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.

Spectral Element Method

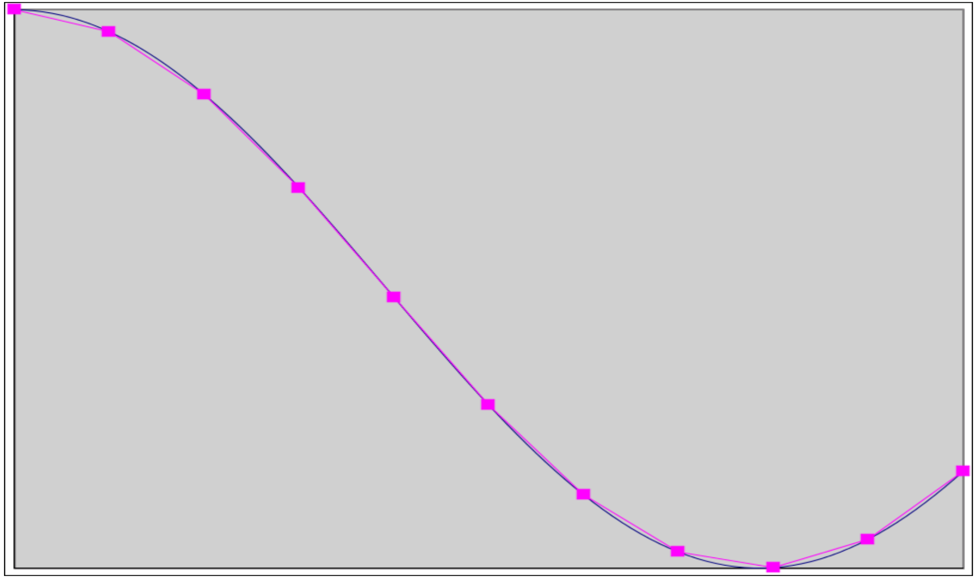
Basis Functions

- 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:

$$u(\vec{x}) \approx \sum_{\vec{\imath}} u(\vec{\xi}_{\vec{\imath}})\phi_{\vec{\imath}}(\vec{x})$$

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

$$\vec{\nabla} u(\vec{x}) \approx \sum_{\vec{\imath}} u(\vec{\xi}_{\vec{\imath}})\vec{\nabla}\phi_{\vec{\imath}}(\vec{x})$$



Spectral Element Method

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.:

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

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

$$\int_\Omega \phi_{\vec{i}} \frac{\partial u}{\partial t} dx = B.C. - \int_\Omega \vec{\nabla} \phi_{\vec{i}} \cdot \vec{\nabla} u dx \rightarrow \sum_{\vec{j}} \frac{\partial u(\vec{\xi}_{\vec{j}})}{\partial t} \int_\Omega \phi_{\vec{i}} \phi_{\vec{j}} dx = B.C. - \sum_{\vec{j}} u(\vec{\xi}_{\vec{j}}) \int_\Omega \vec{\nabla} \phi_{\vec{i}} \cdot \vec{\nabla} \phi_{\vec{j}} u dx$$

- The integrals are calculated using a Gaussian quadrature.

Spectral Element Method

Gaussian Quadrature

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

$$\int_{\Omega_m} f(\vec{x})dx \approx \sum_{\vec{k}} f(\vec{x}_{\vec{k}}) J_m(x_{\vec{k}}) w_{\vec{k}}$$

- 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:

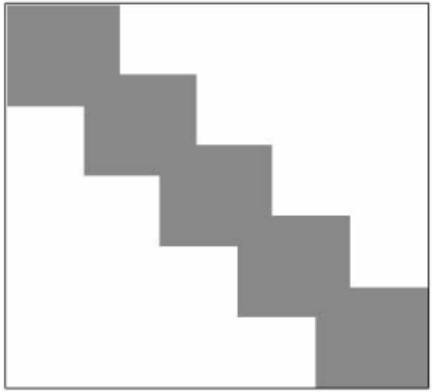
$$\int_{\Omega_m} \phi_{\vec{\imath}}\phi_{\vec{\jmath}} dx \approx \sum_{\vec{k}} \phi_{\vec{\imath}}(\vec{\xi}_{\vec{k}})\phi_{\vec{\jmath}}(\vec{\xi}_{\vec{k}}) J_m(\vec{\xi}_{\vec{k}}) w_{\vec{k}} = J_m(\vec{\xi}_{\vec{\imath}}) w_{\vec{\imath}} \delta_{\vec{\imath}\vec{\jmath}}$$

