18.337 Report - Parallel Spectral Elements for Nuclear Applications

Colin Josey

12 December 2015

1 Introduction

One method rarely used in analysis of a nuclear reactor is the spectral element method. The spectral element method is a method of solving partial differential equations in which the unknowns have been replaced with spectral polynomials. The actual derivation of this method is rather complex, but the method has numerous advantages. The most interesting advantage is the fact that the method can parallelize to extreme levels when properly implemented. The task then was to use spectral elements to analyze a reactor and try to approach this level of parallelism.

2 Problem Introduction

The specific problem of interest is the LRA rod ejection transient problem. The full problem is described in [1]. In this particular problem, there are 5 unknowns. The first two are ϕ_1 and ϕ_2, the neutron fluxes for energy group 1 and 2 respectively. These are solved with the neutron diffusion equation, Equation (1).

$$\frac{1}{v_i}\frac{\partial \phi_i}{\partial t} - \nabla \cdot D_i \nabla \phi_i = S(\phi_i) \tag{1}$$

where:

$$\begin{aligned} v_i =&\ \text{velocity of group } i \\ D_i =&\ \text{diffusion coefficient of group } i \\ S(\phi_i) =&\ \text{the neutron source due to flux in group } i \end{aligned}$$

The neutron diffusion equation can be most succinctly described as the Boltzmann transport with the angle of transport integrated in spherical harmonics, truncated to the linear anisotropic and isotropic terms, and the anisotropic term replaced with Fick's law of diffusion. A more thorough derivation is found in [2], page 518–523.

The next two variables, C_1 and C_2, are the concentration of delayed precursors. These are neutrons produced due to decay of fission products rather than directly through fission. Their inclusion massively changes the time evolution of the problem. Their population is described by Equation 2.

$$\frac{\partial C_i}{\partial t} = \beta_i \sum_j \left(\nu \Sigma_{f,j} \phi_j\right) - \lambda_i C_i \tag{2}$$

where:

$$\begin{aligned} \beta_i &= \text{ fraction of fission neutrons that appear in decay group } i \\ \nu \Sigma_{f,j} \phi_j &= \text{ neutron production rate from flux group } j \\ \lambda_i &= \text{ decay constant for decay group } i \end{aligned}$$

The final unknown is T, or temperature. The probability of a neutron colliding is a function of the vibration of the target nucleus, which is due to temperature. As such, the reaction rate will change with temperature. On properly designed reactors, this change is negative. For the LRA problem, the temperature is handled with adiabatic heatup, as shown in Equation 3.

$$\frac{\partial T}{\partial t} = \alpha \sum_j \left(\Sigma_{f,j} \phi_j\right) \tag{3}$$

where:

$$\begin{aligned} \alpha &= \text{ fission rate to heat conversion factor} \\ \Sigma_{f,j} \phi_j &= \text{ fission rate from flux group } j \end{aligned}$$

The transient involves the change of absorption cross section in 4 fuel bundles. This corresponds to a rod ejecting out of the top of the reactor. This rapidly increases the reaction rate, which heats up the fuel, which lowers the reaction rate. The result is a rapid peak in power followed by a drop and an almost steady state final condition.

3 Spectral Elements

My understanding of spectral elements comes pretty much exclusively from two sources. The first one, “Spectral element methods: theory and applications” [3] introduces the mathematics and quadrature methods commonly used. The second is the set of lecture notes of Paul Fischer [4] which detail practical implementation considerations.

To derive spectral elements, first one replaces all the unknowns of a cell with an orthogonal set of polynomials. Then, the PDE is cast into the weak formulation, with test functions being the same orthogonal set of polynomials. The problem is then integrated over the cell with a quadrature that is easy to use on said polynomials, and what results is a linear algebra problem.

Boundary conditions between cells are handled by boundary condition matrices. The boundary condition matrix maps a vector onto a cell. The matrix multiply is then applied cell-by-cell. The sum of the boundary result is then the final boundary condition. Effectively, an operator $\mathbf{A}$ is represented as follows:

$$\mathbf{A} = \mathbf{Q}^{\mathbf{T}}\hat{\mathbf{A}}\mathbf{Q}$$

In this, the matrix $\mathbf{Q}$ is the boundary matrix and $\hat{\mathbf{A}}$ is a block-diagonal matrix containing the spectral element discretization.

