Commun. Comput. Phys.
doi: 10.4208/cicp.070114.271114a

Vol. **18**, No. 1, pp. 37-64
July 2015

GPU Accelerated Discontinuous Galerkin Methods for Shallow Water Equations

Rajesh Gandham[1,*], David Medina[1] and Timothy Warburton[1]

[1] *Department of Computational and Applied Mathematics, Rice University, 6100 Main Street, MS-134, Houston, TX-77005, USA.*

Received 7 January 2014; Accepted (in revised version) 27 November 2014

Abstract. We discuss the development, verification, and performance of a GPU accelerated discontinuous Galerkin method for the solutions of two dimensional nonlinear shallow water equations. The shallow water equations are hyperbolic partial differential equations and are widely used in the simulation of tsunami wave propagations. Our algorithms are tailored to take advantage of the single instruction multiple data (SIMD) architecture of graphic processing units. The time integration is accelerated by local time stepping based on a multi-rate Adams-Bashforth scheme. A total variational bounded limiter is adopted for nonlinear stability of the numerical scheme. This limiter is coupled with a mass and momentum conserving positivity preserving limiter for the special treatment of a dry or partially wet element in the triangulation. Accuracy, robustness and performance are demonstrated with the aid of test cases. Furthermore, we developed a unified multi-threading model OCCA. The kernels expressed in OCCA model can be cross-compiled with multi-threading models OpenCL, CUDA, and OpenMP. We compare the performance of the OCCA kernels when cross-compiled with these models.

AMS subject classifications: 35Q35, 35L60, 35L65, 65Z05, 65M99

Key words: Shallow water equations, discontinuous Galerkin, positivity preserving, slope limiting, GPUs, accelerators, multi-threading.

1 Introduction

The shallow water equations (SWE) are of great interest in the modeling of tsunamis, storm surges and tidal waves. They are the simplest nonlinear models for water wave propagation. The shallow water assumptions simplify three-dimensional wave propagation to two-dimensional hyperbolic partial differential equations, reducing the complexity of the model. The reduced complexity makes the shallow water equations attractive

*Corresponding author. *Email addresses:* Rajesh.Gandham@rice.edu (R. Gandham), David.S.Medina@rice.edu (D. Medina), timwar@rice.edu (T. Warburton)

for tsunami modeling. These equations are valid for long waves but may represent the wave propagation of short waves or dispersive waves poorly. However, these simplified equations provide satisfactory solutions of tsunami wave propagation [18] over long distances.

The shallow water equations are two-dimensional hyperbolic PDEs with velocity and fluid height as unknown quantities. These equations are complicated by the presence of largely varying length scales, varying bathymetry, and nonlinear effects near the shore. Stable, accurate and efficient algorithms are of great interest for these applications.

There is an extensive literature for finite difference [4, 21], finite volume [2, 18] and finite element [22] methods for shallow water equations. Recently, there has been growing interest in using discontinuous Galerkin methods (DG) for solutions of the shallow water equations [1,8,11,17]. DG methods are locally mass conservative like finite volume methods and can achieve high order accuracy on unstructured meshes like finite element methods. This allows flexibility in handling irregular boundaries with out compromising accuracy for problems with sufficiently smooth solutions. DG methods can achieve $\mathcal{O}(H^{N+1/2})$ accuracy with a piecewise degree N polynomial approximation [14]. Where, H is the largest length scale in the mesh.

In DG formulations, elements are coupled using weak penalty terms, resulting in localized memory access. Furthermore, a high order polynomial representation of the solution in each element results in high arithmetic intensity per degree of freedom. Both of these features are well suited for the GPU hardware architecture [12, 15, 16]. This motivated us to adopt GPUs along with a nodal DG discretization for large-scale tsunami simulations. Furthermore, to alleviate the need to write kernels for thread models like OpenMP, CUDA, and OpenCL separately, we developed OCCA: A unified approach to multi-threading languages. Kernels written in OCCA are cross compiled with any of these thread models at runtime. This gives us the flexibility in choosing the most efficient multi-threading model for a given hardware architecture, without writing new codes.

This paper is organized as follows: In Sections 2 and 3, we outline the governing equations and nodal discontinuous Galerkin discretization. In Section 4, we describe local time stepping using multi-rate Adams-Bashforth time integration. In Section 5, we explain the stabilization of numerical scheme using positivity preservation, wetting drying treatment and modified total variational bounded (TVB) limiter. Several tests for verification of accuracy and robustness are presented in Section 6. We discuss GPU kernels and their performance in Section 7. In Section 8, we will describe the fundamentals and features of the OCCA multi-threading model and compare the performance of the kernels written in OCCA, when they are cross compiled with OpenCL, CUDA, and OpenMP.

2 Governing equations

The shallow water equations are depth averaged incompressible Navier-Stokes equations, and are given in conservative form by [18],

$$\frac{\partial h}{\partial t}+\frac{\partial (hu)}{\partial x}+\frac{\partial (hv)}{\partial y}=0, \tag{2.1a}$$

$$\frac{\partial}{\partial t}(hu)+\frac{\partial}{\partial x}\left(hu^2+\frac{1}{2}gh^2\right)+\frac{\partial}{\partial y}(huv)=-gh\frac{\partial B}{\partial x}, \tag{2.1b}$$

$$\frac{\partial}{\partial t}(hv)+\frac{\partial}{\partial x}(huv)+\frac{\partial}{\partial y}\left(hv^2+\frac{1}{2}gh^2\right)=-gh\frac{\partial B}{\partial y}, \tag{2.1c}$$

where h, u, and v are water depth, depth averaged velocity components in longitudinal and latitudinal directions. B is bathymetry and g is the acceleration due to gravity (see Fig. 1 for the notation).

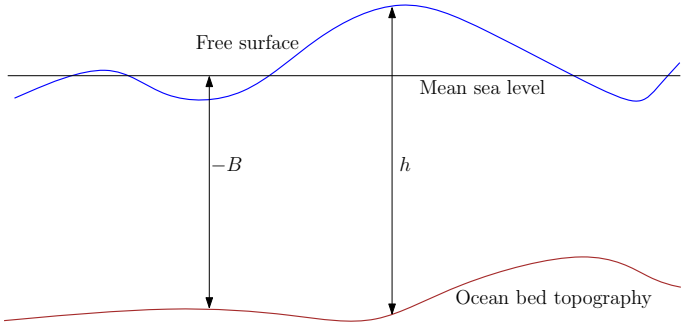


Figure 1: Diagram of notations.

In simplified form, these equations are represented as

$$\frac{\partial Q}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=S. \tag{2.2}$$

The state vector Q, nonlinear flux vectors F, G and the source vector S are given by

$$Q=\begin{pmatrix} h \\ hu \\ hv \end{pmatrix}, \quad F=\begin{pmatrix} hu \\ hu^2+\frac{gh^2}{2} \\ huv \end{pmatrix}, \quad G=\begin{pmatrix} hv \\ huv \\ hv^2+\frac{gh^2}{2} \end{pmatrix}, \quad S=\begin{pmatrix} 0 \\ -gh\frac{\partial B}{\partial x} \\ -gh\frac{\partial B}{\partial y} \end{pmatrix}. \tag{2.3}$$

We use a high order discontinuous Galerkin method to obtain the solution of Eq. (2.2). We explain the method in the next section.

3 Discretization

We assume the domain $\Omega \subset \mathbb{R}^2$, is partitioned in to a set of non-overlapping, conforming triangles $\{\Omega = \cup_k D^k\}$. We approximate the solution Q by Q_H, the components of

which belong to the space of discontinuous piecewise polynomial of a given degree N in each element ($P^N(D^k)$). The PDEs in the Eq. (2.2) are expressed in weak form. For each element we find $Q_H \in (P^N(D^k))^3$ such that,

$$\left(\frac{\partial Q_H}{\partial t},\phi\right)_{D^k}+\left(\frac{\partial F}{\partial x},\phi\right)_{D^k}+\left(\frac{\partial G}{\partial y},\phi\right)_{D^k}-(S,\phi)_{D^k}=0 \qquad \forall \phi \in P^N(D^k). \tag{3.1}$$

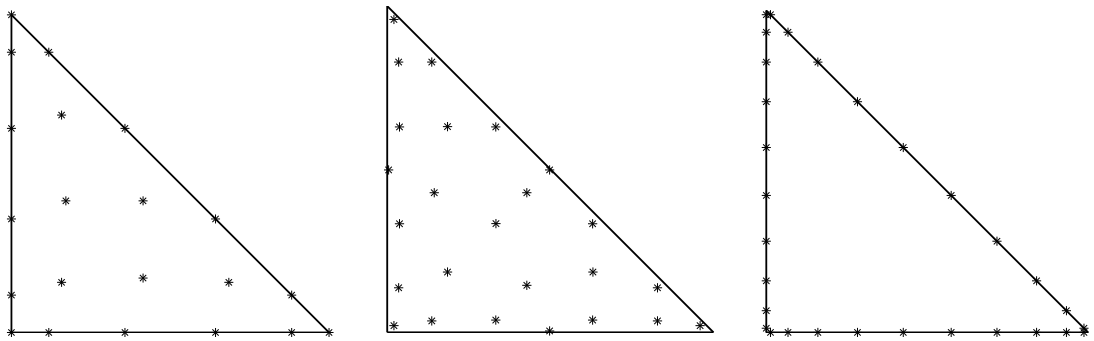
Integrating by parts and replacing the multi-valued fluxes on the boundary of each element with stable numerical fluxes F^* and G^*gives,

$$\left(\frac{\partial Q_H}{\partial t},\phi\right)_{D^k}=\left(\frac{\partial \phi}{\partial x},F\right)_{D^k}+\left(\frac{\partial \phi}{\partial y},G\right)_{D^k}-\left(F^*n_x+G^*n_y,\phi\right)_{\partial D^k}+(S,\phi)_{D^k}. \tag{3.2}$$

Here $(\cdot,\cdot)_{D^k}$ represents the inner product taken over the element D^k, while $(\cdot,\cdot)_{\partial D^k}$ represents the inner product taken over the boundary of the element D^k denoted by ∂D^k. We use Lagrange polynomials with Warp & Blend interpolation nodes [13] as the basis for the polynomial space in each element, and use well-balanced local Lax-Friedrich flux [27] to compute F^* and G^*. The volume integrals are computed using cubature rules for triangles [6], while the surface integrals are computed using Gauss quadrature rules (see Fig. 2), leading to a system of ordinary differential equations given by,

$$\frac{dQ_H}{dt}=\mathcal{R}(Q_H)=\mathcal{N}(Q_H)+\mathcal{S}(Q_H^{g,+},Q_H^{g,-}). \tag{3.3}$$

Here, $\mathcal{R}$ is the spatial discretization operator and $\mathcal{N}$, $\mathcal{S}$ are nonlinear operators corresponding to volume and surface integrations. Q_H^g is a vector of the state variables at the Gauss quadrature nodes. $Q_H^{g,+}$ and $Q_H^{g,-}$ represent the positive and negative traces of the solution at the element interfaces. The ODEs in Eq. (3.3) are integrated using a local time-stepping scheme to obtain the solution at a given time t.



