

**On-node Performance Optimization of a  
Monte-Carlo Transport Code for Leadership  
Architectures**

by

José Luis Salcedo-Pérez

B.S., Chemical Engineering (2015)

University of Puerto Rico, Mayagüez campus (UPRM)

Submitted to the Department of Nuclear Science and Engineering  
in partial fulfillment of the requirements for the degree of  
Master of Science in Nuclear Science and Engineering  
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

© Massachusetts Institute of Technology 2019. All rights reserved.

Author .....

Department of Nuclear Science and Engineering  
May 17, 2019

Certified by .....

Benoit Forget  
Professor of Nuclear Science and Engineering  
Thesis Supervisor

Certified by .....

Kord Smith  
KEPCO Professor of the Practice of Nuclear Science and Engineering  
Thesis Supervisor

Accepted by .....

Ju Li  
Battelle Energy Alliance Professor of Nuclear Science and Engineering  
and Professor of Materials Science and Engineering Chair, Department  
Committee on Graduate Students



# On-node Performance Optimization of a Monte-Carlo Transport Code for Leadership Architectures

by

José Luis Salcedo-Pérez

Submitted to the Department of Nuclear Science and Engineering  
on May 17, 2019, in partial fulfillment of the  
requirements for the degree of  
Master of Science in Nuclear Science and Engineering

## Abstract

The tally system in Monte Carlo neutron transport codes accounts for a significant fraction of the total execution time. This project studied the tally performance of a Monte Carlo neutron transport code (i.e., OpenMC) and implemented several optimizations to address the major bottlenecks. First, a comprehensive profiling analysis was carried out on modern Intel micro-architectures (i.e., Intel Xeon Phi and Intel Xeon Platinum 8180) to understand what hardware and settings configurations were optimal. The specific modules and subroutines that were responsible for the performance drop were also highlighted. The first round of optimizations were specific to the information that the profiling analysis provided. Both the nuclide and the reaction index searches were found to be inefficient. As a result, the two searches were improved with the implementation of direct address tables, which have a single search efficiency of  $O(1)$  and a small memory footprint. Moreover, a linear array cache was also introduced to store the following cross-sections:  $(n, 2n)$ ,  $(n, 3n)$ ,  $(n, 4n)$ ,  $(n, p)$ ,  $(n, \alpha)$ , and  $(n, \gamma)$ . These cross-sections, together with  $(n, fission)$ , are indispensable to solve the transition matrix of the Bateman equations during transmutation analysis. As a result, pre-computing and storing them before tally-time eliminated redundant computations in the case a high energy particle travels through multiple fuel regions without colliding. Overall, these optimizations resulted in speedups of 2.31x and 2.15x for the Xeon Platinum and Xeon Phi, respectively. Further, this project also presents an alternative method to compute reaction rate tallies. In general, tallying all of the aforementioned seven rates through a Monte Carlo simulation can be quite expensive for realistic light water reactors. Another approach would be to collapse a very fine-group flux together with a pregenerated multigroup cross-section (constructed with the same energy grid). While this approach does provide a 3x speedup in the OpenMC active cycles performance, it also introduces a considerable memory penalty. The issue is that thousands of groups are needed to accurately resolve the  $(n, \gamma)$  rates, most notably that of  $^{238}\text{U}$ . This study explores a hybrid approach in which  $(n, \gamma)$  and  $(n, fission)$  are handled with a standard reaction rate tally while the remaining reaction rates are computed through the flux tally route. This

option provides more flexibility in reducing the total number of groups because the remaining reactions outside of  $(n, fission)$  and  $(n, \gamma)$  usually have smoother shapes. Performance was tested on five benchmarks with depleted fuel and increasing geometrical complexity. Results showed that the hybrid tally method provided decent speedups ranging from 1.30x to 1.75x in the active cycles across all benchmarks. Multiple error analyses were also carried out on the proposed hybrid method; the results show that even when going as low as 300 groups, the eigenvalue is still within 100 pcm of a traditional simulation.

Thesis Supervisor: Benoit Forget

Title: Professor of Nuclear Science and Engineering

Thesis Supervisor: Kord Smith

Title: KEPCO Professor of the Practice of Nuclear Science and Engineering

## Acknowledgments

I want to thank both Professor Forget and Professor Smith for giving me a chance to be part of the Exascale Computing Project as a graduate student in the Computational Reactor Physics Group (CRPG). I appreciate their great mentoring, patience and support throughout this whole project. I definitely got a lot out of our group meetings because their input and suggestions were very useful to solve many of the problems I encountered. Next up, I want to thank Jingang Liang (CRPG Post-doc extraordinaire from Tsinghua University) and Paul Romano (Staff Scientist at Argonne National Lab, OpenMC main developer and CRPG alumni) for helping me deal with the more technical aspects of this project. More specifically, for the time they set aside to teach me about the details of the OpenMC source code. I am afraid I might have seriously abuse their generosity; over the past two years, I probably account for the bulk of Paul's messages on Slack as well as for the majority of Jingang's unscheduled meetings (he was my office-mate, sorry, couldn't help it!). On a similar note, I would also like to thank William Boyd (Senior Software Engineer at MITRE and CRPG alumni) for helping me get started with research during the Winter/Spring of 2017. He was available every Friday morning for two hours (sometimes more) to walk me through many of the important concepts I needed to catch-up on in order to finish my project. Not only did he guided me along technical topics that were indispensable for my thesis (like profiling), but he also gave me many advise on how to organize my research, presentations and the way I approach problems in general. Patience is a virtue, and he has a lot of it. I would also like to thank the CRPG for providing good advise on dealing with bugs and graduate school classes.

I want to add a special thanks to Lisa Magnano-Bleheen for checking-up on me every now and then with small-talk conversations and for helping me with setting up the group meetings with Prof. Forget and and Prof. Smith.

I am also grateful to the many advisor/supervisor pairs I had before graduate school started for preparing me for this challenge by giving me a shot to be part of their groups, and for grooming me with intensive work, conference presentations,

manuscript preparations, and many other experiences that helped me improve my research skills: Prof. Oscar Marcelo-Suárez/Sujeily Soto-Medina, M.Sc. (UPRM), Prof. María Curet-Arana/Dr. Brian Montejo-Valencia (UPRM), Prof. Heather Kulik/Dr. Efthymios Ioannides (MIT), Dr. Maciej Haranczyk/Dr. Nils Zimmermann (Lawrence Berkeley National Lab), and Prof. Robert Langer/Dr. James Norman (MIT).

I am also in great debt to my family, specially my parents Luis Salcedo-Fleriz and Damaris Pérez-Vidal, as well as my two siblings, Luis A. Salcedo-Pérez and Liz M. Salcedo-Pérez, for their support and encouragement throughout graduate school. ¡Gracias por todo, preparénme la cena, que regreso pronto!

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration. This material was based on work supported by the U.S. DOE, Office of Science, under contract DE-ACO2-06CH11357.

# Contents

<b>1</b>	<b>Overview</b>	<b>13</b>
1.1	Tallies in Monte Carlo Neutron transport simulations . . . . .	15
1.2	Brief Overview of Cache Hierarchy . . . . .	18
1.3	Previous optimization efforts for OpenMC . . . . .	20
1.4	Performance Optimization based on Profiling Analysis and Hybrid Tallies	25
<b>2</b>	<b>Profiling Analysis</b>	<b>29</b>
2.1	Profiling Test Cases . . . . .	31
2.2	The Phi and Skylake . . . . .	32
2.3	Profiling Analysis . . . . .	35
2.3.1	First approach . . . . .	35
2.3.2	Second Approach . . . . .	40
2.4	Optimizations . . . . .	47
2.4.1	Optimization 1 . . . . .	48
2.4.2	Optimization 2 . . . . .	50
2.4.3	Optimization 3 . . . . .	53
2.4.4	Performance Measurements . . . . .	54
2.4.5	Concluding Remarks . . . . .	55
<b>3</b>	<b>Hybrid Tallies</b>	<b>57</b>
3.1	Methodology . . . . .	57
3.2	Results and discussion . . . . .	60
3.2.1	Pin-cell Performance Assessment . . . . .	60

3.2.2	Pin-cell Error Analysis . . . . .	61
3.2.3	Assembly Comparison . . . . .	67
3.2.4	Performance Comparison across all Benchmarks . . . . .	70
3.3	Nuclear Data Inconsistencies . . . . .	75
3.3.1	$(n, p)$ and $(n, \alpha)$ levels . . . . .	75
3.3.2	Reconstructed Cross-section versus ACE . . . . .	75
3.3.3	Window Multipole Library . . . . .	76
<b>4</b>	<b>Conclusions</b>	<b>79</b>
4.1	Profiling Analyses and First Optimizations . . . . .	79
4.2	Hybrid Tallies . . . . .	80
4.3	Future Work . . . . .	82



# List of Figures

2-1	LWR test problems OpenMC . . . . .	30
2-2	SMR model. . . . .	31
2-3	Phi schematic representation . . . . .	34
2-4	Summary of VTune General Exploration Analysis . . . . .	42
2-5	Call graph of key functions in OpenMC. . . . .	43
2-6	Performance profiling memory-access for the top hotspots subroutines in OpenMC. . . . .	44
2-7	Hotspots analysis of the score_tracklength_tally function in the tally module. . . . .	45
2-8	Hotspots analysis of the score_general_ce function in the tally module.	45
2-9	Hotspots analysis of the expand_and_score function in the tally module.	46
2-10	Hotspots analysis of the calculate_nuclide_xs function in the cross- section module. . . . .	46
2-11	Hotspots analysis of the calculate_nuclide_xs function in the cross- section module . . . . .	47
3-1	$(n, 2n)$ plots for all 423 nuclides. . . . .	63
3-2	$(n, 3n)$ plots for all 423 nuclides. . . . .	64
3-3	$(n, p)$ plots for all 423 nuclides. . . . .	64
3-4	$(n, \alpha)$ plots for all 423 nuclides. . . . .	64
3-5	$k_{\text{eff}}$ versus time . . . . .	68
3-6	$^{235}\text{U}$ and $^{239}\text{Pu}$ Concentration as a Function of Time. . . . .	68
3-7	$k_{\text{eff,error}}$ in pcm units versus time . . . . .	69

3-8	$^{58}\text{Co}$ ( $n, p$ ) cross-section comparison between the original ACE data and the version reconstructed from the levels. . . . .	76
3-9	$^{154}\text{Gd}$ ( $n, \alpha$ ) cross-section comparison between the original ACE data and the version reconstructed from the levels. . . . .	77

# List of Tables

1.1	Memory hierarchy capacity and clock cycles to service memory request	19
1.2	Definition of variables for equation 1.9 . . . . .	25
1.3	Estimated Memory Penalty Per Depletion Region of Tallies in Previous Studies . . . . .	28
2.1	Hardware details for frequently used nodes . . . . .	32
2.2	OpenMC's tally slowdown for the SMR benchmark . . . . .	36
2.3	Processing Throughput for the SMR benchmark . . . . .	37
2.4	OpenMC performance as a function of Phi clustering mode . . . . .	37
2.5	OpenMC performance as a function of the Phi memory modes . . . . .	38
2.6	Fresh and depleted fuel comparison on a Phi node . . . . .	39
2.7	Fresh and depleted fuel comparison on a Skylake node . . . . .	39
2.8	CPU execution time breakdown across most expensive modules. . . . .	42
2.9	Efficiency of OpenMC key subroutines . . . . .	44
2.10	Global Nuclide Array . . . . .	50
2.11	Direct Address Table for Nuclide Search . . . . .	50
2.12	Simplified Example of a Reaction Index Dictionary Hash Table . . . . .	52
2.13	Simplified Example of a Reaction Index Direct Address Table . . . . .	53
2.14	Particle Processing Throughput Comparison using Optimizations (Skylake node). . . . .	55
2.15	Particle Processing Throughput Comparison using Optimizations (Phi node). . . . .	55
3.1	Simple 2,500-Group Energy Structure . . . . .	58

3.2	Performance Comparison for Pin-Cell MC Simulations with Different Tallies. All flux tallies have 2500 groups. . . . .	61
3.3	Pin-cell Error Analysis for simple 500 and 2500 Group Structures . . . . .	62
3.4	Updated Energy Structures with 300 and 500 Total Groups . . . . .	65
3.5	$(n, 2n)$ $\epsilon$ metric errors above the cutoff for 500 simple structure . . . . .	65
3.6	Error Analysis for updated 500 and 300 Group Structures. . . . .	65
3.7	$(n, \alpha)$ $\epsilon$ metric errors above the cutoff for 300 updated structures . . . . .	66
3.8	$(n, 2n)$ $\epsilon$ metric errors above the cutoff for 300 updated structure . . . . .	66
3.9	Performance Comparison Between Direct RR tally and Hybrid tally 5 for Assembly geometry . . . . .	69
3.10	Assembly Error Analysis for simple 500 and 2500 Group Structures. . . . .	70
3.11	Performance Comparison across all Benchmarks using Hybrid 5 . . . . .	73
3.12	Definition of variables for Table 3.11 and equations 3.4 to 3.6 . . . . .	74
3.13	Performance Comparison for the detailed SMR using either a Single or Multiple Generations per batch . . . . .	74
3.14	$\omega$ and $\epsilon$ errors for $^{58}\text{Co}$ $(n, p)$ and $^{154}\text{Gd}$ $(n, \alpha)$ MG rates. . . . .	76
3.15	Error for some threshold RRs with low energy thresholds at 900K. . . . .	77

# Chapter 1

## Overview

Large-scale Monte Carlo (MC) neutronic simulations can facilitate the design of advanced nuclear reactors as well as the performance prediction of current operating reactors. This is convenient given that MC methods can solve the Boltzmann transport equation without discretizing it which leads to more accurate results. The drawback, however, is that the stochastic noise causes uncertainties and potential biases that can only be mitigated by a large number of particle histories. Therefore, great amounts of high-performance computing (HPC) resources are required to achieve acceptable results on highly detailed nuclear reactor geometries. The prospects of CPU-based HPC is moving towards microprocessor designs that favor thread-level and data-level parallelism. Such a paradigm shift in HPC architecture would negatively impact performance in MC codes due to the unpredictable memory access of the random-walk, the high-memory requirements and the spatial resolution needed for convergence. These issues are further exacerbated when performing realistic depletion simulations, which require computing seven reaction rate (RR) tallies for hundreds of nuclides (e.g., up to 423 if using the ENDF library) at each step. The focus of this thesis is thus to analyze the performance of OpenMC [18] –a MC neutron transport simulation application– to implement optimizations that circumvent the current bottlenecks that are plaguing the performance of reaction rate tallies.

Section 1.1 describes the concept of tallies within the context of MC simulations. While the discussion will be about tallies in general, more emphasis will be placed

on depletion reaction rate tallies since they are the focus of this work. Section 1.2 provides a general background of the memory hierarchy of modern CPUs. Section 1.3 then goes over previous efforts to optimize OpenMC for large-scale machines as well as single nodes. Section 1.4 describes the strategies/approaches that were taken in this work to improve the performance in the tally module. One approach will essentially focus on implementing optimizations from a computational science perspective while the second path introduces hybrid tallies as an alternative method to generate reaction rate tallies. This section will, therefore, take care of laying out the reasoning and motivation behind each approach.

Chapter 2 discusses the profiling analysis for OpenMC and diagnoses the main bottlenecks within the tally and cross-section modules. This chapter also covers the introduction of the five benchmark geometries that were utilized to gauge performance as well as an overview of the two nodes that were frequently used: Intel Xeon Platinum 8180 and Intel Xeon Phi. OpenMC was profiled over multiple hardware configurations (i.e MPI/OMP), including those that were more specific to the Xeons Phi's (e.g., memory modes like Flat and Cache). While this thesis doesn't include any work on event-based algorithms or vectorization for OpenMC to leverage the Xeon Phi's Vector Processing Units, some effort was put into understanding which of the many topological and memory configurations of the Phi are better for performance. This chapter also discusses several computational-science based optimizations to tackle the modules/sub-routines that are degrading performance. In general, these optimizations revolve around improving the search for the nuclides/reactions that will be tallied and on minimizing data movement by caching relevant data.

Chapter 3 focuses solely on hybrid tallies. Implementation details for OpenMC and other nuances will be elaborated upon in a methodology sub-section. The chapter will also include a discussion of several error metrics that were used to tailor the energy structure, especially in the high-energy range, to arrive at an optimal structure that works well for all the nuclides that were analyzed. Speedups and results analysis of the hybrid tallies are also presented.

Chapter 4 concludes the thesis with an overview of future work.

## 1.1 Tallies in Monte Carlo Neutron transport simulations

Running MC simulations after just specifying the geometries, materials and settings will, most likely, only provide the multiplication factor ( $k_{\text{eff}}$ ). However, if the user is interested in measuring other quantities like RRs, the particle flux, the current, energy deposition, then they need to do so by specifying a “tally” file with specific filters and scores. The formal definition of a volume-integrated tally can be found in equation 1.1 (Adapted from Romano [21] and Haeck [11]). The integration is over the entire phase space, where  $E$  is the energy,  $\Omega$  is the direction and  $r$  the position within a specific volume.

$$T = \int d\mathbf{r} \int d\Omega \int dE \underbrace{h(\mathbf{r}, \Omega, E)\psi(\mathbf{r}, \Omega, E)}_{\text{scores}}, \quad (1.1)$$

$h(\mathbf{r}, \Omega, E)$  is the scoring (or response) function and  $\psi(\mathbf{r}, \Omega, E)$  is the angular flux. Examples of filters include the energy range before or after a particle collision, material regions, universe, cell, among others. The filters limit the events that can score to a tally. Going back to equation 1.1, the filter here are represented by the limits of integrations. The scoring function will be specific to the tally. For example, in case the user wants a RR tally, the scoring function will be the macroscopic cross-section for that particular RR. For a flux tally, the scoring function will be just 1. Inside OpenMC, the integrand of equation 1.1 is represented as “scores” and these are basically the scalar values that are accumulated as sums throughout the simulation (see equations 1.2). Here  $n$  stands for an arbitrary tally.

$$S_{n,\text{sums}} = S_{n,\text{sums}} + S_{n,\text{sums,event}}, \quad (1.2)$$

$$S_{n,\text{squares}} = S_{n,\text{squares}} + S_{n,\text{squares,event}}^2, \quad (1.3)$$

After processing all particles,  $S_{n,\text{sums}}$  will be used to compute the tally (i.e., the expected value) by applying equation 1.4. Here  $N$  is the total number of events.

