MNP: MULTIPHYSICS NEUTRONICS PARALLEL, A PROOF OF CONCEPT FOR MORE EFFICIENT COMPUTATIONAL MODELING OF NEUTRONICS SYSTEMS WITH MULTIPHYSICS

Ethan Smith

Department of Aerospace and Mechanical Engineering
University of Notre Dame
Notre Dame, Indiana USA
esmith4@nd.edu

March 16, 2022

1 INTRODUCTION AND MOTIVATION

The neutron transport equation is an essential tool for modeling the complicated dynamics of nuclear reactor systems. This linear integrodifferential equation tracks the generation, removal, and movement of neutrons into a differential control volume.[1] Due to the extremely high dimensional space of these problems, computational models of this equation must rely on so called "accelerators" to generate workable results in a timely fashion, or alternatively work with non-universal approximations to the equation. Furthermore, the equation by itself does not account for the generation of delayed neutrons, which are born from radioactive isotopes generated by fission processes and are generated at a later time than "prompt" neutrons. Consequently, a system considering delayed neutrons will be critical at a smaller size than a system not considering delayed neutrons. Further, this delayed super-criticality will happen at a time orders of magnitude later than a prompt super-criticality, and indeed long after it would appear that the system had achieved a steady state. As such, it is an important feature for a neutronics code to capture.

The objective of MNP [2] is to implement an efficient and well organized solver that will generally handle setting up the discretized linear system and efficiently solving it. As a stepping off point, a simple 1D, multi-group in energy diffusion system with delayed neutrons was considered. Diffusion approximations to transport are generally accepted to be valid where it is employed consistently with its limiting assumptions. Frequently it is used to get a general understanding of neutronics systems before wasting time on very expensive transport codes. [3] Furthermore, MNP was written with a variable discrete operator in mind. Generally diffusion codes have a constant operator. A more complicated multi-physical coupling would involve thermal heat transfer and heat generation changing the many statistical parameters that determine the values of the discrete operator, and thereby changing the operator as the simulation progresses. Having the capability to easily handle this is an major improvement over my own solvers for this problem. The variable operator feature of MNP is used to optionally use a variable, logarithmic timestep.

I use a similar code in my ongoing research on estimating α, or time eigenvalues of neutron transport and diffusion systems using dynamic mode decomposition (DMD). To perform DMD, one must first run a numerical experiment. As an aside, DMD typically requires a constant time step, but a modified formulation of the method is employed in this research. The paper detailing this modified formulation, VDMD, has been accepted and is to be published in the near future. VDMD requires backwards Euler time stepping to work properly, hence, MNP uses backward Euler to discretize the time dimension.

The impetus for MNP is to speed up this tedius part of the process and allow for further complications to the system. The old code to generate diffusion data has extremely inefficient memory usage due to my own poor implementation of a function to build the discrete matrix. Furthermore, it lacks the portability, ease of use, and consistent input files implemented in MNP. The present implementation of MNP is less time-efficient than the older implementations of solving these systems due to overhead implemented to facilitate parallelization without actually implementing parallelization. MNP will soon prove to be an efficient code for solving these kinds of problems.

2 METHODS

The neutron transport equation has the form shown below.

$$\frac{1}{v}\frac{\partial \Psi}{\partial t} + \hat{\Omega}\cdot\nabla\Psi + \Sigma_T(\vec{x},E)\Psi = (\mathcal{S}+\mathcal{F})\,\Psi + \sum_{i=1}^{I}\chi_{d,i}(E)\frac{\lambda_i}{4\pi}C_i(\vec{x},t) \tag{1}$$

Where $v(E)$ is the scalar velocity of a neutron with energy E, Σ_T is the total cross section, a coefficient that describes the probability of a interaction per unit distance. The right most source term describes the effect of precursors, $\chi_{d,i}$ is the probability of neutron generation for precursor i at energy E. The population of delayed neutron precursor of "flavor" i, C_i, is modeled by the coupled differential equation.

$$\frac{\partial C_i}{\partial t} = \int_0^\infty dE\beta_i\nu\Sigma_f(E)\phi(\vec{x},E,t) - \lambda_i C_i(\vec{x},t) \tag{2}$$

The $\mathcal{S}$ and $\mathcal{F}$ operators are the scattering and fission operators respectively. These operators account for the fission of new neutrons and the scattering of neutrons into other directions and energy levels.

