Tally Derivative Based Surrogate Models for Faster Monte Carlo Multiphysics

by

Sterling M. Harper

B.S., Massachusetts Institute of Technology (2016)
M.S., Massachusetts Institute of Technology (2016)

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

September 2020

Author .....
Department of Nuclear Science and Engineering
August 19, 2020

Certified by .....
Benoit Forget
Associate Department Head and Professor of Nuclear Science and Engineering
Thesis Supervisor

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

Certified by .....
Emilio Baglietto
Associate Professor of Nuclear Science and Engineering
Thesis Reader

Accepted by .....
Ju Li
Battelle Energy Alliance Professor of Nuclear Science and Engineering
Professor of Materials Science and Engineering
Chair, Department Committee on Graduate Theses

Tally Derivative Based Surrogate Models for Faster Monte Carlo Multiphysics

by

Sterling M. Harper

Submitted to the Department of Nuclear Science and Engineering
on August 19, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Nuclear Science and Engineering

Abstract

Existing neutron transport methods used in the nuclear power industry rely on a complex toolchain of modeling and simulation software. Each link in this chain applies various approximations to the spatial, angular, and energy distributions of the problem variables; and these approximations can limit solver predictive capabilities. Monte Carlo (MC) neutron transport is a high-fidelity method that can relax many of these approximations and possibly replace much of the existing toolchain.

However, MC neutron transport is also very slow, particularly when coupled into a multiphysics solver. Some researchers have published runtime costs of over 100 000 cpu-hours to converge a quarter-core multiphysics problem with MC—an expense which makes MC-based tools prohibitive for regular use. In response to this issue, some researchers have developed acceleration techniques using the diffusion-based CMFD (coarse mesh finite difference) method.

This thesis extends that work by coupling the CMFD solver directly to the thermal-hydraulics solvers in a multiphysics simulation. To enable this coupling, MC differential tallies are used to compute the feedback dependence of CMFD parameters. Novel methods based on the windowed multipole cross section representation are used to compute fuel temperature derivatives along with coolant density derivatives. This differential tally approach proves to be flexible; the same procedure is applied to each coarse mesh cell regardless of the presence of control rods, burnable poisons, spacer grids, ^{135}Xe, or other details of the MC model. With the inclusion of a simple pin power reconstruction scheme, these methods create a surrogate neutronics solver capable of bi-directional coupling with thermal-hydraulics. This surrogate can then accelerate multiphysics convergence by reducing the reliance on costly MC simulations.

Furthermore, a novel source-weight clipping procedure is introduced to damp MC-CMFD instabilities. Because this clipping procedure does not require multiple MC generations, CMFD and multiphysics coupling can be performed after each MC generation—even the first generation. This allows simulations to be run with very few MC generations, a feature which alleviates the cost of using many neutrons per MC generation to reduce the impact of fission source distribution undersampling.

This methodology is tested on a quarter-core model of the BEAVRS benchmark, a large pressurized water reactor. Simplified subchannel fluid dynamics, fuel pin heat transfer, and equilibrium xenon solvers are included to form a multiphysics system.

Without the presented acceleration methods, these quarter-core multiphysics simula-

tions using 200 million neutrons per generation are projected to require 3 300 cpu core-hours to reach stationarity. With the presented methods, this cost falls to 270 core-hours. Further results are shown to demonstrate the runtime costs needed to tightly resolve fine-mesh power distributions with projected runtime savings of 6× over prior work.

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

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

Thesis Reader: Emilio Baglietto
Title: Associate Professor of Nuclear Science and Engineering

Contents

Chapter 1

Introduction

1.1 Nuclear reactor multiphysics

Figure 1-1 shows a striking real-world example of multiphysics feedback in the core of a nuclear reactor. This figure shows a series of still images taken from 0.2 s of video focused on the core of a TRIGA reactor. A pulse experiment is shown. A control rod is pneumatically ejected at the beginning of the experiment (some time before these images begin). Following the ejection, the reactor power rises sharply as indicated by the increasing intensity of the

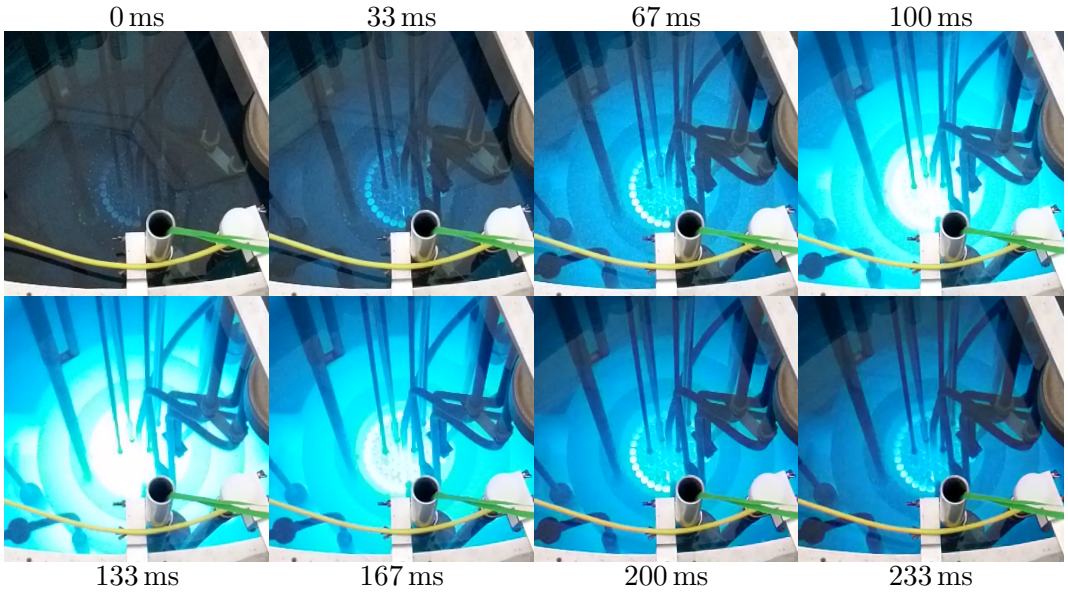


Figure 1-1: Images of a TRIGA reactor pulse at the Jožef Stefan Institute. Visible Cherenkov radiation indicates the reactor power. As the core heats up, multiphysics feedback passively reduces the reactor power. Media credit: Abdulla Al Hajri.

blue-white Cherenkov radiation. Then, without any human intervention, the reactor power decreases quickly.

The spontaneous decrease in power is primarily due to the relationship between neutron flux and fuel temperature. The neutron population causes fission which heats the fuel; the fuel temperature in turn changes the neutron population through a variety of mechanisms.[1] This forms a feedback loop in the physics phenomena that drive the reactor's behavior. In this case, the feedback is negative; a rise in neutron flux leads to a rise in temperature which then reduces the neutron flux.

Multiphysics feedback is not unique to nuclear reactors—rice cookers provide a more common example[2]—but feedback is a particularly complex topic in nuclear engineering. Neutrons are fickle and they can be highly sensitive to the shape, composition, or density of reactor materials as well as the temperature. In high-power reactors, the neutrons can also change each of these variables directly or indirectly. Consequently, modeling the behavior of a high-power reactor usually requires consideration of a broad range of physics disciplines.

Accurate modeling of reactor multiphysics problems is also crucially important because of the implications for the safety and performance of these extremely high-energy-density machines. In an accident scenario, strong negative feedback, like that found in a TRIGA reactor, can prevent destructive power excursions. Positive feedback can do the opposite and lead to grave consequences, as seen in the 1986 accident at the Chernobyl nuclear power plant. Feedback also impacts how easy a reactor is to control, how much power it can produce, how quickly it can respond to changing demands, how long it can operate before refueling, and a host of other operational concerns.

Because of the complexity and consequence of this topic, the nuclear industry pours an extreme amount of work into developing software tools that can efficiently solve these intricate, multidisciplinary physics problems. This thesis will add to that body of work. The contributions focus on the use of Monte Carlo (MC) neutron transport methods with multi-

[1] TRIGA reactors use a U-ZrH hydride metal alloy which forms a combined fuel and moderator material. The neutron scattering properties of the bound-hydrogen moderator are an important part of the feedback [1]. As the hydride temperature rises, the energy spectrum for thermal neutrons hardens which in turn decreases the probability that each neutron leads to a fission event. The Doppler effect also increases the probability of resonant parasitic neutron capture with the low-enriched uranium fuel as the temperature rises.

[2] In a rice cooker, multiphysics feedback is used to switch the heating element to low power when the water boils away. The cooker contains a switching mechanism that is latched by a permanent magnet which is attracted to a ferromagnetic component. When the water boils away, the temperature of the switch rises above the Curie point of the ferromagnetic material which removes the magnetic force and actuates the switch.

physics simulations of large, high-power reactors. To date, MC is very infrequently used for this purpose because it is computationally expensive. However, its use is worth pursuing because its high-fidelity representation of neutron physics brings a host of advantages.

1.2 The value of Monte Carlo neutronics

Much of the complexity in reactor modeling and simulation is due to the physics of neutron transport and how it relates to the temperature, density, and composition of reactor materials. To tackle this challenge, the nuclear power industry typically relies on a complex chain of tools, each of which applies different approximations and aims to resolve one portion of the neutron transport problem.

Knott and Yamamoto describe this conventional toolchain in detail [2]. At one end of the chain is a nodal diffusion solver; this solver uses coarse meshes in space and energy to quickly solve full-core 3D reactor physics problems. The nodal diffusion solver requires parameters (e.g. multigroup cross sections and diffusion coefficients) which are interpolated from a large set of pre-computed values provided by a lattice physics solver. The lattice physics solver itself requires parameters computed by a resonance self-shielding solver. There may also be a pincell calculation and a coupling calculation in between the resonance and lattice physics solvers.

As discussed by Knott and Yamamoto, each of these solvers will explicitly resolve a different subset of the energy and spatial variables. For example, the resonance self-shielding solver has a very fine energy resolution, but it can only consider an infinite homogeneous medium. The lattice physics solver explicitly resolves much of the geometry details (with discrete fuel, cladding, and absorbers), but it only considers 2D geometry features and it generally only solves one fuel assembly at a time. It also uses a coarser energy discretization of a few hundred groups. The nodal method captures the full 3D geometry, but it represents this geometry as large (e.g. roughly 10 cm on a side) homogeneous blocks and uses very few energy groups (e.g. 2).

A large number of approximations and corrections are applied throughout this process. For example, the resonance self-shielding solver can generate multigroup cross sections from basic nuclear data, but it typically assumes that resonances from different isotopes do not overlap. Consequently, an extra calculation step is included to compute resonance interfer-

ence factors. For another example, it is generally prohibitively slow to resolve anisotropic neutron scattering in a lattice physics solver so they instead typically assume isotropic scattering and partly account for the forward-peaked nature of scattering by artificially reducing the total and scattering neutron cross sections.

The point of this discussion is to say that the existing industrial neutronics toolchain is complex in the extreme. The cost of this complexity is difficult to quantify, but imagine the challenge it poses to a startup company that aims to develop a revolutionary new reactor or fuel design. Are the standard approximations valid for this new system? How should the energy group structure change for a ^{233}U-based reactor? Is a different energy group structure needed for uranium nitride fuels with silicon carbide cladding? Are 2D lattice physics solvers valid for twisted cruciform fuel rods? In a business environment, these types of questions can deplete precious time and money. These questions can also be completely eliminated by using a Monte Carlo neutronics solver.

Monte Carlo solvers offer the possibility of replacing the entire industrial toolchain with one comparatively simple tool. This means no tuning of energy group structures. No resonance interference factors. No Dancoff factors, Bell factors, or intermediate resonance parameters. No transport corrections, no discontinuity factors, and no leakage correction. Removing material about all these approximations needed for the existing toolchain would probably turn most tome-like neutronics textbooks into afternoon paperbacks. As a consequence of this simplicity, Monte Carlo solvers are easier to understand, easier to validate, and easier to modify. This offers great promise in terms of reducing nuclear research and development costs, but there is one catch: Monte Carlo solvers are very slow.

1.3 Making Monte Carlo neutronics practical

Monte Carlo solvers are desirable for their fidelity, but their computational cost poses challenges. MC can be used without other neutronics methods to simulate a large full-core reactor, but it requires large computer clusters and long runtimes.

For example, a 2017 study by Kelly et al. used a MC solver for a quarter-core multi-physics simulation of a large pressurized water reactor at beginning-of-cycle conditions [3]. This simulation required 7 days of wall-clock time with 1,024 cpu cores (170,000 core-hours). This runtime cost is acceptable for occasional research or validation studies, but it is pro-

hibitive for regular use. Nuclear utilities and fuel vendors might perform thousands of calculations for each reactor fuel cycle and expending so much computational effort on each calculation is simply infeasible. For this reason, more practical applications of MC require that it be used along with a simpler coarse mesh method.

There are different approaches to combining MC and coarse mesh methods for reactor modeling. One approach is to use a MC solver much like a traditional lattice physics solver, pre-generating large tables of parameters for use in a nodal diffusion code. The research team behind the Serpent MC code has recently championed this approach [4], and they have demonstrated parity in accuracy relative to the traditional neutronics toolchain for simulations of an operating power plant [5]. However, these methods still require many approximations relating to homogenization and transport effects due to the inclusion of a traditional nodal diffusion solver.

An alternate approach is to use the coarse-mesh diffusion and MC methods in tandem. The full problem is represented in MC, and the coarse mesh parameters are generated during the calculation using the MC solver. The techniques for using a coarse mesh finite difference (CMFD) diffusion calculation concurrently with higher-fidelity methods have been well-established, and these techniques have been used to accelerate a variety of higher fidelity solvers such as method-of-characteristics (MOC) or simply higher order diffusion [6, 7]. Crucially, these methods are carefully designed so that the use of CMFD accelerates the overall calculation without introducing a bias; the accuracy of the simulation is determined by the high-fidelity solver. These methods were originally used with deterministic solvers but have recently been applied to calculations that use MC as the higher-fidelity method [8].

Although combined CMFD-MC solvers have successfully been applied to full-core reactor calculations [8], they have rarely been applied to problems that include multiphysics feedback. Likely, this is due in part to stability issues with the CMFD-MC method. CMFD-MC instability can easily arise without feedback [8, 9], and multiphysics only complicates the issue. A related issue is the difficulty in making the CMFD solver responsive to multiphysics feedback. The CMFD solver relies on parameters computed by MC tallies, and there are no well-established on-the-fly methods for computing how these parameters vary with temperature and other multiphysics variables.

Herman has previously tackled this challenge of using MC and CMFD methods on a large reactor with multiphysics feedback [10]. To address the instabilities, Herman used

under-relaxation in the form of moving-window MC tallies. Herman also used a machine learning approach so the temperature and density dependence of CMFD parameters were "learned" on-the-fly during simulations, which further reduced the impact of multiphysics instabilities. The resulting solver demonstrates an impressive reduction in the total number of MC generations needed to converge the multiphysics problem.

This thesis adopts a similar approach, and aims to address some of the downsides of Herman's methodology.

1.4 Contributions of this thesis

The primary objective of this thesis is to demonstrate efficient and general coupling techniques for a multiphysics solver with combined MC and CMFD diffusion neutronics solvers.

The key contribution is in the use of MC differential tallies to compute feedback-dependent parameters for CMFD diffusion. These tallies improve upon the machine learning method used by Herman [10] in that they do not require a user to specify and tune a feature vector. This makes the differential tally approach both easier to use and more general. In this work, the CMFD diffusion solver combined with feedback-dependent parameters computed via differential tallies is essentially used as a surrogate for the MC solver in some iterations which accelerates the convergence of the multiphysics problem. Because the surrogate parameters are periodically updated using high-fidelity MC results, some approximations can be made in the computation of derivatives, diffusion coefficients, etc. without biasing the final solution.

A further contribution of this work is a collection of MC software improvements which have been added to the publicly-available OpenMC code [11]. Chief among these are an efficient shared-memory neighbor list scheme and a geometry search tree procedure. These methods improve MC ray tracing performance, and they significantly reduce the runtime cost for multiphysics geometries. Additionally, a myriad of small usability features have been added such as rectilinear tally meshes which can better resolve geometry features.

This thesis also addresses the common practice of accumulating MC tallies over multiple generations. This is a popular but ineffective means of reducing stochastic errors. Here, the tallied values used by the coupled CMFD diffusion, heat transfer, and fluid solvers are accumulated over only one MC generation at a time while using many (e.g. 200 million)

source neutrons per generation. Multiple MC generations are used, but the tallies are reset after each generation. Overall, fewer simulated neutrons are needed to resolve tallied quantities with this approach.

This methodology is tested against a steady-state quarter-core problem of a large pressurized water reactor (PWR) and includes feedback due to fuel temperature, coolant temperature, coolant density, and ^{135}Xe concentration. A simplified subchannel fluid dynamics solver is used to compute coolant conditions, and a simplified heat transfer solver is used to compute the temperature distribution within each fuel rod. The details of the computed results are compared to data measured from an operating power plant.

In order to make this project achievable over the course of a PhD thesis, some physics phenomena have been left outside the scope of this work. Probably the most important of these phenomena for the benchmark considered here is fuel nuclide depletion. Consequently, the presented solver does not achieve parity in accuracy with industrial solvers. Nevertheless, the impacts of these phenomena are discussed, and including them in future work is not expected to change the core conclusions of this thesis.

1.5 Layout of this thesis

Chapter 2 describes the subchannel fluid dynamics solver implemented for this work. This chapter provides some background on subchannel methods and how they contrast with typical CFD (computational fluid dynamics) solvers. There is also some discussion of the turbulent phenomena that are important for distributing energy throughout the reactor coolant, and how it can be treated with simplified models. Chapter 3 similarly describes the heat transfer solver.

Chapter 4 describes the neutron transport methods used here. Some background is provided on recent advances in MC methods that enable its use for large multiphysics problems like the one tackled here. Differential tallies are discussed in detail along with techniques for their use. The methodology for the CMFD diffusion solver is also described in this chapter.

Chapter 5 focuses on the coupling of these disparate solvers. Details are provided on the geometry model, and how data from each solver is mapped for use in the others. This chapter also touches on the software architecture of the project.