Simultaneously, sums of squares are also accumulated to estimate the tally's uncertainty (see equations 1.3 and 1.5). Going back to the RR tally example, the value for  $S_{n,sums,event}$  that will be accumulated inside the MC simulation for this given tally is simply the product of the cross-section and the weighted flux estimate. The cross-section will be interpolated at the energy of the event. In the case of a flux tally, the  $S_{n,sums,event}$  is simply the weighted flux estimate. OpenMC computes  $S_{n,sums,event}$  and  $S_{n,square,event}$  through the use of tally estimators because they help quantify the events of interest. Either one of three estimators are used to accumulate scores: (1) analog, (2) collision and (3) tracklength. Each estimator works differently and not all filters apply to all three. The analog estimator, the simplest one, counts the number of times the desired events occur and uses that as an estimate (see equation 1.6). Here  $\phi$  is the flux,  $n$  stands for the specific tally,  $V$  is the volume of the cell,  $W$  is the initial weight,  $I$  is the total number of events and  $w_i$  is the particle weight at event  $i$ . The collision estimator only computes scores after the desired collision occurs and will use the total macroscopic cross-sections in the computation (see equation 1.7). The tracklength estimator computes the score after each particle flight (see equation 1.8). Here  $d_i$  is the pathlength for that given event. In this work, only the tracklength estimator was used because, in general, it has the lowest uncertainty.

$$\bar{T}_{MC} = \frac{S_{n,sums,event}}{N}, \quad (1.4)$$

$$\sigma_{\bar{T}_{MC}} = \sqrt{\frac{1}{N-1} \left( \frac{S_{n,squares,event}}{N} - \bar{T}_{MC}^2 \right)} \quad (1.5)$$

Since tallies are representations of the estimates of the mean of a given random variable, the law of large numbers (LLN) and central limit theorem (CLT) can be used to make the following general statements about them. From the LLN, the sample mean (i.e., tally) will approximate the true mean with increasing number of realizations of the random variable. As stated by the CLT, a high number of realizations also means the distribution of the tally random variable will behave like a Normal (or Gaussian) distribution. Moreover, by ensuring that there is a high number of



particles, the uncertainty of the mean will be guaranteed to tend to zero [19].

$$\phi_n = \frac{S_{n,sums}}{N} = \frac{1}{VW} \sum_i^I w_i \quad (1.6)$$

$$\phi_n = \frac{S_{n,sums}}{N} = \frac{1}{VW} \sum_i^I \frac{w_i}{\Sigma_t} \quad (1.7)$$

$$\phi_n = \frac{S_{n,sums}}{N} = \frac{1}{VW} \sum_i^I w_i d_i \quad (1.8)$$

In this work, however, many of the MC simulations had just 5–10 millions particles, which could be considered somewhat of a low number for standard MC. Nonetheless, this is justifiable for profiling runs since the desire is to diagnose bottlenecks and not necessarily to obtain an accurate solution; they only need to be realistic enough to make the profiling analysis worthwhile.

In order to close out this section, let's give a more concrete example using a depletion RR tally since they will be the basis of this work. In order to do transmutation analysis, usually seven tallies are needed:  $(n, fission)$ ,  $(n, 2n)$ ,  $(n, 3n)$ ,  $(n, 4n)$ ,  $(n, \alpha)$  and  $(n, \gamma)$ . As mentioned previously, the cross-sections for each of those rates are the scoring functions. The only filter that has to be specified here is a material filter to ensure the RRs are computed in the fuel region (or any region that we wish to deplete). The material filter will restrict which particle events score to the tallies. Moreover, there will also be up to 423 nuclide bins to account for the build-up of actinides as the core depletes. Putting it all together, at every valid event and for each of the user-defined RR tallies, there will be up to 423 bins that will be incremented by an estimate  $S_{n,sum,event}$  of the RR definition in 1.1. In order to avoid race conditions, OpenMC uses OpenMP directives to update the tally bins atomically (i.e., one thread at a time). Here it should be noted that tallies are only computed for the *active* cycles of a MC simulation. The MC code first needs to make sure the fission source is converged before it starts collecting tally data on the active cycles. The fission source convergence is handled in the *inactive* cycles of the simulation [21]. As a

result, the performance of the inactive cycles will typically be better than that of the active cycles. During a depletion simulation, the RRs that are computed throughout the active cycles will be passed to a depletion solver at the end of every depletion step. At this point, the depletion solver uses the RRs to solve the transition matrix of the Bateman equations, which will result in an updated nuclide inventory. This is then passed to the MC solver and another transport simulation starts to compute new RRs. The cycle will continue until the final burn-up is reached. From this chain of operations it becomes clear that improving the tally performance on the MC side will ultimately have a positive impact on the overall performance of the depletion simulation.

## 1.2 Brief Overview of Cache Hierarchy

During the MC transport simulation, neutron particles are followed around the geometry until they are leaked or absorbed. Cross-section data, which are obviously material dependent, need to be requested at each event to determine the distance to next collision and also to assess which specific reaction will take place in the case there was a collision. If expensive depletion RR tallies are specified then, at every event in the fuel, OpenMC will need to lookup additional cross-sections (e.g.,  $(n, 2n)$ ,  $(n, p)$ , among others) because they are not computed by default like  $(n, fission)$ . All these steps require data from different materials and nuclides and the stochasticity inherent in MC codes makes it difficult to predict which memory blocks will be needed at any given time [21]. This section will provide a brief overview of the cache hierarchy in modern architectures in order to better explain how the random-walk logic of MC codes impacts performance.

During a job execution, data is allocated inside main memory, which is also referred to as the dynamic random access memory (DRAM). The total size can be on the order of gigabytes (GB). While DRAM has high capacity, servicing a memory transaction here is expensive and leads to a performance loss due to high latency. To get around this, a memory hierarchy is established between main memory and the

CPU by using multiple levels of cache: L1, L2 and L3. A cache is a memory register that holds recently accessed data. Caches are smaller in size than DRAM but data takes a lot less clock cycles to reach the processor [12]. Table 1.1 summarizes the capacity and clock cycle of a standard memory hierarchy.

Table 1.1: Memory hierarchy capacity and clock cycles to service memory request

Level (-)	Capacity (KB)	Clock Cycles (-)
L1	32	3
L2	256–1024	9
L3	8040–39420	40
DRAM	$O(\text{GB})$	$\geq 100$

Let’s say the CPU needs some specific data and starts a load instruction, first it will look for a memory block in the L1 cache, and in case it is not there it will move up on the hierarchy and look for it in the L2 cache. It will continue to do this until it finds the data. If the data was in the L1, then that would be considered an L1 hit. Otherwise it is a L1 cache miss. Same thing with the L2 and L3. There are many more hardware details regarding memory hierarchy that will not be talked about here for the sake of brevity. However, what is important to understand is that there are ways to improve the cache performance, including laying out the data in contiguous 1D arrays when possible as well as exposing spatial and temporal locality. Functions, subroutines and operations that rely on the same data should be called as close to each other as possible. This approach will improve the cache performance by making better use of the nearby data. Otherwise, the data may be evicted (kicked out) from a low-level cache to higher level or even to main memory, which will incur a latency penalty when it is requested again. In-lining functions (calling them explicitly) can also be beneficial to performance [2]. Later in Chapter 2, these concepts will be revisited since they form the basis of the optimizations that were implemented following the profiling analysis.

### 1.3 Previous optimization efforts for OpenMC

Previous research in MC methods for highly-detailed simulations has focused mainly on implementing different parallel computing schemes to take advantage of distributed-systems. Walsh [26], for example, analyzed the cross-section data requirement issues by introducing a nuclear data server algorithm, which proposed to separate a subset of computer nodes to work as servers and store nuclear data that was not accessed regularly throughout the MC simulation. All tests were ran on the Titan Cray XK7 supercomputer. His study first consisted on profiling OpenMC to understand which data was accessed more frequently. He found that exotic inelastic scattering reactions along with secondary angular and energy distributions were among the least accessed data regardless of whether the temperature was at BOC, HZP and HFP. It is important to keep in mind that at different temperatures and burn-up some data blocks will be more relevant than others. For example, in a MC code,  $^{239}\text{Pu}$  data blocks will be accessed more frequently than  $^{236}\text{U}$  at EOC HFP conditions. Therefore, to determine which data blocks needed to be stored off-node, the data blocks for all nuclides were sorted in decreasing value based on an empirical equation that takes into account their memory size and access frequency. After testing multiple scenarios, the conclusion was that significant on-node memory reductions are possible with the server model at the expense of a modest overhead. In some cases, 50 % of the on-node cross-section memory was reduced with only 1 % overhead.

This approach was actually inspired by Romano's [21] tally-server algorithm, which sought to address the high tally memory-footprint by storing tally data on dedicated server nodes  $s$ . The compute processes  $c$  will then focus only on storing the geometry, cross-sections, and the material information together with the meta-data that describes the tally and on handling the transport logic. The total length of each tally array is the product of the number of filters and scoring functions. Therefore, the individual scoring arrays were combined (concatenated) into a sole global array and each server had access to an equal number of indices (or bins). A look-up table was then constructed to map the indices of the new global array to the individual

arrays. At each event, the compute nodes would have to compute the scores and determine to what server nodes to send it. The authors also presented the optimization idea of buffering scores to minimize the communication between the compute process and the servers. The tally server algorithm was implemented in OpenMC and tested on a light water reactor (LWR) depletion problem. Tests were run on either the Blue Gene/P or the Titan Cray XK.

The tallies that were used for this study had two filters, six scoring functions and the number of nuclides varied from 5 to 320. Taking all of this into account, the authors estimated that at each event the data that needed to be transferred to the nodes ranged between 240 bytes (B) and 15.36 kilobytes (KB). Results were obtained for varying ratios of compute nodes to server nodes (i.e.,  $c/s = 1, 3, 7, 15$ ) and architectural traits (i.e., bandwidth, latency). In general, the overhead was found to be minimal [21, 17].

One important downside of this method, however, is that it complicates the implementation of a threaded version of OpenMC through OpenMP. In this case, each thread would need to communicate scores to the corresponding servers, which is not trivial. Even more so, the performance of the tally server algorithm is dependent on the  $c/s$  ratio as well as the machine's bandwidth.

To get around these hurdles, Dun [7] proposed another data decomposition scheme for the tally system that facilitated the implementation of on-node shared parallelism while simultaneously alleviating the excessive tally data footprint. His approach was based on global view arrays (GVA) to partition tallies into small blocks that fit on the computer nodes but are still globally addressable to other processes. GVA facilitates large-scale simulations by providing a programming model that enables global address spaces. Parallel tasks are thus able to communicate with each other through this logically/physically shared memory space. The GVA implementation is built on top of MPI-3 which enables one-side communication. This is useful to send asynchronous messages and to reduce the communication matching cost of send/receive messages.

Adapting GVA to an OpenMC tally accumulation scheme required modifying less than 2% of the total source code (note: this was done for OpenMC 0.5.4). The GVA

is initiated as a 1D array before the transport algorithm begins. Its total length is the product of the number of filters and scores and it is visible to all processors. Just like in the tally server model, there is a mapping between the original index of the tallies and their index in the 1D global array. For the sake of an example, let's say the user defined 4 tallies and that there are exactly 4 processes in the simulation. Each individual processor would then be assigned a tally to keep track of, meaning it will be responsible for adding the scores to the corresponding tally bins in the GVA. This way, the tally data is essentially divided evenly across all processors. The important detail to note is that, even though each processor in this example is responsible for updating particular/discrete tally bins in the GVA, they are *all* still computing and buffering scores for the remaining 3 tallies. In other words, every processor will buffer scores for each of the 4 tallies and once they are full the scores will be sent to the remaining processors (using one-sided communication); thus exploiting the on-node shared memory. Unlike the tally servers, this algorithm is also scalable with memory because it uses exactly the amount of memory it needs to deal with tally data. For example, if a simulation requires that 1 terabyte (TB) of tally data be decomposed, then the GVA implementation will only require at least 1000 nodes with 1 GB of memory each. In the case of the tally server, things get complicated because it is dependent on the  $c/s$  ratio. This variable needs to be either 3 or 5, otherwise the tally server could face a communication bottleneck. The memory overhead of the tally server can be describe with  $m \times s$  where  $m$  stands for the memory of the server and  $s$  for the total number of servers. Since the tally server algorithm adheres to the restriction of  $c/s$ , the total memory cost will grow with the number of total processes  $p$ :  $s \propto p$ . Overall, the authors reported that the GVA implementation outperformed all previous tally decomposition algorithms found in the literature at the time. To be more specific, after running a simulation on the Cray XC30 with a tally size of 2.4 TB and a total of 16,383 processes on 1,366 nodes, they measured an efficiency of 85 % versus a case with 1,024 nodes [7].

It is important to note that the GVA implementation in OpenMC was implemented in 2014 and was never pushed to the main repository. Based on the afore-

mentioned positive results, it might be worthwhile to revisit this idea and adapt it to the latest OpenMC release.

While OpenMC has already demonstrated excellent performance on distributed systems, there is still room for improvement for on-node performance. The prospects of high performance computing (HPC) are moving towards more data and instruction level parallelism, which forces application developers to design new algorithms to expose it in order to see a performance boost. Leveraging multi- and many- core machines is not trivial because of the subtleties in the memory hierarchy, which unfortunately cannot be tuned.

Tramm [23, 24, 25] provided evidence that MC codes are subject to performance degradation for on-node simulations due to the inefficient use of the shared memory. However, these original studies had some limitations. The first thing to note is that it analyzed a proxy mini-app called XSbench, which didn't include the logic to handle tallies as it was mostly focused on analyzing the cross-section data movement. As was mentioned previously, tallies account for a lot of the memory space and for the majority of the activity in the active cycles.

To relax these limitations, Romano [20] analyzed OpenMC's performance over multiple simulations of the standard reactor physics benchmark Hoogetboom-Martin (HM). In total, four simulations were tested: (1) using the general settings of the benchmark, (2) including four RR tallies for each nuclide over a  $289 \times 289$  mesh, (3) 423 nuclide bins were added to the standard case and (4) same as the previous one but with the tallies of the second test. All cases were run on two separate nodes, each with two CPUs. The first one had 10 cores of Intel Xeon E5-2680 v2 (Intel Ivy Bridge-EP microarchitecture). All cores had independent L1 and L2 caches together with a shared L3 cache. The second node, on the other hand, had 12 cores of Intel Xeon E5-2680 v3 (Haswell-EP micro-architecture). These cores have the same memory hierarchy as the Intel Xeon E5-2680 v2. The authors experimented with multiple hardware configurations (i.e., number of MPI procs/ OpenMP threads).

For the Intel Ivy Bridge-EP machine, the authors found that the configuration with 20 threads and just 1 MPI process (1/20) had the worst performance. This

actually ties in directly with the topology of the node. Given that each node has two CPUs, they are both configured as individual non-uniform memory access (NUMA) nodes with their private memory bank and shared L3 cache. MPI processes are assigned to one of the NUMA nodes, ensuring that they only access the memory space for that particular NUMA node, thus minimizing latency. However, if only one MPI processor is specified then the threads will not bind to any specific NUMA node and will request data from each half regardless of how physically far the memory bank is. As expected, the performance will degrade as a result of the increase in latency.

The other extreme scenario uses a configuration of 20 MPI processes / 1 thread (20/1) and this also erodes performance as a consequence of replicating data across all processors. The two best configurations for the Ivy Bridge were 2/10 and 4/5. The authors concluded that having the MPI processes equally divided across the NUMA nodes helped reduced the latency. The L3 cache hits increased since the data loaded by any given thread was leveraged by the rest before it got removed from the last-level cache.

The authors also developed an equation to quantify the effective latency  $\bar{\alpha}$ :

$$\bar{\alpha} = N_{L2}((1 - M_{L2})\alpha_{L2} + M_{L2}((1 - M_{L3})\alpha_{L3} + M_{L3}\alpha_{main})) \quad (1.9)$$

Table 1.2 defines each variable in equation 1.9. The inverse of the effective latency  $\bar{\alpha}$  gives the expected speedup. This is under the assumption that  $\bar{\alpha}$  scales as the inverse of performance. This value was then compared to the actual speedup one would measure by changing the MPI/OMP configuration. In total there were 5 configurations: 20/1, 10/2, 4/5, 2/10 and 1/20. The actual speedup was computed against the latency (per 1000 cycles) of the 20/1 configuration. All of them, except for 1/20, showed a speedup when compared to the latency of 20/1 for the reasons stated above regarding the assignments of MPI processes and the benefits of adding sufficient threads. Overall, the actual speedup and the effective speedup were in good agreement. This illustrates that OpenMC is primarily latency-bound because  $\bar{\alpha}$  was derived with the loose assumption that performance was only dominated by latency



(see equation 1.9).

Table 1.2: Definition of variables for equation 1.9

Variable (-)	Definition (-)
$n$	memory hierarchy level: $L2$ , $L3$ , main
$N_n$	total memory access per particle at level $n$
$M_n$	Miss rate at level $n$
$\bar{\alpha}_n$	latency at level $n$

## 1.4 Performance Optimization based on Profiling Analysis and Hybrid Tallies

Monte Carlo (MC) methods have long commanded serious attention in the nuclear community for their accurate representation of the neutron transport phenomena. In recent years, this quality coupled with their ability to handle arbitrary 3D geometries, has increased the appeal of coupling them with burn-up applications. Already an integral component in nuclear engineering, burn-up applications are the basis of proper fuel management calculations, which in turn are essential for estimating parameters such as fuel composition, cycle length, and power density variations.

The link between MC methods and burn-up applications are the incident neutron reactions that result in the transmutation of the target nuclide (i.e.,  $(n, fission)$ ,  $(n, 2n)$ ,  $(n, 3n)$ ,  $(n, 4n)$ ,  $(n, p)$ ,  $(n, \alpha)$  and  $(n, \gamma)$ ). Since depletion RR tallies are calculated for all nuclides at each particle event that takes place in the fuel region, computing them over realistic LWR models can become taxing because of the high amount of memory required and operations that need to be handled. The total tally memory footprint in a representative LWR model has been estimated to be on the order of TB, and this more than exceeds the available memory of typical modern computer nodes [11].

In this work, two approaches were taken to speedup the tally system in OpenMC for depletion simulations. The first approach is rooted in more traditional compu-

tational science. A thorough profiling analysis of OpenMC will help spot bottleneck modules and subroutines. Based on this analysis, several optimizations will be implemented to address the hotspots. This approach will be covered in more detail in Chapter 2. The other approach tackles the problem of speeding up the active cycles from a physical angle. The general idea and background behind it will be discussed in the remainder of this section. Results will be presented in Chapter 3.

During the computation of depletion tallies, the vast majority of time is spent interpolating the cross-section data. An alternative option is to unionize the grid over all nuclides and temperatures, but this is also prohibitively costly in terms of memory. Haeck [11] proposed the multibinning (MB) method to replace the expensive RR tallies with a simpler flux tally. The flux tally and a precomputed multigroup (MG) library cross-section are then folded together to compute the rates. His implementation was done for MCNPX 2.5.0, and speedups of up to 42 for a single pin-cell with MOX fuel were reported. Accuracy was verified by making sure the relative difference of the MB method was within one standard deviation of the reference’s relative uncertainty. It took approximately 43,000 groups to ensure that all rates met the accuracy criterion. The main reason was the  $(n, \gamma)$  rate in  $^{238}\text{U}$ , whose accuracy would not have reached acceptable levels otherwise [11]. We now turn to Equations 1.10 and 1.11 to compare the traditional RR tallies, the MB method, and the other approaches from previous studies by estimating their tally memory penalty:

$$M_n = [L_{n,RR} + L_{n,F}] \times B_{double} \quad (1.10)$$

$$L_{n,RR} = N \times R, \quad (1.11)$$

where  $N$  and  $R$  are the number of nuclides and rates, respectively, to be tallied;  $M_n$  is the estimated memory penalty per depletion region for method  $n$ ;  $L_{n,RR}$  is the length of the RR tally array;  $L_{n,F}$  is the length of the flux tally array and  $B_{double}$  is a constant of 8 bytes.