The spectral elements method has both advantages and disadvantages relative to other methods such as finite difference or finite elements. So long as all functions in the cell are smooth, spectral elements will converge at an exponential rate to the exact undiscretized solution. Conversely, a discontinuity will prevent this. The reactor described earlier has discontinuities of cross section between each fuel bundle. As such, to get the proper convergence, each fuel bundle is given its own cell. Secondly, almost all quadratures in use are 1D quadratures multiplied together to form squares, cubes, and hypercubes. The cell shape must be transformed into a cubic form. Some transformations may be poor choices and prevent convergence.

Finally, the key advantage is the fact that the $\hat{\mathbf{A}}$ part of a given operator $\mathbf{A}$ is block diagonal. Each matrix multiply can be done completely independent from one another. It is through this capability that the method can be parallelized.

4 General Algorithm

There are two phases required to perform a transient diffusion analysis. The first is to calculate the steady state solution before the transient. The algorithm used is given in Section 4.1. Then, once this initial condition is calculated, it is substituted into the transient equation and solved. This algorithm is given in Section 4.2.

4.1 Steady State

Starting with Equation (1), two changes are made. First, the time dependent component is set to zero, and second, the source is split into two. The first source, S_f is the fission source. The second source, S_{nf} contains all the non-fission components, such as absorption and scatter from one group to another. Due to approximations made either in the formation of the diffusion equation or in the raw data itself, there is no guarantee that the equation is truly steady state, even if the reactor represented is indeed stable. In order to correct for this, the fission source is multiplied by a ratio, $1/k$, in which k is the eigenvalue of the reactor. The result is Equation (4).

$$-\nabla \cdot D_i \nabla \phi_i + S_{nf}(\phi_i) = \frac{1}{k} S_f(\phi_i) \tag{4}$$

The resulting equation can be discretized and transformed into the following:

$$\mathbf{A}\phi = \frac{1}{k}\mathbf{M}\phi$$

Where A contains the diffusion, absorption and scatter terms, and the F term contains the fission generation terms. Multiplying both sides by k yields the general eigenvalue problem.

There are a wide variety of ways to efficiently solve the generalized eigenvalue problem. Due to practical concerns, only a few are common. For almost all algorithms, spectral elements included, $\mathbf{A}$ and $\mathbf{M}$ only exist as linear operators rather than matrices. In the majority of nuclear engineering programs, non-linear power iteration is used. The algorithm is shown in Algorithm 1. This method has two advantages. The first is that it is very easy to implement. The second is that it can be accelerated by solving for approximate eigenvectors which tweak ϕ closer to the correct result.

Algorithm 1 Non-linear power iteration as used in OpenMOC [5]

```
$\phi \leftarrow$ vector of norm 1
$k \leftarrow 1$
**while** not converged **do**
    $\phi \leftarrow (\mathbf{A} - \frac{1}{k}\mathbf{M})\phi$
    $k \leftarrow \frac{||\mathbf{M}\phi||_2}{||\mathbf{A}\phi||_2}$
    $\phi \leftarrow \frac{\phi}{||\phi||_2}$
**end while**
```

Other methods, far less commonly used but otherwise still possible, are linear power iteration, implicitly restarted Arnoldi (IRAM), and Jacobi-Davidson. Both linear power iteration and IRAM require inverting the $\mathbf{A}$ operator to form a normal eigenvalue problem ($k\phi = \mathbf{A}^{-1}\mathbf{M}\phi$). Jacobi-Davidson requires an approximate operator, such as an approximate inverse. From prior experience, Jacobi-Davidson will be fastest, followed by IRAM, and then followed by linear power iteration.

