldsymbol{Q}_{h,skew}^i$ and $\boldsymbol{F_i}$ into two kernels. The first computes the Hadamard product of the first N_q columns and accumulates the row sum into a vector, which corresponds to the part of the matrix in red box in Figure 12. The second kernel repeats the same process the upper right part of the matrix in the blue box as in Figure 12, then updates the accumulating vector counter for the first N_q entries. This eliminates the work of computing the Hadamard product of zeros entries in the matrix $\boldsymbol{Q}_{h,skew}^i$, which consists of approximately one quarter of the computational work.

5.3 Performance comparison

We conduct our GPU experiments on Google Cloud and provide the iteration time per step for K = 65536 elements. The results are presented in table 3. We notice that the percentages of improvement cluster around 50% for most of the experiments except for $N = 1$. The "baseline code" includes optimization Step 1, but not Steps 2-4.

5.4 Performance comparison between ESDG and traditional DG

The main difference between an entropy stable DG scheme and a traditional DG scheme is the volume kernels, which computes the DG approximation to a flux derivative. An entropy stable DG

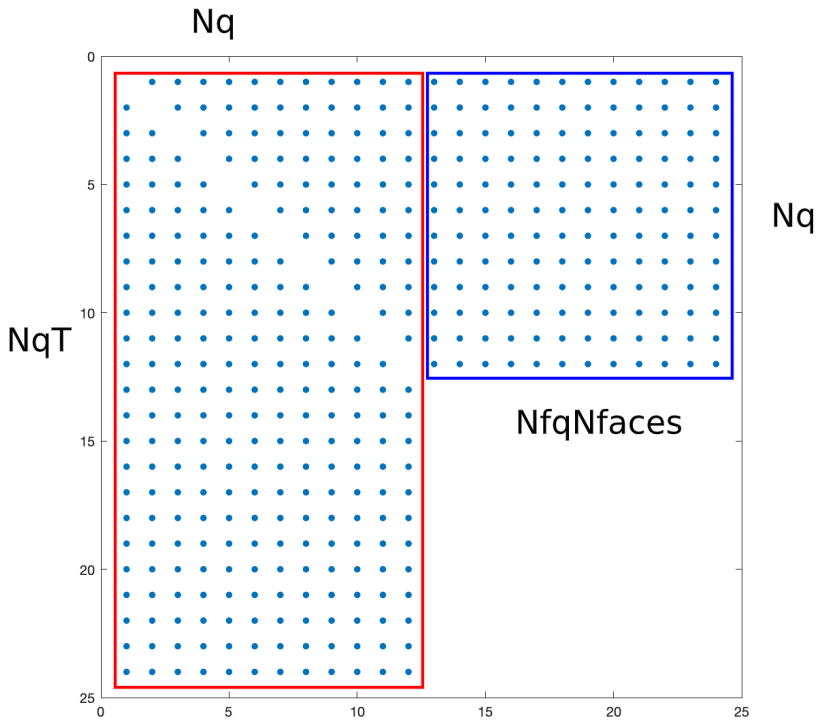

Figure 12: Spy diagram for $\boldsymbol{Q}^i_{h,skew} = \boldsymbol{Q}^i_h - (\boldsymbol{Q}^i_h)^T$

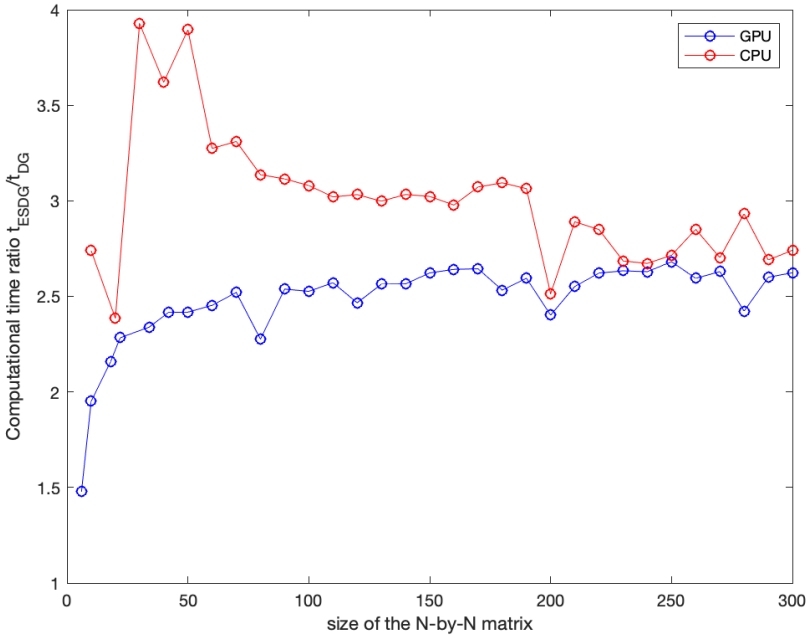

Figure 13: Ratio of runtime between an entropy stable and traditional DG volume kernel.