Results for Equations 1.10 and 1.11 are summarized in Table 1.3. Even in the upper bound case where all seven reaction rates are computed for 422 nuclides (rep-

representative of a long depletion chain),  $M_1$  is still 13 times higher than the baseline  $M_0$ , which highlights the major drawback of the original MB method. Furthermore, Fridman [8] later reported that the MB method, as originally proposed, did not include a background cross-section that accounted for the unresolved-resonance range (URR). Their study updated the MB method by adding a background cross-section, estimated by Segev [22], and simplified further on the assumptions that both the Dancoff and Bell factors were exactly one. The accuracy of the MB method was significantly improved for thermal and fast spectrums. For several important isotopes, such as  $^{238}\text{U}$  and  $^{232}\text{Th}$ , the authors measured reductions in the relative error by a factor of 3 to 10.

In another approach presented by the TRIPOLI development team, the  $^{238}\text{U}(n, \gamma)$  rate is computed with an RR tally while the remaining rates are computed from the collapse of an 11,500-group flux [3, 6]. While this approach does avoid adding a background cross-section for properly capturing the  $^{238}\text{U}$  unresolved range, it requires  $\sim 4$  times more memory than the baseline (see row 3 in Table 1.3). In this study, we adopt a hybrid approach that has similar memory requirements to the baseline method; namely, the  $(n, \gamma)$  and  $(n, \text{fission})$  rates are directly tallied for all nuclides and a flux tally with 2,500 (or fewer) groups is used to collapse the remaining reaction rates (herein referred to as “MG rates”). Doing a direct tally for capture and fission enables us to significantly reduce the total number of groups because these two cross-sections require the most groups (unlike the majority of the remaining relevant reactions for depletion). As can be noted from row 4 in Table 1.3, with 2,500 groups, the total memory requirement for this hybrid approach is comparable to the baseline, indicating that this method does not further exacerbate memory needs. Assuming that an acceptable level of accuracy can be obtained, we need to verify whether there is any performance degradation that would make the hybrid method not worthwhile. Furthermore, because one of the goals is to minimize the number of groups to the extent feasible, this thesis also presents an analysis of the accuracy of the method as a function of the number of groups to ensure the smaller energy group structures are still accurate. It is important to emphasize that this particular hybrid tally method

implementation will only be tested on LWR systems. Fast reactors might require a modified version of this approach in order to address their physics and particularities. As a result, this analysis was performed using two LWR benchmarks with the aim of getting a full understanding of the viability of hybrid tallies for depletion simulations.

Table 1.3: Estimated Memory Penalty Per Depletion Region of Tallies in Previous Studies

$n$	$N$	$R$	$L_{n,F}$	$L_{n,RR}$	$M_n$ [kB]
0. baseline	422	7	0	2954	23.6
1. Haeck [11]	0	0	43,000	0	344.0
2. Brun [3]	1	1	11,500	1	92.0
3. This work	422	2	$\leq 2500$	824	$\leq 26.8$

# Chapter 2

## Profiling Analysis

This chapter covers a series of profiling efforts to diagnose major bottlenecks in OpenMC. The profiling was done over five benchmarks: (1) pin-cell, (2) assembly, (3)  $2 \times 2$  fuel assemblies with periodic boundary conditions, (4)  $2 \times 2$  fuel assemblies with water-reflector and (5) a small modular reactor (SMR) (see Figure 2-1). These test cases provide multiple options to benchmark performance and to assess how OpenMC scales with increasing geometrical complexity.

A brief overview of the Intel Xeon Phi 7210 (Knights Landing, KNL) and Intel Xeon 8180 (Skylake) will also be provided since it will facilitate the discussion of the profiling analysis later on.

As far as the profiling goes, two approaches/strategies were taken to guarantee a thorough analysis. The first approach used solely the particles processing throughput as a metric. This is nothing more than the rate at which particles were processed. It has units of neutrons per second and it is printed by default as a standard output once an OpenMC simulation has finished. This metric makes it easier to evaluate whether or not there was a speedup: if the baseline throughput is lower than the throughput of the modified version then that signals there was a speedup. Therefore, now it is straightforward to test and compare multiple settings (e.g., Window multipole [14] vs. Pointwise, fresh fuel vs. depleted fuel) as well as hardware configurations (e.g., optimal configuration of MPI processes / OpenMP threads or the Phi's memory and clustering modes). The second strategy relies on using Intel VTune, a performance

profiler, that highlights the modules, subroutines and source-lines that are causing performance drops. Overall, it is a powerful tool to understand what sections of the code are having a detrimental impact of performance and to analyze the cache hierarchy. The downside is that it takes more time to get results, and the output files have a considerable memory size (e.g., on the order of GB for the SMR).

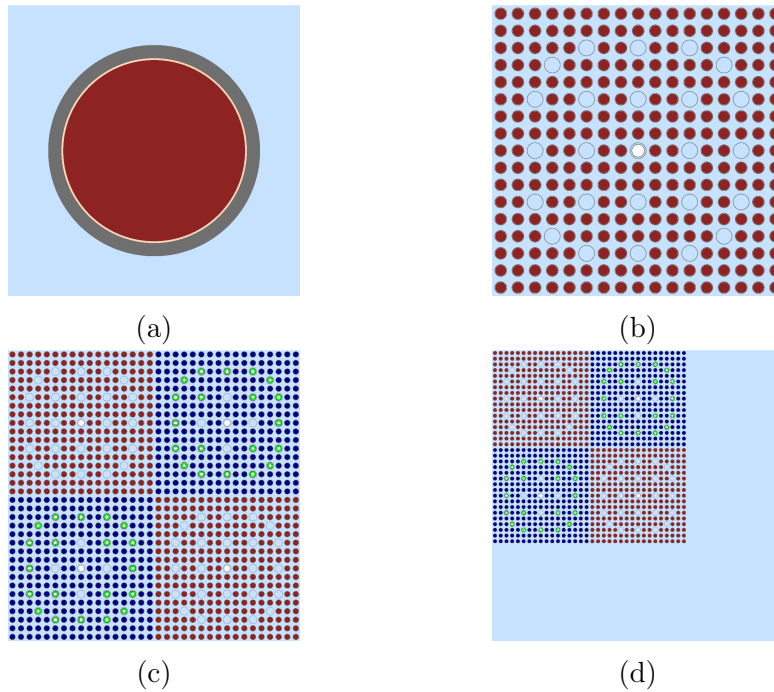


Figure 2-1: A series of test problems with increasing geometric complexity were used to measure OpenMC's performance on Intel nodes: a fuel pin (a), fuel assembly (b),  $2 \times 2$  fuel assembly, with PB conditions (c) and a  $2 \times 2$  fuel assembly with water reflector (d).

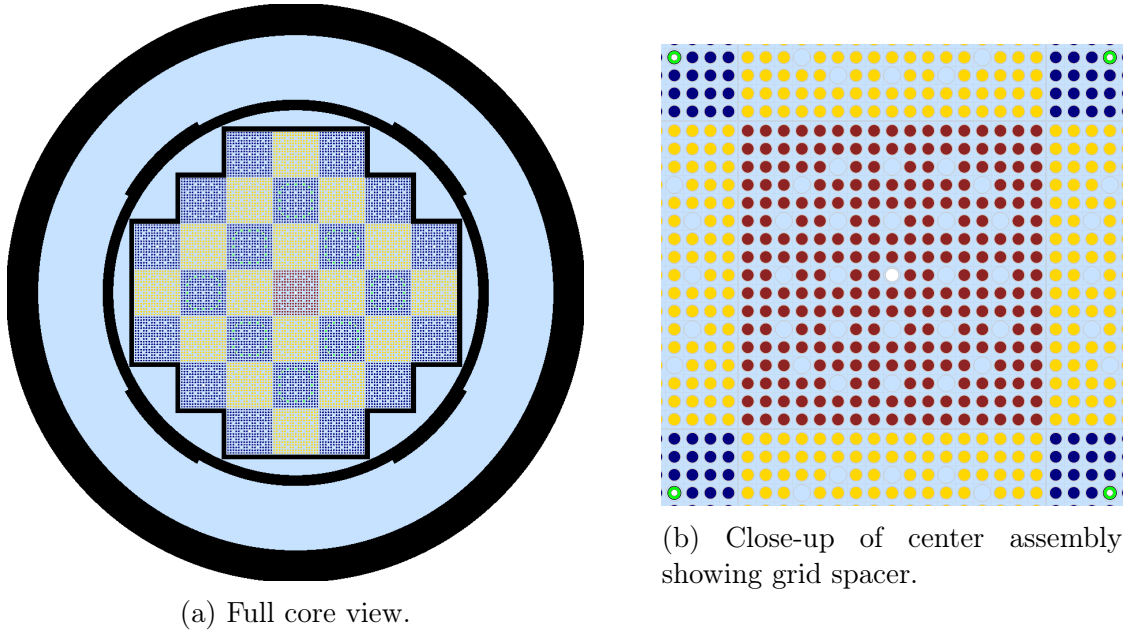


Figure 2-2: SMR model.

## 2.1 Profiling Test Cases

As mentioned previously, five benchmarks were considered to analyze the performance of the code, all of which were based on the LWR full-core Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS). This is a very detailed benchmark that includes three enrichments for  $\text{UO}_2$  fuel assemblies (e.g., 1.6%, 2.4% and 3.1%), borated water, zircaloy, helium, air, borosilicate glass and stainless steel. All test cases, except for the SMR, are in 2D.

The first benchmark was a standard fuel pin with only 1.6% enriched fuel, a helium gap, zircaloy cladding and borated water moderator (see Figure 2-1a). The second one was a  $17 \times 17$  assembly with control guide tubes and a central instrument tube (see Figure 2-1b). The third benchmark consisted of 4 assemblies arranged in a  $2 \times 2$  configuration. Two of the assemblies were identical to the second benchmark while the last two varied only in that they had 3.1 % fuel enrichment instead of 1.6 % (see Figures 2-1c).

The fourth benchmark expands on the third one and surrounds the fuel assemblies with water, essentially turning it into a  $3 \times 3$  configuration with reflective boundary

conditions (see Figure 2-1d). The last benchmark is the Small Modular Reactor (SMR) and it is loosely based on the NuScale model. In total, there are 37 assemblies and each one includes 264 fuel pins, 1 instrument tube and five spacer grids. The rod pitch is 0.496 and the active fuel length was settled at 200 cm (see Figure 2-2).

## 2.2 The Phi and Skylake

While multiple nodes, each with different architectures, were used throughout this research, only the Phi and Skylake were featured consistently. These are two of Intel’s most modern micro-architectures. It is very likely that exascale machines will keep similar hardware traits and, therefore, doing profiling and optimizations with them as testbeds will help prepare OpenMC for future designs. Table 2.1 summarizes the general hardware traits for both Skylake and the Phi.

All simulations were ran on Argonne National Lab (ANL)’s experimental cluster called the Joint Laboratory for System Evaluation (JLSE).

Table 2.1: Hardware details for frequently used nodes

Node (-)	CPU (-)	cores (-)	frequency (GHz)	threads (-)	L1/L2/L3 Size KB/MB/MB
Skylake	2× Intel Xeon Platinum 8180	56	2.4	2	32.0/1.0/39.4
Phi	Intel Xeon Phi 7210	64	1.3	4	32.0/1.0/0.0

As can be noted from Table 2.1, both nodes have a high number of total threads, which once again accentuates the importance of exposing thread-level parallelism to exploit performance as much as possible in modern architectures. The cache hierarchy is similar up until the L3 level. While Skylake has a big L3 cache level, the Phi does not have any L3 so a miss at the L2 level means the memory address for the requested data needs to be fetched from main memory. To be fair, the Phi presents multiple hardware innovations on other fronts with the goal of mitigating some of the shortcomings of its memory hierarchy.

The first thing to note about the Phi is that it is organized into *tiles* and each one is made up of 2 cores (each with their private L1 cache), a shared 1MB L2 cache and 2



vector processing units with 512-bit vector instructions through AVX-512 extensions (see Figure 2-3b). A distributed directory-based protocol and tag directory ensure cache-coherency across all tiles. The Phi also introduced *cluster modes* to increase the affinity of threads and memory with the goal of reducing latency. The three primary modes are: all-to-all, quadrant and sub-NUMA cluster (SNC). These modes impact how the chip handles L2 cache misses. In the event that there is a miss at this level, the tile will request the data's memory address from the tag directory, which will first survey all tiles to see if they have it. In case they don't, the request is sent to main memory. This is where the clustering modes come in. At boot-time, if the user configures the Phi to be in the all-to all clustering mode then there will be no affinity between tile, tag directory and main memory. This implies that the memory addresses will be hashed and distributed completely arbitrarily. In other words, the physical locations of the tiles with respect to each other will not be taken into account. In this mode, if a particular tile has a L2 miss then when it tries searching for the memory address through the tag directory it could end up finding the address on a tile that's physically far away just as it could find it on one that's close by. The quadrant clustering mode, on the other hand, divides the chip into four quadrants and introduces an affinity between the tag directory and the nearest memory. This means that a tag directory in any particular quadrant will have the memory addresses for only the memory blocks that are located on the memory that is closest to that given quadrant. From Figure 2-3a, it can be noted that there are multiple accesses to main memory such as Double Data Rate (DDR) and MultiChannel Dynamic Random Access Memory (MCDRAM). In quadrant mode, once a L2 miss occurs, the memory address will be requested from a tag directory regardless of the quadrant. If the directory can't find the address on any tile, then it will forward the request to the closest main memory channel. Quadrant mode is usually the default clustering mode when the Phi is booted with no specifications. The last clustering mode, the SNC- $n$ , creates an even stronger affinity between the tiles, tag directory and main memory such that tiles will only request memory addresses from a tag directory on their quadrant.  $n$  stands for the number of quadrants, which could be either 2 or 4. At

this point, the quadrants behave essentially like cache-coherent NUMA modes. SNC-n is expected to have the lowest latency because data locality is increased. Skylake can also be configured at booth time to do a sub-NUMA division. It should be noted as well that the Phi was designed to favor compute intensive applications (i.e., high number of FLOPs) and that’s one of the reasons it provides multiple memory modes: Flat-MCDRAM, Flat-DDR, Flat, Cache and hybrid mode. MCDRAM is a 16 GB capacity high-bandwidth on-package memory. DDR is the default main memory (see Figure 2-3a). Flat-MCDRAM, Flat-DDR and Flat designate what will be treated as main memory during that particular boot. For example, if the user wants the main memory to be the high-bandwidth MCDRAM then it should specify Flat-MCDRAM. If more than 16 GB of storage is needed from main memory then either Flat-DDR or Flat should be used. In Flat-DDR, the DDR (which has a capacity of  $\sim 400$  GB) will be treated as the main memory. In the Flat mode, both the high capacity DDR and the high bandwidth MCDRAM are treated as main memory. If the user configures the Phi to be in Cache mode, then 100 % of the MCDRAM will act as a 16 GB cache for the 1 MB L2 cache of every tile. The hybrid mode treats a fraction  $f$  of the MCDRAM as a cache to the L2 (just like in Cache mode) and the remaining fraction is configured as main memory in conjunction with the DDR (just like in Flat mode). The variable  $f$  can be tuned by the user to be 25%, 50% or 100% [13].

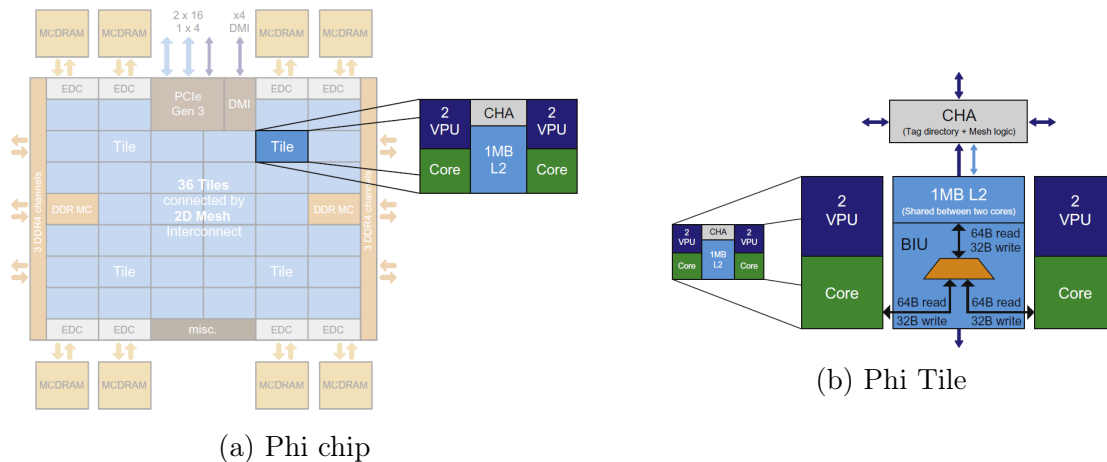


Figure 2-3: Phi schematic representation, taken from [13].

## 2.3 Profiling Analysis

### 2.3.1 First approach

This section includes the results for the particle processing throughput-based profiling. As was mentioned previously, the higher this number, the better the performance. The following tables include results for both the inactive and active cycles. However, the active cycles are always more relevant because this is where tallies are tracked and thus it becomes straightforward to assess what is their impact on performance.

All simulations were done with OpenMC. A total of 20 batches were used, 10 of which were for the inactive cycles. Each batch simulated 250,000 neutrons. The temperature was kept at 300K and both ACE pointwise and multipole data formalism were tested. Fresh and depleted fuel were also compared. The fresh fuel had six nuclides while the depleted fuel had 293. The number densities for the depleted nuclides were obtained by depleting to 1 month using OpenMC's depletion module, which at the time was called OpenDeplete. While 1 month is not enough to have an accurate and physical representation of the build-up of actinides, this really does not affect the performance analysis. In all simulations, the  $(n, fission)$ ,  $(n, 2n)$ ,  $(n, 3n)$ ,  $(n, 4n)$ ,  $(n, p)$ ,  $(n, \alpha)$  and  $(n, \gamma)$  reaction rates were tallied for all the nuclides in the fuel.

#### Tally slowdown and MPI/OMP configuration

A quick way to quantify the tallies' impact on performance is through the tally slowdown metric, which is simply the ratio of the particle processing throughput of the inactive and active cycles. Ideally, one would hope the slowdown to be close to 1, thus indicating that the tallies are not degrading performance. As can be noted from Table 2.2, this is not the case since the tally slowdown deviated from 1 regardless of data formalism or fuel type. For this particular case, the Phi was configured with SNC-4 and Flat memory mode.

The second key point from Table 2.2 is that the tally slowdown increases with increasing number of threads, thus highlighting there is poor scaling with respect to

the number of threads. Such results should not come as a surprise because each tile only has a shared 1 MB L2 cache. Since each thread follows a particle through its lifetime, there is a higher chance of taking a performance penalty when increasing the number of threads due to all the requests to main memory.

As for Table 2.3, this one includes a similar comparison except that now the metric is directly the particle processing throughput. First thing to note is that the performance suffers considerably when going from fresh fuel to depleted fuel. Once again, the data formalism does not have a real impact on performance. Even more so, Table 2.3 also shows that the particle throughput increases with the number of threads. At first glance this might appear to contradict the conclusions from Table 2.2. However, on closer inspection this is suggesting that, while the performance does increase with the number of threads, the overhead of executing tallies still becomes worse. In other words, the particle processing throughput of the inactive cycles scales better with the number of threads than the active cycles.

Table 2.2: OpenMC’s tally slowdown for the SMR benchmark in kiloneutrons per second [kn/s] with fresh/depleted fuel on a Phi node (SNC-4/Flat) with various MPI/OpenMP configurations.