(a) Interpolation nodes (b) Cubature nodes (c) Gauss quadrature nodes

Figure 2: The distribution of interpolation nodes, cubature integration nodes, and Gauss quadrature integration nodes for the interpolating polynomial order 5, and the integration order 10.

4 Local time stepping

Explicit time integration is conditionally stable; the time step Δt, has to satisfy Courant-Friedrichs-Lewy criterion,

$$\Delta t = \min_{D^k} \left\{ \frac{H^k}{a} C \right\}, \qquad a = |(u,v)| + \sqrt{gh}. \tag{4.1}$$

Here, H^k is the characteristic length of the element D^k, a is the wave speed, and C is a constant that depends on the stability region of the time integration scheme and spatial polynomial order.

For a global scheme, the smallest length scale in the mesh determines the overall time step, which leads to a very small allowable time step size. To allow each element to be integrated with its own allowable time step, we adopt a local time stepping based on the multi-level third order AB scheme [10, 12] to nonlinear ODEs. For the shallow water applications, mesh resolutions can vary in few orders of magnitude because a very fine mesh is used near the shore regions to resolve high frequencies in the waves, while a coarse mesh is sufficient to resolve the flow in the deep oceans. The global allowable time step is few orders of magnitude smaller than the allowable time step for a coarse element.

Local time stepping is done efficiently by grouping the elements into levels based on their characteristic lengths and integrating the elements in a level with a fixed time step size. After mesh generation, the individual time step for each element is evaluated and elements with an allowable time step Δt are grouped in to a level l if $2^{l-1}\Delta t_{\min} \leq \Delta t < 2^l \Delta t_{\min}$. All elements in level l are integrated with a time step $2^{l-1}\Delta t_{\min}$. Once the elements are grouped in to levels, they are re-grouped such that any two neighbor elements are at most one level apart. The elements with smallest allowable time step size (or finer elements) are integrated first followed by elements with larger allowable time step size (or coarser elements). At the interface of the coarse and fine elements, the field values of the coarse elements at intermediate time step are obtained using the extrapolation of the AB time stepping scheme, that uses right hand side evaluations at previous time steps.

For a given initial condition Q^0, the time integration procedure is described in the Algorithm 1. $\Pi_H Q^0$ is the projection of the initial conditions on to the space of piecewise discontinuous polynomials, n is a time level, and $Nlevels$ is the total number of levels in multi-rate scheme. α_i's are coefficients corresponding to the 3^{rd} order Adams-Bashforth linear multi-step method and β_i's are coefficients corresponding to extrapolation at the coarse element interfaces. $\Delta t_l (= 2^{l-1}\Delta t_{\min})$ is the time step size for elements in level l, $Q_H^{n+\frac{nstep}{Nsteps}}\big|_l$ is the numerical solution corresponding to the elements in level l and time $t = t^n + \frac{nstep}{Nstep}\Delta t_{Nlevels}$, here $Nsteps (= 2^{Nlevels})$ is the number of intermediate time steps needed to complete a time step update.

Algorithm 1 Multi level Adam-Bashforth explicit time stepping

$Q_H^0 = \Pi_H Q^0$ { project initial conditions on to polynomial space }
for $n = 0,1,2,\cdots$ **do** { time stepping until final time }
  $\mathcal{R}(Q_H^n) = \mathcal{N}(Q_H^n) + \mathcal{S}(Q_H^{g+,n}, Q_H^{g-,n})$ { evaluate right hand side function }
  for $nstep = 1,2,\cdots,Nsteps$ **do** { loop over intermediate time steps }
    for $l = 1,2,\cdots,Nlevels$ **do** { loop over the levels from fine to coarse }
      if $nstep \% 2^{l-1} = 0$ **then** { if the step is divisible by the level time step }

$$Q_H^{n+\frac{nstep}{Nsteps}}\Big|_l = Q_H^{n+\frac{nstep-2^{l-1}}{Nsteps}}\Big|_l + \Delta t_l \sum_{s=1}^{3} \alpha_s \, \mathcal{R}\left(Q_H^{n+\frac{nstep-s\times 2^{l-1}}{Nsteps}}\right)\Big|_l$$

{ integrate elements in this level }

$$Q_H^{g-,n+\frac{nstep}{Nsteps}}\Big|_{l+1} = Q_H^{g-,n+\frac{nstep-2^{l}}{Nsteps}}\Big|_{l+1} + \Delta t_l \sum_{s=1}^{3} \beta_s \, \mathcal{R}^{g-}\left(Q_H^{n+\frac{nstep-s\times 2^{l}}{Nsteps}}\right)\Big|_{l+1}$$

{ update coarse interface fields, if any }

$$\mathcal{R}\left(Q_H^{n+\frac{nstep}{Nsteps}}\right)\Big|_l = \mathcal{N}\left(Q_H^{n+\frac{nstep}{Nsteps}}\right)\Big|_l + \mathcal{S}\left(Q_H^{g+,n+\frac{nstep}{Nsteps}}, Q_H^{g-,\frac{nstep}{Nsteps}}\right)$$

{ update the right hand side for this level }

From the Eq. (4.1), it can be observed that size of the time step for the stability changes over time with the change in characteristic speeds and the elements need to be regrouped in to levels. Grouping the elements into levels carries a nontrivial setup cost and hence is inefficient to do this every time step. In this work, the allowable time step for each element is fixed throughout the simulations, and our experiments indicate that changes in characteristic speeds are not strong enough to cause instabilities in the solutions and do not pollute the spectral accuracy. However, if needed, elements can be regrouped after every few time steps to satisfy the updated CFL conditions. Please note that, in doing so, the history of the data for the solution and fluxes may need to be interpolated or extrapolated.

5 Well-balancing, positivity preserving and slope limiting schemes

5.1 Well-balancing

In the numerical solutions of shallow water equations, it is possible to have nonzero flux derivatives while maintaining a steady state. In these cases, the derivatives are balanced by the source terms from the bathymetry distribution. The numerical schemes have to exactly balance these terms (exact C-property). We choose a numerical flux proposed in [26,27] to satisfy the exact C-property.

$$h^{*,\pm} = \max(0, h^{\pm} + B^{\pm} - \max(B^+, B^-)), \tag{5.1a}$$

$$Q^{*,\pm} = \begin{pmatrix} h^{*,\pm} \\ h^{*,\pm}u^{\pm} \\ h^{*,\pm}v^{\pm} \end{pmatrix}. \tag{5.1b}$$

Here $+$ and $-$ refer to positive and negative trace of field values at the element interfaces. Q^* is intermediate field vector. The well-balanced fluxes are given by,

$$F^{*,\pm} = F_{LF}(Q^{*,+},Q^{*,-}) + \begin{pmatrix} 0 \\ \frac{g}{2}(h^{\pm})^2 - \frac{g}{2}(h^{*,\pm})^2 \\ 0 \end{pmatrix}, \tag{5.2a}$$

$$G^{*,\pm} = G_{LF}(Q^{*,+},Q^{*,-}) + \begin{pmatrix} 0 \\ 0 \\ \frac{g}{2}(h^{\pm})^2 - \frac{g}{2}(h^{*,\pm})^2 \end{pmatrix}, \tag{5.2b}$$

where F_{LF}, G_{LF} are Lax-Friedrich fluxes at the element interfaces.

5.2 Positivity preserving limiter

Another concern in the numerical simulation of the shallow water equations is the appearance of dry areas where no water is present. The shallow water equations implicitly assume non-negative depth of water and hence this property has to be ensured by the numerical scheme. Otherwise, the flux Jacobian will have non-real eigenvalues and the PDE will no longer be hyperbolic. For a finite volume method, the water depth is a constant in each element. An element is flagged as dry if the fluid height is below a threshold value and is not considered as part of the simulation until the fluid height reaches the threshold value. This is no longer valid for a high order DG discretization since the height is typically not a constant but a non-monotonic high order polynomial. A wetting drying scheme that conserves mass and momentum is required. Positivity preserving algorithms have been developed under the assumptions of piecewise linear approximation [3] for the shallow water equations and rectangular elements [28] for the Euler equations. These were later extended to high order polynomials on rectangular elements for shallow water flows [27]. These schemes cannot be extended to triangles due to the assumptions made on numerical integration schemes. We adopt the approach presented for linear polynomials and triangular meshes in [3], and extend it to an arbitrary order polynomial approximation.

We represent the positive preserving operator with $M\Pi_H$, this operator is used along with a TVB limiter $\Lambda\Pi_H$ at every intermediate time step in the multi-rate integration. First, we discuss in detail the operator $M\Pi_H$ that ensures positivity of the fluid height. To reduce the notational complexity, we represent the numerical solution Q_H with q. A given polynomial $q = (h, hu, hv)^T$ in an element D^k is modified such that the polynomial corresponding to the fluid height (h) is positive ($\geq h_0$) in the element, where h_0 is a threshold value of the fluid height for considering an element/region as dry land. At a time

Algorithm 2 Positivity preserving limiter $M\Pi Q_H := \tilde{q}^n$

$h^n_{\min} = \min_j h_j$ { minimum height across all interpolation and integration nodes }
if $h^n_{\min} > \epsilon$ **then** { if nodal values for height are +ve }
  $\tilde{q}^n = q^n, \quad \forall \boldsymbol{x} \in D^k$ { do not modify the solution }
else
  $q^{n,1} = \Pi^n_1 q^n$ { project the solution to linear polynomial }
  if $\bar{h^n} < h_0$ **then** { if dry element, i.e., mean is less than cutoff }
    $h^n = h_0$, $hu^n = 0$, and $hv^n = 0$ { modify the solution }
  else
    if $\bar{h^n} \geq h_0$ **then** { if mean is positive }
      $h^{n,1}_{\min} = \min_i h^{n,1}_i, \quad \forall \boldsymbol{x}_i \in$ { vertices of the triangle }
      $\tilde{q}^n_j = \theta(q^{n,1}_j - \bar{q}^n) + \bar{q}^n, \qquad \theta = \min\left\{1, \dfrac{\bar{h}^{n,1} - h_0}{\bar{h}^n - h^{n,1}_{\min}}\right\},$

level n, the polynomial solution $q^n = (h^n, hu^n, hv^n)^T$ is modified to obtain $\tilde{q^n}$ using Algorithm 2.

