Chemical Transport Modelling with Spectral Element Method

Cynthia Xinshi Chen and Sam Loomis

Mentors: Dr. John Drake, Dr. Joshua Fu, Dr. Kwai Wong
Overview

- **Background**
- **Spectral Element Method**
  - Basis Function
  - Weak Form
  - Quadrature
  - Global v.s Local Formulation
  - Euler and Newton Methods
  - Continuous Galerkin v.s Discontinuous Galerkin
- **Coding**
  - Serial Code
  - Parallel Code
Background

- **CESM (Community Earth System Model):**
  Aimed at understanding and predicting the climate system.

- **Climate and chemical transport models:**
  Require the use of unstructured grids and conservation of mass and energy. Therefore, the model is written using the FVM (Finite Volume Method), which is explicitly conservative.

- **CESM and the HOMME equations:**
  Formulated through the FEM (Finite Element Method) because of its use on unstructured grids. This is convenient when solving the equations on a globe.

- **Spectral Element Method:**
  Mark Taylor [1] has shown that the Spectral Element Method, a type of FEM, is explicitly locally conservative, as well as having other ideal properties, such as a diagonal mass matrix.

- **Object:**
  We want to show that chemical transport problems can be accurately modeled with the SEM, so that they may be integrated with the HOMME equations.
Basis Functions

- A type of continuous-Galerkin Finite Element Method with explicit local and global conservation, and a diagonal mass matrix
- A typical continuous-Galerkin formulation of a problem starts by replacing the fields $u$ with a piecewise polynomial function.
- Because of this, $u$ can be approximately represented as a sum of basis functions:

$$ u(\tilde{x}) \approx \sum_{\ell} u(\xi_{\ell}) \phi_{\ell}(\tilde{x}) $$

- Because of this, we can also write the first derivatives of $u$ as

$$ \nabla u(\tilde{x}) \approx \sum_{\ell} u(\xi_{\ell}) \nabla \phi_{\ell}(\tilde{x}) $$
Weak Form

- As a result of the polynomial approximation, $u$ is not second-differentiable.
- However, differential equation with a second derivative can be transformed into a differential-integral equation through integration by parts.
- Integrate the equation with a test function $v$. e.g.:

$$\frac{\partial u}{\partial t} = \nabla^2 u \quad \rightarrow \quad \int_{\Omega} v \frac{\partial u}{\partial t} dx = \int_{\Omega} \nabla v \nabla^2 u dx \quad \rightarrow \quad \int_{\Omega} v \frac{\partial u}{\partial t} dx = \int_{\partial \Omega} v \nabla u \cdot \vec{n} \, d\Omega - \int_{\Omega} \nabla v \cdot \nabla u dx$$

- By inserting the polynomial approximation, turn the integro-differential problem into a linear algebra problem:

$$\int_{\Omega} \phi_i \frac{\partial u}{\partial t} dx = B.C. - \int_{\Omega} \nabla \phi_i \cdot \nabla u dx \quad \rightarrow \quad \sum_j \frac{\partial u(\xi_j)}{\partial t} \int_{\Omega} \phi_i \phi_j dx = B.C. - \sum_j u(\xi_j) \int_{\Omega} \nabla \phi_i \cdot \nabla \phi_j u dx$$

- The integrals are calculated using a Gaussian quadrature.
Gaussian Quadrature

- To approximate an integral by quadrature, we write it as a weighted sum over discrete points:

\[ \int_{\Omega_m} f(\vec{x}) d\vec{x} \approx \sum_{\vec{k}} f(\vec{x}_{\vec{k}}) M(\vec{x}_{\vec{k}}) w_{\vec{k}} \]

- In the Spectral Element Method, we set the nodal points used for polynomial interpolation equal to the quadrature points. This means the mass matrix can be simplified as:

\[ \int_{\Omega_m} \phi_i \phi_j d\vec{x} \approx \sum_{\vec{k}} \phi_i(\vec{\xi}_{\vec{k}}) \phi_j(\vec{\xi}_{\vec{k}}) M(\vec{\xi}_{\vec{k}}) w_{\vec{k}} = M(\vec{\xi}_i) w_i \delta_{ij} \]

- This greatly simplifies the linear algebra involved.
- The Gauss-Lobatto quadrature (which SEM uses) is exact for polynomials of degree \( d \leq 2n-1 \), where \( n \) is the number of quadrature points. This means many vector-calculus identities are preserved in this formulation, which leads to local and global conservation.
Global vs. Local Formulation

