Domain Decomposition for Monte Carlo Particle Transport Simulations of Nuclear Reactors

by

Nicholas Edward Horelik
B.S., Tufts University (2009)
M.S., Massachusetts Institute of Technology (2012)

Submitted to the Department of Nuclear Science and Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Nuclear Science and Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2015

© Massachusetts Institute of Technology 2015. All rights reserved.
Domain Decomposition for Monte Carlo Particle Transport

Simulations of Nuclear Reactors

by

Nicholas Edward Horelik

Submitted to the Department of Nuclear Science and Engineering on January 20, 2014, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Nuclear Science and Engineering

Abstract

Monte Carlo (MC) neutral particle transport methods have long been considered the gold-standard for nuclear simulations, but high computational cost has limited their use significantly. However, as we move towards higher-fidelity nuclear reactor analyses the method has become competitive with traditional deterministic transport algorithms for the same level of accuracy, especially considering the inherent parallelism of the method and the ever-increasing concurrency of modern high performance computers. Yet before such analysis can be practical, several algorithmic challenges must be addressed, particularly in regards to the memory requirements of the method. In this thesis, a robust domain decomposition algorithm is proposed to alleviate this, along with models and analysis to support its use for full-scale reactor analysis. Algorithms were implemented in the full-physics Monte Carlo code OpenMC, and tested for a highly-detailed PWR benchmark: BEAVRS.

The proposed domain decomposition implementation incorporates efficient algorithms for scalable inter-domain particle communication in a manner that is reproducible with any pseudo-random number seed. Algorithms are also proposed to scalably manage material and tally data with on-the-fly allocation during simulation, along with numerous optimizations required for scalability as the domain mesh is refined and divided among thousands of compute processes. The algorithms were tested on two supercomputers, namely the Mira Blue Gene/Q and the Titan XK7, demonstrating good performance with realistic tallies and materials requiring over a terabyte of aggregate memory.

Performance models were also developed to more accurately predict the network and load imbalance penalties that arise from communicating particles between distributed compute nodes tracking different spatial domains. These were evaluated using machine properties and tallied particle movement characteristics, and empirically validated with observed timing results from the new implementation. Network penalties were shown to be almost negligible with per-process particle counts as low as 1000, and load imbalance penalties higher than a factor of four were not observed or predicted for finer domain
meshes relevant to reactor analysis.

Load balancing strategies were also explored, and intra-domain replication was shown to be very effective at improving parallel efficiencies without adding significant complexity to the algorithm or burden to the user. Performance of the strategy was quantified with a performance model, and shown to agree well with observed timings. Imbalances were shown to be almost completely removed for the finest domain meshes.

Finally, full-core studies were carried out to demonstrate the efficacy of domain-decomposed Monte Carlo in tackling the full scope of the problem. A detailed mesh required for a robust depletion treatment was used, and good performance was demonstrated for depletion tallies with 206 nuclides. The largest runs scored six reaction rates for each nuclide in 51M regions for a total aggregate memory requirement of 1.4TB, and particle tracking rates were consistent with those observed for smaller non-domain-decomposed runs with equivalent tally complexity. These types of runs were previously not achievable with traditional Monte Carlo methods, and can be accomplished with domain decomposition with between 1.4x and 1.75x overhead with simple load balancing.

Thesis Supervisor: Benoit Forget
Title: Associate Professor of Nuclear Science and Engineering

Thesis Reader: Kord S. Smith
Title: KEPCO Professor of the Practice of Nuclear Science and Engineering
Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, US Department of Energy, under Contract DE-AC02-06CH11357. In addition, this research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory (Mira Blue Gene/Q), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory (Titan Cray XK7), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

I am extremely grateful to my thesis co-adviser Professor Benoit Forget for his support and patience ever since I came to MIT in 2009. He guided me through the entire process of becoming a graduate student and was constantly available to provide the vision needed for my research and the encouragement needed to keep me going. I couldn't have asked for a better mentor.

I would also like to express my sincere appreciation to my other thesis co-adviser Professor Kord Smith for his help and encouragement. I have learned an enormous amount from his insight and experience, both during the creation of this work as well as while developing the BEAVRS benchmark. Neither endeavors would have been successful without his guidance and support.

This work would also not have been possible without the experience and guidance of Dr. Andrew Siegel - the core logic of this implementation was conceived while working for him at Argonne National Lab, where his expertise in regards to high performance computing were instrumental in helping me achieve scalability and efficiency.

I am also particularly grateful to Paul Romano for creating OpenMC in such a clear and easy-to-understand manner, which impacted my development as a programmer and code designer significantly. Indeed, I have followed in Paul's footsteps nearly every step of the way at MIT, including with my Master's work.

I would also like to acknowledge many of my fellow graduate students who worked on OpenMC with me, particularly Bryan Herman, Will Boyd, and Derek Lax. The
development of OpenMC is very much a collaborative effort, and their contributions have been invaluable to my research efforts.

Finally, I wish to express my deepest appreciation to all my friends and family who provided me support and encouragement throughout. In particular, I must thank my Mother for feeding and sheltering me as I hid in her basement during the final stages of writing this document. Lastly, I want to thank my girlfriend and best friend, Carolyn, on whose love and support I have relied on since the beginning.
# Table of Contents

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>7</td>
</tr>
<tr>
<td>List of Figures</td>
<td>11</td>
</tr>
<tr>
<td>List of Tables</td>
<td>13</td>
</tr>
<tr>
<td>List of Algorithms</td>
<td>15</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Background</td>
<td>17</td>
</tr>
<tr>
<td>1.2 The Monte Carlo Method</td>
<td>19</td>
</tr>
<tr>
<td>1.3 Status and Trends in Computing</td>
<td>23</td>
</tr>
<tr>
<td>1.4 Motivation</td>
<td>25</td>
</tr>
<tr>
<td>1.5 Challenge Problem</td>
<td>26</td>
</tr>
<tr>
<td>1.5.1 NEA Monte Carlo Performance Benchmark</td>
<td>26</td>
</tr>
<tr>
<td>1.5.2 BEAVRS</td>
<td>27</td>
</tr>
<tr>
<td>1.6 The OpenMC Monte Carlo Code</td>
<td>29</td>
</tr>
<tr>
<td>1.6.1 Background</td>
<td>29</td>
</tr>
<tr>
<td>1.6.2 Overview of Program Flow</td>
<td>30</td>
</tr>
<tr>
<td>1.6.2.1 Particle Tracking</td>
<td>30</td>
</tr>
<tr>
<td>1.6.2.2 Tallies</td>
<td>31</td>
</tr>
<tr>
<td>1.6.2.3 Method of Successive Generations</td>
<td>33</td>
</tr>
<tr>
<td>1.6.2.4 Parallel Model</td>
<td>35</td>
</tr>
<tr>
<td>1.7 Summary of Issues</td>
<td>36</td>
</tr>
<tr>
<td>1.7.1 Time to Solution</td>
<td>36</td>
</tr>
<tr>
<td>1.7.2 Memory Requirements</td>
<td>38</td>
</tr>
<tr>
<td>1.7.2.1 Cross Section Data</td>
<td>38</td>
</tr>
<tr>
<td>1.7.2.2 Materials and Tallies</td>
<td>39</td>
</tr>
<tr>
<td>1.7.3 Domain Decomposition</td>
<td>41</td>
</tr>
<tr>
<td>1.8 Objectives</td>
<td>42</td>
</tr>
<tr>
<td><strong>Definitions and Acronyms</strong></td>
<td></td>
</tr>
<tr>
<td><strong>2 Domain Decomposition</strong></td>
<td>45</td>
</tr>
<tr>
<td>2.1 Background</td>
<td>45</td>
</tr>
<tr>
<td>2.1.1 Review of Prior Work</td>
<td>46</td>
</tr>
</tbody>
</table>

**Definitions and Acronyms**

**2 Domain Decomposition**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Background</td>
<td>45</td>
</tr>
<tr>
<td>2.1.1 Review of Prior Work</td>
<td>46</td>
</tr>
</tbody>
</table>
2.1.2 Recent Developments ................................................. 48
2.2 Current Approach ......................................................... 48
  2.2.1 Key Definitions ....................................................... 48
  2.2.2 Parallel Model ....................................................... 49
  2.2.3 Main Loop ............................................................. 50
  2.2.4 Inter-Domain Particle Communication ......................... 52
  2.2.5 Random Number Reproducibility ................................. 54
    2.2.5.1 Particle Tracking .............................................. 54
    2.2.5.2 Starting Source ............................................... 56
    2.2.5.3 Fission Bank Synchronization ................................. 58
  2.2.6 Memory Management ............................................... 58
    2.2.6.1 Particle and Fission Site Arrays ............................ 59
    2.2.6.2 Distributed Indexing .......................................... 60
    2.2.6.3 On-the-fly Allocation of Materials and Tallies .......... 61
  2.2.7 Tally Reduction ..................................................... 62
    2.2.7.1 Reduction of OTF tallies ..................................... 64
    2.2.7.2 Limitations .................................................... 65
  2.2.8 Inputs and Outputs ................................................ 69
    2.2.8.1 OTF Materials and Tallies ................................... 70
    2.2.8.2 Limitations .................................................... 71
2.3 Conclusion ................................................................. 71

3 Particle Communication Overhead and Load Imbalances .................. 73
  3.1 Introduction ............................................................. 73
    3.1.1 Communication Overhead .......................................... 74
    3.1.2 Load Imbalances .................................................. 75
    3.1.3 Prior Model Development ......................................... 78
  3.2 Analysis ................................................................. 79
    3.2.1 Definitions and Assumptions .................................... 79
    3.2.2 Perfect Load Balance ............................................ 81
      3.2.2.1 Constant Isotropic Leakage From All Domains ........ 81
      3.2.2.2 Variable Isotropic Leakage ................................. 83
    3.2.3 Uneven Load Balance ............................................. 84
  3.3 Results ................................................................. 88
    3.3.1 Communication and Load Imbalance Overhead ................. 89
    3.3.2 Model Evaluation ................................................ 92
  3.4 Conclusions ............................................................. 94

4 Load Balancing Strategies ............................................... 97
  4.1 Introduction ............................................................. 97
  4.2 Strategies ............................................................... 98
    4.2.1 Matching the Mesh to Load ...................................... 98
    4.2.2 Matching Resources to Load .................................... 100
      4.2.2.1 Intra-domain Fission Site Synchronization ............ 105
    4.2.3 Particle Density Manipulation .................................. 105
4.3 Results ........................................... 107
4.4 Conclusions ................................... 109

5 Full-Core Analysis .................................. 113
5.1 Introduction .................................... 113
5.2 Results ......................................... 115
  5.2.1 Non-OTF Tally Performance ............... 115
  5.2.2 OTF Tally Performance ..................... 120
    5.2.2.1 Overhead .................................... 121
    5.2.2.2 Large-Scale Runs ......................... 124
    5.2.2.3 Projections ................................. 127
5.3 Practicalities ................................... 128
  5.3.1 Geometry and Tally Specification ........... 129
  5.3.2 Domain Mesh Specification .................. 129
5.4 Conclusions .................................... 132

6 Conclusions ...................................... 135
  6.1 Particle Communication and Load Balancing ... 136
    6.1.1 Contributions of Present Work ............ 136
    6.1.2 Future Work .................................. 137
  6.2 Memory Treatment ............................... 138
    6.2.1 Contributions of Present Work ............ 138
    6.2.2 Future Work .................................. 139
  6.3 Full Core Analysis ............................. 140
    6.3.1 Contributions of Present Work ............ 141
    6.3.2 Future Work .................................. 142

Appendices ......................................... 143

A OpenMC Particle Data Structure .................. 145

B Depletion Nuclides ................................ 149

References ......................................... 151
# List of Figures

1-1 Uranium-238 capture cross section ........................................ 21
1-2 BEAVRS geometry ............................................................... 28
1-3 Example of OpenMC tally memory layout ............................... 34

2-1 Scheme for inter-domain particle communication algorithm .......... 54
2-2 Schematic of the 2D von Neumann neighborhood of range \( r = 2 \) .... 55
2-3 Example of OTF tally treatment ............................................. 63
2-4 Example of OTF tally results array reordering .......................... 67
2-5 Example of tally bin on more than one domain .......................... 68

3-1 Average leakage fraction vs. domain size for LWR problems ........... 75
3-2 Example domain leakage fraction distribution ............................ 76
3-3 Example domain work distribution for BEAVRS ........................... 77
3-4 Example domain mesh over the BEAVRS geometry ....................... 89
3-5 Simple domain decomposition overhead ................................... 90
3-6 Simple domain decomposition overhead vs load per domain .......... 91
3-7 Model evaluation for BEAVRS load imbalance and communication overhead 92

4-1 Example load balancing strategy: matching domain mesh to load ........ 99
4-2 Example load balancing strategy: matching resources to load .......... 101
4-3 Example load balancing strategy: Particle density manipulation ....... 106
4-4 Load balancing results for BEAVRS ......................................... 107
4-5 Model evaluation for restricted mesh load balancing strategy ......... 108
4-6 Process/load match quality for BEAVRS load balancing ............... 109
4-7 Model evaluation for process/load matching load balancing strategy .. 110

5-1 Non-OTF tally meshes for terabyte tally BEAVRS runs ................ 116
5-2 Non-OTF tally timing results for terabyte tally BEAVRS runs ........ 117
5-3 Non-OTF tally particle tracking rate overhead ............................ 119
5-4 BEAVRS domain mesh for depletion tallies ............................... 121
5-5 Domain decomposition overhead with OTF materials and tallies ....... 123
5-6 Load balancing with OTF materials and tallies ........................... 123
5-7 Active vs inactive calculation rate for full depletion tallies ............ 126
List of Tables

1.1 Summary of computer properties ........................................... 24
1.2 Meta-data memory requirements for key OpenMC data structures .... 40
3.1 Updated communication model verification ............................. 93
3.2 Updated communication model results for full reactor problems ...... 94
5.1 Non-OTF tally meshes for terabyte tally BEAVRS runs ............. 116
5.2 Non-OTF tally timing results for terabyte tally BEAVRS runs ...... 118
5.3 Full-core depletion test scenarios ........................................... 125
5.4 OTF tally timing results for terabyte tally BEAVRS runs .......... 126
## List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>OpenMC eigenvalue iteration main loop</td>
<td>35</td>
</tr>
<tr>
<td>2-1</td>
<td>OpenMC eigenvalue iteration outer loop with domain decomposition</td>
<td>51</td>
</tr>
<tr>
<td>2-2</td>
<td>Particle communication routine</td>
<td>53</td>
</tr>
<tr>
<td>2-3</td>
<td>Initial source distribution algorithm</td>
<td>57</td>
</tr>
<tr>
<td>2-4</td>
<td>OTF tally reduction algorithm</td>
<td>66</td>
</tr>
<tr>
<td>4-1</td>
<td>Resource matching routine</td>
<td>104</td>
</tr>
</tbody>
</table>
**Definitions and Acronyms**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAVRS</td>
<td>Benchmark for Evaluation and Validation of Reactor Simulations</td>
</tr>
<tr>
<td>CSG</td>
<td>Constructive Solid Geometry</td>
</tr>
<tr>
<td>CMFD</td>
<td>Coarse Mesh Finite Difference</td>
</tr>
<tr>
<td>DD</td>
<td>Domain Decomposition</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>LWR</td>
<td>Light Water Reactor</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>OTF</td>
<td>On-the-fly</td>
</tr>
<tr>
<td>PWR</td>
<td>Pressurized Water Reactor</td>
</tr>
<tr>
<td>PRN</td>
<td>Pseudo Random Number</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
<tr>
<td>SPMD</td>
<td>Single Program Multiple Data</td>
</tr>
<tr>
<td>UFS</td>
<td>Uniform Fission Site</td>
</tr>
<tr>
<td>URR</td>
<td>Unresolved Resonance Range</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

The ability to understand and predict the manner in which neutrons interact with matter is at the core of nuclear engineering. These interactions produce heat, transmute and distort materials, and fission the nuclei of heavy atoms to create more neutrons. Reactor physicists must understand the distribution and frequency of these interactions in order to arrange materials in such a way as to sustain a chain reaction of neutron generations. Properly managed, a population of neutrons can be used for a wide variety of important functions. Advanced scientific exploration, radioisotope production for medicine and industry, high-fidelity imaging, advanced cancer therapies, large-scale heat generation for electricity production... all are examples of what is possible with a stable and safe source of neutron production.

There are many intricacies to be accounted for in the design of an efficient and safe nuclear reactor, and all of them rely on an accurate understanding of the neutron distribution. For instance, local energy deposition rates affect structural material and coolant properties, which could lead to boiling, corrosion, melting, and release of hazardous materials if improperly managed. From a design perspective, the spectrum of neutron energies affects the distance neutrons will travel and the probability of interaction with matter, which in turn affects fuel utilization and decisions about sizing and positioning of fuel and moderator. All of these factors must also be understood.
in a time-dependent manner, since short term transients are very important to safety analyses of accident scenarios, and the nature and quantity of fission product production is critical for safe long-term disposal of waste.

We care very deeply about the accuracy and precision with which we can predict all of these phenomena, since even the smallest changes in safety margins can have far-reaching effects on the economics and public perception of nuclear technologies. As a result, the nuclear community has been striving towards achieving higher- and higher-fidelity solutions of the physics at length and time scales that have only recently become feasible with modern advances in computing. However, if these efforts are to be successful it is becoming increasingly clear that modifications will be needed to traditional methods of obtaining these solutions, especially in light of trends in computing.

Modern supercomputing has progressed a considerable amount in the last few decades, with massive installations enabling an unprecedented level of fidelity in a number of scientific fields. Much like how large datacenters have evolved since the dot-com era, these machines utilize hundreds of thousands of distributed servers with optimized interconnects and specialized fileservers to achieve aggregate performance of the order of petaFLOPs, along with terabytes of memory and petabytes of storage. Some of the codes that utilize these resources simulate materials for superconductor research, atmospheric models for climate science predictions, and molecular physics for combustion explorations - nuclear physics simulations should be able to benefit as well.

The algorithms traditionally used for simulating neutron spatial distributions in nuclear systems are varied, but in general they have always relied on approximations to various physics due to constraints in computing power and storage space. However, with the proliferation of high-performance machines it is desirable to revisit the methods with an eye for achieving higher fidelity results, revising the algorithms where necessary to efficiently utilize these new computing architectures.

This thesis focuses on improving our ability to simulate neutron populations using a well-established model and one of the highest-fidelity methods available for evaluating the relevant physical models. In particular, it explores modifications to the parallelization
algorithm in an effort to make high-fidelity simulations with this method feasible on modern and future supercomputing architectures.

### 1.2 The Monte Carlo Method

The physical process of neutron transport can be conceptualized as free particles moving through a medium in a random walk. For a simple example, an individual neutron traveling in a straight line can undergo an interaction with the nucleus of an atom in its path, changing its direction much like a ping pong ball bouncing off a bowling ball. Of course, since neutrons are the subatomic particles that make up nuclei these interactions are better described with a quantum mechanical formulation of a wave-function interacting with a potential well. Indeed, neutrons can also undergo inelastic scattering and absorption events, some of which destabilize the nucleus and lead to radioactive decay or fission events. These are processes that depend greatly on both the structure of the target nucleus and the energy (speed) of the incident neutron, and which must be well-described for us to have any hope of modeling the distribution of neutrons in any geometry.

Fortunately, our quantum-mechanical understanding of how neutrons are transported through materials has matured a great deal in the 20th-century, and a large amount of interaction probability data (or “cross section” data) has been amassed for a wide array of isotopes and molecular species for a wide range of incident neutron energies [1, 2, 3]. This topic encompasses a large field of both theory and experimentation, where the best-available sources of data are combined to create evaluated nuclear data libraries. For instance, the R-Matrix theory [4] and its various approximations, from single- and multi-level Breit-Wigner to the Reich-Moore [5] and multipole formalisms [6], allow us to build both point-wise tables and functional expressions for cross sections at arbitrary temperatures from measured and calculated resonance parameters.

This data allows us to solve for neutron particle distributions using statistical methods for a given system of atoms and neutrons. The Boltzmann equation is one of the more popular formalisms for this kind of system, where the density of particles is assumed
high enough over the length scales of interest such that the averaged behavior of all particles in a certain region of phase space is sufficient to describe the bulk behavior in that region. For typical nuclear reactor systems this is certainly the case, with neutron fluxes on the order of $10^{14}$ n/cm$^2$-s.

**Deterministic methods** are a historically popular class of methods that have been developed to solve the Boltzmann equation. They utilize discretized numerical methods to solve some mathematical formulation of the equation directly, for example in its differential form with discrete ordinates or spherical harmonics, or in its integral form with the method of characteristics [7, 8]. Low-order simplifications of the higher-order transport equation have also been used over the years, including diffusion approximations [9]. With proper treatment of the required discretizations, these methods can perform very well, allowing for multiple calculations per second of a full reactor core on consumer computing hardware with reasonable accuracy. However, these methods are not normally used for high-fidelity simulations, since they typically require coarse, regular meshes for stability as well as numerous approximations to the physics to accommodate discretization of the phase space.

This is particularly true in the energy domain, where many important cross sections exhibit a fine resonant structure that varies over several orders of magnitude. For example, Figure 1-1 shows a typical multi-group approximation to the fine energy structure of the Uranium-235 fission cross section. Herein lies the traditional difficulty with deterministic calculations, where the method of collapsing cross sections to fit in these groups can incur significant approximations. Specifically, to properly conserve each reaction $t$, cross-sections $\Sigma$ for the $g^{th}$ energy group in $E_g \leq E < E_{g-1}$ need to be determined using the angular neutron flux $\phi$:

$$\Sigma_{\epsilon,g}(\vec{r}) = \frac{\int_{4\pi} d\Omega \int_{E_{g-1}}^{E_{g-1}} dE \Sigma_{t}(\vec{r}, E)\psi(\vec{r}, \vec{\Omega}, E)}{\int_{4\pi} d\Omega \int_{E_g}^{E_{g-1}} dE \psi(\vec{r}, \vec{\Omega}, E)}.$$  \hspace{1cm} (1.1)

Since the angular neutron flux is indeed the unknown quantity we’re trying to solve for,
Figure 1-1: Uranium-235 fission cross section showing a typical multi-group discretization for deterministic methods.

typically an approximate scalar flux is used here instead by assuming that the energy and angular components of the flux are separable, which allows the angular component of the flux to cancel. For problems containing areas of highly anisotropic flux, such as near the outer reflector of a nuclear reactor core or at material discontinuities, this assumption of separability can break down. Furthermore, the scalar flux used here is typically the result of a finer-group deterministic calculation carried out with significant additional approximations that may fail to resolve all the spatial self-shielding effects encountered in a full-core reactor simulation. As a result, higher-fidelity methods are desirable to both verify the accuracy of deterministic multi-group methods as well as to tackle problems for which multi-group methods are not well-suited for.

*Stochastic methods* make up the other dominant class of methods for solving the Boltzmann equation, whereby random variables are sampled from their governing probability distributions and outputs are constructed as statistical estimates from the combination of many independent realizations. *Monte Carlo* [10] is one such method
widely used for simulation of neutron populations. Here, individual particles are tracked explicitly through an arbitrarily complex geometry and results are combined from these “histories” to estimate the statistically-averaged particle behavior in any region of phase space. The trajectories of particles are determined by randomly sampling the distance to collision using cross section data, and the type and outcome of interaction events (be they scattering or absorption) is further sampled using the ratios of event probabilities and associated data. Since this method can use point-wise continuous-energy data, it allows us to treat the physics with almost no approximations: the quality of our solution is limited only by the quality of the nuclear data and the computer resources available. This is precisely why Monte Carlo methods were selected for further exploration in the pursuit of higher-fidelity neutronics methods, especially in light of the availability of high-performance computers.

The need for large-scale computing resources when using Monte Carlo is primarily driven by the vast number of particle histories that must be simulated to arrive at an accurate solution to prototypical nuclear engineering problems. For instance, this thesis is primarily concerned with determining solutions for nuclear reactor systems, which of course means that neutrons are being transported through a multiplying medium (the fuel) to propagate the chain reaction of neutron production. This inherently makes the steady-state Boltzmann equation an eigenvalue problem, since the rate of change in the neutron population is itself dependent on that population. Since neutrons can travel a relatively large distance through reactor geometries before interacting, the solution to the distribution of neutrons between two regions of phase space is often very loosely coupled. This means that many of the higher eigenmodes for typical reactor problems lie close together, making them difficult to resolve with both deterministic and stochastic methods alike. As a result, a large number of power iterations are required to resolve the fundamental mode.

In addition, many billions or trillions of histories must be recorded before a Monte Carlo estimates of quantities will converge with acceptable statistical uncertainty and minimal biases. This is especially true considering the size of the problem, as discussed in Section 1.7. Combined with the large number of power iterations required, this aspect
makes the method very expensive computationally, and is the reason why Monte Carlo methods have historically not been used for routine analysis. However, recent advances in modern computing have made the method increasingly attractive, especially in light of the natural parallelism of the method.

One very advantageous feature of the Monte Carlo algorithm stems from one of the simplifying assumptions made when writing the neutron transport equation: neutrons don't interact with one another. This means that all particle histories can be carried out independently of one another within a generation, making the method inherently “embarrassingly parallel” [11]. This aligns well with the current trends in computing architectures. However, as discussed in subsequent sections, several challenges still need to be addressed before we can faithfully treat the true high-fidelity nuclear reactor problem with this method.

1.3 Status and Trends in Computing

Computing capability has grown exponentially according to Moore’s Law [12] throughout the latter half of the 20th century: roughly a factor of two increase of performance has been observed every 18 months. This has been driven by a number of advances in manufacturing capability that allow more transistors to fit on a single integrated circuit, and has effected all metrics of performance from processor speeds and memory capacity to network bandwidth. One key enabling component of this growth was Dennard Scaling [13], which stated that as transistors shrink the power density required to operate them stays constant. However, in around 2006 Dennard Scaling appeared to have broken down, heralding the arrival of “dark silicon,” where current leakage, thermal burdens, and other challenges began preventing all transistors on a chip from being fully utilized [14, 15, 16]. As a result, the computing community began pursuing multi- and many-core architectures, which is the only way Moore’s Law has been able to continue as applied to processing power according to industry road maps [17].

