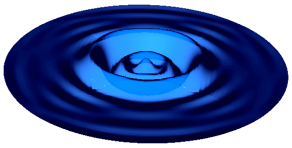
Spectral Element Libraries in Fortran (SELF)

Shallow Water Equations



PCSRI Projects

Joseph Schoonover

Contents

Abstract

1 Background and Motivation

1.1 The Conservative Shallow Water Equations

The inviscid shallow water equations are a system of coupled partial differential equations that is hyperbolic. The Nodal Discontinuous Galerkin Spectral Element Method (NDGSEM) is well suited for hyperbolic systems. It is highly accurate through its use of arbitrarily high order piecewise polynomial representation of the approximated solution and geometry. It is well known for exhibiting minimal numerical dispersion and dissipation errors for linear problems. It additionally allows for a flexible treatment of the geometry through the use of a structured or unstructured arrangement of *spectral elements* that can represent boundaries through high order polynomial interpolants. The discretization yields dense and local matrix-vector multiplies for computing derivatives and only requires nearest neighbor communication to calculate exchanges between elements. This last property makes the NDGSEM trivial to parallelize. For these reasons, the NDGSEM is applied to the shallow water equation systems (1.1), (1.2), and (1.3) to demonstrate its accuracy and scalability for a system of equations that hold many of the important dynamics in geophysics.

1.2 Flavors of the Shallow Water Equations

The nonlinear shallow water equations can be written in either "conservative" (Eqs. 1.1) or "skew-symmetric" (Eqs. 1.2) form :

$$U_t + \nabla \cdot \left(\frac{\vec{U}U}{h+\eta} + g\left(h + \frac{\eta}{2}\right)\eta\hat{x} \right) - fV = g\eta h_x + q_U \tag{1.1a}$$

$$V_t + \nabla \cdot \left(\frac{\vec{U}V}{h+\eta} + g\left(h + \frac{\eta}{2}\right)\eta\hat{y} \right) + fU = g\eta h_y + q_V \tag{1.1b}$$

$$\eta_t + \nabla \cdot \vec{U} = q_\eta \tag{1.1c}$$

$$u_t + \nabla \cdot ((e+p)\hat{x}) - (f+\zeta)v = q_u \tag{1.2a}$$

$$v_t + \nabla \cdot ((e+p)\hat{y}) + (f+\zeta)u = q_v \tag{1.2b}$$

$$p_t + \nabla \cdot ((gh+p)\vec{u}) = q_p \tag{1.2c}$$

For either system, and throughout the rest of this documentation the symbols have the following meaning :

x, y : Cartesian coordinate positions ($Length$)

t : Time

U, V : The x, y components of the the fluid *transport* ($\frac{Length^2}{Time}$)

H : The total fluid thickness, bathymetric plus free surface ($Length$)

h : Resting fluid depth, the fluid bathymetry ($Length$)

g : Local acceleration of gravity ($\frac{Length}{Time^2}$)

f : Coriolis parameter ($Time^{-1}$)

u, v : The x, y components of the *depth average* fluid velocity ($\frac{Length}{Time}$)

e : The kinetic energy of the depth averaged velocity field ($\frac{Length^2}{Time^2}$)

p : Barotropic pressure, the product of gravitational acceleration and free surface anomaly ($\frac{Length^2}{Time^2}$)

ζ : The relative vorticity of the depth averaged velocity field ($Time^{-1}$)

q_ϕ : Any additional non-conservative source for the variable ϕ ($\frac{\phi}{Time}$)

The linear shallow water equations (1.3) are given as

$$u_t + \nabla \cdot (p\hat{x}) - fv = q_u \tag{1.3a}$$

$$v_t + \nabla \cdot (p\hat{y}) + fu = q_v \tag{1.3b}$$

$$p_t + \nabla \cdot (gh\vec{u}) = q_p \tag{1.3c}$$

Any flavor of the shallow water equations arise from a conservation law, and as such, (1.1), (1.2), or (1.3) can be written in the compact form

$$\vec{s} + \nabla \cdot \vec{f} = \vec{q}. \tag{1.4}$$

The solution vector is represented by $\vec{s}$, $\vec{f}$ is a conservative flux tensor, and $\vec{q}$ is a vector of non-conservative source terms. The derivation of the discrete equations is presented through a discretization of (1.4).