It is easy to see that linear polynomial representation of the fluid height $h^{n,1}$ is positive ($\geq h_0$) at all interpolation nodes, and so at the integration points. Note that local conservation of mass and momentum are not violated for wet or partially wet elements. This is achieved by keeping the constant modes in the orthogonal polynomials unaltered during the limiting.

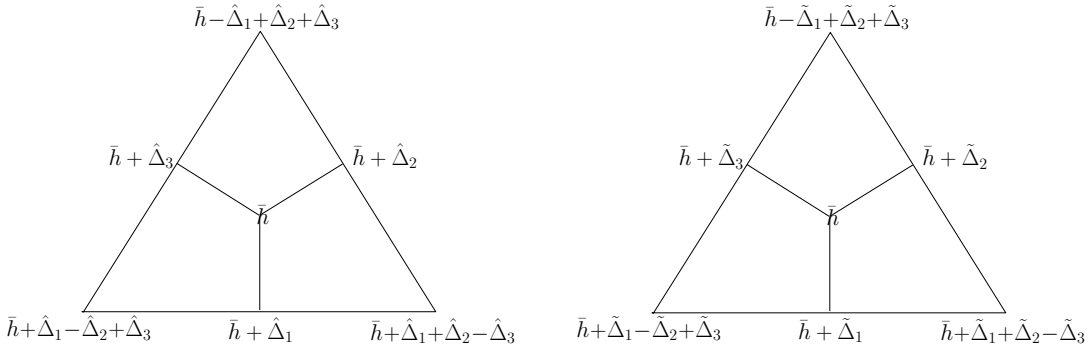
5.3 TVB slope limiter

A slope limiter removing high frequency oscillations in numerical solutions is applied to avoid instabilities due to nonlinear effects. We use Cockburn and Shu's characteristic based TVB limiter [5, 23] designed for Runge-Kutta methods. However, the TVB limiter does not ensure the positivity of the solution. Therefore, we perform a post processing of the solution to ensure positivity of the fluid height. We discuss the post-processing step without going into the details of the TVB limiter.

TVB slope limiter restricts the local solution to a linear polynomial. A linear polynomial attains its maximum and minimum at the vertices. We modify $\hat{\Delta}_1$,$\hat{\Delta}_2$,and $\hat{\Delta}_3$ (refer to [5] for the notation) to $\tilde{\Delta}_1$,$\tilde{\Delta}_2$,and $\tilde{\Delta}_3$ without changing the average of these to ensure the positivity of the fluid height at the vertices. The values at the vertices are given in Fig. 3,

$$\tilde{\Delta}_i = \bar{\Delta} + \theta(\hat{\Delta}_i - \bar{\Delta}), \qquad \text{for } i = \{1,2,3\}, \tag{5.3a}$$

$$\bar{\Delta} = \text{avg}(\hat{\Delta}_1, \hat{\Delta}_2, \hat{\Delta}_3), \tag{5.3b}$$



(a) TVB limiter (b) Modified TVB limiter

Figure 3: Modification of TVB limiter to ensure positivity of the fluid height.

$$\theta=\frac{\bar{h}+\bar{\Delta}-h_0}{\bar{\Delta}-\min\limits_{i,j,k=\{1,2,3\},i\neq j\neq k}\{-\hat{\Delta}_i+\hat{\Delta}_j+\hat{\Delta}_k\}}. \tag{5.3c}$$

As reported in [3,7], a positive preservation and/or a wetting drying treatment combined with a slope limiter may artificially activate each other causing instability. To avoid this instability, the TVB limiter is not applied for elements that are considered dry. For high order approximations, the TVB limiter may be artificially activated in the immediate neighbors of the dry elements also. Therefore, we do not apply the TVB limiter for dry elements and the immediate neighbors of dry elements as well. The robustness of these limiters and effect on the solution accuracy are demonstrated in the next section.

6 Verification

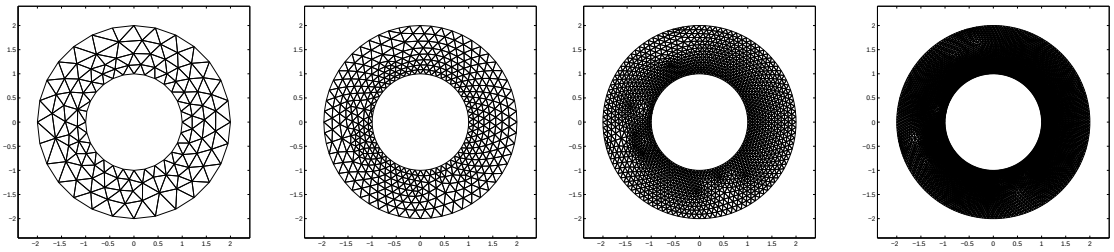
In this section, we use several test cases with known analytical solutions to study the discussed limiters. In all these test cases, we do not employ local time stepping, in other words, we use a local time stepping with only one level.

6.1 Accuracy test for smooth solution: Couette flow

We consider Couette flow between two concentric cylinders spinning at different velocities with the exact solution given by,

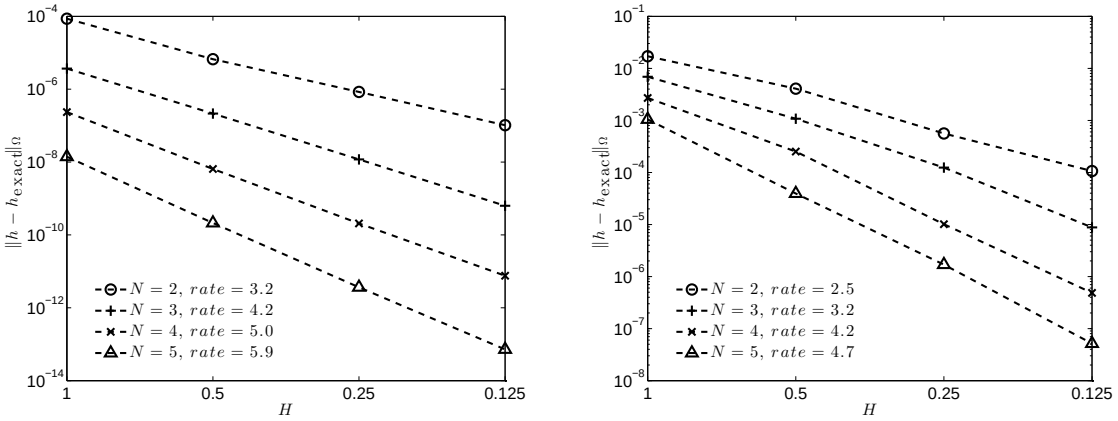
$$h=1, \qquad u=-\sin(\theta)u_\theta, \qquad v=\cos(\theta)u_\theta, \tag{6.1}$$

where azimuthal velocity u_θ and bathymetry B are given by $u_\theta=\frac{1}{75}\left(-r+\frac{16}{r}\right)$ and $B=\frac{1}{75^2}\left(\frac{r^2}{2}-32\log(r)-\frac{128}{r^2}\right)$. Here, $\theta=\tan^{-1}(\frac{y}{x})$ and $r=\sqrt{x^2+y^2}$. Since the solution corresponds to a steady state, the simulations are started with the exact solution as the initial



(a) (b) (c) (d)

Figure 4: The sequence of meshes used to perform convergence analysis of rotating Couette flow between two concentric cylinders.



(a) Couette flow (b) Translating vortex

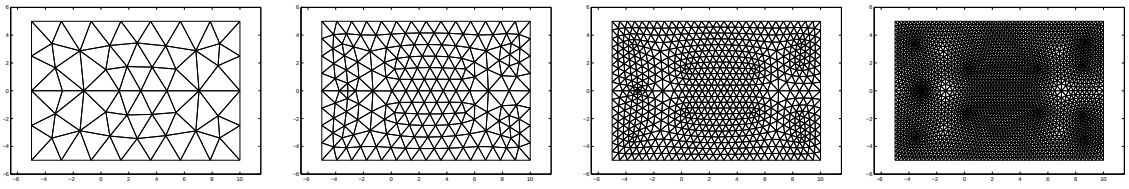
Figure 5: Plot of L^2 error in fluid height vs H/H_0 for smooth solutions, with polynomial orders N=2,3,4, and 5.

conditions and run for a long time ($t=10s$) to compute the spatial errors. We observe L^2-errors decaying like $\mathcal{O}(H^{N+1/2})$ (see Fig. 5). The order of convergence for each polynomial order is computed by determining a polynomial that fits the error in the least square sense.

6.2 Accuracy test for smooth solution: Translating vortex

We use this problem to test the accuracy of the time dependent solutions. An isentropic vortex translates in space with a constant speed and satisfies the two dimensional Euler equations [13, p.209]. By replacing the density (ρ) with the fluid height (h), and choosing the gas constant $\gamma=2$ and gravity $g=2$, an analytical solution for the shallow water equations is obtained

$$h=1-\frac{\beta^2}{32\pi^2}e^{2(1-r^2)},\quad u=1-\beta e^{1-r^2}\frac{y-y_0}{2\pi},\quad v=\beta e^{1-r^2}\frac{x-t-x_0}{2\pi}, \tag{6.2}$$



(a) (b) (c) (d)

Figure 6: The sequence of meshes used to perform convergence analysis for a translating vortex in a rectangular domain.

where $r=\sqrt{(x-t-x_0)^2+(y-y_0)^2}$ and the bathymetry is a constant. A rectangular domain $[-5,10]\otimes[-6,6]\in\mathbb{R}^2$ is chosen as the domain for this problem and both the initial conditions and Dirichlet boundary conditions are set to the exact solution. The sequence of meshes used are given in the Fig. 6. Again we observe that L^2-error converges like $\mathcal{O}(H^{N+1/2})$ (see Fig. 5) as expected for hyperbolic PDEs [14].

