Spectral Element Method: Schrödinger's Equation

CMSE 822

Amit Rotem

Summary

This project will implement a continuous Galerkin spectral element method for simulating the 2D Schröodinger equation:

$$\begin{gathered} iu_t = \Delta u + V(x,y)u, \qquad (x,y) \in \Omega, t \geq 0 \\ \mathcal{B}u = g, \qquad (x,y) \in \partial\Omega, t \geq 0 \\ u(x,y,0) = u_0(x,y). \end{gathered}$$

The time discritization will be implemented using an implicit SDIRK formula, so each step will require solving a large complex symmetric linear system. To solve this system, we can use MINRES or a modified conjugate gradient method (since the real part of the system is positive definite, there exist matrix splitting techniques for this type of problem).

The code will develop upon my existing implementation of the spectral element method for solving the elliptic PDE:

$$\Delta u + \lambda u = f(x,y).$$

The code is already MPI parallelized (using Boost MPI), however, I am interested in using OpenMP with GPU offloading as an alternative to distributed memory for smaller scale problems (hundereds rather than tens of thousands of elements), and comparing the results to the MPI implementation.

Parallelization Strategy

I will modify my code to use OpenMP instead of MPI,

This spectral element implementation is not adaptive, so there is no need to perform adaptive load balancing, however, it already performs a heuristic load balancing strategy on the data using quad trees. Since I am translating my code to shared memory parallelism, this load balancing method may be unnecessary.

The main operation that needs to be parallelized is the computation of Δu over the each element. An element consist $(p+1)^2$ nodes for which we approximate values of the polynomial approximation of u on that element where p is the order of the polynomial. The Galerkin formulation approximates these values by a system of linear equations which make use of the integral:

$$\iint_{\text{element}} v\Delta u \, dA.$$

This integral is computed using a quadrature rule defined on the $(p+1)^2$ nodes. Nodes on the boundary of each element, therefore, receive information from all connecting elements. Namely, nodes at an element edge recieve information from two elements connected on that edge, and nodes at an element corner can recieve information from multiple elements which depends on the quality of the mesh. In a distributed memory setting, this transfer of information between elements requires communication when two connected elements are not on the same processor. In a shared memory setting, the communication overhead is largely elliminated so long as processors are safely reading and writing data.

Validation and Optimization

I intend to compare the implementation with my MPI code and see if I can get a better speedup with GPUs.