scheme computes a Hadamard product between two matrices, while a traditional DG method computes a regular matrix vector product. In order to explore how the GPU affects the computational efficiency of each approach, we compare both traditional and entropy stable volume kernels on both the CPU and GPU. The traditional volume kernel computes a dense matrix-vector product, while the entropy stable volume kernel performs flux differencing. Instead of actual SBP differentiation matrices, the kernels use square matrices with randomly generated entries in order to test computational performance for a wide range of matrix sizes. We note that these kernels most closely resemble volume kernels for traditional SBP schemes. The cost of the volume kernel for hybridized SBP operators will be higher due to the larger size of the matrices involved.

We define the run times to finish the entropy stable DG volume kernel and traditional DG volume kernel as t_{ESDG} and t_{DG}, respectively. We denote the cost ratio R_{CPU} as the ratio of t_{ESDG} to t_{DG} on the CPU, and define R_{GPU} to be the same ratio for the GPU run time. We plot R_{CPU} and R_{GPU} for various matrix size in Figure 13.

We observe that the cost ratio is always lower for the GPU, but not very significantly. This implies that GPU implementation reduces the gap between ESDG and traditional DG computational times only slightly. For low to moderate degree N, GPU implementations perform better for the ESDG kernel relative to the CPU implementation. To interpret Figure 13, we point out that a 50×50 matrix size usually translates into a set of SBP nodes for a degree 5 or 6 polynomial approximation in 2D. The ratio drops again when the matrix size reaches around 200×200, which corresponds to a polynomial approximation of degree $N \approx 12$.

The number of FLOPS for the computation of the term $(\boldsymbol{Q}^i_{h,skew} \circ \boldsymbol{F}^i)\mathbf{1}$ in our our triangular ESDG kernels is asymptotically the same as the number of operations needed to compute the matrix-vector product $\boldsymbol{Q}^i \boldsymbol{f}^i$ for the conventional DG method. Both involve operations involving

N_q^2 matrix entries; however, we have N_q^2 flux evaluations in ESDG but only N_q flux evaluations for the conventional DG method. In our ESDG implementation, we calculate entries of the flux matrix $\boldsymbol{F}_{ij}$ for each non-zero entry of $\boldsymbol{Q}_{h,skew}^i$.

Since the number of operations is $O(N_q^2)$ for both regular and ESDG, we expect the ratio of runtimes to eventually converge to a constant value. The ratio of 2.5 suggests that the cost of evaluating the flux for each non-zero matrix entry is 2.5x more expensive than computing contributions from a single matrix entry to a matrix-vector product for the conventional DG method.

This behavior is very different from the results shown in Figure 7 in [3], where the author found that routines of the traditional kernel and ESDG volume kernel were about the same for degree $N = 1, ..., 7$ on the GPU. The ESDG GPU implementation on quadrilateral meshes is memory-bound due to the fact that quadrilateral SBP operators are Kronecker products of 1D operators. Because of this, computational kernels can apply SBP operators in a more efficient manner using small $(N+1) \times (N+1)$ matrices corresponding to one-dimensional discretizations. The memory-bound nature of this kernel implies that memory transfers are the bottleneck. Thus, increasing the number of arithmetic operations does not increase the overall runtime until the cost of these operations exceeds the cost of memory traffic. As the polynomial order N increases, the additional computation introduced in the ESDG volume kernel eventually increases the cost beyond that of the volume kernel for traditional DG. For N sufficiently large, we expect the ratio of runtimes between ESDG and traditional DG on quadrilateral meshes to also approach a fixed value. However, it is not immediately clear that the ratio should approach 2.5, as this may depend on the hardware used and specific implementations. On triangular meshes, our results show that the ESDG volume kernel is at least 1.5 times slower than the traditional DG volume kernel, and usually about 2.5 times slower for higher polynomial degrees. This difference arises due to the use the triangular meshes instead of quadrilateral meshes. Traditional volume kernels on triangles are not bandwidth bound [39], and performing additional arithmetic operations results in a more significant increase in runtime.