6.3 Parabolic bowl

We use this test case [7,24] to study the effect of the positive preserving limiter on solution accuracy. The bathymetry is given by a parabola, $b(x,y)=\alpha r^2$, where $r=\sqrt{x^2+y^2}$. The exact solution for fluid height is non zero for $r<\sqrt{(X+Y\cos\omega t)/\alpha(X^2-Y^2)}$, where $X>0$, $|Y|<X$ and $\omega^2=8g\alpha$. The nonzero exact solution is given by

$$h(x,y,t)=\frac{1}{X+Y\cos(\omega t)}+\alpha(Y^2-X^2)\frac{r^2}{(X+Y\cos(\omega t))^2}, \tag{6.3a}$$

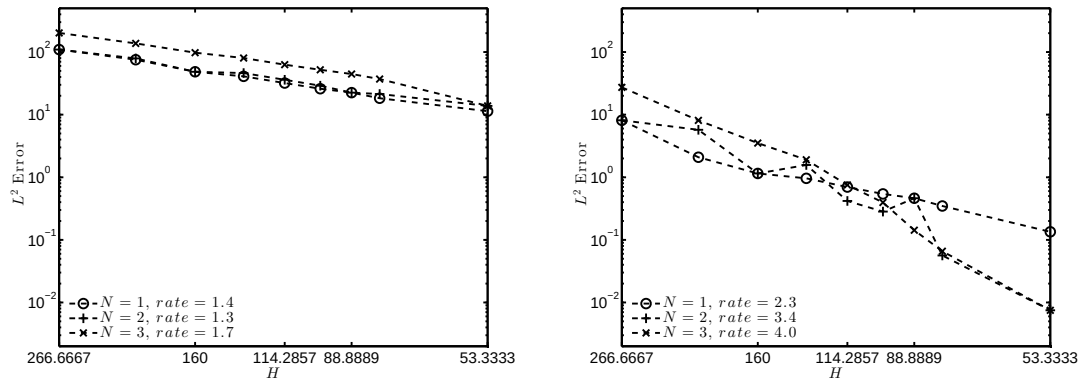
$$u(x,y,t)=-\frac{Y\omega\sin(\omega t)}{X+Y\cos(\omega t)}\frac{x}{2}, \tag{6.3b}$$

$$v(x,y,t)=-\frac{Y\omega\sin(\omega t)}{X+Y\cos(\omega t)}\frac{y}{2}. \tag{6.3c}$$

Here, the constants are $\alpha=1.6\times10^{-7}m^-$, $X=1m^-$, and $Y=-0.41884m^-$. The solutions are obtained in the square domain with side length $8000m$ centered at the origin. The solution is continuous, but not continuously differentiable. Fig. 7 demonstrates the $\mathcal{O}(H^{1.5})$ behavior of the global error in fluid height for approximations with polynomials of order $N=1,2$, and 3, while the local errors computed in regions far from the wet/dry front behave like $\mathcal{O}(H^{N+1})$. Fig. 8 indicates that the errors are localized near the wet/dry front and further mesh refinements reduce these local errors. Here m, s refer to meters and seconds respectively.

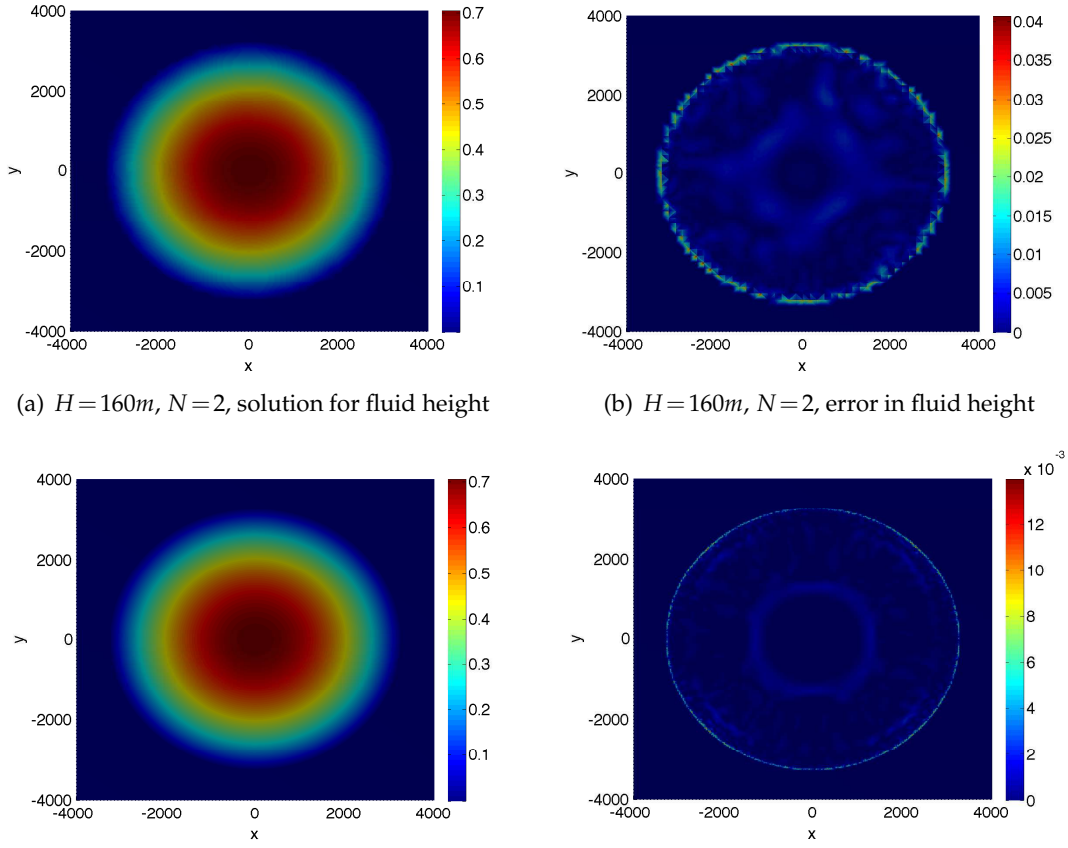
6.4 Positivity preserving test: Rarefaction wave

We use this test case introduced in [7] to demonstrate the effectiveness of positive preserving limiter. We consider a rectangular domain of $50m\times40m$, with a flat bottom. The



(a) Global error in fluid height (b) Local error in fluid height

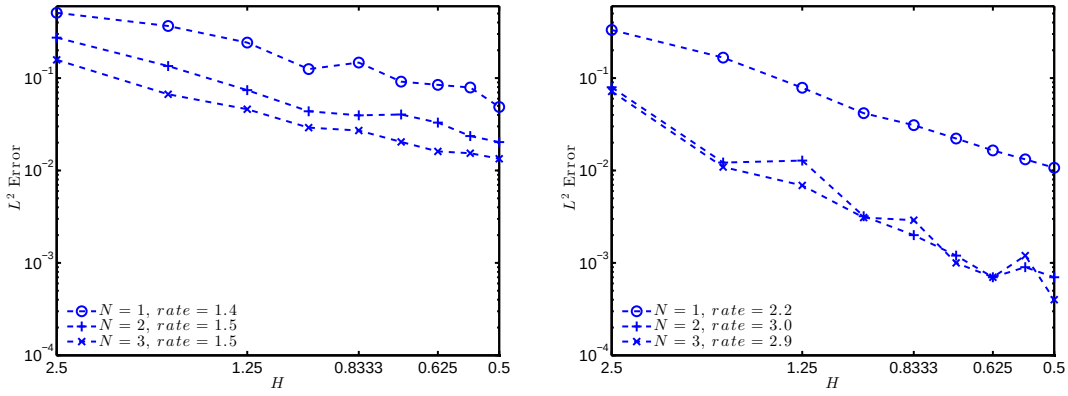
Figure 7: Parabolic bowl test case: (a) global L^2 error in fluid height (b) L^2 error in fluid height for region $r<500m$.



(a) $H=160m$, $N=2$, solution for fluid height (b) $H=160m$, $N=2$, error in fluid height

(c) $H=40m$, $N=2$, solution for fluid height (d) $H=40m$, $N=2$, error in fluid height

Figure 8: Parabolic bowl test case: example numerical results at time $t=T/2$.



(a) Global error in fluid height (b) Local error in fluid height

Figure 9: Rarefaction wave: global and local L^2 errors in the solution for fluid height.

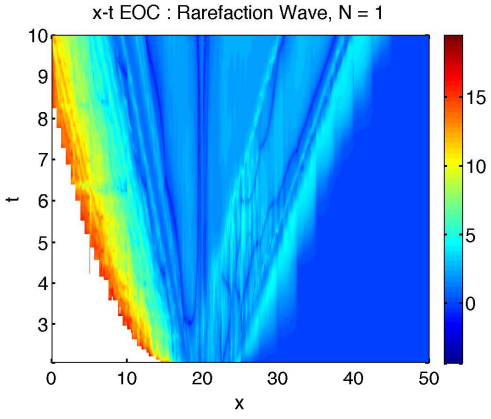
analytical solution depends on $\xi = \frac{(x-20)}{t}$, and is given by

$$(h,u,v) = \begin{cases} (h_0,0,0) & \text{if } \xi < -\sqrt{gh_0}, \\ (0,0,0) & \text{if } \xi > 2\sqrt{gh_0}, \\ (\frac{1}{9g}(\xi - 2\sqrt{gh_0})^2, \frac{2}{3}(\xi + \sqrt{gh_0}), 0) & \text{otherwise.} \end{cases} \tag{6.4}$$

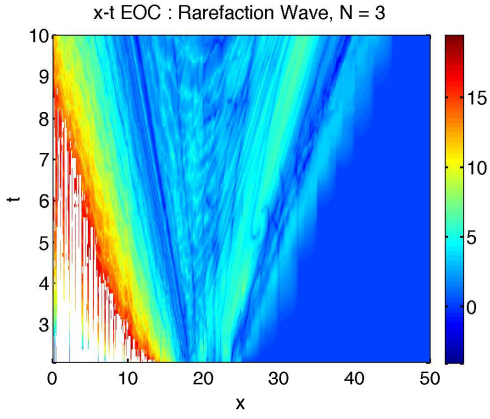
We consider $h_0 = 1$ and $g = 1$. The simulations are run until final time $T = 10s$. A CFL number of 0.03 is used for these simulations in order to minimize transient error and study only the spatial accuracy. The analytical solution at time $t = 2s$ is used as the initial condition so that the solution is in $C^0(\Omega)$. The L^2 errors in the fluid height are presented in Fig. 9. The TVB limiter is not applied for this test case since the solution is smooth enough for a stable computation. We observe that global L^2 error in fluid height is $\mathcal{O}(H^{1.4})$, $\mathcal{O}(H^{1.5})$, and $\mathcal{O}(H^{1.5})$ for the polynomial approximations N=1,2,and 3 respectively.

Since the solution is projected onto linear polynomials, we can expect the global error to behave like $\mathcal{O}(H^{1+1/2})$, which is observed in the estimated order of convergence. To quantify the effect of the positive preserving limiter on the solution accuracy, the L^2 errors are measured for the region, $x \in [15,35]$, where the solution is continuously differentiable. The estimated convergence are $\mathcal{O}(H^{2.2})$, $\mathcal{O}(H^{3.0})$, and $\mathcal{O}(H^{2.9})$ for the polynomials $N =$ 1,2,and 3 respectively (see Fig. 9). The fluid height is a quadratic polynomial in space and rational polynomial in time, hence increasing the spatial polynomial interpolation beyond $N = 2$ does not improve the spatial accuracy which explains the estimated order of convergence for $N = 3$.