One would be inclined to start with Jacobi-Davidson, but due to some issues that occurred during program development, transient analysis was never successful. My goal was to demonstrate parallelism during a transient solve. As a compromise, instead, I made the steady state operate very similar algorithmically to the transient analysis. The transient method chosen is described in Algorithm 3. The linear power iteration in Algorithm 2 is structurally very similar, and should provide similar performance. The primary difference is the

Algorithm 2 Linear power iteration as used in the spectral elements code

```
$\phi \leftarrow$ vector of norm 1
$k \leftarrow 1$
while not converged do
    $b \leftarrow \mathbf{M}\phi^{(n)}$
    $\phi^{(n+1)} \leftarrow \mathbf{A}^{-1}b$, inverse performed by GMRES
    $k \leftarrow \phi^{(n)} \cdot \phi^{(n+1)}$
    $\phi^{(n+1)} \leftarrow \frac{\phi^{(n+1)}}{\left|\left|\phi^{(n+1)}\right|\right|_2}$
end while
```

matrix multiply by **M** is not required and temperature is not considered. Algorithm 2 is mostly sourced from `IterativeSolvers.jl` [6].

For this particular problem, the steady state solution looks like Figure 1 and Figure 2. With no transverse leakage, the eigenvalue is 1.000112 when solved with order 5.

4.2 Transient Analysis

Transient analysis is significantly more complicated for two reasons. First, the timescale of interest, typically 0.1 seconds, is much longer than the timescale for fast neutrons to propagate across the reactor, 10^{-5} seconds. Second, cross sections are non-linear functions of temperature.

To begin transient analysis, Equation 1 is discretized into the following form:

$$\begin{aligned}
\frac{\phi(t_1) - \phi(t_0)}{vh} &= \left(\frac{1}{k}\mathbf{M} - \mathbf{A}\right)\phi(t_1) \\
\phi(t_0) &= \left(\mathbf{I} + vh\mathbf{A} - vh\frac{1}{k}\mathbf{M}\right)\phi(t_1) \\
\phi(t_0) &= \mathbf{T}\phi(t_1)
\end{aligned}$$

In this form, time integration is performed by implicit Euler with a timestep of h. Implicit Euler was chosen due to the wide range of characteristic timescales in the problem. If integrated explicitly, far more timesteps would be required to prevent stability issues. $\phi(t_1)$ can be calculated by explicitly inverting the operator **T**.

For the temperature non-linearity, a simple iteration scheme is performed. At the end of each matrix inverse, the temperature is updated. This new temperature is used to reconstruct the operator **T**. The timestep is performed again. This is repeated until ϕ converges. As a result, the algorithm for transient analysis is given in Algorithm 3.

I was successful in implementing all the components of this algorithm. I had deliberately made it so that for the first second of the transient nothing would happen. As a consequence, steady state should be maintained. However, this

