School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University

PhD Thesis Proposal

Piper's Ponderings

Joanna Piper Morgan

Advisor: **Kyle E. Niemeyer**

September, 2024

Contents

Contents **i**

1 Introduction and background **1**
1.1 A brief overview . . . 1
1.1.1 Research Questions . . . 1
1.2 Neutron transport . . . 2
1.3 Deterministic Methods . . . 3
1.3.1 S_N approximation in angle . . . 3
1.3.2 Time-Space discretization schemes . . . 4
1.3.3 Source-Iteration . . . 5
1.3.4 One Cell Inversion-Iteration . . . 6
1.3.5 Acceleration schemes . . . 7
1.3.6 Novelness and relation to research questions . . . 8
1.4 Monte Carlo methods . . . 8
1.4.1 Portability Frameworks for Monte Carlo methods . . . 9
1.4.2 Delta tracking . . . 12

2 Objectives **13**
2.1 GPU enabled time dependent neutron radiation transport . . . 13
2.2 Explorations into GPU optimized deterministic schemes . . . 13
2.3 Exploration of acceleration schemes for One-cell inversion . . . 14
2.4 Exploration of delta tracking using a hybrid tally . . . 14
2.5 Possible difficulties and contingencies . . . 15

3 Proposed methods **16**
3.1 Deterministic transport . . . 16
3.1.1 Derivation of Higher order Methods in Iterative Schemes . . . 16
3.1.2 Fourier Analysis . . . 18
3.1.3 Verification using method of manufactured solution . . . 18
3.1.4 Implementation in Code . . . 19
3.1.5 Exploration of Acceleration Scheme . . . 19
3.2 Monte Carlo transport methods . . . 22
3.2.1 Python Based Portability for Rapid Methods Development . . . 22
3.2.2 Delta tracking in MCATK and MC/DC . . . 22

4 Preliminary Results **24**
4.1 Deterministic transport 24
4.1.1 Fourier Anylisys of Discretization Scheme 24
4.1.2 Verification of Discretization Scheme 24
4.1.3 Performance Results 24
4.2 Portable Monte Carlo solver 24
4.2.1 Comparison of Python-Portability Frameworks 24
4.2.2 Monte Carlo / Dynamic Code 24

5 Planned schedule **25**

A appendices **26**
A.1 Conference Publication, and Presentation for proposed investigations . 26
A.1.1 OCI Investigations 26
A.1.2 Hybrid-Delta Tracking Schemes 26
A.1.3 Acceleration and Abstraction of Python 26
A.2 All Publications 26
A.3 Submitted Draft: Performant and Portable
Monte Carlo Neutron Transport via Numba 26
A.4 Hybrid-Delta Tracking on a Structured Mesh in MCATK 39
A.5 Draft: Therefore paper 49

Bibliography **49**

1 Introduction and background

1.1 A brief overview

Predicting how neutrons move through space and time is important when modeling inertial confinement fusion systems, pulsed neutron sources, and nuclear criticality safety experiments, among other systems. Simulating these problems is computationally difficult using any numerical method, as the neutron distribution is a function of seven independent variables: three in space, three in velocity, and time. The equations of transfer can also be derived for raddiateive heat transport Modern HPC systems now enable high-fidelity simulation of neutron transport for problem types that have seldom been modeled before due to limitations of previous computers. Specifically, large scale, highly dynamic transport problems require thousands of compute nodes using modern hardware accelerators (i.e., GPUs) [1, 2].

My research is broad in its application but thematically specific, to satiate the ever-present need to go faster: faster and more accurate numerical methods, and faster implementations of numerical methods into HPC-worthy code. I will first present my research questions, then provide more introduction to demonstrate the gaps in knowledge they seek to probe. Then I will present my itemized research objectives and tasks required to implement those objectives. I will briefly describe the methods already implemented and more thoroughly describe methods yet to be done. Finally I will present preliminary results and draw initial conclusions.

This proposal is heavily supplemented by publications included as appendices. Explanations of methods and results is intentionally terse as to respect my committees' already generous time commitment and service. On that note I extend my most heartfelt gratitude to you and am keen and eager for the lively conversations sure to ensue about the work that I love.

with mutual respect,
Joanna Piper Morgan

1.1.1 Research Questions

1. Can relying on abstraction thru using software libraries enable non-expert users to produce efficient performing software for heterogenous computers?

2. Will a space-parallel deterministic iterative solution algorithm out perform the standard angle-parallel iterative algorithm on heterogenous architectures?

3. For deterministic neutron transport, does accounting for transient effects alter convergence rates of the iterative solution algorithms?

4. Can information coming from a streaming only problem be used to inform cell boundary information and increase convergence rates of a one-cell inversion iteration algorithm?

5. How can alternative Monte Carlo tracking schemes (namely Woodcock-delta tracking) be used be used to converge the QoI faster

1.2 Neutron transport

Transport equations are at their most fundamental—equations of continuity. Due to their ubiquity in nature transport equations are used to describe an incredible number of physical systems including fluid dynamics, chemical reactions, electro-magnetism, energy and heat. My research contained herein focuses on neutron radiation transport specifically, but could easily be applied to other neutral particles, most relevantly photons [3, 4]. In fact many of the methods of solution described in this document started as approximations for radiative heat transfer or photon transport. Assuming no neutrons are being produced by fission, the neutron transport equation (NTE) takes the form of an intergro-partial differential Boltzmann-type equation with seven independent variables [5]. As with any other continuity equation it is written as a set of sources (on the right) and sinks (on the left)

$$\frac{1}{v(E)}\frac{\partial\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)}{\partial t}+\hat{\boldsymbol{\Omega}}\cdot\nabla\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)+\Sigma(r,E,t)\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)= \int_{4\pi}\int_{0}^{\infty}\Sigma_s(\boldsymbol{r},E'\to E,\hat{\boldsymbol{\Omega}}'\to\hat{\boldsymbol{\Omega}},t)\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)dE'd\hat{\boldsymbol{\Omega}}'+s(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)\ , \quad (1.1)$$

where ψ is the angular flux, v is the velocity of the particles, Σ is the macroscopic total material cross section, Σ_s is the macroscopic scattering cross section, $\boldsymbol{r}$ is the location of the particle in three-dimensional space, $\hat{\boldsymbol{\Omega}}$ is the direction of travel in three-dimensional space, s is the source of new particles being produced, t is the time, and E is the energy of the particles for $\boldsymbol{r}\in V$, $\hat{\boldsymbol{\Omega}}\in 4\pi$, $0<E<\infty$, and $0<t$ [5]. We also prescribe the initial condition

$$\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},0)=\psi_{initial}(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}}) \quad (1.2)$$

and the boundary condition

$$\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)=\psi_{bound}(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)\text{ for }\boldsymbol{r}\in\partial V\text{ and }\hat{\boldsymbol{\Omega}}\cdot\boldsymbol{n}<0\ . \quad (1.3)$$

A number of additional implicit assumptions are needed for the validity of this equation and my research including: particle-particle interactions are rare and can be neglected, neutrons are points in space with no volume, collision events occur instantaneously, and nuclear properties are known [6]. This equation is commonly solved using both deterministic and Monte Carlo solution methods as analytic solutions are sparse.

1.3 Deterministic Methods

To recast the neutron radiation transport equation (eq 1.1) into a form solvable on digital computers continuous functions of space, time, angle, and energy must be discretized. Due to the nature of the NTE (intergro-PDEy-ness) an iterative scheme is also required. In this section I describe the discretizations and their assumptions for the 1D, time-dependent, multi-group solver proposed within this work. I will also describe the two fixed-point iterative schemes that I implement and discuss gaps in previous work of deterministic iterative solvers.

My work with determinsitic schemes into two catagories

1. How to use software engineering libraries to implement work more efficiently (RQ 1); and
2. Considering novel methods to converge the solution faster on modern hardware
   (a) A space-parallel iterative scheme on modern HPC GPUs (RQ2);
   (b) How transient behavior impacts convergence of a space-parallel iterative scheme (RQ3); and
   (c) How to accelerate the space-parallel iterative scheme to converge faster (RQ4)

1.3.1 S_N approximation in angle

The S_N or discrete ordinance approximation turns the introgro-PDE describing radiation transport into a coupled (simultaneous) set of linear PDEs in each angular direction. It was first formulated by Chandrasekhar to describe radiative heat transfer in stellar media [4]. The S_N approximation was soon applied to neutron transport by Carlson [7], Lee [8], and Lathrop [9]. The method of discrete ordinance was also adapted to general radiative heat transfer by Fiveland [10] and Truelove [11].

At this point it becomes nessacary to describe the other governing assumptions I use in this work including: slab geometry (1D-rectilinear coordinates), isotropic scattering and sources, as well as the multi-group assumption in energy distribution. The methods I propose in this work are not restricted by these assumptions—and future work may extend the methods described herein to be valid for anisotropic distributions in angle and/or solutions on unstructured meshes—-but are made here for simplicity. In-fact I make specific decisions in this work with the underlying discretization schemes to allow for eventual extension to these regimes.

When applied to equation 1.1 the resulting initialization point for my work is described by

$$\frac{1}{v_g}\frac{\partial \psi_{m,g}(x,t)}{\partial t} + \mu_m \frac{\partial \psi_{m,g}(x,t)}{\partial x} + \Sigma_g(x)\psi_{m,g}(x,t) = \frac{1}{2}\left(\sum_{g'=0}^{G} \Sigma_{s,g'\to g}(x) \sum_{n=1}^{N} w_n \psi_{n,g'}(x,t) + Q_g(x,t)\right) , \quad g = 1 \dots G , \quad m = 1 \dots N , \quad t > 0 , \quad x \in [0, X] , \quad (1.4)$$

where ψ is the angular flux, t is time, x is location in 1D space, g refers to the group, v is velocity, w_m is angular quadrature weight, μ_m is the angular quadrature ordinate ($\cos(\hat{\Omega}_\theta)$), m is the quadrature index, and Q is the isotropic material source. In this work I will exclusively use Gauss-legendre quadrature, but if extended to higher dimensions other quadratures sets would be required to evaluate the double integral over both angular directions (e.g. level-symmetric). The initial and boundary conditions are again prescribed angular flux distributions:

$$\psi_{m,g}(x,0) = \psi_{m,0}(x), \qquad m = 1 \dots N \ ,$$
$$\psi_{m,g}(0,t) = \psi_{m,L}(t), \qquad \mu_m > 0 \ ,$$
$$\psi_{m,g}(X,t) = \psi_{m,R}(t), \qquad \mu_m < 0 \ .$$

From here additional discretization can be used to turn the continuous functions of angular flux, source, material data, and differential operators into numerical approximations.

1.3.2 Time-Space discretization schemes

To enable this work in both space and time, discretization schemes are needed to treat the differential operators and continuous functions. As the goal of the research is for development on heterogenous architectures compute to work ratio may become an issue. Numerical algorithms that require lots of communication back and forth between the host (CPU) and device (GPU) will limit the maximum allowable performance for most GPU accelerators. To abate this issue higher (second) order discretization schemes can be used to add to the compute work required in every iteration, thus improving the compute to work ratio.

To treat the space discretization various classes of numerical discretization can be used. Finite difference, finite element, and finite volume methods are all often employed with the most common scheme being Diamond-differencing (O2). Diamond-differencing is popular as it is the only second order space discretization that uses a single interior evaluation point, however it only works on rectilinear grids. Intended future investigation of the scheme I develope in this research is into non-uniform meshes. While I will not include that analysis in this work, I do want to leave that opportunity open in order to better mesh (pun intended) with the modern state of the art of transport solvers which is on unstructured girds. Therefore, I used a family of finite volume schemes specifically designed for use on arbitrary-grids called corner-balance methods originally defined by Adams (1997) [12]. As they are a finite volume scheme they enforce conservation within a spatial cell. Simple corner balance was determined to be sufficient as it is higher (O2) order. Simple corner blance is not acrurate in the thin-limit however a slight alteration to it Simple corner balance is the same as lumped-linear discontionious a DFEM scheme. [13]

The transport equation requires some kind of iterative scheme to converge the linkage between scattering source and transport operators (this is discussed more in section 1.3.3). Thus implicit time marching schemes are normally used to time step as there is no added cost of the iterations they require. Specifically, implicit (backward) Euler (O1) or Crank-Nicolson (O2) are the most often employed. These schemes are

advantageous as they often act like a wrapper around a steady state deterministic transport solver with few changes needed to implement. While Crank-Nicolson is second order accurate it is not robust and can produce negative solutions and spurious osculations. Time dependent multiple balance (TDMB) was derived by Variansyah, Larsen, and Martin (2021) [14] to be a robust higher (second) order time discretization scheme. TDMB is developed from the "multiple-balance" scheme which is TDMB's intended use is for multiphysics shcemes and again as the idea is desnging methods When implementing a higher order method to allow for greater work-to-communicate ratios I decided to use this scheme.

1.3.3 Source-Iteration

The transport equation requires some kind of iterative scheme to converge the linkage between scattering source and transport operators. The source iteration (SI) method is commonly used to do this, often accompanied by preconditioners or synthetic accelerators, where the contribution to the solution from the scattering source (summation in the RHS of equation 1.4) is allowed to lag, while the angular flux is solved in every ordinate via transport sweeps through the spatial domain [12]. SI sweeps in Cartesian geometries are readily parallelized over the number of angles, as the source term is known from the previous iteration, allowing the angular flux in each ordinate to be computed independently. While any parallelization is a boon to performance, a scheme that is embarrassingly parallel over the dimension with the greatest number of degrees of freedom—space—may be advantageous. In a single spatial dimension SI is *annoying serial* in space and cannot be parallelized.

In higher spatial dimensions, many S_N production codes that implement SI use some kind of wavefront marching parallel algorithm also known as a Kockh-Baker-Alcouff scheme [15] also called "full parallel sweeps" in literature. In this scheme a sweep begins in a spatial location where all cell dependencies are known from boundary information (e.g. a corner). From there on a hypothetical 2D grid, the two nearest neighbor cells are now able to be computed independently, the next step would be 4 cells. This diagonally expanding wavefront continues to march and is able to better parallelize as many cells spatially as possible eventually saturating the number of work threads if the problem is large enough. On CPUs this has been shown to be performant but this changing amount of work is not optimal on GPUs where KBA algorithms are also tricky to efficiently implement in domain decomposed where wave front propagation between boundaries can be tricky. While this work is concerned with 1 spatial dimension when analyzing the state of the art it is important to consider that this is done.

This has proven successful in modern transport applications on CPUs (e.g., PARTISN, which implements the Koch–Baker–Alcouffe or KBA algorithm). The state of the art in deterministic S_N radiation transport is multi-group in energy distribution, diamond differencing first order space discretizations or other FEM FVM schemes on unstructured meshes, backward euler time stepping, domain decomposition via parallel block Jacobi, wave front marching schemes like KBA in higher spatial dimensions within a subdomain, and source iterations with diffusion synthetic acceleration (DSA) and potentially accompanied by GMRES solvers. Some examples of production codes that implement this are Partisn, Ardra, Minerate, Capsaicin, Denovo, Silver Fir, etc.

Here again this is included as a Each one of these points becomes more difficult to implement due to angle-parallel SI with KBA, for example domain decompisition

Published work on what schemes are implemented and how on GPUs is sparse. Most of my understanding of what is currently used comes from conversations held at conferences with the developers of the codes themselves. That is all to say that while I make every attempt to have as detailed a review of what is implemented I am almost certainly missing context. With that grain of salt we can move forward.

1.3.4 One Cell Inversion-Iteration

One Cell Inversions (OCI) (also called cell-wise parallel block Jacobi) is an alternative to SI where all angular fluxes in all ordinates and groups within a cell are computed in a single linear algebra step. It assumes that the angular fluxes incident on the surfaces of the cell are known from a previous iteration. OCI allows for parallelizing over the number of cells as each cell is solved independently of the others in a parallel.