- Due to the piecewise approximation, elements are mostly independent of each other.
- The only interdependence between elements occurs at the boundaries, where elements share nodal points.
- When integrating, one can solve the matrices globally, or one can distribute local element data to different processors and integrate each element in parallel.
- This gives the exact same result as the global method for interior points, but points on elemental boundaries may disagree between elements. This discontinuity is resolved by a weighted average.
- After the parallel integration + weighted sum, the local method and global method are mathematically equivalent.
Euler and Newton methods

- Once the time derivative is known, we can calculate the next time step through the Forward Euler Method:
  \[
  \frac{du_i}{dt} = F_i(u) \quad \text{Then,}\quad u_i(t + \Delta t) = u_i + \Delta u_i \\
  \Delta u_i = F_i(u)\Delta t
  \]

- This method can have instabilities, however, which requires a CFL factor to control it (typically has the form $\Delta t = c\Delta x^n$ for some $c$ and $n$).
- One way to remove these instabilities is to use an implicit Euler method:
  \[
  u_i(t + \Delta t) = u_i + \Delta u_i \\
  \Delta u_i = (1 - \theta)F_i(u(t)) + \theta F_i(u(t + \Delta t))
  \]

- This is typically a nonlinear equation that must be solved via iterative Newton methods and the Jacobian matrix $J$:
  \[
  \Delta u_i^{(n+1)} = \Delta u_i^{(n)} + \sum_j \frac{1}{\Delta t(1 - \theta)} J(u(t) + \Delta u^{(n)})^{-1} \Delta u_j^{(n)} \\
  J(u)_{ij} = \frac{\partial F_i}{\partial u_j}(u)
  \]
Euler and Newton methods (cont.)

- The Jacobian matrix will have dimensions \((N_F \times (d + 1)^{\text{dim}})^2\), where 
  \(N_F\) = number of fields, \(\text{dim}\) = dimension, and \(d\) = interpolation degree.
- For a problem with 10 fields, polynomial degree 4 and 3 dimensions, this is approximately \((10 \times 5^3)^2 \approx 1.5 \times 10^6\) components. If each number is stored in 16 bytes (double precision) this comes to approximately 25 MB.
- Each compute node in BEACON has 256 GB of memory and 16 cores, which comes to 16 GB per core. Problems with a large number of fields or high interpolation degree can strain this memory restriction.
- Most problems only have a few chemical interactions, which gives the Jacobian a sparse or banded structure that can be split among processors using TRILINOS.
Finite Element Method

Continuous vs. Discontinuous Galerkin Method

\[ \frac{\partial u}{\partial t} = \nabla^2 u \]

Continuous

\[ \sum_m \int_{\Omega_m} v \frac{\partial u}{\partial t} \, dx = - \sum_m \int_{\Omega_m} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} \, dx + \text{B.C.} \]

Discontinuous

\[ \sum_m \int_{\Omega_m} v \frac{\partial u}{\partial t} = - \sum_m \int_{\Omega_m} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} \, dx + \text{B.C.} - \sum_{m=1}^{M-1} \left[ \frac{\partial u}{\partial x} \right]_{x_m} \]

Code Finished

Working on
Serial Code (C):

\[ \frac{\partial u_{(\alpha)}}{\partial t} = d_{(\alpha)} \frac{\partial^2 u_{(\alpha)}}{\partial x^2} + R_{(\alpha)} (u) \]

- 1-dimensional multiple-species diffusion equation with source term
- Method:
  - Spectral Element Method
  - Guassian-Lobatto Quadrature
  - Euler Backward Method and Newton Method

Parallel Code (Fortran):

\[ \frac{\partial u_{\alpha}}{\partial t} = D_{\alpha} \nabla^2 u_{\alpha} - \vec{v} \cdot \nabla u_{\alpha} + R_{\alpha}(u) \]

- Fortran modules designed to solve multi-dimensional PDEs with Spectral Element Method
- module HESIOD:
  - Stores data about mesh, fields and equation
- module HOMER:
  - Performs time-integration on each element separately (parallel)
  - Resolves boundary discontinuities via weighted sum (serial)
- Currently tested on:
  - 1- and 2-dimensional multiple-species diffusion equations with convection and source terms
  - Forward Euler Method, no splitting
- **Testing Example:**
  - chemical equation: $Cl_2 \rightleftharpoons Cl + Cl$
  - 2-species math model:
    \[
    \begin{align*}
    \frac{\partial [Cl_2]}{\partial t} &= d_1 \frac{\partial^2 [Cl_2]}{\partial x^2} - 0.001[Cl_2] + 0.05[Cl]^2 \\
    \frac{\partial [Cl]}{\partial t} &= d_2 \frac{\partial^2 [Cl]}{\partial x^2} + 0.002[Cl_2] - 0.1[Cl]^2
    \end{align*}
    \]

