Explorations of Monte Carlo Solution and Implementation Methods for Thermal Radiation and Neutron Transport

by

Joanna Piper Morgan

A PROJECT REPORT

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 12th, 2022
Commencement June 2023

ACKNOWLEDGEMENTS

I would like to first thank my committee members: Prof. Joshua Gess whose booming voice and philosophical side conversations leave me smiling even on the most frustrating of days, Prof. Todd Palmer whose patience in the face of my copious *less-then-knowledgeable* questions has surly earned him a place among the saints, and my advisor Prof. Kyle Niemeyer who is an exemplar academic and represents many of the things I wish to emulate in my own life. You have made me feel welcome at this institution and without you I would not have been able to make it this far.

I would like to thank my coauthors and all those I work with at Los Alamos National Lab whose professionalism and public service mentality gives me nothing but confidence in our Nation. I would also like to thank my fellow researchers in the Center for Exascale Monte Carlo Neutron Transport and my lab-mates in Niemeyer Research Group who are not only some of the smartest and well equipped individuals in their fields, but the best set of good humored colleges a novice academic could ask for.

I would like to thank my friends across my life from high school, my undergraduate institution, internships, and graduate school, you all have made the hard days bearable and the good days phenomenal. It goes without saying that I'd like to thank my teachers, professors, and mentors throughout my education who took steps and made sacrifices to steer me true.

Finally I'd like to thank my family: All my cousins, aunts, and uncles; my grandparents: Evan, Alice Faye and Maureen; my siblings: Brooks & Amelia, Conor Chelsea & their kids, Maison, Marissa, Ethan, Spencer and of course my mom Mary who fought to keep me alive when I was born and still keeps alive with games of canasta, and my dad Lee who has taught me everything I ever really needed including long division.

I am incredibly lucky to have so many beautiful people to thank. I am also lucky to say that this is only the half-way mark of my graduate studies and I am over-zealously excited to continue this work with you all. Tho there might be changes in my life going forward I know that through truth and your counsel I will find deliverance. Thank you.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

this above all: to thine own self be true,
and it must follow, as the night the day,
thou canst not be false to any man

Hamlet

Chapter 1: Introduction

This project consists of explorations into both software methods of implementation and numerical methods in neutron and thermal radiation transport, in both production and research codes. The central point of these endeavours has been to answer the call of the ever-present need to go faster: faster and more accurate numerical methods, and faster implementation of numerical methods into HPC-worthy code.

Here two published conference proceedings are presented that have made contributions in both of these efforts. While directly relating to the applied field of nuclear engineering, aspects of this work have ramifications for computations across all disciplines including mechanical engineering and its sub-discipline of thermal-fluid sciences. This introduction presents some background in these topics.

1.1 Neutron Transport

Neutron transport is required when modeling nuclear reactor physics as it describes where and how neutrons produce fission reactions which in turn generate heat in the fuel rods [1]. Modeling this is necessary for both the analytical design and safety analysis of a reactor. 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:

$$\frac{1}{v(E)}\frac{\partial\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)}{\partial t}+\hat{\boldsymbol{\Omega}}\cdot\nabla\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)+\Sigma_t(r,E,t)\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)= \int_{4\pi}\int_0^\infty\Sigma_s(\boldsymbol{r},E'\rightarrow E,\hat{\boldsymbol{\Omega}}'\rightarrow\hat{\boldsymbol{\Omega}},t)\psi(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)dE'd\hat{\boldsymbol{\Omega}}'+s(\boldsymbol{r},E,\hat{\boldsymbol{\Omega}},t)\ , \quad (1.1)$$

where ψ is the angular flux, v is the velocity of the particles, Σ_t is the total material cross section, Σ_s is the 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 $0 < t$. We also prescribe the initial condition