In the applications discussed in this document, we are primarily focused on the Conservative Form of the Shallow Water equations. In all of the benchmarks, we use the `dipole` example with a fixed unstructured mesh.

1.3 Nodal Discontinuous Galerkin Spatial Discretization

The NDGSEM discretizes (1.4) in its weak form. The physical space over which (1.4) is solved is denoted by Ω. Solutions to (1.4) are real functions that are piecewise continuous on Ω. This space of functions is denoted $\mathbb{C}_0(\Omega)$, and is defined mathematically

$$\mathbb{C}_0(\Omega) \equiv \left\{ \phi(\vec{x}) \in \mathbb{R} : \int_\Omega \phi^2 d\Omega < \infty \right\}, \tag{1.5}$$

" the set of all functions ϕ that are square integrable over the physical space Ω ".

$\mathbb{C}_0(\Omega)$ has an inner product associated with it, defined by

$$(u, v) = \int_\Omega uv \, d\Omega. \tag{1.6}$$

From the definition of the inner product, the *induced norm* of a function that is in $\mathbb{C}_0(\Omega)$ is defined as

$$||u|| = \sqrt{(u, u)}. \tag{1.7}$$

Similar to vectors, the inner-product (1.6) is a measure of the "common information" between u and v. Two functions are said to be *orthogonal* if

$$(u, v) = 0. \tag{1.8}$$

Any piecewise continuous function can be represented as a linear combination of a *basis functions* that span $\mathbb{C}_0(\Omega)$. The weak form of (1.4) is obtained by taking the inner product of (1.4) with each basis function of $\mathbb{C}_0(\Omega)$,

$$(\vec{s}_t, \phi) + (\nabla \cdot \vec{f}, \phi) = (\vec{s}, \phi), \quad \forall \phi \in \mathbb{C}_0(\Omega) \tag{1.9}$$

In Eqs. (1.1), (1.2), and (1.3), the conservative flux depends on the solution. In (1.9), the divergence of the flux requires that the solution is at least once differentiable. This differentiability constraint is relaxed in the weak form by performing integration by parts, which moves the differentiability criteria onto the basis function (ϕ). Performing integrations by parts gives

$$(\vec{s}_t, \phi) + \oint_{\partial\Omega} \phi \vec{f} \cdot \hat{n} \, dS - (\vec{f}, \nabla\phi) = (\vec{s}, \phi), \quad \forall \phi \in \mathbb{C}_0(\Omega) \tag{1.10}$$

To develop a discrete set of equations, all of $\mathbb{C}_0(\Omega)$ is approximated by piecewise continuous polynomials of degree N, a space we denote $\mathbb{P}^N$. The solution, conservative flux, and the non-conservative source term are approximated by a linear combination of a suitable basis for $\mathbb{P}^N$. Since the solution is piecewise continuous, the domain Ω, is decomposed into a set of non-overlapping elements Ω_l. The integration is broken into integration over each of the elements,

$$\sum_{\kappa=0}^{nEl} \left\{ \int_{\Omega_\kappa} (\vec{s}_t + \vec{q})\phi_\kappa \, d\Omega_\kappa - \int_{\Omega_l} \vec{f} \cdot \nabla\phi_\kappa \, d\Omega_\kappa + \oint_{\partial\Omega^\kappa} \phi_\kappa \vec{f} \cdot \hat{n} \, dA_\kappa = 0 \right\}, \quad \forall \phi_\kappa \in \mathbb{P}^n. \tag{1.11}$$