Fuel Type	Threads per core	Pointwise [kn/s]	Multipole [kn/s]
		Tally Slowdown	Tally Slowdown
Fresh	1	4.73	3.96
	2	5.25	4.61
	4	6.55	5.47
Depleted	1	4.50	3.97
	2	4.95	4.60
	4	5.22	5.0

### Clustering modes

Here the goal is to elucidate which Phi clustering mode works best for OpenMC. Important to note that the all-to-all mode was not tested because, from a theoretical perspective, it does not look promising for this application. That leaves only SNC-4 and Quadrant. Table 2.4 shows results for simulations over the  $2 \times 2$  assemblies with reflector test case. The first thing to note is that performance for the 4/64 configura-

Table 2.3: Processing Throughput for the SMR benchmark in kiloneutrons per second [kn/s] with fresh/depleted fuel on a Phi node (SNC-4/Flat) as a function of the number of OpenMP threads.

Fuel Type	Threads per core	Pointwise [kn/sec]		Multipole [kn/sec]	
		Inactive	Active	Inactive	Active
Fresh	1	31.1	6.73	24.8	6.26
	2	48.2	9.19	39.0	8.47
	4	67.5	10.3	53.8	9.85
Depleted	1	6.20	1.38	5.33	1.34
	2	9.70	2.00	9.27	2.02
	4	13.7	2.63	12.8	2.56

tion is always better than 64/4, which is why 4/64 is the predominant configuration in this work when using the Phi. As for the clustering modes, SNC-4 mode always outperformed quadrant mode by a small margin. This is not surprising because SNC-4 establishes an affinity between the tiles, tag directory and main memory such that when there is a L2 cache miss, only the tag directory of the tiles in that particular quadrant will be surveyed to find the memory address. The main memory will be the closest memory channel to the quadrant, which also helps mitigate the latency.

Table 2.4: OpenMC’s processing throughput on the 2×2 assembly with reflector benchmark in kiloneutrons per second (kn/sec) for Phi’s Quadrant and SNC-4 cluster modes (depleted fuel).

NUMA Mode	MPI Procs	OpenMP Threads	ACE [kn/sec]		Multipole [kn/sec]	
			Inactive	Active	Inactive	Active
Quadrant	4	64	13.29	3.17	12.63	3.20
	64	4	6.42	2.26	7.43	2.30
SNC-4	4	64	14.08	3.30	12.83	3.35
	64	4	8.13	2.37	7.72	2.41

## Phi Memory Modes

This section covers the analysis of the memory modes. Three memory modes were tested: Flat-MCDRAM, Flat-DRAM (or Flat-DDR) and Cache. To review, Flat-MCDRAM configures the high-bandwidth MCDRAM as the main memory while Flat-DRAM instead uses DRAM as the main memory. The command `numactl` allows the

user to select one or the other or a combination of both. The cache mode, on the other hand, configures MCDRAM as a cache to the L2 in every tile. This could be beneficial for bandwidth-bound applications. However, the application might take a significant performance hit if there is a miss at the MCDRAM level because the latency to main memory will be higher now. Table 2.5 shows the results for multiple simulations with variations in the fuel type, memory modes and data formalism. There is no appreciable performance difference among the memory modes. It is important to note that the SMR problem does not exceed the 16 GB memory of MCDRAM, which is probably why it was irrelevant what memory mode was used. In future studies, it might be worth it to test the memory modes with bigger problems and explore if there are different trends. Another point to make is that Intel has developed the `Memkind` library to provide a mechanism to select which specific memory blocks in the application should be allocated on MCDRAM versus DRAM. In this work there was no experimentation with the library so this could also be an additional topic for future work.

Table 2.5: OpenMC’s processing throughput across different memory modes for the SMR benchmark with fresh/depleted fuel on a Phi node (SNC-4/Flat) with 4 processes and 64 OpenMP threads/process.

Fuel Type	Memory Mode	Pointwise [kn/sec]		Multipole [kn/sec]	
		Inactive	Active	Inactive	Active
Fresh	Flat-dram	63.7	35.1	53.8	31.8
	Flat-mcdram	63.3	35.4	52.6	31.5
	Cache	63.3	35.0	51.1	32.0
Depleted	Flat-dram	14.2	1.66	12.9	1.50
	Flat-mcdram	13.9	1.63	12.8	1.57
	Cache	15.0	1.50	12.7	1.57

## Phi versus Skylake

This section compares the processing throughput between both nodes and across all test cases. Here, only the fuel type was varied. Table 2.6 shows the results for the Phi simulation, using SNC-4, Flat memory mode and a 4/64 MPI/OMP configuration. Once again, there is also a significant drop in performance when going from fresh to

depleted fuel (ranges from 19-25x in the active cycles).

Table 2.7 presents the results for the Skylake node. The MPI/OMP configuration was 2/56. Similar to the Phi results, there is a big performance drop between the two fuel types (ranges from 13-20x). However, more impressive is the fact that there is a 3–5x difference between the Phi results and Skylake. There could be a few reasons for why this is the case, but arguably the most important one is the difference in memory hierarchy. The Phi does not have a L3 cache so a miss at the L2 level will force a request to main memory. Skylake, on the other hand, has a high capacity L3 cache and it is better suited to handle L2 cache misses, which are common for latency-bound applications like OpenMC.

Table 2.6: Baseline performance for each test problem with fresh and depleted fuel loadings. All simulations were performed on a Phi node (SNC-4/Flat) with 4 mpi processes and 64 OpenMP threads/process. The inactive/active columns show performance in kiloneutrons per second.

Test Problem	Fresh fuel		Depleted fuel	
	Inactive	Active	Inactive	Active
Fuel pin	88.1	66.9	11.1	2.60
Assembly	71.2	56.7	11.6	2.91
2×2-assembly	70.5	53.0	11.4	3.01
2×2-assembly w/ Reflector	74.5	56.8	14.1	3.30
SMR	66.9	34.4	13.9	1.81

Table 2.7: Performance for each test problem with fresh and depleted fuel loadings on a dual-socket Intel Xeon Platinum 8180 node. All simulations were performed with 2 mpi processes and 56 OpenMP threads/process. The inactive/active columns show performance in kiloneutrons per second.

Test Problem	Fresh fuel		Depleted fuel	
	Inactive	Active	Inactive	Active
Fuel pin	322	99.8	50.0	7.9
Assembly	259	192	48.1	9.0
2×2-assembly	255	186	49.3	9.6
2×2-assembly w/ Reflector	267	200	57.2	10.5
SMR	228	151	59.4	12.4

## Concluding Remarks

In order to wrap-up this section let's review a few take-aways and key points from the throughput-based profiling results. The first thing to go over is that using depleted fuel takes a toll on performance regardless of the test case or the computer node. The fresh fuel active cycle performance is 13-25x higher. Realistic depletion simulations will require 250+ nuclides for the majority of the depletion steps so addressing this issue will be fundamental to improve MC performance. As for the clustering modes, between SNC and Quadrant mode, the latter was faster but only slightly so. Both modes configure the Phi chip to establish affinities between the tag directory and main memory. The only major difference between the two is that in the SNC-4 mode the affinity is extended to the tile, which further reduces latency. There was also no appreciable difference between the Phi memory modes. Given that all test cases consumed less than 16 GB, it is possible that none of the nuances of each individual memory mode were exploited because of this. The optimal MPI/OMP configuration for the Phi was 4/64, which essentially translates to a NUMA mode per MPI process. The scaling in this configuration was better than for 4/16 and 4/32. However, it is important to point out that the tally slowdown metric revealed the overhead of doing tallies actually increases with increasing number of thread; in other words, the inactive cycles scale better. Moreover, Skylake was 3-5x faster than the Phi. As was mentioned previously, this is attributed to the memory hierarchy. The Phi's lack of a L3 cache introduces high latency penalties whenever there is a L2 miss, which is quite often in MC depletion simulations. As for the data formalism, there was no significant difference between using pointwise versus multipole. Finally, no Phi-specific optimization were attempted in this research. This would have been inconvenient given that Intel discontinued the Phi on November 2017 [1].

### 2.3.2 Second Approach

The goal of this section is to dive deeper into the performance analysis of OpenMC with more detailed profiling. Intel VTune Amplifier XE was used to collect perfor-



mance data and that way inform the implementation of performance optimizations [5]. VTune is capable of collecting memory access analysis, thread profiling, stack sampling and hardware event sampling. More importantly, VTune has support for the Phi and is able to comment on why certain performance metrics are low.

For this section, all of the profiling was done using the SMR benchmark, 10 total cycles (5 of which were for the active cycles), 250,000 neutrons per batch with pointwise data. The Phi was used for all simulations. Seven RRs were tallied for 293 nuclides in all fuel regions. The MPI/OMP configuration was 4/64 and the boot-time configuration was SNC-4/Flat mode.

## General Exploration

Figure 2-4 shows a summary from a **General Exploration** VTune analysis. The first aspect to highlight is that the cycles-per-instructions (CPI) of the overall application is approximately 4. In general, any CPI above 2 signals that OpenMC is not making efficient use of the CPU [13].

Furthermore, based on the instruction pipeline, VTune classified OpenMC as a back-end bound application, which means it is inefficient in executing instructions and this could be due to frequent stalls. When it comes to the cache performance, OpenMC is bound at the L2 level with a 50 % miss bound. This falls in line with what was previously observed in the literature [20]. Since each Phi tile has two cores sharing a single 1MB L2 cache it is not surprising that this is the case. The random memory access for data relevant to the tally process further exacerbates the situation. Note that in Figure 2-4 there is also information for three other VTune classifications: front-end (fetching and decoding instructions), bad speculation (branch mis-prediction), and retiring (useful execution). However, OpenMC does relatively well on each of them. The vector processing unit performance is the only exception because it is 30 % efficient. This was pretty much expected considering no algorithmic changes have been made to favor Event-based Monte Carlo.

Additionally, a timing profile of OpenMC was performed to determine which modules are consuming the most CPU execution time. Table 2.8 breaks down the most

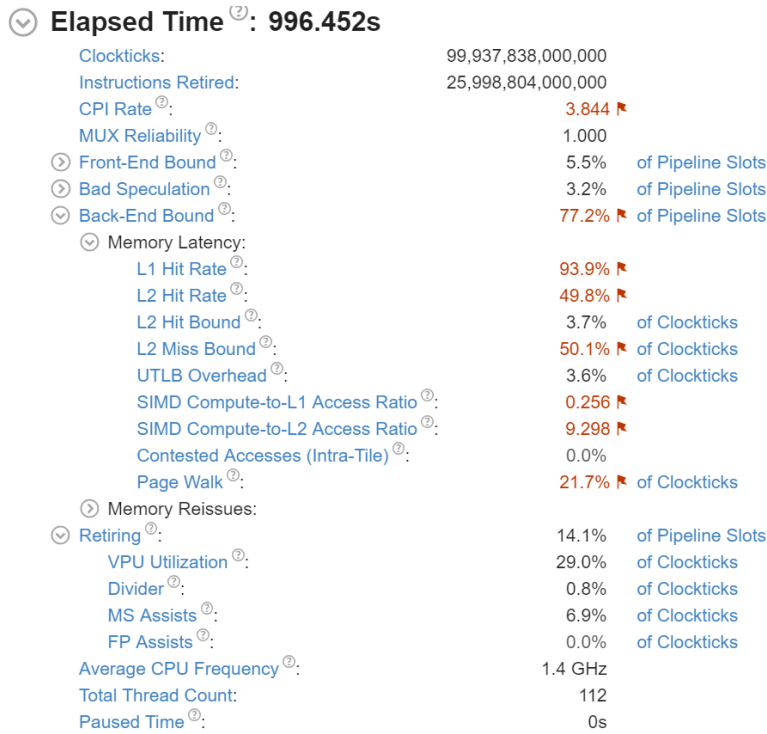


Figure 2-4: Summary of VTune General Exploration analysis of OpenMC on SMR benchmark with depleted fuel type and pointwise cross sections.

expensive OpenMC modules by their fraction of the total CPU execution time. At 38.7 %, the tally module accounts for the biggest chunk of CPU execution time. The cross-section module follows through with 33.4 %. The transport, physics and geometry modules are next. However, the profiler did not highlight them as being inefficient.

Table 2.8: CPU execution time breakdown across most expensive modules.

Module (-)	CPU execution time percentage (%)
Tally	38.72
Cross-section	33.41
Geometry	4.53
Transport and Physics	1.12
Other	22.22

A module-level profiling was also performed to understand which specific sub-routines were responsible for the bottlenecks. Table 2.9 breaks them down by their CPU execution time, instructions retired fraction and CPI. For the tally module, the

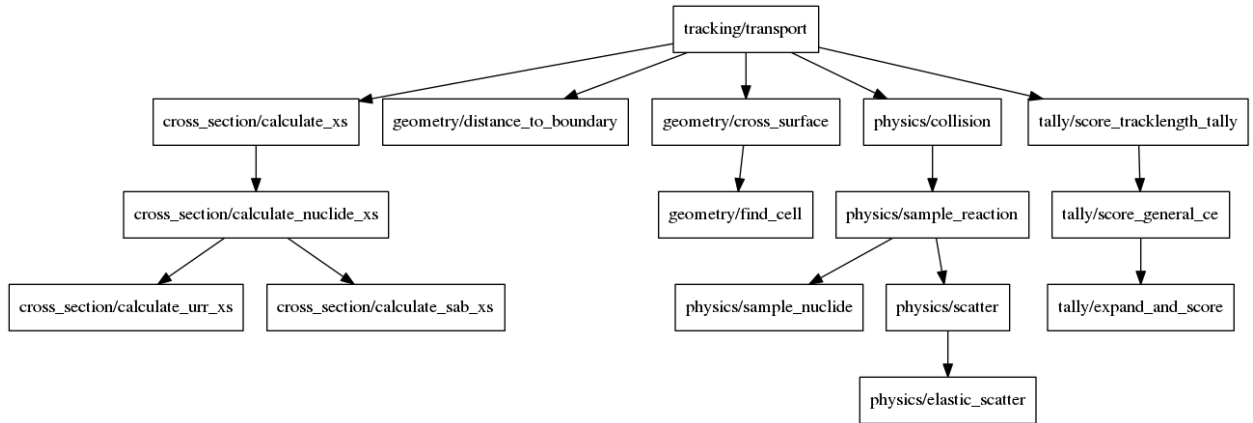


Figure 2-5: Call graph of key functions in OpenMC.

following three subroutines were classified as hotspots: (1) `score-general-ce`, (2) `score-track-length`, and `expand-and-score`. An OpenMC call-graph was generated through the use of another profiler called Valgrind (see Figure 2-5). This is a good way of looking at the module and subroutine dependencies within the code.

As can be noted from the call-graph, during an OpenMC run the first subroutine to be called within the tally module is `score_track_length`, which is where the code determines what nuclide bins need to be updated. This subroutine will then call `score_general_ce` to compute  $S_{n,sums,events}$ . Finally, `expand_and_score` updates the corresponding tally bins (i.e.,  $S_{n,sums}$ ). As for the cross-section module, the subroutine `calculate_cross-section_xs` stands out due to its high CPI of almost 10. Figure 2-6 shows the cache performance of the most problematic subroutines in the code. The majority of them are also on Table 2.8. One exception is `dict_header_MOD_dict_general_ce`. On closer inspection with VTune, this subroutine was found to be called extensively from `score_general_ce` to do a reaction index search for the nuclides. Overall, these results accentuate just how latency-bound OpenMC is at the L2 level since the miss rates and miss bound percentages are high.

Table 2.9: Efficiency of OpenMC key functions. Three metrics: CPU time, instructions retired, and cpi rate are presented for the SMR benchmark (depleted fuel loadings).

Module/Function	SMR-depleted-pointwise		
	CPU time	Instr. retired	cpi rate
tally/score_general_ce	26.2%	18.4%	5.41
tally/score_tracklength_tally	6.87 %	28.1%	0.931
tally/expand_and_score	5.56 %	8.9%	2.36
cross_section/calculate_xs	2.79 %	4.88%	2.15
cross_section/calculate_nuclide_xs	26.7%	10.6%	9.52
cross_section/calculate_urr_xs	3.94%	2.36%	6.34
cross_section/calculate_sab_xs	0.0346%	0.0174%	7.29
geometry/distance_to_boundary	1.47%	2.40%	2.68
geometry/find_cell	0.89%	0.86%	3.93
geometry/cross_surface	0.11%	0.78%	5.46
geometry/simple_cell_contains	0.37%	0.76%	1.83
physics/sample_nuclide	0.10%	0.078%	4.93
physics/sample_reaction	0.049%	0.024%	7.92
physics/collision	0.049%	0.02%	8.98
physics/scatter	0.10%	0.041%	9.38
physics/elastic_scatter	0.147%	0.059%	9.70
tracking/transport	0.417%	0.055%	2.92
random_lcg/prn	0.089%	0.113%	3.06

Function / Call Stack				
	L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound
▶ <code>__cross_section_MOD_calculate_nuclide_xs</code>	80.9%	43.0%	5.0%	88.8%
▶ <code>__tally_MOD_score_general_ce</code>	94.7%	42.5%	2.4%	44.5%
▶ <code>__dict_header_MOD_dict_get_elem_ii</code>	94.3%	44.2%	2.9%	50.1%
▶ <code>__tally_MOD_score_tracklength_tally</code>	99.3%	92.9%	1.9%	2.0%
▶ <code>__tally_MOD_expand_and_score</code>	96.9%	76.5%	3.0%	12.4%
▶ <code>__cross_section_MOD_calculate_urr_xs</code>	86.4%	11.6%	1.3%	100.0%

Figure 2-6: Performance profiling memory-access for the top hotspots subroutines in OpenMC.

### Hotspots for the Tally module

As mentioned previously, the three most expensive subroutines in the tally module handle the logic of scoring tallies. Figure 2-7 illustrates that the source-line code that is highlighted from `score_track_length` is related to the nuclide-in-material search. This is handled as a linear search with only a *for loop*, which is very inefficient. For

score\_general\_ce, Figure 2-8, highlights the code block where OpenMC searches for the reaction index with a dictionary hash table and constructs the cross-sections at tally-time.

Source	Instructions Retired	CPI Rate
! Get index of nuclide in nuclides array		
i_nuclide = t % nuclide_bins(k)	19,136,000,000	1.931
if (i_nuclide > 0) then	33,696,000,000	2.512
if (p % material /= MATERIAL_VOID) then	28,860,000,000	1.908
! Get pointer to current material		
mat => materials(p % material)	14,248,000,000	2.165
! Determine if nuclide is actually in material		
NUCLIDE_MAT_LOOP: do j = 1, mat % n_nuclides	2,075,528,000,000	0.827
! If index of nuclide matches the j-th nuclide listed in the		
! material, break out of the loop		
if (i_nuclide == mat % nuclide(j)) exit	2,796,664,000,000	0.882

Figure 2-7: Hotspots analysis of the score\_tracklength\_tally function in the tally module.