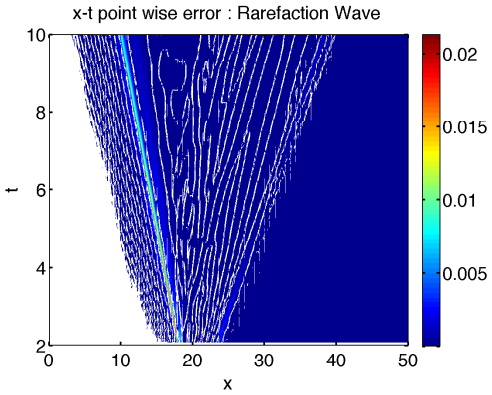
There is an increase in error for further refinements. This is due to large oscillations in the numerical solutions, and the slope limiter should be applied to control these oscillations. The point wise EOCs are plotted in the Fig. 10. It can be observed that the pollution



(a) $N=1$



(b) $N=3$



(c) Point wise error in fluid height, $N=2$ and $H=0.714$

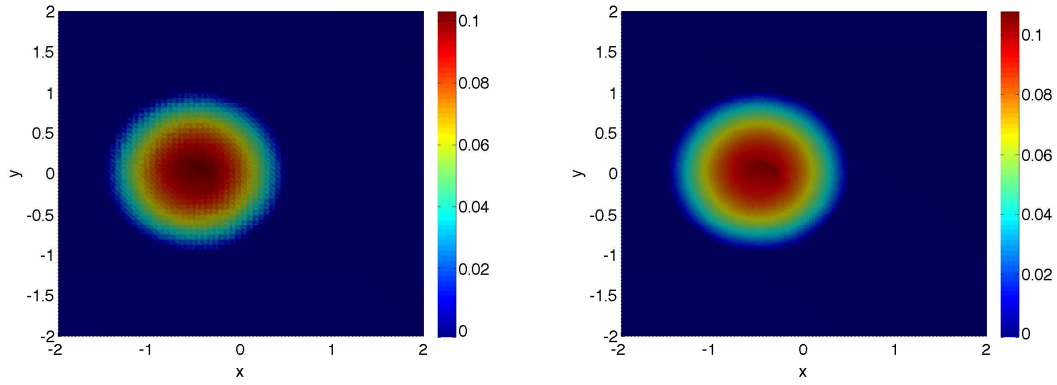
Figure 10: Rarefaction wave: Empirical order of convergence for polynomial orders 1 and 3, and point wise error in fluid height for polynomial order 2.

of the solution accuracy is localized to an area of size $\mathcal{O}(T)$, which is the distance traveled by the rarefaction wave in time T. The white regions in the plots indicate the error is zero and the light blue color indicate that the error is constant due to the positivity preserving limiter in the dry regions.

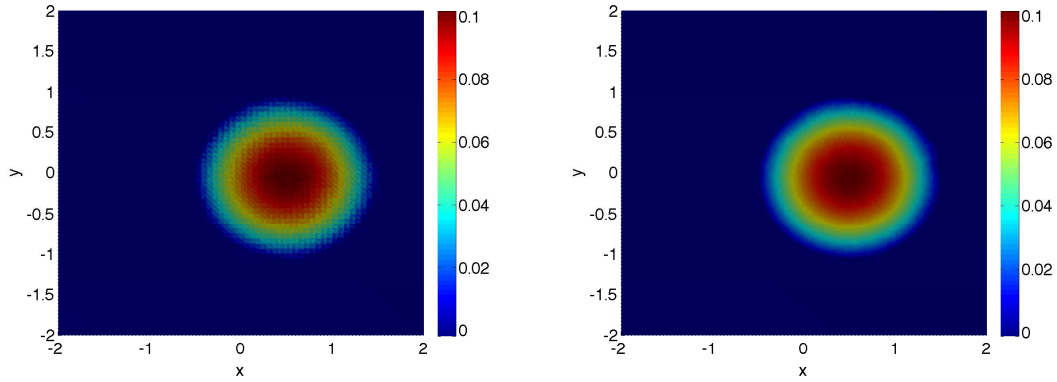
6.5 Limiter test: Two dimensional oscillating lake

We use this test case proposed in [9,27] to test the effectiveness of the positivity preserving limiter and the modified TVB limiter. We consider a rectangular domain $[-2,2]\otimes[-2,2]$ with a parabolic bottom topography given by

$$B(x,y)=h_0\frac{x^2+y^2}{a^2}, \tag{6.5}$$



(a) $H=0.0625$, $N=4$, final time$=T/2$ (b) $H=0.03125$, $N=2$, final time$=T/2$



(c) $H=0.0625$, $N=4$, final time$=T$ (d) $H=0.03125$, $N=2$, final time$=T$

Figure 11: Oscillating lake 2D: example numerical solutions for fluid height.

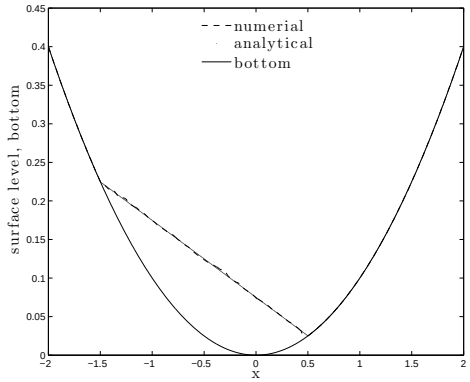
where h_0 and a are specified constants. The analytical solution is given by

$$h(x,y,t)=\max\left(0,\frac{\sigma h_0}{a^2}(2x\cos(\omega t)+2y\sin(\omega t)-\sigma)+h_0-b\right), \tag{6.6a}$$

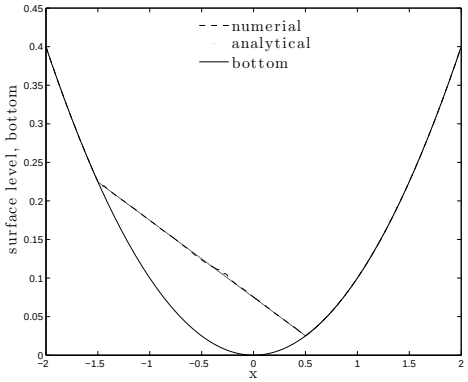
$$u(x,y,t)=-\sigma\omega\sin(\omega t), \qquad v(x,y,t)=\sigma\omega\cos(\omega t), \tag{6.6b}$$

with frequency $\omega=\sqrt{2gh_0}/a$ and time period $T=2\pi/\omega$.

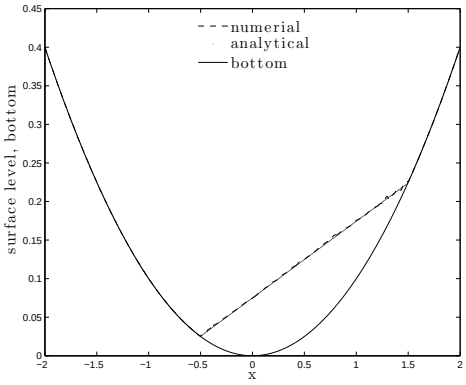
The constants considered are $a=1$, $\sigma=0.5$, and $h_0=0.1$. The initial conditions are defined by Eq. (6.6) with $t=0$. Reflecting boundary conditions are used for all the boundaries. Simulations are run until time T with various uniform meshes ($H=0.25$, 0.125, 0.0625, and 0.03125). The numerical solutions for $H=0.0625$, and 0.03125, with polynomials of order 3 and 2 respectively are plotted in the Fig. 11. The solutions on the line $y=0$ for these simulations are compared with analytical solutions in Fig. 12. For a fine mesh, the numerical solutions are in good agreement with the analytical solutions. The



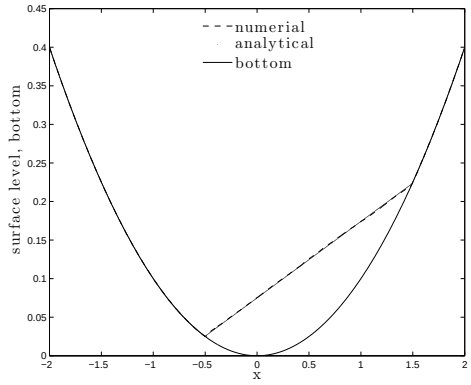
(a) $H=0.0625$, $N=4$, final time $t=T/2$



(b) $H=0.03125$, $N=2$, final time $t=T/2$

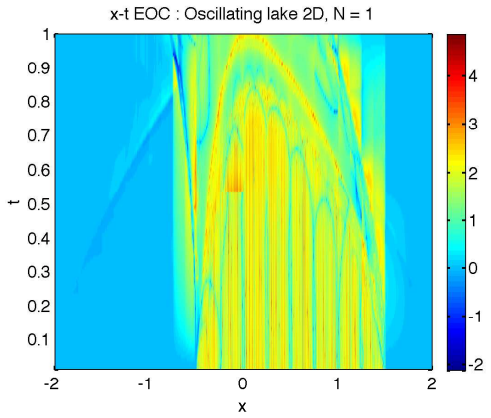


(c) $H=0.0625$, $N=4$, final time $t=T$

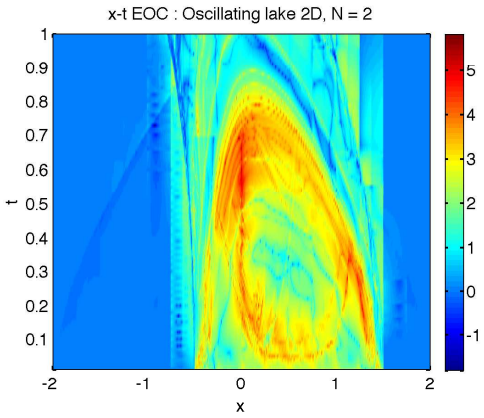


(d) $H=0.03125$, $N=2$, final time $t=T$

Figure 12: Oscillating lake 2D: Comparison of numerical solution with analytical solution along the line $y=0$.



(a) $N=1$



(b) $N=2$

Figure 13: x-t empirical order of convergence of the numerical solution for the fluid height. The errors are computed along the line segment $y=0$ for $t\in[0,1]$.

wet-dry front pollutes the accuracy of the solution. To study the convergence properties of spatial errors, the point wise errors are computed at very small time $t \in [0,1]$. Fig. 13 shows the improvement of point wise EOCs in the solution for $N=2$, compared to $N=1$. These point wise errors are computed along the line $y=0$.