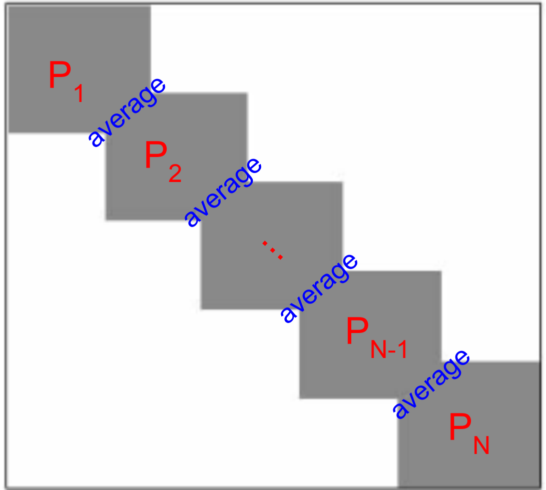
- This greatly simplifies the linear algebra involved.
- The Gauss-Lobatto quadrature (which SEM uses) is exact for polynomials of degree d <= $2n$-1, where n is the number of quadrature points. This means many vector-calculus identities are preserved in this formulation, which leads to local and global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

Spectral Element Method

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:

$$\frac{du_i}{dt} = F_i(u) \qquad u_i(t+\Delta t) = u_i + \Delta u_i$$
$$\Delta u_i = F_i(u)\Delta t$$

- 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:

$$u_i(t+\Delta t) = u_i + \Delta u_i$$

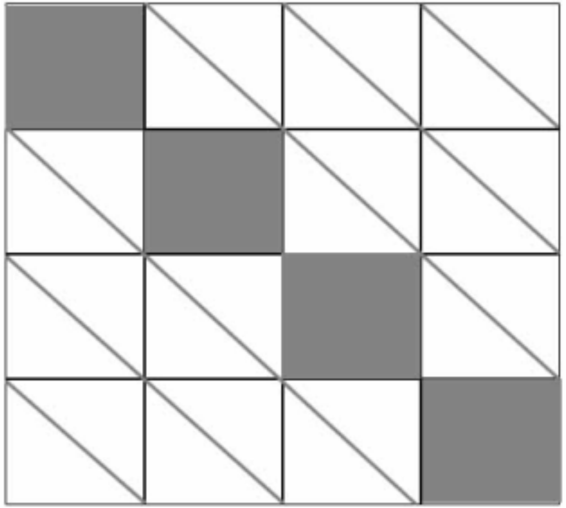
$$\Delta u_i = (1-\theta)F_i(u(t)) + \theta F_i(u(t+\Delta t))$$

- This is typically a nonlinear equation that must be solved via iterative Newton methods and the Jacobian matrix **J**:

$$\Delta u_i^{(n+1)} = \Delta u_i^{(n)} + \sum_j \frac{1}{\Delta t(1-\theta)} \mathbf{J}(u(t) + \Delta u^{(n)})_{ij}^{-1} \Delta u_j^{(n)} \qquad \mathbf{J}(u)_{ij} = \frac{\partial F_i}{\partial u_j}(u)$$

Spectral Element Method

Euler and Newton methods (cont.)



- The Jacobian matrix will have dimensions $(N_F \text{ x } (d + 1)^{dim})^2$, where N_F = number of fields, dim = dimension, and d = interpolation degree.
- For a problem with 10 fields, polynomial degree 4 and 3 dimensions, this is approximately $(10 \text{ x } 5^3)^2 \approx 1.5 \text{ x } 10^6$ 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.