Source	Instructions Retired	CPI Rate
score = ZERO	17,576,000,000	2.879
if (i_nuclide > 0) then	185,640,000,000	0.968
if (nuclides(i_nuclide)%reaction_index%has_key(score_bin)) then	730,184,000,000	1.131
m = nuclides(i_nuclide)%reaction_index%get_key(score_bin)	327,184,000,000	1.106
! Retrieve temperature and energy grid index and interpolation		
! factor		
i_temp = micro_xs(i_nuclide) % index_temp	228,228,000,000	1.386
if (i_temp > 0) then	165,620,000,000	3.668
i_energy = micro_xs(i_nuclide) % index_grid	19,084,000,000	1.249
f = micro_xs(i_nuclide) % interp_factor	1,612,000,000	13.661
associate (xs => nuclides(i_nuclide) % reactions(m) &		
% xs(i_temp))		
if (i_energy >= xs % threshold) then	827,372,000,000	16.167
score = ((ONE - f) * xs % value(i_energy - &		
xs % threshold + 1) + f * xs % value(i_energy - &		
xs % threshold + 2)) * atom_density * flux	210,288,000,000	22.742

Figure 2-8: Hotspots analysis of the score\_general\_ce function in the tally module.

Finally, Figure 2-9 highlights the source-lines responsible for the bottleneck in the expand-and-score subroutine. Here, interestingly enough, only the one line to update the tally bins is responsible for the performance degradation. The issue with this operation is that it needs an atomic clause to prevent OpenMP race conditions; this performance bottleneck becomes even more pressing with the Phi since it has such a high number of threads each running their own independent histories.

Source	Instructions Retired	CPI Rate
case default		
!\$omp atomic	1,576,432,000,000	2.497
t % results(RESULT_VALUE, score_index, filter_index) = &		
t % results(RESULT_VALUE, score_index, filter_index) + score		
end select		

Figure 2-9: Hotspots analysis of the expand\_and\_score function in the tally module.

### Hotspots for the Cross-section module

This section discusses the specific source-lines that are responsible for the bottleneck in the cross-section module. OpenMC computes cross-sections for every nuclide each time a particle crosses to a new material or if the energy changes. The hotspot in Figure 2-10 is for the subroutine calculate\_nuclide\_xs . This one has the highest CPI. The section highlighted here has the logic for the binary search on the energy grid to determine which energy points bound the particle’s current energy. Using this information, OpenMC then interpolates for the corresponding cross-section. Figure 2-11 still refers to the same subroutine, the only difference now being the focus is on the section where the common macroscopic cross-sections are computed in OpenMC (i.e, (n,total), (n,fission), (n,absorption)). Here, multiple lines have a CPI rate of more than 10, which once again indicates long stalls. However, even though this subroutine is an obvious candidate for optimization, this thesis will only focus on presenting optimizations for the tally module.

Source	Instructions Retired	CPI Rate
macro_xs(i_nuclide) % interp_factor = zero		
else		
! Find the appropriate temperature index.		
kT = sqrtkT**2	6,292,000,000	4.570
select case (temperature_method)		
case (TEMPERATURE_NEAREST)		
i_temp = minloc(abs(nuclides(i_nuclide) % kTs - kT), dim=1)	264,576,000,000	9.723

Figure 2-10: Hotspots analysis of the calculate\_nuclide\_xs function in the cross-section module.

### Concluding Remarks

To recap, Intel VTune was used to obtain more detailed information about OpenMC’s performance on the Phi. As expected, both the tally and cross-section modules were

Source	Instructions Retired	CPI Rate
<code>associate (grid =&gt; nuc % grid(i_temp), xs =&gt; nuc % sum_xs(i_temp))</code>		
<code>! Determine the energy grid index using a logarithmic mapping to</code>		
<code>! reduce the energy range over which a binary search needs to be</code>		
<code>! performed</code>		
<code>if (E &lt; grid % energy(1)) then</code>	217,516,000,000	13.975
<code>  i_grid = 1</code>		
<code>elseif (E &gt; grid % energy(size(grid % energy))) then</code>	107,536,000,000	11.427
<code>  i_grid = size(grid % energy) - 1</code>		
<code>else</code>		
<code>  ! Determine bounding indices based on which equal log-spaced</code>		
<code>  ! interval the energy is in</code>		
<code>  i_low = grid % grid_index(i_log_union)</code>	4,108,000,000	20.712
<code>  i_high = grid % grid_index(i_log_union + 1) + 1</code>	1,716,000,000	17.129
<code>  ! Perform binary search over reduced range</code>		
<code>  i_grid = binary_search(grid % energy(i_low:i_high), &amp;</code>		
<code>    i_high - i_low + 1, E) + i_low - 1</code>	163,280,000,000	13.229

Figure 2-11: Hotspots analysis of the `calculate_nuclide_xs` function in the cross-section module

responsible for more than 30 % of the CPU execution time. A general exploration analysis also confirmed that OpenMC is back-end bound, which means it is not executing instructions efficiently. This inefficiency is due to long stalls while the cores are waiting for data from main memory. This information is supported by a memory-access analysis that revealed several important subroutines in OpenMC have a high amount of cache misses at the L2 level. The CPI of the entire application is almost 4, which again points to an inefficient CPU use. Three subroutines in the tally module and one in the cross-section module were primarily responsible for hurting performance. These subroutines essentially handle the logic of (1) identifying which nuclide bins should be updated, (2) computing their cross-sections to later compute  $S_{n,sums,event}$  and (3) updating the tally bins with an OpenMP atomic clause. As mentioned previously, only the bottlenecks related to the tally module will be addressed in this thesis. The implementation of optimizations for the cross-section module will be left for future work.

## 2.4 Optimizations

OpenMC's execution time was shown to be strongly dominated by cross-section lookups and scoring tallies. The SMR benchmark, in particular, had a 3–4x slowdown with respect to the inactive cycles during simulations with depletion reaction

rate tallies in each fuel region.

The previous section highlighted several bottlenecks within OpenMC’s tally module. This section will describe some key optimizations that were essential to improve performance.

### 2.4.1 Optimization 1

In OpenMC, the user can specify a list of nuclides for each score. This allows OpenMC to easily acquire microscopic reaction rates for all the nuclides on the list rather than having to specify a separate score for each desired microscopic reaction rate. This feature is utilized, for example, to obtain microscopic RRs needed for depletion tallies. One performance issue that was identified was in the nuclide search at tally-time. If the user asks for, say, the fission reaction rate for  $^{235}\text{U}$ ,  $^{238}\text{U}$ , and  $^{239}\text{Pu}$ , the code has to determine 1) if these nuclides are in the current material and 2) where to find the corresponding nuclide densities if they are (see Figure 2-7). OpenMC keeps a global array of nuclide data, and each material has an array of indices into the global nuclide array that is used to compare against when tallying. The search within this array of nuclide indices was being performed as a linear search; consequently, if the material contained 100s of nuclides, the search could end up being quite time-consuming. Effectively, if there are  $N$  nuclides in a material and  $M$  nuclides specified in the tally, the complexity of the “nuclide-in-material” search was  $O(MN)$ .

The simplest way to avoid the linear search over nuclides in a tally would be to store a direct-address table for each material. This data structure will then map a global nuclide index (which is listed in the tally) to a position in the material’s array of nuclide indices that could then be used to determine the appropriate offset in the array of nuclide densities. Further, direct-address tables have a small memory footprint per material while at the same time retaining an algorithmic complexity of  $O(1)$  for a single search [4]. It is important to note, however, that direct-addressing should only be used as a replacement for hashing when the known universe of keys is small. In this case, the universe of keys consists of indices into the global nuclide array, whose size will never exceed the number of nuclides for which incident neutron



interaction data exists (for example, 423 nuclides have such data in the ENDF/B-VII.1 library). A direct-address table is conceptually simple and easy to implement. It is represented by a single array where the index in the array *is* the key and the value would typically be a pointer to a piece of “satellite data” which contains the given value [4]. However, when the values are simple pieces of data themselves (in this case, integers), there is no need to store a pointer and instead the value is stored in the array directly provided that a suitable “null” value can be used for slots in the table for which no key/value pair is present. For this particular application, the null value requirement can be achieved by initializing the direct-address table to zero. Then, only the slots that correspond to nuclides that exist in the material will be filled with a non-zero value that represents the order of the nuclide in that particular material. To better illustrate how it works in OpenMC, suppose there is a global nuclide array like the one in Table 2.10 where each nuclide in the simulation corresponds to a position in the array. The corresponding direct-address table for a specific material will look like what is shown in Table 2.11. Each slot in the array corresponds to the index in the global nuclide array and hence, if the nuclide is present in the material, then a non-zero element will be found when that slot is accessed. With a search cost of  $O(1)$  per specified nuclide, using a direct-address table reduces the overall complexity of the nuclide-in-material search to  $O(N)$  from the original  $O(MN)$ .

Table 2.10: Global Nuclide Array

Index	Nuclide
1	$^1\text{H}$
2	$^{16}\text{O}$
...	...
16	$^{235}\text{U}$
17	$^{238}\text{U}$
...	...
22	$^{239}\text{Pu}$
...	...

Table 2.11: Direct Address Table for a material that contains  $^{235}\text{U}$ ,  $^{238}\text{U}$ , and  $^{16}\text{O}$  (in that order).

Slot	Element	Nuclide
1	0	$^1\text{H}$
2	3	$^{16}\text{O}$
...	...	...
16	1	$^{235}\text{U}$
17	2	$^{238}\text{U}$
...	...	...
22	0	$^{239}\text{Pu}$
...	...	...

## 2.4.2 Optimization 2

One other aspect of the reaction rate tallies that merits mentioning is how it is determined whether a reaction is present for a nuclide at all. Previously, a hash table that maps an integer reaction identifier (the so-called MT value) to an index into the array of reactions was stored for each nuclide. This hash table was highlighted

by the profiler as being extremely inefficient (see Figure 2-8). The issue here is that the aforementioned hash table had a naïve implementation because it was statically sized with  $\sim 5,000$  slots regardless of the number of key/value pairs. For example, for this specific search of a nuclide’s reaction, there was no need to have 5,000 slots since there can only be, at most, 891 reactions; this is according to the ENDF-6 format specification [10], which assigns MT values from 1 up to 891. Further, for each slot entry, the reaction index dictionary in OpenMC requires 8 bytes for each key/value pair and an extra 8 bytes for a null pointer. This was done for the particular case of a “collision”, which happens when two or more keys are assigned to the same slot by the dictionary’s hash function. Since hashing functions look to satisfy the assumption of simple uniform hashing, every key has an equal likelihood of getting hashed to any of the slots, which in this case is 5,000. Once a collision happens for a given slot, the pointer of the last inserted key will no longer be null and will now point to a new linked list that contains the new incoming key along with its corresponding value and null pointer. The downside to having multiple collisions is that a single search through the dictionary will be slower than the theoretical  $O(1)$ . The advantage of the naïve implementation is that it minimizes the chances of running into a collision because there are multiple slots available [4]. This, however, is introducing a non-negligible 80 KB memory penalty per nuclide ( $5,000 \times 16\text{B} = 80\text{KB}$ ). Table 2.12 illustrates a simplified example of a reaction index dictionary hash table for  $^{238}\text{U}$ , whom in this case only has the following reactions in this same order:  $(n, total)$ ,  $(n, fission)$ ,  $(n, \gamma)$ ,  $(n, 2n)$  and  $(n, 3n)$ .

Table 2.12: Simplified example of a reaction index dictionary hash table for a  $^{238}\text{U}$  nuclide with only  $(n, total)$ ,  $(n, fission)$ ,  $(n, \gamma)$ ,  $(n, 2n)$ ,  $(n, 3n)$  (in that order)

Slot	Key (MT value)	Value	Pointer
1	0	0	NULL
2	0	0	NULL
...	...	...	
8	1 $(n, total)$	1	NULL
...	...	...	
24	16 $(n, 2n)$	4	NULL
..	...	...	
42	17 $(n, 3n)$	5	NULL
...	...	...	...
53	18 $(n, fission)$	2	NULL
...	...	...	...
69	37 $(n, 4n)$	0	NULL
...	...	...	...
145	102 $(n, \gamma)$	3	NULL
...	...	...	...
173	103 $(n, p)$	0	NULL
...	...	...	...
202	107 $(n, \alpha)$	0	NULL
...	...	...	...
5000	...	...	...

In order to get around this hurdle, the reaction index dictionary hash table was replaced with another direct-address table of 891 elements (see Figure 2.13). This allows for a very simple  $O(1)$  determination of whether a reaction is present for a particular nuclide and if so, where to find cross-section data for it in the array of reactions. Even more so, since the direct-address table is a linear array of integers, the memory footprint is only 3.9 KB per nuclide ( $4\text{B} \times 891 = 3.9\text{KB}$ ) which is 20x

less than that of the dictionary hash table.

Table 2.13: Simplified example of a reaction index direct-address table for a  $^{238}\text{U}$  nuclide with only  $(n, total)$ ,  $(n, fission)$ ,  $(n, \gamma)$ ,  $(n, 2n)$ ,  $(n, 3n)$  (in that order)

Slot (MT value)	Element
1 ( $n, total$ )	1
2 ( $n, n0$ )	0
...	...
16 ( $n, 2n$ )	4
17 ( $n, 3n$ )	5
18 ( $n, fission$ )	2
...	...
37 ( $n, 4n$ )	0
...	...
102 ( $n, \gamma$ )	3
103 ( $n, p$ )	0
...	...
107 ( $n, \alpha$ )	0
...	...
891 ( $n, 2n_c$ )	0

### 2.4.3 Optimization 3

The final tally optimization was specific to tallies for depletion. As mentioned previously, in OpenMC, the rate of change of nuclide densities that is needed for transmutation calculations is determined by directly tallying microscopic reaction rates for each nuclide in a fuel region. The vast majority of transmutation comes from  $(n, fission)$ ,  $(n, 2n)$ ,  $(n, 3n)$ ,  $(n, 4n)$ ,  $(n, p)$ , and  $(n, \alpha)$  and  $(n, \gamma)$  reaction rates, so these seven reactions are tallied for each nuclide. While the fission cross-section is calculated by default, the other-cross sections must be evaluated at the time a tally is scored. This can become quite expensive, especially when a particle at high en-

ergy travels through many fuel materials without suffering a collision, resulting in the same cross-sections being re-evaluated. It is important to note, however, that this is only a concern when using a tracklength estimator. In order to mitigate this, the cross-section evaluation routine was modified to pre-calculate these reaction cross-sections when depletion tallies are present. Then, at tally-time, no extra work needs to be performed. This optimization is especially important if a single fuel pin is subdivided into multiple cells because it increases the likelihood that a particle will stream through multiple regions without changing energy. The only downside is that the depletion cross section cache introduces, at most, a memory penalty of 20.3 KB ( $423 \text{ nuclides} \times 6 \text{ cross-sections} \times 8\text{B/cross-section} = 20.3\text{KB}$ ). However, this small trade-off in memory is worthwhile considering the benefits of having the cross-sections readily available in a contiguous linear array [2].

#### 2.4.4 Performance Measurements

All the aforementioned optimizations<sup>1</sup> were implemented in OpenMC and tested against a baseline version (see Tables 2.14 and 2.15). The simulations were carried out on the SMR benchmark for both the Phi and Skylake. As for the settings, 1 million particles per batch were simulated in 30 batches (15 were used for the active cycles). The optimizations were implemented separately to gauge their individual impact on performance. A final version combines all of them together (see row 5). The speedup is computed by using the particle processing throughput in the active cycles ( $T_a$ ). The tally slowdown was also tabulated for each optimization. The tally slowdown in Table 2.14 ranged from 3.55 to 4.26 for the individual optimizations. The speedups, on the other hand, ranged from 1.14x to 1.31x. The depletion cache (optimization 3) had the best impact on performance. It can also be noted, however, that optimization 3 had the lowest  $T_i$  while the other two optimizations are closer to the baseline. This is also sensible because optimization 3 introduces an additional

---

<sup>1</sup>The results for optimization 3 on Table 2.14 did not include  $(n, p)$  and  $(n, \alpha)$  in the depletion cache. However, they were included in the final version together with all optimizations (see row 5, Table 2.14)

Table 2.14: Particle Processing Throughput Comparison using Optimizations (Skylake node).

Version (-)	Inactive ( $T_i$ ) [kn/s]	Active ( $T_a$ ) [kn/s]	Slowdown $T_i/T_a$	Speedup $T_a/T_{baseline}$
Baseline	74.0	15.2	4.87	1.00
Optimization 1	74.0	17.4	4.26	1.14
Optimization 2	73.5	18.9	3.90	1.24
Optimization 3	70.6	20.0	3.55	1.31
Optimizations 1,2,3	70.4	35.0	2.01	2.31

Table 2.15: Particle Processing Throughput Comparison using Optimizations (Phi node).

Version (-)	Inactive ( $T_i$ ) [kn/s]	Active ( $T_a$ ) [kn/s]	Slowdown $T_i/T_a$	Speedup $T_a/T_{baseline}$
Baseline	14.6	4.0	3.6	1.00
Optimization 1	14.7	4.3	3.4	1.07
Optimization 2	14.8	4.9	3.0	1.21
Optimization 3	14.0	5.7	2.5	1.37
Optimizations 1,2,3	14.2	8.7	1.6	2.15

linear array of type float while the other two do not. Once all the optimizations were put together, the tally slowdown and the overall speedup further improved to 2.01 and 2.31x, respectively. The Phi results follow a similar trend (see Table 2.15). Once all optimizations were implemented, the tally slowdown and speedup were 1.6 and 2.14x, respectively. However, as was pointed out in the last section, there is still a 3-5x difference between the performance of the active cycle of the Skylake and Phi.

## 2.4.5 Concluding Remarks

Three optimizations were implemented in OpenMC to address the performance bottlenecks in the tally module that were identified with the detailed profiling analysis. Optimization 1 improved the nuclide-in-material search in the tally module from  $O(MN)$  to  $O(N)$  by implementing a direct-address table. Here  $M$  stands for the number of nuclides specified for the RR tally while  $N$  stands for the number of nuclides in a given material. In this case, direct-address tables are a very simple data

structures of integers with a very small memory footprint. Optimization 2 replaced the naïve dictionary hash table in the reaction search with another direct-address table. The initial dictionary had a static length of 5,000 elements and a memory footprint of 80 KB. The direct-address table only had a 3.9 KB memory footprint while still keeping a single search complexity of  $O(1)$ . Finally, optimization 3 implemented a depletion cross-section cache to save all depletion cross-sections except  $(n, fission)$ . While there could be a memory penalty as high as 20.3 KB with this approach, the trade-off is worth it considering that depletion cross-section are frequently called for on LWR depletion simulations. In general, all optimizations resulted in a decent speedup, with optimization 3 having the highest one. The total speedup once all optimizations were put together was 2.31x and 2.15x, for the Skylake and Phi, respectively.



# Chapter 3

## Hybrid Tallies

Chapter 3 presents the analysis of the hybrid tally method and demonstrates it as a viable alternative to improve the performance of reaction rate (RR) tallies without introducing a memory penalty. As discussed in Chapter 1, the hybrid tally method works by tallying  $(n, fission)$  and  $(n, \gamma)$  RRs directly with a traditional RR tally while the remaining threshold rates are computed through the collapse of a flux-tally with a pre-generated MG library (using the same energy grid). Even at 2,500 groups, the memory requirements are in the vicinity of the baseline RR-only case. The goal is to, not only improve the performance, but to also do so with as few groups as possible. Therefore, this chapter will cover a performance assessment section as well as an error analysis to ensure the few-group structures are still capable of producing accurate results. The performance assessment will cover all five benchmarks discussed in Chapter 2. The error analysis, however, will be limited to only the pin-cell and the assembly. While testing the hybrid method, several issues were encountered with regards to the nuclear data and these will also be discussed as a way to caution developers that are interested in implementing this method.