Simulation results focusing on a single 3D fuel pin are presented in Chapter 6. On this small problem, the presented methodology can be compared against other approaches. This chapter demonstrates that increasing the number of MC particles used per generation is far superior to accumulating tallies over multiple MC generations in terms of error reduction. The performance of the accelerated algorithm is also shown for the fuel pin problem, and the implications of the performance are discussed.

The newly-developed methodology is applied to a quarter-core problem in Chapter 7. Details of the convergence behavior are given, and the runtime is compared to other published works.

Chapter 2

Subchannel Solver

This chapter focuses on the fluid dynamics portion of the reactor multiphysics problem. A fast and accurate fluids solver has proven to be difficult to develop, but the process was made easier by limiting the scope of this work to single-phase, steady-state normal operating conditions in a PWR.

The discussion will begin with a comparison of the subchannel discretization method against typical CFD (computational fluid dynamics) solvers. This serves as a useful description of the subchannel approach as well as justification for why it was chosen over CFD for this work. In short, the subchannel method uses a coarser discretization which makes it faster, easier to implement, and easier to use; but it is more limited to the bounds of experimental data.

Some of that experimental data is then highlighted in a following section. In particular, researchers have observed that turbulent flows are crucial for distributing energy in rod bundle geometries. They also find that the turbulence in this geometry is not well described by the simple isotropic models typically used for channel flow. Consequently, subchannel models must be tuned to experiments that are closely representative of the target problem (rod bundles with a similar pitch-to-diameter ratio and Reynolds number).

Finally, this chapter will describe the methodology behind the simple solver implemented for this thesis. A comparison to experimental data is shown, and the model parameters are tuned to match this data.

2.1 CFD versus subchannel solvers

Subchannel solvers use a finite volume technique for numerical discretization. A typical volume is sketched in Figure 2-1. Each volume is an axial segment of a subchannel—the coolant region that lies between adjacent fuel rods, guide tubes, and core baffles. The mass, momentum, and energy conservation equations are integrated over these volumes and solved numerically.

Compare this to a typical CFD mesh shown in Figure 2-2. Here the subchannel region is divided up into many smaller areas. This particular mesh shows 84 cells in the xy-plane per subchannel.

The key differences between subchannel and CFD solvers can be explained in part by these different meshes. The fine mesh allows a CFD solver to explicitly resolve the fine distributions of velocity, temperature, etc. within a subchannel. The blessing of CFD is that it *can* resolve these fine distributions, but the curse of CFD is that it *must* resolve them.

A good example to clarify this point is the effect of wall shear on the coolant axial momentum. In a subchannel solver, this is computed using a Darcy friction factor that is parameterized with a flow Reynolds number [13]. Models for the friction factor must be fitted to empirical results, and there is no guarantee that models fitted to circular pipe flows, for example, are appropriate for rod bundle subchannels.

In contrast, CFD solvers explicitly resolve much of the fluid shear which removes the need for such models. Note the presence of fine mesh cells directly next to the fuel rods in

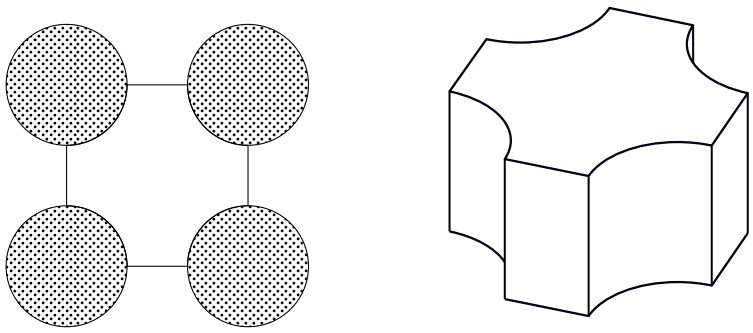


Figure 2-1: Top-down and isometric views of a subchannel volume. This subchannel fills the space between four fuel pins.

Figure 2-2 aimed at resolving this effect. This makes CFD solvers more general, but also more expensive because they must explicitly compute solution values for each of these fine mesh cells.[1]

A discussion of CFD would be incomplete without a mention of turbulence models. A wide variety of turbulence modeling approaches are used in CFD solvers with huge impacts on accuracy and computational expense. These approaches can be broadly classified into DNS (direct numerical simulation), LES (large eddy simulation), and RANS (Reynolds-averaged Navier-Stokes). The fidelity of the method is inversely related with the number of proper names in its acronym (counting Eddy as a name).

Studying the meshes published for use with the different solvers is a good way to quickly understand how they differ. The mesh shown in Figure 2-2 was built for a RANS simulation. Contrast this mesh with one used for LES shown in Figure 2-3. The LES mesh is much finer which allows resolution of smaller turbulent structures, but it brings a consequently

[1] CFD simulations typically use *wall models* to handle unresolved near-wall effects, but these models are more generally applicable than the coarse Darcy friction factor models.

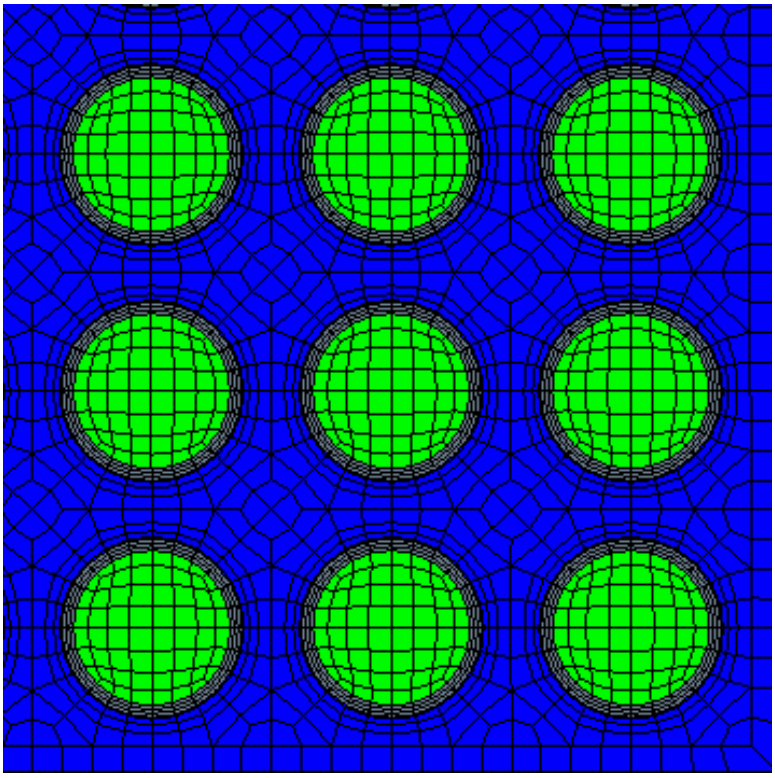


Figure 2-2: An example CFD (RANS) mesh for a portion of a PWR assembly. Image from [12].

higher computational cost.

For perspective on the cost of LES, consider the work of Bieder et al. who produced the mesh shown in Figure 2-3 and used it to simulate a rod bundle experiment [14]. This system is far smaller than a PWR core. The setup was a 5×5 rod-bundle with a pitch and diameter similar to a typical PWR fuel assembly. The test section is 1.2 m in length (compare to about 4 m for a fuel assembly) and run with coolant achieving a Reynolds number of about 100 000 (compare to 500 000 for a typical PWR). Even this small problem required 1.7 million cpu-hours!

Because of this expense, full-core simulations with LES are out of reach with very few exceptions. One of those exceptions is an effort under the DOE's Exascale Computing Project to simulate a full SMR (small modular reactor) core with LES [15]. Even with the power of exascale computing, some modeling simplifications will need to be made such as neglecting subcooled boiling and not explicitly meshing spacer grid mixing vanes. Also note that this exascale project is focused on the simulation of the NuScale SMR which is smaller than typical PWRs and has a 10× lower Reynolds number. LES simulations of something like a 4-loop Westinghouse reactor are therefore far off.

In contrast, full-core simulations with RANS CFD solvers might be practical for research purposes and infrequent industrial calculations. For one example, Kochunas et al. used a RANS solver for a steady-state quarter-core multiphysics simulation of a Combustion

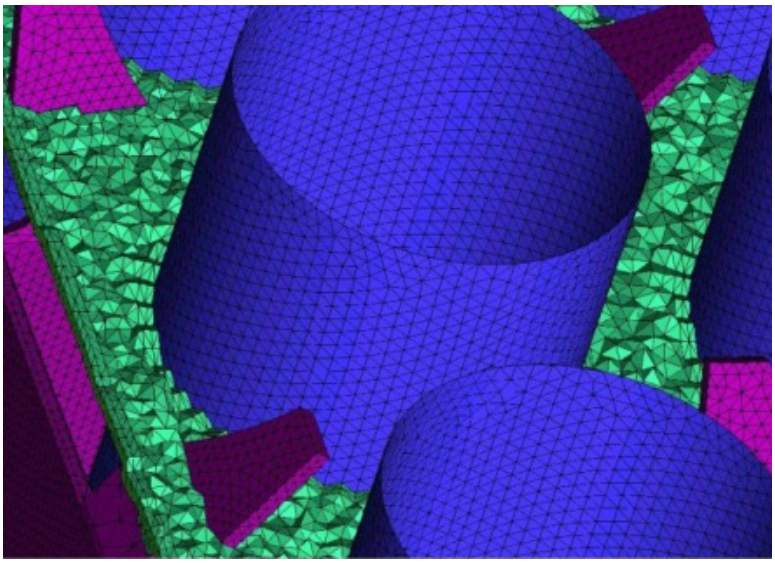


Figure 2-3: An example LES CFD mesh for a portion of rod bundle experiment. Note the explicitly modeled mixing vanes. Image from [14].

Engineering PWR, and they found a cost of 8,500 cpu-hours [12].

However, caution must be taken with RANS solvers as they are susceptible to errors introduced by approximations in turbulence models. Most RANS solvers use 2-equation isotropic eddy viscosity models, and many researchers have shown these models to be inaccurate for use in rod bundle geometries. Bieder et al. offer a comparison of such a RANS simulation against LES and experimental data in the wake of a grid spacer with mixing vanes [14]. Their results show that RANS captures the large scale flows induced by the mixing vanes, but misses much of the fine detail within each channel that may be important for mixing between channels. Others have reported that some rod bundle geometries generate large, regularly-occurring vortex structures that requires anisotropic eddy viscosity models and unsteady RANS (URANS) to capture [16, 17], but these techniques have not yet seen widespread adoption.

Two-phase fluids are also very difficult to simulate with CFD, and this is an active research field [18]. The two-phase limitation is not an issue for the single-phase conditions considered here, but it would limit future extensions of this work.

Finally, it is worth noting that mesh generation is a further non-trivial complication of CFD over subchannel solvers. Mesh generation is difficult not just because the user must precisely define all relevant surfaces in contact with the fluid, but because the mesh cells must properly resolve gradients, obey the CFL condition (Courant–Friedrichs–Lewy condition), and complement the chosen wall shear model. The mesh fineness is also likely limited by constraints on computation time and memory footprint.

All this is to say that CFD for reactor multiphysics is desirable but very difficult at present. The complexities of turbulence modeling and mesh tuning make it difficult to use for a non-expert like myself, and it is costly in terms of both computation time and user burden. For these reasons, the subchannel method is used here.

2.2 Rod bundle crossflow

The lynchpin of the subchannel method is in the choice of crossflow models. The flow in a fuel assembly is predominately axial, but this part of the flow is easy to solve for with 1D fluid methods treating the assembly as a heated pipe. However there is a great deal of turbulent mixing between adjacent subchannels that efficiently transfers energy from one

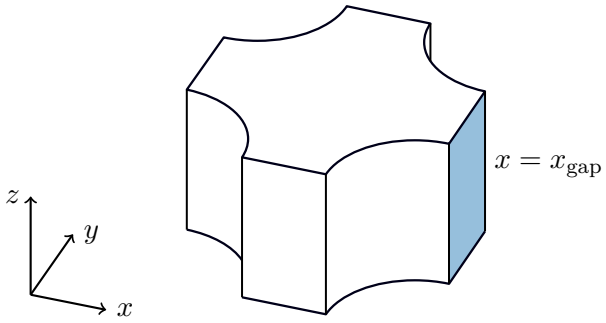


Figure 2-4: Coordinate system used for modeling a single subchannel gap. The gap of interest is highlighted.

channel to another. Boiling, channel blockage, and mixing vanes can also directly divert fluid from one channel to another which redistributes energy.

This section will briefly discuss the physics of crossflow, experimental observations, and the models employed by subchannel codes. Ultimately, an extremely simple model will be adopted here that is unsatisfying given the complexity of the underlying physics. Nevertheless, this simple model is expected to be close enough for the narrow case considered here (steady-state, single-phase, Re $> 10^5$ PWR flows) given the main objective of studying multiphysics iteration schemes.

2.2.1 Turbulent energy transfer

Consider the flow between two adjacent subchannels. Let z be the axial direction, x be the direction between the subchannel centers, and y be the orthogonal direction. The boundary between two subchannels is a yz-plane. This coordinate system is sketched in Figure 2-4.

Subchannel derivations conventionally use the variable W to represent the mass flow per unit length in z through a gap between subchannels. Using the stated coordinate system and specifying a gap located at x_{gap} that spans from y_0 to y_1, W can be computed from the integral,

$$W(t, z) = \int_{y_0}^{y_1} \rho(t, x_{\text{gap}}, y, z) v_x(t, x_{\text{gap}}, y, z) \mathrm{d}y \tag{2.1}$$

where ρ is the fluid density and v_x is the component of the velocity in the x-direction (the direction across the gap). Integrating W over z gives a mass flow rate (example units: kg/s) of coolant from one channel to the other.

Similarly, the transfer of enthalpy per unit length can be computed as,

$$H(t,z) = \int_{y_0}^{y_1} h(t, x_{\text{gap}}, y, z)\rho(t, x_{\text{gap}}, y, z)v_x(t, x_{\text{gap}}, y, z)\mathrm{d}y \tag{2.2}$$

where h is the specific enthalpy.

There are two conceptually distinct flow phenomena with different impacts on W and H. If there is a large, steady flow of mass across the gap then an integral of W in time is non-zero. That steady flow will also carry a proportional amount of energy across the gap,

$$\left| \int W(t,z)\mathrm{d}t \right| \gg 0 \qquad \text{and} \qquad \int H(t,z)\mathrm{d}t \approx h_{\text{avg}} \int W(t,z)\mathrm{d}t$$

However, a covariance between h and v_x can decouple H from W and lead to a case where there is significant heat flow even without significant mass flow,

$$\int W(t,z)\mathrm{d}t \approx 0 \qquad \text{and} \qquad \left| \int H(t,z)\mathrm{d}t \right| \gg 0$$

Essentially, turbulent flow draws hot fluid from one channel and replaces it with an equal amount of cold fluid from the other channel, resulting in a net transfer of energy without a net transfer of mass. Measurements show that this turbulent energy transfer is significant for rod bundle geometries. Some of these measurements will be described in the following subsection.

2.2.2 Rowe and Angle crossflow experiments

The COBRA subchannel solver was published in 1967 by Rowe and Angle as a two-part technical report. The first part described the code and methodology [13]. The second described a heated two-channel experiment that COBRA was compared to [19]. One of the experimental cross sections from that report is shown in Figure 2-5.

One interesting take-away to note before diving into the details is that Rowe describes significant difficulties in making a stable solver. This has also been my experience. However, Rowe and Angle also describe significant physical instabilities that occurred in the experimental setup. Some of these instabilities even damaged the equipment. This suggests that a developer should not be disheartened by fighting instabilities in their subchannel solver—if the experimentalists must struggle against instability then it is only fair that the

theoreticians face the same difficulties!

Rowe and Angle's measurements included channel outlet mass flow and enthalpy as a function of heat flux. One of their plots is included here in Figure 2-6. The upper half of the plot shows the enthalpy rise of each subchannel. Note that at the highest heat fluxes shown here, the smaller channel just barely reaches saturation. The lower half of the plot shows the exit mass flow of each channel. When the coolant is well below saturation, the balance of mass flow between the channels is very nearly invariant with heat flux.

For the case in Figure 2-6 at 0.53×10^6lb/hr-ft^2 the outlet temperature between the channels differs by about 35 K. This will result in a lower density, and correspondingly higher velocity in the hot channel which leads to more frictional force. Hence, a change in the ΔP between the channels should be expected. Despite this, there is no significant change in the outlet flow of either channel relative to the unheated case. This suggests that there is little net mass exchanged between the channels under single-phase conditions, i.e. small net W.