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

and the boundary condition

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

1.2 Thermal Radiation Transport

Thermal radiation transport shows how photons are emitted (via plankian emission), scattered, and absorbed in a given system. This can be important for modeling radiative heat transfer in a system when when the gray assumption the diffusive assumption or non participating media assumption cannot be made. As emission of photons is governed by the temperature of the material, two coupled equations can be written where one describes the transport of particles trough space

$$\frac{1}{c}\frac{\partial I(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, t)}{\partial t} + \hat{\boldsymbol{\Omega}} \cdot \nabla I(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, t) + \sigma I(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, t) = \sigma B(E, T) + \frac{Q_r(\boldsymbol{r}, E, t)}{4\pi} \tag{1.4}$$

and the other is an energy balance between the material and the emission of photons

$$c_v(\boldsymbol{r}, t)\frac{\partial T(\boldsymbol{r}, t)}{\partial t} = \int_0^\infty \int_{4\pi} (I(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, t) - B(E, T)) d\Omega' dE' + Q_m(\boldsymbol{r}, t) \tag{1.5}$$

where I is the specific intensity, B is Planck's function for radiation, c_v is the specific heat of the material, T is the material temperature, Q_r is the in-homogeneous photon source, Q_m is the in-homogeneous material energy source, c is the speed of light, and σ is the absorption opacity for $\boldsymbol{r} \in V$, $\hat{\boldsymbol{\Omega}} \in 4\pi$, $0 < E < \infty$ and $0 < t$ [2]. We also prescribe the initial conditions

$$I(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, 0) = I_{initial}(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}) \tag{1.6}$$

$$T(\boldsymbol{r}, 0) = T_{initial}(\boldsymbol{r}) \tag{1.7}$$

and the boundary condition

$$I(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, t) = I_{bound}(\boldsymbol{r}, E, \hat{\boldsymbol{\Omega}}, t) \text{ for } \boldsymbol{r} \in \partial V \text{ and } \hat{\boldsymbol{\Omega}} \cdot \boldsymbol{n} < 0 \tag{1.8}$$

1.3 Direct Simulation Monte Carlo Solution Method

Whereas deterministic numerical methods (e.g., forward Euler, Crank–Nicholson, Gaussian Quadrature) are applied to the governing integro-partial differential equations themselves, direct simulation Monte Carlo (or just Monte Carlo) is used to simulate the actual subatomic particles and how they interact with a given system [3]. Here known material composition and interaction rates of the particles at all speeds can be leveraged with pseudo-random numbers to individually simulate a particles life, aka history.

Take for example a shielding problem: where a beam of neutrons is impinging a slab of purely absorbing material. Based off a cumulative probability distribution function we can supply a pseudo-random number between zero and one and the total material cross section (a known perimeter) to sample a distance to a collision. We then compare the known width of the slab and the sampled distance to see if the particle made it through or was absorbed within. If this process is repeated a *sufficient* number of times a statistically prevalent solution can be produced.

As the simulation grows more complex (e.g. scattering events, fission events, track length estimators, etc.) the computational work needed to simulate one neutron life time will go up. Also as Monte Carlo methods converge at a rate of $O = \frac{1}{\log n}$ a massive number of particles may be required to gain a *converged* solution. All this is to say when using the Monte Carlo method on modern compute architectures it is often imperative to parallelize the algorithm.

Novel MC TRT Method: Vectorizable Variance Reduction for Energy Spectra

J. P. Morgan[1,2], Alex Long[3], Kendra Long[3], and Kyle E. Niemeyer[1,2]
[1] Oregon State University
[2] Center for Exascale Monte Carlo Neutron Transport
[3] Los Alamos National Laboratory

Transactions of the American Nuclear Society
Vol. 126, 276-278, 2022.
https://doi.org/doi.org/10.13182/T126-38066

Chapter 2: Novel MC TRT Method: Vectorizable Variance Reduction for Energy Spectra

2.1 Introduction

Several production-scale multigroup Thermal Radiation Transport (TRT) codes use the Monte Carlo method as their primary solution technique. One such example is *Jayenne* from Los Alamos National Lab, which samples a single energy group for a particle then transports it [4]. In this work we created a novel TRT variance-reduction method to better resolve the energy spectra with large group structures and fewer particles, with a goal to decrease computational time at a given fidelity of solution in the energy spectra.

2.2 TRT Equations

The explicit TRT equations discretized in frequency using a multigroup approximation, and in zero-dimensional space, are [5]

$$\frac{1}{c}\frac{\partial I_g(t)}{\partial t} + \sigma_{a,g}(T)I_g(t) = x_g(T)B_g(T) \tag{2.1}$$

$$c_v\frac{dE}{dt} = \sum_0^G \sigma_g(T)I_g(t) = x_g(T)B_g(T) \tag{2.2}$$

for time $t > 0$ and the number of groups $G \geq 1$, where $I_g(T)$ is the specific intensity in a given group g, c is the speed of light, $\sigma_{a,g}(T)$ is the absorption opacity in a given group g, $B_g(T)$ is Planck's function for group g, c_v is the specific heat of the material (which is assumed constant), and $x_g(T)$ is the Planck-weighted opacity for group g.