### 3.1 Methodology

For this study, the hybrid tallying method has been implemented in OpenMC [18], which has a rich Python API that facilitates the generation of XML input files,

postprocessing, and depletion simulations. The nuclear data capabilities of the API were updated to call NJOY’s GROUPR module (to handle the MG cross-section generation) upon receiving an energy structure as input. The API was also extended to manage the automation of input files for the simulations with hybrid tallies and to facilitate the postprocessing analysis. The first step is to decide which energy structure should be used to generate the MG library. Initially for this study, a simple 2,500 group structure was selected. This structure ranges from  $10^{-5}$ eV to 20 MeV and was divided into three regions, using equal-lethargy bins in each one. The middle region goes from 1 eV to 1 MeV and accounts for 70 % of the total groups, while the remaining two regions have 15 % each (see Table 3.1). This group structure is based on one implemented by Fridman [8] and will serve as a good starting point to test the hybrid tally implementation and assess performance. In order to reduce the number of groups while retaining accurate rates, it was necessary to design an error metric to quantify the deviation from the reference RRs. Moreover, it is also important to

Table 3.1: Simple 2,500-Group Energy Structure

Energy Interval	Percentage of Total Groups
$10^{-5}$ eV – 1 eV	15 %
1 eV – 1 MeV	70 %
1 MeV – 20 MeV	15 %

gauge how relevant that MG rate will be for the depletion simulation. Equation 3.1 takes all of this into account by quantifying how much the use of the MG rates error contributes to the overall error:

$$\epsilon_{n,i} = \frac{|[RR_{n,i}]_{MC} - [RR_{n,i}]_{MG}|}{\sum_n \sum_i [RR_{n,i}]_{MC} \times 10^{-5}}, \quad (3.1)$$

where  $\epsilon_{n,i}$  is the error for nuclide  $n$  and reaction rate  $i$  in units of pcm,  $[RR_{n,i}]_{MC}$  is the reference reaction rate for nuclide  $n$  and reaction rate  $i$ , and  $[RR_{n,i}]_{MG}$  is the multigroup reaction rate for nuclide  $n$  and reaction rate  $i$ . A second, simpler metric

was also used to quantify the relative error of  $[RR_{n,i}]_{MG}$  with respect to  $[RR_{n,i}]_{MC}$ :

$$\omega_{n,i} = \frac{|[RR_{n,i}]_{MC} - [RR_{n,i}]_{MG}|}{[RR_{n,i}]_{MC}} \times 100, \quad (3.2)$$

which can help us assess the accuracy of fewer-group energy structures. In order to identify problematic rates, cutoffs of  $\epsilon_{n,i} > 10^{-3}$  pcm and  $\omega_{n,i} > 0.1\%$  were established for metric 1 and 2, respectively. Although the cutoff values are picked arbitrarily, they are conservative and are meant only to help analyze the impact of the energy structures on the accuracy of the MG reaction rates. The first metric  $\epsilon_{n,i}$  is more stringent because the denominator is the total absorption of the system, and as a result it highlights only those reaction rates whose contribution to the total error is significant. One previous study [9] found that threshold reactions such as  $(n, 4n)$ ,  $(n, p)$ , and  $(n, \alpha)$  can typically be neglected from the tallies without losing accuracy in depletion simulations because they make negligible contributions to the transmutation analysis. The metric  $\epsilon_{n,i}$  quantifies this observation and identifies those threshold rates that are more relevant and should be a priority. Hence, the number of reactions with  $\epsilon_{n,i} > 10^{-3}$  pcm will always be low for threshold MG rates. The drawback is that now it becomes harder to analyze how accurate these rates are for any given group structure regardless of their contribution to the total error. Since the second error metric  $\omega_{n,i}$  is simply the relative error with respect to the reference, it is more sensitive to modifications in the energy grid and hence is a convenient tool to analyze the general accuracy of *all* threshold MG rates.

In general, the procedure can be summarized with the following steps:

1. Select an energy group structure  $E$ .
2. Run NJOY (GROUPT), with  $E$  as input, to generate a multigroup library.
3. Generate the hybrid tally's XML file, using  $E$  as the filter of the flux tally.
4. Run the MC simulation.
5. Assess performance implications of the hybrid tally.
6. Verify accuracy with equations 3.1 and 3.2 to keep track of problematic rates.
7. Modify energy grid based on the corresponding cross-section data to improve

accuracy.

8. Repeat all previous steps until metrics and number of groups are desirable.

A performance comparison across all benchmarks was made to ensure that the hybrid method was expandable to more complex geometries. The error analysis, however, was limited only to the pin-cell and the assembly benchmarks. All simulations were ran on a single node with two Intel Xeon Platinum 8180 (Skylake) processors.

## 3.2 Results and discussion

### 3.2.1 Pin-cell Performance Assessment

OpenMC results for seven simulations of the pin-cell model, each with a different method of tallying, are summarized in Table 3.2. The first row corresponds to a pure RR tally (which is used as the baseline), the second row corresponds to a pure flux tally, and the remaining five rows are hybrids. Each hybrid tally here is based on the same 2,500-group flux tally. Table 3.2 also includes information on the number of nuclides and reactions that were directly tallied for each case. In the case of the number of nuclide bins, either 422 or 162 was used to account for long and short depletion chains, respectively. As for the rates, the hybrid tallies with just one rate correspond to a single  $(n, \gamma)$  RR tally, while the remaining ones include both  $(n, \gamma)$  and  $(n, fission)$  RR tallies. The intent was to study whether any severe performance implications arise when tallying only the two more relevant rates. The speedup for each case was compared with the baseline RR tally. Each simulation used 500,000 particles per batch and 50 total batches, with 20 of those used for the inactive cycles. The particle processing throughput in the active cycles was used as a metric to gauge the speedups of the hybrid tallies. Once again, the slowdown metric in Table 3.2, is used as a way of quantifying the negative impact of tallies on performance. The worst performance degradation can be observed for the simulation with pure RR tallies (row 1), as expected. The second row shows results for a simulation with just a single flux tally; hence the number of nuclides and rates is zero. This result serves

Table 3.2: Performance Comparison for Pin-Cell MC Simulations with Different Tallies. All flux tallies have 2500 groups.

Tally Version (nuclides/rates)	Inactive ( $T_i$ ) [kn/s]	Active ( $T_a$ ) [kn/s]	Slowdown $T_i/T_a$	Speedup $T_a/T_{RR}$
RR: 422/7	44.7	17.3	2.58	1
Flux: 0/0	44.8	43.6	1.02	2.51
Hybrid 1: 1/1	43.2	42.0	1.03	2.42
Hybrid 2: 162/1	43.6	38.6	1.13	2.22
Hybrid 3: 422/1	43.0	33.4	1.29	1.92
Hybrid 4: 162/2	43.0	36.2	1.18	2.09
Hybrid 5: 422/2	43.7	29.8	1.47	1.72

only as an upper bound value since all the hybrids will include RR tallies that will decrease performance. The first hybrid tally includes only a single RR tally for  $^{238}\text{U}$  ( $n, \gamma$ ), and it attains performance close to the upper bound, as one would expect. This is not a realistic option, however, because it would take more than 2,500 groups to ensure that the MG rates reach acceptable accuracy [3]. The other hybrid tallies that follow present more viable options. Hybrids 4 and 5, in particular, are the most appealing candidates. Not only do they provide a considerable speedup, but they also capture the two most relevant rates directly and take away the burden of escalating the number of groups further. Therefore, these options provide the best opportunity for utilizing a smaller energy grid, thereby minimizing the memory requirements.

### 3.2.2 Pin-cell Error Analysis

In addition to evaluating the impact on performance, it is important to assess whether a reduction in the number of energy groups results in unacceptable levels of error. To this end, equations 3.1 and 3.2 were used to analyze the accuracy of the MG rates. For each RR, all  $\epsilon_{n,i}$  above  $10^{-3}$  pcm and  $\omega_{n,i}$  above 0.1% were counted. Table 3.3 shows the number of reaction rates whose error was greater than the cutoff values for each of the error metrics. These results are specifically for the simple structure. As expected, there are more reaction rates with errors above the cutoff for the 500 energy group case. The ( $n, \gamma$ ) RR case in particular stands out with a count difference of

38 for the  $\epsilon$  metric. As for the second metric  $\omega$ , the count difference is higher, at 91. The fission counts are relatively low regardless of group structure, which makes sense because the bulk of groups for these structures are concentrated in the low and resolved resonance ranges. As for the remaining reactions and according to the  $\epsilon$  metric, there is not a significant difference between the number of counts of both group structures, with the exception of  $(n, 2n)$ , which goes up by 10. The count that stays constant between the two groups structures belongs to  $^{238}\text{U}$ , whose  $(n, 2n)$   $\epsilon$  error stands at 0.74 pcm. While the  $^{238}\text{U}$   $(n, 2n)$  cross-section is simple, the  $\epsilon$  error does not go below the cutoff at 2,500 groups because of  $^{238}\text{U}$ 's high number density and the low number of groups in the high energy range. However, all the remaining 10 counts for  $(n, 2n)$  are borderline cases that barely exceeded the stringent  $10^{-3}$  pcm cutoff (see Table 3.5). The  $\omega$  metric shows many more reaction rates exceeding the cutoff for the 500 group structure, with  $(n, 2n)$ ,  $(n, 3n)$ ,  $(n, p)$ , and  $(n, \alpha)$  all above 200 counts

Table 3.3: Pin-cell Error Analysis for simple 500 and 2500 Group Structures. Table tabulates the number of rates that exceeded the cutoff for each metric.

Groups	Metric	Fission	$(n, 2n)$	$(n, 3n)$	$(n, 4n)$	$(n, p)$	$(n, \alpha)$	$(n, \gamma)$
500	$\omega$	20	413	332	40	293	281	255
	$\epsilon$	17	11	1	0	1	1	142
2500	$\omega$	5	6	207	35	4	6	164
	$\epsilon$	12	1	0	0	0	1	114

The sole  $(n, \alpha)$  reaction rate that exceeded the  $\epsilon$  cutoff (which was found to be  $^{16}\text{O}$ ) also deserves more attention because it stays constant between the two group structures. With 500 groups, the error stands at 1.56 pcm, while at 2,500 groups it decreases to only 0.73 pcm, still two orders of magnitude above the threshold. Inspecting the  $^{16}\text{O}$   $(n, \alpha)$  cross section as a function of energy indicates why there is a problem: the cross section has multiple high energy resonances between 3 and 8 MeV, an energy range that was not covered by many groups in the simple structure (only 15% of the total number of groups are used for the energy region above 1 MeV).

Since this study focuses on using hybrid cases 4 and 5, the goal is to reduce the

error in the high-energy range while simultaneously reducing the number of total groups. To that end, the following steps were taken. The first action was to identify, for each nuclide, the threshold energy of all threshold cross-sections and use the minimum energy to adjust the cutoff between the resolved resonance and high-energy ranges. Figures 3-1 through 3-4 show the plots for each nuclides' threshold cross-section, except for  $(n, 4n)$ . The  $(n, 2n)$  minimum threshold energy is from  ${}^9\text{Be}$  and it is  $\sim 1.75$  MeV. As for  $(n, 3n)$ ,  ${}^{241}\text{U}$  has the minimum energy at  $\sim 10.5$  MeV.  $(n, p)$  and  $(n, \alpha)$  are trickier because several nuclides have cross-sections that are defined as low as  $10^{-6}$  eV. Nonetheless, the bulk of them cover the high energy range above 1 MeV. In order to address the error in the  ${}^{16}\text{O}$   $(n, \alpha)$  reaction rate, all resonance peaks were found and used to subdivide the high-energy range further and to concentrate more groups around the peaks.

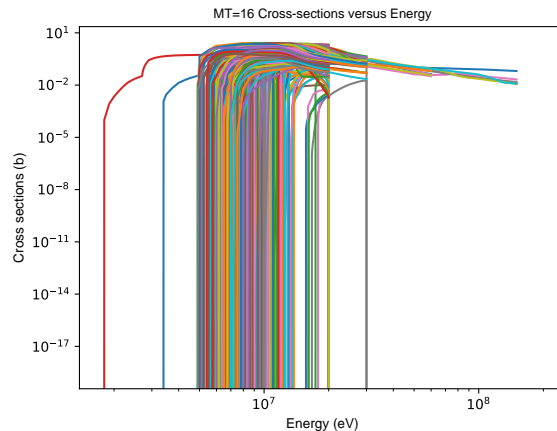


Figure 3-1:  $(n, 2n)$  plots for all 423 nuclides.

Table 3.4 shows the new energy group structures, one with 300 and the other with 500 groups. Comparing the  $\omega$  metric results of the new 500 group structure with the initial one, one can see that in general substantial improvements were made in the accuracy of the threshold reactions (see Table 3.6). Regarding the  $\epsilon$  metric results, however, 1 extra count was added to  $(n, \alpha)$ . The  ${}^{16}\text{O}$   $\epsilon$  error is now at 0.30 pcm, which is an  $\sim 80$  % drop from what was measured in the initial 500 energy-group structure. The extra count is for  ${}^{17}\text{O}$  and it is only an error of 0.00159 pcm. For the 300 updated energy-group structure, this error increased to 0.0280 pcm and the  ${}^{143}\text{Nd}$  and  ${}^{95}\text{Mo}$

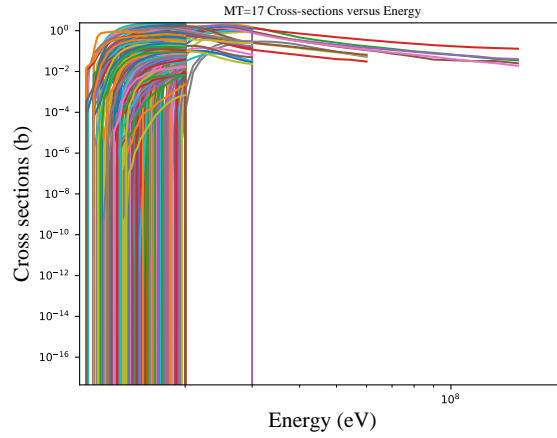


Figure 3-2:  $(n, 3n)$  plots for all 423 nuclides.

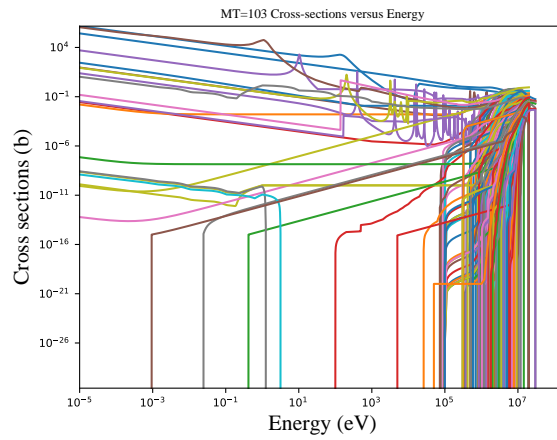


Figure 3-3:  $(n, p)$  plots for all 423 nuclides.

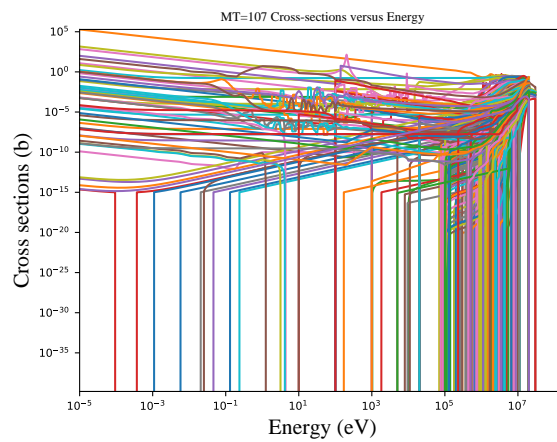


Figure 3-4:  $(n, \alpha)$  plots for all 423 nuclides.



$(n, \alpha)$  rates also exceeded the cutoff. The issue here is that the corresponding cross-sections for these nuclides are also defined in low energy regions, which now do not include as many groups in the updated structure. That being said, the  $\epsilon$  metric is still close to the threshold (i.e., on the order of  $10^{-3}$  pcm, see Table 3.7).

Table 3.4: Updated Energy Structures with 300 and 500 Total Groups

Energy Ranges	500 Groups	300 Groups
$10^{-5}$ eV – 400 KeV	50	10
400 KeV – 3.21 MeV	100	10
3.21 MeV – 8.025 MeV (26 intervals, 10 groups each)	260	260
8.05 MeV – 20 MeV	90	20

Table 3.5:  $(n, 2n)$   $\epsilon$  metric errors above the cutoff for 500 simple structure

Nuclides	$\frac{\epsilon}{10^{-3}}$ [pcm]
<sup>97</sup> Mo	1.10
<sup>101</sup> Ru	1.11
<sup>136</sup> Xe	1.82
<sup>142</sup> Ce	1.60
<sup>144</sup> Nd	1.08
<sup>145</sup> Nd	1.29
<sup>235</sup> U	2.08
<sup>236</sup> U	2.48
<sup>238</sup> U	740
<sup>239</sup> Pu	1.04
<sup>241</sup> Pu	1.14

Table 3.6: Error Analysis for updated 500 and 300 Group Structures. Table tabulates the number of rates that exceeded the cutoff for each metric.