However, as previously discussed, H can still be large despite a small W. The data published by Rowe and Angle suggests that this turbulent mixing effect is significant. The upper sub-plot of Figure 2-6 shows how the empirical results compare to COBRA simula-

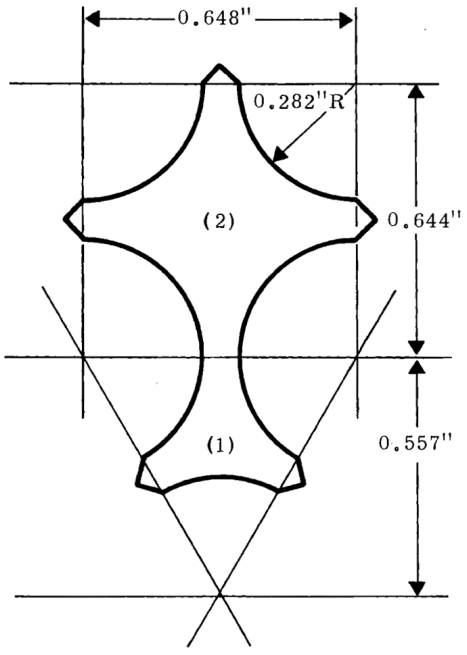


Figure 2-5: The cross section of the crossflow experiment published in [19]

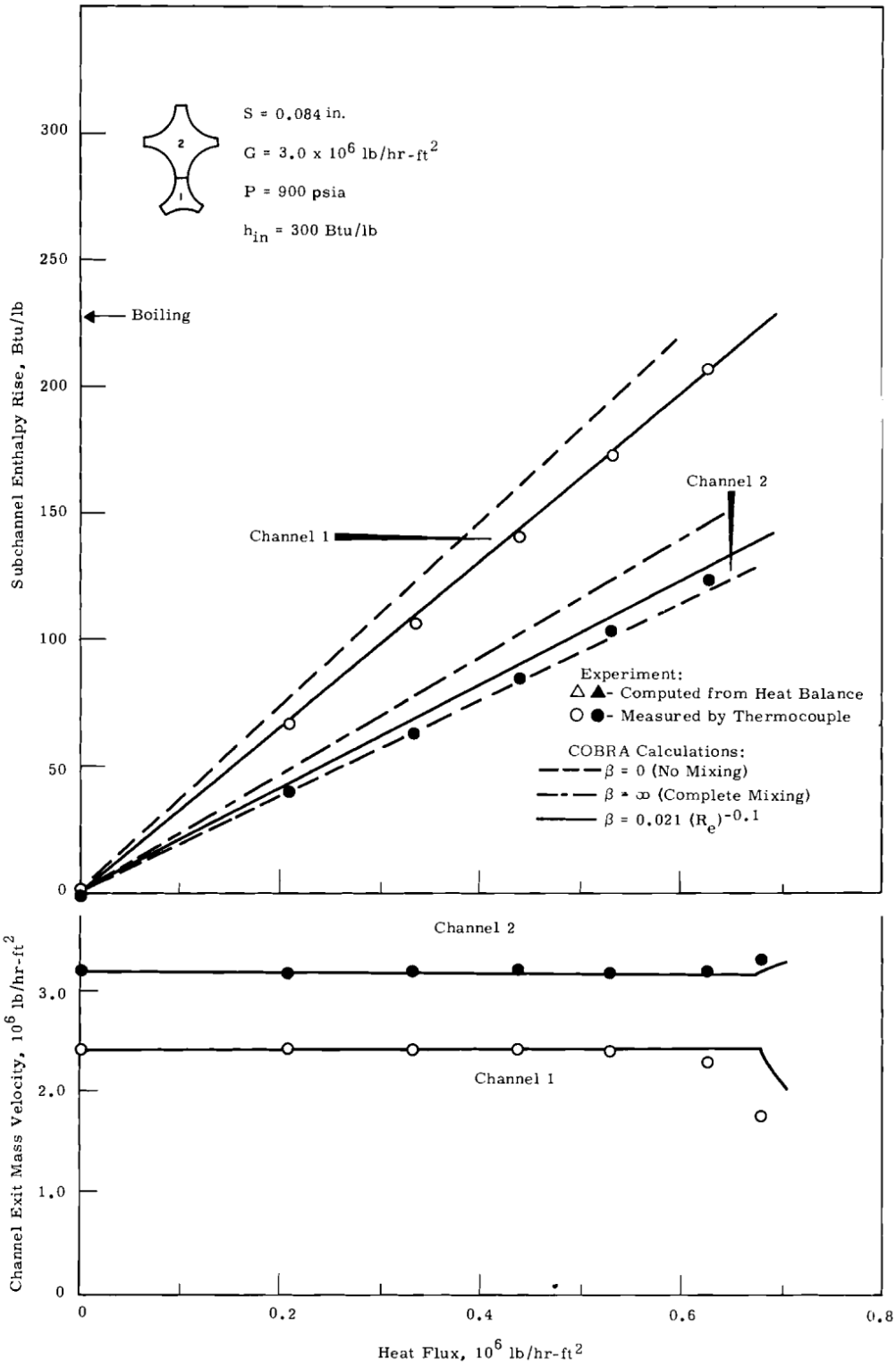


Figure 2-6: A plot from [19] showing enthalpy rise and outlet mass flow in the two experimental subchannels as a function of heat flux.

tions that use different mixing assumptions. The dashed lines assume there is no turbulent exchange of energy, and they do not match the data in the hot channel well. The solid lines use a mixing model fitted by Rowe and Angle that models the data better. (The cold channel data in this case is closer to the no-mixing assumption, but this is an exception to the general trends seen in [19] and elsewhere.)

2.2.3 Eddy diffusion models

Experiments like those of Rowe and Angle show that turbulent heat transfer is significant so practical models must be developed for it. The effect of turbulence is often modeled as a linear diffusion process, i.e. Fick's law. In the context of fluid dynamics, this model is referred to as the Boussinesq assumption. Applied to the heat transfer across a rod bundle gap, this diffusion approximation gives the relationship,

$$H_{ij,\text{turb}} = -\epsilon_h s \frac{\partial h}{\partial x} \tag{2.3}$$

where s is the width of the gap (the minimum distance between adjacent rods) and ϵ_h is the eddy diffusion coefficient for enthalpy.

Unfortunately, it turns out that blindly using Fick's law to model transport is as much of a sin for fluid dynamics as it is for neutronics. It is a useful approximation, but it has limitations that are significant for reactors.

Fick's law was originally used to describe the diffusion of particles in a fluid, and it can be derived from kinetic theory by making the approximation that the characteristic length for particle interactions—the mean free path between collisions—is small compared to the length scales over which variables like h change significantly. Instead of a particle random walk process, the diffusion considered here is caused by turbulent eddies in the fluid. The particle mean free path from kinetics is analogous to the diameter of a turbulent eddy. Applying Fick's law assumes that these eddies are small relative to gradients in h which is not true for reactor coolant conditions. This will be clear from LES simulation results shown later in this section. Nevertheless, the linear model is a useful approximation, and it is frequently used to describe experimental results.

Note that for literature relating to the issues with the Boussinesq assumption, it is better to search for publications relating to momentum transport rather than heat transport. The

underlying mechanisms are essentially the same, but far more literature is available for momentum transport. For momentum, the ϵ parameter is called the eddy viscosity. The most common RANS turbulence models assume that this parameter is isotropic. However, many researchers have demonstrated that eddy viscosity in rod bundles is anisotropic, and finding models for the eddy viscosity is an active field of research [16, 17].

There is another issue related to the difficulties with the ϵ_h parameter. Implementing the eddy viscosity model as a finite difference gives,

$$H_{ij,\text{turb}} \approx -\epsilon_h s \frac{h(x_2) - h(x_1)}{x_2 - x_1} \tag{2.4}$$

where x_1 and x_2 are arbitrarily chosen evaluation points. Given that ϵ_h can vary greatly within a subchannel, care must be taken to ensure the model for ϵ_h matches the choice of x_1 and x_2.

For further insight into the complexities of turbulence modeling, it will be helpful to describe two turbulent flow phenomena found in rod bundle geometries: mean secondary flows within a subchannel and cross-gap flow pulses.

2.2.4 Mean secondary flows

Turbulent flow through non-circular geometries like a reactor subchannel tends to generate steady secondary flows in the xy-plane. For a reactor geometry, these secondary flows take the form of vortices that pump fluid from the subchannel center towards the gaps. Figure 2-7 shows computational results that illustrate the shape of these secondary flows.

Be careful not to draw many conclusions about the average flows from Figure 2-7. This plot shows instantaneous velocity, $\overline{v} + v'$, which partly obscures the mean secondary flow, $\overline{v}$. That said, the four vortices adjacent to the gap at the center of the image show the expected shape of the mean secondary flow.

The implication of these secondary flows is that the specific enthalpy, $h(x)$, will vary slowly along the flow lines. Fluid from the center of each subchannel is moved closer to the gap by the secondary flow, effectively reducing the distance $x_2 - x_1$ used in Equation 2.4 and increasing the heat transfer across the gap.

These secondary flows can complicate the issue of modeling heat transfer as a function of bundle pitch. A simple application of Equation 2.4 would find that decreasing pitch

by 20% would increase the heat gradient by 20% and consequently increase the heat flux across the gap proportionally. But decreasing the pitch would also decrease the size of the secondary vortices (if they are still present) and counteract the change in gradient. Hence, a diffusivity value, ϵ_h, measured with one lattice pitch likely cannot be used with another pitch.

2.2.5 Large cross-gap pulses

Mean secondary flows play an important role in subchannel mixing, but as discussed previously, heat can still be carried by transient effects in the absence of a steady flow.

Figure 2-8 shows the instantaneous velocity at a different subchannel gap from the same LES results shown in Figure 2-7. Instead of the usual mean secondary flow vortices, this plot shows a large pulse of flow across the gap. Due to the problem symmetry, this flow must be transient and balanced at other times by flow in the opposite direction.

Under some conditions these pulses are very strong and occur in a regular pattern, much to the surprise of the researchers who first observed them. One early report of these pulses can be found in a 1970 article by Van Der Ros [20].

Van Der Ros performed experiments on a pair of square channels connected by a small gap. One channel was heated, and mixing was inferred from the enthalpy rise in the other

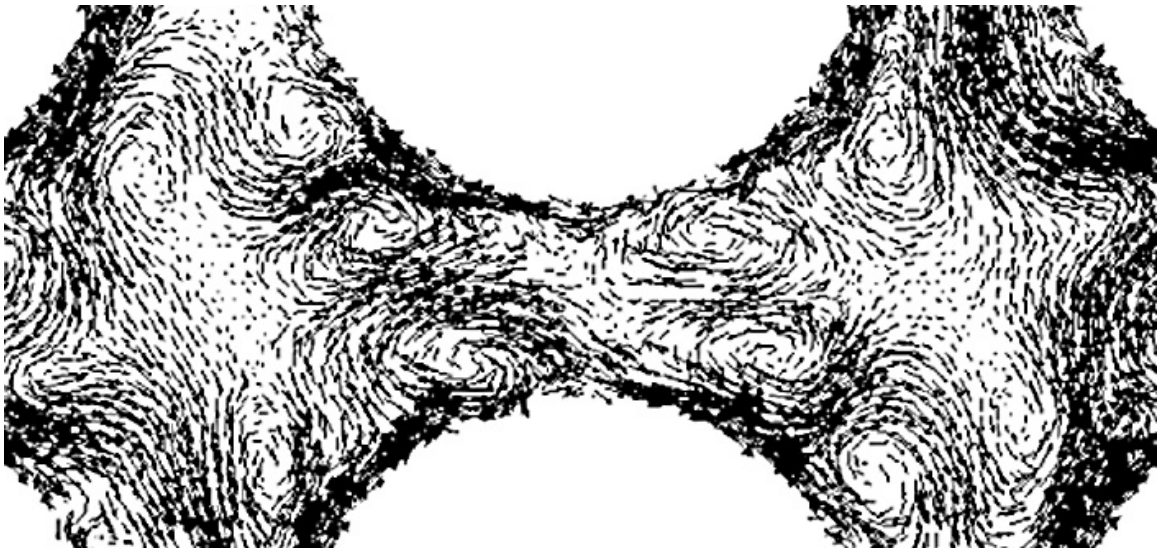


Figure 2-7: Plot of instantaneous velocity from an LES simulation of a bare rod bundle. Vortices can be seen near the gap that are representative of the mean secondary flows. Image from [14].

channel. Figure 2-9 is a plot of the measured heat transfer as a function of bulk Reynolds number for both a 1mm and 2mm gap. The y-axis of this plot is proportional to the total heat flow across the gap, not the heat flux. Note the counter-intuitive result that between 6 000 and 17 000 Re, doubling the cross-sectional area of the gap has no impact on the sum flow of heat through that area.

Van Der Ros writes [20],

> "The increase in the slope of the 2 mm line at Re above 17 000 indicates a different mechanism of enthalpy exchange between the subchannels. Besides these data, visual observations and fast thermocouple signals clearly indicate that at Re $<$ 17 000 the transport mechanism for the mass- and heat-exchange during radial transport through the gap can not be explained by eddy diffusivity, although the data have been analysed as such. The thermocouple signals at locations outside the centre gap are random turbulent but the recordings as well as the visible waves are very regular in amplitude and frequency for locations inside the gap."

A 2010 review paper by Meyer lays out the history of observations like these and the development of an explanatory mechanism [21]. It is a fascinating story because reports of these flow pulsations were largely ignored for decades. Researchers tried unsuccessfully to explain rod bundle mixing purely with mean secondary flows even when these regular pulsations turned out to be the dominant mechanism. It was not until the 1990s that a

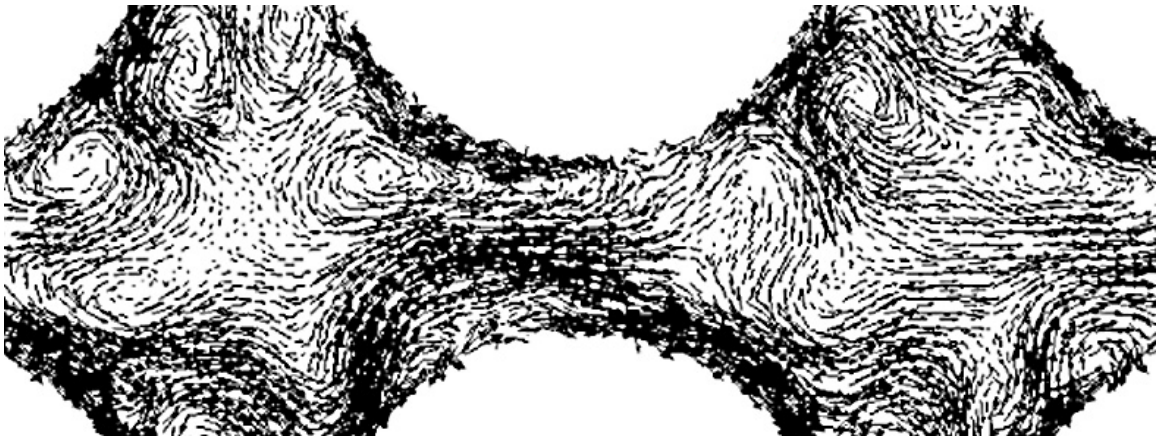


Figure 2-8: Plot of instantaneous velocity from an LES simulation of a bare rod bundle. A large pulse of cross-gap flow can be seen. Image from [14].

consensus was built on the existence and mechanism of these pulsations. As an interesting side note, Meyer lists several cases where a researcher arrived at these findings early, but left their results in internal reports.

The mechanism underlying the flow pulses as described in Meyer's review is that of large vortices in the xz-plane that carry fluid into and out of the gap. See Figure 2-10 for a sketch of the vortex structure. The size of the vortices is on the order of the fuel pin radius. They come in nearly regular, counter-rotating trains slightly offset from the gap center and travel axially with the primary flow of the fluid.

For a fixed point in the gap, these passing vortices look like large, regularly-occurring pulses of cross-gap flow. See Figure 2-11 for experimentally measured data showing this behavior. The lower subplot in that figure shows the cross-gap flow with the nearly regular pulses. The peak values of this fluctuating cross-gap velocity are about 20% of the subchannel average axial velocity. For reference, mean secondary flows are roughly less than 2% of the axial velocity in reactor subchannel geometries.

These regular vortices occur more readily in bundles with small P/D (pitch-to-diameter

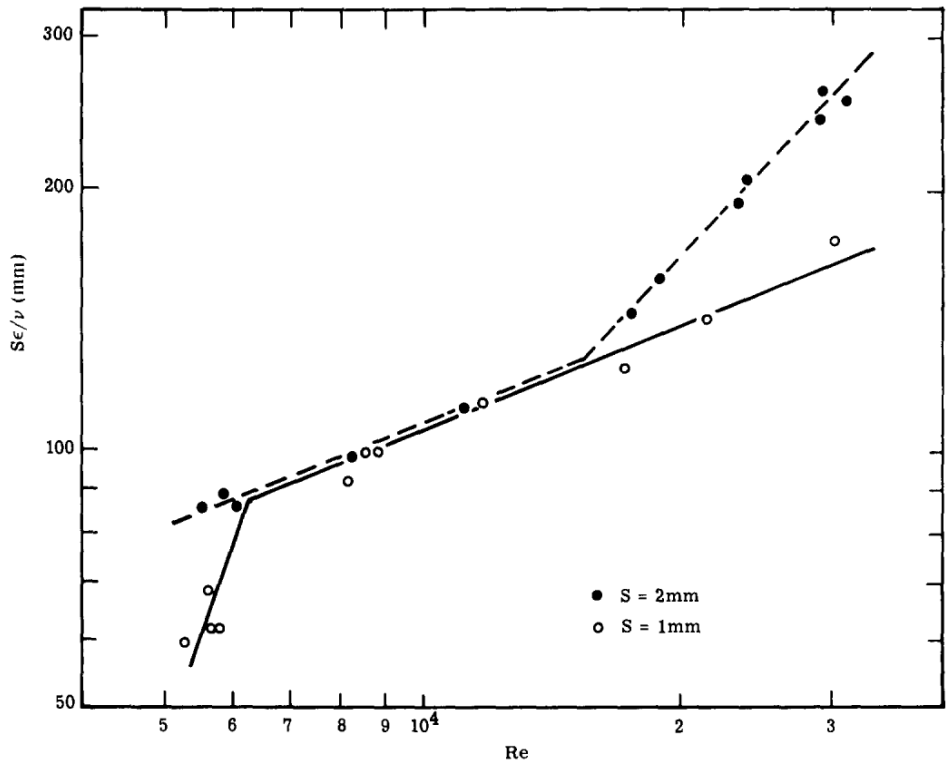


Figure 2-9: Measured heat flow across subchannel gaps as a function of bulk Reynolds number. Plot from [20].

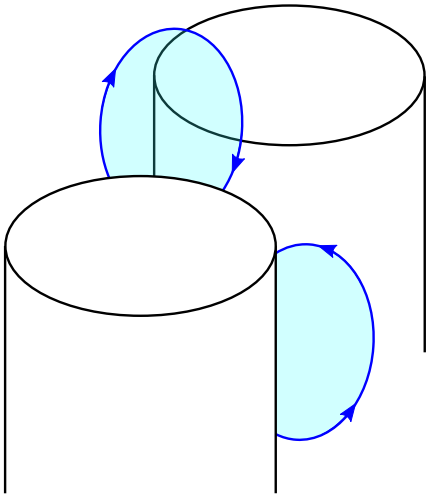


Figure 2-10: A diagram qualitatively showing the structure of the vortices that develop in subchannel gaps.