6 Conclusions

In this work we present and compare two high-order entropy conserving and entropy stable schemes for the two dimensional shallow water equations on general curved triangular meshes. We construct well-balanced and high order entropy stable DG schemes for nonlinear conservation laws using hybridized SBP operators. The resulting schemes satisfy a discrete conservation or dissipation of entropy. We compared entropy stable DG methods with DG-SBP methods based on traditional SBP operators, and compared the computational cost of entropy stable DG methods with traditional DG methods for both GPUs and CPUs.

Acknowledgement

Authors Xinhui Wu and Jesse Chan gratefully acknowledge support from the National Science Foundation under awards DMS-1719818 and DMS-1712639. Author Ethan Kubatko gratefully acknowledges support from the National Science Foundation Grants ICER-1854991 and EAR-1520870. The authors also thank Benjamin Yeager for helping with the creation of the SBP quadrature rules.

References

[1] Jesse Chan. On discretely entropy conservative and entropy stable discontinuous Galerkin methods. *Journal of Computational Physics*, 362:346–374, 2018.

[2] Tianheng Chen and Chi-Wang Shu. Entropy stable high order discontinuous Galerkin methods with suitable quadrature rules for hyperbolic conservation laws. *Journal of Computational Physics*, 345:427–461, 2017.

[3] Niklas Wintermeyer, Andrew R Winters, Gregor J Gassner, and Timothy Warburton. An entropy stable discontinuous Galerkin method for the shallow water equations on curvilinear meshes with wet/dry fronts accelerated by GPUs. *Journal of Computational Physics*, 375:447–480, 2018.

[4] Clint N Dawson and Christopher M Mirabito. The shallow water equations. *University of Texas, Austin*, 29, 2008.

[5] Clint N Dawson, Ethan J Kubatko, Joannes J Westerink, Corey Trahan, Christopher Mirabito, Craig Michoski, and Nishant Panda. Discontinuous Galerkin methods for modeling hurricane storm surge. *Advances in Water Resources*, 34(9):1165–1176, 2011.

[6] Muhammad Akbar and Shahrouz Aliabadi. Hybrid numerical methods to solve shallow water equations for hurricane induced storm surge modeling. *Environmental modelling & software*, 46:118–128, 2013.

[7] Gregor J Gassner, Andrew R Winters, and David A Kopriva. A well balanced and entropy conservative discontinuous Galerkin spectral element method for the shallow water equations. *Applied Mathematics and Computation*, 272:291–308, 2016.

[8] Sebastian Noelle, Yulong Xing, and Chi-Wang Shu. High-order well-balanced finite volume WENO schemes for shallow water equation with moving water. *Journal of Computational Physics*, 226(1):29–58, 2007.

[9] Niklas Wintermeyer, Andrew R Winters, Gregor J Gassner, and David A Kopriva. An entropy stable nodal discontinuous Galerkin method for the two dimensional shallow water equations on unstructured curvilinear meshes with discontinuous bathymetry. *Journal of Computational Physics*, 340:200–242, 2017.

[10] Ulrik S Fjordholm, Siddhartha Mishra, and Eitan Tadmor. Well-balanced and energy stable schemes for the shallow water equations with discontinuous topography. *Journal of Computational Physics*, 230(14):5587–5609, 2011.

[11] Yulong Xing. Exactly well-balanced discontinuous Galerkin methods for the shallow water equations with moving water equilibrium. *Journal of Computational Physics*, 257:536–553, 2014.

[12] Xiao Wen, Wai Sun Don, and Yulong Xing. Entropy Stable Discontinuous Galerkin Method for Shallow Water Equations. *Submitted to Journal of Scientific Computing*, 2020.

[13] Jared Crean, Jason E Hicken, David C Del Rey Fernández, David W Zingg, and Mark H Carpenter. High-order, entropy-stable discretizations of the euler equations for complex geometries. In *23rd AIAA Computational Fluid Dynamics Conference. American Institute of Aeronautics and Astronautics*, 2017.

[14] Tianheng Chen and Chi-Wang Shu. Review of entropy stable discontinuous Galerkin methods for systems of conservation laws on unstructured simplex meshes. *submitted to CSIAM Transactions on Applied Mathematics*, 2019. Accessed 3/18/20.

[15] Eitan Tadmor. Entropy stability theory for difference approximations of nonlinear conservation laws and related time-dependent problems. *Acta Numerica*, 12:451–512, 2003.

[16] Michael S Mock. Systems of conservation laws of mixed type. *Journal of Differential equations*, 37(1):70–88, 1980.

[17] Travis C Fisher and Mark H Carpenter. High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains. *Journal of Computational Physics*, 252:518–557, 2013.