Note that we discretize the material equation in time using a forward Euler scheme [6]:

$$T_{n+1} = T_n + \frac{\Delta t}{c_v\rho}\frac{dE_n}{dt}\ , \tag{2.3}$$

where n is the time index and ρ is the material density. This means error is proportional

to $\mathcal{O}(\Delta t)$.

2.3 Flocking Particles

We propose a method in which a single pseudo-particle carries a vector of energy weights, representing particles across all energy groups in a given simulation. This could be thought of as a "flock" of particles all moving together through space, angle, and time. We implement continuous energy deposition so a distance to event is found by comparing the distance to a spatial cell boundary and a distance associated with the time left in a step (effectively the temporal cell boundary) and selecting the minimum between the two. We are considering several ways to implement physical scattering, which is not currently implemented in this work. A scheme with explicit Euler does not introduce "effective scattering" and physical scattering is generally much smaller than absorption opacity.

We also required this method to be time dependant by using a particle census to move particles between time steps. A known, desired number of particles is used to find the census population (particles brought forward from the last time step) and emission population (particles produced by material emission within a time step) governed by their ratio of total energy. If the population of the census is larger than its allocation, then we use Russian Roulette for population control (based on Legrady et al. [7]). This addresses the unbounded particle growth that happens when implementing continuous energy methods.

A Roulette normally does not conserve either total energy or total group energy for a finite number of particles; our method requires both. This is due to our desire to conserve spectral shape to hopefully lead to further variance reduction. This requirement means that if the energy weight for a given group across all particles was computed, it would be the same before and after the Roulette, as well as total energy of all particles. Figure 1 **??** shows the population Roulette, where a vector of energy weights is accumulated for each Rouletted particle in each group. That accumulation vector is then divided by the number of particles remaining at the end of the Roulette. And finally this per particle accumulation vector of energy weights is added to the energy of the still living census particles, group by group. Histories will only terminate when met with the end of the problem or the Roulette.

After a distance to event is sampled, the energy weights in **all** groups must be

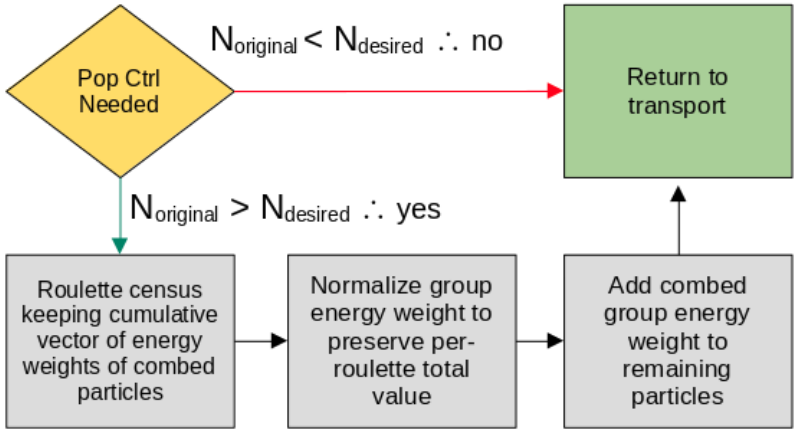

Figure 2.1: Flowchart of the Russian Roulette population control process as implemented in our method.

attenuated, requiring the computation of an exponential function, and the manipulation of energy weights in **all** groups. This is done independently as if it were a normal particle in that energy group by

$$w_g^{n+1} = w_g^n e^{-\sigma_g d} \,, \tag{2.4}$$

where w is the energy weight of a particle, n is the time index, g is the group number, and d is the distance to event (same distance across all groups).

Consider a 200-group problem using our novel technique: instead of computing a single exponential attenuation as in the single-group MC scheme, we must repeat this exponential calculation 200 times (once for each frequency group in the problem). Thus for *one* pseudo-particle's history, in *one* time step, the number of exponentials required is now proportional to the number of groups, hindering effective performance gains.

However, there is still hope. The attenuation calculation involves applying the same operation to multiple discrete pieces of data; said another way, the attenuation calculation is an example of single instruction/multiple data (SIMD) processing. SIMD hardware (a type of vector-processing unit) is widely deployed, and can be found in production x86 CPUs with AVX instructions as well as machines purpose-built to operate on vectors.