Rosa Warsaw and Perks (2013) [16] previously investigated OCI as a potentially superior iterative scheme over SI on vectorized heterogenous architectures. They supposed that because of the parallelism over the dominant domain, inherit data locality, ability to take advantage of LAPACK type libraries, and highly floppy operations present in an OCI type algorithm, it might out-perform an SI based implementation in wall clock runtime. The study was conducted on state of the art (at the) time RoadRunner super computer at Los Alamos National Lab and took advantage of its 64 bit PowerXCell vectorized accelerator—a precursor to GPGPUs seen in HPCs today. Rosa et. al. implemented OCI in a 2D, multi-group, steady state, code using first order diamond differencing to discretize space, a multi-group scheme in energy distribution, and parallel block Jacobi and Gauss-Sidle iterations.

The authors concluded that the acceleration seen per-iteration in OCI was not enough to make up fo the decay in convergence rate that OCI incurs. Because there is no communication of information between cells within an iteration, OCI can require more iterations to converge to a solution for some of problems. Specifically, as cellular optical thickness goes down, OCI's relative performance degrades. Figure 1.1 illustrates this behavior, showing the spectral radii of the two iteration schemes as a function of cell thickness (in mean free path) and the scattering ratio. These values were computed numerically from an infinite medium problem (via reflecting boundaries) using steady-state calculations in S_4. The smaller the spectral radius, the faster a method is converging. The spectral radius for SI depends linearly on the scattering ratio, and for problems that are many mean free paths in size, it is nearly independent of cell optical thickness. The spectral radius of OCI decreases substantially as the optical thickness of the cells increases. Rosa et. al. also suggested that future developments in GPGPU accelerators might overcome this convergence rate decay. The idea there as that even though more iterations will be required to converge a solution those iterations will be able to be done sufficiently faster (in wall-clock-runtime) to mean a solution is computed faster (again in wall clock runtime) then source iterations.

Other investigations have explored OCI as an acceleration scheme for SI [17, 18] and a solution to the integral transport matrix method [19] and in the inexact parallel jacobi scheme. Previous investigations of OCI as an iterative scheme have been limited

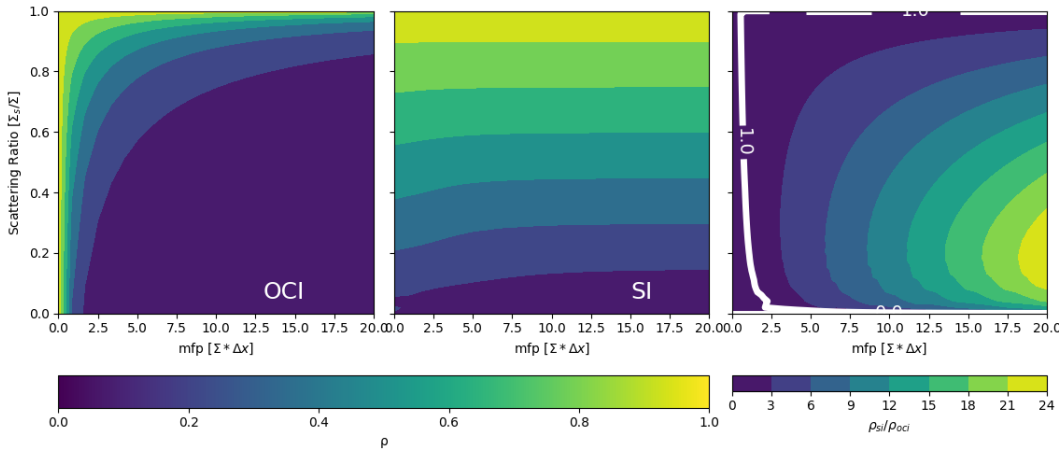

Figure 1.1: Spectral radii ($\boldsymbol{\rho}$) of OCI (left) and SI (middle) and the ratio between the two (right), where $\boldsymbol{\Sigma}$ is the total cross section, $\boldsymbol{\Delta x}$ is the cell width, and $\boldsymbol{\Sigma_s}$ is the scattering cross section

to steady state computations.

Regardless of the time stepping method an OCI iterative algorithm might come with some added befits when used in a transient scheme. Returning to OCI's spectral radius shown at left in Fig. 1.1, since both dimensions are governed by relationships with the cross section of the cell (Σ), altering that value will impact convergence behavior. As the scattering ratio decreases, both iterative algorithms requirer fewer iterations to converge. However, the spectral radius of OCI also decreases with increasing optical thickness, *which is an added benefit.* When solving optically thick and highly scattering problems, small increases in Σ may drastically improve the relative performance of OCI in comparison with SI. Time step and cellular optical thickness are inversely proportional to each other, meaning a smaller time step will yield a larger effective total cross section, thus theoretically improving the spectral radius. This behavior is not expected to happen in source iterations as SI does not directly depend on cellular optical thickness—behaving linearly in that dimension.

1.3.5 Acceleration schemes

In order to converge iterative schemes faster many acceleration techniques have been explored and implemented over the years [20]. The most common used in conjunction with source iterations is diffusion synthetic acceleration (DSA) which converges the slowest mode—the so called diffusive-limit [20]. The failure of SI in the diffusive limit can be physically understood as there is no linking between angles within an iteration and a diffusive problem is a coupling of the angles. So DSA adds a mid-step correction term based off of a cheap diffusion approximation of transport effects. Other synthetic acceleration techniques include quasi-diffusion, Boundary projection acceleration Synthetic acceleration techniques are actually special cases of preconditioners.

Much less work has gone into exploring synthetic acceleration techniques for one cell inversions. The slowest mode of convergence of OCI is known to be the thin limit due to the a-synchronicity of the scheme (lagging cell-wise boundary fluxes) [18]. The

spectral radius will decay unbounded in the thin limit, regardless of scattering ratio [16]. Rosa and Warsa (2009) investigated using transport synthetic acceleration (TSA) as the low order synthetic accelerator via fourier analysis [21–23]. TSA uses a low order transport sweep with potentially fewer angles (when accompanied by SI) and a smaller scattering ratio (often controlled by the user by term $\beta \in [0, 1]$) to inform the error correction equation [24]. It is just a less expensive transport sweep, but a transport sweep none-the-less and comes with all the baggage I am trying to avoid over a transport sweep. Rosa and Warsa say as much when they suggest a scheme where TSA is only "turned on" after a given number of iterations fails to converge to a solution [21]. TSA While they suggest future work to implement, no published results of an OCI+TSA scheme.

Both SI and OCI are forms of a fixed-point (or Richardson) iteration after some operator splitting. in a fixed-point iteration an initial guess of the angular or scalar flux is supplied as a known to the solver which in turn returns an angular or scalar flux. This becomes the new guess and is iterated until the relative error between the guess and the output falls to below a given tolerance [25]. Kyrlov methods on the other hand store multiple copies of the “guess" as to compute the next guess in a way that is orthogonal of the previous [26, 27]. Generalized minimal residual method (GMRES) specifically, has been shown to make DSA schemes that are inconsistent converge [28] They work by keeping copies of the guess for a given number of previous itteraties, compute risdiuls and ensure the next guess is orthogonal to the previous ones[29]. For OCI

1.3.6 Novelness and relation to research questions

The hypothesized additional benefit of of OCI in a time dependent code (RQ 3) coupled with the vast improvements in GPGPU accelerators in the past decade since Rosa et. al. [16] conducted their investigations (RQ 1 and 2), along with the ability to explore higher order spatial and transient discretization schemes (RQ 2) motivates this work. Transients schemes coupled to an OCI based solver has not previously been explored. OCI on modern GPUs has also not been investigated. TDMB has not previously been implemented with SCB

Furthermore, as previously mentioned, source iterations are often implemented in production accompanied by some kind of synthetic-acceleration scheme or other pre-conditioner to converge it’s slowest converging mode (the diffusive limit). OCI has never been implemented with a prevonditon OCI’s slowest converging mode is known to be the thin limit (RQ 4). Searching for a synthetic acceleration scheme

If these investigations prove fruitful that would go along way in identifying OCI as an iterative scheme for use on modern HPCs.

1.4 Monte Carlo methods

While deterministic schemes produce an exact solution to an inexact problem Monte Carlo methods provide an inexact solution —with an associated error—to an exact problem. Monte carlo methods can treat the independent variables of the NTE (space,

angle, time, energy) as continuos, thus eliminating the discretization errors seen with deterministic methods. The behavior of neutrons can be modeled with a Monte Carlo simulation, where psude-particles with statistical importance are created and transported to produce a particle history [6]. A particle's path and the specific set of events that occur within its history are governed by pseudo-random numbers, known probabilities (e.g., from material data), and known geometries. Data about how particles move and/or interact with the system are tallied to solve for parameters of interest with an associated statistical error from the Monte Carlo process. The analog Monte Carlo method is slow to converge (with a convergence rate of $\mathcal{O}(1/\sqrt{n})$ where n is the number of simulated particles). New Monte Carlo schemes could converge the solution faster in wall-clock time with fewer simulated particles and may be needed to effectively simulate some systems.

Monte Carlo methods where orignal propased by Stanislaw Ulam in 1946 after work on the Manhattan project. It's orginal use in fact was the tracking neutrons thru phase space using handheld computers called Fermiacs. The challenge problem I seek to investate also dates back this time. Since then from a report in 1995 over 60% of computational time on US Government super computing systems was concerend with converging the Monte Carlo method

My work with Monte Carlo schemes—as with my deterministic work—broadly is clavered into two catagories

1. How to use software engineering libraries to implement work more efficiently (RQ 1); and

2. Considering novel methods to converge the solution faster on modern hardware (RQ 5).

The first section digs into the first goal examining performance portability schemes in high level languages and the work that is currently deployed in Monte Carlo/Dynamic Code (MC/DC) as part of my work with the Center for Exascale Monte Carlo Neutron Transport (CEMeNT) The second contains work initially done in a production code at Los Alamos National Laboratory under the direction of Travis Trahan, PhD Timithoy Burke PhD and Collin Josey, PhD, and I propose extending the scheme in an implementation in MC/DC.

1.4.1 Portability Frameworks for Monte Carlo methods

In this section I intorduce inital investgations into high-level performance portability frameworks and discuss

Simulating these problems is computationally difficult using any numerical method, as the neutron distribution is a function of seven independent variables: three in space, three in velocity, and time. This problem demands HPCs and specifically the exascale Modern HPC systems now enable high-fidelity simulation of neutron transport for problem types that have seldom been modeled before due to limitations of previous computers. Specifically, large scale, highly dynamic transport problems require thousands of compute nodes using modern hardware accelerators (i.e., GPUs) [1, 2].

In recent years, many frameworks have appeared to solve portability issues, both between accelerator/HPC architectures and between different vendors of those architectures. Kokkos [30] is a portability framework developed by Lawrence Livermore National Laboratory. When developing with Kokkos, a user declares and allocates memory, writes appropriate compute kernels (in Fortran, C++, or with limited operability in Python [31]), and executes those functions with calls to the portability framework. When executed, the portability framework will make all architecture-specific API calls (i.e., `hipMemalloc`) independently of user-defined code.

Python is one of the most widely used programming languages, both generally and in academia. It is dynamic, untyped, and supports a rich development environment for data processing with core scientific and numerical packages like NumPy [32]. For GPU support, tools like Cython, PyCUDA, and PyOpenCL [33] can be used to compile and bind GPU functions—often written in `C++`—to Python code. Often called Python-as-glue, these schemes can help in simplifying a code base that targets multiple accelerators but still require as many separate compute-kernel sources as hardware targets. Different schemes have been implemented to alleviate source divergence for hardware targets when using Python-as-glue thus solving portability issue between hardware targets.

Many tools have been developed to solve portability problems using Python. One example is through using a domain-specific language (DSL) along with Python compiler tools to avoid needing a low-level language (e.g., FORTRAN, C). A domain-specific language is designed specifically to alleviate development difficulties for a group of subject-area experts. It can even abstract hardware targets if it is defined with that goal.

PyFR, for example, is an open-source computational fluid dynamics solver that implements a DSL+Python structure [34] to run on CPUs and Nvidia, Intel, and AMD GPUs. Portability is accomplished by writing the most compute-intensive kernels in a domain-specific language derived from the MAKO templating library. These kernels are called at run-time to compile the specialized numerical methods for a specific hardware target and bind it to Python. PyFR then uses Python tools to manage MPI calls, conduct I/O, pre- and post-process, compile, and manage the environment. Python, in this case, works as the glue language for the domain-specific one provided by MAKO. The overhead of this Python glue is less than 1% in PyFR [35]. While this development scheme does mean PyFR is written in two languages, only about 15% of its structure is abstracted using the MAKO DSL. As a result, PyFR has proven to be portable to various accelerators with compute kernels written in a single source. Withdren, et. al. [36] have discussed that this Python based scheme allows for rapid numerical methods development at deployment HPC scales. Using this tactic, they have successfully demonstrated PyFR's performance at the petascale [36].

While this scheme works for PyFR and other codes like it, the nature of neutron transport means that employing a PyFR-like approach would result in a much higher percentage of a code-base implemented in the secondary DSL (whether MAKO, C++, or Fortran). In neutron transport, the compute kernel is the physics kernel. A developer experimenting with and/or writing a novel numerical method will often be writing the most expensive compute kernels themselves. For example, when adding a tally to compute a new value, a developer could drastically alter the memory footprint of the computation. When adding a new event for a novel variance-reduction technique, they

could introduce a critical divergence that will limit performance of that technique on GPUs. A DSL+Python scheme still requires developers to know two languages—one which is unique to the code at hand—and knowledge of the accelerator architecture to develop performant code.

Other modern codes have used completely Python-based solutions to create a portable, performant code base for both CPUs and GPUs. Native Python is slow in terms of raw numerical performance. However, packages like NumPy [32] can do these operations significantly faster than the same algorithms in native Python for example if an algorithm depends on array-wise elemental operations. Furthermore, multi-core CPU processing and targeting hardware accelerators can be provided by libraries that link into functions often written in Fortran or C (e.g., BLAS).

For example, Veros [37], a global ocean modeling code, implements this workflow: numerical methods used to model the physics are deterministic, relying on array-type operations, like linear algebra. Veros uses the JAX library [38] to compile NumPy-based operations into GPU compute kernels and Python-based MPI implementations for distributed memory parallelism. Hafner et al. [37] show wall-clock speedup when implementing a JAX based development scheme, compared with the same algorithm traditionally implemented in a Fortran code. They also show reasonable scaling across nodes up to 1000 MPI processes. JAX and NumPy allow Veros developers to implement performant code exclusively in pure Python, which can target both CPUs and GPUs. Similar workflows that rely exclusively on commonly abstracted operations could be implemented by the CuPy [39] library, which binds into vendor-specific implementations of LAPACK/BLAS libraries.

Unfortunately, Monte Carlo kernels do not rely on these standard array-type operations. Monte Carlo algorithms have very complex control flow that depends on the physics of the problem being modeled and decisions made by the user at run time. Even the most fundamental building blocks of a Monte Carlo code might have to be altered to implement a novel numerical method.

Other projects have addressed the need to write user defined compute kernels in a Python-based framework. Sakras [40] is a molecular dynamics suite for plasma physics. It uses the Numba compiler to lower Python code with Numpy functions into LLVM, then just in time (JIT) compile those functions to a specific hardware [41]. The algorithms implemented in Sakras using Numba performed the same as when implemented purely in C. While Sakaras only supports shared memory parallelism on a single-node, this development scheme could easily be paired with MPI to achieve distributed memory parallelism (as in the previous examples).