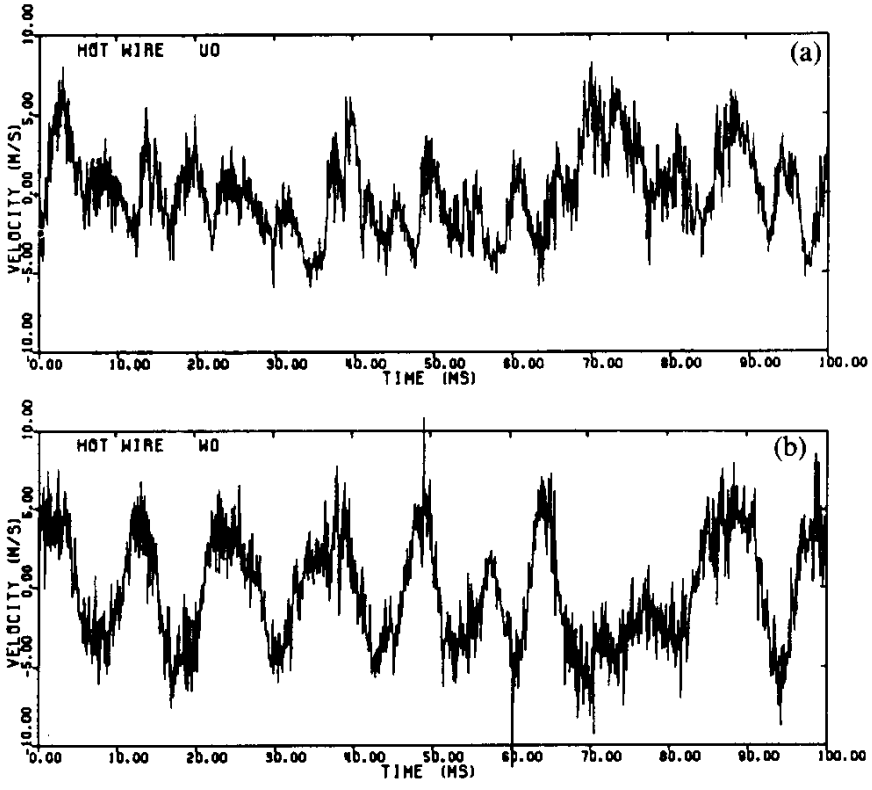


Figure 2-11: Axial (a) and perpendicular (b) velocities as a function of time measured via hot wire anemometry in the middle of an experimental subchannel gap. Plot from [22]. For reference, the subchannel average velocity is 25.5 m/s and the average axial velocity in the gap is 20.8 m/s.

ratios) so they probably do not occur in typical PWRs. Rowe found regular pulses in a bundle with a $P/D = 1.1$ but not $P/D = 1.25$ (Re between 50 000 and 200 000) [23]. Möller found regular pulses with $P/D = 1.224$ at 150 000 Re [22]. Neither Rowe nor Möller identified a high Reynolds number cutoff for the regular pulses, but one should be expected from Van Der Ros [20]. Consequently, these regular vortices are probably not present in typical PWRs with $P/D = 1.38$ and Re = 500 000.

Nevertheless, even irregular pulses likely contribute significantly to the thermal mixing between subchannels. They will also generate highly anisotropic transport that is difficult to model with simple eddy diffusivity.

2.2.6 Turbulent energy transfer model

Instead of an eddy diffusivity model, turbulent enthalpy exchange between subchannels i and j is modeled here as,

$$H_{ij,\text{turb}} = -\beta s \frac{G_i A_i + G_j A_j}{A_i + A_j} (h_i - h_j) \tag{2.5}$$

where β is a dimensionless fitted parameter, s is the width of the gap, G is the mass flux in each subchannel, A is the subchannel area, and h is the subchannel average specific enthalpy. This model is roughly the same model used by Rowe [13] and other subchannel developers.

The value of β will be fitted to experimental rod bundle mixing data with the appropriate geometry (discussed in Section 2.4). The distance between subchannel centers varies little in a typical PWR core so the issue of parametrizing β to this distance can be ignored.

Rowe and Angle find that this parameter, β, depends very weakly on Reynolds number, proportional to $\text{Re}^{-0.1}$ [19]. The recently filed Watts Bar Unit 2 FSAR cites data for PWR-relevant Reynolds numbers (134 000 to 745 000) and concludes that β does not depend significantly on Reynolds number, pressure, or even steam quality [24].

2.2.7 Diversion crossflow

Diversion crossflow—a steady net transfer of mass from one channel to another—can be important under some conditions. One of those conditions is in the case of localized boiling which causes a large change in a subchannel friction factor. For evidence of this, refer back

to Rowe and Angle's data shown in Figure 2-6. As one channel approaches saturation, there is a large change in the outlet mass flows indicating a significant crossflow.

A blockage in one subchannel will also introduce crossflow. The resistance to crossflow is small so coolant will readily divert back and forth between subchannels to circumvent the restriction. This proves to be an important safety feature for reactors as it minimizes the impact of an accidental blockage. The Watts Bar Unit 2 FSAR cites a number of experiments with the conclusion that for a local blockage there is no significant change in the subchannel enthalpy rise and likely a small change to the DNBR (Departure from Nucleate Boiling Ratio) [24].

However, this small crossflow resistance which is beneficial for safety also leads to difficulties for a numerical solver. Small pressure drops drive large flows which can destabilize a solver. Large crossflows like these are no doubt found transiently in real rod bundles (as suggested by the LES results shown in Figure 2-8), but a solver must damp them out to find the steady-state solution.

However, the diversion crossflows are likely not significant for the problems considered here: single-phase, unblocked channels. As explained later in Section 2.3.3, this solver adjusts the inlet flow velocity of each subchannel to equalize the total axial pressure drop. For the cases considered here, this balancing procedure is able to reduce the cross-channel pressure drops everywhere to under 1 Pa—a negligibly small value. Hence it is assumed that there is no diversion crossflow.

The approximation of no diversion crossflow is reasonable for the problems modeled here, but note that it has implications for the implementation of the solver. The discretization and iteration procedures used here are likely inappropriate for a solver that must account for flow diversion.

For an example of other simple crossflow models, consider publications of the early THINC-I and THINC-II thermal-hydraulics codes [25]. The computational volume for THINC-I is a collection of many subchannels, a single assembly or a group of assemblies. Considering crossflow through multiple subchannels significantly increases the crossflow resistance making it practical for this code to solve for both the inlet flow distribution and the diversion crossflow.

THINC-II is a solver that instead operates on subchannels in a single assembly (with boundary conditions that account for crossflow from neighboring assemblies). To avoid

stability issues, THINC-II was implemented with the simplifying approximation that the crossflow resistance is negligibly small and thus no pressure gradient is allowed between subchannels. The solver then implicitly computes the crossflow values needed to realize this flat pressure profile. This approach has a side-effect of minimizing the impact of the inlet velocity distribution so no iteration on the inlet flow is needed.

2.3 Subchannel solver methodology

2.3.1 Conservation equations

This work uses a finite-volume discretization for solving the subchannel thermal-hydraulics equations. The equations are derived to conserve the integrals of mass, axial momentum, and energy over each volume. The derivation of the following equations starting from basic PDEs is laid out in Appendix A. Several simplifications and approximations are made which limit the use of these equations for problems beyond those considered here.

For the axial region, k, of each subchannel, i, the conservation of mass takes the form,

$$\langle \rho v_z \rangle_{i,k+1} = \langle \rho v_z \rangle_{i,k} - \frac{\Delta z_k}{A_i} \sum_j W_{ij,k} \tag{A.30}$$

where angle brackets with the subscript k indicate an average value on the lower face of the volume, angle brackets with $k+1$ indicate an average on the upper face, ρ is the fluid density, v_z is the axial velocity, A_i is the cross-sectional area of the volume, Δz_k is the height of the volume, j indicates an adjacent subchannel, and W_{ij} is the diversion mass crossflow from i to j.

The conservation of momentum takes the form,

$$\begin{aligned} \langle P \rangle_{i,k+1} = \langle P \rangle_{i,k} - \left(\langle \rho v_z \rangle_{i,k+1} - \langle \rho v_z \rangle_{i,k} \right) \langle v_z \rangle_{i,k} - \frac{\Delta z_k}{A_i} \sum_j M_{ij,k} \\ - \frac{\Delta z_k}{A_i} \left(\frac{f_k}{D_e} + K_k \right) \frac{\langle \rho \rangle_{i,k} \langle v_z \rangle_{i,k}^2}{2} \mathrm{d}z - g \Delta z_k \langle \rho \rangle_k \end{aligned} \tag{A.32}$$

where P is pressure, M_{ij} is the momentum crossflow, f is the Darcy friction factor, D_e is the equivalent diameter of the channel, K_k is a an area-change loss coefficient (zero except for volumes at the edge of a spacer grid), and g is the acceleration due to gravity.

The conservation of energy is,

$$\langle \rho h v_z \rangle_{i,k+1} = \langle \rho h v_z \rangle_{i,k} - \frac{\Delta z_k}{A_i} \sum_j H_{ij,k} + \frac{1}{A_i} q_{\text{wall},i,k} \tag{A.33}$$

where h is the specific enthalpy, H_{ij} is the crossflow of enthalpy, and $q_{\text{wall},i,k}$ is the energy flowing in from the adjacent fuel pins.

Values of individual fields like h are computed by assuming they may be separated from the averages,

$$\langle h \rangle_{i,k} = \frac{\langle \rho h v_z \rangle_{i,k}}{\langle \rho v_z \rangle_{i,k}}$$

2.3.2 Upwind discretization

In the derivation of the conservation equations, integrals along the z-axis were approximated as,

$$\int_{\Delta z_k} H_{ij}(z) \mathrm{d}z \approx \Delta z_k H_{ij,k} \tag{2.6}$$

where $H_{ij,k}$ is the quantity evaluated at the lower surface of the volume, the upwind surface.

This approach is similar to the upwinding scheme popular in advective finite volume solvers. However, it is only 0^{th}-order accurate. Don't tell my numerical methods professor. Nevertheless, a sufficiently fine mesh to mitigate truncation errors is practical and efficient for this solver. The choice to use upwinding here is not for numerical stability (the usual motivation for upwinding), but instead algorithmic simplicity and efficiency. Due to this upwinding scheme, the conservation equations can be solved in a single sweep from the inlet to the outlet. A coarse iteration is needed to converge the inlet velocity distribution, but no iteration is required otherwise.

Furthermore, because all quantities at axial level $k+1$ depend only on values at k, the values on each axial layer may be computed in parallel. This makes for a fast solver.

2.3.3 Inlet flow balancing

The axial boundaries for the subchannel solver match the top and bottom of the actively fueled regions of the core. It is approximated that the pressure is constant over these boundaries.

In order to get an equal pressure drop across each subchannel, the inlet flow velocity

for each channel must be adjusted. The inlet flow is parameterized with the unitless factor f_i which represents the fraction of the total core mass flow that flows through the inlet of channel i. The channel inlet velocity can then be computed from,

$$v_{\text{in},i} = \frac{f_i \dot{m}_{\text{total}}}{A_i \rho_{\text{in}}} \tag{2.7}$$

where $\dot{m}_{\text{total}}$ is the total core mass flow, A_i is the channel cross-sectional area, and ρ_{in} is the inlet density. The inlet density is computed from the inlet pressure, P_{in}, and temperature, T_{in}, which is specified by the user along with $\dot{m}_{\text{total}}$.

Friction is the dominant driver of pressure loss in this system and it scales approximately with v^2. Consequently, the computed pressure drop in each subchannel will scale roughly with v_{in}^2,

$$\Delta P_i = C_i v_{\text{in},i}^2 \tag{2.8}$$

After performing a sweep, the solver will find different values of ΔP for each subchannel. New inlet flow factors f_i can then be computed in order to equalize the pressures. For this purpose, the pressure drop in the first channel is arbitrarily chosen as the reference pressure drop.

Equating the drop in a channel, ΔP_i, to the reference pressure drop, ΔP_0, and substituting in Equations 2.7 and 2.8 gives,

$$f_i = f_0 \frac{A_i}{A_0} \sqrt{\frac{C_0}{C_i}} \tag{2.9}$$

This value is computed for each channel, and then the f_i factors are renormalized so that they sum to unity. Because of this renormalization, the resulting pressure drop in each channel might differ from ΔP_0. This renormalization also makes the solver insensitive to the choice of reference pressure drop. Any channel could be used as the reference, or the average value could be used; the end result will be similar.

This process of balancing the pressure drops forms an outer iteration scheme for the subchannel solver. After a sweep, the channel ΔP values are computed. If they differ by more than a preset tolerance then the factors, f_i, are updated and the sweep repeated until the pressure drops are within tolerance. To update the factors, first the friction coefficients, C_i, are computed from Equation 2.8. Then the f_i factors are computed from Equation 2.9

and renormalized so they sum to unity.

Tests with tight tolerances on the PSBT benchmark (discussed in Section 2.4) confirm that this scheme can reduce the cross-gap pressure drops throughout the geometry to under 1 Pa which justifies the assumption of negligible diversion crossflow. For results presented throughout this thesis, an absolute tolerance is set at 0.5 kPa as this is achievable in few iterations (typically fewer than 10) and believed to be negligibly small.

2.3.4 Water property tables

The temperature- and pressure-dependent water properties used here are based on the IAPWS releases [26, 27, 28]. However, the equations used in the IAPWS releases were found to be restrictively slow to evaluate directly in the solver. Computing the specific enthalpy of liquid water, for example, requires evaluating a 34-term polynomial with exponents reaching as high as 32 and as low as -42.

Instead, custom functions were fitted to the IAPWS data with faster evaluation times in mind. The specific enthalpy of liquid water (IAPWS-IF97 region 1) is modeled as a hyperbola,

$$h(P,T) = f_1(P) - f_2(P)\sqrt{x^2 - 1}$$

where x is a parameterized temperature,

$$x = \frac{f_4(P) - T}{f_3(P)}$$

and the functions f_1, f_2, f_3, f_4 are fitted second-order polynomials in terms of pressure.

The isobaric specific heat capacity and the specific volume were derived from this model of h as,

$$c_p(P,T) = f_2(P)\frac{x}{f_3(P)\sqrt{x^2 - 1}}$$
$$\nu(P,T) = C_1 e^{x^{-4}} + C_2 + C_3 T$$

Custom models are similarly used for viscosity and thermal conductivity.

2.4 Comparison to experimental data

In order to validate the solver and tune the β parameter, results were compared to the PSBT (PWR Subchannel and Bundle Tests) benchmark [29]. This benchmark includes an experimental campaign with electrically-heated subchannels and rod bundles.

The data relevant for this work come from exercise II-1, the steady-state fluid temperature benchmark. This test run uses a 5×5 rod bundle with the same pitch, rod diameter, and heated length as a Westinghouse 17×17 fuel assembly. The bundle includes a number of spacer grids, some with flow mixing vanes. A non-uniform radial power distribution was imposed as shown in Figure 2-12.

Thermocouples measured the coolant temperature at the outlet of each subchannel. The raw data is restricted to benchmark participants, but a useful subset can be found in open literature. Valette provides the results of two experiments (averaged along the y-axis) for validation of CATHARE 3 (in 3D mode with one mesh cell per subchannel) [30].

Calculations using the subchannel solver developed for this thesis are plotted alongside PSBT benchmark data from [30] in Figure 2-13. The value of $\beta = 0.08$ used in the solver was tuned to match the run 6232 results.

The calculated results agree well with the run 6232 data, but not as well with the run 5252 data. The 5252 run has a similar input power and mass flow rate (within 10%), but

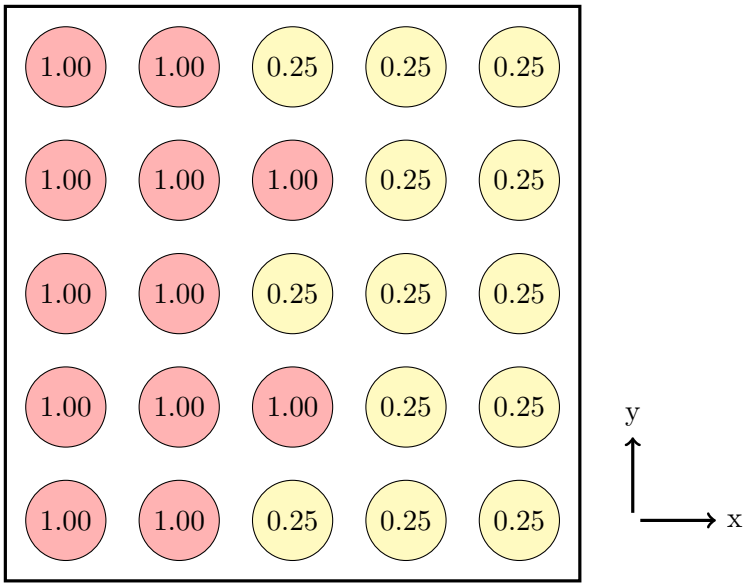


Figure 2-12: The radial power distribution in the relevant PSBT experimental test series.