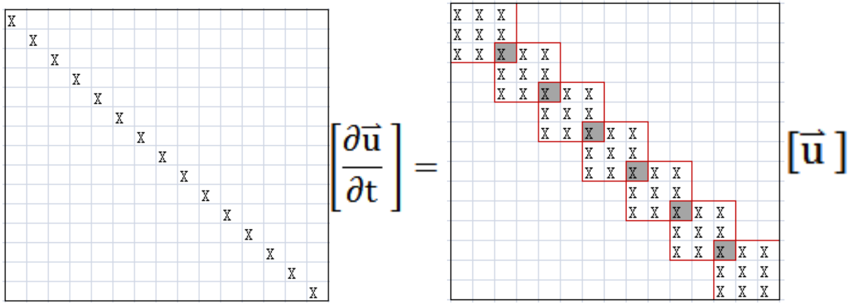
Finite Element Method

Continuous vs. Discontinuous Galerkin Method

$$\frac{\partial u}{\partial t} = \nabla^2 u$$

Continuous

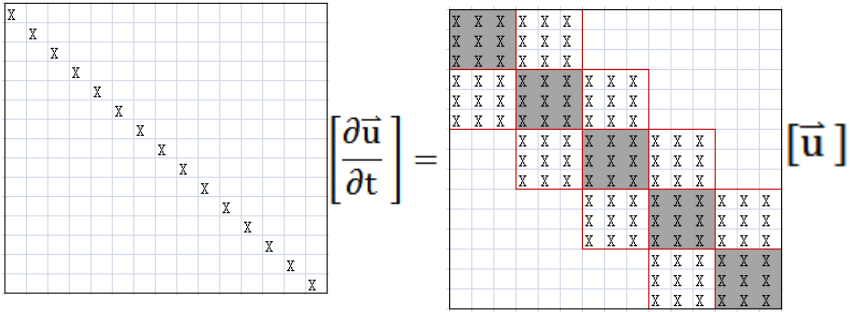
$$\sum_{m} \int_{\Omega_m} v \frac{\partial u}{\partial t} = -\sum_{m} \int_{\Omega_m} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} dx + B.C.$$



Code Finished

Discontinuous

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



Working on

Coding

- **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):**

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

  - 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

Serial

Parallel

Coding

- **Testing Example:**
  - **chemical equation:**

$$Cl_2 \rightleftharpoons Cl + Cl$$

  - **2-species math model:**

$$\frac{\partial[Cl_2]}{\partial t} = d_1 \frac{\partial^2 [Cl_2]}{\partial x^2} - 0.001[Cl_2] + 0.05[Cl]^2$$

$$\frac{\partial[Cl]}{\partial t} = d_2 \frac{\partial^2 [Cl]}{\partial x^2} + 0.002[Cl_2] - 0.1[Cl]^2$$

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

domain : [0, 6]

Serial

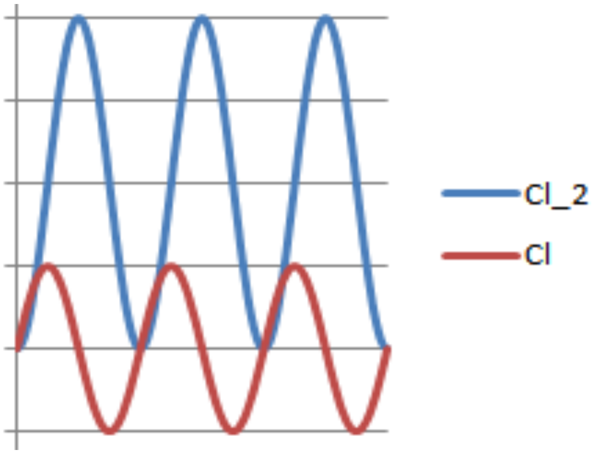
Parallel

- **Initial Condition :**
  - **value of [Cl_2] and [Cl]**

[Cl_2] = 2-2cos(PI*x)

[Cl] = sin(PI*x)

```
for(i=0;i<n1;i++){
    u0[i*spi]=2-2*cos(PI*x[i*spi]);
}
for(i=0;i<n1;i++){
    u0[i*spi+1]=sin(PI*x[i*spi+1]);
}
```