Numba also compiles global and device functions for Nvidia GPUs from compute kernels defined in Python. API calls are made through Numba on both the Python side (e.g., allocate and move data to and from the GPU) and within compiled device functions (e.g., to execute atomic operations). When compiling to GPUs, Numba supports an even smaller subset of Python, losing most of the operability with NumPy functions. If functions are defined using only that smallest subset, Numba can compile the same functions to CPUs or GPUs, or execute those functions purely in Python. Numba data allocations on the GPU can be consumed and ingested by functions from CuPy if liner-algebra operations are required in conjunction with user-defined compute kernels.

As any of these portability schemes would be novel when implemented in a Monte Carlo neutron transport application, we conducted an initial set of investigations to compare what we considered to be the most viable options at the time. I compared the same algorithm in various implementations using PyKokkos [31], PyCUDA/PyOpenCL [33], and Numba [42]. I found that all three methods produced similar runtimes for our workflows on CPUs and GPUs for a simple transient Monte Carlo neutron transport simulation [42]. Ultimately, we decided to use a Numba + mpi4py development scheme to build out a Monte Carlo neutron transport code for rapid numerical methods development, portable to various HPC architectures [43–45] (RQ 1).

1.4.2 Delta tracking

Woodcock, or delta, tracking [46] is a variance-reduction technique that computes the majorant cross-section for the whole problem space, then uses this to determine a distance to collision for all particles. Coupled with rejection sampling to sort for phantom collisions, and a collision estimator to compute scalar flux, delta tracking often improves performance over analogue Monte Carlo in problems that warrant it (problems with a long mean free path). Many production Monte Carlo Neutron transport codes like Serpent [47–49], and others [50] investgated or use this method. In traditional delta tracking first the macroscopic majorant cross section is copmuted for the entire problem space

$$\Sigma_M(E) = \max\left(\Sigma_{T,b}(E), ..., \Sigma_{T,B}(E)\right) , \tag{1.5}$$

where E is energy, Σ_M is the microscopic majorant cross-section, and $\Sigma_{T,b}$ is the microscopic total cross-section of the $= b^{\text{th}}$ material. Now to sample a distance we calculate the distance to a collision

$$D = \frac{-\ln \xi}{\Sigma_M(E)} , \tag{1.6}$$

where ξ is a random number between zero and one and N_i is the number density in cell i. If the potential collision occurs within the cell, we move the particle to the sampled distance and do rejection sampling, since we are now potentially forcing collisions that did not occur. We sort out these phantom collisions by allowing particles to continue to a new sampled distance if

$$\xi < \frac{\Sigma_{T,j}(E)}{\Sigma_M(E)} , \tag{1.7}$$

where ξ is a new random number between zero and one and $\Sigma_{T,j}(E)$ is the macroscopic total cross-section of the cell. Standard delta tracking is required to use a coliision estimator which is less efficant [51]. If I can find a way to use the track-lenght estimator while doing delta tracking we could improve the performacen of a Monte Carlo code (RQ 5).

2 Objectives

2.1 GPU enabled time dependent neutron radiation transport

This exploration attempts to answer RQ 1.

1. **DONE** Compare portability frameworks for Monte Carlo neutron transport deployment on both GPU and CPU platforms in a transient code
2. **DONE** Support the production of a full Monte Carlo transport code and aid in the deployment to other accelerators
3. **Submitted** Publish this work for a general software engineering audience
4. **ONGOING** Conduct GPU performance analysis for fully time dependent problems of interest and publish

2.2 Explorations into GPU optimized deterministic schemes

This investigation attempts to answer RQ 1 and 2, 3.

1. **DONE** Derive discretization scheme for time dependent, multi-group, Sn, neutron transport in slab geometry
2. **DONE** Conduct Fourier analysis to show the stability of the discretization scheme;
3. **DONE** Implement a mono-energetic proof of concept of discretization schemes in both Source iterations and One-cell inversions in a Python code for both CPUs and GPUs;
4. **DONE** Extend simulation to be energy dependent with the multi-group assumption and explore in C++;
5. **ONGOING** Use the method of manufactured solution to verify the discretization scheme in problems and measure its convergence rate;

6. **DONE** Implement discretization scheme in both a Source iteration and One-cell inversion iterative scheme on GPUs using vendor supplied linear algebra libraries to abstract device kernels, and measure wall clock runtime;
7. **DONE** Investigate transient effects on convergence rate of one-cell inversion and compare to source iteration; and
8. **ONGOING** Publish this work.

2.3 Exploration of acceleration schemes for One-cell inversion

This exploration attempt to answer RQ 4

1. **ONGOING** Derive and postulate various potential acceleration schemes, then down select to one or two schemes
2. Do fourier analysis on selected scheme
3. Implement acceleration scheme in code
4. Verify convergence to correct value using method of manufactured solutions and/or Monte Carlo results
5. Explore convergence rate in various problems
6. Publish this work

2.4 Exploration of delta tracking using a hybrid tally

This study attempts to answer RQ 5

1. **DONE** Implement the hybrid-delta tracking algorithm on a structured mesh in MCATK
2. **DONE** Compute runtime for speedup in materially complex problems using this novel scheme
3. **DONE** Publish this work
4. **ONGOING** Compute the macroscopic majorant cross section for an entire problem as a pre-process in MC/DC;
5. Under a boolean condition—set by the user at runtime—add rejection sampling and distance to collision sampling using the majorant;
6. When running in delta tracking mode use a path length estimator to tally the scalar flux and a collision estimator to tally other quantities of interest (i.e. reaction rates);

7. Test using a problem of interest and compare FOM for quantities of interest;
8. Publish this work

2.5 Possible difficulties and contingencies

The largest and most risky open question left in this research is weather we can find an acceleration scheme for one cell inversions in the thin limit. While explorations that result in a negative result are still essential and worth while, I list a few contingencies if acceleration schemes prove more difficult to come by:

1. Investigations into a multi-grid multi-group scheme enabled by the solution of all angles and groups within a cell
2. 2D explorations of OCI with further performance analysis on GPUs using roofline models
3. Focusing an additional investigations into delta tracking in MC/DC, specifically looking at an adaptive energy scheme, where high energies with large MFPs use delta tracking and low energies use surface tracking

3 Proposed methods

In this section I will describe the methods I am employing in my research but is intentionally brief. This section is heavily supplemented by publications included as appendices to this document.

3.1 Deterministic transport

For deterministic methods appendix A.5 contains:

- expanded derivations of the higher order space-time-energy discretization scheme;
- derivation of fourier analysis of the discretization scheme;
- derivation of the method of manufactured solution; and
- description of the implementation in code

The remainder of this section will be devoted to describing the current state of the investigations into an acceleration scheme for OCI in the thin limit (see 3.1.5).

3.1.1 Derivation of Higher order Methods in Iterative Schemes

When the Sn transport equation is descritzed in space using simple corner balance and in time with time dependent multiple balance a 4-equation system is begote:

$$\begin{aligned}\frac{\Delta x_j}{2}\frac{1}{v_g}\left(\frac{\psi_{m,g,k+1/2,j,L}-\psi_{m,g,k-1/2,j,L}}{\Delta t}\right)&\\ +\mu_m\left[\frac{(\psi_{m,g,k,j,L}+\psi_{m,g,k,j,R})}{2}-\psi_{m,g,k,j-1/2}\right]&\\ +\frac{\Delta x_j}{2}\Sigma_j\psi_{m,g,k,j,L}=\frac{\Delta x_j}{2}\frac{1}{2}\left(\sum_{g'=0}^{G}\Sigma_{s,j,g'\to g}(x)\sum_{n=1}^{N}w_n\psi_{n,g,k,j,L}+Q_{k,j,L}\right)&,\end{aligned}\quad (3.1a)$$

$$\frac{\Delta x_j}{2}\frac{1}{v}\left(\frac{\psi_{m,k+1/2,j,R}-\psi_{m,k-1/2,j,R}}{\Delta t}\right)+ \mu_m\left[\psi_{m,k,j+1/2}-\frac{(\psi_{m,k,j,L}+\psi_{m,k,j,R})}{2}\right] + \frac{\Delta x_j}{2}\Sigma_j\psi_{m,k,j,R}=\frac{\Delta x_j}{2}\frac{1}{2}\left(\sum_{g'=0}^{G}\Sigma_{s,j,g'\to g}\sum_{n=1}^{N}w_n\psi_{n,g',k,j,R}+Q_{k,j,R}\right)\ , \quad (3.1b)$$

$$\frac{\Delta x_j}{2}\frac{1}{v_g}\left(\frac{\psi_{m,g,k+1/2,j,L}-\psi_{m,g,k,j,L}}{\Delta t/2}\right) + \mu_m\left[\frac{\left(\psi_{m,g,k+1/2,j,L}+\psi_{m,g,k+1/2,j,R}\right)}{2}-\psi_{m,g,k+1/2,j-1/2}\right] + \frac{\Delta x_j}{2}\Sigma_j\psi_{m,g,k+1/2,j,L}=\frac{\Delta x_j}{2}\frac{1}{2}\left(\sum_{g'=0}^{G}\Sigma_{s,j,g'\to g}\sum_{n=1}^{N}w_n\psi_{n,g',k+1/2,j,L}+Q_{k+1/2,j,L}\right)\ , \quad (3.1c)$$

$$\frac{\Delta x_j}{2}\frac{1}{v_g}\left(\frac{\psi_{m,g,k+1/2,j,R}-\psi_{m,g,k,j,R}}{\Delta t/2}\right)+ \mu_m\left[\psi_{m,g,k+1/2,j+1/2}-\frac{\left(\psi_{m,g,k+1/2,j,L}+\psi_{m,g,k+1/2,j,R}\right)}{2}\right] + \frac{\Delta x_j}{2}\Sigma_j\psi_{m,g,k+1/2,j,R}=\frac{\Delta x_j}{2}\frac{1}{2}\left(\sum_{g'=0}^{G}\Sigma_{s,j,g'\to g}\sum_{n=1}^{N}w_n\psi_{n,g',k+1/2,j,R}+Q_{k+1/2,j,g,R}\right)\ , \quad (3.1d)$$

where Δx is the cell width, j is the spatial index, L and R denote the right and left half-cell averaged quantities, k is time averaged quantities, $k+1/2$ is time edge quantities. These equations contain a spatial—the angular flux at the cell midpoint is a simple average of the two half-cell average quantities—and a time—*upstream* prescription for the cell-edge angular flux—closures. This defines system defines angular flux on a stencil shown in figure 3.1.

In the traditional source iteration, the scattering source is presumed known from a previous iteration, which leads to the following set of equations to be solved in transport "sweeps". This means that new estimates of both the end of time-step value of angular flux and time-averaged angular flux are computed together in each cell. So a 4×4 sized system (from stincle values) of equations must be solved to go from cell to cell in each angle. This process is annoyingly serial over space, but embarrassingly parallel over angles. The whole system sparse matrix is block (in each ordinate) lower triangular (over all cells). Source iterations couple angles together in the iterative method by use of the scalar flux term.

In OCI, the scattering source is subtracted to the left-hand side, and the information that comes from cells other than cell j is assumed to be known from a previous iteration.

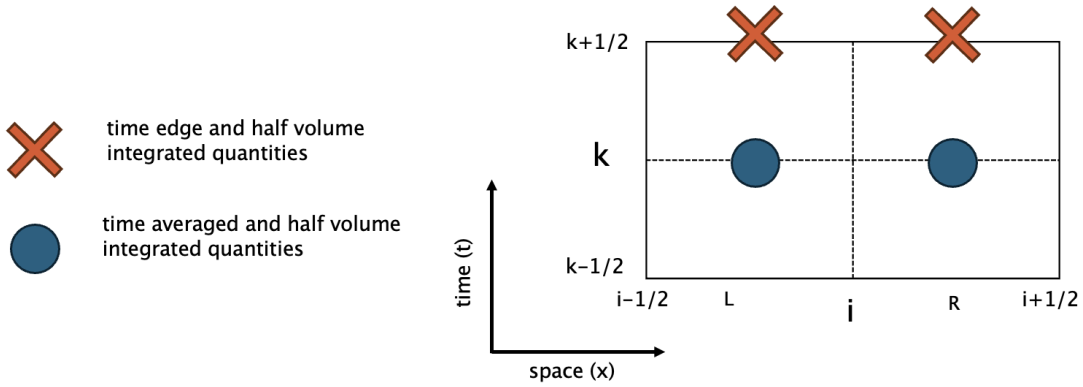

Figure 3.1: The discretization stencil

This means that all $4NG$ angular fluxes (N angles, G groups, and stencil values L, R, k, $k+1/2$) are computed simultaneously in cell j. For SCB in slab geometry, this means there is a local $4NG \times 4NG$ matrix to be solved in each cell.

3.1.2 Fourier Analysis

To ensure that our combination of higher-order discretization schemes is still unconditionally stable, we performed a Fourier analysis to find the slowest-converging mode of the method. Assuming no scattering, no sources, mono-energetic problem, and make physical assumptions (only constant positive values of Δx, Δt, Σ, and v) we derived SCB-MB's eigenfunction and numerically solve it. We start by defining our Fourier ansätze as,

3.1.3 Verification using method of manufactured solution

Finding benchmark problems that simulate transient, energy dependent, single dimension solutions to the neutron transport equation is difficult. I used the Method of Manufactured solutions to derive my own benchmark [52]. It is wildly accepted in the felid [53] and has previously been used to verify other deterministic neutron transport codes including the production code solver MOOSE [54, 55].

The method of manufactured solutions follow a backward solving technique where a solution is made-up—expressing any desired behavior (e.g. smoothness, differentiability, positivity, etc.)—then inserted into the governing equation and solved for the source term. The manufactured source source is supplied to the solver to verify that it converges to the correct solution at the expected rate. To derive our method of manufactured solution we start with a conditions functional representation of the angular flux in angle, energy, space and time

$$\psi_1(x,t,\mu) = (1-\mu^2)\sin(\pi x)e^{-t} \tag{3.2a}$$

$$\psi_2(x,t,\mu) = (1-\mu^2)(-x^4+x+1)e^{-t/2} \tag{3.2b}$$

These equations can now be inserted into the NTE (eq 1.4) and solved for a continuous functional representation of Q. From there the source equations are placed on the stencil by averaging (via numerical integration) in space and time where nessacary (see 3.1). This discretized set of sources is then supplied to the solver.