The High Performance Computing (HPC) community is keenly aware of these challenges, which have been taken into account in plans for next-generation exascale
Table 1.1: Approximate properties of current and projected computing platforms for high-fidelity physics simulations, drawn from a number of sources ([18, 19, 20, 21, 22]).

<table>
<thead>
<tr>
<th></th>
<th>Consumer Clusters</th>
<th>Current HPC</th>
<th>Projected HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Nodes</td>
<td>10-500</td>
<td>10K-50K</td>
<td>1M</td>
</tr>
<tr>
<td>Threads/Node*</td>
<td>8-32</td>
<td>32-3000†</td>
<td>1K-10K‡</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>0.1 byte per FLOP</td>
<td>1 byte per FLOP</td>
<td>0.2 byte per FLOP</td>
</tr>
<tr>
<td>Memory (total)</td>
<td>200GB-10TB</td>
<td>0.5-2PB</td>
<td>50-100PB</td>
</tr>
<tr>
<td>Memory (per node)</td>
<td>12-200GB</td>
<td></td>
<td>50-200GB</td>
</tr>
<tr>
<td>Storage Capacity</td>
<td>1-100TB</td>
<td>50-100PB</td>
<td>1EB</td>
</tr>
<tr>
<td>Filesystem I/O</td>
<td>100MB/s-50GB/s write</td>
<td>1TB/s write</td>
<td></td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td>1Gbps</td>
<td>100 Gbps</td>
<td></td>
</tr>
<tr>
<td>Network Latency</td>
<td>1-100µs</td>
<td>1-5µs</td>
<td>1µs§</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>10kW-1MW</td>
<td>10-20MW</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>$10K-100K USD</td>
<td>Install: $100-200M USD</td>
<td>Power: $1M USD per MW per year</td>
</tr>
</tbody>
</table>

* Hardware threads
† Includes GPUs and many-core coprocessors
§ Limited by the speed of light

machines. In particular, boosts in performance will increasingly rely on parallelism at lower chip voltages [18, 19, 20]. However, at the same time the total memory is growing at a slower rate than the aggregate available processing power, so that the amount of memory available to each individual processing unit is decreasing [21, 22]. This new balance will leave us in a regime quite distinct from what has become familiar over the past twenty years: algorithms will need to become more memory efficient to avoid starvation, and more aware of cache hierarchies to achieve good performance.

At this point it is useful to have a better understanding of what exactly these computing architectures look like; Table 1.1 gives some context to the magnitudes of various metrics for current and projected systems. An important point of understanding regarding HPC machines is that systems are made up of a large number of discrete computing “nodes” that contain separate processors and memory, and simulations that fully-utilize these systems require fast network interconnections to move data between nodes. Also note that the category of consumer clusters is widely variable, since installations can be made up of any combination of servers or other other machines.
all connected on the same network.

Metrics for HPC machines are typically aggregated over all the available compute nodes, though in practice is it typically very difficult for any simulation code to achieve theoretical peak calculation rates. This will be exacerbated in future exa-scale designs unless algorithms change, especially in light of increasing concurrency and decreasing memory availability. Indeed, as shown in Table 1.1, the amount of memory available per hardware thread\(^1\) is expected to decrease dramatically. This is primarily driven by the cost of adding memory capacity as well as the cost of powering that capacity, if the new machines are to stay within the design constraints of approximately $200M USD and 20MW. Exotic new configurations are also being explored [22], including clever new cache hierarchies, intra-node “networks-on-a-chip,” and a return to instruction bundling within the Single Instruction Multiple Data (SIMD) paradigm, but in general the memory outlook is not encouraging.

Non-trivial modifications will be needed to both deterministic and stochastic methods in order to take advantage of the increased parallelism within this new memory-constrained environment. Fortunately, Monte Carlo is already well-situated in this respect due to the embarrassingly parallel nature of particle histories. However, the key to actually making it work well will depend to a large extent on the actual configuration of the machines the algorithm will be run on.

### 1.4 Motivation

We desire higher-fidelity solutions of full nuclear reactor systems. This will help us better understand the current fleet of aging reactors, as well as better evaluate new designs. Even though Monte Carlo methods are capable of providing the highest-quality solution in this regard, for reasons previously discussed they have typically only been thought viable for either small reactors and experiments or for selective verification of coarse-grid deterministic methods. As a result, discretized deterministic methods have

---

\(^1\)“Threads” are sequences of programmed instructions fed to a processing unit often associated with multi-tasking. Different software threads can be executed by a single processing unit that alternates back and forth between them, whereas hardware threads are executed on separate processing units in parallel.
seen dominant use for routine design and analysis.

To achieve the level of fidelity desired it is clear that with deterministic methods we will be required to refine the discretization of the phase space considerably. With ultra fine meshes, the number of unknowns to solve for can become staggeringly large. In these cases it is clear that HPC systems will be required, and methods must be able to scale to many thousands of processors. In practice, this can be quite difficult for existing methods. Indeed, we come to a cross-over point where the amount of computational effort required to solve reactor systems deterministically is similar to that required for a Monte Carlo simulation to achieve similar accuracy.

In this context the advantages of using Monte Carlo come into better focus. Not only should it be much easier to achieve good scaling on highly-parallel machines, but the quality of the solution would be superior for the same level of computational effort. For instance, Monte Carlo methods can work with nearly exact fine geometrical structures as well as use continuous-energy cross section data to treat the physics exactly - precisely - without significant approximations. However, from a practical standpoint there still are challenges to overcome. In particular, some level of discretization will be needed to accomplish multi-physics coupling to depletion, thermal-hydraulics, and fuel performance routines. This issue results in enormous memory requirements for tallies and materials: solving these issues is the core focus of this thesis.

Further discussion of the exact nature and magnitude of the difficulties in carrying out a real high-fidelity reactor problem with Monte Carlo would benefit from a more explicit description of the problem to be solved. In addition, a more detailed description of the Monte Carlo algorithm will also make issues more clear.

1.5 Challenge Problem

1.5.1 NEA Monte Carlo Performance Benchmark

Over the last several years the largest Light Water Reactor (LWR) problem addressed by the reactor physics community has been the benchmark proposed by Hoogenboom and
Martin in 2009 [23]. This benchmark was developed in response to a talk by Smith in 2003 [24], where he challenged Monte Carlo methods developers to solve the power distribution for a highly-idealized, uniform temperature reactor problem to within 1% statistical uncertainty, with tallies on 10 radial rings per fuel pin along with 100 axial zones. Since then multiple credible solutions to the problem have been produced (e.g., [25, 26, 27]), demonstrating that while solvable, current Monte Carlo methods still required considerable amounts of memory as well as thousands of CPU-hours.

1.5.2 BEAVRS

In 2013 Smith and Forget issued a follow-on challenge that expanded the scope of the problem [28]. This included a more detailed requirement on the type and number of tallies needed to support a rigorous high-fidelity depletion treatment over two cycles of operation of a real 4-loop Westinghouse Pressurized Water Reactor (PWR). To support the challenge, a companion paper and detailed benchmark specification was co-developed by the present author: Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS) [29].

The BEAVRS PWR benchmark describes a realistic 3D full core nuclear reactor; a high-level overview of the benchmark geometry is shown in Figure 3-4. The specification describes a PWR with 193 standard 17x17-pin optimized fuel assemblies. Unlike the NEA benchmark, BEAVRS includes accurate descriptions of grid spacers, the core baffle, neutron shield panels, and other structural components. In addition, real plant data is provided for two cycles of operation, including detailed power histories, axial fission chamber traces from 58 instrument tube positions, zero power physics test data, and boron letdown curves.

Since the release of the new benchmark a number of preliminary attempts have been made to solve the problem according to Smith’s updated challenge with Monte Carlo. Several codes have demonstrated good agreement with the data for single statepoints as well as with preliminary core-follow studies including depletion ([30, 29, 31, 32]), but none so far have achieved the level of detail in the depletion mesh prescribed by the
<table>
<thead>
<tr>
<th>Elevation (cm)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>460.000</td>
<td>Highest Extent</td>
</tr>
<tr>
<td>431.876</td>
<td>Top of Upper Nozzle</td>
</tr>
<tr>
<td>423.049</td>
<td>Bottom of Upper Nozzle</td>
</tr>
<tr>
<td>421.532</td>
<td>Top of BPRA Rod Plenum</td>
</tr>
<tr>
<td>419.704</td>
<td>Top of Fuel Rod</td>
</tr>
<tr>
<td>417.164</td>
<td>Top of Fuel Rod Plenum</td>
</tr>
<tr>
<td>415.558</td>
<td>Top of Control Rod Plenum</td>
</tr>
<tr>
<td>415.164</td>
<td>Grid 8 Top</td>
</tr>
<tr>
<td>413.806</td>
<td>Grid 8 Bottom</td>
</tr>
<tr>
<td>403.778</td>
<td>Bottom of Control Rod Plenum</td>
</tr>
<tr>
<td>402.508</td>
<td>Top of Active Fuel</td>
</tr>
<tr>
<td>401.238</td>
<td>Top of Active Absorber</td>
</tr>
<tr>
<td>400.638</td>
<td>Control Rod Step 228</td>
</tr>
<tr>
<td>364.725</td>
<td>Grid 7 Top</td>
</tr>
<tr>
<td>359.010</td>
<td>Grid 7 Bottom</td>
</tr>
<tr>
<td>312.528</td>
<td>Grid 6 Top</td>
</tr>
<tr>
<td>306.813</td>
<td>Grid 6 Bottom</td>
</tr>
<tr>
<td>260.331</td>
<td>Grid 5 Top</td>
</tr>
<tr>
<td>254.616</td>
<td>Grid 5 Bottom</td>
</tr>
<tr>
<td>208.134</td>
<td>Grid 4 Top</td>
</tr>
<tr>
<td>202.419</td>
<td>Grid 4 Bottom</td>
</tr>
<tr>
<td>155.937</td>
<td>Grid 3 Top</td>
</tr>
<tr>
<td>150.222</td>
<td>Grid 3 Bottom</td>
</tr>
<tr>
<td>103.746</td>
<td>Grid 2 Top</td>
</tr>
<tr>
<td>98.0250</td>
<td>Grid 2 Bottom</td>
</tr>
<tr>
<td>41.8280</td>
<td>Bottom of Lower Absorber (AIC)</td>
</tr>
<tr>
<td>40.5200</td>
<td>Bottom of Active Absorber</td>
</tr>
<tr>
<td>39.9580</td>
<td>Control Rod Step 0</td>
</tr>
<tr>
<td>38.6500</td>
<td>Bot. of BPRA Rod</td>
</tr>
<tr>
<td>37.1621</td>
<td>Grid 1 Bottom</td>
</tr>
<tr>
<td>36.7480</td>
<td>Bottom of Active Fuel</td>
</tr>
<tr>
<td>35.0000</td>
<td>Bottom of Fuel Rod</td>
</tr>
<tr>
<td>20.0000</td>
<td>Bottom of Support Plate</td>
</tr>
<tr>
<td>0.0000</td>
<td>Lowest Extent</td>
</tr>
</tbody>
</table>

**Figure 1-2:** Geometry overview of the BEAVRS PWR benchmark. *Top Left:* Radial top-down view. *Top Right:* Radial top-down view of a single fuel assembly with 20 burnable absorber pins. *Bottom:* Axial side view showing a list of axial planes.
challenge.

It should be noted that the scope of new problem description extends beyond Monte Carlo, challenging all high-fidelity transport methods to tackle it at full-scale. The aggregate of solutions to this problem should verify if in fact we are at the cross-over point of effort/accuracy between stochastic and deterministic methods.

1.6 The OpenMC Monte Carlo Code

1.6.1 Background

OpenMC is an open-source general Monte Carlo neutron transport code written in modern Fortran and developed at MIT beginning in 2009 [33]. It was built from scratch as part of Paul Romano’s dissertation work expressly for the purposes of exploring high-performance algorithms, and has been shown to scale very well up to hundreds of thousands of processors on leadership-class HPC architectures [34]. It was decided early-on that realistic geometry and physics were required to faithfully demonstrate the capabilities of new algorithms, and by 2012 it had grown into a nearly\(^2\) full-physics code capable of utilizing continuous-energy cross sections and general Constructive Solid Geometry (CSG) geometries defined by quadratic surfaces. Since then it has become an ideal testbed for a wide array of research efforts, including the incorporation of CMFD acceleration [35] and several efforts to address temperature dependence of cross section data [36], among other things.

OpenMC is an ideal code to further the exploration of algorithms to make Monte Carlo work for large problems, as started by Romano with the exploration of improved algorithms for treatment of the fission bank [37], tally reduction [38], as well as initial explorations into both data and domain decomposition [39, 33]. The former topics were successfully shown to improve the scalability of particle transport and tally algorithms on HPC machines, but the decomposition schemes still must be proven at large scale to address memory concerns. The present thesis aims to do this via domain decomposition

\(^2\)Does not currently treat particles other than neutrons, e.g., photons.
with a robust implementation in OpenMC.

The high-level flow of OpenMC operation is not significantly different from most Monte Carlo transport codes, but in order to provide context to the issues and challenges that arise for large realistic problems it is useful to describe the flow of the main loops as well as any components relevant to HPC applications. Even though OpenMC is capable of fixed-source calculations, this discussion will focus on eigenvalue problems.

### 1.6.2 Overview of Program Flow

#### 1.6.2.1 Particle Tracking

Particle tracking is at the core of any Monte Carlo transport code. This is the process by which independent histories are created, from which estimates of quantities of interest can be calculated, or “tallied.” This process entails simulating the movement of particles through a geometry as a result of many random samplings of relevant physics. For example, a single history follows the flow:

1. A neutron generated by a fission event is initialized with some position \((x, y, z)\), some direction \((u, v, w)\), and some energy \(E\). The particle exists in a particular cell of the geometry, which contains a material.

2. A distance to collision \(d\) is sampled with a uniform random number \(\xi\) in \([0,1)\) using the total cross section \(\Sigma_t\) of the material at the particle’s position:

\[
    d = \frac{-\ln \xi}{\Sigma_t}
\]

   (1.2)

   This macroscopic cross section calculated using the atom densities of each isotope in the material as well as the microscopic cross sections from the nuclear data, which depends on the particle’s energy and the temperature of the material the particle is moving through.

3. The particle is advanced along its direction by the sampled distance. If the new position would move the particle to a different cell, the particle is instead moved...
to the boundary of the cell and step 2 is repeated for the new material in the adjacent cell.

4. An interaction nuclide is sampled using probabilities $P_i$ set by the ratios of individual cross sections for each nuclide $i$ in the material:

$$P_i = \frac{\Sigma_{i,i}}{\Sigma_i} \quad (1.3)$$

5. An interaction type $r$ is sampled in the same manner as step 4, using probabilities set by the ratios of microscopic cross sections $\sigma_r$ for each reaction contributing to the total microscopic cross section $\sigma_i$ of the selected isotope:

$$P_r = \frac{\sigma_r}{\sigma_i} \quad (1.4)$$

6. If a scattering event is sampled, a new direction and energy are sampled from scattering law data, and the process continues from step 2. Otherwise, if an absorption event is sampled particle tracking is over. Particle tracking also ends if a fission event is sampled, where a number of new neutrons are sampled using data for neutron multiplicity $\nu$ and stored for subsequent tracking in the next “generation” of particles.

### 1.6.2.2 Tallies

Tallies are the result of a Monte Carlo simulation, calculated from particle histories by counting the number of events of interest. As discussed in [33], OpenMC follows the methodology employed by the MC21 [40] Monte Carlo code, which defines a general tally quantity $X$ as

$$X = \int d\mathbf{r} \int d\Omega \int dE \psi(r, \Omega, E) f(r, \Omega, E), \quad (1.5)$$

where $\psi$ is the angular neutron flux, $f$ is a scoring function, and $r$, $\Omega$, and $E$ denote
the spatial, angular, and energy domains. This allows users to specify both “filters” and “scores” to define a tally, where scores denote a specific reaction type (e.g., fission events) and filters define a region of the domain to score in (e.g., a particular fuel pin of interest). Typically we are concerned with volume-integrated scores,\(^3\) for which OpenMC supports analog, collision, and tracklength estimators [33].

Tallies are fundamentally the estimate of the mean of a random variable as defined by the particular filter and score combination, meaning that we must calculate the associated uncertainty if we are to have any faith in our results. If we run enough particles to accumulate a large number of independent realizations of these random variables, the law of large numbers says the average value of those realizations should converge to the true average. Furthermore, the central limit theorem also guarantees that these realizations will arrive in a normal distribution. These allow us to calculate the mean \(\bar{x}\) and variance \(s_x^2\) for each tally as

\[
\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i
\]

\[
s_x^2 = \frac{1}{N-1} \left[ \sum_{i=1}^{N} x_i^2 - \left( \frac{\sum_{i=1}^{N} x_i}{N} \right)^2 \right]
\]

where \(x_i\) is an independent realization of the random number and \(N\) is the number of realizations.

OpenMC uses batch statistics, where combined tally results from all histories in each batch are considered one independent realization of the random variable. This differs from history-based statistics, where tallies from each individual history are a different independent realization. The use of batch statistics means that the averages calculated from each batch will be normally distributed as long as the conditions of the central limit theorem are met.

Algorithmically, the implications of batch statistics are that during simulation three floating-point numbers are required for each score/filter combination (or “bin”): a

\(^3\)OpenMC does support surface-integrated scores as well.
temporary variable during a batch for accumulating scores, and accumulators for
the sum and sum-squared terms needed for calculating the mean and variance in
Equations 1.6 and 1.7 between batches.\footnote{4}{i.e., since the result from the batch itself is the independent realization of the tally bin (rather than single particle histories), the sum and sum-squared accumulators aren’t used during particle tracking. The temporary variable is required to accumulate a result \textit{within a batch}, which is then contributed to the sum and sum-squared accumulators between batches.}

One point of particular relevance to this thesis is the way in which tallies are arranged
in memory and accessed in OpenMC. Specifically, bins (which consist of a data structure
containing the three accumulators) are strided into a single large array in a way that
indices for each bin can be easily calculated during simulation. Figure 1-3 attempts to
demonstrate this with an example of the arrangement in memory for a number of filter
and score bins. Note that in OpenMC this is implemented as a 2D array with scores in
the columns, so only the filter bin indices need to be calculated with striding meta-data.\footnote{5}{i.e., stride is the distance between consecutive bins of a given filter in the long 1D array.}

The use of a large single array for tally results is very important for efficient tally
reductions, and will be especially relevant in the discussion of the domain decomposition
implementation presented in Chapter 2.

1.6.2.3 Method of Successive Generations

The main loop of the OpenMC eigenvalue routine is shown in Algorithm 1-1. Full
particle histories are tracked one-by-one following the method of successive generations
first described in [41]. The key concept here is that a number of particles \textit{n\_particles}
are tracked in a batch until all particles in that batch are complete, and any new fission
particles created are stored in a buffer. In a subsequent batch, the same number of
particles are randomly sampled for tracking from this buffer, which is referred to as
the “fission bank.” It is important to note that the size of the fission bank is kept
under control by scaling the number of neutrons sampled during a fission event by the
globally-tallied eigenvalue of the problem.

The locations of the starting source of particles are typically sampled from a uniform
distribution over the geometry, but they can also be sampled from some analytical
distribution or from a restart file that contains sites from a previous run. Since tally
**Figure 1-3:** Example of OpenMC tally memory layout for a tally with two scores, 4 cell filter bins and multiple energy filter bins. Scores and filters are arranged as a 2D array in Fortran, so only filter indices need to be calculated during simulation using the stride metadata stored on the tally data structure. Note that nuclide bins are treated as separate scores in OpenMC, rather than as filters.
Algorithm 1-1: OpenMC eigenvalue iteration main loop.

1: Guess initial fission source distribution
2: for each generation of particles do
3: Sample particles from fission source distribution
4: for each particle sampled do
5: while Neutron not yet absorbed or leaked do
6: Sample \( d_{\text{collision}} \), find \( d_{\text{boundary}} \)
7: Sample physics, calc. tallies, advance particle
8: end while
9: end for
10: Synchronize tallies
11: Calculate eigenvalue
12: Rebuild fission source distribution
13: end for

Results of the simulation depend greatly on an accurate distribution of particles across the geometry, it is very important to converge the spatial distribution of the fission source. An important implication of this is that tallies cannot be accumulated until this distribution is accurate; otherwise, results will be biased. As a result, OpenMC distinguishes batches as either "inactive" or "active," depending on whether or not tallies are being accumulated.

1.6.2.4 Parallel Model

OpenMC follows the "single program, multiple data" (SPMD) parallel programming model, whereby every individual processor runs the entire algorithm independently of one another. For OpenMC, this means that all parallel computing units read the same input files and allocate the same memory at the beginning of any simulation. Where they differ is in the particles they track: at the beginning of a simulation each processor samples a different subset of the total \( n_{\text{particles}} \) for the batch and tracks them through geometry loaded into local memory.

At the end of each batch, information global to the problem is then collected, or "reduced" to one master process for combination of tally accumulators and calculation of statistics. This has been historically applied to tallies and fission bank sampling, both of which can be a major point of inefficiency if not done correctly. Previous works have
addressed both in OpenMC [37, 38], and subsequent works\(^6\) continue to explore ways to further optimize the treatment of tallies.

This model is often referred to as “task parallelization” or “domain replication,” and it requires that all processors duplicate all memory for the whole problem.

### 1.7 Summary of Issues

Several challenges become apparent given the scope of the problem of interest; it is clear that Monte Carlo solutions will require an exorbitant amount of computational effort for large reactor problems. For instance, a single depletion statepoint will require a solution to the Bateman equations for millions or billions of independent regions, requiring a very detailed tally mesh. Billions or trillions of particle histories must be carried out to converge the statistics of these tallies, potentially requiring millions of CPU-hours. Next-generation machines promise to address this with increased parallelism given the inherent independence of particle histories, but this only solves part of the problem, leaving a number of areas that still need attention.

#### 1.7.1 Time to Solution

It has long been recognized that a major disadvantage of using Monte Carlo for reactor problems stems from the vast number of particles that must be tracked, which would require an excessive amount of computation time. This is driven by two factors: (1) a need to converge the fission source site distribution before tallies can be accumulated; and (2) a need to converge tally statistics to a reasonable accuracy.

The first issue of slow source convergence is primarily a result of the high dominance ratio of reactor geometries. This requires a large number of power iterations (neutron generations), which each require a considerable number of particle histories to avoid under-sampling the spatial distribution of fission sites. This is problematic because all particle histories tracked during inactive batches to converge the fission source cannot

\(^6\)e.g., [39] as well as the present work.
be used to accumulate tally results: a significant amount of computational effort would be wasted. Fortunately, the issue of Monte Carlo source convergence has been studied extensively. As detailed by Romano in [33], numerous acceleration techniques have been devised, and several have been implemented in OpenMC.⁷

The second issue is largely the result of the size of the problem, both in regards to the mean-free-path of neutrons in a reactor and the number of regions to tally over. For a crude approximation, consider the case where 50M spatial regions are to be tallied. If each neutron travels through approximately 100 spatial regions before being absorbed, and the problem is approximately perfectly load-balanced, running a simulation with 1M particles per batch would only score to each spatial tally bin twice. This situation would obviously lead to terrible statistical accuracy unless many more neutron histories are simulated or a large number of batches are carried out. Analogously, filters over energy or angle⁸ create even more tally bins, requiring an even larger number of particle histories, especially considering spatial self-shielding.

This situation is made even worse due to correlation effects between generations of particles. In particular, from Equation 1.7 we might expect tallies to converge as $1/\sqrt{N}$. Unfortunately, as shown by Herman et al. this can be difficult to achieve without running a significant number of particle histories per generation (on the order of 50M for 3D reactor problems) [42]. This is likely a result of the particle clustering that arises when particle absorptions are uniformly-distributed and particle births are spatially-localized and correlated to the positions of parent particles [43].

Beyond simply running more particles, a number of methods have been explored to improve the statistics of Monte Carlo solutions. These can include the use of implicit estimators for tally scores (e.g. tracklength estimators for volume-integrated flux), as well as various “variance reduction” techniques such as weight windows, implicit capture, particle splitting and Russian Roulette, etc. [44]. Such methods can force extra particle scoring events in specific regions of phase space while proportionally adjusting the “weight” of particles to avoid bias. Unfortunately, this often requires

---

⁷e.g., coarse-mesh finite-difference low-order operator acceleration [35].
⁸e.g., surface current tallies often filter over neutron angles, which facilitates coupling to low-order acceleration scheme.
intricate user-knowledge of the problem to choose optimal parameters, though some automatic techniques have been developed, e.g., adjoint-informed weighting [45] and the Uniform Fission Site (UFS) method [27, 46].

1.7.2 Memory Requirements

Memory utilization in Monte Carlo simulations is of particular importance. Specifically, the memory required to support high-fidelity simulations can become prohibitive especially in light of shrinking memory available per processor.

1.7.2.1 Cross Section Data

One of the difficulties of simulating the BEAVRS problem according to Smith's updated challenge lies in the temperature treatment of cross section data: every region in the problem is at a different temperature. Since evaluated cross section data is reported at 0K, data libraries are typically pre-processed to produce data files at particular temperatures to account for Doppler broadening, for example using NJOY [47, 48]. Interpolation between data files for different temperatures is then used during particle transport to calculate cross sections for materials at any temperature.

The number of cross section interpolation libraries needed for accurate simulations of LWRs was studied extensively by Trumbull in [49], finding that intervals between 28K and 111K would be needed for interpolations to result in minimal errors, depending on the nuclide and the energy range (and heavier isotopes with detailed resonance behaviors might require even finer intervals). Since steady-state and transient simulations require temperature data between approximately 300K and 2500K, this means around 50 libraries would be needed. Each one can be on the order of gigabytes in size, so this would impose a significant memory burden: data for all cross sections would certainly not fit within memory for a single compute node.

Several techniques have been explored to address this challenge with the aim of replacing pre-processing and memory-storage at multiple temperatures with on-the-fly calculations using only the 0K data. This includes treating the data with a multipole
series expansion to do Doppler broadening on-the-fly [50, 51, 52, 36], as well as techniques that treat thermal motion explicitly during particle tracking [53, 54]. Both of these methods include some drawbacks in terms of additional overhead, but they do promise to solve the memory issue.