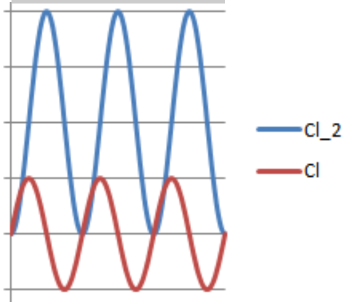



```
time=8.00, step=1000
0.000000e+00     0.000000e+00 0.000000e+00
3.015075e-02     4.114227e-02 1.823607e-04
6.030151e-02     8.225594e-02 3.642530e-04
9.045226e-02     1.233130e-01 5.452404e-04
1.206030e-01     1.642854e-01 7.248864e-04
1.507538e-01     2.051451e-01 9.027544e-04
1.809045e-01     2.458642e-01 1.078408e-03
2.110553e-01     2.864152e-01 1.251444e-03
2.412060e-01     3.267708e-01 1.421468e-03
2.713568e-01     3.669042e-01 1.588099e-03
3.015075e-01     4.067898e-01 1.751034e-03
3.316583e-01     4.464022e-01 1.909972e-03
3.618090e-01     4.857160e-01 2.064609e-03
3.919598e-01     5.247058e-01 2.214643e-03
4.221106e-01     5.633487e-01 2.359897e-03
4.522613e-01     6.016203e-01 2.500125e-03
4.824121e-01     6.394989e-01 2.635200e-03
5.125628e-01     6.769638e-01 2.765040e-03
5.427136e-01     7.139945e-01 2.889560e-03
5.728643e-01     7.505702e-01 3.008677e-03
6.030151e-01     7.866706e-01 3.122324e-03
6.331658e-01     8.222800e-01 3.230610e-03
6.633166e-01     8.573787e-01 3.333507e-03
6.934673e-01     8.919524e-01 3.431159e-03
7.236181e-01     9.259867e-01 3.523709e-03
7.537688e-01     9.594670e-01 3.611300e-03
7.839196e-01     9.923791e-01 3.694075e-03
8.140704e-01     1.024711e+00 3.772236e-03
8.442211e-01     1.056451e+00 3.846043e-03
8.743719e-01     1.087590e+00 3.915736e-03
9.045226e-01     1.118120e+00 3.981612e-03
9.346734e-01     1.148033e+00 4.043969e-03
9.648241e-01     1.177322e+00 4.103102e-03
9.949749e-01     1.205979e+00 4.159311e-03
1.025126e+00     1.234001e+00 4.212935e-03
1.055276e+00     1.261383e+00 4.264317e-03
1.085427e+00     1.288122e+00 4.313783e-03
1.115578e+00     1.314218e+00 4.361660e-03
1.145729e+00     1.339670e+00 4.408278e-03
1.175879e+00     1.364477e+00 4.453964e-03
1.206030e+00     1.388638e+00 4.499034e-03
1.236181e+00     1.412156e+00 4.543781e-03
```

Coding

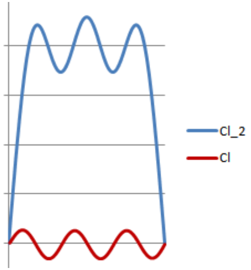
- **Initial Condition**

T =0

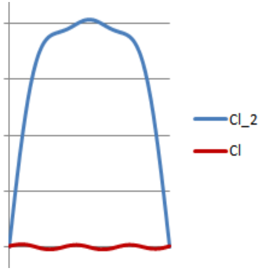


- **Solution run by the serial code**

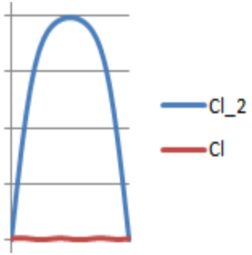
T = 2.0



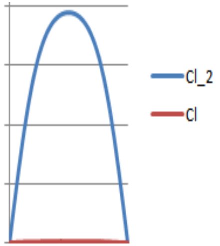
T = 4.0



T = 6.0



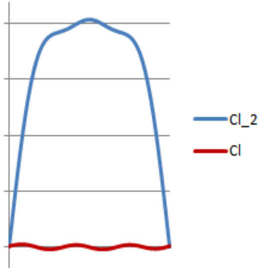
T = 8.0



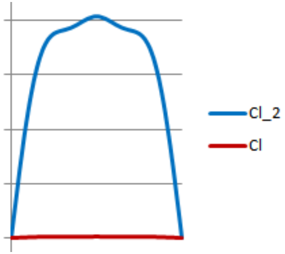
- **Compare with FVM**

T = 4.0

SEM



FVM



FVM : solution given by Jian Sun(UTK)

