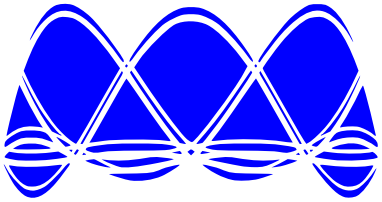
SELF - Fluids



Documentation

Joseph Schoonover

©2017-2018 Joseph Schoonover (joe@fluidnumerics.com

Contents

Part I

Theory

Equations and Discretizations

In the SELF, a conservation law is viewed in the general form

$$\vec{s}_t + \nabla \cdot \vec{f} = \vec{q}, \tag{1.1}$$

where $\vec{s}$ is a vector of *prognostic* variables, $\vec{f}$ is a vector of *conservative flux vectors*, and $\vec{q}$ is a vector of *non-conservative source terms*.

For the compressible Navier-Stokes in conservative form,

$$\vec{s} = \begin{pmatrix} \rho u \\ \rho v \\ \rho w \\ \rho' \\ (\rho\theta)' \end{pmatrix} \tag{1.2a}$$

$$\vec{f} = \begin{pmatrix} \rho\vec{u}u + p'\hat{x} - \vec{\nu}_1 \cdot \nabla u \\ \rho\vec{u}v + p'\hat{y} - \vec{\nu}_2 \cdot \nabla v \\ \rho\vec{u}w + p'\hat{z} - \vec{\nu}_3 \cdot \nabla w \\ \rho\vec{u} \\ \vec{u}\rho\theta - \vec{\kappa} \cdot \nabla T \end{pmatrix} \tag{1.2b}$$

$$\vec{q} = \begin{pmatrix} \mathcal{F}_u \\ \mathcal{F}_v \\ -\rho' g + \mathcal{F}_w \\ \rho \\ \rho\theta \end{pmatrix} \tag{1.2c}$$

(1.2d)

In this formulation, the static density, potential temperature, and pressure are removed from the prognostic variables. They are split as follows

$$\rho = \bar{\rho}(z) + \rho'(x,y,z,t) \tag{1.3a}$$

$$P = \bar{P}(z) + P'(x,y,z,t) \tag{1.3b}$$

$$\rho\theta = (\bar{\rho}+\rho')(\bar{\theta}+\theta') \tag{1.3c}$$

$$= \bar{\rho}\bar{\theta} + \bar{\rho}\theta' + \bar{\theta}\rho' + \rho'\theta' \tag{1.3d}$$

$$(\rho\theta)' = \bar{\rho}\theta' + \bar{\theta}\rho' + \rho'\theta' \tag{1.3e}$$

The DGSEM discretizes (1.1) in its weak form. To obtain the weak form, (1.1) is weighted with a test function and integrated over the physical domain, denoted by Ω. To approximate the integrals in the weak form, the domain is first divided into non-overlapping elements (Ω^κ) . Integration is performed over each element and the solution and test function are permitted to be piecewise discontinuous across elements. These assumptions lead to the statement

$$\int_{\Omega^\kappa} (\vec{s}_t + \vec{q})\phi \, d\Omega^\kappa - \int_{\Omega^\kappa} \vec{f}\cdot\nabla\phi \, d\Omega^\kappa + \oint_{\partial\Omega^\kappa} \phi\vec{f}\cdot\hat{n} \, dA^\kappa = 0, \quad \forall\phi \in \mathbb{C}_0(\Omega_\kappa), \;\; \kappa = 1,2,...,K \tag{1.4}$$

where $\mathbb{C}_0(\Omega^\kappa)$ is the space of functions that are continuous over Ω^κ.

Equation (1.4) assumes that the integration over each element is independent; this *compactness* results from allowing piecewise discontinuous solutions. The third term in (1.4) is the integral of the conservative flux over the element boundary. This is the only term that involves communication with other elements.