In the *Discontinuous* Galerkin approximation, the ϕ_l are identically zero outside of the l^{th} element, so that the contributions from each element are decoupled (except for the element boundary integration). This feature of DG is called *compact support*. Again, the solution, flux, and source are approximated by a polynomial of degree N. We choose to write each term as a linear combination of polynomial basis function; here the Lagrange interpolating polynomials are used,

$$\vec{s} \approx I^N[\vec{s}] = \vec{S} = \sum_{i,j=0}^{N} \vec{S}_{i,j} l_i(\xi^1) l_j(\xi^2) \tag{1.12a}$$

$$\vec{q} \approx I^N[\vec{q}] = \vec{Q} = \sum_{i,j=0}^{N} \vec{Q}_{i,j} l_i(\xi^1) l_j(\xi^2) \tag{1.12b}$$

$$\vec{f} \approx I^N[\vec{f}] = \vec{F} = \sum_{i,j=0}^{N} \vec{F}_{i,j} l_i(\xi^1) l_j(\xi^2) \tag{1.12c}$$

The basis functions for $\mathbb{P}^N$ are chosen as each of the Lagrange interpolating polynomials,

$$\phi_{m,n} = l_m(\xi^1) l_n(\xi^2) \tag{1.13}$$

With this, equation (1.11) becomes

$$\int_{\Omega^\xi} (\vec{S}_t^\kappa + \vec{Q}^\kappa)\phi_{m,n}\, d\Omega^\xi - \int_{\Omega^\xi} \vec{F}^\kappa \cdot \nabla_{\vec{\xi}} \phi_{m,n}\, d\Omega^\xi + \oint_{\partial\Omega^\xi} \phi_{m,n} \vec{F}^{\kappa,*} \cdot \hat{n}\, dA^\xi = 0, \quad m,n = 0,1,...,N \tag{1.14}$$

The final step is to replace the integrals in (1.14) with discrete quadrature. For this we use the Legendre-Gauss quadrature, which yields exact integration for each term in (1.14). Additionally, the interpolation nodes are specified as the Legendre-Gauss nodes, which simplifies the integration.

$$\begin{aligned} \left(J_{m,n}\vec{S}_{m,n}\right)_t = &- \left[\sum_{i=0}^{N} \hat{D}_{m,i}^{(\xi^1)} \tilde{F}_{i,n}^{(\xi^1)} + \left(\frac{l_m(1)}{w_m^{(\xi^1)}} \tilde{F}^*(1,\xi_n^2) - \frac{l_m(-1)}{w_m^{(\xi^1)}} \tilde{F}^*(-1,\xi_n^2)\right) \cdot \hat{\xi^1}\right] \\ &- \left[\sum_{j=0}^{N} \hat{D}_{n,j}^{(\xi^2)} \tilde{F}_{m,j}^{(\xi^2)} + \left(\frac{l_n(1)}{w_n^{(\xi^2)}} \tilde{F}^*(\xi_m^1,1) - \frac{l_n(-1)}{w_n^{(\xi^2)}} \tilde{F}^*(\xi_m^1,-1)\right) \cdot \hat{\xi^2}\right] \\ &+ J_{m,n}\vec{Q}_{m,n}; \quad m,n = 0,1,...,N \end{aligned} \tag{1.15}$$

Each element can support curvilinear geometry. Define the mapping from physical space $\vec{x}$ to computational space $\vec{\xi}$ using

$$\vec{x} = \vec{x}(\vec{\xi}). \tag{1.16}$$

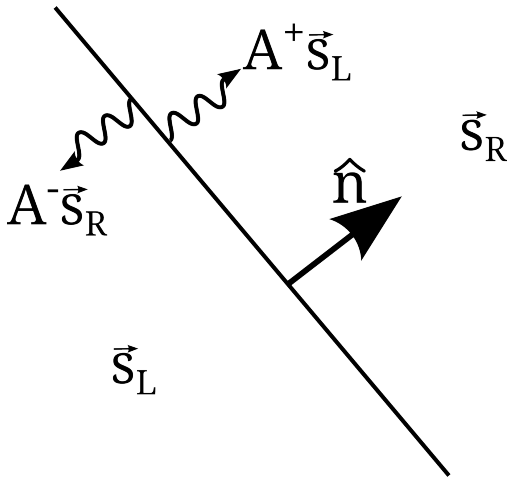


Figure 1.1: A depiction of the setup for computing the flux across an edge. The flux is split by upwinding the characteristic variables of the Jacobian matrix.