1.7.2.2 Materials and Tallies

From the description of the full nuclear reactor problem discussed in Section 1.5, it is clear that the number of independent depletion regions gives rise to another immense memory burden. For instance, if each of the 50,952 fuel pins in the BEAVRS geometry are treated with 10 radial rings and 100 axial sections, the model will have almost 51 million fuel regions, each of which containing a material with roughly 300 nuclides. This is problematic for a number of reasons.

The primary challenge comes from the tallies required to carry out the depletion calculation, which involves solving the Bateman equations [55]. These equations require up to six reaction rate tallies for each nuclide: the \((n, \text{fission})\), \((n, \gamma)\), \((n, 2n)\), \((n, 3n)\), \((n, \alpha)\), and \((n, p)\) reaction rate tallies for every nuclide. Since every tally requires 24 bytes, this amounts to \((51M \text{ regions}) \times (300 \text{ nuclides}) \times (6 \text{ tallies}) \times (24 \text{ bytes per tally}) = 2,201,126,400,000 \text{ bytes, or 2 terabytes.}\) This requirement is only expected to grow higher with further refinements in the depletion mesh and increased availability of nuclear data libraries.\(^9\)

In addition to tallies, a sizable amount of memory is also required to support particle tracking. For instance, number densities must also be stored for each nuclide in each material for use during cross section calculation adding another several hundred gigabytes.\(^{10}\) Furthermore, the geometry, material, and tally data structures themselves will add another hundred gigabytes if not properly optimized: Table 1.2 shows the memory requirements for each of these data structures, excluding the memory for nuclide number densities and tally scores.

Clearly the current SPMD model cannot support this environment, since the amount

\(^9\)The set of 300 nuclides is chosen because particle interaction data exists for them - actual fuel contains thousands of nuclides.

\(^{10}\)(51M regions) \times (300 nuclides) \times (8 \text{ bytes per density}) = 122,284,800,000 \text{ bytes, or 114 gigabytes.}\)
Table 1.2: Approximate meta-data memory requirements for the key OpenMC data structures, assuming 300 nuclides per material and tally as well as geometrical complexity similar to that required for typical LWR geometries. The size of material and tally objects is largely determined by the number of nuclides.

<table>
<thead>
<tr>
<th>Object</th>
<th>Memory Requirement</th>
<th>Total (gigabytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface</td>
<td>70</td>
<td>.5</td>
</tr>
<tr>
<td>Cell</td>
<td>250</td>
<td>11</td>
</tr>
<tr>
<td>Universe</td>
<td>800</td>
<td>-</td>
</tr>
<tr>
<td>Lattice</td>
<td>1200</td>
<td>.5</td>
</tr>
<tr>
<td>Material</td>
<td>1300</td>
<td>62</td>
</tr>
<tr>
<td>Tally</td>
<td>1500</td>
<td>71</td>
</tr>
</tbody>
</table>

^ Negligible

of memory available per processor is too low by several orders of magnitude. The amount of aggregate memory available is large, and indeed it would not take very many distributed nodes to accommodate the complete memory requirement. However, in order to utilize it, alternatives to pure task parallelism must be pursued to distribute the storage of material and tally data, which in general can come in one of two flavors: data decomposition and domain decomposition.

One form of data decomposition has been previously explored by Romano for tally data in [39]. In this approach, several distributed compute nodes were dedicated for tally storage, to which other particle tracking nodes would send tally data via asynchronous network operations at each scoring event. Good parallel efficiencies and minimal network overhead was observed for this approach for a number of scoping studies, indicating that it would most likely be a viable path forward for treating tally data. Recent explorations have extended this same form of data decomposition treatment to cross sections, likewise predicting that a large portion of the data can be stored off-node with tolerable overhead for a range of machine parameters [56].

It is still not clear whether or not data decomposition will adequately address material number density and data structure storage issues, which are not as amenable to off-node storage as tallies and nuclear data. For instance, off-node tally scoring

\(^{11}\)e.g., it is typical for nodes in current HPC architectures to have 16 gigabytes available per compute node: only 256 nodes would be needed for an aggregate of 4 terabytes.
can take advantage of asynchronous network operations, and large portions of cross section data are rarely accessed. However, particle tracking cannot continue without synchronous network operations, since upon entering a material number densities for all nuclides in that material are required to calculate a cross section. In addition, depletion calculations require both material and tally data, so material data is best stored where tally results are stored. These quantities naturally correspond spatially, so even though networks in HPC might be able to support a pure data decomposition scheme, a better alternative might lie in a domain decomposition scheme that does not require as much data movement.

1.7.3 Domain Decomposition

Domain decomposition holds the potential to solve both the material and tally memory problems by dividing the spatial domain among particle tracking nodes. In this approach, particle tracking nodes would only allocate memory for tallies and materials that are relevant to the domain they are assigned to, and then communicate particles to one another if the trajectory would take the particle out of the domain they are tracking. However, this holds the potential for significant network overhead as well as serious load imbalances.

Particle communication costs have been cited as a prohibitive drawback of using domain decomposition for Monte Carlo particle transport as far back as the first time it was described by Alme et al. [57] in 2001. This is echoed throughout the literature, for example in two prominent review papers by Brown in 2011 [58] and Martin in 2012 [59]. By the conventional wisdom, the extremely large number of histories that need to be tracked will create intolerable network logjams between distributed compute nodes. This was challenged by Siegel et al. in 2012 [60] with the development of a performance model that uses network parameters of current HPC machines. They demonstrated the validity of the model with a simple toy code, which indicated that network performance is likely not a limiting factor. However, it was noted that they did not account for the significant amount of bookkeeping that would affect efficiency.
of a real implementation, as well as the fact that a real implementation would require significantly more data to be transferred across the network per particle that was done in their study.

Another oft-cited drawback of domain decomposition is the potential for load imbalances, and the parallel inefficiencies it would cause. This was also explored by Siegel et al. in 2013 [61] with another performance model, which related network properties, processor speeds, and domain leakage rates to predict an upper bound on such inefficiencies. Using particle movement data tallied from non-decomposed runs of the NEA performance benchmark, they concluded that inefficiencies were tolerable for modest domain sizes, but that the method was still inherently limited for domain mesh sizes smaller than a single fuel assembly in the problem.

Another perhaps more mundane challenge with a true implementation of domain decomposition stems from the data management that must take place. For example, for arbitrary domain meshes and arbitrary CSG geometries it is neither immediately straightforward to determine nor computationally cheap to calculate which tally regions should be tracked by which processors. In addition, a significant amount of bookkeeping is required to keep track of which tally bins are allocated where, since domains will not necessarily cut tally arrays into contiguous chunks. This also complicates tally reduction algorithms, which must determine which processors and which sections of memory to accumulate scores with. Furthermore, check-pointing\textsuperscript{12} and finalization I/O is no longer an insignificant task at the scale of terabytes. All of these things must be done efficiently in order for a large-scale run to be achievable with domain decomposition.

1.8 Objectives

The over-arching objective of this thesis is to advance Monte Carlo neutron transport methods towards carrying out high-fidelity nuclear reactor analyses with the full complement of tallies needed for multi-physics coupling and nuclide depletion. In particular

\textsuperscript{12}\textit{i.e.,} saving the entire state (tally results, source fission site locations, etc.) so the simulation can stopped and restarted from this point if need be.
we are concerned with adapting the Monte Carlo algorithm for high-fidelity runs on HPC architectures, which if done correctly will help alleviate some of the previously-described issues by running many millions of particle histories in parallel.

Given the significant amount of attention variance reduction and fission source acceleration schemes have received in the literature, these issues will not be explored in detail in this work. Likewise, the cross section data treatment will not be discussed further. Instead, with memory constraints identified as one of the other key limiting factors, a more detailed exploration into domain decomposition will be pursued.

Major objectives of this thesis are:

1. Implement a robust domain decomposition scheme into a full-physics code
2. Demonstrate and model network communication burdens and load imbalance penalty prediction as the full scale of the problem
3. Identify load balancing techniques to mitigate parallel inefficiencies
4. Identify optimizations in internal handling of data structures and data management required for efficiency and scalability during simulation and I/O
5. Demonstrate the viability of carrying out simulations with full tally memory requirements

Domain decomposition will be implemented in OpenMC, and the algorithms and optimizations will be described in detail in Chapter 2. In particular, an efficient interdomain particle communication algorithm will be presented, a number of algorithms that manage the bookkeeping of data management will be developed. Of particular importance will be the maintenance of random number reproducibility throughout the implementation with a consistent treatment of fission bank sampling, tally reduction, and cross section calculation.

In chapter 3, the particle communication burden and load imbalance inefficiencies will be explored with the large-scale runs using the new implementation. The performance models developed by Siegel et al. will also be confirmed and expanded upon to allow for direct calculation of penalties, rather that just an upper bound.
In chapter 4, several load balancing strategies will be characterized and applied to the new implementation. It will be shown that not only are load imbalances for the LWR problem not as inefficient as previously predicted, but that they can be mitigated rather easily by a variety of methods.

In chapter 5, the full tally burden will be applied and the performance of the implementation demonstrated. Finalization I/O will also be discussed, along with a number of post-processing strategies and tools for managing the terabytes of data produced.

Chapters 3, 4, and 5 show results from the “Mira” Blue Gene/Q supercomputer at Argonne National Laboratory (ANL) as well as the “Titan” Cray XK7 supercomputer at Oak Ridge National Laboratory (ORNL), as well as from consumer computing clusters.
Chapter 2

Domain Decomposition

2.1 Background

The memory challenges discussed in Chapter 1 can be addressed with domain decomposition. We specifically focus on the spatial domain because that is where the largest memory burdens are incurred: spatially heterogeneous tallies and materials needed for full-core depletion. While many nuances to the concept can be conceived, the core idea of domain decomposition is easily conceptualized: distributed compute resources load materials into memory for different subsections of the problem domain, and score tally results for regions only within that domain.

The key question is whether or not this can be done efficiently with Monte Carlo particle transport methods, which is primarily a matter of implementation. Thus, as part of the present work a simple and robust domain decomposition scheme was implemented in the OpenMC neutron transport code. This implementation provides a framework to develop and evaluate a number of design choices and optimization methods with the goal of making full-core problems tractable from both a computer resource and usability perspective.

This implementation grew out of the main inter-domain particle communication logic implementation, which was developed to evaluate the effect of network communication and load imbalance penalties. That implementation addressed the major network performance and load balancing concerns outlined in [57, 58, 59] to confirm the
viability of the method, but it did not incorporate any domain-aware memory treatment. It became apparent that such an implementation for materials and tallies would come with its own performance concerns, so an on-the-fly allocation and assignment scheme was devised and implemented, including the creation of a distributed indexing scheme to support general cell tallies while maintaining simple universe geometry specifications. Since then, parallel I/O mechanisms, a domain load estimation mode, and several post-processing utilities were also developed to build up the ecosystem necessary for carrying out realistic runs using the method.

2.1.1 Review of Prior Work

The domain decomposition scheme that was added to OpenMC in this implementation was first described by Alme et al. [57]. Task parallelism was employed within each domain in this approach, where each could be replicated on any number of processes. By following heuristics based on mean-free-path estimations, the number of processes assigned to each domain could be set proportional to the distribution of work on each. An improvement in parallel efficiency was observed in this work for up to 100 processes with two simple problems decomposed into 4 domains, but the method did not scale well due to excessive time spent communicating particles.

Another implementation of domain decomposition was added to Mercury, a production Monte Carlo code developed at Lawrence Livermore National Laboratory (LLNL) [62, 63]. An adaptive load balancing algorithm was applied in this implementation, whereby processes could be dynamically reassigned between batches to track particles on different domains based on an estimation of the work performed in each domain on the previous batch. In this work a two-fold increase in performance over non-load-balanced cases was observed for small 2D problems with less than 40 processes over fewer than 20 domains. Subsequent work improved the inter-domain particle communication logic of Mercury when the code moved to general CSG geometries, by adding network routing tables [64]. In this work, bounding boxes where used to determine cell-domain membership, and robust scaling was observed. However, the problem tested was still
small (64 domains across 16 processes, with 36GB needed for problem geometry), and scalability issues were reported during initialization.

Domain decomposition has also been explored in the KULL IMC [65] and Milagro [66] thermal radiation transport codes from LLNL and Los Alamos National Laboratory (LANL), as reported by Brunner et al. [67]. Here, good performance was observed for two 3D problems where 216,000 spatial zones were divided among 256 processes. However, perfect load balancing was assumed. Subsequent work [68] improved these algorithms with asynchronous particle transfers between processes. Both strong and weak scaling studies from this work showed good performance when up to 6000 processors where evenly distributed across 270,000 spatial zones in another perfectly load-balanced toy problem, but no discussion of tally or problem memory usage was included.

More recently, the algorithm described in [68] was implemented in the Shift Monte Carlo code being developed at Oak Ridge National Laboratory [69, 70]. This implementation also includes a new scheme described by Wagner et al. [71], where domains could be defined to overlap by a significant degree to reduce inter-domain particle communication. These works demonstrated parallel efficiencies ranging from 70% to 50% over 20, 12-processor compute nodes on a consumer cluster for several 2D and 3D C5G7 [72] and CASL VERA benchmarks [73]. However, little discussion is included regarding the domain decomposition parameters employed in these studies.

A common theme of discussion in the majority of prior works is the focus on communication burdens and load imbalances arising from inter-domain particle transfer as the dominant factor holding back the expanded use of domain decomposition in Monte Carlo particle transport methods. Indeed, Brown states in [58] that domain decomposition methods are “doomed to fail” on future systems, preferring instead pure task parallelism with complete data decomposition to handle billions of spatial domains. However, more recent works (including this one), propose that the optimal configuration might be somewhere in the middle.
### 2.1.2 Recent Developments

More recent studies have suggested that the particle communication and load imbalance burdens may not be as bad as predicted in the literature for large-scale LWR problems. In particular, Siegel et al. [60] note that all prior work does not allow for any generalizations to be made regarding feasibility of using domain decomposition. As a result, they developed a general model that allows for prediction of both communication costs and load imbalance penalties from measurable problem and machine characteristics. Note that this was done for the simplest “brute force” domain decomposition approach, whereby all particles are tracked and sent to neighbor domains synchronously between stages of a single batch.

In [61, 33], Siegel et al. use the new model to further predict that these communication and load imbalance penalties are not dramatically burdensome with the particle density and movement patterns characteristic of LWR problems, as measured with full-core tallies from non-DD runs. These studies utilized coarse domain meshes (sized on the order of a PWR assembly), which should be sufficient to handle the memory requirements of the problem with current architectures.

These results motivated the present work, which in Chapter 3 attempts to confirm and expand upon the model predictions using a real DD implementation described here.

### 2.2 Current Approach

The simple ‘brute force’ DD scheme discussed in [60, 61] was implemented in OpenMC. This section attempts to describe every detail and optimization that went into this implementation to produce a code suitable for large-scale full-core runs.

#### 2.2.1 Key Definitions

The terminology used to describe domain decomposition (DD) and its performance is often not consistent throughout the literature. For clarity, the following terms are defined as they pertain to the present implementation:
• **Process** - “Process” is used to describe a distributed computing unit with its own dedicated/isolated memory space. In the performance runs of the subsequent chapters this most often refers to a single physical processor within the nodes of a cluster or supercomputing architectures, though in general a process could refer to any computing unit, such as a single core of a multi-core processor. Processes are distinct from “threads,” which share a memory space.

• **Domain/Subdomain** - “Domain” and “subdomain” are used interchangeably to refer to a single spatial portion of the global problem. A single domain encompasses the geometry, materials, and tally constructs that exist inside that spatial portion, through which particles are tracked over. Domains are distinct from “cells,” which are geometry regions defined by the model, each of which might contain a different material.

• **Communication** - “Communication” refers to the act of transferring data from one process to another. This typically refers to network operations between distributed compute nodes, which inherently include a time cost to initiate (“latency”) and to transfer data (“bandwidth”). Often we say particles are communicated to neighboring domains, which means transferred to the memory space of a process assigned to that domain, whether over the network or otherwise. OpenMC uses an Message Passing Interface (MPI) implementation [74] for all communication, where processes are grouped into “communicators” for collective operations.

### 2.2.2 Parallel Model

Fundamentally, the present approach strives to further implement the methodology put forth in [57], whereby the traditional task parallelization scheme can be used within domains. This allows additional processing units to be distributed across domains in an arbitrary fashion, which will help with subsequent load balancing efforts. The kind of adaptive re-balancing algorithm described in [62] was not pursued because of the large amount of material and tally data migration that would be required when a process switches its assignment from one domain to another, as well as the fact that the work
distribution is not expected to change much between batches for large-scale eigenvalue problems such as the one considered. With the present implementation this process distribution must be specified manually by the user, and it will remain fixed for the duration of the simulation. For proper load balancing this requires some knowledge of the particle work distribution, which for nuclear reactors is usually not difficult to tally or estimate.

All algorithms and schemes should be thought of in the context of a message-passing parallel programming model, whereby each distributed process operates independently and maintains its own memory address space. When needed, data is communicated between different processors, or “ranks.” Within a group of processes, or a “communicator group”, one process can be considered the “master” process with which all other processes must coordinate. For instance, the global master process across all ranks is typically what performs all I/O.

For the present domain decomposition implementation we also group all processes working on a single domain into another communicator group, which is a subset of the global process communicator. Therefore, one process on each domain acts as the master process local to that domain. This local master is used to coordinate inter-domain communication and collect tally results within a domain, among other things.

2.2.3 Main Loop

The present work implements domain decomposition via the 'brute force' method as described in [60] by adding particle synchronization stages during each generation of particles. Here, a particle is tracked across the geometry in a single domain until it hits a domain boundary, at which time it is buffered to be sent to the appropriate neighbor domain. Nearly the entire particle data structure is communicated: approximately 220 bytes per particle in OpenMC, depending on how many independent random number streams are used.

The changes to the outer loop of the eigenvalue iteration routine are shown in lines

---

13 This might not be true for other run modes, e.g., for time-dependent Monte Carlo implementations.
14 see Appendix A for the breakdown
Algorithm 2-1: OpenMC eigenvalue iteration outer loop with domain decomposition modifications, from the perspective of one spatial domain. Only particles in the present domain are run (the local source) at each stage. While transporting particles, if the distance to a collision $d_{\text{collision}}$ is larger than the distance to a domain boundary $d_{\text{boundary}}$, the particle is buffered for communication.

1: Guess initial fission distribution
2: for each generation of particles do
3: Sample particles from fission distribution
4: Compile local source
5: repeat
6: for each particle in the local source do
7: while Neutron not yet absorbed do
8: Sample $d_{\text{collision}}$, find $d_{\text{boundary}}$
9: if $d_{\text{collision}} > d_{\text{boundary}}$ then
10: Buffer particle for send, Break
11: end if
12: Sample physics, calc. tallies
13: end while
14: end for
15: Communicate particle buffer
16: Rebuild local source from incoming particles
17: until All particles in generation are absorbed
18: Synchronize tallies
19: Calculate eigenvalue
20: Rebuild fission source distribution
21: end for
4, 5, 9-11, and 15-17 of Algorithm 2-1 (compare to Algorithm 1-1). Unlike some of the more sophisticated methods presented in the literature, this method blocks all processes at each synchronization stage (line 15 in Algorithm 2-1), which clearly presents a potential inefficiency for load-imbalanced problems. This effect can be predicted with the performance model first described by [61] which is expanded in Chapter 3, and it is shown in Chapter 4 that it can be largely mitigated with load-balancing techniques.

2.2.4 Inter-Domain Particle Communication

The particle communication logic (line 15 of Algorithm 2-1) is shown in detail in Algorithm 2-2. It was designed for cases when the number of domains is less than or equal to the number of compute processes available, which greatly improves prospects for load balancing. In other words: more than one compute process can be assigned to work on the same domain if extra resources are available, with the particle tracking load in that domain divided evenly among them (i.e., traditional task parallelism). This is handled efficiently by conceptualizing the total number of particles entering a domain from any neighbor as belonging to an ordered array. Then, analogous to the fission bank treatment described in [37], the boundaries of slices “owned” by each process can be computed and used to buffer and send particles to the correct location. Figure 2-1 attempts to clarify this concept with an example.

It should be noted that each domain must receive information from all second-degree neighbors regarding the number of particles being sent to its direct neighbors (line 2 of Algorithm 2-2). For example, domains d0 and d1 in Figure 2-1 are not direct neighbors that communicate particles at each stage, but processes on each do need to know how many particles are being sent to their shared direct neighbor domain d3. Thus, at most each domain must communicate with a maximum of 24 domains at each stage (the 3D von Neumann neighborhood of range \( r = 2 \) - see Figure 2-2 for a 2D example). This means that the computational complexity of this algorithm is not a direct function of the number of domains. However, it does scale linearly with the number of processes on neighboring domains, which relates to the number of domains depending on the
Algorithm 2-2: Particle communication routine

1: Count no. of particles to send to each neighbor domain i: MPI_ALLREDUCE among processes in just this domain
2: Synchronize domain send info in local neighborhood (i.e. all n_send_domain(i)→domain(i))

Determine sending process starts

3: for i = 1 → 6 do
4: starts(i) = 0
5: for j = 1 → 6 do
6: starts(i) = start(i) + n_send_domain(j)→domain(i)
7: if domain(j) == my_domain then
8: Break
9: end if
10: end for
11: Find proc_offset with MPI_EXSCAN within domain
12: starts(i) = starts(i) + proc_offset
13: start(i) is now the current process start

Determine receiving process (pid) finishes

15: for i = 1 → 6 do
16: mod = modulo(n_sendall→domain(i), n_procs(i))
17: for pid = 1 → n_procs do
18: finishes(i, pid) = i × n_sendall→domain(i)
19: if i - 1 < mod then
20: finishes(i, pid) = finishes(i, pid) + i
21: else
22: finishes(i, pid) = finishes(i, pid) + mod
23: end if
24: end for
25: end for

Determine send info to each process (pid)

26: for i = 1 → 6 do
27: pid = search(starts(i), finishes(i, :))
28: for s = 1 → nbuffered_particles→i do
29: Buffer site s for sending to process pid
30: starts(i) = starts(i) + 1
31: if starts(i) > finishes(i, pid) then
32: pid = pid + 1
33: end if
34: end for
35: end for

36: Resize particle buffer and fission bank if needed
37: Send/Recv particles

Symbol Table

i, j  Local indices of first- and second-degree neighbor domains, respectively
domain(i)  Global domain index of local neighbor i
my_domain  Global domain index of the current process
n_send_domain(j)→domain(i)  No. of particles being sent to direct neighbor i from second-degree neighbor j
starts(i)  Starting index of the current process's slice in the total particle array going to domain i
finishes(i, pid)  Ending index of process pid's slice in the total particle array going to domain i
proc_offset  Starting index of the current process's slice in the particle array from my_domain only
n_procs(i)  No. of processes working on domain i
nbuffered_particles→i  No. of particles to send to domain i
load-balancing strategy employed.

### 2.2.5 Random Number Reproducibility

#### 2.2.5.1 Particle Tracking

Random number reproducibility was retained in this implementation by adding the random number seed to the particle data structure and communicating it with particles as they cross domain boundaries. New fission particles received initial seeds that were randomly sampled using the random number stream of the parent, and source sampling between successive fission generations was done using these new streams. This ensures that eigenvalue and tally results will be identical regardless of the number of domains and compute processes used: a highly desirable quality of the code for debugging and
validation purposes. It should be noted here that random number streams might now overlap, since no attempt is made to ensure that all particles start with a different random number seed.\footnote{Previously the random number seeds for each starting particle in a batch were spaced out along the stream of pseudo random numbers such that no two random numbers would be used more than once throughout the simulation.} However, this is justified in that the chance of two particles starting with the same seed with the same position, direction, and energy is vanishingly small. Thus, a user can run the same problem with different domain sizes, or only one domain, and achieve identical results.

Under this scheme several nuances of particle tracking must be accounted for to maintain perfect reproducibility. In particular, immediately after being communicated to a new domain particles must be tracked in a consistent way. For instance, particles continued on neighbor domains must use the previously-sampled distance to collision instead of a newly-sampled distance. Thus, this distance is stored in the particle data structure, communicated across domain boundaries, and used when beginning tracking of that particle on the new domain.

More interestingly, cross sections must also be calculated exactly as they were on
the previous domain, which often contains a stochastic component due to the use of probability tables in the unresolved resonance range (URR). Thus, the random number seed or seeds used to calculate these cross sections are also communicated across domain boundaries with particles. This is complicated by a desire to account for correlations that arise when treating the URR at different temperatures, where we want to use the same random number for sampling the cross section for a given nuclide (e.g., when a particle streams through multiple cells at different temperatures).\textsuperscript{16} To handle this generally, a different random number seed for each nuclide in the problem would need to be communicated across domain boundaries with every particle (for 300 nuclides, adding approximately 2 or 3 kilobytes for every particle sent). As a result, the functionality to account for these correlations was disabled for this implementation of domain decomposition, though it is not hard to add if desired.

\textbf{2.2.5.2 Starting Source}

The starting source - typically sampled from some analytical spatial distribution - also needs to be consistently chosen to retain random number reproducibility. This is not entirely straightforward, since it is not known \textit{a priori} for a given domain mesh how many particles will start in each region. This could be handled by having processes on all domains sample all particles of the starting source in the same order with the same random number stream, keeping only those that fall in their domain (i.e., rejection sampling). However, this is extremely inefficient for runs with fine domain meshes and a large number of particles per generation, so the current implementation follows a scheme where all processes sample a subset of the source using the appropriate section of the random number stream (like for the traditional parallel implementation), and then communicate particles to the appropriate domain depending on the sampled location of each. This process is described in Algorithm 2-3.

\textsuperscript{16}Some discussion of this can be found in the NJOY manual [47] and Section VI-B of [48].
Algorithm 2-3: Initial source distribution algorithm. Note that PRN seeds are stored on banked sites during initial sampling - this seed is used to select which process on destination domains to send particles to. Also, all processes store information about which processes are tracking which domains.