The formulation presented in (1.4) only requires that we know the geometry of each element and the connectivity of a collection of elements. This allows for the use of either structured or unstructured mesh frameworks. Additionally, the elements which comprise the mesh can have curvilinear geometry. Define the mapping from physical space $\vec{x}$ to computational space $\vec{\xi}$ using

$$\vec{x} = \vec{x}(\vec{\xi}). \tag{1.5}$$

Section **??** provides the details on the metric terms that are introduced along with the form of the divergence, gradient, and curl under such a mapping. For simplicity, the computational domain is formed from tensor products of intervals $[-1,1]$ in each coordinate direction

restricting the elements to curvilinear hexahedrons (3-D). Under the mapping (1.5), the weak form (1.4) becomes

$$\int_{\Omega^\xi} (\tilde{s}_t^\kappa + \tilde{q}^\kappa)\phi \, d\Omega^\xi - \int_{\Omega^\xi} \tilde{f}^\kappa \cdot \nabla_{\vec{\xi}}\phi \, d\Omega^\xi + \oint_{\partial\Omega^\xi} \phi \tilde{f}^{\kappa,*} \cdot \hat{n}^\xi \, dA^\xi = 0, \;\; \forall\phi \in \mathbb{C}^0(\Omega^\xi) \tag{1.6}$$

where $\tilde{s} = J^\kappa \vec{s}$, $\tilde{q} = J^\kappa \vec{q}$, $\tilde{f} = (J^\kappa \vec{a}^{\kappa,i} \cdot \vec{f})\hat{a}^{\kappa,i}$, J^κ is the Jacobian of the transformation over the κ^{th} element, and $\vec{a}^{\kappa,i}$ are the contravariant basis vectors associated with the transformation of element κ (the $\hat{a}$ denotes a unit vector).

Given a conservative flux, a non-conservative source, internal element metrics, and global element connectivity, the discrete algorithm solves (1.6) by approximating the integrands with interpolants and the integrals by discrete quadratures. First, the prognostic solution, source term, conservative flux, and mapping are approximated by Lagrange interpolants of degree N.

$$\vec{s} \approx I^N(\vec{s}) = \sum_{i,j,k=0}^{N} \vec{S}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.7a}$$

$$\vec{q} \approx I^N(\vec{q}) = \sum_{i,j,k=0}^{N} \vec{Q}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.7b}$$

$$\vec{f} \approx I^N(\vec{f}) = \sum_{i,j,k=0}^{N} \vec{F}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.7c}$$

$$\vec{x} \approx I^N(\vec{x}) = \sum_{i,j,k=0}^{N} \vec{X}_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.7d}$$

An additional simplification is to approximate the product of interpolants by an interpolant of degree N which can incur an additional aliasing error.

$$\tilde{s} \approx I^N(I^N(J)I^N(\vec{s})) = \tilde{S} = \sum_{i,j,k=0}^{N} (J\vec{S}_{i,j,k}) l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.8a}$$

$$\tilde{q} \approx I^N(I^N(J)I^N(\vec{q})) = \tilde{Q} = \sum_{i,j,k=0}^{N} (J\vec{Q})_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.8b}$$

$$\tilde{f}_{(i,n)} \approx I^N(I^N(Ja^i_{(n)})I^N(\vec{f})) = \tilde{F}_{(i,n)} = \sum_{i,j,k=0}^{N} (Ja^i_{(n)}\vec{F}_{(n)})_{i,j,k} l_i(\xi^1) l_j(\xi^2) l_k(\xi^3) \tag{1.8c}$$

Last, the space of test functions ($\mathbb{C}_0$) is approximated by the $\mathbb{P}^N$, the space of polynomials of degree N. Thus, the test function ϕ is replaced by each of the Lagrange interpolating polynomials,

$$\phi_{m,n,p} = l_m(\xi^1) l_n(\xi^2) l_p(\xi^3) \tag{1.9}$$

With this, equation (1.6) becomes

$$\int_{\Omega^\xi} (\tilde{S}_t^\kappa + \tilde{Q}^\kappa)\phi_{m,n,p}\, d\Omega^\xi - \int_{\Omega^\xi} \tilde{F}^\kappa \cdot \nabla_\xi \phi_{m,n,p}\, d\Omega^\xi + \oint_{\partial\Omega^\xi} \phi_{m,n,p} \tilde{F}^{\kappa,*} \cdot \hat{n}^\xi\, dA^\xi = 0, \quad m,n,p = 0,1,...,N \tag{1.10}$$