It is also relatively easy to enable through the use of compiler flags on modern C++ compilers [8]. As a result, this method will be able to use accelerators that already exist and commonly remain idle, to speed up our novel method and dampen the hit to performance.

The method presented here is similar to, though distinct from, a method described by McKinley, Brooks, and Szoke [9]. They apply a vector-of-weights approach to the Symbolic Implicit Monte Carlo (SIMC) algorithm, while this method is applied directly to a time-explicit discretization of the TRT equations. We also investigate this method for specific use on vector hardware.

2.4 Results and Discussion

We developed a test code in C++ to examine this method's performance, first as a time-dependent zero spatial dimension problem, then as a time-dependent single spatial dimension problem. For both we ran gray and multigroup test cases and compared results to analytical solutions (if available) as well as the *Jayenne* Implicit Monte Carlo code from Los Alamos National Lab.

2.4.1 Zero-dimensional problem

The zero-dimensional test is a simple time-dependent equilibration problem. We compared a gray case to both the analytical Mosher [10] solution as well as solutions from Jayenne. Then, we performed simulations with up to 200 energy groups in Jayenne, examining both time to equilibration and the spectrum of the results.

Table 2.1 shows that to get roughly the same fidelity of solution our novel method required the simulation of only 1×10^4 particles, while the production code required 1×10^6 particles. These were computed against a benchmark solution from the production code of 5×10^6 particles. With these results we felt confident to move our novel method into one dimension.

Table 2.1: L2 norms between Jayenne and Flocking for the zero-dimensional solution. Flocking at 1×10^4 particles is as accurate as Jayenne at 1×10^6.

Time Step	Jayenne (10^4)	Jayenne (10^6)	Flocking (10^4)
1000	1.03×10^{-1}	4.60×10^{-3}	3.66×10^{-3}
2000	7.59×10^{-2}	4.53×10^{-3}	3.83×10^{-3}

2.4.2 One-dimensional problem

To compare the results of our testbed with those from Jayenne, we employ a Marshak wave test, again starting from the gray case, then moving to multigroup with up to 200 groups. The test case is a 1 cm long slab of iron ($c_v = 0.1\,\mathrm{GJ/(g\,keV)}$, $\rho = 1\,\mathrm{g/cm^3}$), where the material and radiation temperatures are initially at equilibrium at 1×10^{-6} keV. The left boundary is an infinite plane wall source, with temperature 3 keV and the right boundary is a vacuum. We used a time step of $\Delta t = 10^{-12}$ s and a mesh width of $\Delta x = 0.005$ cm As time goes forward we expect a temperature wave to propagate through the material from left to right. As time goes to infinity we expect the solution to come to equilibrium.

After confirming that the temperature profiles over time matched both our expectations and solutions from Jayenne, we moved to examine their spectra in specific mesh cells at specific time steps. Figure 2.2 shows the spectrum produced by either code at ten-thousand particles compared to a benchmark solution of the production code ran with 5 million particles. The spectral solution from Jayenne is very noisy (lots of jagged peaks) while the spectrum produced by our novel method is smooth and accurate compared to the benchmark solution at this low particle count. This demonstrates that the variance reduction works.

However, a slight deviation of the spectral shape near the peak is observed. We believe this stems from the use of spatial tilt for source position sampling in Jayenne that has not yet been implemented in our test-bed code. When implemented we expect this discrepancy to disappear and we will be able to use a traditional Figure of Merit comparison to fully demonstrate the novel method's variance reduction abilities.

We must also consider run times. Table 2.2 confirms that this method is vectorizable, and will be able to take advantage of the specialized vector processing components of