1: Sample $n_{\text{particles}}/n_{\text{processes}}$ particles and store in $\text{buf}$
2: Allocate 1D array $n_{\text{send\_rank}}$ to size of No. processes

Send particles

3: for particle $i$ in $\text{buf}$ do
   4:   Determine $\text{to\_domain}$ of particle $i$
   5:   if $\text{to\_domain} == \text{my\_domain}$ then
   6:     Store particle in $\text{source\_array}$
   7:   else
   8:     Randomly select $\text{to\_rank}$ from among processes tracking $\text{to\_domain}$
   9:     Post MPI_ISEND of particle $i$ to $\text{to\_rank}$
  10:    Increment counter $n_{\text{send\_rank}}(\text{to\_rank})$
  11:  end if
4: end for

Prepare to receive particles

13: MPI_ALLREDUCE $n_{\text{send\_rank}}$
14: Resize $\text{source\_array}$ if necessary

Receive particles

15: for $i = 1 \rightarrow n_{\text{send\_rank}}(\text{my\_rank})$ do
   16:   Post MPI_Irecv for new particle
   17:   Store new particle in $\text{source\_array}$
4: end for

Symbol Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Loop counter</td>
</tr>
<tr>
<td>$n_{\text{particles}}$</td>
<td>No. of particles in starting source</td>
</tr>
<tr>
<td>$n_{\text{processes}}$</td>
<td>No. of compute processes</td>
</tr>
<tr>
<td>$\text{buf}$</td>
<td>Array of sampled particles</td>
</tr>
<tr>
<td>$n_{\text{send_rank}}(:)$</td>
<td>Array of counters for No. of particles sent to each rank</td>
</tr>
<tr>
<td>$\text{my_domain, to_domain}$</td>
<td>Domain of local process and dest. domain</td>
</tr>
<tr>
<td>$\text{my_rank, to_rank}$</td>
<td>Id of local process and dest. process</td>
</tr>
<tr>
<td>$\text{source_array}$</td>
<td>Array of particles to track locally in first batch</td>
</tr>
</tbody>
</table>
2.2.5.3 Fission Bank Synchronization

Finally, this scheme requires modifications to the inter-batch particle sampling phase of eigenvalue calculations described in [37], where the starting fission source for the next batch is determined and distributed among processes. A nearly identical process is carried out amongst all processes within a single domain, rather than globally across all processes. Since fission sites are specific to a given location, they will start on the same domain they were generated on. However, whereas previously fission sites were either randomly discarded (if $k > 1$) or duplicated (if $k < 1$) in order to start each batch with the same number of particles globally, now the global population between batches is allowed to float.

This is required because fission sites could be generated on any domain, and the sampling probability is controlled by the random number seed attached to each fission site at the time of creation. Any alternative that keeps this population fixed while carrying out a site sampling scheme that maintains reproducibility will require some kind of distributed sort over all n\_particles and all compute processes, which is untenable for large-scale runs. Under the current scheme the population between batches does not increase exponentially or die out completely because the number of new fission sites is scaled by the global eigenvalue at the time of sampling during simulation. Thus, the total number of particles run in each batch will fluctuate randomly around the user input number, and the total starting weight of all particles in a batch must be globally reduced for proper calculation of tally statistics.

2.2.6 Memory Management

The details discussed in the previous sections necessitate changes in the way memory is allocated and accessed throughout the simulation. Indeed, this aspect is central to any DD implementation designed for large-scale problems, as discussed in Section 1.4. The core concept is of course that for a given process, memory only needs to be allocated when it pertains to quantities relevant to the spatial domain that process is assigned to. This primarily applies to the materials of different cells, as well as to tallies with a
A potentially significant amount of additional memory is also needed to support the machinery of carrying out domain decomposition itself. Specifically, buffer arrays are needed to support the process of sending and receiving particles between the stages of a batch, and the fission bank array needs to be managed during initial source creation as well as during simulation. In addition, a certain amount of book-keeping overhead is required to keep track of which regions different sections of memory refer to, in a distributed fashion. Lastly, it is also not trivial to determine which processes should allocate memory for which regions in the geometry, requiring either a potentially costly pre-processing step or a change in the way this memory is stored and accessed during simulation.

The current implementation attempts to address these issues while striking a balance between ease of use for the user and a minimization of computational overhead.

### 2.2.6.1 Particle and Fission Site Arrays

The current implementation described by Algorithms 2-1 and 2-2 tracks all particles in a stage before sending any to neighboring domains. An array of particle data structures is used to hold the particles buffered for communication, which needs to be at minimum the size of the number of particles run in the first stage to account for the worst case where all particles hit a domain boundary before being absorbed or leaking out of the problem. However, this size is not fixed since the number of particles is no longer held constant from batch to batch, as discussed in Section 2.2.5.3. Load imbalances will also result in processes on some domains running more particles than others. As a result, this particle buffer is implemented as a vector, which is re-sized between stages (Line 36 of Algorithm 2-2). The fission bank is treated in the same way.

If the problem is well load-balanced, then the size of these arrays will be about equal to $\frac{n_{particles}}{n_{procs}}$. This is used as the initial guess during initialization, with some extra headroom for anticipated stochastic fluctuations.

It should be noted that in the worst case these arrays could grow to encompass all $n_{particles}$ with this scheme. However, this situation would require extreme load
imbalances that are not characteristic of most realistic reactor problems (i.e., all particles would need to stream to the same domain at the same time). Regardless, the load balancing strategies discussed in Chapter 4, as well as a little common sense in specifying the problem, are expected to relieve any significant memory burdens that could arise from these particle arrays.

2.2.6.2 Distributed Indexing

One of the key optimizations required for efficiency with large problems entails storing distributed quantities in contiguous arrays that can be operated on in bulk. This applies to any quantity or data structure that is repeated for every one of the 51M fuel region in the problem as discussed in Section 1.7.2. For instance, it would not make sense to allocate space for separate cell, tally, and material data structures for each one of those regions: as indicated in Table 1.2, the meta-data alone would eat up another 150 gigabytes.

As a result, a general indexing scheme was developed by Lax et al. to easily allow for the specification of distributed materials and tallies with a single universe definition [75, 76]. In this scheme, the pincells that make up the fuel cell discretization can be defined only once, and re-used in multiple lattice positions. Then, a long array of values can be stored contiguously inside the OpenMC tally and material data structures, where each array position corresponds to a different location, or “instance,” where the cell appears.

The indexing scheme is required to efficiently access the positions in these large arrays that correspond to the cell instances particles are tracked through. For example, during transport a particle might be tracked into a new fuel cell. In order to calculate cross sections to sample the next distance to collision, the atom densities of each nuclide are required, which are stored inside a data structure for materials. Rather than have a separate material data structure for each instance of that cell, one large array of atom densities is stored in a single array inside one data structure. To access the proper atom densities for the cell instance the particle is currently in, the index can be calculated from a mapping stored on the cell data structure that is built during initialization and
stored only once. The details of map generation and use are described thoroughly in [75].

This scheme better facilitates bulk tally operations such as the calculation of statistics and intra-domain tally reduction between processes. In particular, it allows for tally arrays to be reduced in bulk: one large reduction per repeated cell type instead of a smaller reductions per unique instance of that cell. This is crucial for tolerable execution times during active batches.

2.2.6.3 On-the-fly Allocation of Materials and Tallies

The practicalities of implementing domain-specific material and tally allocation present a challenge. Specifically, with arbitrary geometries and arbitrary domain meshes it is not always straightforward to know which processes should allocate memory for which material and tally cells. This burden could be shifted to the user (i.e., requiring that domains be specified for each cell and tally in input files), but this makes mesh refinement studies difficult, especially for the kinds of high-fidelity models targeted by this work that will have millions of unique regions.

An automatic determination of cell-domain residence is preferable, but is also not straightforward to accomplish for general CSG cells defined by quadric surfaces. It can certainly be done by taking advantage of knowledge of structure in the geometry (e.g., the regular lattices in LWRs) or through various other means [77, 78] to compute bounding boxes. These methods can be costly when done for millions of cells in a high-fidelity model, and may result in the over-allocation of arrays depending on the aspect ratios and orientations of the cells (i.e., depending on the quality of the bounding box, space could be allocated on domains that don’t actually overlap with the cell).

Instead, the current implementation pursues an on-the-fly (OTF) approach, deferring the assignment of this memory until particles are actually being tracked. Under this scheme data structures for all geometry objects, materials, and tallies in the problem are allocated and initialized on all processes at the beginning of a run. Only the components of these structures that are spatially-dependent are handled in a domain-aware fashion.

\footnote{\textsuperscript{17}discussed in Section 2.2.7}
For materials this is the array that holds the atom densities of each nuclide, and for
tallies this is the results array. This arrangement is feasible because of the small amount
of memory required for each of the core data structures, as listed in Table 1.2. For
the fully-detailed BEAVRS benchmark, the entire non-domain-dependent geometry and
materials memory requirement in OpenMC is on the order of a megabyte.

The domain-aware arrays are initially allocated like the particle buffer and fission
bank arrays as discussed in Section 2.2.6.1, where perfect load balance is assumed
and space is allocated for $global\_size/n\_domains$ items, plus some headroom. Here,$global\_size$ refers to the size of these arrays in the equivalent non-DD run, and an item
refers to either a set of atom densities for one material\textsuperscript{18} or the set of accumulators for
one tally result bin.

Space is used in these arrays on-the-fly as particles are tracked into regions that
require their use. For example, under this scheme the number densities for a given
material (a “composition”) are only read into memory from the input file when a
particle enters that material and a distance to collision needs to be sampled. The
newly-read composition is stored in the next available slot of the atom densities array
for that material, and the index for that composition is recorded in a mapping dictionary
for when future particle tracking requires the same data. If more slots are needed,
additional space is allocated. The tally OTF treatment is analogous - Figure 2-3 attempts
to demonstrate the concept with an example.

The primary costs of this OTF treatment lie in increased memory and time during
simulation to store and process maps, increased time to reduce tallies, and increased
time to write tally results. In aggregate, these penalties are found in Chapter 5 to be
tolerable for practical LWR problems.

\subsection*{2.2.7 Tally Reduction}

As discussed in [33, 38] and Section 1.6.2.2, tally reduction is required to calculate
statistics, either between batches or once at the end of the simulation. Either way,
\textsuperscript{18}This implementation requires the use of distributed materials as described in Section 2.2.6.2, where
one material object contains the number densities for a large number of distributed cells.
Figure 2-3: Schematic example of OTF memory assignment during simulation, for tallies. Top: Results array for the non-OTF case for an example geometry, showing 6 TallyResult filter bins: 3 cells and two energy groups. Bottom: Progression of OTF array assignment, (from left to right). As particles cross cells tallies are scored to the next available bin in the results array, and a map to the global array index is stored. In this example, if another particle were to cross cell 1 in energy group 1, the map would be used to score to the third position of the results array, since that bin has already been assigned.
reduction entails summing the contributions of independent realizations on all processes and combining the results in a single location at the end. For instance, with the traditional non-DD implementation all processes in the simulation reduce their tally accumulators to the global master process.

The present domain-decomposed implementation treats the tally reduction step in a manner similar to how the fission bank synchronization step is treated, as discussed in Section 2.2.5.3. Here, since each domain can be replicated among multiple processes reduction is carried out almost exactly like the non-DD case, but only among groups of processes assigned to a single domain. This is easily accomplished with the same routines already in place by using a different MPI communicator object that groups these processes. Under this scheme, each domain reduces their tally accumulators to the local domain master process, with no inter-domain communication.

There are a few caveats to this process. As before, eigenvalue tallies still need to be globally reduced so that the number of fission sites generated is properly scaled, as discussed in Section 2.2.5.3. In addition, for proper calculation of statistics the total starting weight of all particles participating in this realization needs to be globally reduced, since it is no longer fixed between batches. Finally special treatment is needed to handle the reduction of OTF tallies.

### 2.2.7.1 Reduction of OTF tallies

As one of the dominant consumers of memory in large-scale runs, the tally reduction step can be a major point of inefficiency if done improperly. Specifically, operations on tally arrays should be carried out in a vectorizable fashion, whereby the same operation is carried out on all elements of the results array at the same time. Loops over each individual scoring bin should be avoided. This is particularly true during the tally reduction step, when the entire results array must be communicated over the network to other distributed processes. For tallies that were allocated in a non-OTF fashion it is straightforward to reduce among a group of processes, since the results arrays are all allocated to the same size and scoring bins are assigned in the same order - i.e., we can communicate the entire results array as one continuous block. This is not so
straightforward for OTF tallies, since the scoring bins in the results arrays for each are unlikely to have been assigned in the same order on each process that tracks a domain.

In order to avoid sending items of the results array one-by-one, under the current implementation each process re-orders the results arrays of each tally before sending values to the local domain master process. This enables the same tally reduction routines to be used as before, ensuring efficient network performance. As described in Algorithm 2-4, this is accomplished by first broadcasting the OTF score bin mapping from the local master process to each other process in the domain, which will carry out the re-ordering locally. Figure 2-4 depicts an example.

Depending on the size of the tally arrays, the re-ordering on Line 24 of Algorithm 2-4 can be an expensive process. This is implemented as an in-place exchange sort with memory usage complexity $O(1)$, akin to a heap sort but where the heap is replaced by the OTF score maps. The score maps are implemented as a hash map, so the time complexity to find the next element is on average $O(1)$ and at worst $O(n)$, depending on the quality of the hashing. Thus, on average the time complexity of this sort will be $O(n)$, and $O(n^2)$ in the worst case.

Regardless, for most cases this re-ordering only needs to be done once, since after the first tally reduction operation the results arrays will be in a consistent order across all processes on a given domain. As long as enough particles were run such that all scoring bins were accessed at least once on each process, the sort will not need to be repeated for subsequent reductions. The current implementation checks the score bin mapping between every batch to determine if the reordering needs to take place - a potential optimization could disable this check after the first active batch if the user is confident that all bins were allocated.

2.2.7.2 Limitations

The current implementation works well when tally bins are confined to one domain only, since no inter-domain tally reduction is being carried out (with the exception of global eigenvalue tallies). However, complications will arise if the geometry and tallies are specified such that the same scoring bins appear on more than one domain, resulting in
Algorithm 2-4: OTF tally reduction algorithm for a single tally. This ensures that all results array bins seen by any process on this domain get allocated and assigned on every other process, and in the same order.

1: Set up MPI communicator for processes on the current domain

Collect OTF bin mappings

2: MPI_REDUCE n_result_bins to domain master with MPI_MAX operation
3: if domain_master then
4: Allocate 2D array proc_bin_maps to size n_result_bins by n_domain_procs
5: for i = 1 → n_domain_procs do
6: Post MPI_Irecv from process i for bin map, store in proc_bin_maps(:,i)
7: end for
8: else
9: Allocate 2D array proc_bin_maps to size n_result_bins by 1
10: Post MPI_IRECV to domain master with bin map
11: end if

Update OTF bin mappings

12: if domain_master then
13: for i = 1 → n_domain_procs do
14: for bin in proc_bin_maps(:,i) do
15: Assign position for bin in local results array if not present
16: end for
17: Post MPI_ISEND to process i with new bin map
18: end for
19: else
20: Post MPI_Irecv from domain_master for bin map, store in proc_bin_maps(:,1)
21: for bin in proc_bin_maps(:,1) do
22: Assign position for bin in local results array if not present
23: end for
24: Reorder results array to match proc_bin_maps(:,1)
25: end if

Reduce tallies

26: MPI_REDUCE results arrays to domain_master with MPI_SUM operation

Symbol Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Loop counter for process on a domain</td>
</tr>
<tr>
<td>domain_master</td>
<td>Whether or not the current process is the domain master</td>
</tr>
<tr>
<td>n_domain_procs</td>
<td>No. of processes assigned to a domain</td>
</tr>
<tr>
<td>n_result_bins</td>
<td>No. of tally results bins assigned to local results array</td>
</tr>
<tr>
<td>proc_bin_maps</td>
<td>Buffer to store results array bin mappings</td>
</tr>
<tr>
<td>bin</td>
<td>Results array global index</td>
</tr>
</tbody>
</table>
OTF Results Array Re-ordering

Collect Maps on Master

Allocate and Reorder

Figure 2-4: Schematic example of OTF tally results array reordering before a reduction, using the same scenario described in Figure 2-3.
Figure 2-5: Example showing a tally bin (dashed diamond) cut by domain boundaries between four domains. Tally reduction for this bin is not supported by the current implementation. Instead each section that lies on a different domain (each of the colors) will be treated independently, as if it was a separate tally bin.

“cut bins” that are cut by domain boundaries. An example is depicted in Figure 2-5.

The current OTF allocation scheme will function for cut bins, but the results for the bin on different domains should be understood as different independent realizations of that bin. Or rather, in this case the cut bin is implicitly defining multiple different bins: the results cannot be combined. This may not be so bad for some situations, but in practical scenarios it still presents a challenge. For instance, it would be easy for a domain boundary to cut near the edge of cell, implicitly producing “slivers,” or cells with high aspect ratios and cells that might not see many particle tracks (leading to terrible statistics).

In addition, in many cases the volumes of each tally region need to be known for anything practical to be done with the results. The presence of cut cells makes this difficult, since any volumes would need to be calculated either with another tally (prone to poor statistics for slivers) or via analytical or heuristic methods (potentially expensive).

Even though it is possible to overcome these difficulties to use cut bins, the current implementation does not robustly support them, though the functionality may be added in future work. However, it is unclear whether or not that is the best direction to go. This is especially true considering that different tally results in each bin section would also require the creation of different material sections. This could lead to undesirable artifacts in material and temperature gradients down the line as materials are depleted.

\[19\] only by failing to write out the different sections separately, as discussed in Section 2.2.8.1.
according to tallies that depend on the domain mesh.

Alternatively, cut bins could be treated with some further reduction, employing inter-domain communication to fully combine the accumulators for each cut section. A scheme for doing this can be readily conceived of, whereby after a reduction using the current implementation domain master processes perform another round of reductions amongst themselves. However, unless the results array structure is significantly modified this would require individual bins in the results arrays to be treated one-at-a-time, since in principle the master copy for each bin could reside on a different process: block treatment of tally arrays would not be possible. This is expected to perform poorly for very large tally array sizes.

It may be likely that the tally server approach described in [39, 79] is preferable when dealing with cut cells. However, before jumping to such an implementation the practicalities of carrying out a full-core depletion calculation should be weighed. Since such calculations also require spatially-dependent materials data, it is still unclear how well the data decomposition concept would perform (on which processes are the Bateman Equations solved for each depletion region? Can it be done without needing to communicate all atom densities or tally results over the network?). Perhaps the optimal implementation will be found with some hybrid scheme.

For the present work, these limitations are moot as long as no tally regions are cut by domain boundaries.

### 2.2.8 Inputs and Outputs

Input and output (I/O) file specification might be regarded as one of the more mundane aspects of any simulation method, but it does deserve some attention for large problems such as those discussed in Section 1.5. In particular, two aspects need to be considered in the context of the present DD scheme: material inputs/outputs and tally result outputs. This also has implications for any post-processing tasks or multi-physics coupling schemes.
2.2.8.1 OTF Materials and Tallies

The current implementation has two options for outputting materials and tallies: (1) a user-friendly option that aggregates results from all domains; and (2) a mode that maximizes performance.

The first mode attempts to be as user-friendly as possible by using a single HDF5 file for all materials, and by incorporating tally results into single statepoint files. The arrays in these files are in order, which requires using the bin maps to unscramble the arrays that were allocated and assigned in an OTF fashion. This can incur significant expense for data on the order of Terabytes since it necessitates individual treatment of tally filter bins, even when done in a distributed fashion. In particular, to maximize filesystem write speeds the OTF arrays need to be sorted in memory in the order they will be written to the file\(^{20}\) so one large collective MPI write operation can be carried out. This is implemented using a standard heapsort algorithm which is \(O(1)\) in memory and \(O(n \log(n))\) in time.

Also, the present implementation does not robustly handle tally bins cut by domain boundaries, as discussed in Section 2.2.7.2. During statepoint creation each domain master writes each bin that has been allocated locally to the appropriate section of the global array in the file using parallel I/O. When more than one domain master contains the same bin, each will write to the same location, and it is ambiguous which one will be written last.

Unfortunately, the aggregated “single-file” output mode does not perform well enough to be suitable for large-scale I/O. This was tested extensively on the Lustre filesystem on Titan, where write speeds higher than 200 MB/s were not achieved.\(^{21}\) This is consistent with several recent investigations of parallel HDF5 performance on these filesystems [80, 81]. However, it may be possible to achieve higher performance if better data chunking and caching parameters are chosen to align with filesystem server striping [81, 82].

As a result of these difficulties, a second I/O mode was implemented where processes

\(^{20}\)This is the same order dictated by the indexing scheme discussed in Section 2.2.6.2.

\(^{21}\)This would result in several hours to write tally results to disk, per statepoint.
on each domain dump their portions of the OTF materials and tally results arrays to a separate file, along with the OTF index mapping discussed in Section 2.2.6.3. In this mode the Lustre filesystem performs admirably, handling over 2 terabytes of file I/O from 1000 MPI processes in under 20 seconds. This is encouraging, since it means the OTF aggregation and sorting algorithms are not prohibitive operations, meaning in-memory multi-physics coupling should not be difficult to achieve.

One advantage of this alternate implementation is unambiguous treatment of cut cells: when separate results are dumped for each domain, the different sections of cut cells will be dumped separately. These results are usable as long as the volumes for each section can be calculated or tallied, but the potential for the creation of slivers remains.

Of course, this “file-per-process” output mode adds a potentially significant burden to the user during post-processing, where OTF index maps must be used to extract data for specific cell instances. Functionality to accomplish this was added to the set of data processing utilities that ship with OpenMC, but it will likely require optimization for efficiency with large-scale results.

2.2.8.2 Limitations

Currently only HDF5 files are supported for DD-related I/O. This vastly simplifies the process of reading and writing to the files in a distributed fashion. Also, as mentioned in Section 2.2.6.3, only regions using distribcell indexing are handled in a domain-aware fashion. This means that any post-processing or coupling schemes will need to be aware of how to construct distribcell maps, both to specify individual material compositions and retrieve tally results for different spatial cells.

2.3 Conclusion

A practical implementation of domain decomposition for Monte Carlo neutron transport codes is described, suitable for full-core nuclear reactor analysis. Random number reproducibility is maintained, and the memory treatment is described in detail. Further chapters analyze and demonstrate its performance on different machines as the problem
size scales up.

This implementation contains several key features:

- Domains are defined with an arbitrary mesh, overlaid on an arbitrary geometry.
- There is no need to specify which domains contain material and tally regions.
- Simple universe-based problem descriptions can be used.
- Additional compute resources can be distributed across domains according to the work distribution.\textsuperscript{22}
- Random number reproducibility is maintained regardless of the number of domains and/or processes that are used.

This implementation also has several key limitations:

- Tally cells must reside in one domain each (i.e., cells may not be cut by domain boundaries if tallies are applied to them).
- Efficient material handling relies on the use of distributed cells, as described in Section 2.2.6.2.
- Effective load balancing requires knowledge of particle work distributions.
- Each process can only track particles on one domain.

Subsequent chapters explore the performance of this implementation, both from a communication and load balancing perspective as well as overall ability to tackle the full-scale problem of interest.

\textsuperscript{22}This significantly addresses load imbalances, as discussed in Chapter 4.
Chapter 3

Particle Communication Overhead and Load Imbalances

3.1 Introduction

Particles move, and therefore any Monte Carlo domain decomposition implementation will need to communicate particles between spatial domains as they cross domain boundaries. When distributed computing resources track particles on different domains, this communication incurs overhead. In addition, the uneven spatial distribution of particles introduces parallel inefficiencies due to load imbalances.

In this chapter the domain decomposition implementation detailed in Chapter 2 is tested for communication performance. In addition, an updated performance model is presented to quantitatively understand both communication and load imbalance inefficiencies in terms of machine and problem characteristics. With real timing data, the analytical model predictions can be empirically validated.

The analysis and results of this chapter are derived from a paper published in *Parallel Computing* [83].
3.1.1 Communication Overhead

In light of the many billions of particle histories required to converge source distributions and fine-mesh tallies for LWR problems, it has been widely speculated that such inter-domain particle communication would make Monte Carlo domain decomposition prohibitively computationally expensive for real problems [57, 58, 59]. To some extent, this has been the motivation behind much of the prior work to refine inter-domain communication algorithms in [62, 68]. However, as shown by the more recent theoretical work by Siegel et al. in [60, 61] and confirmed here, the situation appears more favorable than previously thought.

The overhead for communicating particles between domains comes from two places: (1) network considerations, and (2) algorithmic considerations. The network component is driven by the number of connections needed and the volume of data to be communicated (i.e., the number of particles and the size of the particle data structures). The intrinsic bandwidth and latency of the links between distributed computing resources then determine the absolute magnitude of the overhead. As discussed in Section 1.3, networks are exceedingly fast and becoming faster. Therefore even though the amount of network traffic is high, any resulting inefficiencies should be a relatively small part of the overhead.

The algorithmic component of the particle communication overhead relates to the additional machinery of domain decomposition needed to send/receive particles. For example, the methods discussed in Section 2.2 describe how particles are buffered on both the sending and receiving sides, how only a subset of the full particle data structure is actually communicated between domains, and how meta-data is first communicated between domains before sending/receiving particles. All of these operations incur some overhead, but can be done very efficiently as shown in Section 3.3.

The actual quantity of data that needs to be communicated over the network is problem dependent; Siegel et al. were the first to quantitatively study the network traffic that would be encountered for domain-decomposed LWR problems [61, 33].

---

23Configuration nuances that affect those machine characteristics can also play a role, e.g., network topology.
Figure 3-1: Reproduction of data from [61], showing average domain leakage fractions for the NEA performance benchmark with several mesh sizes. The single assembly, quarter-assembly, and ninth-assembly radial meshes used 20, 40, and 60 axial levels, respectively.