The final step is to replace the integrals in (1.10) with discrete quadrature. For this we use the Legendre-Gauss quadrature, which yields exact integration for each term in (1.10). Additionally, the interpolation nodes are specified as the Legendre-Gauss nodes, which simplifies the integration.

$$\begin{aligned}
\left(J_{m,n,p}\vec{S}_{m,n,p}\right)_t = &- \left[\sum_{i=0}^{N} \hat{D}^{(\xi^1)}_{m,i} \tilde{F}^{(\xi^1)}_{i,n,p} + \left(\frac{l_m(1)}{w_m^{(\xi^1)}} \tilde{F}^*(1,\xi_n^2,\xi_p^3) - \frac{l_m(-1)}{w_m^{(\xi^1)}} \tilde{F}^*(-1,\xi_n^2,\xi_p^3)\right) \cdot \hat{\xi^1}\right] \\
&- \left[\sum_{j=0}^{N} \hat{D}^{(\xi^2)}_{n,j} \tilde{F}^{(\xi^2)}_{m,j,p} + \left(\frac{l_n(1)}{w_n^{(\xi^2)}} \tilde{F}^*(\xi_m^1,1,\xi_p^3) - \frac{l_n(-1)}{w_n^{(\xi^2)}} \tilde{F}^*(\xi_m^1,-1,\xi_p^3)\right) \cdot \hat{\xi^2}\right] \\
&- \left[\sum_{k=0}^{N} \hat{D}^{(\xi^3)}_{m,n,k} \tilde{F}^{(\xi^2)}_{m,n,k} + \left(\frac{l_p(1)}{w_p^{(\xi^3)}} \tilde{F}^*(\xi_m^1,\xi_n^2,1) - \frac{l_p(-1)}{w_p^{(\xi^3)}} \tilde{F}^*(\xi_m^1,\xi_n^2,-1)\right) \cdot \hat{\xi^3}\right] \\
&+ J_{m,n,p}\vec{Q}_{m,n,p}; \qquad m,n,p = 0,1,...,N
\end{aligned} \tag{1.11}$$

The numerical flux vector is split into a hyperbolic component, which includes advection and sound wave propagation, and a parabolic component, which includes momentum and heat diffusion processes. The hyberbolic flux is estimated using the Lax Friedrich's Riemann solver, and the parabolic flux is estimated using the Bassi-Rebay flux with an interioir penalty method.

Hyperbolic Riemann Flux

This section needs some attention!

Regardless of which system we are solving, the DG approximation requires that we compute

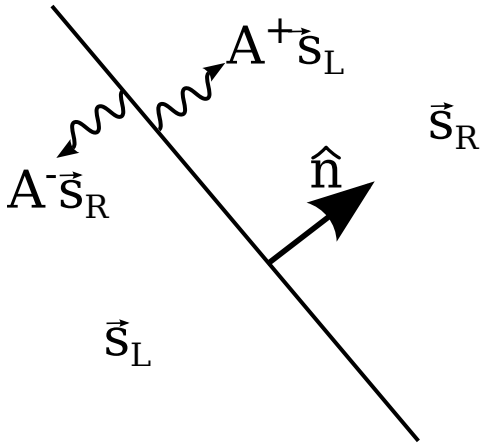


Figure 1.1: A depiction of the setup for computing the flux across an edge. The flux is split by upwinding the characteristic variables of the Jacobian matrix.

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, (**??**), can be written

$$\vec{s}_t + \frac{\partial f^n}{\partial \vec{s}} \frac{\partial \vec{s}}{\partial n} = 0, \tag{1.12}$$

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,

$$\vec{s}_t + \frac{\partial f}{\partial \vec{s}}|_{t=t_0} \frac{\partial \vec{s}}{\partial n} = \mathcal{O}(\Delta t). \tag{1.13}$$

In (1.13), 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

$$\frac{\partial f}{\partial \vec{s}}|_{t=t_0} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1}, \tag{1.14}$$