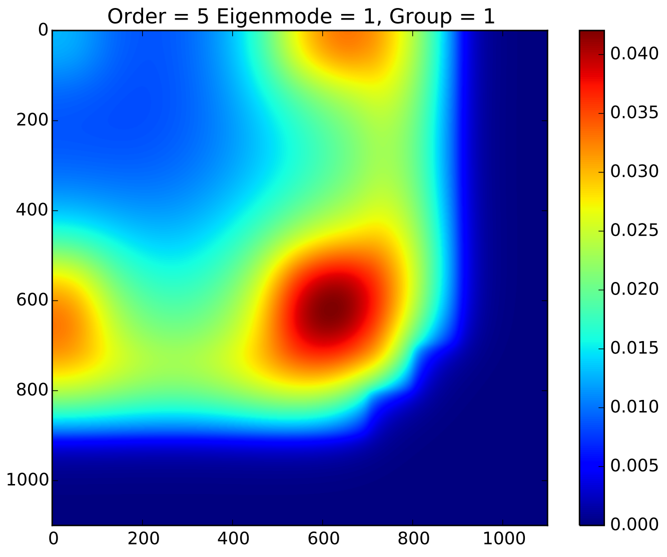

Figure 1: Group 1 flux primary eigenmode

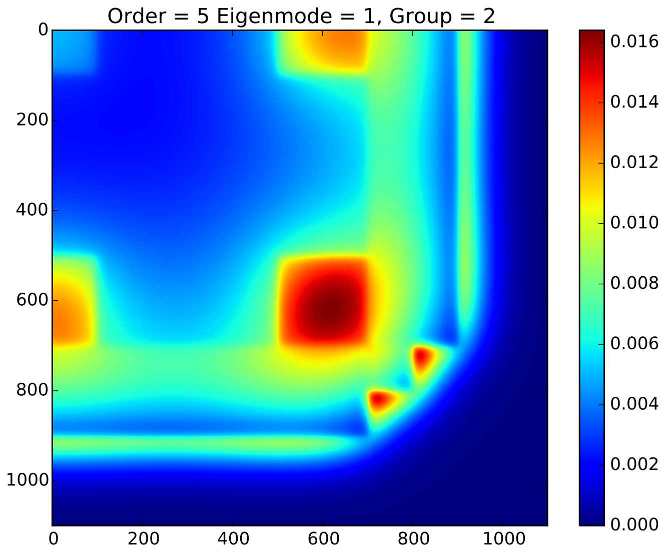

Figure 2: Group 2 flux primary eigenmode

Algorithm 3 Transient analysis as used in the spectral elements code. Delayed precursors ignored for clarity.

```
$\phi \leftarrow$ steady state result
$k \leftarrow$ steady state result
$T \leftarrow 300$ K
**while** not finished with time stepping **do**
    $\tilde{\phi}(t_{n+1}) \leftarrow \mathbf{T}^{-1}\phi(t_n)$, with $\mathbf{T}$ evaluated at $T(t_n)$
    Calculate $\tilde{T}(t_{n+1})$ from $\tilde{\phi}(t_{n+1})$ and $T(t_n)$
    **while** $\mathbf{T}$ not converged **do**
        $\tilde{\phi}(t_{n+1}) \leftarrow \mathbf{T}^{-1}\phi(t_n)$, with $\mathbf{T}$ evaluated at $\tilde{T}(t_{n+1})$
        Calculate $\tilde{T}(t_{n+1})$ from $\tilde{\phi}(t_{n+1})$ and $T(t_n)$
    **end while**
    $\phi(t_{n+1}) \leftarrow \tilde{\phi}(t_{n+1})$
    $T(t_{n+1}) \leftarrow \tilde{T}(t_{n+1})$
**end while**
```

did not occur for unknown reasons. Time constraints forced the modification of the steady state problem into a form that was functionally very similar to the above algorithm as mentioned prior.

5 Parallelization

In order to perform Algorithm 2, several components are necessary. These would be the implementation of the matrix vector product and simple vector operations such as `norm` and `dot`. The parallelization of these components could then be built out to construct GMRES for the matrix inverse. The GMRES method used is heavily based on the GMRES implementation in `IterativeSolvers.jl`, with a few performance issues fixed. When time allows, a pull request will be made to the repository to include these enhancements.

5.1 Matrix Multiply

As explained earlier in Section 3, a spectral elements operator will have an inner block-diagonal matrix and an outer boundary condition matrix. The inner part is trivial to parallelize (for each core, multiply the decomposed variable to get a decomposed result). The boundary condition is the tricky part. The process I used is as shown in Algorithm 4.

The result is that each cell has an "authoratative" value for each unknown in the problem, which eliminates the need to recompose and decompose the vector.

Algorithm 4 Matrix multiply.

```
$v_i \leftarrow$ spatially decomposed vector
$v_i$ is sent to each cell $i$ via MPI
$y_i \leftarrow \mathbf{M}_i v_i$
Boundary, $s_i$ is sent to cell $i$ from all neighbors of $i$
$y_i \leftarrow y_i + s_i$
```

5.2 Vector Operations

Vector operations are simplified due to the fact that each cell has an authoritative value. At the beginning of the simulation, a responsibility vector is created. This vector contains a “1” if this is the first cell that has said unknown, and “0” otherwise. This is used in the `norm` calculation as shown in Algorithm 5.