They tallied the fraction of particles that would leak out of several domain meshes for the NEA benchmark, using a non-domain-decomposed version of OpenMC that simply tracked particle movement. As expected, they found that leakage fractions increase as domain sizes are shrunk. Figure 3-1 is a reproduction of their data. The distribution for the BEAVRS problem is analogous, though coarser meshes were used in this work. For instance, Figure 3-2 shows the distribution of axially-integrated leakage data for a 3x3-assembly mesh (i.e., squares 64.51092cm per side, or 3x the BEAVRS fuel assembly pitch).

### 3.1.2 Load Imbalances

Parallel inefficiencies arise when compute resources that track different spatial domains have an uneven amount of work to do - this is distinct from the overhead specific to particle communication. For example with LWR problems, domains on the periphery will have fewer particles to track in the reflector regions than domains that track particles
Figure 3-2: Radial view of the BEAVRS geometry showing the tallied distribution of axially-integrated particle leakage fractions on a 7x7 domain mesh for the first particle synchronization stage. The highest leakage fractions tend be near reflector regions, whereas near the center of the core leakage fractions are highly dependent on the size of domains (see Figure ??). *Few particles are tracked to the corners, so leakage tallies in these regions are not reliable due to poor statistics.
Figure 3-3: Radial view of the BEAVRS geometry showing the tallied distribution of work on a 7x7 domain mesh, axially-integrated. Regions around the periphery track fewer particles and thus processes will have less work to do (green), while regions near the center of the core will have more work to do (red).

through fuel assemblies. With the simple scheme described in Section 2.2, processes that track particles through these domains will sit idle between each particle synchronization stage. As a result, simulations will take longer than the equivalent run using the same computer resources with the traditional parallelization scheme.

Figure 3-3 shows an approximation of the work distribution with a coarse mesh over the BEAVRS LWR problem, as tallied from collision and cell boundary crossing events during particle tracking. This data can be used in performance models, such as the one established by Siegel et al. in [60, 61] to describe load imbalance slowdowns. This model will be expanded upon in Section 3.2, and several load balancing strategies will
be discussed in Chapter 4.

3.1.3 Prior Model Development

An initial study by Siegel et al. [60] modeled communication costs for inter-domain particle transport in terms of the latency and bandwidth properties of the network as well as particle leakage rates of each domain. Here, they defined a dimensionless partition size to describe cubic domains, and used the Wigner rational approximation [84] with a uniform source in each domain to approximate domain leakage rates. A Taylor expansion of the model then allowed them to solve for an optimal domain size that minimizes communication costs, expressed in terms of the global number of starting particles. Parameters for several scaling regimes were explored, and communications overhead of between 1-2% of total run time was predicted for domains of moderate size and small particle counts.\(^{24}\)

However, the framework established in [60] did not adequately account for both the changes in leakage rates with different particle synchronization stages (see Figure 3-1) as well as the spatial variation of starting source counts and leakage rates among domains. Indeed, these variations are precisely what cause load imbalances. As a result, Siegel et al. and Romano further developed a model that takes into account these variations [33, 61]. Using continuous approximations for large sums over leakage terms, and by assuming worst-case leakage patterns, they were able to describe load imbalance penalties solely in terms of initial particle distributions, arriving at an approximate upper bound. Using data tallied from the NEA performance benchmark, Siegel et al. then predicted reasonable performance for LWR problems with assembly-sized domains and coarser. However, the upper-bound model predicted a rapid increase in load imbalance inefficiencies as radial domain meshes were refined further.

The theoretical framework established in [33, 60, 61] for understanding inter-domain particle communication overhead and load imbalance slowdowns is relevant to the domain-decomposition implementation described in Section 2.2. However, the

\(^{24}\)Predictions were made using network parameters of the IBM Blue Gene/P architecture.
previous predictions can be improved upon if some of the simplifications to the model are relaxed. In particular, the upper-bound predictions of [61] are most likely an over-prediction, as will be shown in the following sections.

3.2 Analysis

To support the present analysis, the performance models first established in [60, 61, 33] are re-derived to a slightly different form. In particular, parameters in the final equations can be arranged in a way that is more amenable to making direct predictions of load imbalance penalties from tallied domain leakage data without resorting to an upper-bound only.

The derivation is logically broken up into two parts: (1) a perfectly-load balanced case where starting sources and leakage rates are constant in all domains; and (2) a case with uneven particle sources and leakage rates across domains. By comparing the two cases, an estimate for the inefficiencies due to load imbalances can be studied.

3.2.1 Definitions and Assumptions

All analysis that follows pertains a single generation of particles run across \( N \) domains. The present algorithm requires \( M \) communication stages to synchronize particles across all domains until all particle histories have been terminated and no more particles need to be communicated. After each stage \( i \), the number of particles that move out of domain \( j \) is equal to the number of particles that were run in that domain \( (p_{ij}) \) multiplied by the leakage out of that domain in that stage \( (\lambda_{ij}) \).

Thus, the global average number of particles run in each domain in a stage is

\[
\bar{P}_i \equiv \frac{1}{N} \sum_{j=0}^{N-1} p_{ij}.
\] (3.1)

We care to estimate the load imbalance penalty of a simulation, defined as

\[
\Delta \equiv \frac{\tau' - \tau}{\tau}.
\] (3.2)
where \( \tau' \) is the run time of a domain-decomposed simulation and \( \tau \) is the run time of that same simulation if the load was perfectly balanced across domains. This load balance \( \Gamma \) can be defined for each stage \( i \) as the relative difference between the average particle count and the maximum particle count among domains, i.e.

\[
\Gamma_i \equiv \frac{\bar{p}_i}{p_{i}^{\text{max}}}
\]

where \( p_{i}^{\text{max}} \) denotes the number of particles run during stage \( i \) in the domain that ran the most.

Note that this implicitly assumes that further analysis will use particle counts to describe load imbalances, rather than some more general measure of “work” that needs to be done in each domain. This is fine for most cases, but may not be adequate for cases where the number of particles run in a domain does not correlate to the amount of work to be done in a manner consistent across all domains.\(^{25}\) In other words, this assumes a constant particle tracking rate in all domains and at each stage.

Subsequent derivations will describe run times in terms of two components that distinguish between particle tracking and communications, i.e.:

\[
\tau = \tau_{\text{tracking}} + \tau_{\text{communication}}
\]

The communication time is further decomposed into latency and bandwidth components:

\[
\tau_{\text{communication}} = \tau_{\text{latency}} + \tau_{\text{bandwidth}}
\]

These times will be expressed in terms of constants that describe the particle tracking rate

\[
\mu[=\frac{\text{seconds}}{\text{particle tracked}}]
\]

as well as the network latency.

\(^{25}\text{e.g., if geometrical complexity is widely varying, and thus cell border crossing event counts are not consistent across domains, particle density alone will not adequately describe load imbalances.}\)
\[
\alpha \left[ \frac{\text{seconds}}{\text{connection established}} \right]
\]

and network bandwidth

\[
\beta \left[ \frac{\text{seconds}}{\text{particle transferred}} \right].
\]

This analysis will also assume rectilinear domain meshes, where each domain can only transfer particles to up to six direct neighbor domains.

### 3.2.2 Perfect Load Balance

We first examine the case where domains are chosen such that the load is perfectly balanced across them. In other words,

\[
p_{ij} = \bar{p}_i \forall j.
\]

### 3.2.2.1 Constant Isotropic Leakage From All Domains

We start by assuming that in this ideal case particles leak out of all domains isotropically then begin by writing each component of run time \( \tau \) in terms of particle densities and leakages.

First, since each domain needs to transfer particles to six direct neighbors once per stage, the latency term can be written in terms of the number of stages \( M \):

\[
\tau_{\text{latency}} = 6\alpha M.
\]

As shown in \([33, 61]\), the number of stages can be further written in terms of the geometric mean of domain leakages, but this is not required for the present analysis. Indeed, any diagnostic tool that uses the resulting equations from this model should already have leakage rates tallied on a per-stage basis, meaning the number of stages should already be known.

Next, the particle tracking term is written as a sum over the number of particles run
in each stage, which in this case is constant for all domains. For this we can expand the sums by taking advantage of the fact that for spatially-constant $\lambda$ and $p_{ij}$, the number of particles run in a domain at each stage is a simple combination of the fraction of particles that leak in from the 6 direct neighbors, i.e.

$$p_{i+1} = \lambda_{ix} + p_{ix} + \lambda_{ix} - p_{ix} + \lambda_{iy} + p_{iy} + \lambda_{iy} - p_{iy} + \lambda_{iz} + p_{iz} + \lambda_{iz} - p_{iz}$$

(3.11)

where the $xyz$ subscripts indicate the Cartesian neighbors in each direction. Since in this case the loads and $\lambda$'s are spatially constant and leakage is isotropic,

$$p_{i+1} = \frac{\lambda_i}{6} p_{ix} + \frac{\lambda_i}{6} p_{ix} - \frac{\lambda_i}{6} p_{iy} + \frac{\lambda_i}{6} p_{iy} - \frac{\lambda_i}{6} p_{iz} + \frac{\lambda_i}{6} p_{iz}$$

(3.12)

This assumes a grid of domains large enough such that we can ignore edge effects. With this expression, we can write the tracking term as

$$\tau_{tracking} = \mu \sum_{i=0}^{M-1} \tilde{P}_i$$

(3.13)

$$= \mu \left( \tilde{P}_0 + \tilde{P}_1 + ... + \tilde{P}_{M-1} \right)$$

$$= \mu \left( \tilde{P}_0 + \lambda_0 \tilde{P}_0 + \lambda_0 \lambda_1 \tilde{P}_0 + ... + \lambda_0 \lambda_1 ... \lambda_{M-2} \tilde{P}_0 \right)$$

$$= \mu \tilde{P}_0 \left( 1 + \lambda_0 + \lambda_0 \lambda_1 + ... + \lambda_0 \lambda_1 ... \lambda_{M-2} \right)$$

$$= \mu \tilde{P}_0 \left( 1 + \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k \right)$$

with

$$\|\lambda\| = \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k.$$  

(3.14)
Here it is understood that that last term in the sum is zero (i.e. \( \lambda_{M-1} = 0 \), no particles are scattered out of the last stage), but we leave the sum to \( M - 1 \) for convenience. Note that the \( \|\lambda\| \) term here is not directly comparable to the same term as defined in [61]. However, this definition allows for leakage data to be more easily utilized when evaluating the model.

The bandwidth term can be treated in a similar way, where we write the time it takes to transfer particles between domains in terms of the number of particles that need to be transferred between domains at each stage:

\[
\tau_{\text{bandwidth}} = \beta \sum_{i=0}^{M-1} \lambda_i \tilde{P}_i
\]

\[
= \beta \left( \lambda_0 \tilde{P}_0 + \lambda_1 \tilde{P}_1 + \ldots + \lambda_{M-1} \tilde{P}_{M-1} \right)
\]

\[
= \beta \left( \lambda_0 \tilde{P}_0 + \lambda_0 \lambda_1 \tilde{P}_0 + \ldots + \lambda_0 \lambda_1 \ldots \lambda_{M-1} \tilde{P}_0 \right)
\]

\[
= \beta \tilde{P}_0 \left( \lambda_0 + \lambda_0 \lambda_1 + \ldots + \lambda_0 \lambda_1 \ldots \lambda_{M-1} \right)
\]

\[
= \beta \tilde{P}_0 \prod_{i=0}^{M-1} \lambda_k
\]

Combining Equations 3.10, 3.13, and 3.15, we arrive at an expression for the time it should take a perfectly load-balanced simulation:

\[
\tau = 6\alpha M + \mu \tilde{P}_0 \left( 1 + \|\lambda\| \right) + \beta \tilde{P}_0 \|\lambda\|
\]  

(3.16)

3.2.2.2 Variable Isotropic Leakage

Now we examine the case where leakage rates are allowed to vary across domains. In this scenario it is useful to define an average leakage quantity

\[
\tilde{\lambda}_j = \sum_{j=0}^{N-1} \lambda_{ij} P_{ij} / P_i
\]  

(3.17)

where \( P_i \) is the total number of particles per stage and \( \lambda_{ij} \) is the total leakage out of domain \( j \) in stage \( i \). This choice of \( \tilde{\lambda}_j \) preserves the total number of stages needed to
finish a cycle,

\[
\frac{\hat{P}_{i+1}}{P_i} = \frac{1}{N} \sum_{j=0}^{N-1} P_{i+1,j} = \frac{\sum_{j=0}^{N-1} \lambda_ij P_{ij}}{\sum_{j=0}^{N-1} P_{ij}} = \frac{\sum_{j=0}^{N-1} \lambda_ij P_{ij}}{\sum_{j=0}^{N-1} P_{ij}} = \frac{\lambda_i}{P_i},
\]

and allows us to write an estimate for what the initially perfectly load-balanced time \( \tau \) would be for the variable leakage case:

\[
\tau = 6\alpha M + \mu \bar{P}_0 \left( 1 + \|\hat{\lambda}\| \right) + \beta \bar{P}_0 \|\hat{\lambda}\| \tag{3.19}
\]

for

\[
\|\hat{\lambda}\| = \sum_{i=0}^{M-1} \prod_{k=0}^{i} \hat{\lambda}_k \tag{3.20}
\]

Note that conceptually this doesn’t truly represent the same perfectly load-balanced timing as described by Equation 3.16, as variable leakage rates can be expected to introduce imbalances as stages progress. Thus, this expression for \( \tau \) really only refers to perfect load balance in the first stage of the cycle.

### 3.2.3 Uneven Load Balance

Next we examine the case where loads are not evenly balanced across all domains. Since the algorithm described in Section 2.2 blocks processes within a stage until all domains have finished tracking particles, the time for particle tracking will be determined by the domain that takes the longest to finish. As a result it is also useful to define the maximum particle count

\[
P_i^{\text{max}} = \max\{p_{ij} : 0 \leq j \leq N - 1\} \tag{3.21}
\]

and leakage

\[
\lambda_i^{\text{max}} = \max\{\lambda_{ij} : 0 \leq j \leq N - 1\}, \tag{3.22}
\]

as well as the maximum count of particles transferred between domains.
\[(\lambda_i p_i)^{max} = \max\{\lambda_{ij}p_{ij} : 0 \leq j \leq N - 1\}, \quad (3.23)\]

noting that these maximums need not necessarily arise from the same domains.

As before, we can write each of the run time components as sums over the stages, arriving at an expression for the time \(\tau'\) for the load imbalanced case:

\[
\tau' = 6\alpha M + \beta \sum_{i=0}^{M-1} (\lambda_i p_i)^{max} + \mu \sum_{i=0}^{M-1} p_i^{max} \quad (3.24)
\]

Equation 3.24 is not easily expanded as in the perfectly load-balanced case. However, an upper-bound can be established as was done in [61] if a spatially isotropic leakage is assumed equal to the maximum leakage for each stage. As will be shown later, this is not really needed for making performance predictions. However, we will take the present formulation through this process as well in order to compare to previous results and show equivalency.

First, in order to compare with the perfectly load-balanced case we write the number of particles run on each domain \(j\) on stage \(i\) as a deviation from the average \(\delta p_{ij}\):

\[
p_{ij} = \bar{p}_i + \delta p_{ij}. \quad (3.25)
\]

Replacing this into Equation 3.24 with the previous new assumption, we arrive at

\[
\tau' = 6\alpha M + \beta \sum_{i=0}^{M-1} \lambda_i^{max} (\bar{p}_i + \delta p_i^{max}) + \mu \sum_{i=0}^{M-1} (\bar{p}_i + \delta p_i^{max}) \quad (3.26)
\]

\[
\tau' = \tau + \beta \sum_{i=0}^{M-1} \lambda_i^{max} \delta p_i^{max} + \mu \sum_{i=0}^{M-1} \delta p_i^{max}.
\]

Next, to calculate an upper bound the worst case leakage pattern is used, i.e.
\[ p_{i+1} = \lambda_{ix} \left( \tilde{p}_i + \delta p_{ix+} \right) + \lambda_{ix} \left( \tilde{p}_i + \delta p_{ix-} \right) + \lambda_{iy} \left( \tilde{p}_i + \delta p_{iy+} \right) + \lambda_{iy} \left( \tilde{p}_i + \delta p_{iy-} \right) + \lambda_{iz} \left( \tilde{p}_i + \delta p_{iz+} \right) + \lambda_{iz} \left( \tilde{p}_i + \delta p_{iz-} \right) \]
\[ p_{i+1} = \frac{\lambda_{ix}^\max}{6} \left( \tilde{p}_i + \delta p_{ix+} \right) + \frac{\lambda_{ix}^\max}{6} \left( \tilde{p}_i + \delta p_{ix-} \right) + \frac{\lambda_{iy}^\max}{6} \left( \tilde{p}_i + \delta p_{iy+} \right) + \frac{\lambda_{iy}^\max}{6} \left( \tilde{p}_i + \delta p_{iy-} \right) + \frac{\lambda_{iz}^\max}{6} \left( \tilde{p}_i + \delta p_{iz+} \right) + \frac{\lambda_{iz}^\max}{6} \left( \tilde{p}_i + \delta p_{iz-} \right) \] (3.27)

\[ p_{i+1} = \lambda_{i}^\max \tilde{p}_i + \frac{\lambda_{i}^\max}{6} \left( \delta p_{ix+} + \delta p_{ix-} + \delta p_{iy+} + \delta p_{iy-} + \delta p_{iz+} + \delta p_{iz-} \right) \]

\[ p_{i+1} \leq \lambda_{i}^\max p_{i}^\max \]

\[ p_{i+1} \leq \lambda_{i}^\max p_{i}^\max \]

In other words, the worst case for any stage would be if all six neighbors surrounding a domain had the maximum load imbalance. With this bound, we can write the sums in Equation 3.26 as

\[ \sum_{i=0}^{M-1} \delta p_{i}^{\max} = \delta p_{0}^{\max} + \delta p_{1}^{\max} + \delta p_{2}^{\max} + ... + \delta p_{M-1}^{\max} \]
\[ \leq \delta p_{0}^{\max} + \lambda_{0}^{\max} \delta p_{0}^{\max} + \lambda_{1}^{\max} \delta p_{0}^{\max} + ... + \lambda_{0}^{\max} \lambda_{1}^{\max} \delta p_{0}^{\max} ... \lambda_{M-2}^{\max} \delta p_{0}^{\max} \]
\[ \leq \delta p_{0}^{\max} \left( 1 + \lambda_{0}^{\max} \lambda_{1}^{\max} ... \lambda_{M-2}^{\max} \right) \]
\[ \leq \delta p_{0}^{\max} \left( 1 + \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_{k}^{\max} \right) \] (3.28)
\[ \sum_{i=0}^{M-1} \lambda_i \delta p_i^{\max} = \lambda_0^{\max} \delta p_0^{\max} + \lambda_1^{\max} \delta p_1^{\max} + \lambda_2^{\max} \delta p_2^{\max} + \ldots + \lambda_{M-1}^{\max} \delta p_{M-1}^{\max} \]

\[ \leq \lambda_0^{\max} \delta p_0^{\max} + \lambda_1^{\max} \lambda_0^{\max} \delta p_0^{\max} + \ldots + \lambda_0^{\max} \lambda_1^{\max} \ldots \lambda_{M-1}^{\max} \delta p_{M-1}^{\max} \]

\[ \leq \delta p_0^{\max} \left( \lambda_0^{\max} + \lambda_0^{\max} \lambda_1^{\max} + \ldots + \lambda_0^{\max} \lambda_1^{\max} \ldots \lambda_{M-1}^{\max} \right) \]

\[ \leq \delta p_0^{\max} \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k^{\max} \]

where we again recognize that \( \lambda_{M-1}^{\max} = 0 \). Plugging into our expression for \( \tau' \) we get the upper bound

\[ \tau' \leq \tau + \mu \delta p_0^{\max} \left( 1 + \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k^{\max} \right) + \beta \delta p_0^{\max} \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k^{\max} \]

\[ \leq \tau + \mu \delta p_0^{\max} (1 + \| \lambda^{\max} \|) + \beta \delta p_0^{\max} \| \lambda^{\max} \| \]

for

\[ \| \lambda^{\max} \| \equiv \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k^{\max} \]  \hspace{1cm} (3.31)

With this we can evaluate an expression for \( \Delta \):

\[ \Delta \equiv \frac{\tau' - \tau}{\tau} \leq \frac{\mu \delta p_0^{\max} (1 + \| \lambda^{\max} \|) + \beta \delta p_0^{\max} \| \lambda^{\max} \|}{6 \alpha M + \mu \hat{P}_0 (1 + \| \hat{\lambda} \|) + \beta \hat{P}_0 \| \hat{\lambda} \|} \]

\[ \leq \frac{\delta p_0^{\max} \left[ \mu (1 + \| \lambda^{\max} \|) + \beta \| \lambda^{\max} \| \right]}{6 \alpha M + \hat{P}_0 \left[ \mu (1 + \| \hat{\lambda} \|) + \beta \| \hat{\lambda} \| \right]} \]

\[ \leq \frac{\delta p_0^{\max}}{\epsilon + \frac{1}{c} \hat{P}_0} \]

for

\[ \epsilon \equiv \frac{6 \alpha M}{\left[ \mu (1 + \| \lambda^{\max} \|) + \beta \| \lambda^{\max} \| \right]} \]  \hspace{1cm} (3.33)

and

\[ 87 \]
\[ C \equiv \frac{\mu (1 + \|\lambda_{\text{max}}\|) + \beta \|\lambda_{\text{max}}\|}{\mu (1 + \|\lambda\|) + \beta \|\lambda\|}. \] (3.34)

We note that \( \epsilon \) is presumed to be small for typical problem sizes and parameter regimes, since it is a multiplicative combination of \( a, \beta, \) and \( M. \) This allows us to write \( \Delta \) entirely in terms of the initial particle configuration:

\[ \Delta \leq C \frac{\delta P_{0}^{\text{max}}}{P_{0}} = \frac{C}{\Gamma_{0}} - C. \] (3.35)

As discussed in [61], this form of the parameter \( C \) shows how the network parameters affect the model more when local leakage patterns deviate more from the average. However, as will be shown in subsequent sections this parameter is highly sensitive to the value of \( \|\lambda_{\text{max}}\|, \) which was derived on the assumption of spatially isotropic leakage equal to the maximum leakage for each stage. This assumption turns out to be particularly bad for later stages with very few particles, leading to severe over-predictions of the load imbalance penalty.

### 3.3 Results

The results that follow use the domain decomposition implementation described in Chapter 2 to empirically validate the performance model. This was done for two test problems: (1) an infinite medium case; and (2) the BEAVRS PWR model. For both, domain meshes consisting of evenly-spaced cubes were overlaid on top of the geometry for a variety of domain sizes - i.e., \( 2x2x2 = 8 \) domains total, \( 3x3x3 = 27 \) domains total, etc., up to the \( 8x8x8 = 512 \) domain case, which is shown in Figure 3-4.

With these runs, particle simulation counts and domain leakages were tallied for use in evaluating the model. These data allow for direct calculation of run time estimates using Equations 3.16 and 3.24, which can then be compared to the observed run times as well as the upper-bound prediction of Equation 3.35.

All runs were carried out on the Mira Blue Gene/Q supercomputer at ANL, and none of the BEAVRS runs were done with tallies or utilized any significant material
discretization.

### 3.3.1 Communication and Load Imbalance Overhead

Load imbalance penalties were calculated for both test problems from Equation 3.2 using observed run timings. The DD problem was run for each case to obtain $\tau'$ directly, and $\tau$ was approximated by running the problem with identical parameters and the traditional parallelization scheme. Since the traditional parallelization scheme evenly divides particles across all processes, these run timings will be equivalent to $\tau$ (the time for the DD run to finish if it were perfectly load-balanced) as long as the particle tracking term in Equation 3.16 is significantly larger than the latency and bandwidth terms. On Blue Gene/Q $\alpha$ and $\beta$ are somewhere between $10^{-8}$ and $10^{-6}$ - whereas $\mu \sim 10^{-5}$ - so this is a decent approximation to make. Note that all timings used in these equations refer to particle transport only, and exclude initialization and finalization time. This is consistent with the scaling study in [34], which demonstrates excellent parallel efficiency for particle transport in OpenMC using domain replication.

Figure 3-5 shows results for $\Delta$ as the domain mesh was refined. The two curves show results for the BEAVRS and infinite medium problems, where all runs used one compute node per domain. The infinite medium case is intended to demonstrate the
Figure 3-5: Timing penalties (Equation 3.2) for the OpenMC domain decomposition implementation for two eigenvalue test problems, calculated from transport time only (total time - initialization and finalization time). Domain meshes were evenly-spaced cubes overlaid on top of the geometry, i.e., 2x2x2, 3x3x3, ..., 8x8x8. Each mesh was run with one node per domain. All runs were conducted as a weak-scaling study, where the total number of particles transported was equal to 10000 per batch per node (i.e., for all 512-domain runs, 5.12 million particles per batch).
overhead introduced by the domain decomposition mechanics (i.e., the entire process of buffering and sending particles) as distinct from load imbalances, since the problem is naturally load-balanced in that case.

It should be noted that the domain boundaries in the 8-domain BEAVRS run split the particle load along lines of symmetry. As a result, for those runs it is expected that the load imbalance penalties mirror the infinite medium case.

All of the runs in Figure 3-5 were carried out in a weak-scaling manner with 10,000 particles per domain. Figure 3-6 shows how the implementation behaves when this parameter is varied. As can be expected, performance degrades as the amount of work to do between synchronization points decreases and the latency and bandwidth terms in Equation 3.16 start to dominate. However, this result is encouraging when we note that this regime does not begin until we get below 1k particles per domain. This means that we can safely use very fine domain meshes and achieve similar performance without needing to increase total particle counts. It should also be noted that this plot was
Figure 3-7: Load imbalance penalties for the BEAVRS problem compared to direct calculation using the performance model as well as predictions for the upper bound.

created without any tallies, so in real cases particle tracking will be slower and the situation more favorable.

3.3.2 Model Evaluation

The domain particle sources and leakage fractions were tallied for all runs in Figure 3-5. These allowed for the calculation of load imbalance penalties predicted by the model using Equations 3.16, 3.24 and Equation 3.2. The same data can also be used to calculate the approximate upper-bound penalties using Equation 3.35. These results are shown in Figures 3-7. Consistent with [85], the values of $\alpha$ and $\beta$ were estimated as $10^{-6}$ seconds per connection and $10^{-8}$ seconds per particle respectively (for $\beta$, 2 GB/s links with roughly 200 bytes per particle transferred). The particle tracking rate coefficient $\mu$ was set to $10^{-5}$ seconds per particle, which is consistent with tracking rate values observed for all runs on this machine that had no tallies.