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 (1.14) into (1.13) and multiplying on the left by $\boldsymbol{P}^{-1}$ gives

$$\vec{w}_t + \boldsymbol{D}\boldsymbol{P}^{-1} \frac{\partial \vec{s}}{\partial n} = \mathcal{O}(\Delta t), \tag{1.15}$$

where $\vec{w} = P^{-1}\vec{s}$ are the *characteristic* variables. Equation (1.15) can be rewritten, approximately as

$$\vec{w}_t + \boldsymbol{D}\frac{\partial \vec{w}}{\partial n} \approx \mathcal{O}(\Delta t). \tag{1.16}$$

where variations in the eigenvectors with n have been ignored. Equation (1.16) has approximate solutions

$$w^i = w_0^i(n - \lambda_i(t - t_0)) \tag{1.17}$$

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 $t_0 + \Delta t$,

$$w^i(0, \Delta t) = w_0^i(-\lambda_i \Delta t) \tag{1.18}$$

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,

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

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 (1.17), the flux at the boundary can be approximated

$$\vec{f}^n \approx \vec{f}^{n,*} = A^+\vec{s}_L + A^-\vec{s}_R \tag{1.20}$$

It can be shown that

$$A^+ = \frac{A + |A|}{2} \tag{1.21a}$$

$$A^- = \frac{A - |A|}{2} \tag{1.21b}$$

so that (1.20) can be written

$$\vec{f}^{n,*} = \frac{1}{2}\left(\vec{f}^n(\vec{s}_L) + \vec{f}^n(\vec{s}_R) - |A|(\vec{s}_L - \vec{s}_R)\right) \tag{1.22}$$

Equation (1.22) is the approximate Riemann flux. The choice of approximation for $|A|$ yields different linear flux schemes. In the compressible Navier Stokes solver, we use the Lax-Friedrich's flux where $|A|$ is approximated by the maximum eigenvalue using either the left or the right state,

$$|A| = max(|\lambda_i(\vec{s}_L)|, |\lambda_i(\vec{s}_R)|). \tag{1.23}$$

For the compressible Navier Stokes,

$$|A| = max(|\vec{u}_L \cdot \hat{n} \pm c_L|, |\vec{u}_R \cdot \hat{n} \pm c_R|). \tag{1.24}$$

where $c = \left(\frac{\partial \rho}{\partial P}\right)^{-1/2}$ is the speed of sound, derived from the equation of state.

Viscous Flux

The Compressible Navier Stokes solver comes equipped with an optional Laplacian operator for implementing classic subgridscale model. It is typically necessary for such models to compute the gradient of each of the velocity components, the potential temperature, and, in some cases, the density.

Computing Gradients in the Weak Form

In the SELF-CNS, differential operations are performed in mapped coordinates, where the physical coordinate $\vec{x}$ is related to the computational coordinate $\vec{\xi}$ via a one-to-one mapping

$$\vec{x} = \vec{x}(\vec{\xi}) \tag{1.25}$$

within each element.

The gradient of a function in mapped coordinates is

$$\frac{\partial f}{\partial x_j} = \frac{1}{J} \sum_{i=1}^{3} J a_j^i \frac{\partial f}{\partial \xi^i} \tag{1.26}$$

where J is the Jacobian of the mapping, and a_j^i is the j^{th} component of the i^{th} contravariant basis vector. The contravariant metric tensor is defined

Computing the Second Derivative

Boundary Conditions

In all cases, the boundary conditions are enforced by prescribing an external state that is used in the calculation of the hyperbolic and viscous fluxes.

No Normal Flow (with no Stress)

Radiation

Prescribed

Drag Slip with No Normal Flow

Algorithm sketch

In any number of dimensions, the basic DGSEM can be broken into four main steps

1. For each element, interpolate the solutions to the element boundaries.
2. For each boundary edge/face, update the external states to apply boundary conditions.
3. For each edge/face, calculate the Riemann flux.
4. For each element, apply the DG-derivative matrix and add the weighted Riemann fluxes and source terms to produce a tendency.