In this case, the solution and the source term are weighted by the Jacobian of the mapping and the flux is rotated to the *contravariant* flux. Introducing mappings that are also approximated by polynomial interpolants introduces additional (spectrally accurate) *aliasing* errors. Appendix **??** provides the details on the metric terms that are introduced along with the form of the divergence, gradient, and curl under such a mapping.

Computing the divergence of the conservative flux in this framework can be viewed as calculating a sequence of derivatives in each computational direction. Two steps are required to compute the derivative in each direction. The first is an internal matrix-vector multiply, and the second is computing the weighted Riemann fluxes at the element boundaries. The latter is the only step which requires element-to-element communication. The approximate fluxes are now presented for the systems (1.1), (1.2), and (1.3).

Riemann Flux

This section needs some attention!

Regardless of which system we are solving, the DG approximation requires that we compute an estimate of the flux across an element's edge given the solution on either side of the edge. In general, the solution is discontinuous across the edges. Let $\vec{s}_L$ and $\vec{s}_R$ denote the solution to the "left" and to the "right" of the edge as depicted in Fig. 1.1. The goal is to compute

the flux across an edge given the left and right states. The conservation law, (1.4), can be written

$$\vec{s}_t + \frac{\partial f^n}{\partial \vec{s}} \frac{\partial \vec{s}}{\partial n} = 0, \tag{1.17}$$

where, for the sake of exposition, the source term has been dropped. The flux in the edge-normal direction is $f^n = \vec{f} \cdot \hat{n}$ and the directional derivative of the solution is $\frac{\partial \vec{s}}{\partial n}$. For a short period of time, Δt,

$$\vec{s}_t + \frac{\partial f}{\partial \vec{s}}|_{t=t_0} \frac{\partial \vec{s}}{\partial n} = \mathcal{O}(\Delta t). \tag{1.18}$$

In (1.18), the *Jacobian* of the flux, $\frac{\partial f^n}{\partial \vec{s}}$, is evaluated at the fixed time $t = t_0$. For hyperbolic problems, like the shallow water equations, the Jacobian has real eigenvalues and can be diagonalized. Let

$$\frac{\partial f}{\partial \vec{s}}|_{t=t_0} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1}, \tag{1.19}$$

define the diagonalization, where $\boldsymbol{P}$ is a matrix whose columns are the eigenvectors and $\boldsymbol{D}$ is a diagonal matrix whose diagonal elements are the corresponding eigenvalues of the Jacobian. Substituting (1.19) into (1.18) and multiplying on the left by $\boldsymbol{P}^{-1}$ gives

$$\vec{w}_t + \boldsymbol{D}\boldsymbol{P}^{-1} \frac{\partial \vec{s}}{\partial n} = \mathcal{O}(\Delta t), \tag{1.20}$$

where $\vec{w} = P^{-1}\vec{s}$ are the *characteristic* variables. Equation (1.20) can be rewritten, approximately as

$$\vec{w}_t + \boldsymbol{D} \frac{\partial \vec{w}}{\partial n} \approx \mathcal{O}(\Delta t). \tag{1.21}$$

where variations in the eigenvectors with n have been ignored. Equation (1.21) has approximate solutions

$$w^i = w_0^i(n - \lambda_i(t - t_0)) \tag{1.22}$$

where w^i and λ_i are the i^{th} eigenvector and eigenvalue, w_0^i is the characteristic variable at time $t = t_0$, and n is the physical distance normal to the edge. To evaluate the flux at the edge, we need to know the solution at the edge ($n = 0$). At time $t_0 + \Delta t$,

$$w^i(0, \Delta t) = w_0^i(-\lambda_i \Delta t) \tag{1.23}$$

so that if $\lambda_i > 0$, the solution depends on the state to the left of the edge, and if $\lambda_i < 0$, the solution depends on the initial state to the right of the edge. Because of this, we split the

diagonalization into two components,

$$\frac{\partial f}{\partial \vec{s}}|_{t=t_0} = \boldsymbol{P}\boldsymbol{D}^{+}\boldsymbol{P}^{-1} + \boldsymbol{P}\boldsymbol{D}^{-}\boldsymbol{P}^{-1} = \boldsymbol{A}^{+} + \boldsymbol{A}^{-}, \tag{1.24}$$

where $\boldsymbol{D}^{+}$ is the diagonal matrix with only positive eigenvalues and $\boldsymbol{D}^{-}$ is the diagonal matrix with only negative eigenvalues. The compact notation $\boldsymbol{A}^{+}$ and $\boldsymbol{A}^{-}$ is used for the Jacobian matrix associated with the splitting of the positive and negative eigenvalues.