6.6 Summary of the test cases

We summarize all the test cases in Table 1. For each test case, we describe the regularity of the solution, the limiters used for the simulation, global L^2 error estimates, and local L^2 estimates for the solutions with insufficient regularity.

Table 1: Summary of the test cases. For each test case in column I, column II indicates the number of dimensions and regularity of the analytical solution, column III indicates the limiters used if any, column IV and V provide the observed convergence rates for errors in fluid height. The local errors are measured in regions far from the wave front.

Test case	Solution regularity	Limiters	Global error in fluid height	Local error in fluid height
Couette flow	2D, $C^\infty(\Omega)$	-	$\mathcal{O}(H^{N+1/2})$	$\mathcal{O}(H^{N+1/2})$
Isentropic vortex	2D, $C^\infty(\Omega,T)$	-	$\mathcal{O}(H^{N+1/2})$	$\mathcal{O}(H^{N+1/2})$
Parabolic bowl	2D, $C^0(\Omega,T)$	PP	$\mathcal{O}(H^{1+1/2})$	$\mathcal{O}(H^{N+1/2})$
Rarefaction wave	1D, $C^0(\Omega,T)$	PP	$\mathcal{O}(H^{1+1/2})$	$\mathcal{O}(H^{N+1})$
Oscillating lake	2D, $C^0(\Omega,T)$	PP, TVB	-	-

We demonstrated that the rate of decay in the error matches the predicted rate for sufficiently smooth solutions, while accuracy is lost for the problems with insufficient regularity. For these problems, we illustrated that the low EOCs are confined to regions of wave front. To demonstrate the impact of the limiters on the solution accuracy, we estimated the point-wise empirical order of convergence for the rarefaction wave test case and observed that the pollution in the accuracy is localized to the regions the wave front. The localized errors can be controlled by adaptively refining the meshes near the irregularities in the solution, however the mesh refinement techniques are beyond the focus of this paper.

7 GPU acceleration

A majority of the required computations are performed on an element-local way, with a weak coupling between the neighboring elements. This results in locality of the memory access. In addition, the high order nature of the discontinuous Galerkin methods, make them require more computations per degree of freedom, increasing the overall computational intensity. Both of these features make their computations on GPUs more attractive. This Section describes a mapping of the nodal DG discretization onto the wide SIMD model of GPUs using OpenCL. See [15, 25] for a detailed description of OpenCL implementation of DG for electromagnetic applications.

There are three major computations; volume integration, surface integration, and time step update. OpenCL kernels `VolumeKernel`, `SurfaceKernel`, and `UpdateKernel` perform these computations respectively.

$$\underbrace{\frac{dQ_H}{dt}}_{\texttt{UpdateKernel}} = \underbrace{\mathcal{N}(Q_H)}_{\texttt{VolumeKernel}} \underbrace{+\mathcal{S}(Q_H^{g,+},Q_H^{g,-})}_{\texttt{SurfaceKernel}}. \tag{7.1}$$

An OpenCL work group computes the integrals of one or more elements, while one work item computes the contribution from each integration node in these kernels. For each element, contributions from volume and surface integrals are represented as matrix-vector products of a dense rectangular matrix and a vector of field values or fluxes. Each work item in a work group computes one entry of the matrix-vector product for every field to avoid the memory conflicts. Here, we explain the implementation and discuss the performance tuning of these kernels.

Table 2: Notation.

Symbol		Definition
`Nfaces`	:	Number of interfaces per triangle (3)
`N`	:	Polynomial order
`Np`	:	Number of interpolation points `(N+1)*(N+2)/2`
`Nfp`	:	Number of interpolation points on an interface `N+1`
`Ncub`	:	Number of cubature nodes on a triangle
`Ngauss`	:	Number of Gauss integration nodes on a line segment

7.1 Volume kernel

The contributions from volume integrals ($\mathcal{N}(Q_H)$) are computed in this kernel. $\mathcal{N}(Q_H)$ is a vector of length `Np` per field for each element D^k. $\mathcal{N}(Q_H)_i$ represents i^{th} entry of the vector. The computations are

$$\mathcal{N}(Q_H) = Pr \times cF1 + Ps \times cF2 + P \times cS, \tag{7.2}$$

where $cF1$, $cF2$ are numerical flux components in r-, s-directions and cS is a vector of source terms at the cubature integration nodes. P, Pr and Ps are projection matrices that are pre multiplied with cubature integration weights and depend only on the reference triangle.

- **Interpolation to cubature nodes:** Each work item computes all the field values (cQ) at a cubature node. This involves three matrix vector products, all with the same interpolation matrix. The resulting field values are stored in register memory. `Ncub` work items are assigned for this operation.

- **Volume flux evaluation:** Each work item computes all the volume fluxes ($cF1$ and $cF2$) and source terms (cS) at a cubature node using the cubature field values stored in register memory, and stores them in a shared memory array for the fluxes. `Ncub` work items are assigned for this operation.

- **Projection to interpolation nodes:** Each work item computes the contribution from volume integrals to time derivatives at an interpolation node. This involves eight matrix vector products (three for $Pr\times cF1$, three for $Ps\times cF2$ and two for $P\times cS$) with three projection matrices (Pr, Ps and P) and three flux vectors that were stored in shared memory in flux evaluation. `Np` work items are assigned for this operation.

To accommodate the number of work items required for all the computations, a maximum of `Np` and `Ncub` work items are requested per element. This requires total number of `Kv*max(Ncub, Np)` work items per work group, where `Kv` is the number of elements processed by a work group.

7.2 Surface kernel

The contributions from surface integrals ($\mathcal{S}(Q_H^{g,+},Q_H^{g,-})$) are computed in this kernel. $\mathcal{S}(Q_H^{g,+},Q_H^{g,-})$ is a vector of length `Np` per field for each element D^k. $\mathcal{S}(Q_H^{g,+},Q_H^{g,-})_i$ represents i^{th} entry of the vector. The computations are

$$\mathcal{S}(Q_H^{g,+},Q_H^{g,-}) = -L^g \times F_n^{*,g}, \tag{7.3}$$

where L^g is a projection/lifting operator that projects the contribution from the surface integrals to the interpolation nodes. $F_n^{*,g}$ is a vector of stable numerical fluxes at Gauss quadrature nodes. Q^g, the vector of field values at the Gauss quadrature nodes is computed a priori to this kernel.

- **Numerical flux evaluation:** Each work item computes all the numerical fluxes at a Gauss quadrature node ($F_n^{*,g}$). These numerical fluxes are stored in shared memory for further computations. `Ngauss*Nfaces` work items are assigned for this operation.

- **Lifting the flux to interpolation nodes:** Each work item computes the surface integral contribution at an interpolation node. This involves three matrix vector products with lifting operator (L^g). As discussed in volume kernel, the lifting operator is copied to shared memory. `Np` work items are assigned for this operation.

To accommodate the number of work items required for all the computations, a maximum of `Np` and `Ngauss*Nfaces` work items are used for the computations per element. This requires total number of `Ks*max(Ngauss*Nfaces, Np)` work items per work group, where `Ks` is the number of elements processed by a work group.

7.3 Update kernel

The field values at interpolation nodes (Q_H) and Gauss integration nodes (Q_H^g) for each element are updated in this kernel. The update kernel is divided in to two parts.

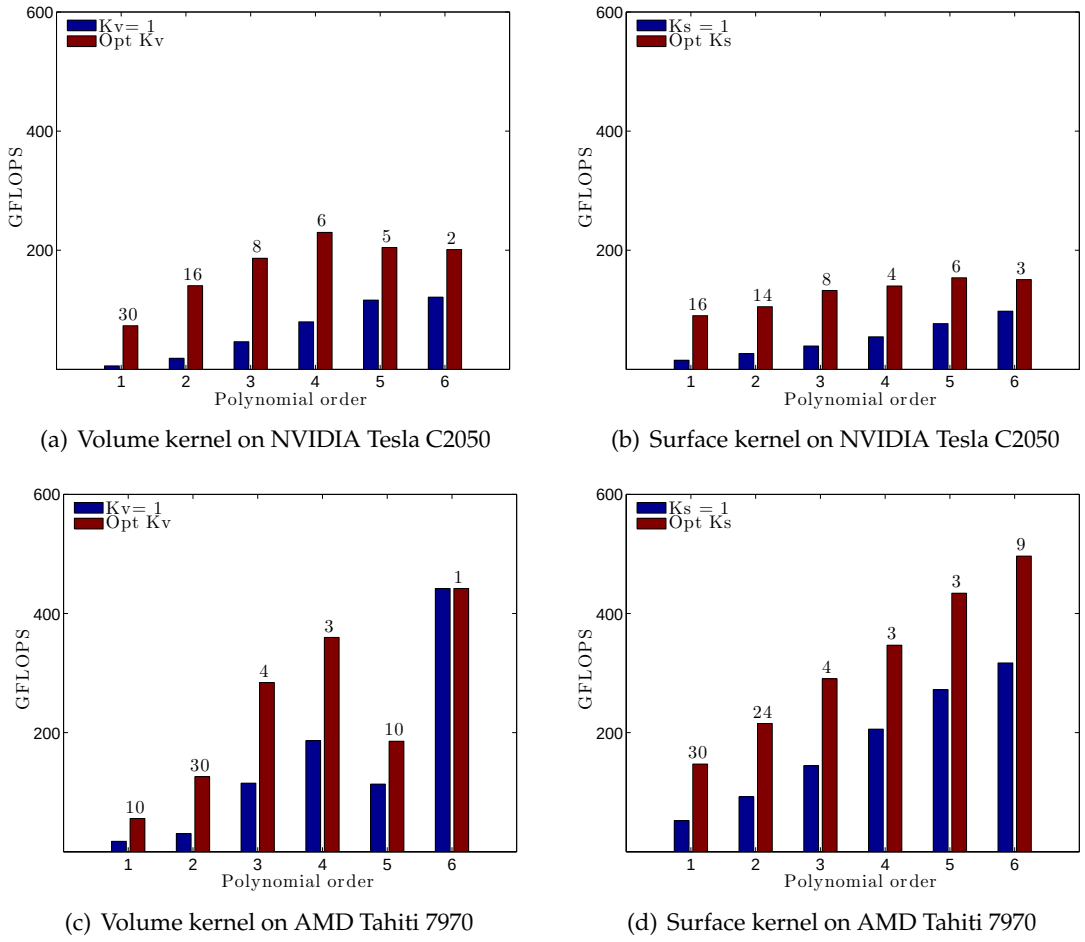
- **Update interpolation nodes:** Each work item updates all the field values at an interpolation node. `Np` work items are assigned for this operation.
- **Interpolate to Gauss quadrature nodes:** Each work item computes all the field values at a Gauss quadrature node. This involves three matrix vector products per element, all with a Gauss interpolation matrix. `Ngauss*Nfaces` work items are assigned for this operation.