Geometry

Unstructured Mesh

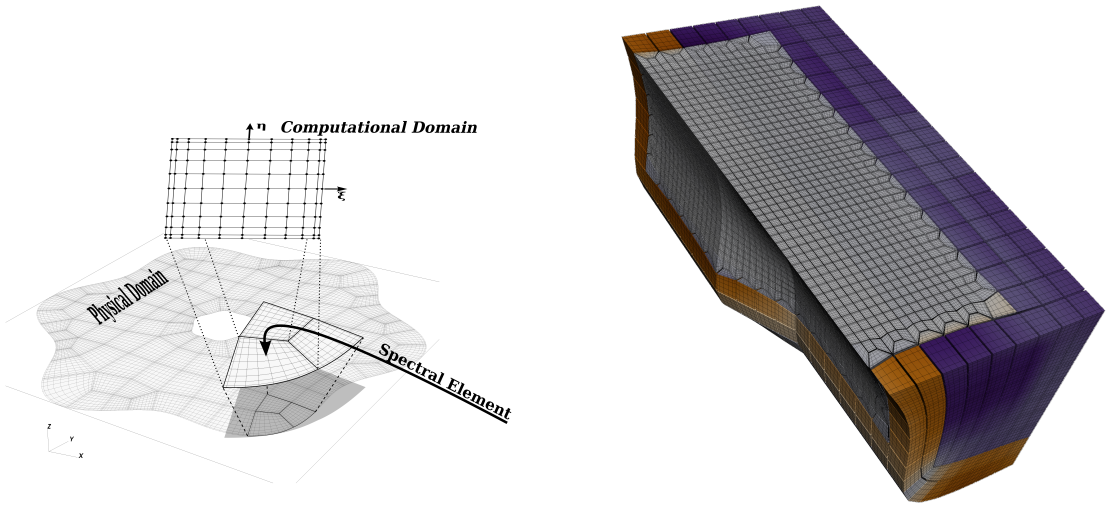


Figure 2.1: A schematic of a spectral element mesh in two-dimensions (left) and an example 3-D unstructured mesh of hexahedrals.

Metric Terms

Time Integration Schemes

Part II

Validation and Verification

Verification

In the verification of this software, there are a number of test programs provided that demonstrate the spectral accuracy of interpolation and differentiation operations in mapped and unmapped coordinates.

Spectral accuracy means that the rate of decrease of the error with increasing polynomial degree depends on the smoothness of the approximated function.

- If the function is a polynomial of degree N, then approximation with a polynomial of degree N or greater should be exact and the error should only be due to round-off errors from floating point arithmetic.
- For smooth functions that are complete (all of its derivatives are defined everywhere), the error decreases exponentially with increasing interpolating polynomial degree.
- Functions with discontinuities, or whose derivatives have discontinuities, result in error decay that is algebraic with increasing interpolating polynomial degree.

These three characteristics of spectral element methods provides a means for verifying that the differentiation and interpolation routines are indeed performing those operations with spectral accuracy.

Testing under spectral subdirectory

The spectral subdirectory provides the most basic routines for performing interpolation and differentiation on a computational grid. A test program, `src/spectral/tests/Spectral_Tests.f90`, is written to test exactness and exponential error decay in three dimensions for interpolation

and differentation. The gradient and divergence routines are tested using strong and weak forms of these operations.

In the directory `SELF-Fluids/testing`, the script `run_tests.sh` can be used to run CPU and GPU versions of the provided tests. Output of the test programs is compared against reference output that has been verified. The `spectral_tests` output includes interpolation and divergence errors (strong and weak forms) for exactness and exponential error decay test cases.

To test both CPU and GPU versions, you must have access to the PGI compilers. Note that the GPU and CPU versions are not bitwise identical.

In all tests, the Legendre Gauss quadrature nodes are used as the interpolation nodes.

Interpolation errors are estimated by mapping the lower order interpolant to Legendre Gauss nodes degree 50, and comparing with a 50^{th} degree interpolant. Differentation (gradient and divergence) errors are estimated by comparing against the exact gradient and divergence at the same interpolation nodes.

Exactness

The scalar function

$$f_1(x,y,z) = xyz \tag{4.1}$$

and the vector function

$$\vec{g}_1(x,y,z) = yz\hat{x} + xz\hat{y} + xy\hat{z} \tag{4.2}$$

are used to test exactness.

The divergence of $\vec{g}_1$ is

$$\nabla \cdot \vec{g}_1 = 0 \tag{4.3}$$

For the weak divergence, the normal component of the vector function is prescribed on the faces of the element.