3.1.4 Implementation in Code

This section describes how OCI and SI are implemented in code and what is being timed in the comparison. For source iteration the algryitm

- Build

3.1.5 Exploration of Acceleration Scheme

This section discusses ongoing work: the derivation of an acceleration scheme for OCI in the thin limit. When working with the iterative schemes it becomes useful to define systems in operator notation. The NTE can be written generally as

$$H\psi = q \tag{3.3}$$

where H is some invertible operator, ψ is the unknown function for angular flux and q is some source. To solve this equation we split H into components that makes sense given the NTE and can then let something lag and something lead (assuming a fixed point iteration). How that operator is split is what differentiate OCI from SI in this perspective. For example if we choose $H = L - S$ where L is the left hand linear side of the transport equation and S is the scattering source and finally let the fluxes on the source lag we get

$$L\psi^{l+1} = S\psi^{l} + q \tag{3.4}$$

which is just a source iteration. For now we will start with OCI's operator notation from Rosa Warsaw and Perks (2010) which they call cellwise bJ

$$(L - S)\psi = q \tag{3.5}$$

where

- L = streaming thru space and time and total integration operator
  $(\mu_m \frac{\partial \psi}{\partial x} + \frac{1}{v}\frac{\partial \psi}{\partial t} + \Sigma\psi)$
- S is the scattering operator (discrete-to-moment operator) which couples all the simultaneous PDEs together
  $(\Sigma_s \int_{-1}^{1} \psi \partial\mu = \Sigma_s \sum_{n=0}^{M} w_n \psi_n = \Sigma_s \phi)$
- ψ angular flux (dependent variable of interest)
- q is the fixed material source

OCI works by lagging the incident cell wall fluxes of the cell. Those flux come from the upstream closure that transport solvers use. Thus it is convenient to define,

$$L = L_c + L_b \tag{3.6}$$

where

- L_c are the angular fluxes within a mesh cell
- L_b are the angular fluxes incident to the mesh cell's surface

Note that what L_b actually consists of depends on the space discretization so from here we will have to suffice with operator notation. After this operator splitting equation 3.5 now becomes

$$\psi[I + (L_c - S)^{-1}L_b] = (L_c - S)^{-1}q \tag{3.7}$$

where I is the identity matrix. When we prescribe what lags in OCI (incident fluxes on the surface of the cell) this becomes

$$\psi^{(l+1)} = (L_c - S)^{-1}(-L_b\psi^l + q) \tag{3.8}$$

When looking for an acceleration scheme examining the structure of L_b becomes important. In my research I use the TDMB+SCB discretization for time dependent single dimension multi-group Sn problems (equation 3.1). If the angular fluxes from the boundary of the cell are identified (denoted by $j \pm 1/2$) we can sus out L_b's structure. It should be something like $\pm\mu_m$ filled along four diagonals with every 4th space filled. This is also similar to how the SI equations look if formed as a lower triangular system. In higher dimensions I expect the structure to remain mostly the same for rectilinear grids but things might get hairy for unstructured meshes where surface areas may become an issue. But for now we generally know the structure of L_b

Next lets think about the within cell operator L_c. We know L_c contains the differential and absorption terms from the NTE. My hunch—or rather conclusion of Fourier analysis of OCI [16] and from my own investigations [56]—is that the poor convergence rate is dependent on the cellular opacity of a problem measured in mean-free paths (MFP)s. This leads to a large condition number of our iteration matrix and thus slow convergence. To isolate this term we start with

$$L_c = K_d + K_a \tag{3.9}$$

where K_d is the differential operators and

$$K_a = \frac{\Delta x \Sigma}{2} I \tag{3.10}$$

K_a's structure is pretty easy to identify from the equations 3.1. I could pull out a solitary Σ but in this form I can identify that

$$K_a = \frac{\text{MFP}}{2} I \tag{3.11}$$

At this point we have to ask ourselves is it truly dependent on MFP alone or does Δx on it's own also ill-condition this system. Δx is contained in other terms that could be isolated and solved, but for now lets put a pin in that and continue on the hunch it is exclusively MFP. I conducted an investigation to prove that it was only dependent on MFP. Now that we know what we are looking at and have isolated our terms of interest lets push forward and break up our equations into a standard "synthetic" approach.

I start with equation 3.8 but recast it with some of the expansions I just identified,

$$\psi^{l+1}(K_d + \delta I - S) = (-L_b\psi^l + q) \tag{3.12}$$

where δ is MFP/2. Now we can write the associated equation for the iteration error as,

$$(K_d + \delta I - S)f^{l+1} = -L_b f^l \tag{3.13}$$

where $f^{l+1} = \psi^{\text{converged}} - \psi^{l+1/2}$ and $f^l = \psi^{\text{converged}} - \psi^l$. Note that this equation is exact in it's current forum. Now we use $-L_b f^{l-1}$ as our special one and subtract from both sides

$$(K_d + \delta I - S + L_b)f^{l+1} = L_b(f^{l+1} - f^l) \tag{3.14}$$

Notably here if we re-add everything back together we should still have equation 3.8. Now we can formally set up our synthetic-two-step starting with equation 3.8 rewritten for half steps

$$\psi^{(l+1/2)} = (L_c - S)^{-1}(-L_b\psi^l + q) \tag{3.15a}$$

the associated equation for mid step error

$$f^{l+1/2} = -ML_b(\psi^{l+1/2} - \psi^l) \tag{3.15b}$$

and the definition of the errors

$$\psi^{l+1} = \psi^{l+1/2} + f^{l+1/2} \tag{3.15c}$$

where,

$$M \approx H^{-1} = (K_d + \delta I + L_b - S)^{-1} \tag{3.16}$$

and is easier to obtain then the actual inverse. If we actually inverted H this would just be another transport step. Note that we can relate this idea of synthetic acceleration to preconditioning by noting

$$P \equiv I + M(I - H) \tag{3.17}$$

From here I take some stabs at identifying what reasonable values for M could be. We know H is painful to invert when δI is small as that is a problem in the thin limit where cell wise communication becomes dominant. When δI is small a good conservative approximation for M may become

$$H^{-1} \approx M = (K_d + L_b - S)^{-1} \tag{3.18}$$

but we just converged our within-cell scattering, time rate of change, and steaming operators from the transport step so lets assume we have a good estimate of that. We can also assume that the boundary information coming into every cell will blah likewise lets scoot that and all we are left with is the differential operators (from a discretization scheme)

$$H^{-1} \approx M = (K_d)^{-1} \tag{3.19}$$

which makes the full mid step solution

$$-K_d^{-1} * L_b \tag{3.20}$$

This leads to the suggestion at the possible algorithm

1. transport step as described in eq. 3.15a
2. mid step corrector (eq. 3.15b) with . This will look like a transport in every angle
3. colsure from eq

But Joanna, your probably asking yourself. This just using a cheap sweep between every OCI transport iteration I say in 1D in higher dimensions this will look like a *a ray trace* something GPUs are literally built for.

From here more work is need to be down to conduct Fourier analysis in both with and with out the discretization scheme

What ever scheme is used it can be supplemented by GMRES

3.2 Monte Carlo transport methods

As in the previous section the methods employed are described in attached publications in the appendix. They are beefily summarized here but far more information is contained there for the edification of the reader.

3.2.1 Python Based Portability for Rapid Methods Development

A software engineering scheme was developed and deployed in Monte Carlo/Dynamic Code [44] using the Numba compiler for Python [41], mpi4py [57], and an a-synchronous GPU event scheduler called Harmonize [58] (*see appendix A.3*). MC/DC can run the same set of physics functions written by subject-area experts in the Python interpreter or compiled to x86-64, ARM64, Power9 CPUs and AMD and Nvidia GPUs. All compilation targets also support the use of MPI to distribute work to multiple nodes of an HPC. It is a demonstration of portability using a high-level language coupled to a portability framework to enable rapid methods development.

3.2.2 Delta tracking in MCATK and MC/DC

In MCATK a hybrid delta-surface tracking scheme was developed and deployed (*see appendix A.4*). This scheme initially implemented by me has been extended by others and is shipped in current production versions of MCATK. The point of the delta-surface tracking scheme in MCATK was entirely to avoid extra macroscopic cross section lookups and be an easy to implement drop in replacement. It still conducted surface tracking using on mesh. After implementation the hybrid delta-tracking scheme was then tested in both k-eigenvalue and fixed source transient simulations using the,

- HEU-MET-FAST-086 benchmark of Godiva IV [59]; and
- Measurement of Uranium Subcritical and Critical IER 488 (MUSiC) experiment [60].

Runtime and errors were then compared to draw conclusions. This work was published in a conference publication [56].

In MC/DC I propose a slight alteration to the hybrid-scheme developed and deployed in my previous work in MCATK. Current versions of MC/DC do not stop particles at mesh crossings and uses a track-length estimator to tally impacts to multiple cells at once. Using this algorithm already in MC/DC, delta tracking could be used in conjunction with the track length estimator as in the previous work in MCATK, but in MC/DC will more closely resemble the delta tracking algorithm and not stop particles at now cell and surface crossings. Other quantities of interest will be tallied with collision estimator as the specific material data is required for values like fission reaction rate. This will require additional computational burden but the hope is that the new tally functionality in MC/DC is sufficiently optimized to still allow this scheme to be more performant.

To evaluate the delta tracking scheme as compared to surface tracking I will use figure of merritt (FOM) values. FOMs are constructed to observe impact of a variance reduction technique on both variance (as computed by the Monte Carlo scheme) and on runtime (as many variance reduction schemes incur additional computation burden). The most common figure of merit and the one I will employ is simply

$$FOM = \frac{1}{\sigma^2 t_{wc}} \tag{3.21}$$

where σ^2 is the variance as already computed by MC solvers and t_{wc} is the total wall-clock-runtime. Using this metric (higher is better) variance reduction shames can be compared to each other and the analog solver. I propose using this metric to evaluate the impact of the hybrid-delta tracking scheme I plan to implement in MC/DC in a problem of interest. Potential problems of interest include:

1. Dragon burst problem [61];
2. C5G7 Small modular reactor [62]; and
3. Kobyashi void dog-leg void problem (multiprop and contentious energy simulations) [63].

4 Preliminary Results

4.1 Deterministic transport

4.1.1 Fourier Anylisys of Discretization Scheme

We found that the MB-SCB scheme is unconditionally stable over $\mu \in [-1, 1]$. While experimenting with this method we did find that, under some conditions, it can produce negative fluxes; however, the negative flux oscillations were critically damped and dissipated with time.

4.1.2 Verification of Discretization Scheme

Charts

4.1.3 Performance Results

Graphs

4.2 Portable Monte Carlo solver

4.2.1 Comparison of Python-Portability Frameworks

List the TNT investigations

4.2.2 Monte Carlo / Dynamic Code

List what was done and some performance figures

5 Planned schedule

My proposed timeline for the remainder of my PhD, included is three additional publications (1) conference proceeding (2025 M&C) about MC/DC's GPU performance, (2) journal article about deterministic work currently completed, and (3) journal article about investigated acceleration schemes.

Task	Oct	Nov	Dec	Jan	Feb	Mar	April	May	June
Performance analysis of MC/DC and drafting publication for M&C 2025									
Drafting journal paper for deterministic work									
Down selection of novel acceleration scheme									
Fourier analysis of Novel acceleration scheme									
Implementation of acceleration scheme in code with SI+DSA comparison									
Verification and exploration of convergence rate									
Drafting acceleration paper									
Submission of acceleration paper									
Defense									

Figure 5.1: Green is Monte Carlo work, purple is the first set of deterministic work, blue is the second set of deterministic work focused on finding an acceleration scheme black is defense.

A appendices

A.1 Conference Publication, and Presentation for proposed investigations

A.1.1 OCI Investigations

Copper Moutian
M&C 2023

A.1.2 Hybrid-Delta Tracking Schemes

M&C 2023

A.1.3 Acceleration and Abstraction of Python

SciPy 2024
S3C 2024
Monte Carlo Computational Summit
M&C 2023
ANS Annual 2022
SciPy 2022

A.2 All Publications

A.3 Submitted Draft: Performant and Portable Monte Carlo Neutron Transport via Numba

This paper was submitted to *IEEE Computing in Science and Engineering* on September 3rd 2024. I have not received a response. A preprint was posted to arxiv and has been the assigned the DOI 10.48550/arXiv.2409.04668.

My coauthors are all members of the Center for Exascale Monte Carlo Neutron Transport (CEMeNT) includes

- Ilham Variansyah, PhD (committee member)
- Braxton Cuneo, PhD

- Todd S. Palmer, PhD (committee member)
- Kyle E. Niemeyer, PhD (advisor)

As MC/DC is the primary deliverable of CEMeNT it is a deeply combative effort.

Performance Portable Monte Carlo Neutron Transport in MCDC via Numba

Joanna Piper Morgan[1,2*] Ilham Variansyah[1,3 †]
Braxton Cuneo[1,4] Todd S. Palmer[1,3] Kyle E. Niemeyer[1,2]

[1]Center for Exascale Monte Carlo Neutron Transport (CEMeNT)[‡]
[2]School of Mechanical Industrial and Manufacturing Engineering, Oregon State University
[3]School of Nuclear Science and Engineering, Oregon State University
[4]Department of Computer Science, Seattle University

Abstract

Finding a software engineering approach that allows for portability, rapid development, open collaboration, and performance for high performance computing on GPUs and CPUs is a challenge. We implement a portability scheme using the Numba compiler for Python in Monte Carlo / Dynamic Code (MC/DC), a new neutron transport application for rapid Monte Carlo methods development. Using this scheme, we have built MC/DC as a single source, single language, single compiler application that can run as a pure Python, compiled CPU, or compiled GPU solver. In GPU mode, we use Numba paired with an asynchronous GPU scheduler called Harmonize to increase GPU performance. We present performance results for a time-dependent problem on both the CPU and GPU and compare them to a production code.

Developing software to simulate physical problems that demand high-performance computing (HPC) is difficult. Modern HPCs commonly use both CPUs and GPUs from various vendors. Years can be spent porting a code from CPUs to run on GPUs, then again when moving from one GPU vendor to the next [1]. Portability issues compound when designing software for rapidly developing numerical methods where algorithms need to be both implemented and tested at scale. Finding a software engineering approach that balances the need for portability, rapid development, open collaboration, and performance can be challenging especially when numerical schemes do not rely on commonly library implemented operations (i.e., linear algebra as in LAPACK or

*morgajoa@oregonstate.edu
†variansi@oregonstate.edu
‡https://cement-psaap.github.io/

Intel MKL). Common HPC software engineering requirements are often met using a Python-as-glue-based approach, where peripheral functionality (e.g., MPI calls, I/O) is implemented using Python packages but compiled functions are called through Python's C-interface where performance is needed. Python-as-glue does not necessarily assist in the production of the compiled compute kernels themselves—what the Python is gluing together—but can go a long way in simplifying the overhead of peripheral requirements of HPC software. With this technique, environment management and packaging uses `pip`, `conda`, or `spack`, input/output with `h5py`, MPI calls with `mpi4py`, and automated testing with `pytest`, which can all ease initial development and continued support for these imperative operations.

Many tools have been developed to extend the Python-as-glue scheme to allow producing single-source compute kernels for both CPUs and GPUs. One tactic is to use a domain-specific language to avoid needing a low-level language (e.g., FORTRAN, C). A domain-specific language is designed to alleviate development difficulties for a group of subject-area experts and can abstract hardware targets if defined with that goal. PyFR, for example, is an open-source computational fluid dynamics solver that implements a domain-specific language plus Python structure to run on CPUs and Nvidia, Intel, and AMD GPUs [2]. Witherden et al. [2] discussed how this scheme allows PyFR developers to rapidly deploy numerical methods at deployment HPC scales and have demonstrated performance at the petascale. Other projects have addressed the need to write user-defined compute kernels entirely in Python script. Numba is a compiler that lowers a small subset of Python code with NumPy arrays and functions into LLVM, then just in time (JIT) compiles to a specific hardware target [3]. Numba also compiles global and device functions for Nvidia GPUs from compute kernels defined in Python. API calls are made through Numba on both the Python side (e.g., allocate and move data to and from the GPU) and within compiled device functions (e.g., to execute atomic operations). When compiling to GPUs, Numba supports an even smaller subset of Python, losing most of the operability with NumPy functions. If functions are defined using only that smallest subset, Numba can compile the same functions to CPUs or GPUs, or execute those functions purely in Python. Numba data allocations on the GPU can be consumed and ingested by functions from CuPy if linear-algebra operations are required in conjunction with user-defined compute kernels.

We used a Numba+Python development scheme to abate portability issues and allow for rapidly developing novel numerical methods at the HPC scale on CPUs and GPUs when we developed a new Monte Carlo neutron transport application called Monte Carlo / Dynamic Code (MC/DC) [4, 5]. In this paper we first provide an overview of neutron transport and the Monte Carlo solution method. We next describe MC/DC's novel (for the field) software engineering approach in greater detail. We discuss how novel numerical methods are prototyped and developed in MC/DC. Then, we analyze the compute performance of MC/DC on both CPUs and GPUs and, where possible, compare it against modern production Monte Carlo neutron transport solvers. Finally, we provide concluding remarks and outline future work.