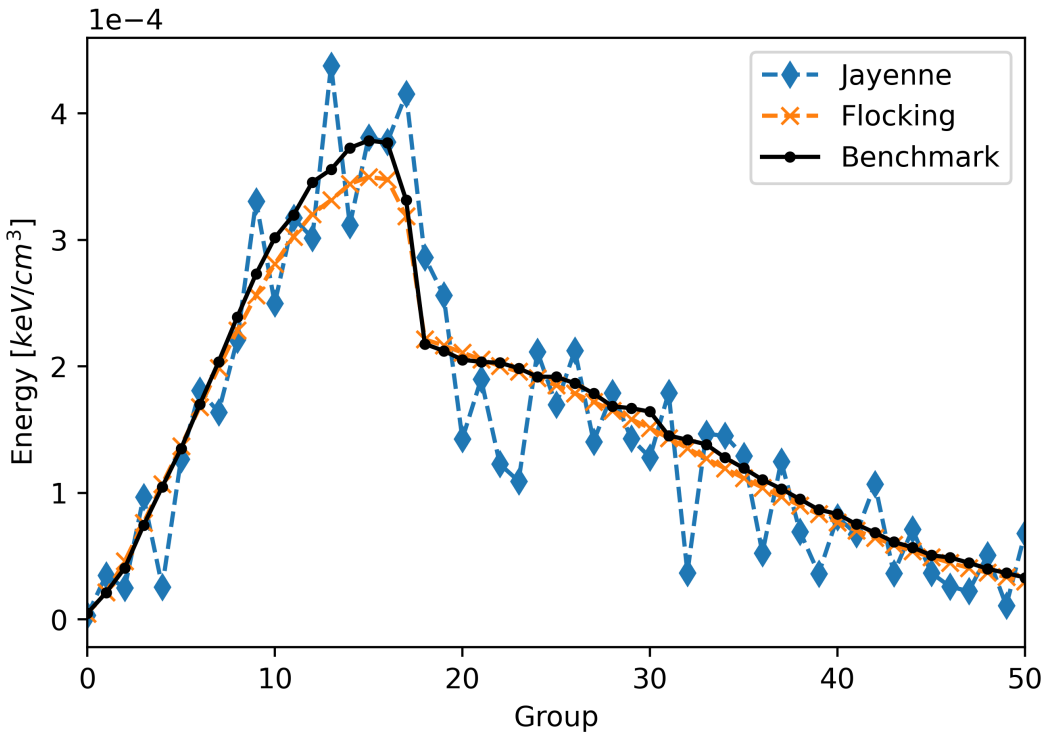

Figure 2.2: Energy spectra from a single cell at a single time step produced with 1×10^4 particles to show the high-variance solution produced from Jayenne, and the low-variance solution produced by our novel method. They are plotted against a benchmark solution from Jayenne computed with 5×10^6 particles.

Table 2.2: Run-times of the 1D novel method test bed under various compiler flag conditions. Simulations run on a single node of Intel Skylake Gold processors.

Optimized?	Vectorized?	Runtime [s]
no	no	1855.2
yes	no	353.2
yes	yes	184.7

Table 2.3: Comparing the run-times of the 1D test case between Jayenne and flocking test bed.

Method	# particles	Runtime [s]
Jayenne	1×10^4	70.97
Jayenne	1×10^6	3005.50
flocking	1×10^4	92.0
flocking	1×10^6	3775.12

modern CPUs. It also shows that enabling vectorization has a huge impact on run time for this method; cutting runtime for a fully optimized executable (using the Intel icpc compiler flag `-Ofast`) in half when they are turned on (using a SIMD reduction flag above the attenuation loop).

To confirm that this method will result in an overall performance increase when considering the energy spectra, we raced the production code against the novel method at various particle counts and examined their spectra. Table 2.3 shows that the novel method does take slightly longer. However, if a well resolved spectrum is the goal of the computation then we have effectively reduced the computational time as fewer particles are required to get a well defined solution. While direct comparisons to production codes can be fraught, we feel this demonstrates that if this method is implemented in Jayenne the figure of merit will increase.

2.5 Conclusions

We have successfully demonstrated a novel variance reduction technique for the energy spectra for Monte Carlo Thermal Radiation Transport. Continued work is required before the method is stable enough to implement on production codes. Specifically we need to implement physical scattering, source tilting [2], and post-collisions group splitting (to prevent highly unlikely interactions across various energy regimes in optically thin material) in the testbed.

2.6 Acknowledgments

Release number: LA-UR-22-20908.

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.

Explorations of Python-Based Automatic Hardware Code Generation for Neutron Transport Applications

J. P. Morgan, Todd S. Palmer, and Kyle E. Niemeyer
Oregon State University
Center for Exascale Monte Carlo Neutron Transport

Transactions of the American Nuclear Society
Vol. 126, 318-320, 2022.
https://doi.org/doi.org/10.13182/T126-38137

Chapter 3: Explorations of Python-Based Automatic Hardware Code Generation for Neutron Transport Applications

3.1 Introduction

Monte Carlo / Dynamic Code (MC/DC) is a soon-to-be released Python-based neutron transport solver, developed as part of CEMeNT (Center for Exascale Monte Carlo Neutron Transport). It is a research code used to investigate novel methods for the development of dynamic simulations. As Monte Carlo (MC) neutron transport applications are often highly computationally taxing, a Python-based technique to rapidly gain parallelism for both GPU and CPU hardware is required so methods implemented in MC/DC can be tested at the high performance computing (HPC) scale.