$$\mathcal{S}\Psi = \int_{4\pi} d\Omega' \int_0^\infty dE'\Sigma_s(\Omega'\to\Omega, E'\to E)\Psi(\vec{x},\vec{\Omega}',E',t) \tag{3}$$

$$\mathcal{F}\Psi = \frac{\chi_p(E)}{4\pi}\int_0^\infty dE'(1-\beta)\nu\Sigma_f(E')\phi(\vec{x},E',t) \tag{4}$$

Where $\Sigma_s(\Omega'\to\Omega, E'\to E)$ is the scattering cross-section describing the transfer of neutrons from direction Ω' and energy E' to direction Ω and energy E. $\chi_p(E)$ is a coefficient for the probability of a neutron promptly being generated at energy E, $(1-\beta)$ is the fraction of neutrons that are emitted promptly. Conversely, β_i is the fraction of neutrons which are emitted from delayed precursor flavor i, and $\sum_i \beta_i = \beta$. $\nu\Sigma_f(E')$ is the number of neutrons emitted from a prompt event multiplied by probability of such an event occurring at energy E'. ϕ is the scalar neutron flux, and can be calculated by the following:

$$\phi(\vec{x},E,t) = \int_{4\pi} d\Omega\Psi(\vec{x},\vec{\Omega},E,t) \tag{5}$$

The multi-group diffusion system can be arrived at by series of simplifying assumptions well explained in [3]. For the sake of brevity, the equation is presented below. Under the necessary assumptions, this is valid only for isotropic media. Symbology is consistent with the neutron transport equation. Subscript g refers to a discrete energy bin, subscript j refers to precursor of "flavor" j, D_g is the diffusion constant of energy group g.

$$\begin{aligned}\frac{1}{v_g}\frac{d\phi_g}{dt} - \nabla\cdot D_g\nabla\phi_g(x,t)\Sigma_{t,g}\phi_g(x,t) = \sum_{g'=1}^{G}\Sigma_{s,g'\to g}\phi_{g'}(x,t) + \\ \sum_{g'=1}^{G}\chi_{p,g'\to g}\nu_{p,g'}\Sigma_{f,g'}\phi_{x,g'}(x,t) + Q_g(x,t) + \sum_{i=1}^{I}\chi_{i\to g}\lambda_i C_i(x,t)\end{aligned} \tag{6}$$

and

$$\frac{dC_i}{dt} + \lambda_i C_i(x,t) = \sum_{g=1}^{G}\nu_{d,g,i}\Sigma_{f,g}\phi_g(x,t) \tag{7}$$

A system of algebraic equations can be arrived at if the spatial variable is treated with the finite volume method, and the time variable is treated with backwards Euler time discretization. Using these methods, one arrives at the following coupled systems.

$$\mathbf{A}\phi_{n+1,g} = \frac{\Delta t}{v_g}\phi_n, g + \sum_{j=0}^{J} \chi_{j\to g}\lambda_j \left(\Delta t + \lambda_j\right)^{-1} \frac{1}{\Delta t} C_j \tag{8}$$

where $\mathbf{A}$ is a block tridiagonal matrix. In the main diagonal blocks, the elements of $\mathbf{A}$ are computed by

$$\begin{aligned}\mathbf{A}_{\mathbf{g},\mathbf{g}'} = &\left(\frac{1}{v_g \Delta t} + \Sigma_t, g\right) - \sum_{g'=1}^{G} \Sigma_{s,g'\to g} - \\ &\sum_{g'=1}^{G} \left[\chi_{p,g'\to g}\nu_{p,g'} + \sum_{j=1}^{J} \chi_{j\to g}\lambda_j \left(\frac{1}{\Delta t} + \lambda_j\right)^{-1} \chi_{d,g',j}\right] \Sigma_{f,g'} + \Sigma_{J,L,g} + \Sigma_{J,R,g}.\end{aligned} \tag{9}$$

For the left and right off-diagonal blocks, these diagonal matrices are computed by

$$\mathbf{A}_{\mathbf{g},\mathbf{g}} = -1 \cdot \Sigma_{J,L,g} \tag{10}$$

for the left diagonal, or

$$\mathbf{A}_{\mathbf{g},\mathbf{g}} = -1 \cdot \Sigma_{J,R,g} \tag{11}$$

for the right diagonal. Σ_J is a function of the spatial relationship between the cell's surface area, volume, and the diffusion constant for the energy groups. As an example, the left Σ_J for an arbitrary cell can be computed by

$$\Sigma_{J,L,g} = D_g \cdot \frac{SA_L}{V} \cdot \frac{1}{dr} \tag{12}$$

where SA_L is the surface area of the left cell face, V is the volume of the cell, and dr is the width of the cell.