The approximate upper-bounds appear valid for these results, but direct calcu-
Table 3.1: Upper-bound model parameters computed using previously-tallied data, presented for verification. Dual table entries denote: (value reported in [61]), (value calculated with the present derivation).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>$C$</th>
<th>$\frac{1}{\Gamma_x}$</th>
<th>$\Delta \leq$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Assembly</td>
<td>1.13, 1.13</td>
<td>3.24, 3.25</td>
<td>3.67, 3.68</td>
</tr>
<tr>
<td>Quarter</td>
<td>2.58, 2.53</td>
<td>3.28, 3.31</td>
<td>8.46, 8.38</td>
</tr>
<tr>
<td>Ninth</td>
<td>6.98, 7.06</td>
<td>3.45, 3.44</td>
<td>24.08, 24.25</td>
</tr>
</tbody>
</table>

The magnitude of the upper-bound is very sensitive to $\|\lambda_{max}^{|i|}\|$, which is a combination of the maximum domain leakage fractions over all stages. In many cases, it was observed that later stages consisted of a small number of neutrons bouncing back and forth between domains (especially common in reflector regions). In these cases the maximum leakage fraction $\lambda_{max}^{|i|}$ tended to be unity, serving to inflate the averaged leakage fraction considerably,\footnote{For finer meshes, $\|\lambda_{max}^{|i|}\|$ trends towards the number of stages needed to complete particle tracking (see Figure 3-1).} and leading to the over-estimation of the load imbalance penalty. This also explains the apparent noise in the upper-bound curve of Figure 3-7: results are highly sensitive to the stochastic movements of a very small number of particles. Overlapping-domain strategies discussed in the literature (e.g. [71]) have been shown to reduce this kind of particle movement, which would lessen the impact of the latency and bandwidth terms. However it appears from these results that such measures may not be necessary.

To verify that the present derivation of the model upper-bound is equivalent to that presented in [61], the previously-reported leakage fraction and particle source data tallied from full-scale OpenMC runs on the NEA benchmark problem (shown in
Table 3.2: Directly-calculated load imbalance penalties predicted by the unapproximated model equations from previously-tallied data. The full case used all data (compare to the mesh in Figure 3-4), whereas the restricted case used data only from assembly regions and stages where particles were run in assembly regions.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>No. Domains</th>
<th>$\Delta$, restricted</th>
<th>$\Delta$, full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Assembly</td>
<td>5780</td>
<td>2.67</td>
<td>3.42</td>
</tr>
<tr>
<td>Quarter</td>
<td>46240</td>
<td>2.39</td>
<td>3.06</td>
</tr>
<tr>
<td>Ninth</td>
<td>156060</td>
<td>2.44</td>
<td>3.12</td>
</tr>
</tbody>
</table>

Figure 3-1) were re-processed with the present formulation. These results are presented in Table 3.1, confirming the equivalence between the two derivations. Note that for these calculations tally data was only considered from assembly regions and “active stages” (stages that had particles running in assemblies, as opposed to reflector regions). This simplification is not needed in general, but was done here to compare with previous results.

The previous tally data were also used to compute the load imbalance penalties directly with Equations 3.16, 3.24, and 3.2, presented in Table 3.2. This was done both by using data from all domains over the whole geometry, as well with only the domains that covered fuel assemblies (the “restricted” column in Table 3.2). Results are consistent with the load imbalance penalties observed in Figure 3-5 for coarser domain meshes, and predict significantly better performance than the upper bounds reported in [61] calculated by Equation 3.35 for finer meshes.

### 3.4 Conclusions

Domain decomposition for Monte Carlo is more efficient on modern machines from an inter-domain particle communication standpoint than was previously thought. This was shown with timing results using the DD implementation in OpenMC described in Chapter 2, as well as verified with an updated performance model. These analyses are directly applicable to steady-state PWR eigenvalue problems, but the same methodology should be applicable to other reactor and problem types.

In particular, a re-derivation of the previously-established performance model pro-
vides a way to obtain more accurate estimates of communication and load imbalance inefficiencies for given problem and machine characteristics. The key here was realizing that certain assumptions used in the calculation of an upper-bound were in fact very overly-conservative, and that the more general unapproximated forms of the equations could be used directly with the same tallied leakage data.

Minimal particle communications overhead was observed for various domain meshes over the BEAVRS geometry, and load imbalances were shown to be between 3x and 4x, consistent with performance model predictions. Performance was also not observed to degrade considerably for particle counts as low as 1000 per domain, indicating that a need to refine domain meshes should not be a limiting factor, especially if particle tracking rates decrease, as would be the case for extensive tallies and detailed materials. Data for such finer meshes was also re-processed from previous studies using the updated formulation, further predicting that finer domain meshes should not incur significant overhead.

However, significantly finer domain meshes may not be needed. For instance, these results were obtained by running the problem in a configuration close to that of full-scale high-fidelity simulations. The finest case used over 5 million particles per batch over 512 domains, for which an aggregate of 8TB of memory was available. Though the use of finer domain meshes should be feasible, this level of decomposition should be sufficient for high-fidelity simulations. Of course, these analysis did not take into account the full memory burden required by materials and tallies as discussed in Section 1.7. This will be considered in Chapter 5.

27 This is with one MPI process per compute node. The local particle load can be divide among the additional cores on a node with shared-memory parallelism, which is already implemented in OpenMC. This was not demonstrated with these results, which did not utilize more than one core per node.
Chapter 4

Load Balancing Strategies

4.1 Introduction

As discussed in the literature and observed in Chapter 3, particle tracking load imbalances are normally expected to occur as domains get smaller and leakage fractions increase, incurring potentially significant parallel inefficiencies when scaling the number of domains. Even though recent results and the updated performance model from Sections 3.2 and 3.3 indicate that this imbalance may not be as bad as previously-thought for PWR problems, efforts can still be made to reduce this overhead.

Several algorithmic strategies exist to address the problem, though it is not immediately clear which ones to use for full-scale reactor simulations. In practice, a combination of each method may be appropriate depending on the number of compute nodes available, the severity of the load imbalance, and the extent to which the spatial load distribution is known or can be approximated.

This chapter will explore several variations of these load-balancing techniques with OpenMC in order to determine which are compatible with the current DD implementation, as well as assess the extent to which they are needed for the PWR problem. Some of the analysis and results are derived from a paper published in Parallel Computing [83].
4.2 Strategies

4.2.1 Matching the Mesh to Load

The primary way to manage load imbalances in DD simulations is to size the domain mesh according to the load. In this approach spatial domain sizes are chosen so that processes on each would have the same amount of work to do, as is discussed in most of the literature. The specifics of this strategy depend to a large extent on the geometry capabilities of the specific Monte Carlo codes in which domain decomposition has been implemented, but in general the approaches can be thought of as intended for two different scenarios:

1. Dividing a larger number of domains among a smaller number of processes
2. Dividing a larger number of processes among a smaller number of domains

The first group is typically found in codes that make no distinction between domains and geometry cells, as was considered by Alme et al. in 2001 [57]. Here, cells in the geometry are assigned to different processes to implicitly define larger domains. This has the advantage of avoiding cut cells for materials and tallies as discussed in Section 2.2.7.2, but it can considerably complicate the inter-domain particle communication pattern, especially since geometry cells assigned to each process need not be spatially contiguous. Indeed, this may have been part of the cause for the significant particle communication overhead observed in [57].

The second group recognizes a difference between domains and cells in the geometry, allowing for the broader domain mesh to be defined explicitly as an “overlay” on top of the geometry as was done by Procassini et al. and Greenman et al. in the Mercury Monte Carlo code [62, 63, 64]. This works better for general CSG geometry cells defined with arbitrary quadratic surfaces, but it introduces the possibility of having cells that reside on more than one domain. The domain mesh can be chosen arbitrarily to match the particle workload, but typically it is most convenient to use structured or unstructured meshes with regular connectivity between cells, which vastly simplifies inter-domain particle communication.
Figure 4-1: Radial view of the BEAVRS geometry showing an example of mesh restriction to better match the particle tracking load for the 8x8x8 DD mesh.

The present domain decomposition implementation follows this approach, using the regular rectilinear meshing capabilities of OpenMC to define domains over the geometry. Later sections will show that a complicated meshing strategy does not appear to be required for adequate load balancing of PWR problems, so a strategy of sizing domains with general unstructured meshes to match the load was not pursued in this work. However, to demonstrate the importance of mesh sizing, a modified run mode was added to the present implementation to allow the outer boundaries of the domain mesh to shrink. Figure 4-1 shows an example of such a “restricted mesh” for the BEAVRS geometry.

In this mode materials outside the domain mesh are loaded on all compute nodes, so particles that travel there can continue to be tracked. This kind of simple mesh/load matching is important for reactor problems like BEAVRS, where the vast majority of particle tracking will take place over the fuel assembly regions, even though the
geometry might be defined out through the reflector and reactor vessel.\textsuperscript{28} Of course, this does complicate the inter-domain communication algorithm somewhat, since in principal particles in the reflector could re-enter the DD mesh anywhere along the periphery.

Everything outside the restricted mesh could also simply be treated as a separate domain tracked by one or more processes, adding one domain to the communication stencil of every domain on the periphery of the inner domain mesh. However, this outer domain would need to communicate with all such periphery domains, which may not scale well.

This overhead can be avoided for both scenarios if particle histories are simply terminated when they leave the domain mesh. This would introduce only minimal error for PWR problems if the outer-most dimensions are chosen just outside the neutron shield panels, as confirmed in [86].

### 4.2.2 Matching Resources to Load

Another common strategy often used in combination with proportional domain mesh sizing is an approach that replicates domains proportional to the load, as was first discussed in Alme et al. [57]. For example, if one domain will have twice the workload of another, twice as many processes can be assigned to track particles on that domain. Figure 4-2 shows an example distribution for the BEAVRS geometry.

Appropriate process counts can be specified at the beginning of a simulation if the work distribution is known for the given domain mesh, or they can be set during simulation using particle mean-free-path heuristics or simple tallies. Procassini et al. also demonstrated an adaptive scheme in [62] where processes were re-assigned to different domains between particle generations, based on calculated loads from previous cycles.

Regardless of how they are assigned, the extra processes distributed across domains will only balance the load to the extent that the process distribution matches the load.

\textsuperscript{28}It is also particularly important when using a rectilinear domain mesh over a cylindrical geometry, to avoid corner domains outside the pressure vessel (see Figure 3-4).
Figure 4-2: Radial view of the BEAVRS geometry showing an example of resource matching to balance the particle tracking load for the 8x8x8 DD mesh. Numbers indicate a potential distribution for the number of processes working on each domain.

distribution. If processes can only work on one domain at a time, as is the case with the present implementation in OpenMC, then it would take an infinite number of processes to perfectly match the load distribution. This is because additional processes can only divide the work in each domain by an integer amount. Of course, this also means that this method could achieve perfect balance if the workloads in each domain are exact integer multiples of one another.

However, in practice we would rather specify a regularly-spaced domain mesh along logical boundaries of the geometry, such as groups of fuel assemblies. This avoids cut tally cells and simplifies post-processing, but it makes the prospect of achieving perfect load balance with a finite number of compute processes unlikely. Indeed, significant load imbalances could remain for domain meshes that do not conform to the workload. For instance, with the distribution of work over the domain mesh shown in Figure 3-3,
at least one compute process would be needed to track particles through each of the domains on the periphery of the core even though those domains have an insignificant percentage of the total load.\textsuperscript{29}

This gives rise to the question of how well a finite number of extra compute processes can alleviate load imbalance penalties if distributed across domains according to a known work distribution. With the current implementation, this can be quantified in the same terms as the previously-described performance model discussed in Chapter 3. Specifically, the load imbalance penalty can be written as

\[ \Delta_{l_b} \equiv \frac{\tau'_{l_b} - \tau_{l_b}}{\tau_{l_b}}. \] (4.1)

where \( \tau'_{l_b} \) and \( \tau_{l_b} \) are the times for the domain-decomposed and non-DD times under this load-balancing scheme.

The previous model assumed one compute process per domain, for a total of \( N \) processes. If instead we use \( K \) processes with \( K > N \), the time for the perfectly load-balanced case simply becomes

\[ \tau_{l_b} = \frac{N}{K} \tau \] (4.2)

if we neglect the latency term and assume perfect scaling, which is reasonable due to the independence of particle histories within a generation.

Next, if each domain \( j \) is assigned \( k_j \) processes where

\[ K = \sum_{j=0}^{N-1} k_j, \] (4.3)

then the time for the domain-decomposed case becomes

\[ \tau' = 6aM + \beta \sum_{i=0}^{M-1} \left( \frac{\lambda_i}{k} \right)^{max} + \mu \sum_{i=0}^{M-1} \left( \frac{p_i}{k} \right)^{max} \] (4.4)

where

\textsuperscript{29}A potential solution to this case would be to simply allow a single compute process to track particles for all domains along the periphery. This amounts to domain mesh resizing, which in general may not be possible due to tally and material memory constraints.
\[
\left( \frac{p_i}{k} \right)^{\text{max}} = \max \left\{ \frac{p_{ij}}{k_j} : 0 \leq j \leq N - 1 \right\} \tag{4.5}
\]

and

\[
\left( \frac{\lambda_i p_i}{k} \right)^{\text{max}} = \max \left\{ \frac{\lambda_{ij} p_{ij}}{k_j} : 0 \leq j \leq N - 1 \right\}. \tag{4.6}
\]

If we again neglect the latency term, assume that the domain with the highest leakage is also the domain with the highest load, and assume that \( \left( \frac{\lambda_i p_i}{k} \right)^{\text{max}} \) corresponds to the same domain as \( \left( \lambda_i p_i \right)^{\text{max}} \), then the time for the domain-decomposed case can be written as

\[
\tau'_{lb} = \frac{1}{k_{\text{max}}} \tau'. \tag{4.7}
\]

where \( k_{\text{max}} \) is the number of processes assigned to the domain with the highest load per process. The latter assumption is subject to the method by which processes are distributed across domains, but it should be valid as long as processes are distributed proportionally in a way that aims to globally minimize the maximum load on any single process.

Finally, with Equations 4.2 and 4.7, Equation 4.1 becomes

\[
\Delta_{lb} = \frac{K}{N k_{\text{max}}} \frac{\tau' - \tau}{\tau}. \tag{4.8}
\]

Equation 4.8 helps inform the algorithm that should be used to distribute extra processes across domains when the work distribution is known. For instance, Algorithm 4-1 describes the routine used to determine this distribution in the OpenMC DD implementation. For a given work distribution and number of available processes, this routine aims to maximize \( k_{\text{max}} \) by proportionally assigning an integer number of processes to domains with a ceiling function, and then removing nodes from domains with the least amount of work one by one.
Algorithm 4-1: Resource matching routine.

1: input $n_{domains}$, $nodes_{available}$, $work_{dist}$
2: for $i = 1 \rightarrow n_{domains}$ do
3:   $frac_{nodes}(i) = \frac{work_{dist}(i)}{\sum(work_{dist})} \times nodes_{available}$
4:   $nodes(i) = \max(1, \text{ceiling}(frac_{nodes}(i)))$
5:   $divisions(i) = frac_{nodes}(i)/nodes(i)$
6:   if $nodes(i) == 1$ then
7:     $divisions(i) = 1$  \hspace{1cm} \triangleright \text{Can't go less than 1 node per domain}$
8:   end if
9: end for
10: while $\sum(nodes) > nodes_{available}$ do
11:   for $i = 1 \rightarrow n_{domains}$ do
12:     if $divisions(i) == \text{minval}(divisions)$ then
13:       $nodes(i) = nodes(i) - 1$
14:     if $nodes(i) == 1$ then
15:       $divisions(i) = 1$
16:   else
17:     $divisions(i) = frac_{nodes}(i)/nodes(i)$
18:   end if
19: Break
20: end for
21: end while
22: output $nodes$

Symbol Table

$n_{domains}$ Number of domains  
$nodes_{available}$ Total number of available compute nodes  
i Domain index  
$work_{dist}(i)$ Relative amount of work to do in domain $i$  
$frac_{nodes}(i)$ Fractional number of nodes needed for domain $i$  
$nodes(i)$ Output actual number of nodes for domain $i$  
$divisions(i)$ Fractional amount of work per process in domain $i$
4.2.2.1 Intra-domain Fission Site Synchronization

One key distinction between this implementation and that first described by Alme et al. is the handling of fission sites between batches of particles: the previous work focused on fixed-source calculations, whereas here we consider eigenvalue problems. Fission site synchronization adds an extra layer of complexity to Monte Carlo simulations that has historically been a major source of parallel inefficiency. This was explored extensively by Romano [33], where a change from master-slave to nearest-neighbor algorithms was shown to drastically improve scalability.

This same sort of fission site treatment must be used within each domain when assigning multiple processes per domain with the resource-matching load balancing strategy. This is handled in a reproducible manner as discussed in Section 2.2.5.3, fundamentally utilizing the same nearest-neighbor algorithms described by Romano in [33] that scale with the number of particles being run as $O(\sqrt{N})$. Thus when using this load balancing strategy these the cost of algorithms are expected to scale with the number of particles that track through the domain, and overhead as a result can be mitigated by choosing domain sizes such that fewer particle track through. However, for most practical cases this overhead should be much less than the particle tracking load itself.

4.2.3 Particle Density Manipulation

Load imbalances might also be addressed by enforcing identical loads in each domain, adjusting weights of particles to preserve the Monte Carlo fair game. This can be done during transport using traditional Monte Carlo variance reduction techniques such as splitting and Russian Rouletting, or between particle generations during source sampling in eigenvalue problems with, for instance, the Uniform Fission Site method [27, 46]. The former can be used to achieve balance for all space, whereas the latter is only helpful over source regions (i.e. fuel assemblies in reactor problems). Figure 4-3 shows an example of a potential weight-window mapping.

It is important to keep in mind that the effectiveness of particle density manipulation
to alleviate load imbalances in DD simulations is no longer adequately described by a simple measure of parallel efficiency. Specifically, the choice of method can strongly affect the statistical convergence of tallies in each region of space. Indeed, this is why such methods are typically used for variance reduction in non-DD simulations. As a result, efficiency should be assessed using a more general figure of merit that includes the statistical convergence in regions of interest if these methods are used. For instance, splitting/rouletting could result in good load balance by manufacturing more work in reflector regions that normally have few particles to run. This improves the statistical quality of any tallies in those regions, but reduces the quality of tallies in fuel regions of interest. On the other hand, UFS serves to flatten the distribution of variances in the fuel region - a desirable quality of the method.
4.3 Results

Each of the previously-discussed load balancing techniques were explored for the PWR problem with the BEAVRS geometry. Figure 4-4 shows results for $\Delta$ as the domain mesh was refined in the same manner as was done in Chapter 3: times were calculated from transport time only (total time - initialization and finalization time), and domain meshes were evenly-spaced cubes overlaid on top of the geometry, i.e., 2x2x2, 3x3x3, ..., 8x8x8.

Curves 1 and 2 show the previous results for the BEAVRS and infinite medium problems, and curves 3-5 demonstrate the effect of each of the three previously-discussed load balancing strategies on the DD BEAVRS runs. Each mesh was run with one node per domain, with the exception of curve 3, where four times as many nodes as domains were distributed across domains according to the converged fission source distribution tallied in previous runs. Curve 4 was run with domain meshes where the outer boundaries were located closer to the edge of the fuel assemblies, as depicted in Figure 4-1. As before, all runs were conducted as a weak-scaling study, where the total number of particles transported was equal to 10000 per batch per node (i.e., for all 512-domain
Figure 4-5: Load imbalance penalties for the BEAVRS problem using the restricted mesh load balancing strategy compared to direct calculation using the performance model as well as predictions for the upper bound.

As expected, the efficacy of UFS as shown in curve 5 decreases as the number of domains that cover corner regions increases: the method only flattens the particle density over fuel assembly regions.

As before, the results in Figure 4-4 can be described by the model analysis in Chapter 3 and Section 4.2.2. For instance, the restricted mesh case shown in curve 4 can also be successfully modeled using tallied particle movement data, as shown in Figure 4-5.

In addition, curve 3 can be described by the quality of the match between process assignment and domain workload. Statistics for the quality of the matches obtained by Algorithm 4-1 are shown in Figure 4-6, where the match discrepancy $d$ is calculated for each domain $i$ as

$$d(i) = \frac{\text{abs}(\text{nodes}(i) - \text{frac}_\text{nodes}(i))}{\text{nodes}(i)}.$$  \hspace{1cm} (4.9)

using the same notation from Algorithm 4-1. Since this algorithm assigns at least one node to all domains, drawing nodes away from domains with the highest loads,
Figure 4-6: Statistics describing the match quality of the resource maps produced by Algorithm 4-1 for the domain meshes described in Figure 4-4.

\( \frac{\text{frac} \_\text{nodes}(i)}{\text{nodes}(i)} \) will always be less than \( \text{nodes}(i) \), and values of one indicate that a node was assigned to a domain that had an insignificant percentage of the total load. It is clear that this is the case for the finer meshes studied, where corner domains may not cover meaningful portions of the problem geometry (e.g., see the corner domains in Figure 3-4).

As expected, even though the quality of the match between the work and number of processes in each domain may not be very good for all domains, the overall load-balancing performance is determined only by the domain with the highest load. Indeed, Equation 4.8 predicts the results from curve 3 in Figure 4-4 very well, as shown in Figure 4-7.

4.4 Conclusions

Clearly the load imbalance penalty can be mitigated to a significant degree for PWR problems, and without requiring large effort on the part of the user. Indeed, we can still use regular rectilinear domain meshes with only minimal tailoring to avoid domains
with zero work, and the UFS variance reduction technique is fully automatic over the same mesh.

Particular emphasis is given to the strategy of matching resources to the load, since compute core counts are large in current HPC architectures and will continue to rise in the future, as discussed in Section 1.3. This means that simulations can normally expect to have many more compute resources available than there are domains, especially since the PWR problem will not require a particularly fine domain discretization. All that is needed is some idea of the work distribution with which to assign additional compute nodes to domains.

Fortunately, determination of this distribution is not difficult for PWR problems. For instance, it is not difficult to quickly tally the work distribution by counting the number of collisions and surface crossings in each domain, or by tallying the fission source distribution as was done for the studies in this chapter.

However the work distribution is obtained, it simply needs to be provided to OpenMC so processes can be assigned to domains at the beginning of the simulation using Algorithm 4-1. Process re-assignment based on automatic calculation of domain workloads
during the simulation as was done in [62] could also be done. However, this was not implemented in OpenMC since it would incur significant overhead re-arranging memory when process domain assignments are changed: all material number densities would need to be re-loaded, and tally data moved.

Ultimately, the results from this chapter indicate that extra complexity in the pursuit of load balancing may not be justified given the ease with which work distributions can be obtained and extra processes distributed accordingly. Indeed, in practical cases this process constitutes only a small fraction of the total effort required to analyze the full problem. For instance, particle movement data would only need to be tallied a small number of times, and then the performance of various domain meshes and process distributions can be evaluated using the performance model.
Chapter 5

Full-Core Analysis

5.1 Introduction

The discussion thus far has focused on the performance of domain decomposition from the perspective of particle communication and load imbalance penalties. With an implementation that has minimized overhead as a result of these, we should now have the ability to treat the full-core PWR problem with the desired tally and material memory burdens. For this to be done efficiently, data management algorithms should be designed with the full scale of the problem in mind, such as the ones discussed in Section 2.2.6. In addition, careful attention must be paid to several practicalities related to problem-specification and post-processing.

The size of the PWR depletion problem was described in Section 1.7.2.2: multiple reaction rate tallies are required for every nuclide in every unique region of the fuel. This results in several terabytes of memory for tallies on a mesh of 10 radial rings and 100 axial sections per fuel pin, the minimum discretization recommended by Smith and Forget in 2013 [28].

The closest demonstration of tallies approaching this scale was presented by Kelly et al. at the PHYSOR conference in 2010 with the MC21 Monte Carlo code [27]. This study primarily focused on converging power density tallies over 6.3 million regions.\(^{30}\)

\(^{30}\)NEA Monte Carlo Performance Benchmark: 241 PWR assemblies of 264 fuel pins, with 100 axial sections
to 1% accuracy, which required 40 billion neutron histories. These tallies required only 0.53 gigabytes, but the authors also reported conducting runs including nuclide reaction rate tallies over the same mesh for the 69 nuclides in the problem, which required 8 gigabytes. However, for these runs they did not score for the variance of each reaction rate tally (Equation 1.7), which reduced the needed memory by two-thirds. Given the relatively small size of these tallies, these demonstrations were small enough to fit within compute node memory using domain replication.

The important metric to consider when conducting this many tallies is the particle tracking rate. In particular, we must be cognizant of slowdowns that arise when scoring tallies during “active batches” (as opposed to “inactive batches” before fission source convergence, where tallies should not be accumulated). For example, Kelly et al reported a 9% increase in batch run time when scoring power density tallies over to 6.3 million regions with MC21, and a 45% increase in run time when scoring flux, absorption rate, and fission rate over the same mesh in a separate run. This magnitude of slowdowns incurred by scoring to tallies depends on the relative amount of work between tracking particles through a geometry and scoring tallies to the relevant data structures. This depends on the complexity of the problem, the particle tracking method employed by the code, and of course the number of tallies that need to be scored to at each particle movement or interaction.

Whatever the tally scoring overhead is, it should ideally not depend on whether or not domain decomposition is being employed. Therefore, any data management algorithms that handle domain-specific tallies and materials should not degrade particle tracking rates significantly in a robust domain decomposition implementation. Since there are two aspects of the routines described in Chapter 2 that have the potential to incur such extra overhead (on-the-fly (OTF) map look-ups and tally reduction), the results and discussions in this chapter are primarily aimed at observing the effect on performance of using this data management approach. To do this, long-running simulations with billions of neutron histories to converge the statistics on full-scale depletion tallies are not needed. Instead, it is enough to confirm particle tracking rate performance, and then extrapolate the overall computational cost of simulations with
more particle histories.