To accommodate the number of work items required for all the computations `Np` work items are requested per element. This requires total number of `Ku*Np` work items per work group. Interpolation to Gauss quadrature nodes is implemented in a separate kernel that requires `Ngauss*Nfaces*Ku` work items per work group. Here, `Ku` is the number of elements processed by a work group.

7.4 Kernel tuning

The performance of the above discussed kernels is very sensitive to the hardware, tuning parameters, optimal usage of shared/local memory. Here, we discuss some of the optimization techniques we adopted and resulted in significant performance improvement.

- **Coalescing:** The nodal values corresponding to each element, projection operator and interpolation operator are accessed contiguously from memory to maximize memory bus utilization.
- **Padding:** The vector of nodal values for each element is padded with a factor of 4 to make sure that the array accesses are aligned.
- **Unrolling:** Loops are unrolled to reduce the number of instructions to be executed, leading to hiding the latencies in reading the data from memory.
- **Multiple elements per work group:** Multiple elements are processed by each work group to increase the occupancy on a given hardware architecture. For low order approximation the speed up is about 5 times compared to using one element processed by a work group. Because of the limited availability of shared memory and the difference in computation patterns, the optimal number of elements per work group varies for each kernel. The optimal parameters depend on the hardware of the GPU to a large extent. For example, we observe a significant difference in the performance on Tesla C2050 (NVIDIA) compared to Radeon 7970 (AMD). We can see from Fig. 14 that the performance improvement by using multiple elements processed by each work group is much more significant for Tesla C2050 compared



(a) Volume kernel on NVIDIA Tesla C2050

(b) Surface kernel on NVIDIA Tesla C2050

(c) Volume kernel on AMD Tahiti 7970

(d) Surface kernel on AMD Tahiti 7970

Figure 14: Single precision GFLOPS of native OpenCL volume kernel and surface kernel vs polynomial order. OpenCL 1.1 is used on NVIDIA GPUs and OpenCL 1.2 is used on AMD GPUs. ECC memory checking is turned off for these experiments. The numbers on each bar represent the number of elements processed by a work group when the optimal performance is observed.

to that of Tahiti 7970. Kv and Ks are the number of elements processed by a work group in volume and surface kernels respectively.

- **Shared/local memory:** The nodal values on each element and/or projection and interpolations operators are stored in shared memory in order to reuse the data efficiently within a kernel. Since shared memory is limited in GPUs, using a large shared memory in a kernel will reduce the number of concurrent active work groups, but since the operators do not vary across the elements, they have to be stored only once for all the elements that are processed by a work group.

8 OCCA unified multi-threading approach

The computational kernels described in previous section were written in an open standard multi-threading model OpenCL. Vendors like NVIDIA, AMD, and Intel provide the implementation of OpenCL for their hardware. Hence, performance of these kernels depend a lot on the vendor implementation of OpenCL. Conversely, there may be an efficient alternative programming model for a given hardware architecture, for example, CUDA for NVIDIA GPUs and OpenMP for Intel and AMD CPUs.

It is tedious and unnecessary for either a programmer or a scientist to redesign an application code to take advantage of another hardware or multi-threading model effectively. To address this to some extent we developed OCCA, a unified approach for multi-threaded programming. Using OCCA, it is possible to write a single code that can be used on several hardware architectures and supporting multi-threading languages. A white paper on OCCA can be found at [19] and the software can be downloaded from [20], which also includes several example codes. There are two major APIs in OCCA, one is the device API that allows us to write kernels that can be cross-compiled with above mentioned programming models, the other is the host API that allows us to compile and execute these kernels from an application code.

8.1 OCCA device API/kernel language

OCCA abstracts kernel languages from several multi-threading approaches and optimization techniques. OCCA takes GPU approach of splitting work by work-groups and work-items for parallelism. Using this abstraction layer, we are able to write kernels with several keywords that can be translated to language specific words. These keywords are represented using macros. These macros are defined for each multi-threading language. For detailed description on the kernel language specifications, we refer the readers to OCCA white paper [19]. The kernels written in OCCA kernel language are compiled at runtime with a user specified compiler.

8.2 OCCA host API

We developed a stand-alone host API to be able to run the kernels written in OCCA kernel language from the application. The host API is written in C++ and has interfaces to programming languages C, C#, Fortran, Matlab, Julia, and Python. This allows OCCA to be portable across several programming languages, hardware architectures, and multi-threading models as illustrated in Fig. 15. There are three major classes in the host API, `device`, `memory`, and `kernel`.

8.3 OCCA device

The `device` class is an abstraction layer between OCCA API and the API from supported models/languages. Using this class, a target platform or vendor and an available device

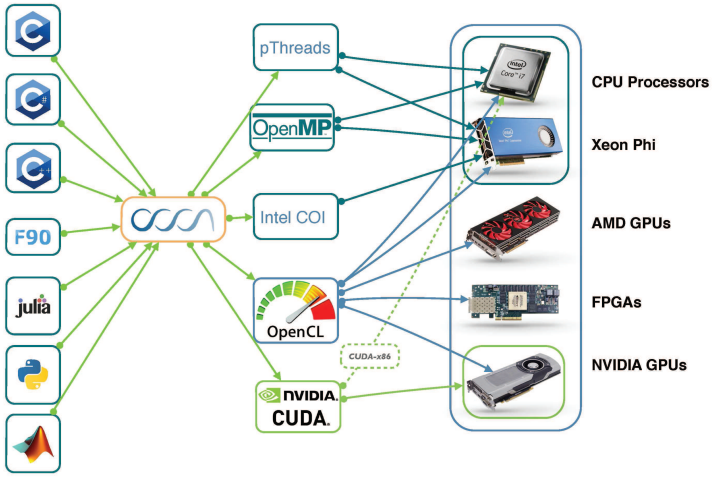


Figure 15: OCCA portability across several programming languages, multi-threading approaches, and hardware architectures. We are currently testing for robustness of IntelCOI and Pthreads back-ends.

from the platform can be chosen at run-time. A `device` is in charge of creating a context and a command queue from the chosen supported device. Asynchronous computations with multiple contexts can be achieved using multiple `devices`. The main purpose of a `device` is to allocate memory and compile kernels for the chosen device.

8.4 OCCA memory

The `memory` class abstracts the different device memory handles and provides some useful information such as device array sizes. Although memory handling in OCCA facilitates host-device communication, the management of reading and writing between host and device, for performance reasons, is still left to the programmer. Having a class dedicated for device memory also allows the kernel class to differentiate and communicate between distinct memory types.

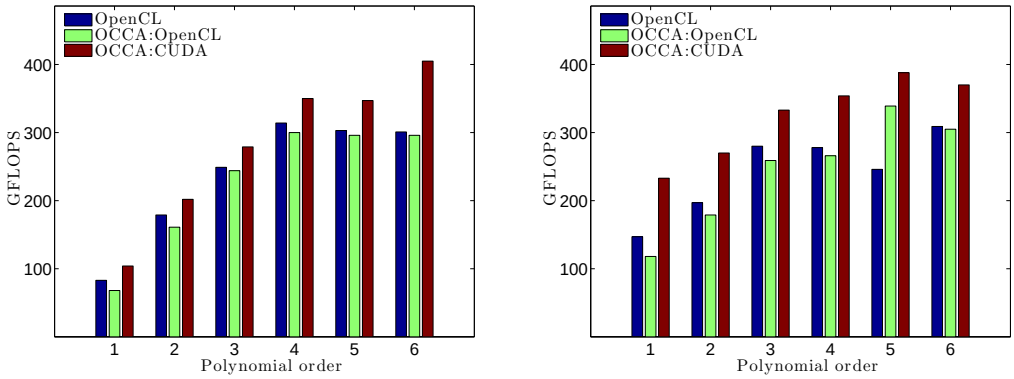
8.5 OCCA kernel

The kernel class unites device function handles with a single interface, whether for a function pointer (OpenMP), `cl_kernel` (OpenCL), or `cuFunction` (CUDA). When using the OpenCL and CUDA kernel handles, passing the arguments through their respective API is simple, but there are discrepancies when comparing to the OpenMP wrapper. For example, OpenCL and CUDA kernels work-items (or threads) have access to work-group (or thread block) and work-item counts implicitly. However, C++ functions only have access to the function scope and global name-space, requiring the work-group and work-item counts to be passed as macros or as an argument to the kernel.

8.6 Performance results

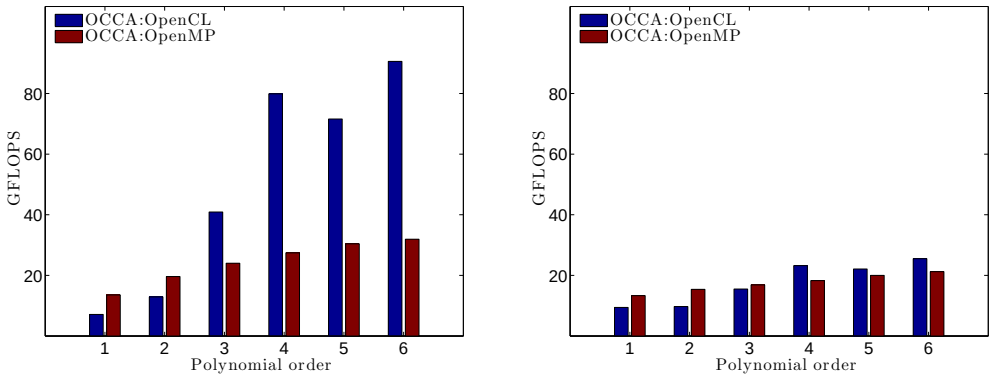
We started with our OpenCL implementation and ported them to OCCA language. In Fig. 16, we compare the performance of the two main kernels, the volume and surface kernels, for each platform together with the original hand-coded OpenCL kernels. The comparisons are done on an NVIDIA Titan GPU with the optimal kernel tunings for each model as shown in Fig. 14. Because CUDA ptx compilers can be optimized to its hardware, the results between CUDA and OpenCL on the Titan are not surprising. However, the focus is that we observe a similar performance between the OCCA kernels compared to our original native OpenCL kernels.