1 Monte Carlo / Dynamic Code

Predicting how neutrons move through space and time is important when modeling inertial confinement fusion systems, pulsed neutron sources, and nuclear criticality safety experiments, among other systems. Simulating these problems is computationally difficult using any numerical method, as the neutron distribution is a function of seven independent variables: three in space, three in velocity, and time. Modern HPC systems now enable high-fidelity simulation of neutron transport for problem types that have seldom been modeled before due to limitations of previous computers. Specifically, large scale, highly dynamic transport problems require thousands of compute nodes using modern hardware accelerators (i.e., GPUs) [6, 7].

The behavior of neutrons can be modeled with a Monte Carlo simulation, where particles with statistical importance are created and transported to produce a particle history [8]. A particle's path and the specific set of events that occur within its history are governed by pseudo-random numbers, known probabilities (e.g., from material data), and known geometries. Data about how particles move and/or interact with the system are tallied to solve for parameters of interest with an associated statistical error from the Monte Carlo process. The analog Monte Carlo method is slow to converge (with a convergence rate of $\mathcal{O}(1/\sqrt{n})$ where n is the number of simulated particles). New Monte Carlo schemes could converge the solution faster in wall-clock time with fewer simulated particles and may be needed to effectively simulate some systems. Monte Carlo/Dynamic Code (MC/DC) was written to enable the rapidly developing these novel numerical methods.

MC/DC has a similar feature set to other Monte Carlo neutron transport applications (e.g., OpenMC [7], Shift [6]) with support for k-eigenvalue and fully time-dependent simulation modes in full 3D constructive solid geometry. It can model the neutron distribution in energy by either using continuous energy or multigroup nuclear data. It also supports domain decomposition. All these features are supported on CPU (x86, ARM, POWERPC) and GPU targets (Nvidia and, soon, AMD GPUs).

The number of novel schemes and simulation techniques implemented in MC/DC in a short time illustrates the success in its software engineering structure MC/DC supports the use of the iterative quasi-Monte Carlo (iQMC) method where deterministic and Monte Carlo transport operations are used in tandem to converge solutions faster than they would in a pure Monte Carlo method. MC/DC supports hash-based random number generation to have a fully-replicated solution for testing, even when running a domain decomposed problem. It supports population control techniques for use in the time-dependent mode where particle populations can grow or decay exponentially. MC/DC has the ability to do uncertainty and global sensitivity analysis to isolate the variance introduced by the Monte Carlo process from the variance due to uncertainty in input parameters (i.e., uncertainty associated with material data). Continuous movement of surfaces (other codes move geometry as discrete steps between time steps) is implemented to better model transient events, such as fuel rod insertion into a

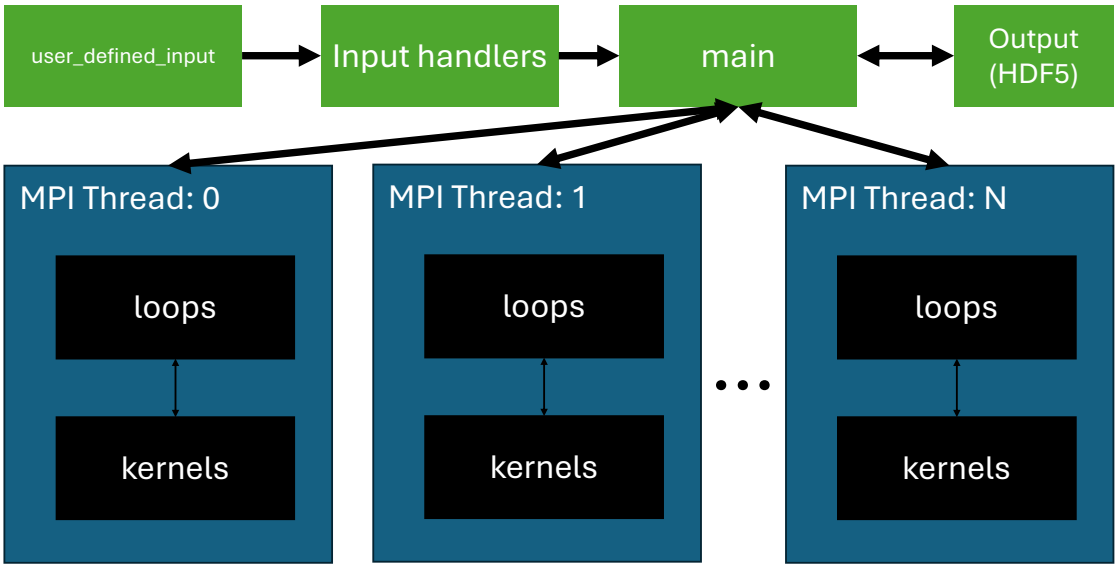

Figure 1: MC/DC's overall structure and how functions get called and interact. Green functions are entirely Python, and black functions are compiled compute kernels if they are running in Numba CPU or GPU modes.

reactor or criticality safety experiments. Several ongoing developments include Quasi Monte Carlo, residual Monte Carlo, and machine learning techniques for dynamic node scheduling.

Figure 1 shows MC/DC's functional layout when running in both MPI and Numba mode. First, a user writes a Python script and imports `mcdc` as a package. Then, the user interfaces with functions described in the input handler within MC/DC describing the physical models, material data, and simulation parameters. This input layout is similar to how other Monte Carlo neutron transport applications input problems. This input script calls a run function, which starts initialization functions within MC/DC. In initialization, a global variable is allocated and constructed containing the user-defined inputs, meshes, particle banks, event tallies, and current global states. This global variable is constructed from a statically typed Numpy `ndarray`, which acts like a Python dictionary, where keywords are used to get numerical arrays out. After the global variable is built, initialization functions dispatch MPI processes if running in MPI mode, and begin the Monte Carlo neutron transport simulation.

Within each MPI process, loops containing the various transport algorithms and modes that MC/DC supports are called. Each transport function is decorated with a Numba JIT (`@jit`) compilation flag declaring that each function must be compiled before being executed if running in Numba mode. These transport logic loops are the highest level at which Python functions will be compiled in MC/DC. For example, a fixed source problem will *loop* over all

```
import numpy

part = numpy.dtype([
    ('x', float64), ('y', float64),
    ('z', float64), ('ux', float64),
    ('uy', float64), ('uz', float64),
    ('v', float64) ])

@jit
def move_particle(P: part, distance):
    P['x'] += P['ux'] * distance
    P['y'] += P['uy'] * distance
    P['z'] += P['uz'] * distance
    P['t'] += distance / P['v']
```

Figure 2: Example of a decorated function and MC/DC's data structures based on `numpy.ndarray`.

the particles and transport them until the particle's history is terminated from a physical event (e.g., capture, fission, time or space boundary), a simulation event (e.g., time census), or a variance-reduction event (e.g., population control, implicit capture). The specific functions within each algorithm that conducts the actual transport operations (e.g., moving particles, tallying events, generating daughter particles from fission) are contained in the kernel set where all functions are `@jit` decorated.

Figure 2 shows an example compute kernel that updates the position and time of a particle as it moves. It also shows the declaration of an `ndarray` data structure used in MC/DC.

After all transport is completed and the simulation is finished, the program returns to the Python interpreter and calls finalization functions. Here requested tally information along with statistical error provided from the Monte Carlo process is saved in an HDF5 file. Data can be extracted from this HDF5 file and used in Python scripts to do post-processing analysis and/or data visualization with tools like Matplotlib, or post-processing can be done in other applications like Visit or Paraview.

Moving to compile and run Numba `jit`ed functions to the GPU requires making a few alterations to the kernels themselves. An even smaller subset of Python functions work in GPU-compiled code, with operations supported on Numba-CPU like `numpy.linalg.solve()` losing support. Atomic operations are required to preserve the side-effects of individual threads in global memory (e.g., adding to a tally). To allow for a unified kernel base in MC/DC for both CPUs and GPUs we track alternate function implementations which are registered through decorators. Figure 3 shows how we implement alternate tally accumulation functions using `@for_cpu` and `@for_gpu` decorator. Here

```
from numba import cuda

@for_cpu
def add(array, value, idx):
    array[idx] += value

@for_gpu
def add(array, value, idx):
    cuda.atomic_add(array, value, idx)
```

Figure 3: Example of GPU and CPU specific API calls as defined in MC/DC.

the `@for_cpu` is adding one to an array, as this is within a single MPI rank we can assume this is a thread safe operation; however on the GPU this may result in a memory race condition thus requiring an `numba.cuda.atomic_add` API call. While this does increase complexity for a programmer implementing numerical methods, it is nowhere near the complexity that might be required to accomplish a similar implementation in a compiled language.

Just because a function compiles for a GPU target does not mean it will perform efficiently on the GPU. MC/DC does implement some specific algorithms designed to increase GPU performance (e.g. event based algorithms, branchless collisions), but more is needed to enable a Monte Carlo neutron transport application to be performant on GPUs. So, when compiling and running MC/DC for GPUs, an asynchronous event scheduling library called Harmonize [9] is used to re-organize the execution of business logic and storage of data to better fit the strengths of GPUs. Harmonize examines the operations that are due to be executed, segregating them into like-operations so that like-work may be executed together in batches. Monte Carlo transport functions lend themselves to asynchronous programming schemes as is intuitive to provide a function for each particle operation, like in the example of a `move_particle` function in Figure 2. Executing in this mode, MC/DC becomes a library of device functions that Harmonize ingests. This has been shown to increase GPU performance by reducing divergence [9].

When compiling functions to GPU targets with Harmonize, intermediate compiler representations (e.g., LLVM-IR or PTX) are used to inject the Monte Carlo neutron transport kernels within MC/DC into the GPU-optimized control flow defined in Harmonize. While Harmonize was built with MC/DC in mind, theoretically it could be used for any generic GPU workflow that exhibits significant thread divergence [9]. Currently, MC/DC only supports execution on the GPU with Harmonize. When running MC/DC in GPU mode, device memory for the global array is allocated then moved from the host (CPU) to the device (GPU) during initialization. After that, MC/DC's transport kernels are executed in Harmonize on the GPU until transport for a given collection of work is complete. Communication of the global variable between the device

and the host may be required during transport for some simulation modes like when conducting a census event for time dependent transport. When transport is completed the global variable is moved back to the host for a final time, and the simulation is finalized.

When initially exploring a novel transport method, a developer can work in a pure Python environment where functions are entirely executed in Python. In this mode, any package can be brought into any function, typing can be done dynamically, and any Python data structure can be used. MC/DC can be executed in MPI mode in Python or compiled CPU mode. While a full Python development environment is great for initial proofs of concept, it proves to be too slow for problems of interest.

When more performance is required, Numba mode can be used to compile functions to approach C-like speeds. However, Numba only supports a small subset of the Python ecosystem. Some Python data structures like dictionaries and lists can no longer be used and must instead come from NumPy implementations. Thus, when using Numba, the small subset of functions supported effectively becomes a domain-specific language. When kernels are written to support Numba mode, they can be compiled to any supported CPU targets (x86, ARM64, and PPC64) automatically. If even more performance is required, the subset of supported Python shrinks further and specific GPU API calls must be appropriately abstracted with function calls defined with `@for_cpu` and `@for_gpu`. At this point, an optimized kernel can be executed on the GPU with Harmonize.

2 Performance

To compare MC/DC to another Monte Carlo neutronics software package, we use a time-dependent version of the Kobayashi dog-leg void-duct problem [10, 5]. Figure 4 at left shows the void duct and the location of the neutron source at the opening of the duct. The initial condition is zero flux everywhere. Radiation moves down the void quickly and then penetrates into the walls of the problem. Figure 4 at right shows the duct clearly with the scalar flux solution in a time-averaging window of 10 s at somepoint time from MC/DC and OpenMC [7] (an open-source code written in `C++`) .

Figure 5 at left shows the runtime results of MC/DC compared with OpenMC on a single node of the Quartz supercomputer (two socket 18-core Intel Xeon E5-2695 v4) at Lawrence Livermore National Laboratory (LLNL). Caching in MC/DC was enabled for these results, meaning there is no JIT compilation overhead. We examine the impact of a variance reduction technique called implicit capture [8], which is supported in both codes. OpenMC is about 20 times faster than MC/DC for this problem with implicit capture. Without implicit capture that performance gap falls to 12 times. This performance gap is due to OpenMC's tracking algorithm, which does not stop particles at mesh crossing events as MC/DC currently does. OpenMC is currently limited to the use of the collision estimator to estimate the scalar flux spatial distribution in

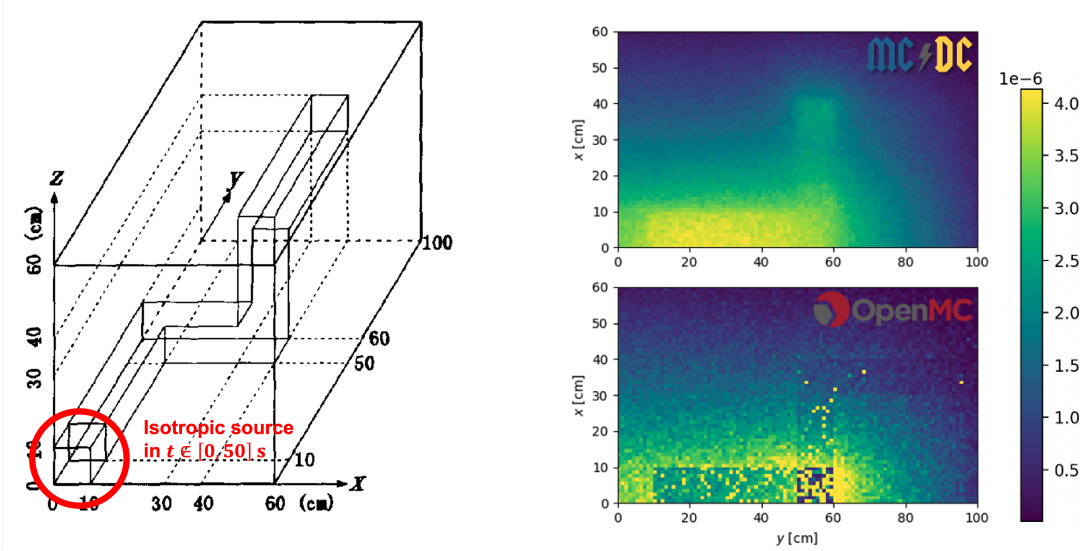

Figure 4: Left: Kobayashi problem schematic. Right: Time and space averaged scalar flux solution to the Kobayashi problem run with 1×10^9 particle histories from MC/DC (top) and OpenMC (bottom).

time-dependent simulation, whereas MC/DC uses the track length estimator [8], which is computationally more expensive but yields better statistics. This is shown in figure 4 at bottom right where OpenMC's result has tally cells with a poorly converged solution (displaying as yellow spots) and MC/DC's (top right) doesn't. Work is ongoing to implement this particle stop command. When this is enabled, we suspect the performance gaps for this problem will close substantially between MC/DC and OpenMC.

MC/DC is the only open-source Monte Carlo neutron transport code that can perform fully transient analysis for systems like the Kobyashi problem on GPUs. Figure 5 at right shows the wall clock runtime results of the same Kobyashi problem on a single Tesla V100 GPU on the Lassen supercomputer at LLNL as compared to MC/DC on 36 CPU cores (this time with caching disabled) on LLNL's Quartz supercomputer across particle counts and with-/without implicit capture. The JIT compilation overhead incurs a constant runtime before enough work saturates the GPU architecture. For the Kobyashi problem with implicit capture, the GPU versions experience a 21 times speed up with saturation occurring around 10^6 particles. For this problem without implicit capture, the GPU remains unsaturated up to 10^9 but incurs a maximum observed speedup of 24 times over on CPU. This speedup of a whole CPU node to a single GPU compares well to the performance of other production Monte Carlo neutron transport applications [11].