Serial

Parallel

Coding

Program Solution VS Theoretical Solution

- **Differential Equation :**

$$\frac{\partial u}{\partial t} = 0.1\frac{\partial^2 u}{\partial x^2} + 0.1u$$

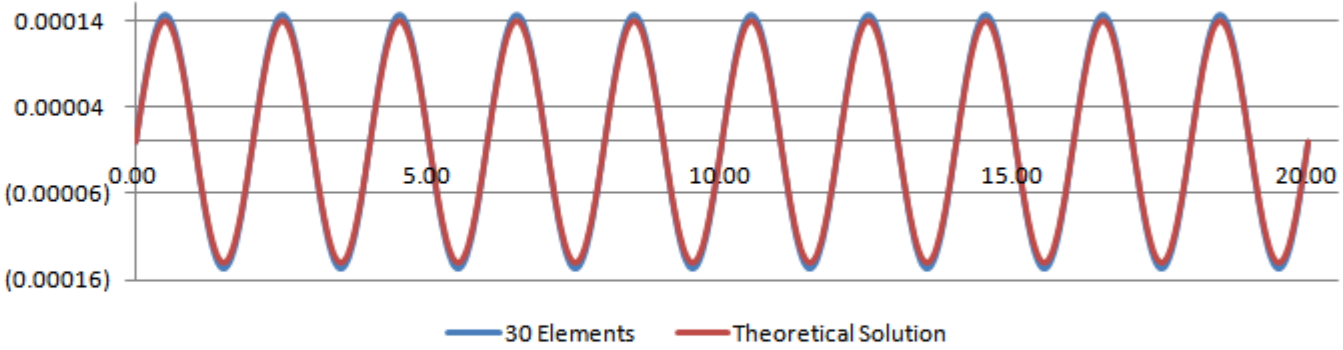
- **Initial Condition :**

$$u = \sin(\pi x) \text{ at } T = 0$$

- **Theoretical Solution :**

$$u = e^{0.1-0.1\pi^2 T}\sin(\pi x)$$