In Fig. 17, we compare the performance of OCCA kernels when they are cross-



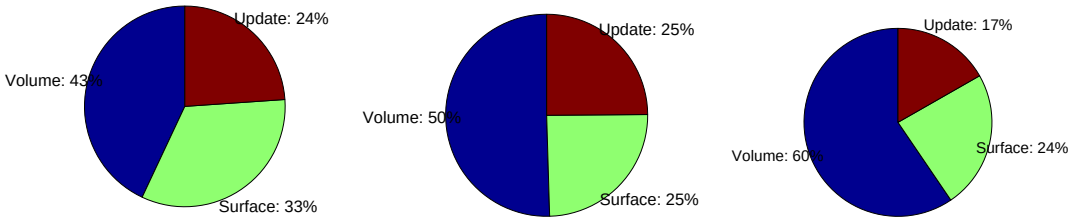
(a) Volume kernel (b) Surface kernel

Figure 16: Single precision GFLOPS of kernels vs polynomial order. A comparison of OpenCL, OCCA:OpenCL, and OCCA:CUDA. Experiments ran on NVIDIA Titan GPU with CUDA-5.5 and OpenCL 1.1. ECC memory checking is turned off for these experiments.



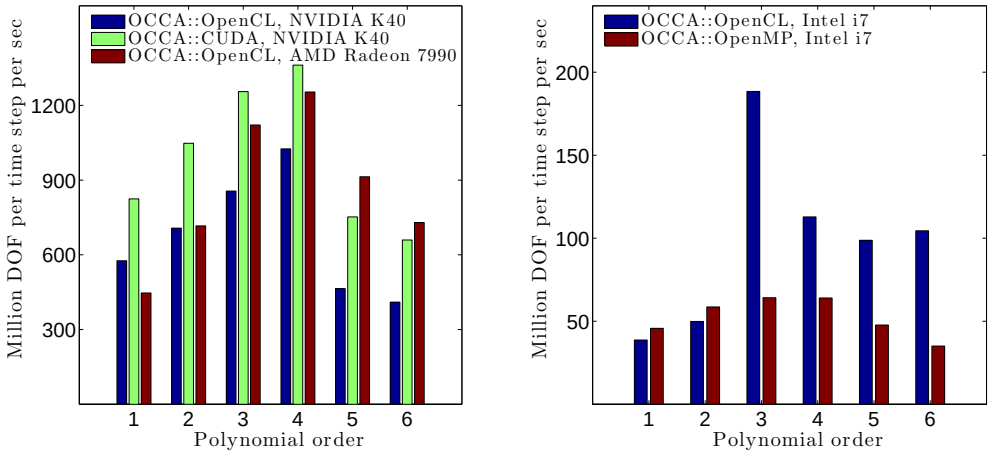
(a) Volume kernel (b) Surface kernel

Figure 17: Single precision GFLOPS of kernels vs polynomial order. A comparison of OCCA:OpenCL and OCCA:OpenMP. Experiments ran on Intel(R) Core(TM) i7-3930K CPU using Intel OpenCL 1.2 and g++ 4.7.



(a) $N=1$ (b) $N=3$ (c) $N=5$

Figure 18: Percentage of time spent by volume, surface, and update kernels for polynomial orders 1,3, and 5. OCCA:CUDA is used on NVIDIA K40 with CUDA-6.0 is used for these experiments.



(a) Time-step efficiency on GPUs (b) Time-step efficiency on CPUs

Figure 19: Single precision efficiency of the time stepping on NVIDIA & AMD GPUs and Intel CPU. CUDA-6.0, OpenCL-2.0, are used on GPUs, while Intel OpenCL and g++-4.7 are used on CPU for these computations. Efficiency is measured by millions of DOF (total no. of nodes $\times$ no. of fields) processed per time step per second. ECC memory checking is turned off for on GPUs for these experiments.

compiled with OpenCL and OpenMP on CPU. We realize that Intel OpenCL on Intel CPU does a very good job of vectorization when the number of work items per work group is a multiple of eight. To take advantage of this, we pad the number of work items to be a multiple of eight, when the number of work items is not a multiple of eight. This results in a significant performance improvement for the volume kernel, while the improvement for surface kernel is insignificant. This is because of the poor vectorization of the surface kernel due to conditional statements.

To study the performance of the overall solver, we used translating vortex test problem on a uniform mesh with 119400 triangles with only one level for the multi-rate time stepping scheme. This test case does not require application of any limiter for the numerical stability. Fig. 18 shows the percentage of time spent by the kernels for polynomial orders 1,3, and 5. The percentage of time spent on volume kernel increases with the polynomial order. In Fig. 19, we compare the overall efficiency of the solver on NVIDIA &

AMD GPUs and Intel CPU, using supported multi-threading approaches. For efficiency, we consider the overall number of degrees of freedom processed by the solver per time step per second. We observe that an AMD Radeon 7990 GPU tends to perform better than an NVIDIA K40 GPU when the computational intensity is higher, i.e., larger N in this context.

9 Conclusions and future work

A GPU accelerated discontinuous Galerkin algorithm for shallow water equations is presented. This algorithm is further accelerated with a multi-rate time stepping scheme. We presented a modified TVB limiter and a positivity preserving limiter for high order approximations. These algorithms are tested for accuracy, robustness and efficiency using several standard test cases in the literature. We compared the performance of OCCA kernels when they are cross-compiled with threading models OpenCL, CUDA, and OpenMP at runtime. Our future work will include developing full three-dimensional incompressible Navier-Stokes models using high order discontinuous Galerkin methods for better understanding of the tsunamis near coastal regions and accelerating such simulations through many core hardware architectures using OCCA.

Acknowledgments

The authors gratefully acknowledge travel grants from Pan-American Advanced Studies Institute, grant from DOE and ANL (ANL Subcontract No. 1F-32301 on DOE grant No. DE-AC02-06CH11357), grant from ONR (Award No. N00014-13-1-0873), fellowships from Ken Kennedy Institute of technology at Rice University and support from Shell (Shell Agreement No. PT22584), NVIDIA, and AMD. The authors also acknowledge Dr. Jesse Chan for fruitful discussions during the preparation of this manuscript.

References

[1] V. Aizinger and C. Dawson, A discontinuous Galerkin method for two-dimensional flow and transport in shallow water, Advances in Water Resources, 25 (2002), pp. 67–84.
[2] M. Berger, D. George, R. LeVeque, and K. Mandli, The GeoClaw software for depth-averaged flows with adaptive refinement, Advances in Water Resources, 34 (2011), pp. 1195–1206.
[3] S. Bunya, E. Kubatko, J. Westerink, and C. Dawson, A wetting and drying treatment for the Runge-Kutta discontinuous Galerkin solution to the shallow water equations, Computer Methods in Applied Mechanics and Engineering, 198 (2009), pp. 1548–1562.
[4] V. Casulli, Semi-implicit finite difference methods for the two-dimensional shallow water equations, Journal of Computational Physics, 86 (1990), pp. 56–74.
[5] B. Cockburn and C.-W. Shu, The Runge-Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems, Journal of Computational Physics, 141 (1998), pp. 199–224.

[6] R. Cools, Monomial cubature rules since "Stroud": a compilation-part 2, (1999).
[7] A. Ern, S. Piperno, and K. Djadel, A well-balanced Runge-Kutta discontinuous Galerkin method for the shallow-water equations with flooding and drying, International journal for numerical methods in fluids, 58 (2008), pp. 1–25.
[8] C. Eskilsson and S. Sherwin, A triangular spectral/hp discontinuous Galerkin method for modelling 2D shallow water equations, International Journal for Numerical Methods in Fluids, 45 (2004), pp. 605–623.
[9] J. M. Gallardo, C. Parés, and M. Castro, On a well-balanced high-order finite volume scheme for shallow water equations with topography and dry areas, Journal of Computational Physics, 227 (2007), pp. 574–601.
[10] C. W. Gear and D. Wells, Multirate linear multistep methods, BIT Numerical Mathematics, 24 (1984), pp. 484–502.
[11] F. Giraldo and T. Warburton, A high-order triangular discontinuous Galerkin oceanic shallow water model, International journal for numerical methods in fluids, 56 (2008), pp. 899–925.
[12] N. Godel, S. Schomann, T. Warburton, and M. Clemens, GPU accelerated Adams-Bashforth multirate discontinuous Galerkin FEM simulation of high-frequency electromagnetic fields, Magnetics, IEEE Transactions on, 46 (2010), pp. 2735–2738.
[13] J. Hesthaven and T. Warburton, Nodal discontinuous Galerkin methods: algorithms, analysis, and applications, vol. 54, Springer Verlag, 2008.
[14] C. Johnson and J. Pitkäranta, An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation., Math. Comput., 46 (1986), pp. 1–26.
[15] A. Klockner, High-performance high-order simulation of wave and plasma phenomena, PhD thesis, Brown University, 2010.
[16] A. Klockner, T. Warburton, J. Bridge, and J. Hesthaven, Nodal discontinuous Galerkin methods on graphics processors, Journal of Computational Physics, 228 (2009), pp. 7863–7882.
[17] E. Kubatko, J. Westerink, and C. Dawson, hp Discontinuous Galerkin methods for advection dominated problems in shallow water flow, Computer Methods in Applied Mechanics and Engineering, 196 (2006), pp. 437–451.
[18] R. LeVeque, D. George, and M. Berger, Tsunami modelling with adaptively refined finite volume methods, Acta Numerica, 20 (2011), pp. 211–289.
[19] D. Medina and T. Warburton, OCCA: A unified approach to multi-threading languages, white paper, (2014).
[20] D. Medina, L. Wilcox, and T. Warburton, OCCA 2.0: Extensible multi-threading programming API, 2014.
[21] D. Merrill, Finite difference and pseudospectral methods applied to the shallow water equations in spherical coordinates, PhD thesis, University of Colorado, 1997.
[22] I. Navon, A review of finite-element methods for solving the shallow-water equations, B. Schrefl er and OC Zienkiewicz, ed., Computer Modelling in Ocean Engineering, Balkeman, Rotterdam, (1988), pp. 273–278.
[23] C.-W. Shu, TVB uniformly high-order schemes for conservation laws, Mathematics of Computation, 49 (1987), pp. 105–121.
[24] W. C. Thacker, Some exact solutions to the nonlinear shallow-water wave equations, Journal of Fluid Mechanics, 107 (1981), pp. 499–508.
[25] W. H. Wen-mei, GPU Computing GEMs Jade Edition, Morgan Kaufmann, 2011.
[26] Y. Xing and C. Shu, High order well-balanced finite volume WENO schemes and discontinuous Galerkin methods for a class of hyperbolic systems with source terms, Journal of

Computational Physics, 214 (2006), pp. 567–598.
[27] Y. Xing, X. Zhang, and C. Shu, Positivity-preserving high order well-balanced discontinuous Galerkin methods for the shallow water equations, Advances in Water Resources, 33 (2010), pp. 1476–1493.
[28] X. Zhang and C. Shu, On positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations on rectangular meshes, Journal of Computational Physics, 229 (2010), pp. 8918–8934.