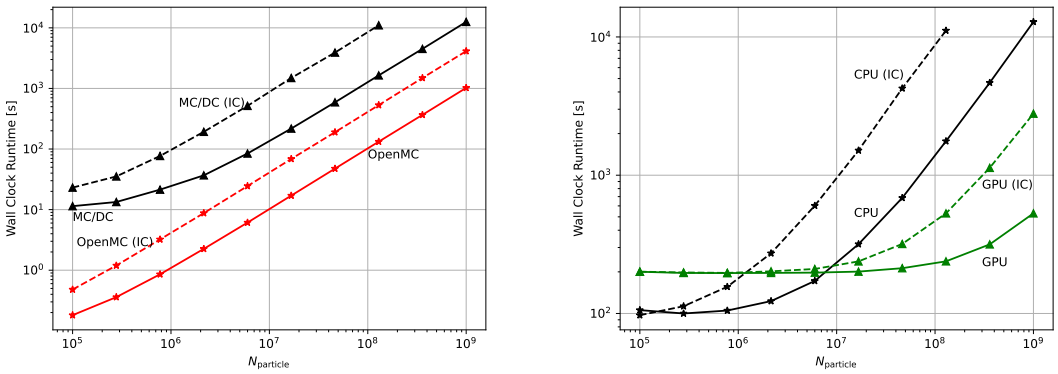

Figure 5: Left: Wall clock runtime of the Kobyashi problem over particle counts with and without implicit capture (IC) between cached MC/DC (black) and OpenMC (red). Right: Wall clock runtime of the Kobyashi problem over particle counts in uncached MC/DC CPU (black) and GPU (green) modes.

3 Discussions, Conclusions, and Future Work

Monte Carlo/Dynamic Code is a single-source, single-language, single-compiler Monte Carlo neutron transport code that targets modern HPC architectures with CPUs and GPUs. Our performance results demonstrate that MC/DC's structure (using a Python + Numba JIT + MPI scheme) can produce similar performance to other Monte Carlo neutron transport solvers. While MC/DC currently performs somewhat worse than other production code, we believe this is primarily due to algorithmic differences rather than architectural ones. MC/DC currently supports many novel transient transport methods and variance reduction techniques that are not supported in any other Monte Carlo neutron transport application.

Numba has been found to have performance deficiencies as compared to other high-level languages and portability frameworks in some workloads [12]. We initially explored other high-level language portability frameworks including PyCUDA and PyKokkos and found that for a Monte Carlo neutron test-bed Numba's performance was in-line with the others considered [13]. Numba has been described as being as difficult to develop in as a traditional low-level language for complex algorithms [14]. We agree with those findings that it can be as difficult—or even harder when debugging—as developing compiled code, but since we are primarily using Numba as a Python-based portability framework, any ease-of-development advantages are only added benefits. MC/DC's software engineering approach has the added benefit of unifying our glue language and kernel production language.

Work in MC/DC is ongoing. We are continually exploring novel variance

reduction techniques and adding new functionality. We will implement a no-stop mesh-boundary tracking algorithm, optimize our data structures, and explore methods of kernel profiling with the goal of identifying code hotspots sections of MC/DC for targeted optimizations. We also plan on adding better tracking schemes to OpenMC to improve it's solution for transient problems as well. On GPUs specifically, work is ongoing to support compilation to AMD GPUs via a patch to Numba that allows for AMD compilation and execution[1]. Intel also offers a patch to compile to their GPUs via Numba[2] and future work include building support for their accelerators as well. On all GPU types we also plan to extend support to multi-GPU simulations via existing MPI frameworks. We will continue to improve MC/DC, making it a portable application for rapid methods development enabled by Python and Numba.

4 Acknowledgments

This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number: DE-NA003967.

References

[1] M. Pozulp, R. Bleile, P. Brantley, S. Dawson, M. McKinley, and A. R. M. Y. M. O'Brien, "Progress Porting LLNL Monte Carlo Transport Codes to Nvidia GPUs," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada, 2023.

[2] F. D. Witherden, "Python at petascale with PyFR or: How I learned to stop worrying and love the snake," *Computing in Science & Engineering*, vol. 23, no. 4, pp. 29–37, 2021.

[3] 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*. Austin Texas: ACM, Nov. 2015, pp. 1–6.

[4] J. P. Morgan, I. Variansyah, S. Pasmann, K. B. Clements, B. Cuneo, A. Mote, C. Goodman, C. Shaw, J. Northrop, R. Pankaj, E. Lame, B. Whewell, R. McClarren, T. Palmer, L. Chen, D. Anistratov, C. T. Kelley, C. Palmer, and K. E. Niemeyer, "Monte Carlo / Dynamic Code (MC/DC): An accelerated Python package for fully transient neutron transport and rapid methods development," *Journal of Open Source Software*, vol. 95, p. 6415, 2024.

[1] https://github.com/ROCm/numba-hip
[2] https://github.com/IntelPython/numba-dpex

[5] I. Variansyah, J. P. Morgan, J. Northrop, K. E. Niemeyer, and R. G. McClarren, "Development of MC/DC: a performant, scalable, and portable Python-based Monte Carlo neutron transport code," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada, 2023.

[6] S. P. Hamilton and T. M. Evans, "Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code," *Annals of Nuclear Energy*, vol. 128, pp. 236–247, Jun. 2019.

[7] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, "OpenMC: A state-of-the-art Monte Carlo code for research and development," *Annals of Nuclear Energy*, vol. 82, pp. 90–97, 2015.

[8] E. E. Lewis and W. F. Miller, *Computational Methods of Neutron Transport*. New York, NY, USA: John Wiley and Sons, Inc., 1984.

[9] B. Cuneo and M. Bailey, "Divergence reduction in Monte Carlo neutron transport with on-GPU asynchronous scheduling," *ACM Trans. Model. Comput. Simul.*, oct 2023, just Accepted.

[10] K. Kobayashi, N. Sugimura, and Y. Nagaya, "3D radiation transport benchmark problems and results for simple geometries with void region," *Progress in Nuclear Energy*, vol. 39, pp. 119–144, 2001.

[11] J. P. Morgan, A. Mote, S. Pasmann, G. Ridley, T. S. Palmer, K. E. Niemeyer, and R. G. McClarren, "The Monte Carlo computational summit – October 25 & 26, 2023 – Notre Dame, Indiana, USA," *Journal of Computational and Theoretical Transport*, 2024.

[12] W. F. Godoy, P. Valero-Lara, T. E. Dettling, C. Trefftz, I. Jorquera, T. Sheehy, R. G. Miller, M. Gonzalez-Tallada, J. S. Vetter, and V. Churavy, "Evaluating performance and portability of high-level programming models: Julia, Python/Numba, and Kokkos on exascale nodes," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2023.

[13] J. P. Morgan, T. S. Palmer, and K. E. Niemeyer, "Explorations of Python-based automatic hardware code generation for neutron transport applications," in *Transactions of the American Nuclear Society*, vol. 126, Anaheim, CA, USA, 2022, pp. 318–320.

[14] S. Kailasa, T. Wang, L. A. Barba, T. Betcke, K. Hinsen, and A. Dubey, "PyExaFMM: An exercise in designing high-performance software with Python and Numba," *Computing in Science & Engineering*, vol. 24, no. 5, pp. 77–84, 2022.

A.4 Hybrid-Delta Tracking on a Structured Mesh in MCATK

This work was submitted to, accepted with mino

- Travis J. Trahan, PhD
- Timothy P. Burke, PhD
- Collin J. Josey, PhD
- Kyle E. Niemeyer, PhD

arXiv:2306.07847v1 [physics.comp-ph] 13 Jun 2023

Hybrid-Delta Tracking on a Structured Mesh in MCATK

J. P. Morgan[1,2,†], Travis J. Trahan[2,*], Timothy P. Burke[2], Colin J. Josey[2], and Kyle E. Niemeyer[1]

[1]School of Mechanical Industrial and Manufacturing Engineering
Oregon State University, 204 Rogers Hall, Corvallis, OR 97331

[2]XCP-3
Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545

[redacted] *tjtrahan@lanl.gov

ABSTRACT

Monte Carlo Application Toolkit (MCATK) commonly uses surface tracking on a structured mesh to compute scalar fluxes. In this mode, higher fidelity requires more mesh cells and isotopes and thus more computational overhead — since every time a particle changes cells, new cross-sections must be found for all materials in a given cell — even if no collision occurs in that cell. We implement a hybrid version of Woodcock (delta) tracking on this imposed mesh to alleviate the number of cross-section lookups. This algorithm computes an energy-dependent microscopic majorant cross section is computed for the problem. Each time a particle enters a new cell, rather than computing a true macroscopic cross-section over all isotopes in the cell, the microscopic majorant cross-section is simply multiplied by the total number density of the cell to obtain a macroscopic majorant cross-section for the cell. Delta tracking is then performed within that single cell. This increases performance with minimal code changes, speeding up the solve time by a factor of 1.5—1.75 for k-eigenvalue simulations and 1.2—1.6 for fixed source simulations in a series of materially complex criticality benchmarks.

KEYWORDS: monte carlo, delta tracking, mcatk, variance reduction, criticality benchmark

1. INTRODUCTION

The Monte Carlo Application ToolKit (MCATK) [1] is a Monte Carlo particle transport code that commonly uses surface tracking on a structured mesh (e.g., a Cartesian grid) to compute scalar fluxes. The structured mesh makes distance-to-boundary calculations cheap compared to simulations using a constructive solid geometry.

Many current simulations need a great number of isotopes to be modeled properly, such that a single mesh cell might contain many isotopes. MCATK's standard algorithm computes the total cross-section of all isotopes in a cell every time a particle moves between cells. Computationally expensive lookup and interpolation functions are called at every boundary crossing. When cell sizes are small compared to a neutron mean free path, often this cross section lookup finds that a particle does not collide in the cell. This process repeats until a collision occurs.

In this work, we introduce a hybrid-delta tracking algorithm to eliminate the macroscopic cross section lookup at surface crossings while still performing standard surface tracking.

2. HYBRID-DELTA TRACKING

Woodcock, or delta, tracking [2] is a variance-reduction technique that computes the majorant cross-section for the whole problem space, then uses this to determine a distance to collision for all particles. Coupled with rejection sampling to sort for phantom collisions, and a collision estimator to compute scalar flux, delta tracking often improves performance over analogue Monte Carlo in problems that warrant it. Many production Monte Carlo Neutron transport codes like Serpent [3] use this method.

We implement a hybrid surface-delta tracking algorithm to reduce the number of cross-section lookups while still using MCATK's surface tracking on a structured mesh to find scalar flux. First we start by computing the microscopic majorant cross-section over the whole problem space, like in traditional delta tracking:

$$\sigma_M(E) = \max\left(\sigma_{T,k}(E), ..., \sigma_{T,K}(E)\right) , \tag{1}$$

where E is energy, σ_M is the microscopic majorant cross-section, and $\sigma_{T,k}$ is the microscopic total cross-section of the k^{th} material. In this work, we use one majorant over the whole problem regardless of location, but regional microscopic majorants could still be used in future work. Now to sample a distance we calculate the distance to a collision

$$D = \frac{-\ln \xi}{N_i \sigma_M(E)} , \tag{2}$$

where ξ is a random number between zero and one and N_i is the number density in cell i. Thus far the number density of the cell is the only parameter that depends on the particle's location in the problem, so there is no need to look up and interpolate total cross-sections when tallying the distance traveled through the mesh before a collision. This is equivalent to assuming all atoms in the cell are of the type with the largest microscopic total cross-section at that energy.

If the potential collision occurs within the cell, we move the particle to the sampled distance and do rejection sampling, since we are now potentially forcing collisions that did not occur. We sort out these phantom collisions by allowing particles to continue to a new sampled distance if

$$\xi < \frac{\Sigma_{T,i}(E)}{N_i \sigma_M(E)} , \tag{3}$$

where ξ is a new random number between zero and one and $\Sigma_{T,i}(E)$ is the macroscopic total cross-section of the cell.

Since we are still moving the particles through each cell, we can use a track-length tally for estimating scalar flux. This may not offer as much raw speed-up as full delta tracking, but may result in greater efficiency, as a track-length estimator has a lower variance than a collision estimator [4].

Consider the region with three materials shown in blue, green, and pink presented in Figure 1. A particle enters the region from the left and undergoes transport with operations required at every distance to a boundary or collision:

1. the particle crosses into region 1 requiring that $\sigma_M(E)$ (pre-computed) is interpolated, multiplied by N_1 to form $\Sigma_{M,1}$, and a new distance to collision is computed;
2. the particle crosses into region 2 requiring that σ_M (pre-computed) is interpolated then multiplied by N_2 to form $\Sigma_{M,2}$, and a new distance to collision is computed;

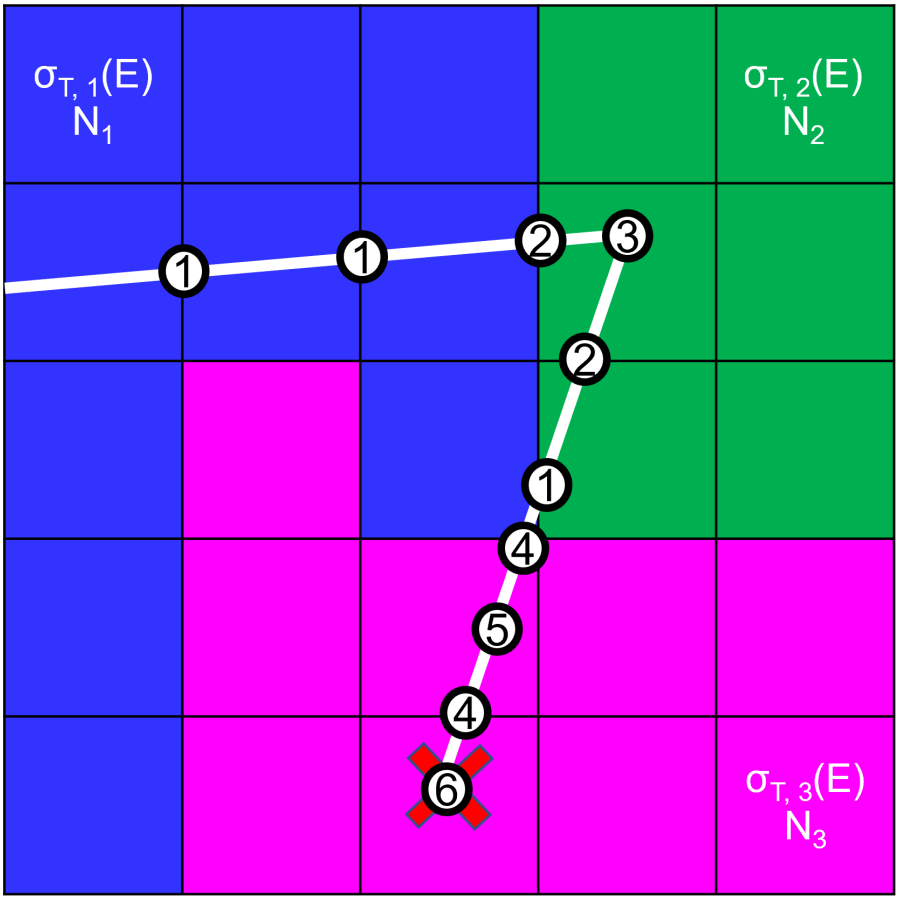

Figure 1: A material region in a structured mesh where each color represents a different material with a different N_i and $\sigma_T(E)$

3. the distance to collision is small enough to place a collision in the cell. Rejection sampling is done and it is found that this was a real event. Particle undergoes a scattering event;
4. the particle crosses into region 3 requiring that σ_M (pre-computed) is interpolated then multiplied by N_3 to form $\Sigma_{M,3}$, and a new distance to collision is computed;
5. the distance to collision is small enough to place a collision in the cell. A potential collision is rejected; and
6. the distance to collision is small enough to place a collision in the cell. Rejection sampling determines that the collision is real, and the particle is absorbed.