Algorithm 5 Norm calculation.

```
$b \leftarrow$ a zero vector which is as long as there are cells
$r$ is the responsibility vector
$v$ is the vector we want the norm of
for each cell $i$ do
    $b_i \leftarrow$ sumabs2($v$.*$r$)
end for
$b_i$ is broadcast from each cell $i$ to all other cells
$norm(v) \leftarrow \sqrt{\texttt{sum}(b)}$
```

This exact same layout can be used for dot products and sums. With these two components, a full GMRES implementation can be made and a steady state or transient analysis performed.

6 Results

The first set of parallel scaling tests were done on four nodes of a cluster. Each node had dual Intel E5420 CPUs operating at 2.5 GHz. The communication topology between nodes was 1Gbps ethernet. OpenMPI was used as the MPI implementation.

The first run had a 5th order problem, in which there were 72 unknowns per cell. Figure 3 shows the runtime. As shown, it appears that scaling almost followed the ideal curve on one node, but as more nodes were included performance fell apart. This most likely indicates that it takes more time to send the results between processes than it does to actually compute the results.

As such, a 10th order problem was run, such that there were now 242 unknowns per cell. The results are shown in Figure 4. For this problem, scaling on a single node was particularly poor, but the scaling between nodes improved.

My first inclination was that I had hit a memory bandwidth issue. The thinking behind this was that memory bandwidth is fixed on a per-node basis