5.2 Results

The investigations in this section are carried out in three stages to characterize the ability of the current implementation to handle large-scale tallies:

1. Confirm the tally system efficiency at scale without OTF memory management
2. Confirm the OTF memory management system efficiency without the full memory burden
3. Demonstrate the performance of both systems together at the full scale of the problem

5.2.1 Non-OTF Tally Performance

Before assessing the performance of the general OTF data management approach described in Section 2.2.6.3, it is useful to first obtain a baseline understanding of how the OpenMC tally system performs with large tallies. It is expected to perform well, especially since the optimization of this system was one of the focuses of Romano’s dissertation in the early development of OpenMC [33]. In particular, we are interested in how the tally scoring and reduction routines affect overall particle tracking rates.

This can be done by carrying out timing runs while scaling up the number of tallies. However, non-domain-decomposed runs can only handle tallies up to several gigabytes before running out of memory on a local compute node, so to see the performance with a more representative tally mesh the use of domain decomposition will still be required. Fortunately, for the purposes of this study it is not difficult to use simple rectilinear mesh tallies without OTF data management algorithms, since it is trivial to determine in which rectilinear domain a single tally mesh cell should be handled. Actual full-core depletion problems will require more complex meshes - cell tallies as described in [87] are more appropriate - but this will serve as an adequate proxy.
Table 5.1: Mesh-tallies used for non-OTF tally system performance characterization. All tally meshes comprised of cubes covering the entire geometry, sized so that no tally mesh bin is cut by a domain boundary of the 8x8x8 domain mesh over the BEAVRS PWR geometry. For instance, mesh 1 had one tally bin per domain coinciding exactly with the domain mesh, mesh 2 had 8 tally bins per domain, etc. Memory size was calculated using one scoring bin per mesh cell, and compared to memory available on Blue Gene/Q for 512 processes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>No. Tally Cells</th>
<th>Tally Size (GB)</th>
<th>% Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>512</td>
<td>$1.144 \times 10^{-5}$</td>
<td>0.00%</td>
</tr>
<tr>
<td>2</td>
<td>4096</td>
<td>$9.155 \times 10^{-4}$</td>
<td>0.00%</td>
</tr>
<tr>
<td>3</td>
<td>$5.316 \times 10^7$</td>
<td>1.188</td>
<td>0.01%</td>
</tr>
<tr>
<td>4</td>
<td>$4.252 \times 10^8$</td>
<td>9.505</td>
<td>0.12%</td>
</tr>
<tr>
<td>5</td>
<td>$3.402 \times 10^9$</td>
<td>76.04</td>
<td>0.93%</td>
</tr>
<tr>
<td>6</td>
<td>$6.816 \times 10^9$</td>
<td>151.6</td>
<td>1.85%</td>
</tr>
<tr>
<td>7</td>
<td>$1.369 \times 10^{10}$</td>
<td>305.9</td>
<td>3.73%</td>
</tr>
<tr>
<td>8</td>
<td>$2.743 \times 10^{10}$</td>
<td>613.2</td>
<td>7.49%</td>
</tr>
<tr>
<td>9</td>
<td>$5.487 \times 10^{10}$</td>
<td>1226</td>
<td>15.0%</td>
</tr>
<tr>
<td>10</td>
<td>$1.095 \times 10^{11}$</td>
<td>2447</td>
<td>29.9%</td>
</tr>
</tbody>
</table>

Figure 5-1: Zoom of mesh-tally boundaries over a 3x3 section of BEAVRS pins (meshes 3 and 4) and a single pincell (meshes 5-10).
Figure 5-2: Timing results for full-core analog fission tallies on meshes 4-10 in Table 5.1, using domain decomposition without OTF memory management. All runs were carried out for 10 batches (5 active) of 5.12 million particles per batch on a half rack of Blue Gene/Q (512 compute nodes).

The meshes used for this study are shown in Table 5.1 and depicted in Figure 5-1. These were chosen to be structured Cartesian meshes across the entire geometry, sized so that each cubic tally mesh cell fits exactly into only one domain in the 8x8x8 cubic domain mesh over the BEAVRS PWR geometry. With these, two types of runs were carried out:

1. DD runs using the 8x8x8 cubic domain mesh with 1 process per domain
2. Non-DD runs using 512 processes with traditional domain replication

The domain-decomposed runs should provide insight into particle tracking rate slowdowns over a detailed tally mesh. These do not include tally reduction overhead, since only one process will ever score to each tally mesh bin. On the other hand, the non-DD runs include tally reduction operations as well.

Timing results with one scoring bin per tally mesh cell (analog fission tallies) are presented in Figure 5-2 and Table 5.2. Tracklength flux tallies took marginally longer, but identical trends were observed. All runs were carried out on the Mira IBM Blue
Table 5.2: Timing results in seconds for full-core tallies on the meshes in Table 5.1, as depicted in Figure 5-2. \( I \): Initialization time; \( F \): finalization time; \( T \): total run time; \( B_i \): Time in inactive batches; \( B_a \): Time in active batches. All runs conducted 10 batches of 5.12 million particles per batch, with 5 inactive batches. No load balancing or OTF data management was employed for DD runs.

<table>
<thead>
<tr>
<th>Tally Mesh</th>
<th>( I )</th>
<th>( F )</th>
<th>( T )</th>
<th>( B_a/B_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No DD no Tally</td>
<td>-</td>
<td>41</td>
<td>278</td>
<td>-</td>
</tr>
<tr>
<td>No DD and Tally</td>
<td>1</td>
<td>39</td>
<td>329</td>
<td>1.76</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>39</td>
<td>325</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>37</td>
<td>626</td>
<td>1.83</td>
</tr>
<tr>
<td>DD no Tally</td>
<td>-</td>
<td>40</td>
<td>977</td>
<td>-</td>
</tr>
<tr>
<td>DD and Tally</td>
<td>1</td>
<td>39</td>
<td>1205</td>
<td>1.72</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>40</td>
<td>1207</td>
<td>1.72</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>46</td>
<td>1222</td>
<td>1.74</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>37</td>
<td>1247</td>
<td>1.76</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>37</td>
<td>1280</td>
<td>1.84</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>39</td>
<td>1329</td>
<td>1.89</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>37</td>
<td>1412</td>
<td>1.97</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>39</td>
<td>1578</td>
<td>2.11</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>44</td>
<td>1859</td>
<td>2.28</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>42</td>
<td>2453</td>
<td>2.57</td>
</tr>
</tbody>
</table>
Figure 5-3: Particle tracking rate slowdown for active batches when tallying one score per mesh bin on meshes 4-10 in Table 5.1, using domain decomposition without OTF memory management.

Gene/Q supercomputer at the Argonne National Laboratory, which has 16 gigabytes of memory on each compute node. As indicated in Table 5.1, the finest tally mesh used 30% of the memory on each node when decomposed, and only the first three meshes fit into node memory with the domain replication scheme of the non-decomposed runs.

As expected, the amount of time needed to write output files grows with tally size. However, for longer production runs this will be constant and essentially negligible compared to particle transport time. We also observe from Table 5.2 that the equivalent non-DD run for mesh 3 is 3.3x faster than the same DD run after subtracting finalization time. This is consistent with the observed load imbalance penalty for this domain mesh from Figure 3-5. It should be noted that the finalization time for mesh 3 is larger in the non-DD case because OpenMC does not normally finalize tallies in parallel, instead consolidating all results to a master process for output writing.

The final column of Table 5.2 is depicted in Figure 5-3, showing the effect of performing tallies on particle tracking rates. As expected, active tally batches take longer to complete when more work needs to be done to score to each tally bin that
particles are tracked through. Mesh 3 is closest in size to the results reported by Kelly et al. for MC21, for which tally scoring overhead is roughly comparable (83% overhead for OpenMC with 53 million tallies vs 45% overhead for MC21 with 21 million tallies). The overhead increases as the number of tally bins grows, but for these runs it does not grow beyond a factor of 2.6.

These results demonstrate that the current domain decomposition implementation can successfully treat an approximation of the full tally memory burden. In addition, since simple mesh tallies were used without OTF memory management, we now have a baseline to which we can compare particle tracking rate degradation when using domain-aware memory allocation.

5.2.2 OTF Tally Performance

Now we wish to move to the actual tally mesh needed for depletion, which with the current domain decomposition implementation will use on-the-fly allocation of memory for materials and tallies in each of the depletable fuel regions. This memory burden was not accounted for in the particle communication results from Chapter 3, so we aim to demonstrate equivalent inter-domain communication and load-balancing performance with these results.

It is also time to switch to a more realistic domain mesh that would be used for actual reactor analyses. Whereas the previous studies in this thesis have used cubic domain meshes over the entire BEAVRS geometry, for real runs the mesh should be specified so no tally cells are cut by domain boundaries. For LWR problems this is not difficult to achieve with rectilinear meshes at the scale required, since fuel assemblies provide natural lines of separation. For example, Figure 5-4 shows the domain mesh that will be used for the depletion tally results in this section, which is sized radially to exactly fit a 3x3 group of assemblies. This choice of mesh is discussed further in Section 5.3.2.

In addition, the real problem requires a more complete set of fuel isotopics, with unique atom densities for each nuclide in each region of fuel. The previous studies only
considered fresh fuel in each enrichment zone of the reactor, which included only three isotopes of Uranium and 2 isotopes of Oxygen. For these studies, we switch to a set of 206 nuclides, taken from an ORIGEN-S verification and validation report [88], and listed in Appendix B.

All the remaining results in this section were obtained on the Titan Cray XK7 supercomputer at Oak Ridge National Laboratory.

### 5.2.2.1 Overhead

First we look to confirm the domain decomposition performance for the PWR problem when using the OTF memory management strategy. This has the potential to incur overhead during particle tracking, since as described in Section 2.2.6.3 the global indices in the material compositions and tally results arrays need to be converted to the proper indices in local memory. In addition, an extra ordering step must be carried out between batches before tallies can be reduced among all processes working on a single domain. These two operations enable the use of the same tally scoring and reduction routines that were shown to be efficient in Section 5.2.1.

The OTF management routines were designed to be efficient, so this overhead is
expected to be minimal. For instance, as mentioned in Section 2.2.7.1, index maps are stored in a hash table dictionary with fast look-up times, and the tally bin reordering should normally only be required after the first batch as long as enough particles were run to score to every tally bin at least once. Regardless of what the overhead is, the performance of both of these operations should only scale with the size of local tally and material memory, and not the global size of the problem or the number of particle histories being run. Thus if ever the overhead of these operations appears to be limiting, a finer domain mesh should help alleviate the problem.

In order to investigate the performance of these algorithms and prepare for the full-scale depletion run, the domain decomposition overhead was first confirmed with simulations similar to those conducted in Chapter 3. This requires relatively small-scale runs in order to carry out the non-DD runs needed to calculate the load imbalance penalty. As a result, a simple pin mesh with one region per fuel pin (no radial rings, no axial segmentation) was tallied over for all 206 nuclides, for the six depletion scores mentioned in Section 1.7.2.2. Unique atom densities for each nuclide were also loaded in an OTF manner for each pin. This resulted in just under 63 million tally quantities, requiring an aggregate of 1.4 gigabytes for tallies and 240 megabytes for material atom densities.

The domain decomposition performance results are shown in Figure 5-5, where the number of domains was scaled up from a 2x2x2 to an 8x8x8 domain mesh in the same way as was done in Chapter 3. The flats of the outer boundaries of each domain mesh in this series of runs were set to the outer radius of the neutron shield panels, which is why the overall load imbalance penalties are lower than those in Figure 3-5, which had very low-work domains at the corners as shown in Figure 3-4. As before, these results used one process per domain to run 10 batches (5 of them inactive) of particles in a weak-scaling manner with a total of 10,000 particles per process per batch. Since only one process was assigned to each domain, OTF treatment was only applied to tally scoring and material loading, and no tally re-ordering and reduction took place. This was done with the next series of runs which used multiple processes per domain.

\[31(193 \text{ assemblies}) \times (264 \text{ pins per assembly}) \times (206 \text{ nuclides}) \times (6 \text{ scores}) = 62,976,672 \text{ bins}\]
Figure 5-5: Domain decomposition overhead with OTF materials and tallies for cubic domains over the BEAVRS geometry, with a radial restriction to the outside of the shield panels. Tallies constituted \(~1.4\) gigabyte for the global problem in each case.

Figure 5-6: Load balancing via resource matching with OTF materials and tallies on a 7x7x5 domain mesh. Tallies constituted \(~1\) gigabyte for the global problem in each case.
Figure 5-6 shows domain decomposition overhead results for the same tally burden, while using the resource matching load balancing strategy to assign multiple processes to each domain according to the particle tracking workload. These runs used 5 axial domains and the 7x7 radial mesh depicted in Figure 5-4, meaning the largest run distributed 1960 processes over 245 domains and ran 19.6 million particles per batch. For that run, the high workload domains each had around 23 processes assigned to them, among which OTF tally reduction was carried out.

From the results in Figures 5-5 and 5-6, clearly the OTF treatment does not significantly affect the domain decomposition overhead for tallies at this scale. Indeed, for all of these runs the aggregate degradation of particle tracking rates in active batches was less than a factor of 1.9, which is consistent with the results from the previous section that carried out tallies of similar complexity without the OTF treatment. Of course, these runs were done with tallies much smaller than what is called for with the full scope of the depletion problem. For instance, for the 7x7x5 mesh used in Figure 5-6, only about 14 megabytes of tally data were reduced among processes in the central domains. With the addition of 10 radial rings and 100 axial sections per fuel pin, this should jump to several gigabytes unless a finer domain mesh is used.

5.2.2.2 Large-Scale Runs

Now that we have some confidence that the OTF memory treatment of tallies and materials does not significantly erode domain decomposition performance or particle tracking rates while tallying, we are ready to simulate the full depletion problem. This was done for the cases shown in Table 5.3, where the overall size of the tally memory burden is scaled up by increasing both the number of regions and the number of scoring bins in each region. All cases used 10 radial rings per fuel pin and had unique material number densities for each of the 206 nuclides in the problem. This corresponds to an aggregate of 19.5, 39.1, and 78.2 gigabytes for material number densities in the cases with 25, 50, and 100 axial segments per fuel pin, respectively.

Timing results for each of these cases are presented in Table 5.4. All were handled with the 7x7x5 = 245 domain mesh depicted in Figure 5-4 where possible, with
Table 5.3: Full-core depletion scenarios run for the BEAVRS geometry, with varying levels of spatial discretization and tally refinement. Tally scores consisted of \((n, fission)\), \((n, \gamma)\), \((n, 2n)\), \((n, 3n)\), \((n, \alpha)\), and \((n, p)\) reaction rate tallies for either the entire region or the number of nuclides specified.

<table>
<thead>
<tr>
<th>Case</th>
<th>Rings per Pincell</th>
<th>Axials per Pincell</th>
<th>Number of Nuclides</th>
<th>Number of Scores</th>
<th>Total No. of Tallies (B)</th>
<th>Total Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>25</td>
<td>1</td>
<td>1</td>
<td>0.01274</td>
<td>0.2847</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>0.02548</td>
<td>0.5694</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>0.05095</td>
<td>1.139</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>25</td>
<td>206</td>
<td>1</td>
<td>2.624</td>
<td>58.65</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>50</td>
<td>206</td>
<td>1</td>
<td>5.248</td>
<td>117.3</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>100</td>
<td>206</td>
<td>1</td>
<td>10.50</td>
<td>234.6</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>25</td>
<td>206</td>
<td>6</td>
<td>15.74</td>
<td>351.9</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>50</td>
<td>206</td>
<td>6</td>
<td>31.49</td>
<td>703.8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>100</td>
<td>206</td>
<td>6</td>
<td>62.98</td>
<td>1408</td>
</tr>
</tbody>
</table>

refinements in the axial domain mesh as needed to accommodate the memory burden on the Titan architecture. As before, all were done with 5 active and 5 inactive batches with one process per domain, here with only 1000 particles per process per batch. For the most detailed case with 10 radial rings and 100 axial sections, the full suite of depletion tallies over 206 nuclides was carried out with 12.25 million particle histories in 104 minutes over 2450 compute processes.

The final column of Table 5.4 is depicted in Figure 5-7, showing the particle tracking rate overhead for tallies at the full scale of the problem. For the most detailed case, particle tracking rates slow down by almost a factor of 8 during active batches. Note that even though the absolute magnitude of these particle tracking rates include load imbalances, this ratio does not since such imbalances are the same for both active and inactive batches. As a result this overhead is entirely the result of the extra work incurred by tallying, and not a function of the number of domains or the number of particles being run. It is not unexpected that this overhead would grow with more detailed tallies, and indeed this level of slowdown is an acceptable price to pay in order to successfully carry out the simulation. In fact, this marks the first time tallies at
Table 5.4: Timing results in seconds for full-core tallies with the depletion mesh scenarios in Table 5.3. \( I \): Initialization time; \( B \): Time in inactive batches; \( B_a \): Time in active batches \( F \): finalization time; \( T \): total run time. All runs were carried out with 5 active and 5 active batches, used 1 process per domain, and did not employ load balancing. Total particles per batch were set to 1000 per process (i.e. 2.45M particles per batch were used for the largest run).

<table>
<thead>
<tr>
<th>Depl. Case</th>
<th>Domains</th>
<th>( I )</th>
<th>( F )</th>
<th>( T )</th>
<th>( B )/( B_a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7x7x5</td>
<td>86</td>
<td>6</td>
<td>1550</td>
<td>2.60</td>
</tr>
<tr>
<td>2</td>
<td>7x7x5</td>
<td>124</td>
<td>12</td>
<td>2415</td>
<td>3.50</td>
</tr>
<tr>
<td>3</td>
<td>7x7x5</td>
<td>220</td>
<td>23</td>
<td>4931</td>
<td>5.19</td>
</tr>
<tr>
<td>4</td>
<td>7x7x5</td>
<td>96</td>
<td>7</td>
<td>1654</td>
<td>2.82</td>
</tr>
<tr>
<td>5</td>
<td>7x7x5</td>
<td>134</td>
<td>13</td>
<td>2560</td>
<td>3.75</td>
</tr>
<tr>
<td>6</td>
<td>7x7x5</td>
<td>248</td>
<td>26</td>
<td>5211</td>
<td>5.49</td>
</tr>
<tr>
<td>7</td>
<td>7x7x10</td>
<td>93</td>
<td>6</td>
<td>3102</td>
<td>6.49</td>
</tr>
<tr>
<td>8</td>
<td>7x7x10</td>
<td>144</td>
<td>12</td>
<td>4219</td>
<td>7.08</td>
</tr>
<tr>
<td>9</td>
<td>7x7x50</td>
<td>565</td>
<td>28</td>
<td>7620</td>
<td>7.83</td>
</tr>
</tbody>
</table>

Figure 5-7: Active vs inactive calculation rate for full depletion tallies.
this scale have been achieved with a single Monte Carlo neutron transport simulation for any amount of computational cost.

5.2.2.3 Projections

Full-scale analyses will differ from the runs carried out in this thesis primarily in the total number of particle histories run to achieve good statistical convergence of tallies. This extra work can be divided among additional compute nodes and estimates for run times can be obtained using observed particle tracking rates. As an illustrative example, we will estimate the expected computational cost of carrying out 10 billion particle histories for the most detailed tally mesh demonstrated in the previous section.

In an absolute sense, the magnitudes of particle tracking rates observed for the full-scale runs in this chapter appear much slower than expected. For instance, Romano et al. reported an estimate of 140 particles/second per CPU for tallying with OpenMC on Titan [39], whereas here for the finest run the aggregate tracking rate during active batches was observed to be 1.595 particles per second per CPU. Thus if the current implementation is used as-is, it would take about 200 CPU-years to run 10 billion histories on Titan.

Fortunately, the situation is much improved when accounting for several practical considerations of the results in this chapter, as well as of current and future supercomputing architectures. In particular:

1. No shared memory parallelism was employed: one MPI process was assigned to each 8-core NUMA node (2 per CPU).
2. No load balancing was employed.
3. The code was compiled without any compiler optimization, in debug mode.\(^{32}\)

For the purposes of estimation, absolute particle tracking rates similar to the estimates by Romano et al. can in fact be projected from these results if each one of these factors are taken into account.

\(^{32}\)This was because of segmentation faults encountered when compiling only on Titan - it is expected that this difficulty can be overcome with more detailed debugging.
First, since the present DD implementation did not utilize shared memory parallelism, 7 hardware threads were left un-used for each NUMA node on Titan. Future HPC architectures are projected to have many more hardware threads available per node, but for current architectures we can conservatively assume a 6x increase in particle tracking rates simply by making use of all hardware threads available on a single CPU with shared memory parallelism (for which efficient implementations have been previously demonstrated in OpenMC, e.g. by Siegel et al. in 2013 [89]). This brings our particle tracking rate to 9.57 particles per second and the time required for the large run to 33 CPU-years.

Second, given the relatively small number of domains needed (2450 for the largest run in this work) compared to the number of available compute nodes on HPC machines (projected to be 1M in 10 years), it will not be difficult to distribute extra resources across domains to nearly eliminate load imbalances. Such imbalances for the finest domain mesh were observed to be about a factor of four, so for the purposes of estimation this brings our particle tracking rate to 38.2 particles per second and the time for the large run to 8.3 CPU-years. This is manageable: 10,000 compute nodes on Titan (a little over half the machine) should complete 10 billion particle histories in just over 7 hours.

The situation is only expected to improve further with compiler optimization, which routinely achieves speedups in particle tracking rates of 3x for OpenMC on other architectures. With this, our estimated particle tracking rate is roughly consistent with those reported by Romano et al.

5.3 Practicalities

In order to carry out large-scale tallies, several practical issues deserve some discussion. In particular, careful attention to the problem specification is required if analyses are to be efficient. This is true of all large-scale Monte Carlo analyses, but several caveats are

33This is true not only from observations of runs in this work, but also with the predictions using the data from Siegel et al. as presented in Table 3.2, which used meshes with up to 60 axial domains
specific to domain decomposition, as well as to the PWR problem in the context of the current implementation.

### 5.3.1 Geometry and Tally Specification

One of the core enabling features of the current implementation is the ability to handle memory in large contiguous chunks. This allows for more efficient tally reduction and scoring, and saves considerable memory in data structure overhead. Though more general implementations are conceivable, in order to take advantage of this with the current implementation in OpenMC the problem must be specified in an optimized way that takes advantage of repeated structures in the geometry.

Specifically, different cell and tally structures should not be individually specified for each of the millions of unique depletion regions in the problem. Instead, a single universe definition should be used repeatedly throughout the geometry with the distributed indexing capabilities described in Section 2.2.6.2 for materials and tallies over each unique occurrence. The goal is to have the smallest number of tally structures possible, with the largest number of scoring bins per tally.

The benefits of this kind of problem specification are generally not specific to domain-decomposed problems (memory savings, particle tracking rate improvements, etc. as discussed further by Lax [76]). However, it may be required when using domain decomposition to treat geometries that are very detailed, since as discussed in Section 2.2.6.3 all over-arching tally and geometry data structures are in fact repeated for all domains. Only the arrays of atom densities and numerical tally scoring bins are actually distributed across domains, thus a significantly larger number of domains might be required to store the problem geometry and tally meta-data if this approach is not adhered to.

### 5.3.2 Domain Mesh Specification

Several practical considerations should be taken into account when specifying the size and spacing of the domain mesh over the PWR problem, but in general it is a fairly
straight-forward process. The primary constraints involve: (1) a desire to avoid tally regions on more than one domain, (2) a desire to minimize particle tracking and communication imbalances, and (3) a need to fit data into the available memory of each process when decomposed. The first two constraints control the size and positioning of domains and should be considered first, whereas the third constraint determines the number of domains that will be needed.

The first issue is simple to address for PWR problems by specifying domains along assembly boundaries, as was done in the previous sections with domains that contained 3x3 groups of assemblies, radially. As a practical matter, since there are an odd number of assemblies along a flat for this particular problem (15 rows and columns), and there are an odd number of pins in an assembly (17 rows and columns), we are limited to domains that group an odd number of assemblies (i.e., single-assembly, 3x3, 5x5, etc) if we are to avoid cutting single pins with a boundary. This could be relaxed if irregular unstructured meshes are implemented in OpenMC, which may in fact be necessary to treat other reactor types. For instance, the spacing between assemblies in BWRs is not always constant, and the number of pins can also vary from assembly to assembly.

The second issue is similarly straightforward to consider, especially since particle movement in PWRs is relatively isotropic over the bulk of the high-work regions of the problem. In this case, cubic domains are clearly the most efficient from an inter-domain particle communication perspective: a balanced scenario will have the same amount of particle communication out of each facet of a domain. However, as demonstrated in Chapter 3, the overall magnitude of particle communication costs as a percentage of total work is small, and thus this shaping of domains is largely a secondary concern.

Both of these two aspects tie into the third, where the number of domains needs to large enough to decompose data into small enough chunks that fit into node memory. For instance, given the memory available to each compute process for the demonstrations in the previous section it was sufficient to stick with the 3x3 assembly mesh radially, and simply refine the axial mesh as needed. However, if the memory burden grows larger or the memory available per process grows smaller (both of which are expected for future problems and HPC architectures), this could produce domains with extremely
poor aspect ratios. In such cases it would be preferential to switch to an assembly-wise mesh radially before refining the axial mesh further.

It is not difficult to estimate the absolute number of domains needed from the total memory needed for materials and tallies. This is straightforward to calculate for PWR problems as

\[ M = M_{\text{materials}} + M_{\text{tallies}} = PCI m_m + PCI S m_t \]

where \( P \) is the number of fuel pins, \( C \) is the number cells per fuel pin (i.e. rings and axial sections), \( I \) is the number of isotopes in the fuel, \( S \) is the average number of scores needed for depletion, and \( m_m \) and \( m_t \) are the number of bytes needed for each nuclide atom density and tally scoring bin, which for OpenMC are 8 and 24 bytes, respectively.