On the boundaries, reflectance terms are multiplied by Σ_J on the boundary, then added to the diagonal of the main diagonal block.

$$\mathbf{BCL}_\mathbf{g} = \left(\frac{1-\alpha_L}{1+\alpha_L}\frac{dr}{4D_g} + 1\right). \tag{13}$$

where α is the fraction of neutrons which are reflected back into the system at the boundary. For $\alpha = 1$, the boundary is totally reflecting and $\frac{\partial \phi}{\partial x}$ at that boundary will be zero.

For clarity, on the left boundary,

$$\begin{aligned}\mathbf{A}_{\mathbf{g},\mathbf{g}'} = &\left(\frac{1}{v_g \Delta t} + \Sigma_t, g\right) - \sum_{g'=1}^{G} \Sigma_{s,g'\to g} - \\ &\sum_{g'=1}^{G} \left[\chi_{p,g'\to g}\nu_{p,g'} + \sum_{j=1}^{J} \chi_{j\to g}\lambda_j \left(\frac{1}{\Delta t} + \lambda_j\right)^{-1} \chi_{d,g',j}\right] \Sigma_{f,g'} + \mathbf{BCL}_g \odot \Sigma_{J,L,g} + \Sigma_{J,R,g}.\end{aligned} \tag{14}$$

where $\odot$ indicates the Hadamard, or element-wise product.

3 IMPLEMENTATION

MNP is run by executing the following on the command line

```
$python wrapper.py *material.npz* *input.in*
```

from inside the installation directory. Material.npz and input.in are user defined material and simulation parameter files respectively and are specified in more detail in the included readme.txt. For a sample, users can simply use the files in the MNP/sample directory. There are two sample problems provided, a subcritical problem and a delayed supercritical problem. Examples presented will be using the subcritical

problem. Wrapper.py can be modified to change the output directory and what is saved, MNP defaults to saving the file phi_Hist.npz, containing the time grid and the time history of the neutron flux in every cell and every group, in the MNP/Outputs/ directory. MNP depends only on numpy [4] and numba [5]. If dependance on numba is unacceptable for the user, they can choose to delete the lines referring to it in delayedFunctions.py; the code will still work properly albeit slower.

MNP is organized into two main classes, grid and cell. The grid object is initialized with the simulation parameters and material properties and generates a list of cell objects. Methods of the grid can be called to update the solution for one time step, or run the simulation over the user specified time grid. Grid methods exist to run the entire simulation and either store every step of the output, or to store only the final flux state of the system. Other methods in the grid object include

The cell objects contain methods to store material data and build their own tridiagonal rows of the global matrix, and their own contribution to the global right-hand-side vector. In this way the code can be parallelized as each cell is capable of building the new local matrices and vector snippet independently. That is to say, the code has not been parallelized yet. Cell objects also contain methods to write their neutron flux and precursor concentration data to files independently, if so desired. The cell object was written in such a way as to facilitate future changes to the physical system. The methods to build the discrete matrices and right-hand side vectors can be re-written to accommodate a different set of diffusion equations. For the specific, previously discussed improvement of thermal physics coupling, the cell object needs only minor modification to accept a dynamic set of cross sections.

MNP was originally intended to be sped up by using the multiprocessing package in python. Parallelizing these implicit time stepping solutions could be seen as a sort of fool's errand, but the computational cost to update the local matrices before the assembly and linear solve step necessary to step forward in time is not negligible. Numba has been used to substantial benefit, but many of the methods cannot be trivially updated to implement numba. A test of the function used to build the local A matrices on the cell level indicates that a speedup of around 50× is achieved using Numba. The current grid methods that step forward must be re-written to contain external function calls which themselves would be accelerated with Numba. While this is not expected to be a severe undertaking, it is not trivial.

4 RESULTS

MNP is able to solve the neutron diffusion equation as formulated here without any loss of accuracy over the previous implementation. The code also runs with a much smaller memory requirement than the previous implementation of a diffusion solver. One consequence of the overhead associated with the object oriented architecture is a slow-down relative to the old implementation. It takes longer to pass the flux solution back and forth between the cells and grid objects than the previous implementation's simple Numpy vector allocation. It was anticipated that using the multiprocessing package would greatly speedup the run time of these portions of the code, however this has as of yet not been tested. The runtime of MNP is plotted against the old implementation of a diffusion solver in Figure 1. The old code was very memory inefficient, as such the maximum number of spatial cells that I could generate with a twelve group test problem was 500 before the program would run out of memory. MNP does not have this problem. Further, the most current implementation of MNP with some use of Numba has universally superior performance compared to the old diffusion solver.