Here, $\sigma_M = \max(\sigma_{T,1}, \sigma_{T,2}, \sigma_{T,3})$. $\sigma_{T,i}$ is only looked up for rejection sampling, required in operations 3, 5 and 6. For this hypothetical problem we have reduced the number of lookups to find $\sigma_{T,i}$ from ten to three. Depending on the number of isotopes in the cell, this could represent a significant number of operations. Speed-up in MCATK with this method comes from the avoidance of these lookup functions to find the dominant total microscopic cross section between *all* the isotopes that are contained within a cell. We expect a significant speed-up as this function is called

frequently in standard surface tracking. We also expect that the speedup will be greater when there are more isotopes within a cell. All other operations remain the same between standard tracking and hybrid surface-delta tracking on the structured mesh.

Extending our implementation of the hybrid-delta tracking scheme in models that use constructive solid geometry instead of structured meshes will require no additional work. Since small cell sizes in structured meshes is analogous to complex and small geometries in constructive solid geometry, we expect similar performance increases when these models also have complex isotopic compositions.

We first implemented hybrid-delta tracking in MCATK's k-eigenvalue algorithm then implemented it for fixed source problems. Since all of MCATK's algorithms (e.g., k-eigenvalue, fixed source, etc.) use the same collision kernel no alterations were required to move from an implementation in one algorithm to the next other than Boolean switches to initialize the mini-delta tracking algorithm.

3. BENCHMARKS

We investigated the performance of hybrid-delta tracking on a series of benchmark problems: one from the Godiva IV experiment (HEU-MET-FAST-086, case 5) and two from the MUSiC experiment. All three are materially complex, having between 48 and 125 isotopes over the whole model, and implement a structured tracking mesh imposed on the geometry. The minimum mesh cell dimension is 1 mm in the fissile region of each problem.

The Godiva IV super-prompt-critical burst experiments use two control rods and one burst rod made of highly enriched uranium and molybdenum to control criticality [5]. As the rods are inserted reactivity goes up, and vice versa when they are removed. The only fission source is the spontaneous fission from the highly enriched uranium in the experiment. Figure 2 shows a cross section of the Godiva IV core assembly and restraints which are modeled in our benchmark. This is the simplest benchmark we model, requiring the fewest number of isotopes.

The Measurement of Uranium Subcritical and Critical (MUSiC) experiments (IER 488) use stackable hemispheres of highly enriched uranium — known as the Rocky Flats Shells — to take data with various detectors [7]. Figure 3 at right shows the highly enriched uranium shells which are about $0.3\,\mathrm{cm}$ thick and have variable radii between about $2\,\mathrm{cm}$ and $10\,\mathrm{cm}$. Figure 3 at left shows the full configuration of the experiment with fission sources and detectors. Fission is induced with a Cf-252 source at the center of the shell array as well as an external deuterium source.

4. PERFORMANCE RESULTS

We verified all models on both k-eigenvalue and fixed source problems with the standard tracking algorithm from MCATK, then computed the speed-up from our hybrid-delta tracking scheme. To produce timing results these benchmarks ran with 1 rank; however, parallelization is already enabled for hybrid-delta tracking in MCATK via MPI. Material, geometry, and mesh between k-eigenvalue and fixed source simulations are the same for a given case.

4.1. k-eigenvalue Simulation

K-eigenvalue simulations were started with 100 inactive cycles before 500 active ones using 1×10^5 particles in each cycle. Table 1 shows the performance increase when hybrid-delta tracking is

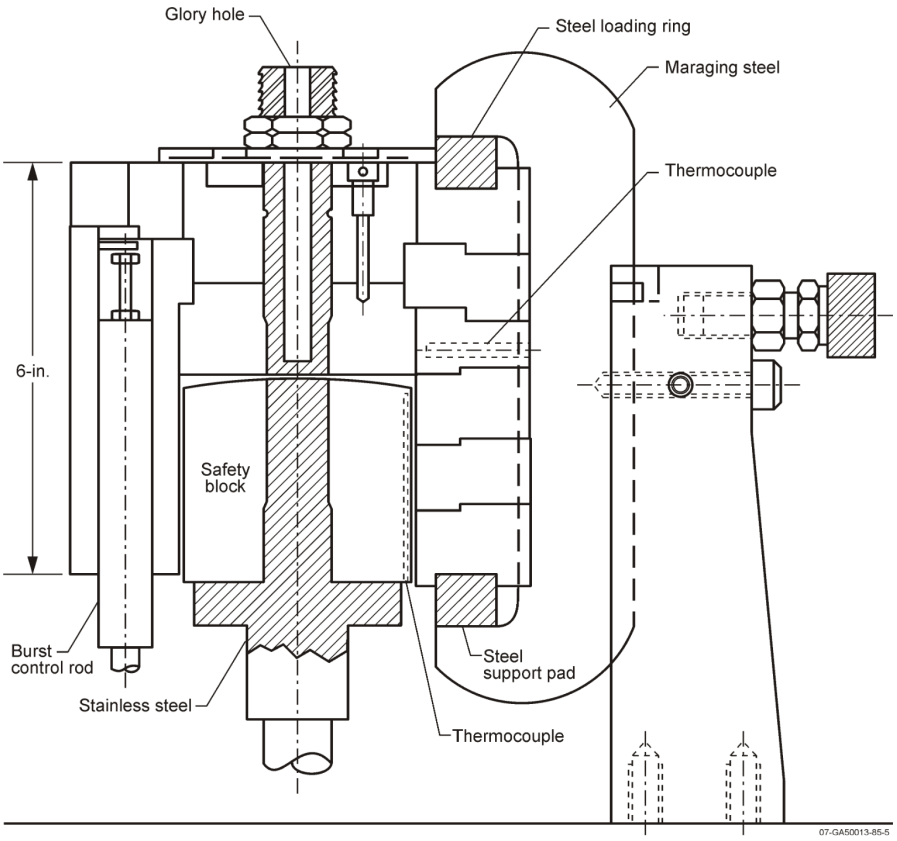

Figure 2: Cutaway view of the Godiva IV core and its restraints used in the HEU-MET-FAST-086 benchmark [6].

enabled. The differences in eigenvalue between simulations with and without the hybrid-delta tracking algorithm are all within 1.12 standard deviations. This table also shows a significant speed-up in the over all solve time of MCATK with delta tracking incurring between a 1.54 — 1.75$\times$ speed-up.

Table 1: Benchmark results: where $\Delta\sigma$ is the difference in number of standard deviations of k_{eff} between the standard algorithm and hybrid-delta tracking.

Test Model	MCATK Surface (s)	MCATK Hybrid-Delta (s)	k	$\Delta\sigma$	speed-up
Godiva Case 5	16900	10987	0.99736	-0.747	1.54
MUSiC Case 8	23937	14832	0.99970	0.130	1.61
MUSiC Case 9	22649	12973	0.99929	-1.105	1.75

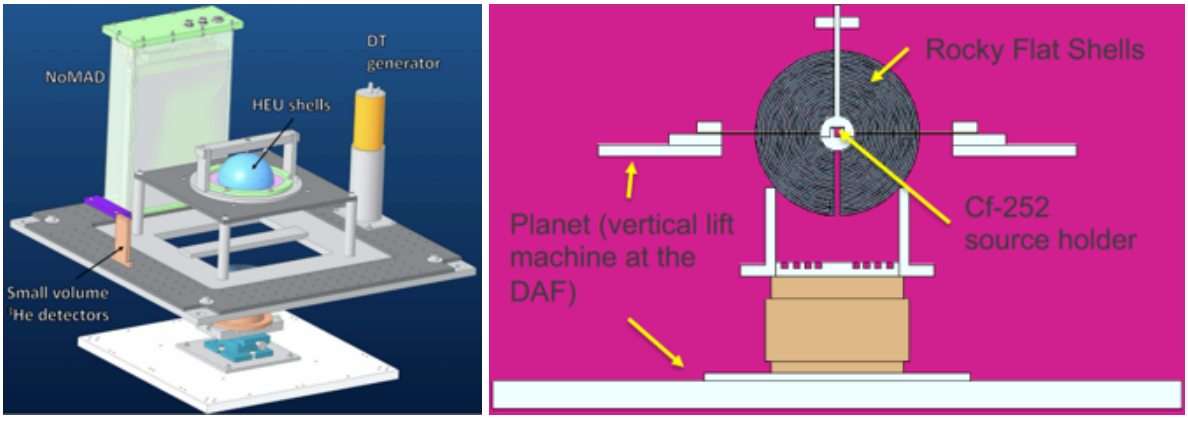

Figure 3: Left: Overall configuration of the MUSiC experiment. Right: Schematic of the Rocky Flats highly enriched uranium hemispherical shells [8]

4.2. Fixed Source Simulations

To compute error between tracking schemes in fixed source simulations we used the estimates of the α-eigenvalue computed using MCATK's time-dependent algorithm. The benchmarks were started at $0\,\mathrm{s}$ and ran to $500 \times 10^{-8}\,\mathrm{s}$ with a time step of $\Delta t = 1 \times 10^{-8}\,\mathrm{s}$. The particle population was combed between every time step up or down to 1×10^5 particles. Table 2 shows less speed-up then for k-eigenvalue computations (only between $1.24\times$ and $1.63\times$) but still significant for minimal alterations to a production code. This table also shows the average α-eigenvalues method within three standard deviations.

Table 2: Benchmark results: where $\Delta\sigma$ is the difference in number of standard deviations of the α-eigenvalues between standard algorithm and hybrid-delta tracking.

Benchmark	MCATK Surface (s)	MCATK Hybrid-Delta (s)	α_{avg}	$\Delta\sigma$	speed-up
Godiva Case 5	2689	2168	-3.54×10^{-3}	-1.91	1.24
MUSiC Case 8	4352	2665	-1.26×10^{-3}	-0.76	1.63
MUSiC Case 9	4440	2786	-1.02×10^{-3}	-2.75	1.59

Figure 4 at left shows spatial averaged scalar flux ($\bar{\phi}$) of the 0.1 to $1.0\,\mathrm{MeV}$ energy bin on an $y-z$ plane slice at $x = 0\,\mathrm{cm}$ in a high fidelity version (same simulation but with 1×10^8 particles per time step) of Godiva IV as a fixed source problem. The source in this simulation is an isotropic burst of neutrons at $(0, 0, 0)\ \mathrm{cm}$ and $t = 0.0\ \mathrm{s}$ with an energy of $1\,\mathrm{MeV}$. At right Figure 4 shows the relative error

$$\epsilon = \frac{\left|\bar{\phi}_{\Delta} - \bar{\phi}_{\text{Standard}}\right|}{\bar{\phi}_{\Delta}}, \tag{4}$$

where $\bar{\phi}_{\Delta}$ is the spatial averaged scalar flux solution from the hybrid-delta tracking scheme and $\bar{\phi}_{\text{Standard}}$ is from the standard tracking scheme. It shows that the relative error in this slice is small be-

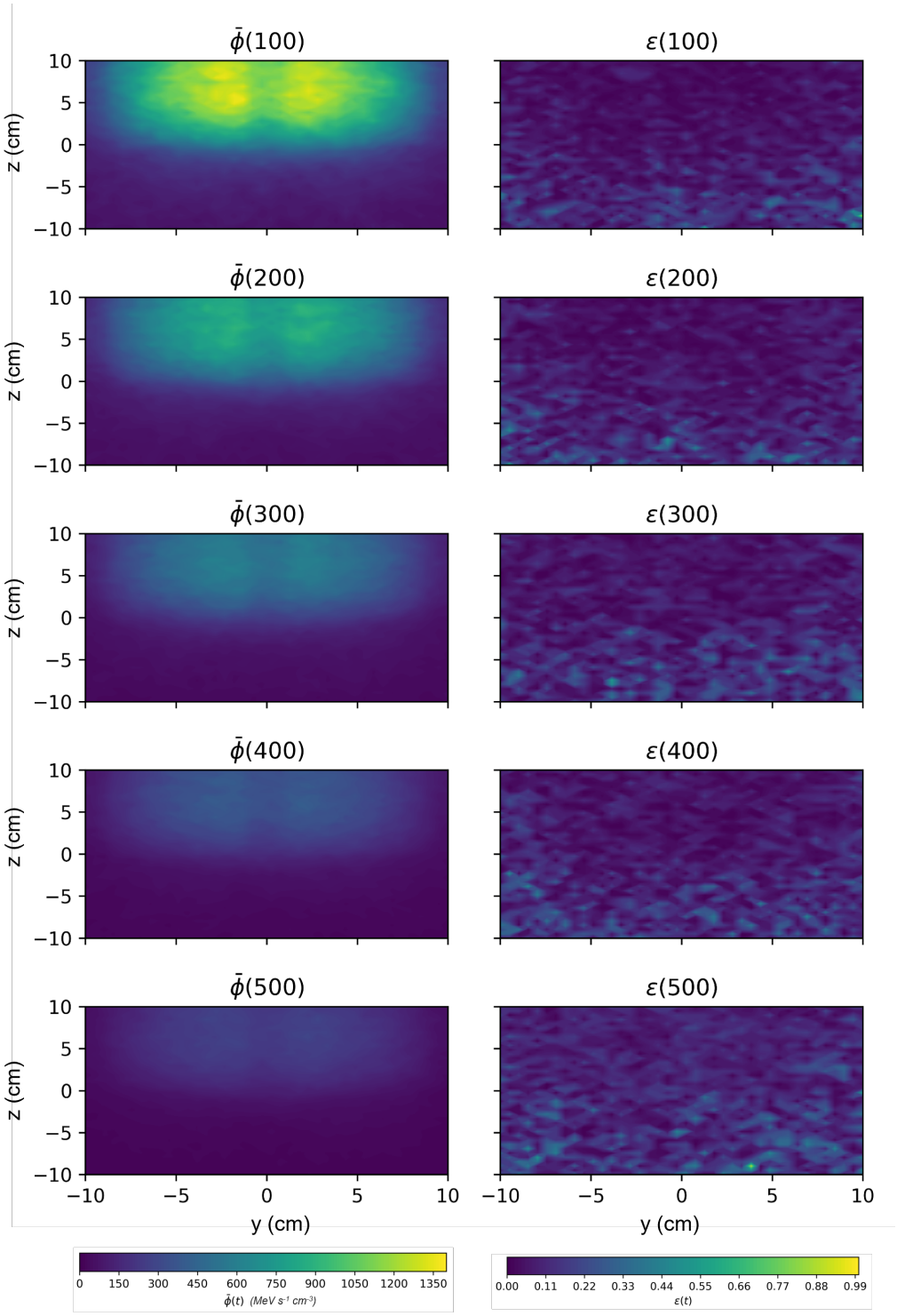

Figure 4: Right: Spatial averaged scalar flux of Godiva IV (HEU-MET-FAST-086) from MCATK. Left: Relative error between solution from standard tracking algorithm and hybrid-delta tracking. Shown through time (every $100 * 10^{-8}$s).

tween the two methods of tracking. This further demonstrates that a hybrid-delta tracking scheme is not biasing the results of MCATK's standard surface tracking algorithm in these models.

5. CONCLUSIONS & FUTURE WORK

Hybrid-delta tracking on a structured mesh improves run time in large and materially complex simulations like the ones we benchmarked: 1.5—1.75× speed-up for k-eigenvalue and 1.2—1.6× speed-up for fixed source problems. We expect further optimizations will improve speed-up for these algorithms.

The solutions found with hybrid-delta tracking match to within three standard deviations of solutions found with MCATK's standard algorithm. The advantages of this algorithm over full delta tracking on structured meshes are that track-length estimators may still be used, and that minimal the changes are required to the code as the treatment of the geometry does not change.