Then, a simple heuristic can be used to choose the number of domains so that the aggregate memory requirement given by Equation 5.2 fits into local available memory. This calculation needs to include the memory needed for other quantities. For instance, if the tally and material memory is divided equally among all domains, the number of domains \( N \) should be set by

\[ N \geq \frac{M}{M_{\text{process}} - M_{\text{OS}} - M_{\text{meta-data}} - M_{\text{XS}}} \]

where \( M_{\text{process}} \) is the installed memory available to the process, \( M_{\text{OS}} \) is the memory overhead required by the operating system, \( M_{\text{meta-data}} \) is the extra memory required to carry out the simulation (i.e., geometry data structures, OTF maps, particle and fission bank arrays, etc.), and \( M_{\text{XS}} \) is the memory required for cross section data. In general this guideline provides a good starting point when scoping out problem requirements, but special considerations will be required if domains will be irregularly-sized or materials and tallies are not regularly distributed across domains.

For instance, such caveats were required for the domain mesh used in the previous section, since tally data was not evenly distributed across all domains. In particular,
looking at Figure 5-4 it is clear that many of the the outer-most domains would treat none of the tally data, which is mainly only distributed among the inner 5x5 section of domains that overlap the fuel assemblies.

Finally, sometimes practical quirks of the particular architecture also need to be taken into account. For instance, in many cases memory available is not always accessible in the amounts quoted by the specifications of the system without special optimization. It is not uncommon for memory failures to be observed when attempting to use upwards of 60% of the memory that is supposed to be available to each node, so as a general rule of thumb it is advisable to design runs that stay below this value if possible.

5.4 Conclusions

The ability of the domain decomposition implementation described in Chapter 2 to facilitate tallies requiring over a terabyte of memory has been successfully demonstrated on the Mira and Titan supercomputers. This was done at the full scale of the depletion problem for the BEAVRS benchmark both with and without the use of on-the-fly memory management for treating domain-specific quantities, for a detailed mesh tally requiring 2.4 terabytes as well as a faithful depletion tally with 10 radial rings and 100 axial sections per fuel pin requiring 1.4 terabytes. Furthermore, particle tracking rates were shown to be comparable with non-domain-decomposed runs for the less memory-intensive cases, with active batches taking roughly twice as long as inactive batches. This overhead was observed to grow for the most detailed cases, with active batches taking roughly 8 times as long as inactive batches.

It is important to remember that these particle tracking rate slowdowns have little to do with domain decomposition itself, with overheads arising primarily because of the increased work to do during particle tracking to score to tallies. Domain decomposition simply enabled the use of a tally mesh with more complexity than has ever been previously accomplished, allowing us to observe the efficiency of the existing tally system at this scale. Further optimizations to the tally system were not explored in this work, but are expected to improve the absolute magnitude of particle tracking rates.
considerably.

Indeed, the results in this chapter were acquired in a sort of worst-case scenario, without using shared memory parallelism, without using compiler optimization, and without employing any load-balancing. By taking these factors into account with conservative assumptions, it can be estimated that the kinds of very large runs are tractable on modern HPC architectures with a reasonable amount of run time, where billions of neutrons would be simulated to adequately converge the statistics of these tallies.

This marks the first time tallies and materials requiring more memory than is available on a single compute node have been successfully demonstrated in a Monte Carlo neutron transport simulation of a fully-detailed nuclear reactor problem. This brings us one step closer to feasibly using this high-fidelity method for more routine use in robust multi-physics analyses. The domain decomposition parameters used to achieve this were easy to determine, and the problem geometry and tally specification does not require any onerous meshing to achieve efficiency. In fact, it was shown that relatively coarse domain meshes are sufficient to fully-accommodate the tally memory burden on modern HPC architectures, and acceptable performance is achievable even with structured rectilinear meshes without employing any load balancing.

In short, the domain decomposition approach was found to be simple, forgiving, and easy to use with Monte Carlo neutron transport methods in the context of typical high-fidelity reactor problems.
Chapter 6

Conclusions

With a desire to move to more detailed models, Monte Carlo is becoming competitive with deterministic methods for high fidelity neutronics simulations of nuclear reactor systems. This is particularly true in light of the inherent parallel nature of the algorithm, which fits well with increasing concurrency of current and predicted high performance computing architectures. In addition, the physics can be treated with much greater accuracy than with deterministic methods, and the geometry of the problem can be treated exactly. However, the method requires a vast number of particle histories in order to achieve adequate statistical convergence over a fine tally mesh, necessitating an immense computational cost. This should be addressable with the scale of HPC machines, but only if the parallel model can successfully deal with the considerable memory burden imposed by the number of material densities and tally results needed for robust analyses.

This memory burden can be handled with domain decomposition. However, until recently it was widely-expected that such an approach was not viable due to particle communication costs and load imbalances. This work aimed to add to the body of work that contests this thinking, by performing a real analysis with a real domain decomposition implementation.

Chapter 1 mentioned several issues: particle communication and load balancing, memory treatments for materials and tallies, and practicalities of carrying out real analyses. In this chapter we will discuss the progress achieved by this thesis, as well as
future work that remains.

6.1 Particle Communication and Load Balancing

The two primary issues thought to impede the use of domain decomposition were particle communication costs and load imbalances [58, 59]. The overhead from these two aspects were previously observed to be significant even for small particle workloads as described first by Alme et al. [57] in 2001, and subsequent works tended to look for ways to reduce it, e.g., with overlapping domains as described by Wagner et al.[71] or with more complicated asynchronous schemes such as those described by Brunner et al. [68].

Recent works by Siegel et al. and Romano showed that communication costs were much less than conventional wisdom indicated for real nuclear reactor analysis. Their studies made this claim using a performance model based on leakage rates tallied from a real core model. However, true validation requires a real implementation at scale.

6.1.1 Contributions of Present Work

In Chapter 2, a complete implementation of domain decomposition was discussed. This included a number of intricacies and practicalities needed to actually make it work, and make it work well. This eschewed the more complicated asynchronous schemes discussed by Brenner et al. in the literature, opting for what was used by Siegel et al. to add a series of blocking particle synchronization stages to the algorithm. At the core was the particle communication algorithms, which ensure scalability by fixing the number of domains processes would need to communicate with between stages, and at the same time facilitating efficient load balancing with the resource matching routines. In addition, Chapter 2 describes the numerous details required for random number reproducibility, which guarantees identical particle tracking by communicating random number seeds with particles across domain boundaries. Modifications to make the initial source site and inter-batch fission site sampling routines reproducible are also discussed.
Chapters 3 and 4 demonstrated the performance of the particle communication and load-balancing algorithms, confirming the efficiency for realistic particle tracking loads over the BEAVRS PWR problem. Chapter 3 also updated the theoretical framework established by Siegel et al. and Romano to better predict load imbalance performance from particle tracking data with fewer approximations. This updated model was used to reprocess data tallied from fine domain meshes of the previous studies to show that performance degradation was significantly over-estimated. Similarly, Chapter 4 describes the effectiveness of the resource-matching load balancing strategy in the same performance model terms, and in that context presents an algorithm to optimize the process distribution across domains and minimize load imbalances.

Ultimately, this work showed that with several relatively-easy load balancing strategies, domain decomposition can be done with Monte Carlo on PWR problems with slowdowns over non-decomposed runs of between 1.4x to 1.75x.

6.1.2 Future Work

Though the current work demonstrates only modest overhead for particle communication and load imbalances that can be largely mitigated, there are a number of extra optimizations that can be pursued to improve the situation further. First, the existing optimizations to particle communication discussed in the literature can be implemented, such as overlapping domains and asynchronous inter-domain particle transfers. However, given the good performance of the current method it is unclear whether or not the additional complexity is worth any speedups achieved. In a similar vein, more nuanced particle buffering can be explored, such as the sparse and in-place buffering schemes discussed by Felker et al. in [90].

Additional contributions can be made to the load balancing situation. For instance, performance could be improved by implementing more general unstructured meshes to better match the mesh to the load, and automatic sizing and/or re-balancing during simulation might pursued in conjunction, such as discussed by Procassini et al. [62]. However, automatic mesh refinement and re-balancing must deal with the difficulty of
moving tally and material data between compute nodes when they are reassigned, which is likely to incur significant overhead. In any case, more general meshing capabilities will be needed to avoid cut cells in different reactor types, such as with hexagonal geometries commonly found in fast reactors. In doing so, care must be taken to ensure equivalent treatment with the communication stencil, where processes only communicate among the local neighborhood.

6.2 Memory Treatment

The scope of the full-core problem presented a challenge, with terabytes of aggregate memory needed for a faithful treatment of depletion. This was an issue for Monte Carlo mainly because in the traditional parallel model the entire problem needed to be replicated across all compute nodes: a situation for which some form of decomposition is needed if Monte Carlo is to have any hope of tackling the problem. Previous work by Romano [33] has predicted that data decomposition is promising avenue for dealing with tally memory, but it is still unclear whether or not this would be efficient for treating nuclide atom densities in each material, which requires over 100 gigabytes over the whole problem. Furthermore, such a method has yet to be demonstrated at the full scale of the problem with a full implementation.

Domain decomposition promises to alleviate both tally and material memory issues, but several practicalities of memory management still presented a challenge. Specifically, the machinery of loading domain-specific quantities into memory needs to be demonstrated at scale, and it needs to be done in a way that doesn't unsustainably grow the number of domains or significantly degrade particle tracking rates. This is made complicated by the necessity of tally reduction within a domain when using the resource matching load balancing strategy.

6.2.1 Contributions of Present Work

In Chapter 2, an on-the-fly (OTF) memory treatment was described that enables memory for domain-specific quantities to be allocated only when needed as a result of particles
tracking through the corresponding regions of the problem. This allows us to avoid
the potentially complicated task of automatic determination of domain residence for
arbitrary tally regions in arbitrary domain meshes, and removes responsibility for the
task from users. Algorithms are also presented to efficiently carry out tally reduction
using OTF-allocated tally results, which requires a sort after the first batch of particles
to utilize the same contiguous-block memory operations for tally reduction that were
previously developed by Romano for efficient tally performance [33]. This was done in
an efficient manner, scaling by the number of tallies in a domain as $O(n)$ in time and
$O(1)$ in memory.

Results in Chapter 5 demonstrated the effectiveness of these treatments to handle
OTF loading of materials and tallies at the full scale of the problem, demonstrating
that any additional overhead generated by the method does not degrade domain
decomposition performance or impede load balancing.

### 6.2.2 Future Work

Though the memory treatment in the current implementation was shown to successfully
enable full-scale domain-decomposed runs with the true memory burden needed for
high-fidelity simulations, several drawbacks are apparent that could use improvements.

First, as discussed in Section 2.2.7.2, the current implementation does not robustly
treat tally cells that are cut by domain boundaries, instead treating the sections that
fall in different domains as unique tallies independent of one another. This could lead
to problems with high aspect ratio (“slivers”), volume calculation difficulties for each
section, and the fact that subsequent depletion steps would change the materials of
a cell in a manner that is dependent on the domain mesh. Thus with the current
implementation this situation must be avoided. However, if a scheme could be devised
to reduce these tallies across domain boundaries, cut cells could be used without issue.
This might remove the need to implement more general domain meshing, which might
be very attractive under certain circumstances. The difficulty here lies in finding a way
to do this efficiently, since in this case a tally might potentially (although highly unlikely)
need to be reduced amongst processes on every other domain, which could cripple parallel scalability. Furthermore, the meta-data required to carry this out threatens to grow exponentially, since every scoring bin might need to maintain a list of all other domains the quantity needs to be reduced with.

Another optimization that could be pursued is a form of “memory load balancing,” whereby data decomposition is used to normalize the memory burden across processes. For instance, as discussed in Chapter 5 there were numerous domains around the periphery that did not overlap with any tally or material data, yet still contained appreciable particle tracking workloads. With the data decomposition scheme discussed by Romano et al. [39, 79], tally results might be stored on processes assigned to these domains. Furthermore, such a scheme might also be used to handle cut-cells, though the issue of meta-data and overhead must still be addressed for this to be done efficiently at scale.

Finally, since memory is allocated in an OTF fashion, the ordering in memory on any particular domain is scrambled. Currently this means that maps must be used in post-processing to unscramble these arrays if results are to be matched with specific locations for analysis or multi-physics coupling. Attempts were made in this work to do this unscrambling in OpenMC during parallel I/O in finalization, but performance suffered significantly when writing to different sections of ordered data files (on the Lustre filesystem on Titan, ~60 gigabytes per second when dumping contiguous chunks to separate files vs ~100 megabytes per second when writing individual values in the proper positions of a single output file). Solving this issue is mainly a matter of convenience for the user, but it could have performance implications during iterative multi-physics coupling schemes in the future.

6.3 Full Core Analysis

Though previous works have attempted to solve the PWR problem according to the specifications of Smith’s first challenge in 2003 using Monte Carlo, none have demonstrated the complete tally and material memory burden imposed by the detailed tally mesh
specified by Smith and Forget's subsequent challenge in 2013. The last objective of this thesis was to use the complete domain decomposition implementation that was added to OpenMC to carry out this kind of analysis and characterize the performance.

6.3.1 Contributions of Present Work

In Chapter 5, the full scale PWR depletion problem was addressed with the new domain decomposition implementation using OTF tally and memory treatment. This was successfully carried out with 10 rings and 100 axial sections per fuel pin in the BEAVRS PWR benchmark in the finest case, with 6 reaction rate tallies for each of 206 nuclides in each region. This constituted an aggregate of 79 gigabytes of memory for material atom densities, and 1.4 terabytes of memory for tallies. Particle tracking rates were observed to be comparable with non-DD runs, with only modest slowdowns observed during active batch tallies as a result of increased tally complexity (and which have nothing to do with the number of domains used or the number of particle histories run). These specific runs were small, with the largest running only 5 batches of 2.45 million particles per batch, but the particle tracking rates allowed for an estimation of the computational cost that would be needed for longer production runs. For example, a run with the same tally mesh that tracks 10 billion particle histories should take 8.3 CPU-years on the Titan Cray XK7 supercomputer. This amounts to just over 2 million core-hours on the same machine, which is manageable given the size of typical allocations on HPC machines. With the incorporation of shared memory parallelism, the use of load balancing, and optimized compilation, this run would be achievable on 10,000 compute nodes in just over 7 hours.

Through this analysis, insight into the practicalities of the problem specification was also achieved, and several heuristics were developed to help with the choice of domain decomposition parameters relevant to reactor problems. Specifically, a domain mesh that contained assemblies in 3x3 radial groupings was determined to be ideal to avoid cut cells, with which axial refinements of up to 50 domains were used to accommodate the full tally memory burden arising from depletion tallies over 10 radial rings and 100
axial sections in each fuel pin.

6.3.2 Future Work

As discussed in Chapter 5, the overall performance of large-scale analyses could be improved with more optimized tally scoring algorithms to increase particle tracking rates during active batches. Importantly, this includes the incorporation of shared memory parallelism, which shouldn’t conflict at all with this domain decomposition implementation as long as only one “master thread” per MPI process is used for inter-domain particle communication and intra-domain tally reduction. This might require a few extra data reductions among threads.

In addition, a more sophisticated suite of post-processing tools should be developed to handle tally results at this scale. This is needed for analysis of results as well as coupling to other multi-physics routines. Importantly, these tools will also need to operate in a parallel and memory-efficient manner, since of course the results are not expected to fit into memory on any single compute node that will carry out such processing.

Finally, future work might continue to scale the problem up with finer depletion meshes, more nuclides, and more reaction rate tallies per nuclide in each region. Performance is not expected to suffer due to the domain decomposition implementation in these cases, except of course if domains must become exceedingly small to accommodate extreme memory burdens.
Appendices
Appendix A

OpenMC Particle Data Structure

Since particles must be communicated across domain boundaries for a domain-decomposed problem, the amount of information that must travel with particles can have important implications given the bandwidth of node interconnects. Every attempt should be made to minimize the amount of sent over the network, and data should be packed in a way that aligns in contiguous memory for best performance.

Listing A.1 shows the full OpenMC data structure for particles. Many of the members of this structure facilitate particle tracking local to a domain and thus don’t need to be communicated. Listing A.2 shows the particle buffer data structure for sending particles over the network, which mirrors the full particle data structure. However, it is stripped down to the base essentials needed to maintain random number reproducibility as discussed in Section 2.2.5. To that end several additional members are included here, notably the random number seeds for tracking, tallying, and cross section probability table sampling. The ordering of fields and the sequence Fortran keyword are important for efficient storage and data transfer with custom MPI datatypes derived from the structure.

With integers, longs, and doubles requiring four, eight, and eight bytes each, respectively, we can see from Listing A.2 that OpenMC will need to send 220 bytes per particle across the network, assuming two random number streams (one for tracking, and one for tallies).
Listing A.1: Full OpenMC particle data structure.

type Particle
  ! Basic data
  integer(8) :: id ! Unique ID
  integer :: type ! Particle type (n, p, e, etc)

  ! Particle coordinates
  type(LocalCoord), pointer :: coord0 => null() ! coordinates on universe 0
  type(LocalCoord), pointer :: coord => null() ! coordinates on lowest univ

  ! Other physical data
  real(8) :: wgt ! particle weight
  real(8) :: E ! energy
  real(8) :: mu ! angle of scatter
  logical :: alive ! is particle alive?

  ! Pre-collision physical data
  real(8) :: last_xyz(3) ! previous coordinates
  real(8) :: last_uvw(3) ! previous direction coordinates
  real(8) :: last_wgt ! pre-collision particle weight
  real(8) :: last_E ! pre-collision energy
  real(8) :: absorb_wgt ! weight absorbed for survival biasing

  ! What event last took place
  logical :: fission ! did the particle cause implicit fission
  integer :: event ! scatter, absorption
  integer :: event_nuclide ! index in nuclides array
  integer :: event_MT ! reaction MT

  ! Post-collision physical data
  integer :: n_bank ! number of fission sites banked
  real(8) :: wgt_bank ! weight of fission sites banked

  ! Indices for various arrays
  integer :: surface ! index for surface particle is on
  integer :: cell_born ! index for cell particle was born in
  integer :: material ! index for current material
  integer :: last_material ! index for last material

  ! Statistical data
  integer :: n_collision ! # of collisions

  ! Track output
  logical :: write_track = .false.

  ! Distributed Mapping Info
  integer, allocatable :: mapping(:) ! records the current sum of all maps
  integer :: inst ! records the current material instance
  integer :: last_inst ! records the previous material instance

  ! Was this particle just created?
  logical :: new_particle = .true.
!

Data needed to restart a particle stored in a bank after changing domains

real(8) :: stored_xyz(3)
real(8) :: stored_uvw(3)
real(8) :: stored_distance ! sampled distance to go after changing domain
integer(8) :: prn_seed(N_STREAMS) ! the next random number seed
integer(8) :: xs_seed(N_STREAMS) ! the previously seed used for xs gen

! Domain information
integer :: dd_meshbin ! DD meshbin the particle is to be run in next
integer :: outscatter_destination ! Which domain to transmit particle to

end type Particle

Listing A.2: OpenMC particle buffer data structure.

type ParticleBuffer

sequence

integer(8) :: id
integer(8) :: prn_seed(N_STREAMS)
integer(8) :: xs_seed(N_STREAMS)
real(8) :: wgt
real(8) :: E
real(8) :: mu
real(8) :: last_wgt
real(8) :: last_E
real(8) :: absorb_wgt
real(8) :: wgt_bank
real(8) :: stored_distance
real(8) :: last_xyz(3)
real(8) :: stored_xyz(3)
real(8) :: stored_uvw(3)
integer :: type
integer :: event
integer :: event_nuclide
integer :: event_MT
integer :: n_bank
integer :: surface
integer :: cell_born
integer :: n_collision
integer :: material
integer :: last_material
integer :: last_inst

end type ParticleBuffer
Appendix B

Depletion Nuclides

The set of 206 nuclides used for depletion tally runs was taken from an ORIGEN-S verification and validation report [88].

<table>
<thead>
<tr>
<th>Ag-107</th>
<th>Ce-143</th>
<th>Gd-156</th>
<th>Lu-176</th>
<th>Pd-108</th>
<th>Ru-104</th>
<th>Sn-118</th>
<th>Th-232</th>
<th>Zr-93</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ag-109</td>
<td>Ce-144</td>
<td>Gd-157</td>
<td>Mo-100</td>
<td>Pd-110</td>
<td>Ru-105</td>
<td>Sn-119</td>
<td>U-233</td>
<td>Zr-94</td>
</tr>
<tr>
<td>Ag-111</td>
<td>Cm-244</td>
<td>Gd-158</td>
<td>Mo-94</td>
<td>Pm-147</td>
<td>Ru-106</td>
<td>Sn-120</td>
<td>U-234</td>
<td>Zr-95</td>
</tr>
<tr>
<td>Am-243</td>
<td>Cs-133</td>
<td>Ge-72</td>
<td>Mo-96</td>
<td>Pm-149</td>
<td>Sb-121</td>
<td>Sn-123</td>
<td>U-236</td>
<td></td>
</tr>
<tr>
<td>As-75</td>
<td>Cs-134</td>
<td>Ge-73</td>
<td>Mo-97</td>
<td>Pm-151</td>
<td>Sb-123</td>
<td>Sn-124</td>
<td>U-238</td>
<td></td>
</tr>
<tr>
<td>Au-197</td>
<td>Cs-135</td>
<td>Ge-74</td>
<td>Mo-98</td>
<td>Pr-141</td>
<td>Sb-124</td>
<td>Sn-125</td>
<td>W-182</td>
<td></td>
</tr>
<tr>
<td>Ba-134</td>
<td>Cs-136</td>
<td>Ge-76</td>
<td>Mo-99</td>
<td>Pr-142</td>
<td>Sb-125</td>
<td>Sn-126</td>
<td>W-183</td>
<td></td>
</tr>
<tr>
<td>Ba-135</td>
<td>Cs-137</td>
<td>Ho-165</td>
<td>Nb-93</td>
<td>Pr-143</td>
<td>Sb-126</td>
<td>Sr-86</td>
<td>W-184</td>
<td></td>
</tr>
<tr>
<td>Ba-136</td>
<td>Dy-160</td>
<td>I-127</td>
<td>Nb-94</td>
<td>Pu-238</td>
<td>Se-76</td>
<td>Sr-87</td>
<td>W-186</td>
<td></td>
</tr>
<tr>
<td>Ba-137</td>
<td>Dy-161</td>
<td>I-129</td>
<td>Nb-95</td>
<td>Pu-239</td>
<td>Se-77</td>
<td>Sr-88</td>
<td>Xe-128</td>
<td></td>
</tr>
<tr>
<td>Ba-138</td>
<td>Dy-162</td>
<td>I-130</td>
<td>Nd-142</td>
<td>Pu-240</td>
<td>Se-78</td>
<td>Sr-89</td>
<td>Xe-129</td>
<td></td>
</tr>
<tr>
<td>Ba-140</td>
<td>Dy-163</td>
<td>I-131</td>
<td>Nd-143</td>
<td>Pu-241</td>
<td>Se-80</td>
<td>Sr-90</td>
<td>Xe-130</td>
<td></td>
</tr>
<tr>
<td>Br-79</td>
<td>Dy-164</td>
<td>I-135</td>
<td>Nd-144</td>
<td>Pu-242</td>
<td>Se-82</td>
<td>Ta-181</td>
<td>Xe-131</td>
<td></td>
</tr>
<tr>
<td>Br-81</td>
<td>Er-166</td>
<td>In-113</td>
<td>Nd-145</td>
<td>Rb-85</td>
<td>Sm-147</td>
<td>Tb-159</td>
<td>Xe-132</td>
<td></td>
</tr>
<tr>
<td>Cd-108</td>
<td>Er-167</td>
<td>In-115</td>
<td>Nd-146</td>
<td>Rb-86</td>
<td>Sm-148</td>
<td>Tb-160</td>
<td>Xe-133</td>
<td></td>
</tr>
<tr>
<td>Cd-110</td>
<td>Eu-151</td>
<td>Kr-80</td>
<td>Nd-147</td>
<td>Rb-87</td>
<td>Sm-149</td>
<td>Tc-99</td>
<td>Xe-134</td>
<td></td>
</tr>
<tr>
<td>Cd-111</td>
<td>Eu-152</td>
<td>Kr-82</td>
<td>Nd-148</td>
<td>Re-185</td>
<td>Sm-150</td>
<td>Te-122</td>
<td>Xe-135</td>
<td></td>
</tr>
<tr>
<td>Cd-112</td>
<td>Eu-153</td>
<td>Kr-83</td>
<td>Nd-150</td>
<td>Re-187</td>
<td>Sm-151</td>
<td>Te-123</td>
<td>Xe-136</td>
<td></td>
</tr>
<tr>
<td>Cd-113</td>
<td>Eu-154</td>
<td>Kr-84</td>
<td>Np-237</td>
<td>Rh-103</td>
<td>Sm-152</td>
<td>Te-124</td>
<td>Y-89</td>
<td></td>
</tr>
<tr>
<td>Cd-114</td>
<td>Eu-155</td>
<td>Kr-85</td>
<td>Pa-233</td>
<td>Rh-105</td>
<td>Sm-153</td>
<td>Te-125</td>
<td>Y-90</td>
<td></td>
</tr>
<tr>
<td>Cd-116</td>
<td>Eu-156</td>
<td>Kr-86</td>
<td>Pd-104</td>
<td>Ru-100</td>
<td>Sm-154</td>
<td>Te-126</td>
<td>Y-91</td>
<td></td>
</tr>
<tr>
<td>Ce-140</td>
<td>Eu-157</td>
<td>La-139</td>
<td>Pd-105</td>
<td>Ru-101</td>
<td>Sn-115</td>
<td>Te-128</td>
<td>Zr-90</td>
<td></td>
</tr>
<tr>
<td>Ce-141</td>
<td>Gd-154</td>
<td>La-140</td>
<td>Pd-106</td>
<td>Ru-102</td>
<td>Sn-116</td>
<td>Te-130</td>
<td>Zr-91</td>
<td></td>
</tr>
<tr>
<td>Ce-142</td>
<td>Gd-155</td>
<td>Lu-175</td>
<td>Pd-107</td>
<td>Ru-103</td>
<td>Sn-117</td>
<td>Te-132</td>
<td>Zr-92</td>
<td></td>
</tr>
</tbody>
</table>

149
References