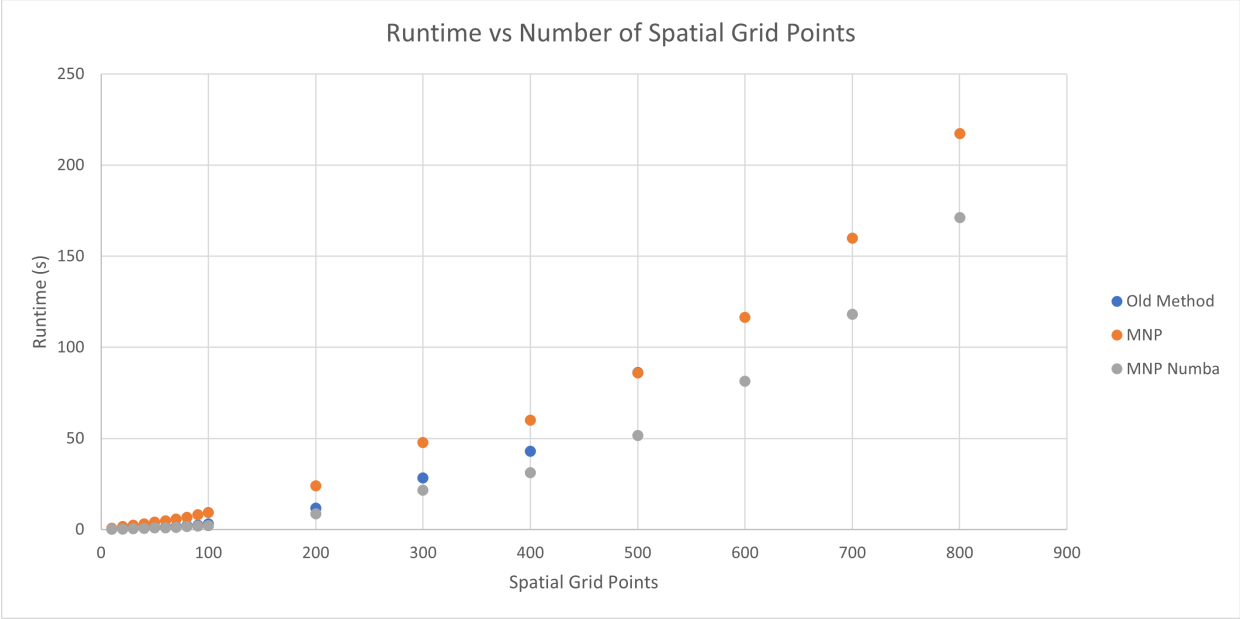

Figure 1: Plot of runtime for Old Code (Blue), MNP without Numba (Orange), and MNP with Numba (Gray). This data was recorded after one simulation of the sample problem with 100 time steps at each grid discretization level on an i7 12700kf desktop. MNP with numba is faster than the old code and does not have the extreme memory usage of that previous implementation.

MNP was tested against the output from the already verified prior implementation of the diffusion code. The results of this are plotted in Figure 2. This test problem is included in the sample directory. It is a subcritical slab problem with vacuum boundary conditions on both boundaries. Despite being subcritical, the problem appears to taper off to a steady state due to the delayed neutron source for a long period of time after the prompt neutron effects have concluded. This sample problem demonstrates the importance of variable time stepping used for the simulation; attempting to capture the early time dynamics of the prompt fission and decay as well as the eventual delayed decay with a fixed time step would require either multiple simulations or an extremely expensively fine time grid. For this problem, approximately 10^{13} time points would be required to capture the dynamics as described. The flux solution as calculated by MNP is plotted against the flux solution from the old diffusion code. This data is taken from the highest energy group at the left boundary of the problem, where the initial flux is set to the neutron speed. In effect, there is 1 neutron per cubic centimeter in every energy group on the left boundary initially.