Compare the solution at T = 10.0



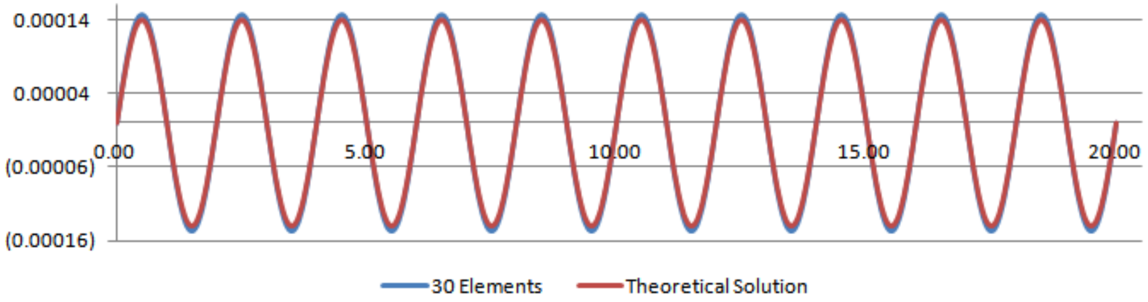
Serial

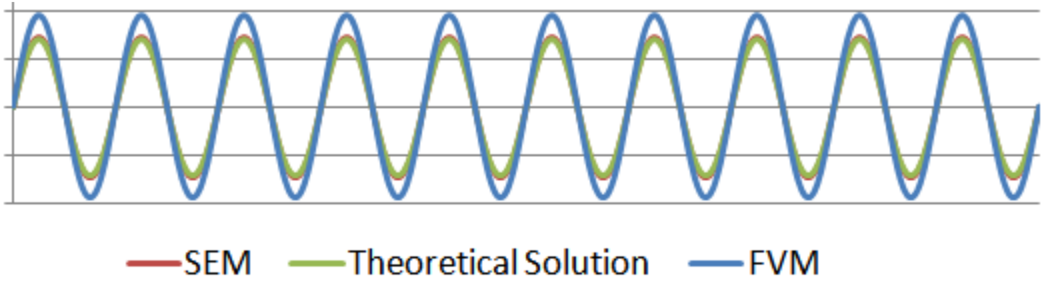
Parallel

Coding

Program Solution VS Theoretical Solution

Compare the solution at T = 10.0





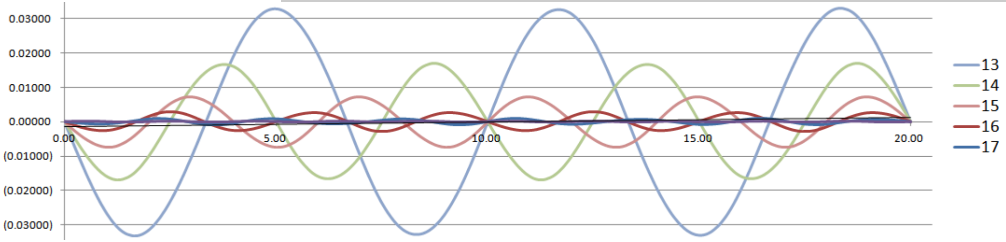
Serial

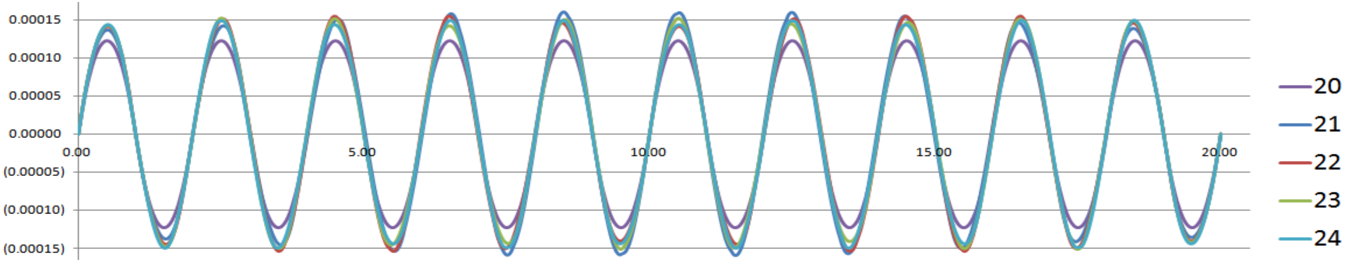
Parallel