Groups	Metric	Fission	$(n, 2n)$	$(n, 3n)$	$(n, 4n)$	$(n, p)$	$(n, \alpha)$	$(n, \gamma)$
300	$\omega$	83	356	332	40	308	309	415
	$\epsilon$	26	9	1	0	1	4	185
500	$\omega$	63	14	163	30	11	32	393
	$\epsilon$	22	1	0	0	1	2	165

The results for the 300 updated group structure, in general, are similar to those of the 500 initial group structure, especially with respect to the threshold RRs. The 8 extra counts in  $(n, 2n)$ 's  $\epsilon$  metric are a result of the 10x decrease in groups in

Table 3.7:  $(n, \alpha)$   $\epsilon$  metric errors above the cutoff for 300 updated structures

Nuclides	$\frac{\epsilon}{10^{-3}}$ [pcm]
$^{16}\text{O}$	444
$^{17}\text{O}$	28.0
$^{95}\text{Mo}$	3.86
$^{143}\text{Nd}$	6.33

the 400 keV–3.21 MeV. Once again, however, these errors are small and close to the threshold. The highest is once again for  $^{238}\text{U}$  ( $n, 2n$ ) at 0.086 pcm, which is still lower than what was measured in the simple 500 group structure. The rest are on the order of  $10^{-3}$  pcm (see Table 3.8). A series of depletion runs for the pin-cell geometry

Table 3.8:  $(n, 2n)$   $\epsilon$  metric errors above the cutoff for 300 updated structure

Nuclides	$\frac{\epsilon}{10^{-3}}$ [pcm]
$^{96}\text{Zr}$	1.30
$^{97}\text{Mo}$	1.01
$^{101}\text{Ru}$	1.04
$^{136}\text{Xe}$	2.31
$^{142}\text{Ce}$	1.76
$^{137}\text{Cs}$	1.15
$^{144}\text{Nd}$	1.52
$^{146}\text{Nd}$	1.01
$^{238}\text{U}$	86.3

were used to further study the accuracy of the hybrid tally method. In this test, the pin-cell was depleted to 50 months, with a constant linear power of 174 W/cm and a 4.5 % enrichment. All simulations had 500,000 particles per batch and 80 total batches, with 60 of those used for the active cycles. As was mentioned in Chapter 1, during a traditional OpenMC depletion simulation, once the transport ends for a given depletion step, all seven rates from the direct RR tallies need to be passed to the depletion module to do the transmutation analysis. In this study, however, the number of direct rates passed to OpenMC’s depletion module was varied from 7 (baseline), 1 (6 MG rates, excluding  $(n, \gamma)$ ), and 2 (5 MG rates, now excluding both  $(n, \gamma)$  and  $(n, fission)$ ). The goal was to quantify how much the number of hybrid tallies impact the  $k_{\text{eff}}$  accuracy. Three group structures were tested: the 300 and 500

group structures from Table 3.4 along with an extra 725 group structure that follows a similar format. Figure 3-5 shows  $k_{\text{eff}}$  for each run versus time. The first thing to note is that even with the 725 group structure, the 7 MG rates case does not come close to convergence as it oscillates around a  $k_{\text{eff}}$  of 1.29. This was expected considering that  $(n, \gamma)$  RRs require the most groups and 725 is not enough to get accurate rates. All remaining cases, however, are in qualitative agreement with the baseline results and follow the same basic trend. Figure 3-6 shows the  $^{235}\text{U}$  and  $^{239}\text{Pu}$  concentration profile as a function of time. The 7 MG case has 725 groups while the remaining others have 300 groups. The baseline behavior is that  $^{235}\text{U}$  and  $^{238}\text{U}$  decrease by about factors of approximately 10 and 1.5, respectively. The  $^{239}\text{Pu}$  concentration, on the other hand, increases until it reaches a similar concentration to that of  $^{235}\text{U}$ . In the 725 / 7 MG case, the issue is that, at each depletion step, all concentrations are seriously mispredicted. The  $^{238}\text{U}$   $(n, \gamma)$  RR is more than 10x higher than the baseline, which causes  $^{238}\text{U}$  to deplete faster. In the case of the  $^{235}\text{U}$  and  $^{239}\text{Pu}$  concentrations, they are off by up to a factor of 5 and 15, respectively, across all depletion steps. These effects prevent the  $k_{\text{eff}}$  from decreasing much. In light of this, the 7 MG rates case was not tested further with a smaller group structure. Figure 3-7 takes a more in-depth look at  $k_{\text{eff}}$ 's error for all remaining cases by comparing their pcm error with respect to the baseline results. Here it becomes clearer that throughout all depletion steps the 5 MG rate cases are more accurate than their 6 MG rate counterparts regardless of the group number. None of the 5 MG rate cases go over the  $\pm 100$  pcm dashed lines. The 6 MG rate cases, on the other hand, do not fall within these limits. For the 300 group / 6 MG rate case, the error falls below the -100 pcm line during the initial steps and above the 100 pcm for the last steps. As for the 500 group / 6 MG and 725 group / 6 MG cases, the error is mostly above 100 pcm after 30 months.

### 3.2.3 Assembly Comparison

The performance and accuracy of the hybrid tallies were also evaluated for the assembly benchmark. The settings that were used for the pin-cell run were also used here. Table 3.9 shows the performance in the inactive and active batches for the baseline

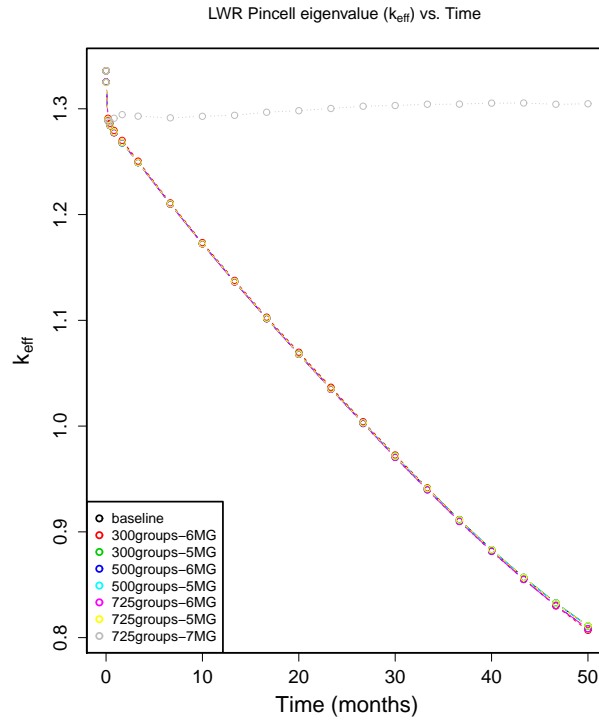


Figure 3-5:  $k_{eff}$  versus time

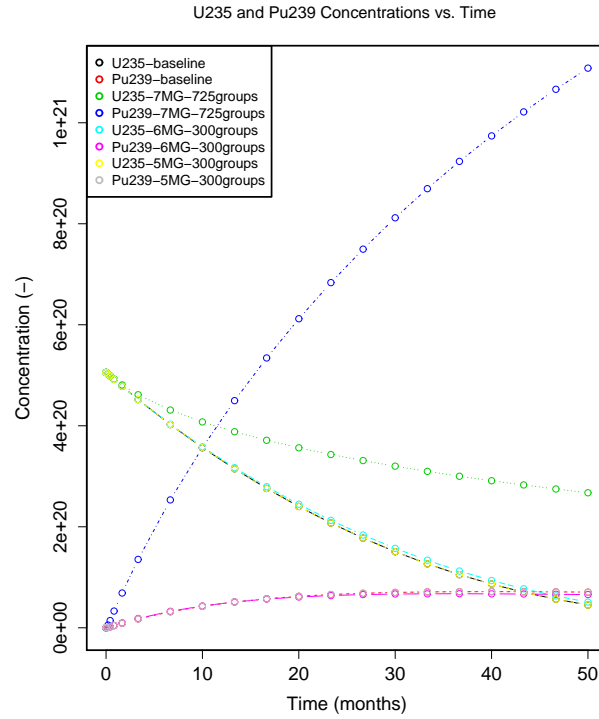


Figure 3-6:  $^{235}\text{U}$  and  $^{239}\text{Pu}$  Concentration as a Function of Time.

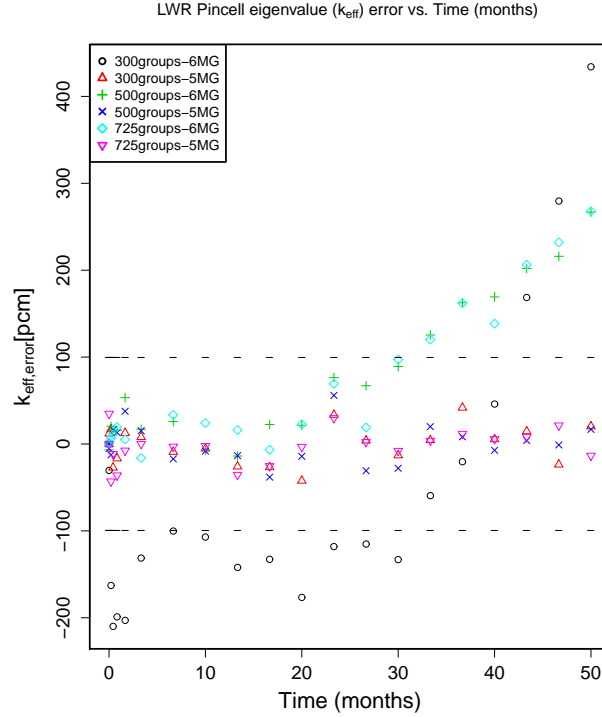


Figure 3-7:  $k_{\text{eff,error}}$  in pcm units versus time

RR tally method against the hybrid method. The results are similar to the pin-cell case with a tally slowdown of 1.28 and speedup of 1.69x. Table 3.10 shows the results for the error analysis using the simple energy group structures. Since there are 264 fuel pins, an average of the counts for each fuel pin was taken along with the standard deviation. In general, most averages are within two standard deviations of the pin-cell counts. These two results demonstrate that hybrid tallies have the potential to be expandable to other geometries.

Table 3.9: Performance Comparison Between Direct RR tally and Hybrid tally 5 for Assembly geometry. The flux tallies has 2500 groups.

Tally Version (nuclides/rates)	Inactive ( $T_i$ ) [kn/s]	Active ( $T_a$ ) [kn/s]	Slowdown $T_i/T_a$	Speedup $T_a/T_{RR}$
RR: 422/7	43.6	19.1	2.28	1
Hybrid 5: 422/2	41.4	32.4	1.28	1.69

Table 3.10: Assembly Error Analysis for simple 500 and 2500 Group Structures. The uncertainty for each value is indicated in parentheses.

RR	Metric	500 Groups	2500 Groups
Fission	$\omega$	22.0 (2.3)	9.0 (1.9)
	$\epsilon$	16.0 (1.1)	12.0 (1.4)
$(n, 2n)$	$\omega$	409.7 (7.4)	61.3 (38.9)
	$\epsilon$	9.0 (3.5)	0.99 (0.01)
$(n, 3n)$	$\omega$	314.4 (17.3)	286.9 (27.7)
	$\epsilon$	0.96 (0.2)	0.61 (0.49)
$(n, 4n)$	$\omega$	30.4 (10.3)	28.8 (10.5)
	$\epsilon$	0.0 (0.0)	0.0 (0.0)
$(n, p)$	$\omega$	291.0 (7.0)	60.9 (44.6)
	$\epsilon$	0.94 (0.24)	0.79 (0.41)
$(n, \alpha)$	$\omega$	278.0 (8.5)	45.3 (29.6)
	$\epsilon$	1.0 (0.0)	1.0 (0.0)
$(n, \gamma)$	$\omega$	264.0 (5.7)	197.0 (6.1)
	$\epsilon$	139 (2.7)	118.0 (12.7)

### 3.2.4 Performance Comparison across all Benchmarks

In order to further validate the hybrid tally method, hybrid 5 was tested for not only the pin-cell, but also for the assembly,  $2 \times 2$  fuel assemblies with water-reflector, the SMR (9,777 fuel regions) and a more detailed SMR (56,820 fuel regions). While the previous section already did a comparison with the assembly benchmark, the focus of this section is to only look at different performance metrics. In order to keep all benchmarks as consistent as possible, the total number of nuclides in the tally file was reduced from 422 to 293. The issue here is that in order to have all benchmarks with 422 nuclides, they need to be depleted to a high burn-up and this can be time consuming for the bigger benchmarks. Another option is to add 422 nuclides with dummy number densities. The approach taken here, however, was to deplete all benchmarks to 1 month and use whatever actinide build-up there was at that point. At the 1 month-mark all benchmarks had, more or less, 293 nuclides in each fuel region. Only 20 total batches (10 for the active cycles) were used in all simulations. It is important to reiterate that the total number of neutrons is still too low for realistic simulations but the results can still be use for performance assessment. Three cases were tested for all benchmarks. The first case is the baseline, which tallies all seven

RRs directly. The next two cases are both using hybrid 5, with a 2,500 energy filter. The only difference between these two is that one does the flux collapse serially while the other is multithreaded. Table 3.11 summarizes the results. Table 3.12 includes a definition of each variable in Table 3.11 and equations 3.4 to 3.6. This comparison is more detailed because it includes values for the  $T_{acc,b,n}$ ,  $T_{proc,b,n}$ ,  $T_{f,b,n}$ , and  $T_{tot,b,n}$ . Here  $b$  stands for the benchmark letter notation, which goes from A to E.  $n$  stands for the index of the three cases that were tested for each benchmark. Just as a quick overview,  $T_{acc,b,n}$  stands for the time spent accumulating tallies and  $T_{proc,b,n}$  for the time processing particles in the active cycles. Their sum makes up  $T_{active,b,n}$  which is the total time spent on the active cycles. Equations 3.3 and 3.4 show how these variables relate to one another and to the particle processing throughput in the active cycles  $T_{a,b,n}$ . Both  $T_{proc,b,n}$  and  $T_{acc,b,n}$  are expected to decrease with the hybrid tally method for two reasons: (1) flux tallies are faster than traditional RR tallies, hence the time processing particles should decrease and (2) the hybrid method reduces the tally memory footprint (as long as the number of groups is below 2,500), which helps speedup the time accumulating tallies. Moreover,  $T_{f,b,n}$  stands for the time spent finalizing the simulation (i.e., printing results, freeing memory, and collapsing the flux-tally in the case of the hybrid method). Therefore, it is a good metric to quantify the impact that the flux collapse has on performance. Comparing the  $T_{f,b,n}$  between cases 2 and 3 (i.e., hybrid-serial and hybrid-threaded) will also help assess if multi-threading is worthwhile. Finally,  $T_{tot,b,n}$  stands for the total time of the simulation. One can quickly determine if the hybrid cases have a better performance simply by comparing the  $T_{tot,b,n}$ . It is important to note, however, that one needs to be careful when using  $T_{tot,b,n}$  to compare cases 2 and 3 directly because, in theory, the hybrid serial version could outperform the threaded version. This is more likely if the benchmark is small because then, if the hybrid serial version finishes the active cycles a bit faster than the threaded version, it could end up with a smaller  $T_{tot,b,n}$ . For a bigger benchmark and a more realistic case with a higher number of neutrons,  $T_{f,b,n}$  will make-up a higher fraction of  $T_{tot,b,n}$  because there are more fuel regions to loop over when calling the collapse function. Therefore, even though there is a speedup in

terms of  $T_{tot,b,n}$  tabulated in Table 3.11 (see Equation 3.6), the more rigorous way of comparing cases 2 and 3 is by looking at  $T_{f,b,n}$ . Table 3.11 also tabulates a speedup in terms of the particle processing throughput in the active cycles (see Equation 3.5).

As can be noted from Table 3.11,  $T_{proc,b,n}$  is 1.31x to 1.75x smaller for the hybrid cases. As expected, cases 2 and 3 here have similar values. For each benchmark, except the detailed SMR assembly,  $T_{acc,b,n}$  does not change much across the three cases. When it comes to the detailed SMR, both  $T_{acc,E,2}$  and  $T_{acc,E,3}$  are considerably higher than the baseline  $T_{acc,E,1}$ . In the case of  $T_{f,b,n}$ , case 3 is consistently lower than case 2 but it is never as fast as the baseline case. This indicates that there is room for improvement in the threaded implementation. However, this is of no immediate concern because  $T_{f,b,n}$  is only a small fraction of the total time.  $S_{a,b,n}$ , on the other hand, ranges from 1.33x to 1.77x across all benchmarks. This is expected considering that the previous section showed evidence that the hybrid tally method is expandable to complex geometries. In the case of  $S_{tot,b,n}$ , the maximum is 1.30x for the pin-cell geometry and continues to decrease with increasing geometrical complexity; eventually  $S_{tot,b,n}$  gets as low as 0.90x for the detailed SMR. One possible explanation is that  $T_{acc,E,n}$  now accounts for most of  $T_{active,E,n}$ . This effect is worse for the hybrid tally cases. In OpenMC, tally accumulations occur at the end of each batch. Since the flux tally that was used to get the MG rates has 2,500 elements, the tally memory footprint is higher than the baseline case. While this is true for all benchmarks, it does become a more serious issue for the SMR due to the substantial increase in fuel regions.

$$T_{proc,b,n} = T_{active,b,n} - T_{acc,b,n} \quad (3.3)$$

$$T_{a,b,n} = \frac{N}{T_{proc,b,n}} \quad (3.4)$$

$$S_{a,b,n} = \frac{T_{a,b,n}}{T_{a,b,1}} \quad (3.5)$$



Table 3.11: Performance Comparison across all Benchmarks using Hybrid 5. All flux tallies have 2500 groups.

b	n	$T_{acc,b,n}$	$T_{proc,b,n}$	$T_{f,b,n}$	$T_{tot,b,n}$	$T_{i,b,n}$	$T_{a,b,n}$	$S_{a,b,n}$	$S_{tot,b,n}$
(-)	(-)	(s)	(s)	(s)	(s)	(kn/s)	(kn/s)	(-)	(-)
A	1	0.000829	218.4	$10^{-5}$	369	71.4	22.9	1.00	1.00
	2	0.00119	124.7	0.0146	281	67.4	40.0	1.75	1.31
	3	0.00114	125.0	0.0107	285	67.0	40.0	1.75	1.30
B	1	0.116	169.3	0.0128	324	72.3	29.6	1.00	1.00
	2	0.170	95.4	0.784	259	68.3	52.4	1.77	1.25
	3	0.167	96.2	0.0789	255	68.9	51.9	1.76	1.27
C	1	0.416	153.7	0.0014	299	81.7	32.5	1.00	1.00
	2	0.607	87.1	3.10	235	77.1	57.4	1.76	1.27
	3	0.609	86.2	0.14	232	78.7	58.0	1.78	1.29
D	1	3.17	71.9	0.0022	184	78.5	34.8	1.00	1.00
	2	5.75	43.1	24.3	190	75.4	58.0	1.67	0.97
	3	5.73	42.7	1.12	165	76.0	58.5	1.68	1.12
E	1	23.34	39.2	0.0056	184	209	128	1.00	1.00
	2	37.0	29.3	25.1	204	206	170	1.34	0.90
	3	36.8	28.8	1.2	193	208	173	1.36	0.95

$$S_{tot,b,n} = \frac{T_{tot,b,1}}{T_{tot,b,n}} \quad (3.6)$$

In order to improve the performance of the SMR cases, the number of generations per batch was adjusted. It is possible to improve the  $T_{acc,b,n}$  performance by increasing the number of generations per batch while reducing the total number of batches. This approach was tested on the detailed SMR benchmark and for each of the three cases  $n$ . The settings were modified to have 10 generations per batch and 10 total batches (5 for the active cycles). These were compared to another set of simulations that used only a single generation per batch and 100 total batches (50 for the active cycles). For all simulations, Hybrid 5 was used once again with 500 and 2,500 group structures (superscripts of 5 and 25 are used to distinguish the two). The results are summarized in Table 3.13. From here it is evident that increasing the number of generations per batch reduces  $T_{acc,E,n}$  for all cases and group structures by  $\sim 10x$ .  $T_{tot,E,n}$  is improved

Table 3.12: Definition of variables for Table 3.11 and equations 3.4 to 3.6

Variable	Definition
N	Number of particles
A	Pin-cell
B	Assembly
C	$2 \times 2$ assembly, with water reflector
D	SMR
E	Detailed SMR
b	benchmark notation: {A,B,C,D,E}
n	case index {1: RR, 2: hybrid-serial, 3: hybrid-threaded}
$T_{i,b,n}$	Particle Processing Throughput in inactive cycles
$T_{a,b,n}$	Particle Processing Throughput in active cycles
$T_{active,b,n}$	Total time spent in active cycles
$T_{proc,b,n}$	Time processing particles in active cycles
$T_{acc,b,n}$	Time accumulating tallies in active cycles
$T_{f,b,n}$	Time spent finalizing results
$T_{tot,b,n}$	Total elapsed time
$S_{a,b,n}$	Speedup in active cycles
$S_{tot,b,n}$	Speedup in elapsed time

by 1.14–1.42x.  $T_{f,E,n}$ , on the other hand, is not affected by increasing the number of generations per batch, as expected.  $T_{f,E,3}$  is 5x smaller than  $T_{f,E,2}$  but higher still than the baseline. Moreover, the 500 group cases outperformed all the 2,500 group cases. This is also sensible given that using a 500 group flux tally will require less memory and this in turn will have the positive impact of reducing  $T_{acc,E,n}$ ,  $T_{f,E,n}$  and  $T_{tot,E,n}$ .