Further work is required to verify and validate the hybrid-delta tracking scheme. For k-eigenvalue problems we will use the criticality validation suite for MCNP [9] as well as other criticality benchmarks such as the Sub-critical Copper-Reflected α-phase Pu (SCRαP) experiment [10]. For fixed source solutions we will verify with the analytical AZURV1 [11] benchmark problems. We also plan to study how the algorithm performs as a function of time step size, since small time steps can suffer from poor performance in a similar manner as small cell sizes.

We expect that this work will increase the overall performance of MCATK when delta tracking is appropriately used in problems that warrant it.

ACKNOWLEDGEMENTS

Research presented in this article was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20220084DR.

This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number: DE-NA003967.

The authors would like to R. Arthur Forster for the discussions that inspired this work as well as Theresa Cutler, Travis Smith, and Robert Weldon Jr. for providing benchmark problem input files.

LA-UR-23-24971

REFERENCES

[1] T. Adams, S. Nolen, J. Sweezy, A. Zukaitis, J. Campbell, T. Goorley, S. Greene, and R. Aulwes. "Monte Carlo Application ToolKit (MCATK)." *Annals of nuclear energy*, **volume 82**, pp. 41–47 (2015).

[2] E. R. Woodcock, T. Murphy, P. Hemmings, and T. C. Longworth. *Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry*. Argonne National Laboratory, Lemont USA (1965).

[3] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho. "The Serpent Monte Carlo code: Status, development and applications in 2013." *Annals of Nuclear Energy*, **volume 82**, pp. 142–150 (2015).

[4] R. G. McClarren. *Computational Nuclear Engineering and Radiological Science Using Python*. Elsevier, Cambridge, USA (2018).

[5] R. D. Mosteller and J. M. Goda. "Analysis of Godiva-IV delayed-critical and static super-prompt-critical conditions." (2009).

[6] R. D. Mosteller and J. M. Goda. "Godiva-Iv Delayed-Critical Experiments and Description of an Associated Prompt-Burst Experiment." **volume II**, pp. 134–156 (2014).

[7] J. Goda. "LANL FY21 NCSP Highlights [Slides]." (2022).

[8] R. A. Weldon, Jr., T. E. Cutler, J. D. Hutchinson, W. L. Myers, G. E. McKenzie IV, A. T. McSpaden, and R. G. Sanchez. "MUSiC: Measurement of Uranium Subcritical and Critical (IER 488) [Slides]." (2021).

[9] R. Mosteller, F. Brown, and B. Kiedrowski. "An expanded criticality validation suite for MCNP." In *Transactions of the American Nuclear Society*, volume 104, pp. 453–455 (2011).

[10] J. B. Briggs, editor. *International Handbook of Evaluated Critical Safety Benchmark Experiments*. Nuclear Energy Agency, NEA/NSC/DOC(95)03/I, Paris, France (2020).

[11] B. Ganapol, R. Baker, J. Dahl, and R. E. Alcouffe. "Homogeneous Infinite Media Time-Dependent Analytical Benchmarks." In *International Meeting on Mathematical Methods for Nuclear Applications*, volume 41(25). Salt Lake City, UT (2001).

A.5 Draft: Therefore paper

Bibliography

[1] S. P. Hamilton and T. M. Evans, *Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code*, Annals of Nuclear Energy **128**, 236–247 (2019).

[2] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, *OpenMC: A state-of-the-art Monte Carlo code for research and development*, Annals of Nuclear Energy **82**, 90–97 (2015).

[3] M. F. Modest, *Radiative Heat Transport*, Academic Press, San Diego, CA, USA (2003).

[4] S. Chandrasekhar, *Radiative Transfer*, Dover, New York, NY, USA (1960).

[5] J. Duderstadt and L. Hamilton, *Nuclear Reactor Analysis*, John Wiley and Sons, Inc., New York, NY, USA (1974).

[6] E. E. Lewis and W. F. Miller, *Computational methods of neutron transport*, John Wiley and Sons, Inc., New York, NY, USA (1984).

[7] B. G. Carlson. *MORE PRECISE DEFINITION OF THE DISCRETE ORDINATES METHOD.* In *Second Conference on Transport Theory*, page 348, (1971).

[8] C. E. Lee. *THE DISCRETE Sn APPROXIMATION TO TRANSPORT THEORY.* Technical report, Los Alamos National Labratory, (1961).

[9] K. D. Lathrop, *Use of Discrete-Ordinates Methods for Solution of Photon Transport Problems*, Nuclear Science and Engineering **24**, 381–388 (1966).

[10] W. A. Fiveland, *Three-dimensional radiative heat-transfer solutions by the discrete-ordinates method*, Journal of Themophysics **2** (1988).

[11] J. S. Truelove, *Discrete-ordinate solutions of the radiation transport equation*, Journal of Heat Transfer (Transcations of the American Society of Mechanical Engineers) **109** (1987).

[12] M. L. Adams, *Subcell balance methods for radiative transfer on arbitrary grids*, Transport Theory and Statistical Physics **26**, 385–431 (1997).

[13] G. Colomer, R. Borrell, F. X. Trias, and I. Rodríguez, *Parallel algorithms for Sn transport sweeps on unstructured meshes*, Journal of Computational Physics **232**, 118–135 (2013).

[14] I. Variansyah, E. W. Larsen, and W. R. Martin, *A ROBUST SECOND-ORDER MULTIPLE BALANCE METHOD FOR TIME-DEPENDENT NEUTRON TRANSPORT SIMULATIONS*, EPJ Web of Conferences **247**, 03024 (2021).

[15] R. S. Baker, *An SN Algorithm for Modern Architectures*, Nuclear science and engineering **185**, 107–116 (2017).

[16] M. Rosa, J. S. Warsa, and M. Perks, *A Cellwise Block-Gauss-Seidel Iterative Method for Multigroup SN Transport on a Hybrid Parallel Computer Architecture*, Nuclear Science and Engineering **174**, 209–226 (2013).

[17] D. Y. Anistratov and Y. Y. Azmy, *Iterative stability analysis of spatial domain decomposition based on block Jacobi algorithm for the diamond-difference scheme*, Journal of Computational Physics **297**, 462–479 (2015).

[18] D. S. Hoagland and Y. Y. Azmy, *Hybrid approaches for accelerated convergence of block-Jacobi iterative methods for solution of the neutron transport equation*, Journal of Computational Physics **439**, 110382 (2021).

[19] S. Schunert, C. T. Garvey, R. Yessayan, and Y. Y. Azmy. *ANALYSIS OF COMMUNICATION PERFORMANCE DEGRADATION OF THE RADIATION TRANSPORT CODE PIDOTS ON HIGH-UTILIZATION, MULTI-USER HPC SYSTEMS*. In *PHYSOR: Physics of Reactors 2018*, Cancun, Mexico, (2018). ANS.

[20] M. L. Adams and E. W. Larsen, *Fast iterative methods for discrete-ordinates particle transport calculations*, Progress in Nuclear Energy **40**, 3–159 (2002).

[21] J. W. M. Rosa. *Extension of a Transport Synthetic Acceleration Scheme to the Cell-Wise Block-Jacobi and Gauss-Seidel Algorithms*. In *Transactions of the American Nuclear Society*, (2009).

[22] M. Rosa, J. S. Warsa, and J. H. Chang. *Fourier analysis of parallel inexact Block-Jacobi splitting with transport synthetic acceleration in slab geometry*. In *American Nuclear Society's Topical Meeting on Reactor Physics*, (2006).

[23] M. Rosa, J. S. Warsa, and J. H. Chang. *Fourier Analysis of Parallel Block-Jacobi Splitting with Transport Synthetic Acceleration in Two-Dimensional Geometry*. In *Nuclear Weapons Highlights*, (2007).

[24] G. L. Ramone, M. L. Adams, and P. F. Nowak, *A Transport Synthetic Acceleration Method for Transport Iterations*, Nuclear science and engineering **125**, 257–283 (1997).

[25] E. E. Lewis and W. F. Miller, *Computational methods of neutron transport*, John Wiley and Sons, Inc., New York, NY, USA (1984).

[26] C. T. Kelley and Z. Q. Xue, *GMRES AND INTEGRAL OPERATORS*, SIAM Journal of Scientific Computing **17** (1996).

[27] B. W. Patton and J. P. Holloway, *Application of preconditioned GMRES to the numerical solution of the neutron transport equation*, Annals of Nuclear Energy **29**, 109–136 (2002).

[28] J. S. Warsa, T. A. Wareing, and J. E. Morel, *Krylov Iterative Methods and the Degraded Effectiveness of Diffusion Synthetic Acceleration for Multidimensional SN Calculations in Problems with Material Discontinuities*, Nuclear Science and Engineering **147**, 218–248 (2004).

[29] J. S. Warsa, T. A. Wareing, J. E. Morel, J. M. McGhee, and R. B. Lehoucq, *Krylov Subspace Iterations for Deterministic k-Eigenvalue Calculations*, Nuclear Science and Engineering **147**, 26–42 (2004).

[30] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke, *Kokkos 3: Programming Model Extensions for the Exascale Era*, IEEE Transactions on Parallel and Distributed Systems **33**, 805–817 (2022).

[31] N. Al Awar, S. Zhu, G. Biros, and M. Gligoric. *A Performance Portability Framework for Python.* In *International Conference on Supercomputing*, pages 467–478, (2021).

[32] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science & Engineering **13**, 22–30 (2011).

[33] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, *PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation*, Parallel Computing **38**, 157–174 (2012).

[34] F. D. Witherden, A. M. Farrington, and P. E. Vincent, *PyFR: An Open Source Framework for Solving Advection-Diffusion Type Problems on Streaming Architectures using the Flux Reconstruction Approach*, Computer Physics Communications **185**, 3028–3040 (2014).

[35] F. D. Witherden, M. Klemm, and P. Vincent. *PyFR: Heterogeneous Computing on Mixed Unstructured Grids with Python.* In *EuroSciPy*, Cambridge, (2015).

[36] F. D. Witherden, *Python at Petascale With PyFR or: How I Learned to Stop Worrying and Love the Snake*, Computing in Science & Engineering **23**, 29–37 (2021).

[37] D. Häfner, R. Nuterman, and M. Jochum, *Fast, Cheap, and Turbulent—Global Ocean Modeling With GPU Acceleration in Python*, Journal of Advances in Modeling Earth Systems **13**, e2021MS002717 (2021).

[38] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. *JAX: composable transformations of Python+NumPy programs*, (2018).

[39] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis. *CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations*. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, (2017).

[40] L. G. Silvestri, L. J. Stanek, G. Dharuman, Y. Choi, and M. S. Murillo, *Sarkas: A fast pure-python molecular dynamics suite for plasma physics*, Computer Physics Communications **272**, 108245 (2022).

[41] 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*, pages 1–6, Austin Texas, (2015). ACM.

[42] J. P. Morgan, T. S. Palmer, and K. E. Niemeyer. *Explorations of Python-Based Automatic Hardware Code Generation for Neutron Transport Applications*. In *Transactions of the American Nuclear Society*, volume 126, pages 318–320, Anaheim, CA, USA, (2022).

[43] I. Variansyah, J. P. Morgan, J. Northrop, K. E. Niemeyer, and R. G. McClarren. *Development of MC/DC: a performant, scalable, and portable Python-based Monte Carlo neutron transport code*. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada, (2023).

[44] J. P. Morgan, I. Variansyah, S. L. Pasmann, K. B. Clements, B. Cuneo, A. Mote, C. Goodman, C. Shaw, J. Northrop, R. Pankaj, E. Lame, B. Whewell, R. G. McClarren, T. S. Palmer, L. Chen, D. Y. Anistratov, C. T. Kelley, C. J. Palmer, and K. E. Niemeyer, *Monte Carlo / Dynamic Code (MC/DC): An accelerated Python package for fully transient neutron transport and rapid methods development*, Journal of Open Source Software **9**, 6415 (2024).

[45] C. f. E. M. C. N. Transport (CEMeNT), I. Variansyah, J. P. Morgan, S. Pasmann, K. Clements, B. Cuneo, A. Mote, C. Shaw, J. Northrop, R. Pankaj, R. G. McClarren, T. S. Palmer, L. Chen, D. Y. Anistratov, C. T. Kelley, C. Palmer, and K. E. Niemeyer. *MC/DC: Monte Carlo Dynamic Code*, (2024).

[46] E. R. Woodcock, T. Murphy, P. Hemmings, and T. C. Longworth, *Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry*, Argonne National Laboratory, Lemont USA (1965).

[47] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho, *The Serpent Monte Carlo code: Status, development and applications in 2013*, Annals of Nuclear Energy **82**, 142–150 (2015).

[48] J. Leppänen, *On the use of delta-tracking and the collision flux estimator in the Serpent 2 Monte Carlo particle transport code*, Annals of Nuclear Energy **105**, 161–167 (2017).

[49] J. Leppänen, *Performance of Woodcock delta-tracking in lattice physics applications using the Serpent Monte Carlo reactor physics burnup calculation code*, Annals of Nuclear Energy **37**, 715–722 (2010).

[50] K. L. Rowland, R. M. Bergmann†, R. N. Slaybaugh, and J. L. Vujie. *Delta-tracking in the GPU-accelerated WARP Monte Carlo Neutron Transport Code.* In *Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Jeju, Korea, (2017).

[51] R. G. McClarren, *Computational Nuclear Engineering and Radiological Science Using Python*, Elsevier, Cambridge, USA (2018).

[52] C. J. Roy, *Review of code and solution verification procedures for computational simulation*, Journal of Computational Physics **205**, 131–156 (2005).

[53] A. K. P. James S. Warsa, Jeffery D. Densmore and J. E. Morel, *Manufactured Solutions in the Thick Diffusion Limit*, Nuclear Science and Engineering **166**, 36–47 (2010).

[54] J. Wang, W. Martin, and B. Collins, *The application of Method of Manufactured Solutions to method of characteristics in planar geometry*, Annals of Nuclear Energy **121**, 295–304 (2018).

[55] F. N. G. II, Y. Wang, D. Gaston, and R. Martineau. *The Method of Manufactured Solutions for RattleSnake, A SN Radiation Transport Solver inside the MOOSE Framework.* In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, volume 106, pages 372–374, (2012).

[56] J. P. Morgan, I. Variansyah, T. S. Palmer, and K. E. Niemeyer. *Exploring One-Cell Inversion Method for Transient Transport on GPU.* In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagra Falls, ON, CA, (2023). CNS.

[57] L. Dalcin and Y.-L. L. Fang, *mpi4py: Status Update After 12 Years of Development*, Computing in Science & Engineering **23**, 47–54 (2021).

[58] B. Cuneo and M. Bailey, *Divergence Reduction in Monte Carlo Neutron Transport with On-GPU Asynchronous Scheduling*, ACM Trans. Model. Comput. Simul. **34** (2023).

[59] J. Goda, C. Bravo, T. Cutler, T. Grove, D. Hayes, J. Hutchinson, G. McKenzie, A. McSpaden, W. Myers, R. Sanchez, and J. Walker, *A New Era of Nuclear Criticality Experiments: The First 10 Years of Godiva IV Operations at NCERC*, Nuclear science and engineering **195**, S55–S79 (2021).

[60] J. Goda. *LANL FY21 NCSP Highlights [Slides]*, (2022).

[61] R. Kimpland, T. Grove, P. Jaegers, R. Malenfant, and W. Myers, *Critical Assemblies: Dragon Burst Assembly and Solution Assemblies*, Nuclear Technology **207**, S81–S99 (2021).

[62] J. (Jia) Hou, K. N. Ivanov, V. F. Boyarinov, and P. A. Fomichenko, *OECD/NEA benchmark for time-dependent neutron transport calculations without spatial homogenization*, Nuclear Engineering and Design **317**, 177–189 (2017).

[63] K. Kobayashi, N. Sugimura, and Y. Nagaya, *3D radiation transport benchmark problems and results for simple geometries with void region*, Progress in Nuclear Energy **39**, 119–144 (2001).