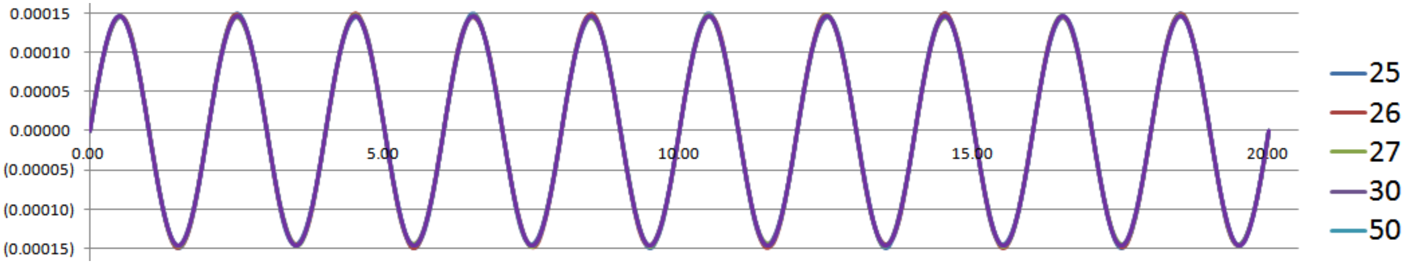
Coding

Convergence of the Solution

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







Serial

Parallel

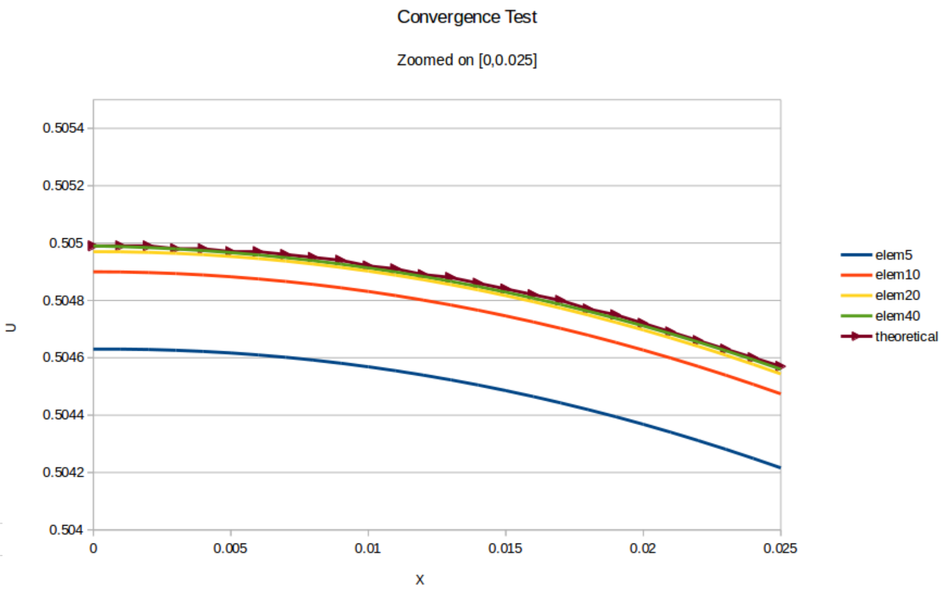
Coding

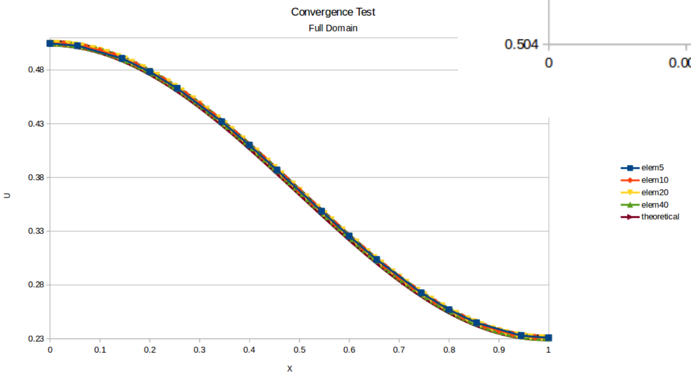
Convergence testing

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

Theoretical Solution:

$$u(x,t) = e^{-1.0t}(1 + e^{-0.1\pi^2 t}\cos(\pi x))$$





Serial

Parallel

Coding

2D test case: $2A \leftrightharpoons A_2$

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

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

$$v_x = 50.0$$

$$v_y = 0.0$$

- Tested on 5 x 5 (=25) elemental grid
- dx = 0.2, dt = 0.0001

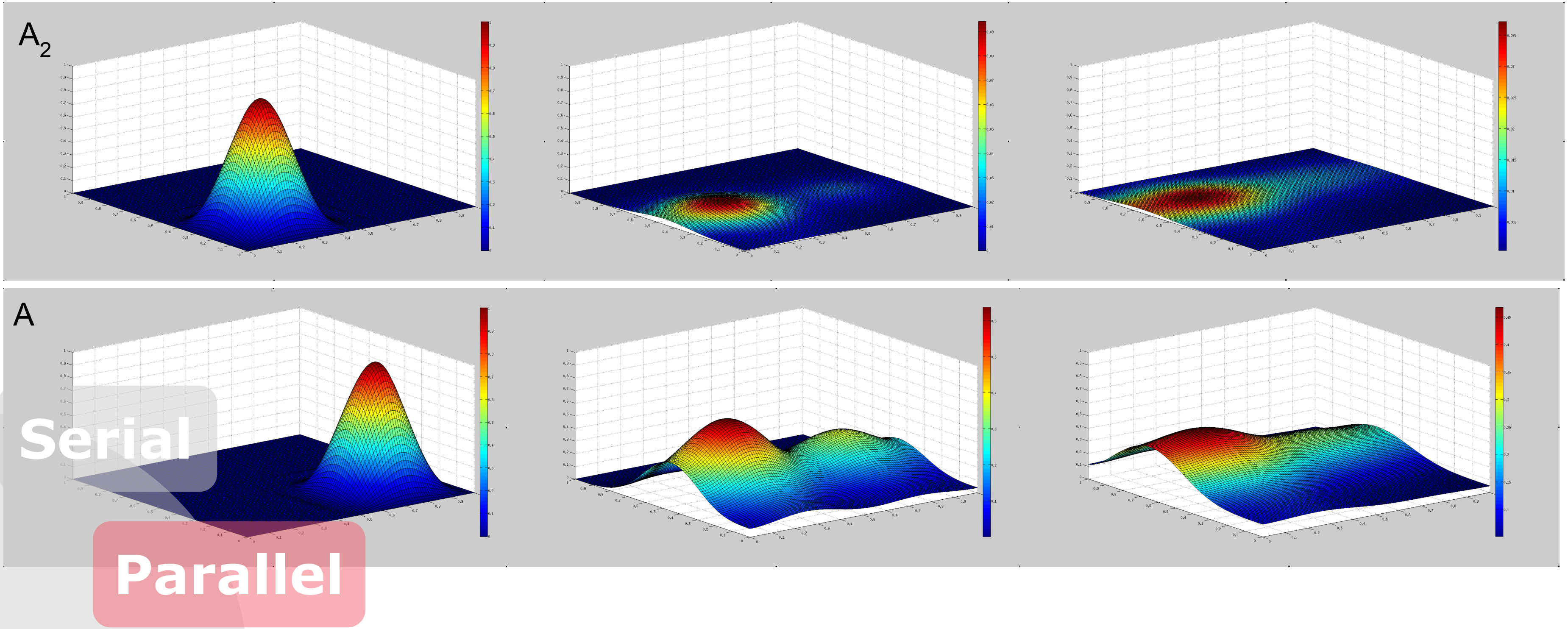
Serial

Parallel

Coding

2D test case: $2A \leftrightharpoons A_2$

t = 0 → t = 0.002 → t = 0.004 →



Coding

Scalability

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

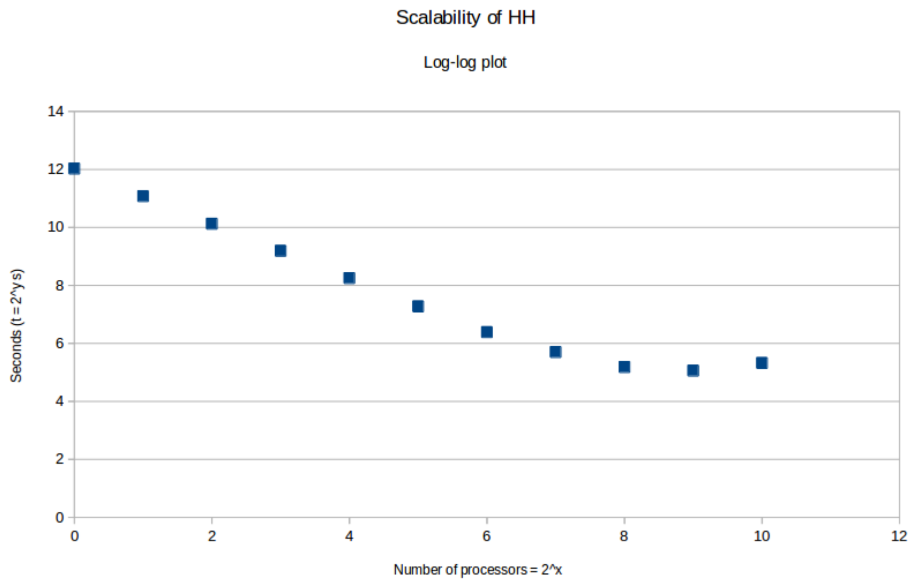
- Minimum time at 512 processors
- Can be improved by parallelizing

the weighted average process

- Computing time may be improved

by adding operator

splitting



Serial

Parallel

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:**
  - Parellelize 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

Serial

Parallel

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

- [1] Mark A. Taylor and Aime Fournier. A compatible and conservative spectral element method on unstructured grids. Journal of Computational Physics, 229(17):5879 – 5895, 2010.