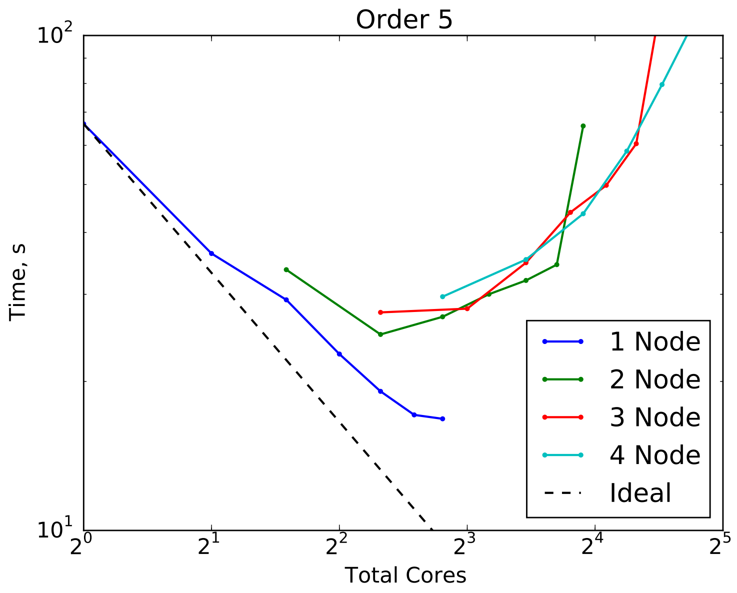

Figure 3: Time as a function of core count, order 5

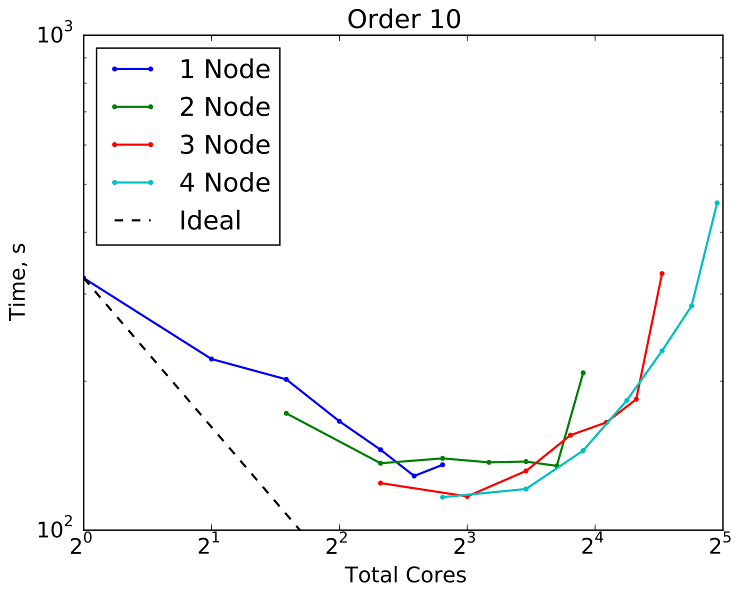

Figure 4: Time as a function of core count, order 10

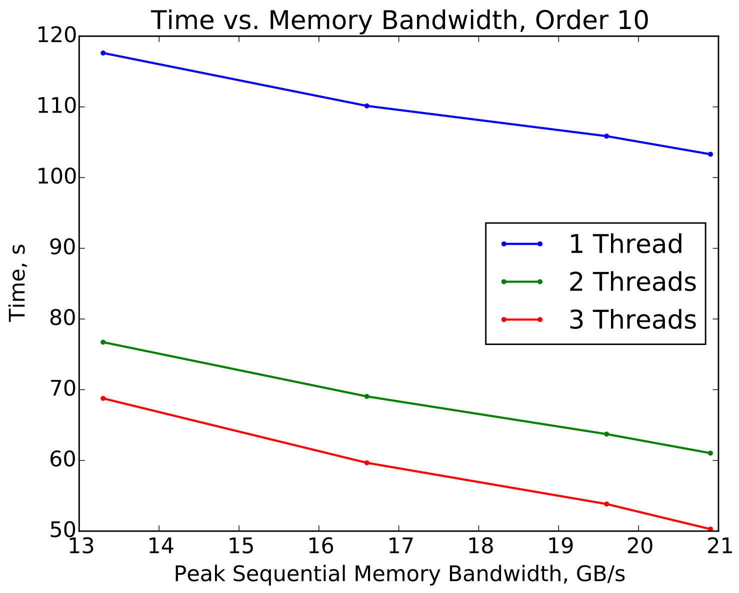

Figure 5: Memory bandwidth scaling

and does not scale with thread count on most architectures. Some architectures, like the Xeon Phi 2nd generation, cannot saturate memory bandwidth with any single thread regardless of what it was doing. That aside, another experiment was run on another computer in which the memory clockspeed could be varied. As such, the clockspeed was changed to all available stable configurations, with DRAM timing kept approximately constant to not affect latency. The results of this process are shown in Figure 5. The approximately linear scaling indicates that memory bandwidth is a significant limiter on current performance.

7 Conclusions

Overall, while the spectral elements method has many promising capabilities, this particular implementation was rather poor. While properly coded spectral elements can and has scaled to hundreds of thousands of cores, the upper limit for this program is probably 8. The algorithm is heavily limited by memory bandwidth, indicating that speedups could be had through de-vectorizing some code, as vectorization makes intermediate variables. Finally, I learned that MPI is probably not the best solution for Julia and should be avoided when reasonable. A prodigious quantity of boilerplate was written, even though the Julia MPI implementation was already rather concise.

References

[1] Argonne code center: Benchmark problem book, Tech. Rep. ANL-7416, Argonne National Laboratory (1977).

[2] A. Prinja, E. Larsen, General principles of neutron transport, in: D. Cacuci (Ed.), Handbook of Nuclear Engineering, Springer US, 2010, pp. 427–542. doi:10.1007/978-0-387-98149-9_5.
URL `http://dx.doi.org/10.1007/978-0-387-98149-9_5`

[3] F. Van de Vosse, P. Minev, Spectral elements methods: Theory and applications, Tech. rep., EUT Report 96-W-001 ISBN 90-236-0318-5, Eindhoven University of Technology (1996).

[4] P. Fischer, Paul fischer's homepage (2011).
URL `http://www.mcs.anl.gov/ fischer/`

[5] 3. eigenvalue calculations – openmoc documentation (2015).
URL `https://mit-crpg.github.io/OpenMOC/methods/eigenvalue_calculations.html`

[6] Julialang/iterativesolvers.jl (2015).
URL `https://github.com/JuliaLang/IterativeSolvers.jl`