Under similar assumptions used to obtain (1.22), the flux at the boundary can be approximated

$$\vec{f}^{n} \approx \vec{f}^{n,*} = A^{+}\vec{s}_L + A^{-}\vec{s}_R \tag{1.25}$$

It can be shown that

$$A^{+} = \frac{A + |A|}{2} \tag{1.26a}$$

$$A^{-} = \frac{A - |A|}{2} \tag{1.26b}$$

so that (1.25) can be written

$$\vec{f}^{n,*} = \frac{1}{2}\left(\vec{f}^{n}(\vec{s}_L) + \vec{f}^{n}(\vec{s}_R) - |A|(\vec{s}_L - \vec{s}_R)\right) \tag{1.27}$$

Equation (1.27) is the approximate Riemann flux. The choice of approximation for $|A|$ yields different linear flux schemes. In the shallow water software, we use the Lax-Friedrich's flux where $|A|$ is approximated by the maximum eigenvalue using either the left or the right state,

$$|A| = max(\lambda_i(\vec{s}_L), \lambda_i(\vec{s}_R)). \tag{1.28}$$

Algorithm

To forward-step the discrete system in Eq. (1.15), the flux divergence and the source terms need to be calculate. To calculate the flux-divergence, the DG-derivative matrix needs to be applied...

```
Loop over elements
     Interpolate solution to element boundaries
```

```
Loop over edges
     Calculate Riemann Fluxes

Loop over elements
     Calculate internal derivative and source operations to compute tendency
```

1.4 Profiles of the Dipole Example

In order to determine where our attention should be focused for optimizing the solution algorithm, call-graphs are generated using callgrind and kcachegrind to indicate the subroutines that dominate the program execution. More detailed information about the types of instructions that are being called (memory access, floating point, vectorization, etc.) is obtained using the Allinea-MAP (V6.0) profiler. The profiles presented here are generated on a *(fill in node information (from cpuinfo and/or lstopo)*. The runtime parameters for this example can be found under `SELF/examples/shallowwater/dipole/run/shallowater.params` and, for completeness, are shown here

```
&ModelForm
ModelFormulation = 'Conservative',
nCutoff = 8,
/
&TimeManagement
dt = 0.1,
iterInit = 0,
nTimeSteps = 1000,
dumpFreq = 1000,
/
&SpaceManagement
SpecMeshFile = 'box.mesh',
polyDeg = 10,
nXElem = 20,
```

```
nYElem = 20,
nPlot = 20,
xScale = 4.0D5,
yScale = 4.0D5,
/
&PhysicalParameters
g = 9.81,
f0 = 1.0D-4,
betaX = 0.0,
betaY = 0.0,
linearDrag = 0.0,
/
&DipoleParameters
vMax = 1.75,
x0 = 2.0D5,
y0 = 205000.0,
x1 = 2.0D5,
y1 = 195000.0,
L  = 1.0D4,
/
&SpongeParameters
Lsponge = 5.0D4,
rFacMax = 5.0D-2,
/
&ShelfParameters
hMin = 200.0,
hMax = 3000.0,
Lshelf = 2.5D5,
dL = 0.0,
steepeningCenter = 1.0D6,
```

```
steepeningZoneLength = 5.0D4,
/
```

The compiler flags are *(list the compiler flags!!)*.

integrate_sw_dipole_boxmesh_pdeg10_noOpt_noOMP.map : `-g`

integrate_sw_dipole_boxmesh_pdeg10_O2_noOMP.map : `-O2 -g`

2 Projects

2.1 Algorithm parallelization through OpenMP

2.2 Algorithm parallelization through MPI

2.3 Forcing Vectorization

2.4 Mixed OpenMP and MPI

2.5 Matrix Multiplication on the GPU

2.6 Data Structure Mods for Reducing Memory Access Costs

2.7 Parallel File I/O