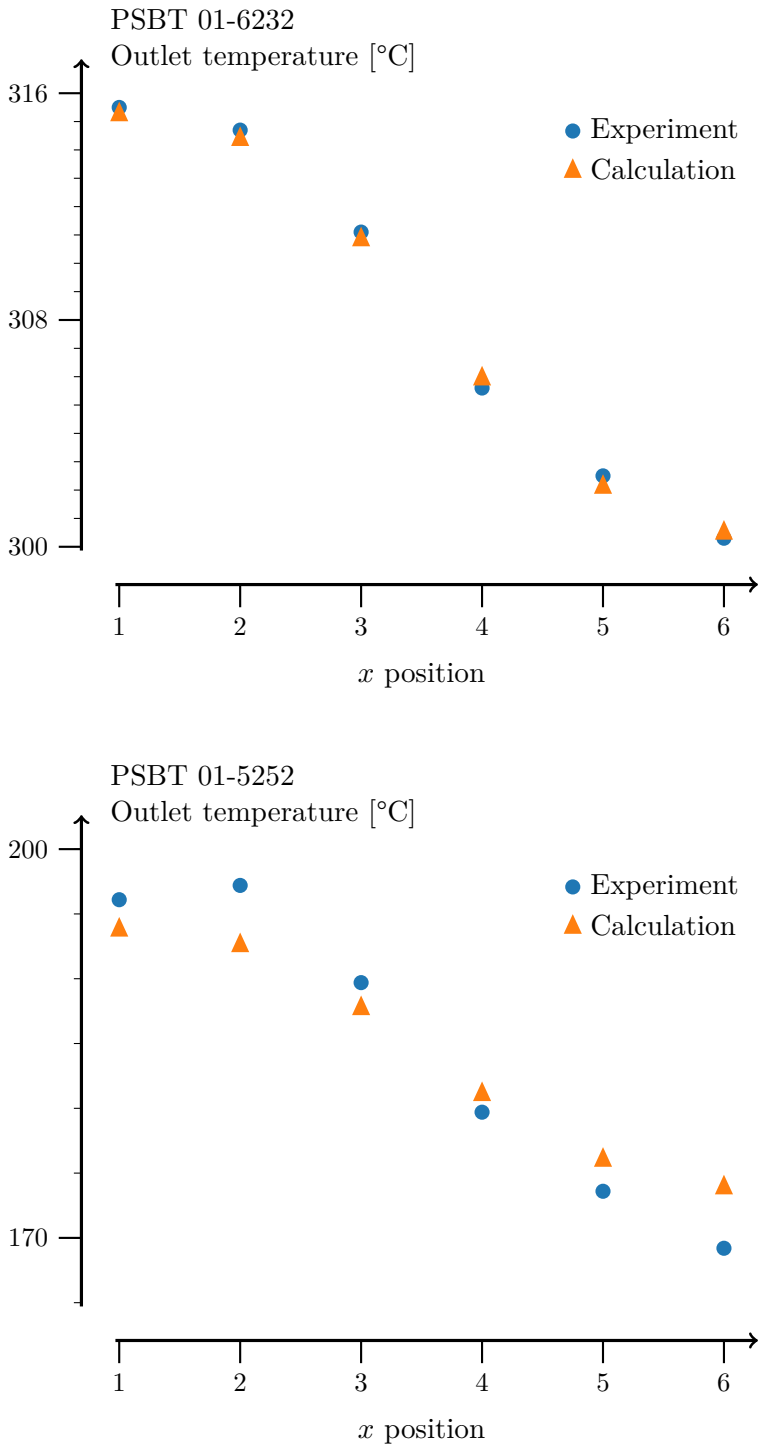


Figure 2-13: Comparison of subchannel outlet temperature calculation to experimental data. Results averaged along the y-axis (Figure 2-12 coordinates). Experimental data from [30].

the inlet coolant has a lower temperature, pressure, and velocity. The Reynolds number in central subchannels is predicted to be about 80 000 for 6232 and 45 000 for 5252.

The experimental temperature gradients are steeper for 5252 than predicted suggesting a smaller mixing factor. (A β of 0.04 or 0.05 matches the overall gradient more closely.) The experimental results also show a jump in temperature from the first column to the second column which is not replicated by the subchannel solver used here or by CATHARE 3 as reported by Valette [30]. Such a jump can be found by using a very small mixing factor (e.g. $\beta = 0.005$), but the overall gradient will be missed.

It is tempting from these results to use a model for β that increases as a function of Reynolds number. However this conflicts with the results reported by multiple researchers that find a small downward trend on bare rod bundles[19, 22] and data collected at PWR-relevant Reynolds numbers that show no significant trend [24]. Consequently, it is assumed here that β is independent of the Reynolds number. It is tuned to match run 6232 as that run is closer to PWR operating conditions.

Chapter 3

Heat Transfer Solver

This chapter discusses the heat transfer component of the presented multiphysics solver. This component accounts for conduction, convection, and thermal radiation in fuel rods in order to compute the fuel and cladding temperature distributions.

For a production solver that might be used by nuclear utilities and vendors, this portion of the coupled physics is critically important because the temperature in the fuel and cladding must be strictly limited to prevent releases of fission products in accident conditions. Fuel temperature also provides an important role in multiphysics feedback and reactor stability. In LWRs, the fuel temperature has a large impact on neutron transport, primarily through the Doppler broadening of cross sections.

For this work, a simple finite difference solver is developed that solves for the radial temperature distribution within a fuel pin. It is assumed that axial and azimuthal heat transfer are negligible which allows for the temperature in a fuel pin to be modeled with a set of uncoupled 1D systems.

Many important phenomena are neglected here including thermal expansion, creep, radiation-induced swelling, and restructuring. These phenomena must be considered for an accurate production solver, but their overall impact on the design of multiphysics algorithms is considered small enough that they are unnecessary for this thesis.

3.1 Discretization

The fundamental equation used by this solver is the conservation of heat which can be written in the general form,

$$\rho c_p \frac{\partial T}{\partial t} + \nabla \cdot \boldsymbol{q}'' - q''' = 0 \tag{3.1}$$

where ρ is the material density, c_p is the specific heat capacity, T is the temperature, $\boldsymbol{q}''$ is the heat flux vector, and q''' is the volumetric heat generation.

For this work only steady-state simulations are considered so the time derivative is equal to zero. Heat transfer in the axial and azimuthal directions are assumed to be negligible relative to the radial heat transfer.

The problem is discretized into a set of radial regions. Figure 3-1 illustrates the discretization. Each region is an annulus and the inner and outer radii for region i are respectively labeled $r_{i-1/2}$ and $r_{i+1/2}$. The innermost region has a zero inner radius, $r_{0-1/2} = 0$, making it a disk. Each region is also assigned a central radius, $r_i = (r_{i-1/2} + r_{i+1/2})/2$.

These regions cover the fuel and cladding but not the gap between them. This means that if region n is the outermost region of the fuel then $r_{n+1/2} \neq r_{(n+1)-1/2}$. Otherwise each region is in direct contact with its neighbors.

Applying this discretization scheme to Equation 3.1 and assuming only steady-state radial heat transfer gives,

$$\frac{r_{i+1/2}}{r_i \Delta r_i} q''_{i+1/2} - \frac{r_{i-1/2}}{r_i \Delta r_i} q''_{i-1/2} - q'''_i = 0 \tag{3.2}$$

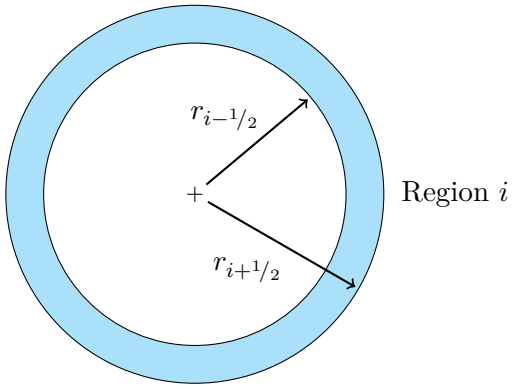


Figure 3-1: The heat transfer spatial discretization scheme. Region i is a circular annulus between radii $r_{i-1/2}$ and $r_{i+1/2}$.

where $q''_{i+1/2}$ and $q''_{i-1/2}$ are the values of of the radial heat flux at $r_{i+1/2}$ and $r_{i-1/2}$ and q'''_i is the heat generation rate in region i.

Enumerating this equation for $i = 0, 1, \ldots, N-1$ provides a system of N equations, each conserving energy for a particular ring. To solve this system, the q'' terms must be expressed in terms of T.

Except for the outer boundary condition, the heat flux across each surface is computed from the difference in temperature of the two adjacent regions e.g.,

$$q''_{i+1/2} = C_{i,i+1} \left(T_{i+1} - T_i \right) \tag{3.3}$$

The form of the coefficients, $C_{i,i+1}$, differ depending on whether the heat transfer mechanism is conduction through the interior of a solid or conduction and radiation through the fuel-clad gap. The value each coefficient takes depends on the temperature of the adjacent regions which makes the heat transfer problem nonlinear. These coefficients are derived in Appendix B.

3.2 Heat flux models

Most of the radial surfaces considered by the solver lie within a solid material—either the fuel or the cladding. Across these surfaces, the heat flux is computed from Fourier's law of conduction. Assuming only 1D radial heat transfer, Fourier's law can be expressed as,

$$q'' = -k \frac{\partial T}{\partial r} \tag{3.4}$$

where k is the thermal conductivity. Appendix B describes the temperature-dependent model used for the fuel thermal conductivity; a fixed value is used for the cladding.

The heat flux across the fuel-clad gap is computed from the linear model,

$$q''_g = h_g \left(T_{\text{fo}} - T_{\text{ci}} \right) \tag{3.5}$$

where h_g is the gap heat transfer coefficient, T_{fo} is the temperature of the outer surface of the fuel, and T_{ci} is the temperature of the inner surface of the cladding.

As described in Appendix B, h_g is evaluated using temperature-dependent models for

helium gas thermal conductivity and for thermal radiation.

The heat flux on the outer surface of the cladding is computed using the Dittus-Boelter model for convective heat transfer and the Chen model for nucleate boiling. For each fuel pin, values from the adjacent subchannels are averaged to find the fluid properties needed for these models.

Appendix B describes how the $C_{i,i+1}$ coefficients (used in Equation 3.3) are computed for each of these heat flux models. This forms a set of linear models that can be used in a typical linear algebra solver. However, the coefficients also depend on the problem temperature values which necessitates a nonlinear iteration as well.

3.3 Numerical solver

3.3.1 Solver overview

The basic structure of the heat transfer solver is shown in Algorithm 1. The solver handles each axial segment of each fuel pin separately. For each of these regions, the solver first computes the cladding outer wall temperature, and this is used as a boundary condition for the heat transfer problem. A linear solver is then used within a nonlinear iteration loop to compute the temperature values of the fuel rod interior.

3.3.2 Linear system

From the discretized heat equation, Equation 3.2, and the linear temperature difference model of the radial heat flux, Equation 3.3, the conservation of heat over the typical region i can be expressed in the form,

$$A_{i,i-1}T_{i-1} + A_{i,i}T_i + A_{i,i+1}T_{i+1} = Q_i$$

Algorithm 1 Basic structure of the heat transfer solver

```
1: for each axial level do
2:     for each fuel pin do
3:         Compute T_co (Algorithm 2)
4:         for nonlinear iteration do
5:             Use temperature solution to update linear system coefficients
6:             Solve the linear heat transfer system
```

where the A coefficients can be computed from the previously discussed heat flux models and Q_i is computed from the heat generation in the region. This form holds for all regions except the innermost where there is no T_{i-1} term.

Enumerating these equations for each ring of the fuel and cladding gives the linear system,

$$\boldsymbol{AT} = \boldsymbol{Q} \tag{3.6}$$

where $\boldsymbol{A}$ is a matrix of the A coefficients, $\boldsymbol{T}$ is a vector of temperature values, and $\boldsymbol{Q}$ is a vector of heat generation.

For convenience in the software implementation, an atypical approach is used here where the outer boundary condition contributes to the left-hand-side of Equation 3.6 rather than the right-hand-side. If there are N regions then the vector, $\boldsymbol{T}$, has length $N + 1$. The last value in $\boldsymbol{T}$ is T_{co}, the temperature on the outer surface of the cladding. This value is computed elsewhere (as discussed in Section 3.3.4) and is considered fixed for the purposes of the linear solver.

For an example 5-region problem, the linear system can be written in the form,

$$\begin{pmatrix} A_{0,0} & A_{0,1} & & & & \\ A_{1,0} & A_{1,1} & A_{1,2} & & & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & \\ & & A_{3,2} & A_{3,3} & A_{3,4} & \\ & & & A_{4,3} & A_{4,4} & A_{4,5} \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_{\text{co}} \end{pmatrix} = \begin{pmatrix} Q_0 \\ Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{pmatrix} \tag{3.7}$$

Note that the term, $A_{4,5}T_{\text{co}}$, could be moved to the right-hand-side of Equation 3.7 and subsumed in Q_4. This would change the system into a more conventional form with a square matrix and the vector $\boldsymbol{T}$ representing only unknown values. However the $A_{4,5}$ coefficient is computed in a similar manner to the other off-diagonal coefficients so it is more convenient to leave this term on the left-hand-side for the purposes of implementing a solver.

To reduce the size of the $\boldsymbol{A}$ matrix in computer memory, it is stored in a banded form. This form is like the banded matrix form used by LAPACK, except that here it uses row-

major order as is conventional in C. For the 5-region example,

$$\text{Band}\left(\boldsymbol{A}\right) = \begin{pmatrix} A_{0,1} & A_{1,2} & A_{2,3} & A_{3,4} & A_{4,5} \\ A_{0,0} & A_{1,1} & A_{2,2} & A_{3,3} & A_{4,4} \\ & A_{1,0} & A_{2,1} & A_{3,2} & A_{4,3} \end{pmatrix}$$

The matrix is tri-diagonal meaning it can be efficiently solved in a direct fashion via Gaussian elimination (also known as the Thomas algorithm for tri-diagonal matrices). The system is solved in-place, reusing the computer memory allocated for $\boldsymbol{A}$ to store intermediate values of the calculation. Note that the solver only computes the first N elements of $\boldsymbol{T}$—neglecting the final element which represents the boundary temperature.

3.3.3 Nonlinear iteration

Most of the elements in $\boldsymbol{A}$ depend on the temperature solution. This is because of the temperature-dependent models for thermal conductivity and the nonlinearity of radiative heat transfer. For this reason, the solver includes a nonlinear iteration loop.

For each of these iterations, the temperature-dependent parameters are computed and used to fill the $\boldsymbol{A}$ matrix. The tri-diagonal linear solver is then used to compute an updated temperature solution. The output of the linear solver is used directly making this a non-relaxed Picard iteration.

After each iteration, an L_2 norm is computed of the relative temperature change in each region. The iteration terminates when this norm falls below a specified tolerance.

A separate iteration is performed for each fuel pin and for each axial region of the pins. This means that fewer iterations will be performed for regions that converge quickly.

3.3.4 Boundary condition iteration

Due to the onset of nucleate boiling, the heat transfer coefficient on the outer surface of the fuel rod is highly nonlinear with respect to temperature. The heat transfer coefficient proved difficult to resolve with go-to methods like Picard iteration and Newton iteration. Instead, a custom search procedure is used.

The heat flux on cladding outer surface, q''_{co}, is modeled as,

$$q''_{\text{co}} = h_{\text{conv}}\left(T_{\text{co}} - T_b\right) + h_{\text{nb}}\left(T_{\text{co}} - T_{\text{sat}}\right) \tag{3.8}$$

where h_{conv} is the convective heat transfer coefficient, h_{nb} is the nucleate boiling heat transfer coefficient, T_{co} is the temperature of the cladding outer surface, T_b is the bulk temperature of the surrounding coolant, and T_{sat} is the saturation temperature of the coolant at the relevant pressure.

In steady-state conditions the heat flux is a known quantity, but the cladding temperature is not. Solving this equation for the cladding temperature gives,

$$T_{\text{co}} = \frac{h_{\text{conv}} T_b + h_{\text{nb}} T_{\text{sat}} + q''_{\text{co}}}{h_{\text{conv}} + h_{\text{nb}}} \tag{3.9}$$

The difficulty arises from the fact that h_{nb} depends strongly on T_{co}. This dependence can be shown explicitly with,

$$T_{\text{lhs}} = \frac{h_{\text{conv}} T_b + h_{\text{nb}}\left(T_{\text{rhs}}\right) T_{\text{sat}} + q''_{\text{co}}}{h_{\text{conv}} + h_{\text{nb}}\left(T_{\text{rhs}}\right)} \tag{3.10}$$

where the subscripts now indicate the cladding temperature on the left-hand-side and right-hand-side of the equation.

The search procedure aims to find the fixed point where the equation is balanced with $T_{\text{lhs}} = T_{\text{rhs}}$. The procedure is described in Algorithm 2. First the cladding temperature is computed assuming only convective heat transfer. If the computed temperature is below saturation, then no boiling is expected and the temperature value is accurate.

If boiling is predicted, then the solver begins searching for the cladding temperature starting with a guess of $T_{\text{rhs}} = T_{\text{sat}}$. It steadily increases T_{rhs} by 1 K increments until it overshoots the solution as indicated by $T_{\text{lhs}} < T_{\text{rhs}}$.

At this point, the solution is known to lie somewhere between T_{rhs} and $T_{\text{rhs}} - 1$ K. So the solver then performs a binary search over this range. This search is considered converged when the relative change in h_{nb} between iterations is less than 10^{-4}. The resulting T_{co} can then be used as a boundary condition for the non-linear iteration on fuel and cladding temperature.

Algorithm 2 Search procedure for the cladding outer temperature considering both convective and boiling heat transfer.

Compute the average fluid properties of adjacent subchannels
Compute h_{conv} from the Dittus-Boelter correlation
$T_{\text{co}} \leftarrow T_b + \frac{q''_{\text{co}}}{h_{\text{conv}}}$
if $T_{\text{co}} > T_{\text{sat}}$ **then**
    $T_{\text{lhs}} \leftarrow \infty$
    $T_{\text{rhs}} \leftarrow T_{\text{sat}} - 1$ K
    while $T_{\text{lhs}} > T_{\text{rhs}}$ **do**
        $T_{\text{rhs}} \leftarrow T_{\text{rhs}} + 1$ K
        Compute h_{nb} from the Chen correlation using T_{rhs}
        $T_{\text{lhs}} \leftarrow \frac{h_{\text{conv}} T_b + h_{\text{nb}} T_{\text{sat}} + q''_{\text{co}}}{h_{\text{conv}} + h_{\text{nb}}}$
    $T_{\text{lo}} \leftarrow \max\left(T_{\text{sat}}, T_{\text{rhs}} - 1 \text{ K}\right)$
    $T_{\text{hi}} \leftarrow T_{\text{rhs}}$
    while Unconverged **do**
        $T_{\text{mid}} = \frac{1}{2}\left(T_{\text{lo}} + T_{\text{hi}}\right)$
        Compute h_{nb} from the Chen correlation using T_{mid}
        $T_{\text{co}} \leftarrow \frac{h_{\text{conv}} T_b + h_{\text{nb}} T_{\text{sat}} + q''_{\text{co}}}{h_{\text{conv}} + h_{\text{nb}}}$
        if $T_{\text{co}} < T_{\text{mid}}$ **then**
            $T_{\text{hi}} \leftarrow T_{\text{mid}}$
        else
            $T_{\text{lo}} \leftarrow T_{\text{mid}}$

Chapter 4

Neutronics Solvers

This chapter will describe how neutron transport has been treated in the multiphysics solver. Two different neutronics solvers are used: OpenMC [11] which is a Monte Carlo transport solver and a custom coarse mesh finite-difference solver that is based partly on neutron diffusion theory.

Monte Carlo is advantageous because it can explicitly represent the geometry features in 3D, use continuous-energy cross sections, and resolve the angular distribution of neutron flux without approximate discretizations. This cuts out much of the development and validation work needed for the classic neutronics toolchain of equivalence theory, lattice physics, and nodal diffusion.

However, converging the neutron source distribution with MC is very expensive so it is useful to include a coarse mesh solver in the system. Given parameters generated by MC, the coarse mesh solver can find accurate solutions with orders-of-magnitude less runtime [8].