Our objective is to find a technique that presents the best software engineering solution to enable rapid prototyping for methods research in MC/DC at the HPC scale using both CPUs and GPUs. We are not looking to alter current development dynamics for production HPC codes or suggest that these techniques will enable Python-based development—or should be pursued—for all production codes. Instead, we seek to examine production techniques for rapidly developing novel methods at large scales with the ability to take advantage of accelerator hardware.

We implemented core components of MC/DC in a testbed to allow for even faster exploration of parallelism in a Python framework. This testbed, MC/DC-Toy Neutronics Testbed (MC/DC-TNT), is a transient, event-based, mono-energetic solver that enables straightforward parallelization of its compute functions. While the performance of large-scale MC codes is generally memory-bound rather than compute-bound [11], our initial efforts examine techniques related to the latter to gain a foothold in Python-based HPC.

3.2 Methods of Parallelization

We examine three methods of parallelization within a Python framework. All three operate by using Python as “glue” language (a language which is used to run other

compiled kernels) with a just-in-time (JIT) compilation scheme to produce and run compute kernels. All three are also coupled with automatic code translation protocols, to aid in the rapid development of these kernels, either with the source being written in full Python or some Python-esq form.

3.2.1 PyKokkos

PyKokkos [12] is a Python library implementation for the `C++` Kokkos HPC portability model developed by Sandia National Laboratories [13]. PyKokkos is under active development and does not implement all the functionality of Kokkos, nor is it as mature as some of the other methods presented here. PyKokkos currently only implements CUDA and OpenMP back-ends.

More than any other method we consider, PyKokkos has the ability to abstract away the production of compute kernels from their target hardware. It even has the ability to abstract data structures from the user, appearing almost as type-annotated NumPy [14] arrays. While the programming model is entirely Python-based (the user only programs in Python) it can be non-intuitive, as it is based on a paradigm specifically designed for abstraction.

3.2.2 Numba + CUDA

Numba [15] is a translator and compiler for Python that implements the LLVM high performance compiler library, and promises "C-like speeds" for functions that implement it. In simple cases, nothing more than a compiler flag is required to produce a C kernel for a given function that is automatically bound and run by Python. This can even apply to simple functions to be parallelized on a CPU, where parallelization can be implemented with one flag option above a function. However, when functions become more complex, the use of more Numba commands in the Python code is required. It directly supports Nvidia GPUs and seems to be the method of choice for interfacing with the CUDA API from Python.

Recently a sub-module to Numba has been developed called `pyomp` [16] that seeks to allow Numba kernels to take advantage of the OpenMP API for CPU threading. We explore this sub-module and Numba's native "threading" techniques separately from one

another as they require slightly different implementations.

While Numba does have some abstractions for specific GPU operations, most of these cannot be leveraged to offload the computational burden in our simulations. In our use case the user will effectively have to write CUDA C kernels within Python. Thus, the user will have to be aware of the hardware architecture, hampering rapid prototyping. Also, Oden found that Numba does include a significant computational overhead, reaching 50–85% of the performance of pure CUDA C versions of compute-intensive benchmarks [17]. In spite of this, we include it in this comparison due to its increasing popularity, ease of use for CPU implementations, and industry support.

3.2.3 Mako Templating Engine

This method has been implemented by the PyFR computational fluid dynamics solver [18] and uses the Mako template engine [19] to abstract and simplify the scripting of compute kernels.

At run time, this approach pushes static kernel templates through the template engine to reform the kernel into a requisite secondary language for a given hardware API. Then, code is compiled to machine code and bound to Python using a hardware code-generating library.

This type of abstraction, while somewhat convoluted, has enabled PyFR to reach the petascale while being able to run on CPUs and AMD/Nivdia/Intel GPUs [20]. The developers of PyFR have reported less than 1% of computational overhead due to the Python interpreter in this method [21].

We are confident this method will work in parallelizing MC/DC-TNT at HPC scale; however, the initial development needed appears to be significant. While the use of templates does simplify the scripting of compute kernels, architecture-specific knowledge is still required, again, hampering rapid prototyping.