Table 3.13: Performance Comparison for the detailed SMR using either a Single or Multiple Generations per batch. Superscripts of 5 and 25 stand for 500 and 2,500 Group Structures respectively.

Generations per batch	Total/Active batches	n (-)	$T_{acc,E,n}^5$ (s)	$T_{acc,E,n}^{25}$ (s)	$T_{f,E,n}^5$ (s)	$T_{f,E,n}^{25}$ (s)	$T_{tot,E,n}^5$ (s)	$T_{tot,E,n}^{25}$ (s)
1	100/50	1	119	116.5	0.0057	0.0057	512.8	515.9
		2	75.3	185.7	5.302	25.35	430.8	574.2
		3	74.5	193.7	0.338	1.23	430.7	553.1
10	10/5	1	11.9	12.7	0.0057	0.0058	420.2	421.1
		2	6.3	18.6	5.26	25.14	368.7	411.7
		3	6.4	18.7	0.30	1.22	362.8	386.5

## 3.3 Nuclear Data Inconsistencies

Several inconsistencies were identified in the cross-section libraries while testing the hybrid tally method in OpenMC for the pin-cell geometry. It is important that developers are aware of these inconsistencies in the nuclear data library in order to prevent additional sources of error. The goal of this section is then to highlight these inconsistencies as an effort to keep developers from doing an erroneous analysis while implementing the hybrid method.

### 3.3.1 $(n, p)$ and $(n, \alpha)$ levels

The first issue is related to the fact that nuclear data evaluators sometime separate the  $(n, p)$  and  $(n, \alpha)$  reactions into multiple reactions that correspond to the excited levels of the residual nucleus. By separating the total into the individual components, the evaluator is providing the user with more detailed information, which could be valuable to someone working on a different application from reactor physics, such as non-proliferation, detector work or medical imaging. For example, for a nuclide like  $^{74}\text{As}$ , the evaluator might choose to include only  $(n, p0)$ ,  $(n, p1)$ , ... ,  $(n, pc)$  (MT = {600, 601, ..., 649}) instead of the total  $(n, p)$ . As a result, when the user tallies the  $(n, p)$  reaction rate, the MC code will erroneously set it to zero because there is no data for MT=103 (i.e.,  $(n, p)$ ). Something similar could happen with  $(n, \alpha)$  (MT=107), in which case the evaluator will provide the levels  $(n, \alpha0)$ , ...,  $(n, \alpha c)$  (MT={800, 801, ..., 849}). As long as the developer is aware of this issue, it is straightforward to handle it by ensuring there is a total  $(n, p)$  and  $(n, \alpha)$  when only the levels are provided.

### 3.3.2 Reconstructed Cross-section versus ACE

There are also instances where an evaluator will provide both the total  $(n, p)$  and  $(n, \alpha)$  reactions as well as their corresponding levels. When OpenMC initially encountered this scenario, the original total  $(n, p)$  and  $(n, \alpha)$  was favored over a reconstructed version using the levels. However, for some cross-sections, like  $^{58}\text{Co}$   $(n, p)$

and  $^{154}\text{Gd}$  ( $n, \alpha$ ), the reconstructed version was found to be substantially different than the total provided (see Figures 3-8 and 3-9). Table 3.14 tabulates the  $\omega$  and  $\epsilon$  errors of the  $^{58}\text{Co}$  ( $n, p$ ) and  $^{154}\text{Gd}$  ( $n, \alpha$ ) MG rates after comparing them with reference rates that were computed using either the reconstructed or the original ACE cross-sections. The results show that the reconstructed version yielded a lower error for both metrics. No additional cases were spotted that were as serious as these. That being said, developers should still keep an eye out for similar cases.

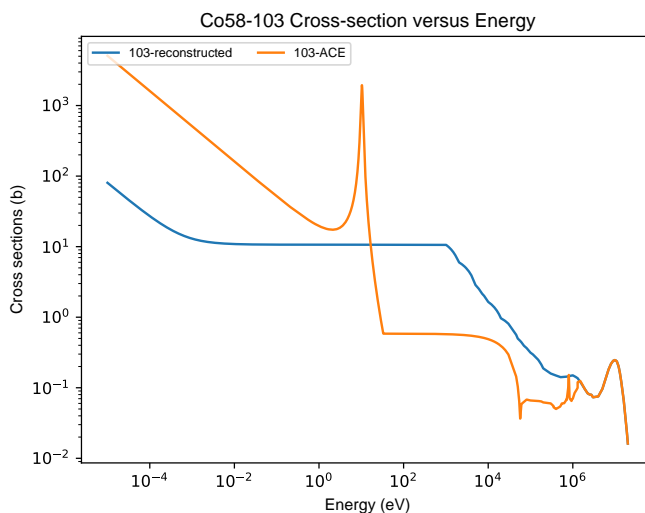


Figure 3-8:  $^{58}\text{Co}$  ( $n, p$ ) cross-section comparison between the original ACE data and the version reconstructed from the levels.

Table 3.14:  $\omega$  and  $\epsilon$  errors for  $^{58}\text{Co}$  ( $n, p$ ) and  $^{154}\text{Gd}$  ( $n, \alpha$ ) MG rates. Superscript  $r$  and  $a$  stand for reconstructed and ACE respectively.

Nuclides	Reaction	$\omega^a$ [%]	$\omega^r$ [%]	$\epsilon^a$ [pcm]	$\epsilon^r$ [pcm]
$^{58}\text{Co}$	( $n, p$ )	78.2	$10^{-4}$	$10^{-18}$	$10^{-24}$
$^{154}\text{Gd}$	( $n, \alpha$ )	18.5	$10^{-2}$	$10^{-10}$	$10^{-13}$

### 3.3.3 Window Multipole Library

OpenMC was recently updated to use the Window Multipole (WMP) formalism [15]. All the results presented in this chapter were from simulations at 300K. Additional simulations at 900K were also tested. However, the WMP formalism still can't handle threshold reactions where the threshold is in the low energy range (i.e.,  $10^{-6}$  eV) . In

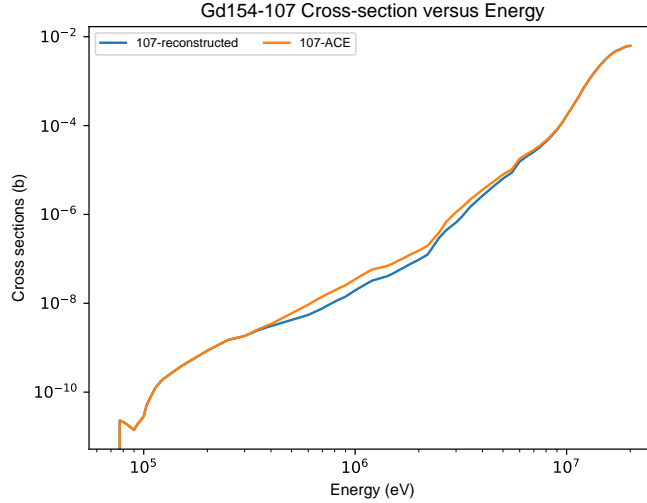


Figure 3-9:  $^{154}\text{Gd}$  ( $n, \alpha$ ) cross-section comparison between the original ACE data and the version reconstructed from the levels.

theses cases, the WMP library seriously mispredicts the cross-sections. Nonetheless, this problem was only identified for the ( $n, p$ ) and ( $n, \alpha$ ) reactions of about 80 nuclides. Table 3.15 summarizes the results for four of them just as a way of showcasing the problem. The  $\omega$  error is usually the highest at over 100 %. However, all these rates have very small  $\epsilon$  which means that they do not contribute much to the transmutation analysis and the overall error.  $^{17}\text{O}$  ( $n, \alpha$ ) has the highest  $\epsilon$  error but even for this case the error is still less than 1 pcm.

Table 3.15: Error for some threshold RRs with low energy thresholds at 900K. 2,500 groups were used.

Nuclides	Reaction	$\frac{\omega}{10^5}$ [%]	$\frac{\epsilon}{10^{-15}}$ [pcm]
$^{10}\text{B}$	( $n, \alpha$ )	3.4	3.5
$^3\text{He}$	( $n, p$ )	37.9	4.33
$^{17}\text{O}$	( $n, \alpha$ )	$10^{-3}$	$10^{14}$
$^{153}\text{Gd}$	( $n, \alpha$ )	$10^{-2}$	$10^6$



# Chapter 4

## Conclusions

Chapter 4 highlights the most relevant results from this entire study. This thesis analyzed the performance of a Monte Carlo transport code called OpenMC. As expected, the profiling analyses showed that the tally and cross-section modules were responsible for the major bottlenecks in the application. Several optimizations were implemented to address these. The first group of optimizations were rooted in more traditional computer science (Chapter 2) while the last one expanded on previous work in the field of reactor physics (Chapter 3). All of them will be briefly reviewed in the following sections to highlight their impact on performance. This Chapter concludes with a brief discussion on future work.

### 4.1 Profiling Analyses and First Optimizations

OpenMC was subjected to a thorough profiling analysis using two approaches. The first approach used the particle processing throughput in the active cycles as a metric to gauge performance. The clear advantage of using this method is that it makes it more straightforward to compare multiple settings such as the fuel type, the cross-section formalism, and the configuration of MPI processes / OpenMP threads, among others. The downside is that it only signals just how bad the performance is but it does not highlight which specific parts of the code are responsible for degrading performance. The second approach addresses this concern by using Intel VTune

Amplifier XE- a performance profiler that collects performance data (e.g., memory access analysis, thread profiling, hardware event sampling, etc). VTune is capable of pointing to the specific modules, functions/subroutines and source-code lines that are hindering performance. Throughout the profiling study five benchmarks were used:(1) pin-cell, (2) assembly, (3)  $2 \times 2$  fuel assemblies with periodic boundary conditions, (4)  $2 \times 2$  fuel assemblies with water-reflector and (5) SMR. With the first profiling approach, it was found that using depleted fuel over fresh fuel decreases performance by up to 25x (see Tables 2.6 and 2.7). This is obviously concerning because in realistic simulations more than 400 nuclides are present at high burn-up. None of the Phi-specific configurations had a significant impact on performance. Moreover, the simulations on Skylake were consistently 3-5x faster than on the Phi, which was attributed to the very distinct memory hierarchies between the two. Since the Phi does not have a L3 cache, a miss in the L2 forces a request from main memory which is very expensive. The detailed profiling with VTune confirmed and quantified that both the tally and cross-section modules in OpenMC were responsible for the bulk of the performance drop. Each one individually accounted for more than 30 % of the total CPU execution time (see Table 2.8). In the case of the tally module, three subroutines were highlighted as the most serious and two of them were addressed with a series of optimizations. In general, these optimizations improved both the nuclide and reaction searches by implementing direct address tables. The benefit of using this data structure is that they have a single search algorithmic complexity of  $O(1)$  and they also don't introduce a significant memory penalty. The last optimization was the introduction of a depletion cross-section cache, which helped improve the memory access. All optimizations were tested on the Phi and Skylake and the speedups were 2.15x and 2.31x, respectively (see Tables 2.14 and 2.15).

## 4.2 Hybrid Tallies

MC solvers can directly provide the reaction rates needed for transmutation analysis in burn-up applications. However, they often suffer from performance degradation



when tallying these quantities due to the high number of cross-section lookups at every event in the fuel region. In this project, this issue was addressed by proposing an alternative physics-based approach using hybrid tallies. Performance improvements of a factor of 1.70–2.50 were measured for the processing throughput of the active cycles, as shown in Table 3.2. The two most appealing hybrid tallies are hybrids 4 and 5 because they provide the best opportunity to reduce the number of energy groups by handling  $(n, fission)$  and  $(n, \gamma)$  reaction rates directly. Equations 3.1 and 3.2 were used to highlight problematic reaction rates and tailor the energy group structure accordingly. When going from 2,500 groups to 500 groups, using the simple structure, multiple error counts of threshold reaction rates went up significantly for both metrics (see Table 3.3). This result was expected because the simple structure has 70% of its groups in the RRR. In order to address this, the group structure was modified according to the counts with the highest error and the minimum threshold energies for each threshold reaction. The  $(n, \alpha)$  rate in  $^{16}\text{O}$ , for example, was found to have the highest error among the threshold reaction rates at 1.56 pcm because of its complex  $(n, \alpha)$  cross-section, which includes multiple resonance peaks at high energies. As a result, the updated group structure included many more groups in the high energy range, especially around the peaks of the  $^{16}\text{O}$   $(n, \alpha)$  cross-section (see Table 3.4). The improvements in the energy structure led to a reduction in the number of reaction rates whose error exceeded the cutoff; many of the remaining rates whose error exceeded the cutoff still had relatively small errors (see Table 3.6). In order to further analyze the hybrid tallies’ accuracy, a series of depletion simulations, each using a different number of energy groups and MG rates, were compared with a baseline simulation that used only direct RR tallies. From Figure 3-7, one can see that the results that correspond to hybrids 4 and 5 (i.e., 5 MG rates) are the most accurate since the error in  $k_{\text{eff}}$  is less than  $\pm 100$  pcm. The performance and accuracy of hybrid 5 were also briefly explored for an LWR assembly with 264 fuel regions. Neither the performance nor the accuracy deviated considerably from what was observed with the pin-cell case. An additional performance assessment across all five benchmarks was also carried out with 2,500 groups. A multi-threaded implementation of the hybrid

method was tested here to speedup the flux collapse. Results showed that there was a decent speedup with respect to the particle processing throughput in the active cycles  $T_{a,b,n}$  as well as in the final time of the simulation  $T_{tot,b,n}$ . The only exception was the SMR benchmark with the 55,680 fuel regions. The reason was that the  $T_{acc,b,n}$  for this case dominated the time in the active region  $T_{active,b,n}$ . This issue was handled by increasing the number of generations per batch, which improved the  $T_{acc,b,n}$  and  $T_{tot,b,n}$  performance by up to 10x and 1.42x, respectively. In general, the hybrid tally approach presents a viable alternative to improve the performance of LWR depletion simulations while simultaneously reducing memory requirements compared with the RR-only or flux-only tally methods. Finally, several nuclear data inconsistencies were reported in an effort to aid developers with keeping track of error sources related to nuclear data.

### 4.3 Future Work

There is still room for future research regarding the hybrid tally method. While this study went down to only 300 groups, future work should explore the accuracy of smaller group structures. This will make the hybrid method more attractive to architectures like the general processing unit (GPU), which has multiple registers with a small memory capacity on the order of KB [16]. There is also room to reduce the number of RR tallies that need to be tallied directly. Even at 300 groups, only a few dozen fission reaction rates were observed to exceed the cutoff with either metric. While directly tallying all the fission rates provides the most flexibility, one might push the analysis further and look into computing fission rates through a flux tally because this approach could result in further performance improvements. Future work should also look into testing this hybrid method on a harder spectrum.

# Bibliography

- [1] Intel Report 2018. Product change notification. 0827:1–2, 2018.
- [2] J. Bentley. *Writing Efficient Programs*. 1982.
- [3] E. Brun, E. Dumonteil, and F. Malvagi. Systematic Uncertainty Due to Statistics in Monte Carlo Burnup Codes: Application to a Simple Benchmark with TRIPOLI-4-D. *Progress in Nuclear Science and Technology*, 2(0):879–885, 2011.
- [4] Th. Cormen, Ch. Leiserson, R. Rivest, and C. Stein. *Introduction to*. 2009.
- [5] Intel Corporation. *Intel VTune Amplifier XE*. 2019.
- [6] F. Damian and E. Brun. ORPHEE research reactor: 3D core depletion calculation using Monte-Carlo code Tripoli-4®. *Annals of Nuclear Energy*, 82:203–216, 2015.
- [7] N. Dun, H. Fujita, J. Tramm, A. Chien, and A. Siegel. Article title: Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays. *International Journal of High Performance Computing Applications*, 29(3):348–365, 2015.
- [8] E. Fridman, E. Shwageraus, and A. Galperin. Efficient Generation of One-Group Cross Sections for Coupled Monte Carlo Depletion Calculations. *Nuclear Sci and Eng.*, 159:37–47, 2008.
- [9] D. Griesheimer, D. Carpenter, and M. Stedry. Practical techniques for large-scale Monte Carlo reactor depletion calculatons. *Progress in Nuclear Energy*, 101:409–423, 2017.
- [10] Cross Sections Evaluation Working Group. ENDF-6 Formats Manual: Data Formats and Procedures for the Evaluated Nuclear Data Files ENDF/B-VI and ENDF/B-VII. Technical report, 2010.
- [11] W. Haeck. *An Optimum Approach to Monte-Carlo Burn-up*. PhD thesis, Ghent University, 2007.
- [12] J. Hennessy and D. Patterson. *In Praise of Computer Architecture : A Quantitative Approach*. 2007.

- [13] J. Jeffers. *Intel Xeon Phi™ Processor High Performance Programming Knights Landing Edition Intel Xeon Phi Processor High Performance Programming Knights Landing Edition By*. Elsevier Ltd, 1 edition, 2016.
- [14] C. Josey, P. Ducru, B. Forget, and K. Smith. Windowed multipole for cross section Doppler broadening. *Journal of Computational Physics*, 307:715–727, 2016.
- [15] J. Liang, X. Peng, Sh. Liu, C. Josey, B. Forget, and K. Smith. Processing of a Comprehensive Windowed. *PHYSOR 2018: Reactor Physics Paving The Way Towards More Efficient Systems*, (June):3241–3258, 2018.
- [16] Nvidia. CUDA C Programming Guide. (April), 2018.
- [17] P. Romano and B. Forget. The OpenMC Monte Carlo particle transport code. *Annals of Nuclear Energy*, 51:274–281, 2013.
- [18] P. Romano, N. Horelik, B. Herman, A. Nelson, B. Forget, and K. Smith. OpenMC : A State-of-the-Art Monte Carlo Code for Research and Development. *Ann. Nucl. Energy*, 82:90–97, 2015.
- [19] P. Romano, A. Siegel, B. Forget, and K. Smith. Data decomposition of Monte Carlo particle transport simulations via tally servers. *Journal of Computational Physics*, 2013.
- [20] P. Romano, A. Siegel, and R. Rahaman. INFLUENCE OF THE MEMORY SUB-SYSTEM ON MONTE CARLO CODE PERFORMANCE. In *ANS MC2015*, Nashville, Tennessee, 2015. American Nuclear Society.
- [21] Paul Kollath Romano. *Parallel Algorithms for Monte Carlo Particle Transport Simulation on Exascale Computing Architectures*. PhD thesis, 2013.
- [22] M. Segev. Applied resonance equivalence. *Nuclear Science and Engineering*, 111(3):271–278, 1992.
- [23] J. Tramm and A. Siegel. Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes. *Annals of Nuclear Energy*, 82:195–202, 2015.
- [24] J. Tramm, A. Siegel, T. Islam, and M. Shulz. Xsbench – the Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, 2014.
- [25] J. Tramm, K. Yoshii, and A. Siegel. Power profiling of a reduced data movement algorithm for neutron cross section data in Monte Carlo simulations. *Proceedings of Co-HPC 2014: 1st International Workshop on Hardware-Software Co-Design for High Performance Computing - Held in Conjunction with SC 2014: The International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 17–24, 2014.

- [26] J. Walsh. *Computational Methods for Efficient Nuclear Data Management in Monte Carlo Neutron Transport Simulations*. PhD thesis, 2014.