The focus of this thesis is on effectively using the MC-backed coarse mesh solver in a multiphysics simulation. For that purpose, a novel technique is introduced where MC differential tallies are used to approximate the temperature and density dependence of the coarse mesh solver parameters. The coarse mesh solver with feedback-dependent parameters can then be used as an approximate surrogate in place of MC for some multiphysics iterations.

This chapter will discuss the MC and coarse mesh solvers in detail. First, some background is provided on recent advances in MC methods that make it practical for use on full-core and quarter-core multiphysics problems. Following that is a discussion of the OpenMC differential tally implementation and how it can be used to generate feedback-

dependent coarse mesh parameters. Finally, this chapter will describe the coarse mesh solver with some focus on parallel numerical methods.

4.1 Background on full-core Monte Carlo simulations

The objective of this work is to design a multiphysics solver that can solve full-core problems in a practical amount of time on a practical computer. A few decades ago, this could not be done with a Monte Carlo solver. However it is now approaching reality thanks to advances in both computer resources and solver methods development.

This section will discuss some of the important advances in methods that have enabled full-core MC calculations. It is helpful to frame this discussion with observations published by Bill Martin in 2012 [31]. Martin highlights these six issues as a barrier at the time for routine usage of full-core MC simulations:

- Prohibitive computational time for acceptable statistics
- Excessive demand on computer memory
- Slow convergence of the fission source
- Apparent versus true variance
- Accounting for multiphysics feedback
- Adapting to future computer architectures

4.1.1 Cross section representation

Relating to the issues of computer memory and multiphysics feedback, Martin brought attention to the large costs of using Doppler-broadened pointwise cross section libraries to simulate reactors with a realistic range of temperature variations.

Historically, MC transport codes usually accepted as input a pointwise cross section library that had been Doppler-broadened to a set of discrete temperatures in pre-processing. This raises a challenge for multiphysics simulations where each region of the reactor might take on a unique temperature from the continuum of real values. One clever work-around is the "pseudo-material" method described by Jeremy Conlin et al. [32].

If, for example, a fuel region's temperature is computed to be 1023.5 K then the fuel material can be modeled as a pseudo-material with some fraction of e.g. ^{235}U at a lower

temperature such as 900 K and the remainder ^{235}U at a higher temperature such as 1200 K. Conlin et al. [32] describe the procedure for choosing the appropriate nuclide ratios based on interpolation using the square root of temperature values, and they demonstrate this method on a high temperature gas reactor multiphysics problem solved with MCNP5 and RELAP5. Others advocate for interpolation that is linear in temperature. Daeubler et al., for example, use linearly-interpolated pseudo-materials in a full core PWR calculation with Serpent 2 and SUBCHANFLOW [33] based on a study of cross section interpolation by Trumbull [34].

One particular advantage of the pseudo-material approach is that it allows for the use of well-established codes like MCNP5 without having to modify the internals of the solver. Pseudo-materials can also be applied equally easily to resonance range pointwise data, unresolved resonance probability tables, and molecule-bound thermal scattering distributions. However, the pseudo-material method comes with high computer memory costs given that the solver must store the pointwise cross sections at many temperature values. It also places a burden on the user to decide how fine of a temperature grid is needed and possibly make a trade-off between accuracy and computational expense.

Pseudo-materials are a useful hack to make the traditional MC cross section representations work for multiphysics calculations, but over roughly the last decade, MC developers have been building a variety of better cross section representations that remove the need for pseudo-materials. These methods include kernel reconstruction, on-the-fly (OTF) Doppler broadening, target motion sampling (TMS), and windowed multipole (WMP).

The kernel reconstruction method is a high-order interpolation technique that uses cross sections evaluated at a carefully chosen set of temperatures [35]. As a pre-processing step, pointwise cross sections are generated on a precise, irregular temperature grid. During the MC simulation, the cross section at a particular temperature can be evaluated from a linear combination of its values on the reference temperature grid. This method can be viewed as a generalization of pseudo-materials in that it uses many points for interpolation rather than two. It is more accurate than simple interpolation schemes, and as a result, a coarser temperature grid with less memory cost can be used to achieve the same target accuracy. However, it still requires pointwise cross sections evaluated at many temperatures.

OTF is a different approach that has been recently adopted by the MCNP developers [36]. With this approach, cross section temperature dependence is modeled as a polyno-

mial using powers of $\sqrt{T}$ and $1/\sqrt{T}$. With this method, a grid of temperatures is effectively replaced with a set of polynomial coefficients. Unfortunately, high-order polynomials are required for accuracy—up to 17 terms to achieve <0.1% error on the cross section values—which complicates their implementation [36].

TMS has been developed primarily by the Serpent MC team [37, 38]. TMS is a rejection-sampling technique (similar to delta tracking) where potential collision sites are sampled using a majorant cross section. At each potential collision, a target nucleus velocity is sampled and the collision is accepted or rejected based on the cross section of a target with that velocity. Inefficient rejection-sampling can lead to runtime issues, but the MC code must only store cross sections at one temperature which offers significant memory benefits over the other pointwise methods.

WMP is the approach taken by the OpenMC team [39, 40]. With WMP, the pointwise representation of cross sections is abandoned in favor of a set of low-order polynomials and parameters that describe the location and shape of cross section resonances. These parameters are independent of temperature, and Doppler broadening is handled analytically while tracking neutrons instead of as a pre-processing step. Cross sections are slightly slower to evaluate with WMP, but large memory savings are offered relative to the pointwise methods.

WMP is the method used here for temperature-dependent resolved resonance range cross sections. This method was chosen primarily because it is the only one of the three implemented in the main branch of OpenMC; it is unclear if any of the others would prove superior for this application.

Note that neither WMP nor TMS can be used to model molecular thermal cross sections (i.e. those computed with $S(\alpha, \beta)$ tables) or unresolved resonance cross sections. For those cross sections, this work uses OpenMC's interpolation feature on a traditional grid of discrete temperatures. This interpolation method is similar to the pseudo-material approach. The thermal and unresolved resonance cross sections require much less memory than the resolved resonance range so the simple interpolation approach is practical.

4.1.2 Acceleration methods

One issue Martin highlighted, "slow convergence of the fission source", can be directly addressed with acceleration methods like CMFD (coarse mesh finite difference). With

CMFD, a modified form of a finite difference diffusion solver (using MC tally results) is used to compute the expected neutron source distribution, and MC source sites are weighted to match that distribution. This usually enables a MC simulation to reach a converged source distribution with many fewer neutron generations.

CMFD was historically developed for multigroup deterministic solvers. It was applied to full-core MC calculations in 2010 by Lee et al. [8]. In this case, CMFD reduced the number of generations needed for convergence (starting from a flat source, convergence estimated via Shannon entropy) by an order of magnitude, from 150 to 10.

In 2014, Herman published an extension of the CMFD developments where multiphysics iterations were performed directly with CMFD [10]. This approach not only reduces runtime spent converging the MC source distribution, it also aims to reduce the total number of active MC generations needed for coupled system convergence because much of the information those generations would provide to the heat transfer and fluid dynamics solvers is now provided by CMFD.

In order to capture thermal feedback with CMFD, the temperature-dependent behavior of the CMFD input parameters (cross sections, diffusion coefficients, and current correction factors) must be modeled. Herman used an on-the-fly machine learning algorithm to fit temperature-dependent models of CMFD parameters. This thesis builds upon the work of Herman by computing the cross section derivatives directly with MC tallies.

The methods presented here have many similarities to those used by Herman so it is worth highlighting some of the distinctions. The machine learning approach used by Herman requires defining a feature vector—the independent variables considered by the machine learning model when fitting data. For the learned model to be useful, there must be a large enough variety and quantity of features so that the model can capture the observed variation in the data. For example, Herman used a feature vector that included for each coarse mesh cell: fuel enrichment, number of burnable poison rods, average coolant density, and average fuel temperature.

However, providing too many features may lead to overfitting where the software begins to identify and use spurious relationships in the data that ruin its predictive capability. So it may be the case that adding another feature for proximity to an axial reflector makes the model more predictive, or it may cause overfitting. Likely, the stochastic results produced by MC codes will exacerbate overfitting issues. Because of these issues, finding an optimal

feature vector is a non-trivial task.

For this work, the surrogate model (CMFD with feedback-dependent cross sections) is created using differential tallies. One advantage to this approach is that extra derivatives (e.g. ^{135}Xe or ^{235}U concentration) can be added without the concern for overfitting.

Another distinction is based on the fact that the machine learning model attempts to measure derivatives by looking at the variation between regions. If the model is missing important variables, then it may try to explain the data variations with the incorrect variables. For example, if the model does not account for control rod insertion, it may try to fit for the impact of control rods using the fuel temperature. Differential tallies will have no such error. Similarly, differential tallies can resolve e.g. a density derivative even when there is no variation in density across the problem (as may be created by a uniform initial condition) or when the variation is so small that its impact is obscured by stochastic noise.

4.2 MC software optimization

This section will discuss a collection of software-focused optimizations that are used in this thesis to reduce the runtime required for the Monte Carlo neutronics solver. This chapter as a whole largely focuses on diffusion-based CMFD neutronics which is essentially a way of reducing the number of required MC generations for a simulation. This section instead focuses on reducing the runtime required for each of those generations.

Two optimizations are presented for OpenMC's ray tracing methodology: shared-memory cell-based neighbor lists and universe partitioning. Fine discretization for multiphysics purposes can reduce ray tracing performance, but these optimizations exploit patterns in the discretization to mitigate the cost. Each of these optimizations leads to a 25% performance boost on the quarter-core multiphysics problem discussed in Chapter 7

A discussion is also provided on the use of collision-estimated tallies in place of tracklength-estimated. This also leads to a roughly 25% runtime reduction due to the complexities of computing tracklength-estimated tallies on a mesh. A demonstration is also provided on a simple 2D pincell problem showing that the difference in resulting tally uncertainty is small for the problems of interest.

4.2.1 Efficient neighbor lists

Surface-based versus cell-based neighbor lists

When a simulated particle crosses a boundary in a MC code (that is not using delta-tracking), the software must determine which cell the particle is entering. An exhaustive search over all cells would be prohibitively expensive so MC codes use neighbor lists to reduce the search space.

There are several different ways to implement neighbor lists. The most fundamental design choice for neighbor lists is what type of geometric entity their adjacency relationship is based on. One option is surface-based adjacency, where there is a neighbor list for each surface in the geometry (or one for each side of each surface). Another option is cell-based adjacency where there is one neighbor list for each cell. Figure 4-1 provides a diagram to highlight the distinction. (As discussed in Appendix C, some MC use a further refined technique where each cell has a separate neighbor list for each surface that it touches.)

For the multiphysics geometries considered here, cell-based neighbor lists are generally shorter than surface-based and thus more efficient. Consider a simple 3D pincell problem that is axially discretized. Figure 4-2 shows that cylindrical surfaces in this geometry will span the entire length, and thus the number of neighbors is proportional to the number of axial regions. In contrast, Figure 4-3 shows that the number of neighbors for a typical cell does not change with the axial discretization.

The downside to cell-based neighbor lists is that they are more difficult to generate. Surface-based lists can be generated relatively easily during the initialization stage of the solver. Cell-based lists are instead generated during the simulation, using the boundary crossings observed by the solver to discover the adjacency relationships.

Building a neighbor list during the simulation in turn poses difficulties for a MC solver

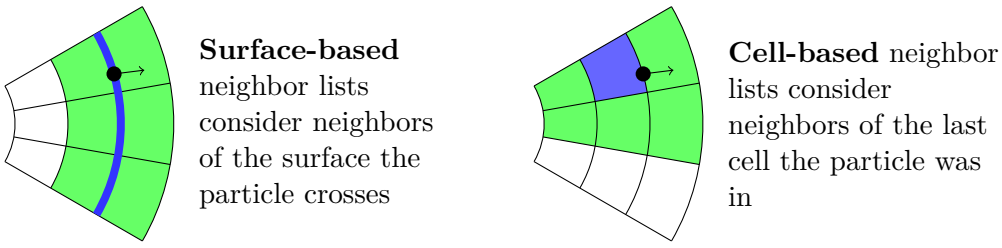


Figure 4-1: Diagram to indicate the difference between surface-based and cell-based neighbor lists on a hypothetical geometry.

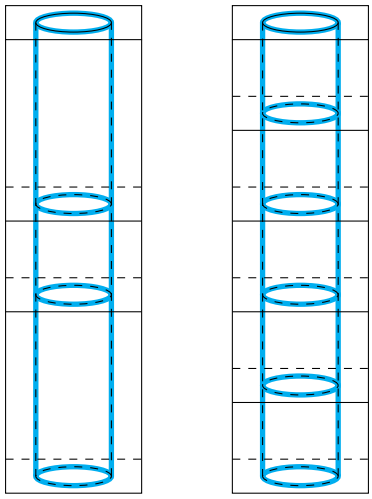


Figure 4-2: A simplified fuel pin geometry with two different levels of axial discretization. The highlighted cylindrical surface divides fuel cells from moderator cells. The number of cells neighboring the highlighted surface scales proportionally with the axial discretization.

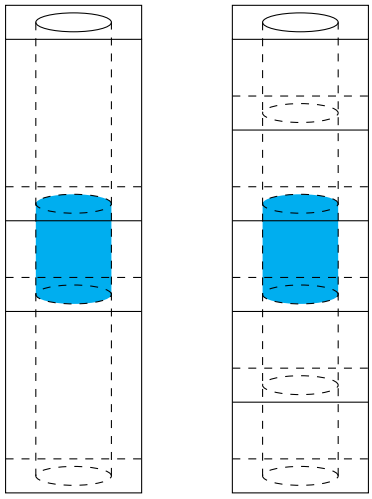


Figure 4-3: A simplified fuel pin geometry with two different levels of axial discretization. The number of cells neighboring the highlighted cell does not scale with the discretization.

with shared-memory capabilities. Generating efficient, cell-based, shared-memory neighbor lists was the topic of a separate publication [41], and the findings of this publication are discussed in Appendix C.

OpenMC originally used surface-based neighbor lists. The cell-based, shared-memory methodology discussed in Appendix C was merged into the main branch of OpenMC with pull request #1140, and the simulations presented in this thesis use those methods.

Quarter-core performance

Simulations were run to test the impact of the neighbor list changes on the quarter-core PWR multiphysics problem described in Chapter 7. These simulations used the same accelerated multiphysics solver discussed in Chapter 7 and the same geometric discretization. This problem uses a 42-region axial mesh with 1 radial region in the fuel. These test simulations used 5 MC generations with 200 million neutrons per generation. (More generations would be needed to converge tallied quantities of interest, but this small number is sufficient to test solver performance.) These simulations were run using Google Cloud Platform on a single "n1-standard-64" instance. Exact details of the machine architecture are unknown, but it presents (in `/proc/cpuinfo`) as a machine with 64 generic Intel Xeon cores.

For this study, surface-based neighbor lists were reimplemented in a recent version of OpenMC. Consequently, the only software difference between the two simulations is the neighbor list implementation. (Both simulations include universe partitioning which is discussed in Section 4.2.2.)

With surface-based neighbor lists, the MC portion of the solver has a wall time cost of 5.3 hours (340 cpu core hours). With cell-based lists, the wall time falls by 25% to 4.0 hours (260 cpu core hours). The MC solver is the most expensive of each of the coupled physics solvers so the overall multiphysics solver runtime reduction is 23%.

4.2.2 Universe partitioning

Background

Neighbor lists are not used in OpenMC when a particle crosses a boundary between universes (including every boundary in a fuel assembly lattice). For these universe crossings, a search is performed over the cells in the destination universe.

Possibly, the neighbor list implementation could be improved to cover these universe crossings. However, in the course of this work, it was discovered that there was a low-hanging fruit for speeding up the cell search, and thus lessening the cost of these universe crossings regardless of the neighbor lists.

The multiphysics geometries considered for this work are axially divided with a large number of z-planes. This forms a pattern in the geometry that the ray tracing code can take advantage of: if a particle lies above a plane at $z = z_0$, then it also lies above the planes at $z = z_0 - 10$ cm and $z = z_0 - 20$ cm and so on.

As originally implemented in OpenMC, a universe crossing would result in a simple linear search for the particle. There is a list of cells that lie in each universe, and the code would step one-by-one through the list, checking to see if the particle lies in each cell. If there are N axial regions, then the cost of this search is $\mathrm{O}(N)$. With an axially-discretized geometry, this procedure can be replaced with a binary search which gives an $\mathrm{O}(\log(N))$ cost. Figure 4-4 illustrates how a binary search tree can be constructed from such a geometry.

In the course of this thesis, the `UniversePartitioner` was added to OpenMC for this purpose. OpenMC now automatically checks during initialization for universes that contain many z-planes.[1] When such universes are found, they are assigned a `UniversePartitioner`

[1] To be precise, OpenMC will currently partition universes that contain more than 10 cells and more than 5 z-planes.

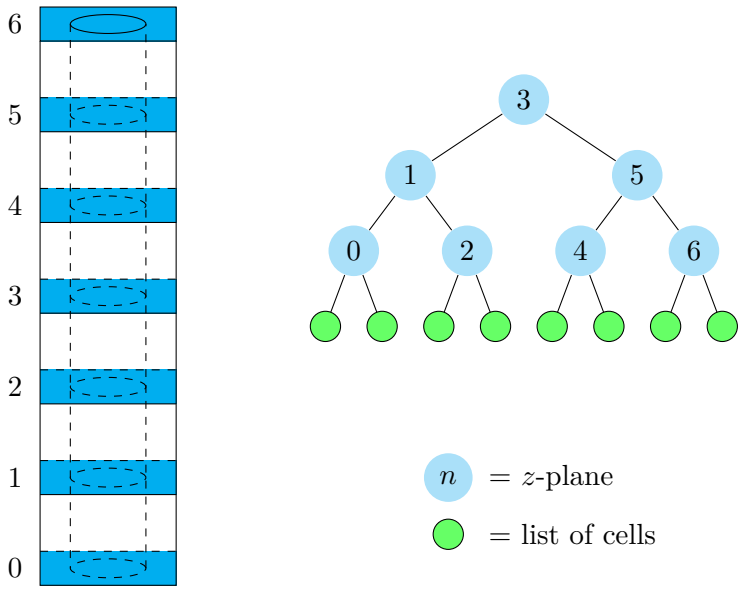


Figure 4-4: Left: a simplified pincell geometry with highlighted z-planes. Right: a binary search tree for geometric cells using the z-planes.

which builds a binary search tree over the z-planes. Given a particle location, this object traverses the search tree and returns the subset of cells that lie within the z-bounds that bracket the particle location. This smaller subset of cells is then used in the traditional search routines. This feature was merged into the main branch of OpenMC with pull request #1210.