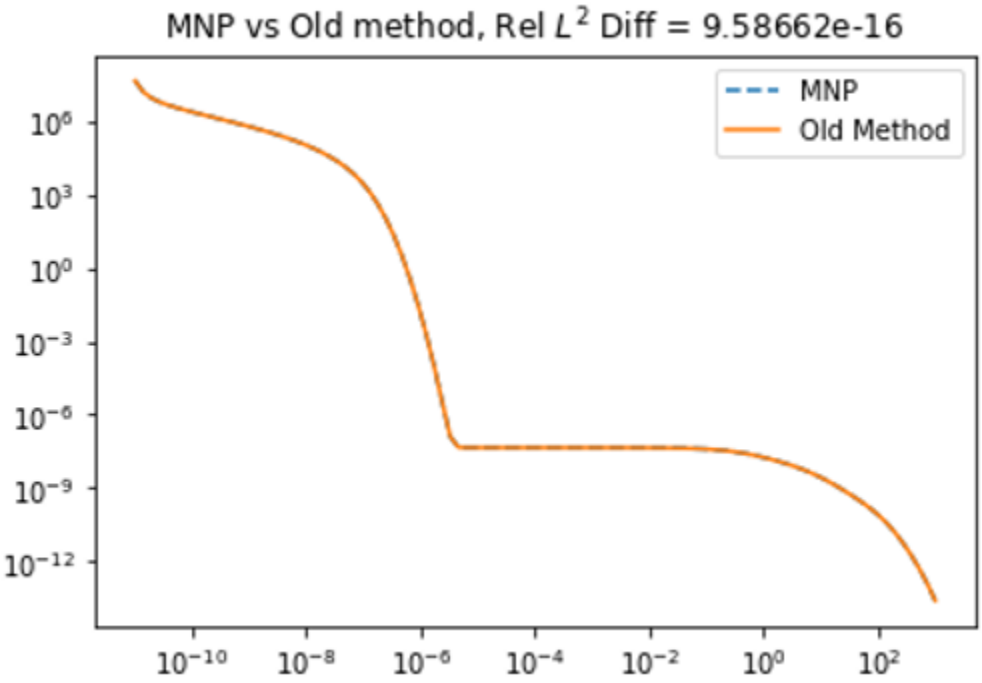

Figure 2: The output of MNP compared to the output of the old diffusion code. The relaive L2 difference between these two series is included in the title, highlighting the close accuracy of these two methods.

A major benefit that comes from MNP is that it is much easier to prepare an input deck and to run the code in a high performance computing environment through the use of job arrays iterating over several input decks. The use of a standardized input file system makes it relatively easy to prepare a suite of simulations that, for example, can span a wide variety of physical or simulation parameters.

5 CONCLUSIONS AND FUTURE WORK

In summary, MNP represents a significant improvement over the previous implementation of the diffusion solving code. Future work will see MNP implement thermal physics and parallelization. MNP was written with the foresight that these diffusion problems would approach the numerical expense of transport codes due to the high dimensionality of the coupled physics systems. MNP using Numba is faster than the old diffusion code, but its performance still could be improved through further leveraging the modularity of the computation. Of course, the global linear solve step is not simply able to be parallelized due to the implicit time stepping method necessary for the variable time stepping DMD method used in research. As such, performance gains achieved through further work towards parallelizing would ultimately be bottlenecked by the relatively expensive linear solve that needs to be performed at every time step. Nevertheless, there is still some potential for improvement.

The implementation of thermal physics is expected to be relatively straight forward, and should only require brief additions to handle the increase of dimensionality in the material input file. This is expected to be used for novel research into the dynamics of reactor systems by using DMD to compute time eigenvalues. The advantages afforded by MNP are significant compared to the previous, unorganized and memory inefficient version of the code that I had been using.

References

[1] G. I. Bell and S. Glasstone. *Nuclear Reactor Theory*. Robert E. Kreiger Publishing, Malabar, Florida (1970).

[2] E. L. Smith. "MNP." (2022). URL `https://github.com/SoftwareDevEngResearch/MNP`.

[3] T. Noh. "A Study on Diffusion Approximations to Neutron Transport Boundary Conditions." *Journal of Nuclear Fuel Cycle and Waste Technology*, **volume 16**, pp. 203–209 (2018). URL `http://www.jnfcwt.or.kr/journal/article.php?code=62571`.

[4] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy." *Nature*, **volume 585**(7825), pp. 357–362 (2020). URL `https://doi.org/10.1038/s41586-020-2649-2`.

[5] S. K. Lam, A. Pitrou, and S. Seibert. "Numba: A LLVM-Based Python JIT Compiler." In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15. Association for Computing Machinery, New York, NY, USA (2015). URL `https://doi.org/10.1145/2833157.2833162`.