3.3 Preliminary Findings

Work is ongoing to implement all methods. Currently Numba (threading and PyOmp) and Pykokkos (OpenMP) are implemented on MC/DC-TNT. Table 3.1 shows run time data for the `Advance` kernel (the most compute-intensive function, which implements

a surface tracking algorithm), as well as a full integration test for a homogeneous slab problem with vacuum bounds on either side (L=1 cm, $\Delta x = 0.01$ cm, $\Sigma_a = \Sigma_f = \Sigma_s = 0.333\,1/\text{cm}$,).

All techniques demonstrate the viability of these parallelization methods with a significant decreases in run time between pure Python and any parallelization technique. As some kernels must be implemented in serial, an integration test for a slab problem shows all of the methods' parallelization abilities as well as the serial speed benefits of JITed code when compared to a pure Python implementation. PyKokkos does seem to be more performant than both Numba implementations, with Numba PyOmp being less performant than Numba threading. PyOmp has not yet been incorporated into the main distribution of Numba and as such does not receive the continual updates that Numba threading does. We expect that, when it is, we will see this gap between performance of the two implementations narrow.

JIT compilation methods require an initial compilation step when they are being run for the first time on a given system with a given code (any alteration to kernel source code will require this to be redone). Often these compiled binaries are cached in order to prevent this extra runtime hit when ran subsequently. However, depending on problem size, the compilation time can be a significant percentage of overall runtime and should be considered. Table 3.2 shows overall compilation time when "warming up kernels" (making an initial call with dummy values) before the simulation starts, per method and hardware target. PyKokkos OpenMP takes 7 times longer than the Numba implementations to translate and compile the kernels though all methods finish a matter of seconds.

We also consider the difficulty of these implementations when coming from a pure Python code. The Numba CPU implementation was easy for simpler kernels, functioning as advertised with a single compiler flag. More complex kernels, especially ones designed to be run in parallel, had to be reformed and use in kernel numba commands to work. Also, we found that when Numba wasn't able to produce kernels for any reason it's error messages directing us to the issues where either cryptic or nonexistent. We found working within Pykokkos to be more difficult than Numba for the CPU implementation. Fueled again by cryptic runtime errors, frustrating type casting issues, and a lack of documentation both for the build process as well as module commands. We expect as PyKokkos matures these issues will be alleviated. It should also be stated the allure of PyKokkos

Table 3.1: Run time of `Advance` kernel and integration test in seconds, using 1×10^8 particles and 16 threads (if method is parallelized); kernel compilation time not included.

Method	Advance	Integration
Pure Python (CPU)	5.140×10^4	5.297×10^4
Numba (Threading)	1.887×10^2	2.323×10^2
Numba (PyOmp)	2.876×10^2	3.825×10^2
PyKokkos (OpenMP)	1.480×10^2	1.548×10^2

Table 3.2: First run translation and compilation wall-clock times for all kernels, in seconds.

Method	Compilation time
Numba (Threading)	4.99
Numba (PyOmp)	5.66
PyKokkos (OpenMP)	37.50
PyKokkos (CUDA)	39.72

is not only a slight increase of performance on CPU implementations when compared to Numba but also in its portability, with promises of CUDA GPU implementations with changing a single variable. Early implementations of PyKokkos CUDA on MC/DC-TNT does seem to confirm this.

3.4 Future Work

Significant work remains before we can select a method to implement in MC/DC. Our immediate goals are to complete the PyKokkos CUDA and Numba CUDA implementations before finally moving onto the Mako templating engine method. We will also implement the AZURV1 [22] transient benchmark to further validate MC/DC-TNT as well as examine performance under different nuclear material data regimes.

3.5 Acknowledgments

The authors acknowledge useful discussions with Dr. Ilham Variansyah.

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.

Bibliography

[1] JJ Duderstadt and LJ Hamilton, *Nuclear Reactor Analysis*, First (Wiley, New York, NY, 1971).

[2] AB Wollaber, "Four Decades of Implicit Monte Carlo", Journal of Computational and Theoretical Transport **45**, 1–70 (2016) 10.1080/23324309.2016.1138132.

[3] EE Lewis and WF Miller, *Computational methods of neutron transport*, First (American Nuclear Society, La Grange Park, IL, USA, 1993).