Quarter-core performance test

The performance was evaluated on the quarter-core multiphysics problem using the same procedures described at the end of Section 4.2.1. The `UniversePartitioner` feature was deactivated, and accelerated multiphysics simulations were run with 5 MC generations and 200 million neutrons per generation on a 64-core "n1-standard-64" Google Cloud Platform instance.

Without universe partitioning, the MC portion of the simulation requires 5.3 wall clock hours (340 cpu core hours). With universe partitioning the time falls to 4.0 hours (260 cpu core hours) which is a 25% reduction. Coincidentally, the runtime savings almost exactly match those offered by switching from surface-based to cell-based neighbor lists.

Notes on the OpenMC implementation

The `UniversePartitioner` implemented in OpenMC is targeted at the axial discretization of multiphysics models. Likely, it could be expanded in the future to assist with fine radial or azimuthal discretizations as well.

It should also be noted that OpenMC's `UniversePartitioner` is effectively a simplified version of the octree and k-d tree techniques that are popular in computer graphics software. Implementing these more advanced methods might lead to further performance increases, although there are some important differences between typical MC neutronics geometry models and computer graphics geometry models.

One limitation to the OpenMC implementation is that only *simple cells* are partitioned. In OpenMC, the volumetric region of each cell is described in terms of intersection, union, and complement operators on surface half-spaces. Cells which are only defined with intersection operators are referred to as *simple*.

Given the region definition of a simple cell, it is a straightforward process to determine the cell's bounding box which then allows it to be easily placed in the search tree of a

`UniversePartitioner`. Determining the bounding box of a non-simple cell is more difficult so these cells are not partitioned; the non-simple cells are added to every leaf of the `UniversePartitioner`'s search tree. This is one of many reasons why simple cells lead to better performance in OpenMC.

4.2.3 Collision-estimated tallies

Another optimization applied here is to use collision-estimated tallies rather than tracklength-estimated tallies. This significantly reduces the solver runtime, largely due to the complications of using tracklength tallies with mesh filters.

Software implementation differences

In OpenMC, filters can be specified by a user to discretize tallies over materials, cells, energy intervals, and a broad array of other variables. This thesis makes extensive use of mesh filters which can discretize tallies on a Cartesian mesh that overlays the geometry.[2] These are very convenient for use with finite difference diffusion solvers because they can compute reaction rates and fluxes that are homogenized directly onto the diffusion solver's mesh.

For the quarter-core calculations described in Chapter 7, there are two OpenMC mesh filters in use. One matches the mesh used by the diffusion solver with 4 cells per fuel assembly in the xy-plane. This mesh filter is used to compute the 2-group cross sections, diffusion coefficients, and $\widehat{D}$ correction factors used by the diffusion solver. The second is a finer mesh with 1 cell per fuel pin in the xy-plane, and this mesh is used to compute the q' distribution.

To implement the mesh filter for collision-estimated tallies, the software must simply be able to locate the mesh bin of a collision point after each sampled particle collision. This can be done efficiently by performing a binary search over the mesh boundaries in each of the three spatial dimensions.

The software for tracklength-estimated tallies is far more complex. Instead of locating a single point in the mesh, the software must now handle the line segment created by a particle transport event. This line segment can pass through multiple mesh cells. The code must determine which mesh cells the segment passes through and what fraction of the line

[2] OpenMC currently offers regular and rectilinear Cartesian meshes. The rectilinear mesh capability was added in the course of this thesis and merged into OpenMC with pull request #1246.

lies within each. The software logic is complicated by accounting for corner-cases such as when the line segment starts or ends outside of the mesh, or when it does both but still intersects the mesh. The software must also guard against very short line segments which could lead to floating-point precision issues. As a result of this complexity, the runtime cost of tracklength-estimated mesh tallies is significantly greater than that of collision-estimated mesh tallies.

Quarter-core performance test

To demonstrate the runtime differences, a test was run on the quarter-core multiphysics problem using the same procedures described at the end of Section 4.2.1. The collision-estimated tallies were replaced with tracklength-estimated tallies, and an accelerated multi-physics simulation was run with 5 MC generations and 200 million neutrons per generation on a 64-core "n1-standard-64" Google Cloud Platform instance.

With tracklength tallies, the MC portion of the simulation requires 5.4 wall clock hours (350 cpu core hours). With collision-estimated tallies, the time falls to 4.0 hours (260 cpu core hours) which is a 27% reduction. This is again similar to the runtime boosts offered by cell-based neighbor lists and universe partitioning.

Estimator efficiency

There is a reason that software developers have gone through the trouble of implementing the complex tracklength tallies. Considering the stochastic uncertainty of the resulting tallied values, tracklength tallies can offer better performance for optically thin regions. In these regions, few collisions are sampled, and collision-estimated tallies consequently lead to a high variance.

However, the tally regions used here cover one or more entire fuel pins and are thus optically thick. For these thick regions, both estimators lead to a similar variance.

To demonstrate this, a small 2D pincell problem was studied. The geometry and materials match the example pincell problem distributed with OpenMC. Simulations were performed in fixed-source mode (with the `create_fission_neutrons` option turned off so secondary fission neutrons are not produced). With a fixed source, batches are statistically independent so the estimated uncertainties reported by OpenMC are not plagued by auto-correlation effects, and these reported values are used here. The source is spatially uniform

and sampled from a fission neutron energy spectrum. Simulations used 10 batches with 100 000 particles per batch.

For representative tally examples, the flux and absorption rate were tallied in two energy groups (boundary at 0.625 eV). No other filters were used which means the tallies are homogenized across the entire pincell. Both tracklength-estimated and collision-estimated tallies were computed.

The uncertainties reported by OpenMC—the standard error of the mean—are shown in Table 4.1. The uncertainty differs by a factor of about 2× for the thermal flux tally. For the other tallies, the uncertainties are similar.

It is also worth noting that optically thin regions have an impact on the uncertainty even when the tally homogenizes over them. This pincell problem includes such a region in the form of the gap between the fuel and cladding. This gap is filled with helium gas, and that gas is modeled explicitly for the simulation behind Table 4.1. Replacing this helium gas with a void reduces the tally uncertainty as seen in Table 4.2.

For the same reason that the helium gas increases tally uncertainty—it has a very small macroscopic cross section—it is also expected to have a negligible impact on simulation results. Consequently, a void is used in place of the gas for the simulations in this thesis.

With the voided gap, the difference in uncertainty is small for each of these example tally results. Consequently, it is expected that collision-estimated tallies offer overall better performance since they decrease the MC runtime and provide a similar variance.

Table 4.1: Standard error of the mean (SEM) for tracklength- and collision-estimated tallies of a pincell problem with a **helium-filled** gap.

Tally	Tally SEM	
	Tracklength	Collision
$\langle\phi\rangle_1$	6.4	7.7
$\langle\Sigma_a\phi\rangle_1$	0.15	0.11
$\langle\phi\rangle_2$	2.6	5.9
$\langle\Sigma_a\phi\rangle_2$	0.28	0.21

Table 4.2: Standard error of the mean (SEM) for tracklength- and collision-estimated tallies of a pincell problem with a **voided** gap.

Tally	Tally SEM	
	Tracklength	Collision
$\langle\phi\rangle_1$	4.5	5.5
$\langle\Sigma_a\phi\rangle_1$	0.12	0.085
$\langle\phi\rangle_2$	2.1	1.8
$\langle\Sigma_a\phi\rangle_2$	0.24	0.23

Table 4.3: Runtime cost with and without the discussed optimizations. The values indicate the runtime spent in the MC portion of a quarter-core multiphysics simulation.

Cell-based neighbor lists	Partitioning	Collision-estimated tallies	Runtime [hours]
X	✓	✓	5.3
✓	X	✓	5.3
✓	✓	X	5.4
✓	✓	✓	4.0

4.2.4 Summary

This section has discussed a collection of software-oriented details aimed at maximizing the performance of the MC simulations used in this thesis. Table 4.3 summarizes the performance tests from the preceding discussion.

The performance gains are significant. By adding the runtime differences, the total time required for a simulation without any of these optimizations can be projected as 8.0 hours—twice the cost of an optimized simulation. It is also worth noting that these optimizations were not difficult to implement in software so the cost-to-benefit ratio of this work is quite low.

Perhaps the biggest take-away from this optimization work is that it is crucial to use software profiling techniques with MC codes and to profile them on the problems of interest. Profiling was used to discover each of the software "hot spots" discussed in this section, and most of these hot spots are only apparent on representative geometries (such as quarter-assembly problems discretized for multiphysics).

MC solvers are intricate, and subtle details can have a large impact on solver performance. Consequently, it is important to check for problem-specific inefficiencies whenever performance is an issue.

4.3 Differential tallies

4.3.1 Differential tally basics

Distributions of flux and reaction rates are the usual desired output from a Monte Carlo solver, but it is also sometimes possible to compute derivatives of these quantities with respect to temperature and other variables. These differential tallies are a crucial component of this thesis.

Differential tallies, also known as differential operator sampling, can be classified as a kind of perturbation method—a method that describes how the solution will respond to a small change in the input conditions. It is very similar to the method known as correlated sampling, and Rief described both of these methods in a 1984 publication [42]. Peplow has published a more recent description that focuses more on the details of implementation [43]. There are some important subtleties in the implementation and use of differential tallies so a derivation is provided here.

In general, a tally in a MC transport code is an approximation of a flux-weighted integral. Considering only collision-estimated tallies for simplicity, the approximation can be written in the form,

$$\int_D f(\boldsymbol{x})\psi(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \frac{1}{N}\sum_{n=1}^{N}\frac{\delta_D(\boldsymbol{x}_n)}{\Sigma_t(\boldsymbol{x}_n)}f(\boldsymbol{x}_n) \tag{4.1}$$

where $\boldsymbol{x}$ is a set of coordinates in the 6-dimension phase space of position, angle, and energy; D is some arbitrary domain of the phase space; $\psi(\boldsymbol{x})$ is the particle flux (normalized to one); $f(\boldsymbol{x})$ is an arbitrary function (it could be unity for a flux tally or a cross section for a reaction rate tally); the coordinates $\boldsymbol{x}_n$ are the locations of collisions sampled by the MC code; $\delta_D(\boldsymbol{x}_n)$ is a delta function indicating that only values of $\boldsymbol{x}_n$ that lie inside the domain, D, contribute to the summation; and $\Sigma_t(\boldsymbol{x}_n)$ is the macroscopic total cross section at $\boldsymbol{x}_n$.

For brevity, tallies will be indicated by angle brackets with the phase space variables neglected,

$$\langle f\psi\rangle_D = \frac{1}{N}\sum_{n=1}^{N}\frac{\delta_D(\boldsymbol{x}_n)}{\Sigma_t(\boldsymbol{x}_n)}f(\boldsymbol{x}_n) \tag{4.2}$$

Taking the derivative of a flux-weighted integral with respect to some parameter θ gives,

$$\frac{\partial}{\partial\theta}\int_D f(\boldsymbol{x})\psi(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \int_D f(\boldsymbol{x})\psi(\boldsymbol{x})\left(\frac{1}{f}\frac{\partial f}{\partial\theta}\bigg|_{\boldsymbol{x}} + \frac{1}{\psi}\frac{\partial\psi}{\partial\theta}\bigg|_{\boldsymbol{x}}\right)\mathrm{d}\boldsymbol{x} \tag{4.3}$$

Applying the same Monte Carlo approximation from Equation 4.1 to Equation 4.3 gives,

$$\frac{\partial}{\partial\theta}\int_D \psi(\boldsymbol{x})f(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \left\langle\frac{\partial f\psi}{\partial\theta}\right\rangle_D = \left\langle f\left(\frac{1}{f}\frac{\partial f}{\partial\theta} + \frac{1}{\psi}\frac{\partial\psi}{\partial\theta}\right)\psi\right\rangle_D \tag{4.4}$$

Essentially, computing a derivative tally amounts to replacing the function f in Equation

4.1 with a new function g given by,

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) \left(\frac{1}{f} \frac{\partial f}{\partial \theta}\bigg|_{\boldsymbol{x}} + \frac{1}{\psi} \frac{\partial \psi}{\partial \theta}\bigg|_{\boldsymbol{x}} \right) \tag{4.5}$$

For a concrete example, consider $f(\boldsymbol{x}) = \Sigma_a(\boldsymbol{x})$ for a tally of an absorption reaction rate. From Equation 4.1, this tally is implemented by summing up the quantity Σ_a/Σ_t each time a collision is sampled in the domain, D. Then, from Equations 4.4 and 4.5, the derivative of the absorption reaction rate is computed by summing up the quantity,

$$\frac{\Sigma_a}{\Sigma_t} \left(\frac{1}{\Sigma_a} \frac{\partial \Sigma_a}{\partial \theta}\bigg|_{\boldsymbol{x}} + \frac{1}{\psi} \frac{\partial \psi}{\partial \theta}\bigg|_{\boldsymbol{x}} \right)$$

at each collision.

4.3.2 The flux derivative

The complexity in implementing (and understanding) differential tallies is mostly due to the flux derivative term, the second term in the parentheses of Equation 4.5. To discuss this term, some rigor is needed in describing how the MC code computes the collision coordinates, $\boldsymbol{x}_n$.

In order for the Monte Carlo approximation (Equation 4.1) to be unbiased, the phase space coordinates, $\boldsymbol{x}_n$, must be sampled from the flux distribution, $\psi(\boldsymbol{x})$. Sampling directly from ψ is intractable so $\boldsymbol{x}_n$ values are instead sampled in a Markov chain starting with source sites. If the source location for history h is $\boldsymbol{x}_{h,0}$ then the uncollided flux resulting from that source can be written as,

$$\psi_0(\boldsymbol{x}|\boldsymbol{x}_{h,0}) = \mathbb{T}(\boldsymbol{x}, \boldsymbol{x}_{h,0}) S(\boldsymbol{x}_{h,0})$$

where S is the source distribution and $\mathbb{T}$ gives the probability of a particle traveling from $\boldsymbol{x}_{h,0}$ to $\boldsymbol{x}$.

The coordinates of the first particle collision site, $\boldsymbol{x}_{h,1}$, can then be sampled from $\Sigma_t \psi_0(\boldsymbol{x}|\boldsymbol{x}_{h,0})$. Starting from the uncollided flux distribution, ψ_0, the fluxes from c^{th}-collided neutrons can be computed recursively as,

$$\psi_c(\boldsymbol{x}|\boldsymbol{x}_{h,c-1}) = \mathbb{T}(\boldsymbol{x}, \boldsymbol{x}_{h,c-1}) \mathbb{C}(\boldsymbol{x}, \boldsymbol{x}_{h,c-1}) \psi_{c-1}(\boldsymbol{x}_{h,c-1}) \qquad c > 0 \tag{4.6}$$

where $\mathbb{C}$ gives the probability of a collision at $\boldsymbol{x}_{h,c-1}$ that results in a particle traveling towards $\boldsymbol{x}$.

The $\mathbb{T}$ and $\mathbb{C}$ terms will not be described in detail here, but are discussed at length elsewhere [42, 43, 44, 45]. The $\mathbb{T}$ term represents the exponential attenuation of a beam and $\mathbb{C}$ is the double-differential scattering cross section.

The derivative of Equation 4.6 (using subscripts on $\mathbb{T}$ and $\mathbb{C}$ to indicate the different phase space coordinates) is given by,

$$\frac{\partial \psi_c}{\partial \theta} = \mathbb{T}_c \mathbb{C}_c \psi_{c-1} \left(\frac{1}{\mathbb{T}_c} \frac{\partial \mathbb{T}_c}{\partial \theta} + \frac{1}{\mathbb{C}_c} \frac{\partial \mathbb{C}_c}{\partial \theta} + \frac{1}{\psi_{c-1}} \frac{\partial \psi_{c-1}}{\partial \theta} \right) \tag{4.7}$$

Equations 4.6 and 4.7 can be applied iteratively to find,

$$\psi_c = \prod_{i=1}^{c} \mathbb{T}_i \mathbb{C}_i S \tag{4.8}$$

$$\frac{\partial \psi_c}{\partial \theta} = \prod_{i=1}^{c} \mathbb{T}_i \mathbb{C}_i S \left(\sum_{j=1}^{c} \frac{1}{\mathbb{T}_j} \frac{\partial \mathbb{T}_j}{\partial \theta} + \sum_{j=1}^{c} \frac{1}{\mathbb{C}_j} \frac{\partial \mathbb{C}_j}{\partial \theta} + \frac{1}{S} \frac{\partial S}{\partial \theta} \right) \tag{4.9}$$

4.3.3 Practical implementation

To implement a differential tally in a MC code, the software must sum up values of the g function defined in Equation 4.5. This g function includes a flux derivative term, $\frac{1}{\psi} \frac{\partial \psi}{\partial \theta}$, which each particle accumulates as it is transported through the problem. This term can be thought of as the derivative of the particle's weight. It indicates the history of the particle, and how changes in θ affect the probability of that particle following the sampled path.

In practical terms, this means that $\frac{1}{\psi} \frac{\partial \psi}{\partial \theta}$ is just another number that must be attached to each simulated particle. In terms of computer data structures, it is analogous to the energy or weight attached to each particle. Like those values, it is a real number that is unique to each particle and it changes as the particle's flight is simulated. It must also be initialized when the particle is born, and its initialization is a detailed topic that is discussed in Section 4.4.

Note that the code must track a weight derivative for each θ of interest. For example, if derivatives due to the temperatures in three different regions—T_1, T_2, and T_3—are desired, then the code must attach three weight derivatives—$\frac{1}{\psi} \frac{\partial \psi}{\partial T_1}$, $\frac{1}{\psi} \frac{\partial \psi}{\partial T_2}$, $\frac{1}{\psi} \frac{\partial \psi}{\partial T_3}$—to each particle that it tracks. This may result in performance issues for multiphysics simulations so an

efficient approximation is discussed in Section 4.5.10.