Figure 4.1 shows semi-log plots of the max norm of the error for interpolation and strong and weak form of the divergence operator as a function of the polynomial degree. Notice that the max norm errors hover around and above 10^{-15} for double precision and 10^{-5} for single

precision. The growth in the divergence errors with increasing polynomial degree suggests that the higher order methods are increasingly susceptible to round-off errors.

Exponential convergence

The scalar function

$$f_2(x, y, z) = \sin(\pi x)\sin(\pi y)\sin(\pi z) \tag{4.4}$$

and the vector function

$$\vec{g}_2(x, y, z) = \pi\left(\cos(\pi x)\sin(\pi y)\sin(\pi z)\hat{x} + \sin(\pi x)\cos(\pi y)\sin(\pi z)\hat{y} + \sin(\pi x)\sin(\pi y)\cos(\pi z)\hat{z}\right) \tag{4.5}$$

are used to test exactness.

The divergence of $\vec{g}_1$ is

$$\nabla \cdot \vec{g}_1 = -3\pi^2 f_2 \tag{4.6}$$

Figure 4.2 shows semi-log plots of the max norm of the error for interpolation and strong and weak form of the divergence operator as a function of the polynomial degree. Exponential decay of the interpolation and divergence errors is observed in all instances until floating point round-off errors dominate error behavior. For single precision, floating point errors dominate beyond a polynomial degree of 10 and for double precision they dominate beyon polynomial degree of 16.

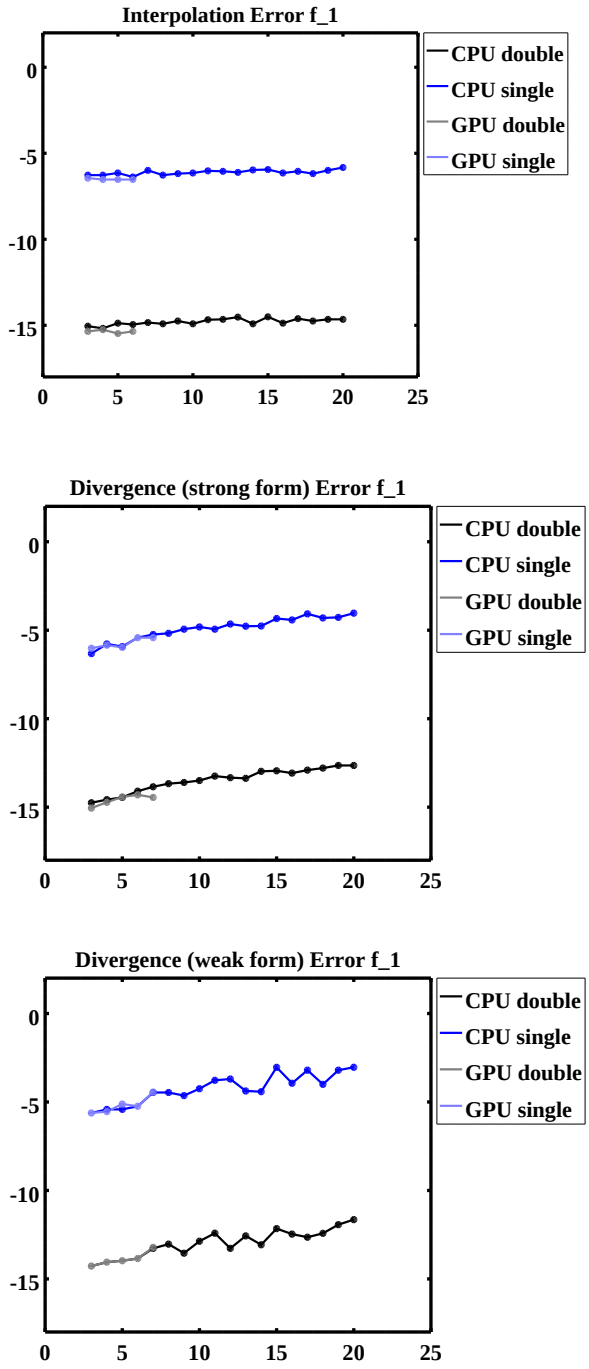


Figure 4.1: The interpolation and divergence (strong and weak form) errors are shown for the exactness test case with single and double precision on the CPU's and GPU's. All errors are depicted in a semi-log plot.