[4] K Thompson, M Cleveland, A Long, K Long, R Wollaeger, B Ryan, and T Kelley, *Jayenne physics manual, revision 1.0 - an implicit monte carlo code for thermal radiative transfer*, tech. rep. (Aug. 2021), 10.2172/1818098.

[5] G Pomraning, *The Equations of Radiation Hydrodynamics*, 1st, Vol. 54, International Series of Monographs in Natural Philosophy (Pergamon Press, Oxford and New York, Oct. 1973).

[6] RG McClarren, *Computational nuclear engineering and radiological science using python* (Academic Press, 2018), pp. 381–406, 10.1016/B978-0-12-812253-2.00024-8.

[7] D Legrady, M Halasz, J Kophazi, B Molnar, and G Tolnai, "Population-based variance reduction for dynamic Monte Carlo", Annals of Nuclear Energy **149**, 107752 (2020) https://doi.org/10.1016/j.anucene.2020.107752.

[8] OpenMP Architecture Review Board, *OPENMP API Specification: Version 5.0*, 2018.

[9] MS McKinley, ED Brooks, and A Szoke, "Comparison of implicit and symbolic implicit monte carlo line transport with frequency weight vector extension", Journal of Computational Physics **189**, 330–349 (2003) 10.1016/S0021-9991(03)00213-4.

[10] SW Mosher, "Exact solution of a nonlinear, time-dependent, infinite-medium, grey radiative transfer problem", Transactions of the American Nuclear Society **95**, 744–746 (2006).

[11] JR Tramm and AR Siegel, "Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes", Annals of Nuclear Energy **82**, 195–202 (2015) 10.1016/j.anucene.2014.08.038.

[12] NA Awar, S Zhu, G Biros, and M Gligoric, "A performance portability framework for python", in Proceedings of the international conference on supercomputing (June 2021), pp. 467–478, 10.1145/3447818.3460376.

[13] H Carter Edwards, CR Trott, and D Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns", Journal of Parallel and Distributed Computing **74**, 3202–3216 (2014) 10.1016/j.jpdc.2014.07.003.

[14] CR Harris, KJ Millman, SJ van der Walt, R Gommers, P Virtanen, D Cournapeau, E Wieser, J Taylor, S Berg, NJ Smith, R Kern, M Picus, S Hoyer, MH van Kerkwijk, M Brett, A Haldane, JF del Río, M Wiebe, P Peterson, P Gérard-Marchant, K Sheppard, T Reddy, W Weckesser, H Abbasi, C Gohlke, and TE Oliphant, "Array programming with NumPy", Nature **585**, 357–362 (2020) 10.1038/s41586-020-2649-2.

[15] SK Lam, A Pitrou, and S Seibert, "Numba: A LLVM-Based Python JIT Compiler", in Proceedings of the second workshop on the llvm compiler infrastructure in hpc, LLVM '15 (2015), 10.1145/2833157.2833162.

[16] TG Mattson, TA Anderson, G Georgakoudis, K Hinsen, and A Dubey, "PyOMP: Multithreaded Parallel Programming in Python", Computing in Science and Engineering **23**, 77–80 (2021) 10.1109/MCSE.2021.3128806.

[17] L Oden, "Lessons learned from comparing C-CUDA and Python-Numba for GPU-Computing", in Proceedings - 2020 28th euromicro international conference on parallel, distributed and network-based processing, pdp 2020 (2020), pp. 216–223, 10.1109/PDP50117.2020.00041.

[18] FD Witherden, AM Farrington, and PE 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) 10.1016/j.cpc.2014.07.011.

[19] M Bayer, *Mako: Templates for python*, 2013.

[20] FD Witherden, "Python at petascale with PyFR or: how I learned to stop worrying and love the snake", Computing in Science & Engineering **9615**, 1–1 (2021) 10.1109/mcse.2021.3080126.

[21] FD Witherden, M Klemm, and P Vincent, "PyFR: Heterogeneous Computing on Mixed Unstructured Grids with Python", in Euroscipy (2015).

[22] B Ganapol, R Baker, J Dahl, and RE Alcouffe, *Homogeneous Infinite Media Time-Dependent Analytical Benchmarks*, Salt Lake City, UT, 2001.

We are not now that strength which in old days
Moved earth and heaven, that which we are, we are,
One equal temper of heroic hearts,
Made weak by time and fate, but strong in will
To strive, to seek, to find, and not to yield.

Tennyson