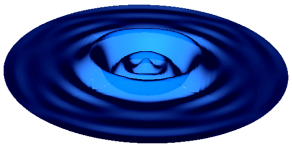
Spectral Element Libraries in Fortran

(SELF)

Shallow Water Equations



Reference Manual

Joseph Schoonover

Contents

1 Equations, Discretization, and the Serial Algorithm

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.

1.1 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).

1.2 Nodal Discontinuous Galerkin 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.

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

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

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

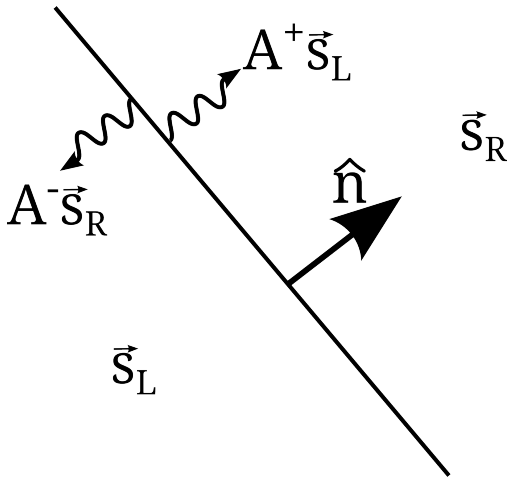


Figure 1.1: A depiction of the setup for computing the flux across an edge. The flux is split by upwinding the characteristic variables of the Jacobian matrix.

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,

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

Comparison with other SEM-Shallow Water Models

The discretization presented here distinguishes itself from previous shallow water models presented in the literature. The models of Iskandarani et al. (1995) (and Ma1993) make use of the Gauss-Lobatto quadrature points, which were shown in Kopriva (2006) to not satisfy the discrete metric identities and lead to the introduction of spurious modes associated with isoparametric elements. Here, the more accurate Gauss quadrature is used. The increased accuracy and satisfication of the discrete metric identities increases the algorithm cost. Since

the Gauss quadrature points do not include the element boundaries, an additional step (with $\mathcal{O}(N^2)$ operations per element) is needed to interpolate the solution to the element boundaries before computing Riemann fluxes. An adpative filtering procedure, similar to **?** is used for maintaining stability when marginally resolved modes violate energy conservation. In this way, the filter lowers the accuracy of the method only in locations where the solution is marginally resolved; the gain in performing the filtering is that numerical stability is maintained. This filtering procedure is different than Taylor et al. (1997) where the filter is applied everywhere at each time step. In the conservative form of the shallow water equations, care was taken in the computation of the bathymetry gradient to guarantee that a motionless fluid remains motionless to machine precision.

2 Getting Started With an Example

The easiest way to get started with the SELF Shallow Water software is run a pre-configured example. Doing so will introduce you to compiling, running, run-time configuring, and viewing the output. Once you are comfortable with a workflow for the example and you are ready to set up the software for your particular experiment, it may be beneficial to browse through other pre-configured examples to find one that is closest to the setup you have in mind. From there, modifications to the code can be made. Details on some of the more important routines and those you are likely to modify are provided in Chapter 3.

2.1 Downloading

The SELF is currently available on github. There are two ways to obtain the code. If you are interested in obtaining the most recent snapshot of the code, a tar file can be downloaded directly from the git website. Clicking here will bring you to the SELF. Once you follow the link (if the embedded link doesn't work, copy and paste

```
https://github.com/schoonovernumerics/SELF
```

into your browser's url bar) click on "Download ZIP", which is located towards the right side of the page. Unpack the tar file into your favorite directory. This will make a directory "SELF" in your favorite directory, underneath which all of the source code and examples included with the SELF.

Alternatively, if you have git installed on your computer, you can clone the SELF repository by issuing the command

```
git clone https://github.com/schoonovernumerics/SELF
```

Again, this will make the directory "SELF" in the current directory where you issued the clone command.

2.2 Compile the example

At this point, you should have downloaded the SELF. We will first navigate to the examples directory and find the example for the "shelf-steepening" demo. This example can be found under

```
SELF/examples/shallowwater/shelfsteepening/
```

Go ahead and change directories to this example.

As with all of the supplied examples, there are three subdirectories

```
build/     localmods/     run/
```

This subdirectory structure is used in order to keep your model output, source code modifications, and model build details separate. For now we won't worry about the `localmods/` subdirectory; this subdirectory will be discussed later in Chapter 3 where we will illustrate how to modify the source code if needed for your application.