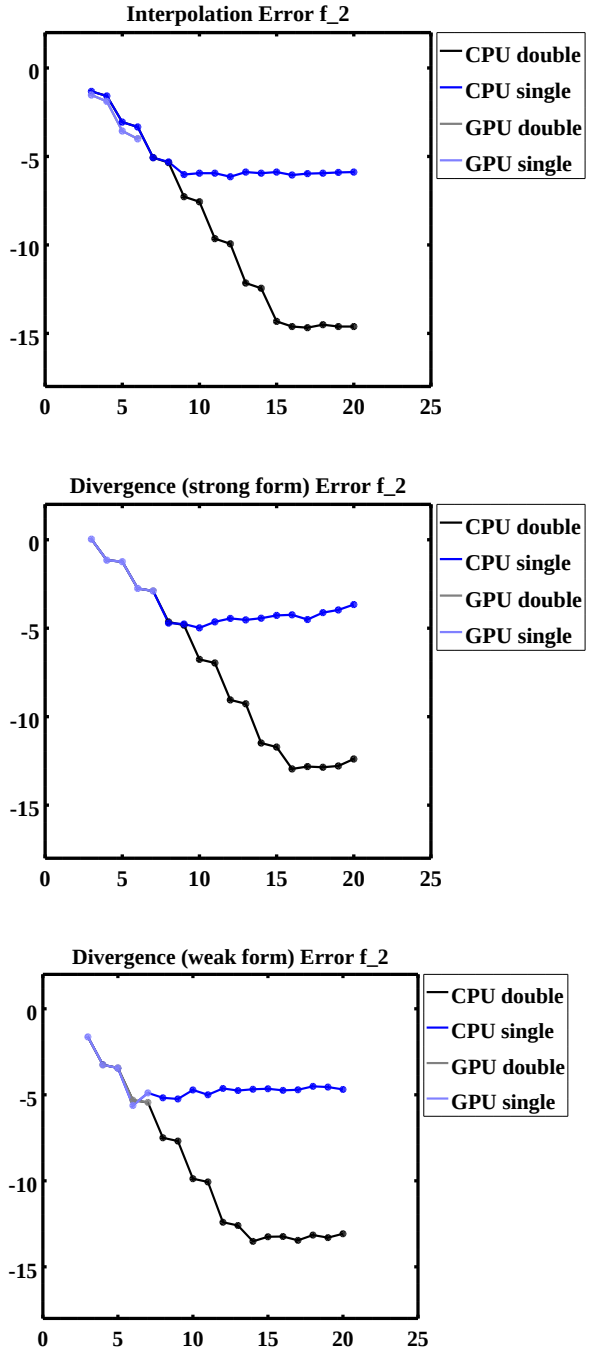


Figure 4.2: The interpolation and divergence (strong and weak form) errors are shown for the exponential convergence test case with single and double precision on the CPU's and GPU's. All errors are depicted in a semi-log plot.

Validation

Thermal Bubble

Lock Exchange in 2-D

Planetary Boundary Layer

The domain consists of 20x20x10 elements with a polynomial degree of 7 representing the solution within each element; giving roughly 10 million degrees of freedom that approximate the fluid state. With the domain being 2km *times* 2km *times* 1km, this yields a resolution of roughly 2 m. The model is stepped forward using Williamson's Low Storage 3rd Order Runge-Kutta with a time step of 2.5 ms in order to satisfy the CFL condition associated with sound waves.

In this section, we compare the solutions obtained and the time-to-solution by using the spectral filtering, spectral EKE model, and Laplacian diffusion with a fixed mixing coefficient.

Benchmarking and Scaling

Benchmark simulations were run on Theia-Fine grain, which has 8 Nvidia-P100 GPU's per node, and ¡CPU Architecture¿. For these tests, version 17.5 of the PGI compilers is used with the `-fast` optimization flag. All floating point arrays are stored with single precision and all floating point operations are single precision.

Flow over a Gaussian Hill

Baroclinic Instability