- **Initial Condition:**
  - value of $[Cl_2]$ and $[Cl]$:
    \[
    \begin{align*}
    [Cl_2] &= 2 - 2\cos(\pi * x) \\
    [Cl] &= \sin(\pi * x)
    \end{align*}
    \]

- **Coding**

```c
int spi=2;
double a=0.0;
double b=6.0;
int element=30;

```

- **Serial**

- **Parallel**

  domain : [ 0, 6 ]

  ```c
  for(i=0;i<n1;i++)
    u0[i*spi]=2*2*cos(PI*x[i*spi]);
  ```

  ```c
  for(i=0;i<n1;i++)
    u0[i*spi+1]=sin(PI*x[i*spi+1]);
  ```
- Initial Condition
  - T = 0

- Solution run by the serial code
  - T = 2.0
  - T = 4.0
  - T = 6.0
  - T = 8.0

- Compare with FVM
  - T = 4.0

SEM

FVM

FVM: solution given by Jian Sun (UTK)
Program Solution VS Theoretical Solution

- Differential Equation:
  \[ \frac{\partial u}{\partial t} = 0.1 \frac{\partial^2 u}{\partial x^2} + 0.1u \]

- Initial Condition:
  \[ u = \sin(\pi x) \text{ at } T = 0 \]

- Theoretical Solution:
  \[ u = e^{0.1-0.1\pi^2 T} \sin(\pi x) \]

Compare the solution at \( T = 10.0 \)
Program Solution     VS     Theoretical Solution

Compare the solution at T = 10.0
Convergence of the Solution

Fix the domain to be [0, 20] and Vary the number of elements from 13 to 70.
Convergence testing

Equation: \( \frac{\partial u}{\partial t} = 0.1 \frac{d^2 u}{dx^2} - 1.0u \)

Theoretical Solution:
\( u(x, t) = e^{-1.0t}(1 + e^{-0.1\pi^2t} \cos(\pi x)) \)
2D test case: $2A \Leftrightarrow A_2$

\[
\frac{\partial [A_2]}{\partial t} = 2.0 \nabla^2 [A_2] - \vec{v} \cdot \vec{\nabla}[A_2] - 500.0[A_2] + 80.0[A]^2
\]

\[
\frac{\partial [A]}{\partial t} = 2.0 \nabla^2 [A] - \vec{v} \cdot \vec{\nabla}[A] + 1000.0[A_2] + 160.0[A]^2
\]

\[
\nu_x = 50.0 \quad \bullet \quad \text{Tested on 5 x 5 (=25) elemental grid}
\]

\[
\nu_y = 0.0 \quad \bullet \quad \text{dx = 0.2, dt = 0.0001}
\]
2D test case: $2A \leftrightarrow A_2$

- **Serial**:
  - $t = 0$
  - $t = 0.002$
  - $t = 0.004$

- **Parallel**:
  - $t = 0$
  - $t = 0.002$
  - $t = 0.004$
Scalability

Tested previous example with 32 x 32 (=1024) element grid on 1, 2, 4, 8, … , 1024 processors.

- Minimum time at 512 processors
- Can be improved by parallelizing the weighted average process
- Computing time may be improved by adding operator splitting
Next Steps

- **Serial Code:**
  - Finish the current discontinuous Galerkin serial code and test the accuracy of this code
  - Write a discontinuous Galerkin code based on Dr. Chung’s (CUHK) algorithm
  - Combine the code of the discontinuous serial code with the parallel code with Trilinos
  - Hopefully, a discontinuous parallel code would be obtained

- **Parallel Code:**
  - Parallelize the averaging process on the boundaries by sending boundary data only to neighbors
  - Implement option for operator splitting to improve computational efficiency
  - Implement options for Implicit Euler and Newton methods
  - Allow user to set more general boundary conditions: Dirichlet, Neumann, Robin and Periodic
  - Interface with Trilinos to allow for higher dimensional problems and a large number of fields
Acknowledgements

- Thanks to the NSF, UT-K, and ORNL for making this experience possible
- Thanks to our mentors Dr. Kwai Wong, Dr. John Drake and Dr. Joshua Fu for guidance and helpful conversations!
- Thanks to coworkers for offering expertise