Right now, the most important thing is to get you familiar with compiling and running the code. Change directories into the build subdirectory. Underneath the build directory, there is a makefile and a simple build script. For now, you don't need to worry about changing anything to either one of these files. In Sec. 2.4, there are directions for changing your fortran compiler, compiler optimizations, warning flags, debugging flags, etc. To compile the code, simply type the command

```
./rebuild.sh
```

and press enter. This will compile all of the necessary modules and driver programs (all of the dependencies are expressed in the makefile) and will generate two executables, `initialize_sw` and `integrate_sw`. Additionally, the `rebuild` script cleans up the build directory by removing the `.o` and `.mod` files and moves the executable to the run directory. If you have

problems running the build script, it may be that the permissions are not set appropriately. To set the proper permissions, do

```
chmod 755 rebuild.sh
```

and then try to run the script again. If this still doesn't work, send me an e-mail, and, for now, issue the commands

```
make initialize_sw
make integrate_sw
make clean
mv initialize_sw integrate_sw ../run/
```

2.3 Running the Code and Viewing the Output

Ok, so now you have successfully compiled the code. Change directories to the `run` directory. In the run directory, you will find the executables (`initialize_sw` and `integrate_sw`), a SpecMesh file (`box.mesh`), and a text file (`shallowwater.params`).

Running the Code

Both executables require that you have a namelist file in the directory named `shallowwater.params`. There should be one included already with your fresh download. This namelist file specifies parameters that are used for the explicit time stepping, spatial discretization, and plotting and sets physical parameters relevant to the shallow water model. If you like, feel free to take a look at the file to familiarize yourself with the parameters, but don't worry about setting anything; the provided parameters file yields a successful run for this example. In Sec. 2.4, we will discuss each parameter.

The work-flow for conducting a simulation is divided into two steps. The first step is the initialization. In the initialization phase, the model grid, bathymetry, forcing terms, and initial conditions are specified. The details of how this is done can be seen in the program `shelfsteepening/localmods/ShallowWater_InitialConditions.f90`.

To run the initialization program, execute

```
./initialize_sw
```

Once it finishes running, there will be three files that are generated:

1. `mesh.tec` : A tecplot file that contains the computational mesh and metric terms that were used in the computation of the solution.

2. `ShallowWater.init.tec` : A tecplot file that contains the initial conditions that are set in `ShallowWater_InitialConditions.f90`

3. `ShallowWater.0000000000.pickup` : A "pickup" file that contains the necessary data to initialize the forward integration of the model.

The `.tec` files can be viewed in VisIt or your favorite visualization software that can read TecPlot files. For this example, the model has already been tested and the initial configuration has been deemed acceptable for the physical problem in quetion. At this point in the work-flow, though, is usually when you would take a look at the initial conditions, the model grid, and the metric terms and verify that the setup is acceptable. Once you are happy with the initial condition configuration, it is time to move on to the forward integration phase.

At this point, it is relevant to discuss three of the parameters in the `shallowwater.params` file.

1. `iterInit` : This is the initial iterate, specified as an integer. This integer corresponds to the iteration stamp of the pickup file that you intend to read in. The iteration stamp is an integer that is padded with enough zeros to yield 10 digits.

2. `nTimeSteps` : This is the number of time steps that the integration program will execute. At the end of execution, the integration program will create a pickup file with an iteration stamp corresponding to `iterInit`+`nTimeSteps`.

3. `dumpFreq` : This is the frequency at which tecplot files are written.

These parameters, if you have not modified them, are set as

```
iterInit = 0,
nTimeSteps = 5000,
dumpFreq = 1000,
```

so that the integration will begin with the pickup file generated from the initialization program, 5000 time steps will be executed, and a tecplot file will be generated every 1000 time steps. To run the integration program, do

```
./integrate_sw
```

Once the program completes, you will see six new tecplot files and a new pickup file (`ShallowWater.0000005000.pickup`). The new pickup file can be used as a new initial condition if you wish to continue integrating past 5000 time steps, by setting

```
iterInit = 5000,
```

At this point, you have successfully compiled and completed the initialization and forward integration steps in the work-flow. This is a good time to discuss visualization before we discuss making modifications to the run-time parameters in `shallowwater.params`.

Viewing the output in VisIt

Now that the code has finished running and you have some output, you can view the tecplot and curve files in your favorite visualization software. Here, we will describe how to view the output using VisIt.

2.4 Configuration

3 Making modifications

Bibliography

M. Iskandarani, D.B. Haidvogel, and J.P. Boyd. A staggered spectral element model with application to the oceanic shallow water equations. *International Journal For Numerical Methods in Fluids*, 20:393–414, 1995.

D.A. Kopriva. Metric identities and the discontinuous galerkin spectral element method on curvilinear meshes. *Journal of Scientific Computing*, 26-3:301–327, 2006.

M. Taylor, J. Tribbia, and M. Iskandarani. The spectral element method for the shallow water equations on the sphere. 130:92–108, 1997.