[18] Gregor J Gassner, Andrew R Winters, Florian J Hindenlang, and David A Kopriva. The BR1 scheme is stable for the compressible Navier–Stokes equations. *Journal of Scientific Computing*, 77(1):154–200, 2018.

[19] Gregor J Gassner, Andrew R Winters, and David A Kopriva. Split form nodal discontinuous Galerkin schemes with summation-by-parts property for the compressible Euler equations. *Journal of Computational Physics*, 327:39–66, 2016.

[20] Eitan Tadmor. The numerical viscosity of entropy stable schemes for systems of conservation laws. I. *Mathematics of Computation*, 49(179):91–103, 1987.

[21] Ulrik S Fjordholm, Siddhartha Mishra, and Eitan Tadmor. Arbitrarily high-order accurate entropy stable essentially nonoscillatory schemes for systems of conservation laws. *SIAM Journal on Numerical Analysis*, 50(2):544–573, 2012.

[22] Mark H Carpenter, Travis C Fisher, Eric J Nielsen, and Steven H Frankel. Entropy Stable Spectral Collocation Schemes for the Navier–Stokes Equations: Discontinuous Interfaces. *SIAM Journal on Scientific Computing*, 36(5):B835–B867, 2014.

[23] Jared Crean, Jason E Hicken, David C Del Rey Fernández, David W Zingg, and Mark H Carpenter. Entropy-stable summation-by-parts discretization of the Euler equations on general curved elements. *Journal of Computational Physics*, 356:410–438, 2018.

[24] Hendrik Ranocha, Philipp Öffner, and Thomas Sonar. Extended skew-symmetric form for summation-by-parts operators and varying Jacobians. *Journal of Computational Physics*, 342:13–28, 2017.

[25] Jesse Chan and Lucas C Wilcox. On discretely entropy stable weight-adjusted discontinuous Galerkin methods: curvilinear meshes. *Journal of Computational Physics*, 378:366–393, 2019.

[26] Jesse Chan. Skew-symmetric entropy stable modal discontinuous Galerkin formulations. *Journal of Scientific Computing*, 81(1):459–485, 2019.

[27] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[28] Jason E Hicken and David W Zingg. Summation-by-parts operators and high-order quadrature. *Journal of Computational and Applied Mathematics*, 237(1):111–125, 2013.

[29] David C Del Rey Fernández, Jason E Hicken, and David W Zingg. Review of summation-by-parts operators with simultaneous approximation terms for the numerical solution of partial differential equations. *Computers & Fluids*, 95:171–196, 2014.

[30] Damrongsak Wirasaet, Steven R Brus, Craig E Michoski, Ethan J Kubatko, Joannes J Westerink, and Clint N Dawson. Artificial boundary layers in discontinuous Galerkin solutions to shallow water equations in channels. *Journal of Computational Physics*, 299:597–612, 2015.

[31] Mark H Carpenter and Christopher A Kennedy. Fourth-order 2N-storage Runge-Kutta schemes. *NASA-TM-109112*, 1994.

[32] Jesse Chan, Zheng Wang, Axel Modave, Jean-François Remacle, and Tim Warburton. GPU-accelerated discontinuous Galerkin methods on hybrid meshes. *Journal of Computational Physics*, 318:142–168, 2016.

[33] Timothy Warburton and Jan S Hesthaven. On the constants in hp-finite element trace inverse inequalities. *Computer methods in applied mechanics and engineering*, 192(ARTICLE):2765–2773, 2003.

[34] Randall J LeVeque. Balancing source terms and flux gradients in high-resolution Godunov methods: the quasi-steady wave-propagation algorithm. *Journal of computational physics*, 146(1):346–365, 1998.

[35] Shinhoo Kang, Francis X Giraldo, and Tan Bui-Thanh. IMEX HDG-DG: A coupled implicit hybridized discontinuous Galerkin and explicit discontinuous Galerkin approach for shallow water systems. *Journal of Computational Physics*, 401:109010, 2020.

[36] Mario Ricchiuto and Andreas Bollermann. Stabilized residual distribution for shallow water simulations. *Journal of Computational Physics*, 228(4):1071–1115, 2009.

[37] Mária Lukácová-Medvid'ová and Eitan Tadmor. On the entropy stability of the Roe-type finite volume methods. In *Proceedings of the twelfth international conference on hyperbolic problems, American Mathematical Society, eds. J.-G. Liu et al*, volume 67, 2009.

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

[39] Jesse Chan and Tim Warburton. GPU-Accelerated Bernstein–Bézier Discontinuous Galerkin Methods for Wave Problems. *SIAM Journal on Scientific Computing*, 39(2):A628–A654, 2017.