Another subtle detail is that the same weight derivative can be re-used for multiple differential tallies that use the same θ. A single weight derivative can be used to score tallies for multiple different reactions, e.g. reusing $\frac{1}{\psi}\frac{\partial\psi}{\partial T_1}$ for the differential tallies $\frac{\partial}{\partial T_1}\langle\Sigma_a\psi\rangle$ and $\frac{\partial}{\partial T_1}\langle\Sigma_s\psi\rangle$. Similarly, a single weight derivative can be used to score tallies in multiple different regions, e.g. reusing $\frac{1}{\psi}\frac{\partial\psi}{\partial T_1}$ for $\frac{\partial}{\partial T_1}\langle\Sigma_a\psi\rangle_{D_1}$ and $\frac{\partial}{\partial T_1}\langle\Sigma_a\psi\rangle_{D_2}$.

Finally, note that implementing differential tallies for any particular θ requires that $\frac{1}{\mathbb{T}}\frac{\partial\mathbb{T}}{\partial\theta}$ and $\frac{1}{\mathbb{C}}\frac{\partial\mathbb{C}}{\partial\theta}$ can be computed by the MC solver while tracking neutrons. The derivations for functional forms of these are nuanced and so curious readers are referred elsewhere [42, 43, 44, 45]. For this work, it is worth noting that the resulting forms are sometimes quite simple and can be efficiently evaluated in a MC code. Taking the example of material density, ρ,

$$\frac{1}{\mathbb{C}}\frac{\partial\mathbb{C}}{\partial\rho} = -\frac{1}{\rho}$$

Consequently, the $\mathbb{C}$ and $\mathbb{T}$ terms for density derivatives are evaluated without approximation in OpenMC. Temperature derivatives, however, are more complex, and an approximation for $\mathbb{C}$ is used in the OpenMC implementation [45].

4.3.4 Derivatives of multigroup cross sections

For this thesis, differential tallies are ultimately used for Taylor series extrapolations of multigroup homogenized cross sections. These cross sections are used to simplify the evaluation of reaction rate integrals, and they can be defined by the relationship,

$$\int_{E_{g-1}}^{E_g}\iiint_{V_i}\int_{4\pi}\Sigma(\boldsymbol{r},E)\psi(\boldsymbol{r},E,\widehat{\boldsymbol{\Omega}})\mathrm{d}\widehat{\boldsymbol{\Omega}}\mathrm{d}\boldsymbol{r}\mathrm{d}E = \Sigma_{g,i}\int_{E_{g-1}}^{E_g}\iiint_{V_i}\int_{4\pi}\psi(\boldsymbol{r},E,\widehat{\boldsymbol{\Omega}})\mathrm{d}\widehat{\boldsymbol{\Omega}}\mathrm{d}\boldsymbol{r}\mathrm{d}E \quad (4.10)$$

where E_{g-1} and E_g are the boundaries of energy group, g; V_i is an arbitrary volume; and $E_{g,i}$ is the multigroup homogenized cross section. The word *homogenized* is frequently omitted here for brevity.

Solving Equation 4.10 for the multigroup cross section gives,

$$\Sigma_{g,i} = \frac{\int_{E_{g-1}}^{E_g} \iiint_{V_i} \int_{4\pi} \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, E, \widehat{\boldsymbol{\Omega}})\mathrm{d}\widehat{\boldsymbol{\Omega}}\mathrm{d}\boldsymbol{r}\mathrm{d}E}{\int_{E_{g-1}}^{E_g} \iiint_{V_i} \int_{4\pi} \psi(\boldsymbol{r}, E, \widehat{\boldsymbol{\Omega}})\mathrm{d}\widehat{\boldsymbol{\Omega}}\mathrm{d}\boldsymbol{r}\mathrm{d}E} \tag{4.11}$$

These integrals can be approximately evaluated using MC tallies per Equation 4.1. Replacing the integrals in Equation 4.11 with tallies gives,

$$\Sigma_{g,i} = \frac{\langle \Sigma\psi \rangle_{g,i}}{\langle \psi \rangle_{g,i}} \tag{4.12}$$

Taking the derivative of Equation 4.12 gives,

$$\frac{\partial \Sigma_{g,i}}{\partial \theta} = \frac{\left\langle \frac{\partial \Sigma\psi}{\partial \theta} \right\rangle_{g,i} - \Sigma_{g,i} \left\langle \frac{\partial \psi}{\partial \theta} \right\rangle_{g,i}}{\langle \psi \rangle_{g,i}} \tag{4.13}$$

Equation 4.13 shows that the derivative of a multigroup cross section includes the contribution from two differential tallies—one for a reaction rate and one for the neutron flux. The subtraction in the numerator of Equation 4.13 has crucial implications for this thesis that are the subject of Section 4.4.

4.4 The unperturbed source approximation for differential tallies

4.4.1 Difficulties arising from the perturbed source

Differential tallies require that a MC code keep track of the derivative of each particle's weight. Equation 4.9 shows that the derivative of the particle source distribution is needed in order to initialize this weight derivative. In other words, differential tallies must consider how changes in the variable θ will perturb the source distribution.

This perturbed source term proves to be particularly difficult to handle. Nagaya and Mori have discussed this term in detail and demonstrate that it can be significant when computing derivatives of the tallied eigenvalue, k [44, 46].

Despite its importance, there is not yet a widely-accepted method for computing the perturbed source term. One tempting approach is to propagate weight derivatives directly from particle to particle across generations. Each time a neutron samples a fission source

site for the next generation, use the neutron's weight derivative to initialize the weight derivative of the progeny neutron born from that site. However, other researchers have observed that this approach often leads to overly large statistical fluctuations that render the method unusable [42, 44].

Nagaya and Mori describe a process for performing this procedure over a limited number of *propagation batches* [44]. They show an optimization trade-off that increasing the number of propagation batches will increase the accuracy of the tallies, but also increase their variance. With this method, the number of propagation batches is a value which must be tuned for each application, and the optimal value likely depends on the reactor geometry and conditions.

Other researchers have suggested mesh-based methods for suppressing the variance of the weight derivatives (e.g. [42]). However, these methods also require tuning the shape and size of the mesh cells to match the application.

As a further complication, the derivations of these techniques do not account for acceleration methods like CMFD. They must be re-derived with acceleration in mind for use here. To avoid the extra tuning parameters and complications with acceleration, none of these techniques for the perturbed source effect are used here.

Instead of trying to accurately capture the perturbed source term, this thesis instead aims to minimize its impact. As will be shown in Section 4.4.2, the impact of the perturbed source on the derivatives of reaction rates can be quite large, but its impact on multigroup cross section derivatives is smaller. Thus, the perturbed source term will be neglected, and only the derivatives of multigroup cross sections are considered.

Also note that the ultimate application of the derivatives is to accelerate the multiphysics solver. Errors due to this approximation may impact the convergence rate, but they will not bias the final answer.

4.4.2 Impact of the perturbed source on differential tallies

A set of MC simulation results is presented here to quantify the impact of neglecting the perturbed source effect. These results focus on a local coolant density perturbation in a PWR fuel pin model.

The fuel pin model is axially homogeneous with vacuum boundary conditions at either end of the pin's 366 cm height. Precise geometry features are unimportant for the present

discussion so features such as grid spacers, a gas plenum, and fuel rod end plugs are neglected. The pitch as well as the radii of the fuel and cladding surfaces are taken from the BEAVRS benchmark specifications [47]. The fuel material is 2.4% enriched UO_2 at a uniform 600 K, and the cladding is a zirconium alloy also at 600 K. Water with 1000 ppm of added boron is used as the coolant.

Two different coolant conditions were used for the simulations:

- A uniform coolant density of 0.742 g/cm^3 (corresponding to water at 15.51 MPa and 566 K)
- Coolant with a density of 0.742 g/cm^3 everywhere except the small perturbed region with a density of 0.662 g/cm^3 (corresponding to 15.51 MPa, 600 K water).

These coolant density values were chosen because they are representative of an operating PWR. The density perturbation is limited to a small region located axially between 9/20 and 10/20 of the full rod height, intentionally introducing an axial asymmetry. Perturbations like this one that are confined to a small region highlight the impact of the perturbed source.

A set of simulations was run for each of the two coolant conditions. Each set consists of 20 independent simulations used to compute uncorrelated averages. Each simulation uses 100 000 particles per generation with 100 inactive generations and 300 active generations. In total, 600 million active neutrons contribute to the presented tally results.

Two-group (with the group boundary at 0.625 eV) reaction rates and fluxes were computed on an axial mesh with 100 uniform cells. The perturbed region fits exactly within 5 of these mesh cells. Tallies on this mesh result in values that are spatially homogenized over the 4 different geometry regions—fuel, gap, cladding, and coolant.

Figure 4-5 shows the tallied fission source distribution for the perturbed and unperturbed problems. Note that this figure shows the result of normal tallies from different simulations; differential tallies will be discussed shortly.

The unperturbed source distribution is smooth and nearly cosine-shaped, as expected. The perturbed source is similar, but it drops sharply in the region where the coolant density has been decreased. Less moderation in this region leads to fewer source neutrons. Note that eigenvalue MC solvers implicitly renormalize their underlying source distribution to match the specified neutron population. Because of this renormalization, a local drop in the source must be balanced by a gain elsewhere. Partly due to this renormalization effect,

the perturbed source is slightly higher just outside the perturbation region.

Note that the change in the source distribution is only visible near the perturbation, but it is still non-zero at far away locations. For example, the source dips slightly in the vicinity of $\frac{3}{4}H$ which is about 1 m away from the perturbation. This indicates that local perturbations can have extremely non-local effects.

Neglecting this change in the source distribution leads to an error in any differential tally results. To quantify this error, the results from differential tallies can be compared to derivatives computed via finite difference. Using the thermal absorption reaction rate as an example, this finite difference reference can be computed as,

$$\left(\frac{\partial}{\partial\rho}\left\langle\Sigma_a\phi\right\rangle_2\right)_{\text{Reference}} = \frac{\left\langle\Sigma_a\phi\right\rangle_{2,\text{perturbed}} - \left\langle\Sigma_a\phi\right\rangle_{2,\text{unperturbed}}}{\Delta\rho}$$

where $\Delta\rho$ is the density change in the perturbed region.

Note that differential tallies will find different derivatives for the two cases. For these results, the average value of the two cases is computed,

$$\left(\frac{\partial}{\partial\rho}\left\langle\Sigma_a\phi\right\rangle_2\right)_{\text{Diff. tallies}} = \frac{1}{2}\left(\left\langle\frac{\partial\Sigma_a\phi}{\partial\rho}\right\rangle_{2,\text{perturbed}} + \left\langle\frac{\partial\Sigma_a\phi}{\partial\rho}\right\rangle_{2,\text{unperturbed}}\right)$$

Derivatives of the thermal absorption reaction rate and the thermal flux were computed

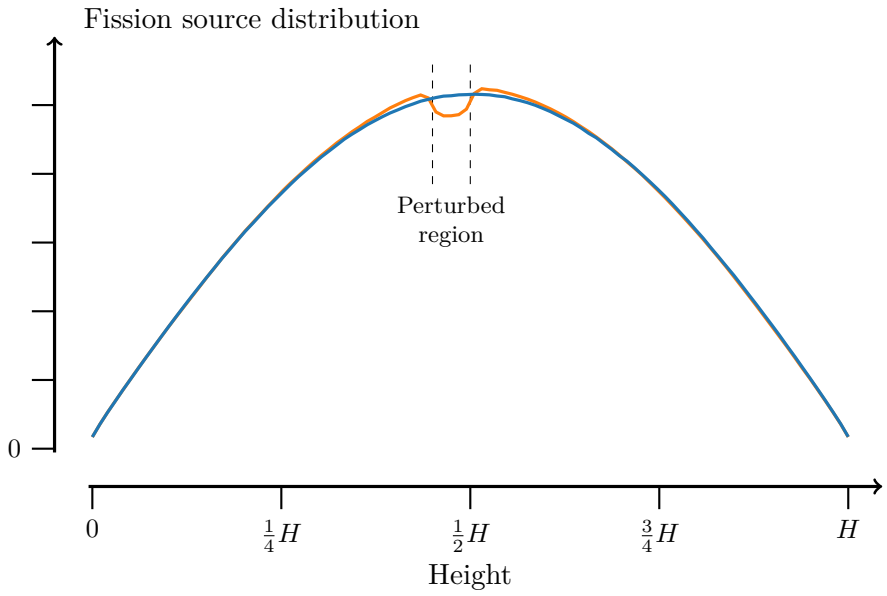


Figure 4-5: Axial fission source distribution with and without the perturbed coolant density.

using the finite difference method and OpenMC's differential tallies. The results are shown in Figures 4-6 and 4-7. Significant errors in the differential tally results are visible.

Contrast this with Figure 4-8. This figure shows the derivative of the thermal group absorption cross section. Here the differential tallies compute a significantly more accurate result. This is somewhat surprising given that the differential tally cross sections shown in Figure 4-8 are based on the erroneous derivatives shown in Figures 4-6 and 4-7 (see Equation 4.13), but there is an apparent cancellation of error.

Notice in Figure 4-6 that the reaction rate derivative is non-zero at locations like $\frac{3}{4}H$ which are far away from the perturbation. Very few neutrons at $\frac{3}{4}H$ will have traveled through the perturbed region so the non-zero derivative at this location can only be due to the perturbed source effect. In other words, the $\mathbb{T}$ and $\mathbb{C}$ terms in Equation 4.9 are zero for those neutrons so the non-zero derivative must be due to $\frac{1}{S}\frac{\partial S}{\partial \rho}$ instead.

Figure 4-6 also shows that the reaction rate derivative is underestimated in the perturbed region. This is consistent with a positive value of $\frac{1}{S}\frac{\partial S}{\partial \rho}$ i.e. a decrease in density leads to a decrease in the local source (as observed in Figure 4-5).

However, the perturbed source has a similar impact on both reaction rates and fluxes. This leads to a smaller error in the computed derivatives of multigroup cross sections as seen in Figure 4-8. Intuitively, this can be explained by observing that perturbations in the source distribution will have a large impact on the number of neutrons in any given location, but less of an impact on the energy spectrum. Multigroup cross sections depend on the neutron spectrum but do not depend on the number of neutrons which leads to a smaller dependence on the source perturbation.

Only thermal absorption tallies are considered here to simplify the discussion, but the impact of the perturbed source will differ depending on the tally. Further tally results from this set of simulations are presented later in Section 4.5.9, and they show the impact of the perturbed source on other multigroup cross sections.

4.5 Diffusion solver

For this work, Monte Carlo neutronics is paired with a simpler 2-group diffusion theory solver. Like MC, this solver aims to find a distribution of neutron flux, $\boldsymbol{\phi}$, which solves the neutron transport equation. Unlike MC, many simplified models are applied to the neutron

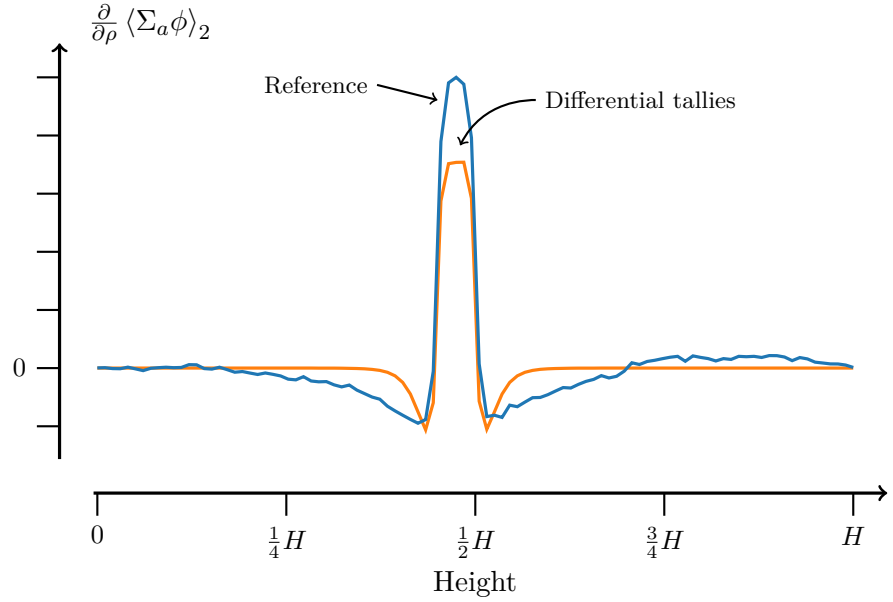


Figure 4-6: Derivative of the thermal absorption reaction rate. Reference results computed via finite difference are shown along with values computed via differential tallies.

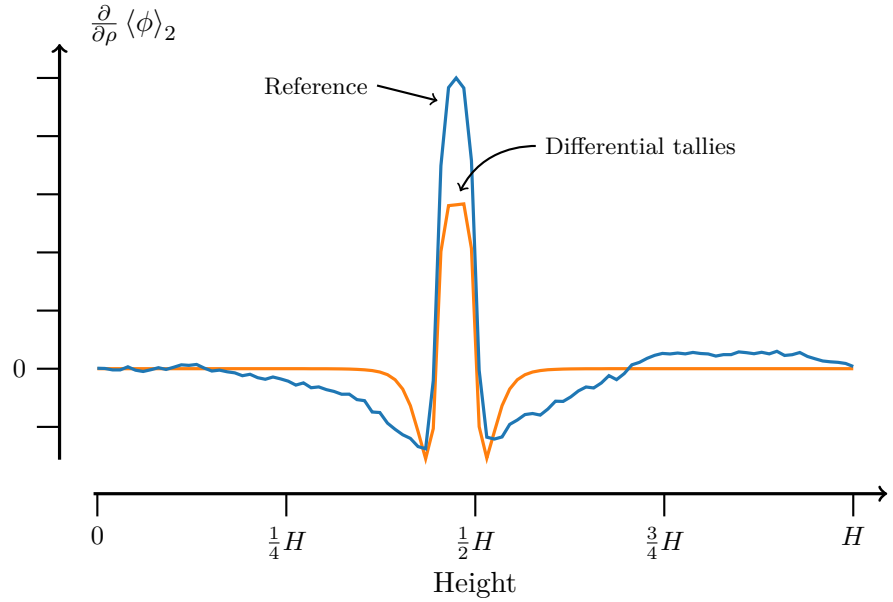


Figure 4-7: Derivative of the thermal flux. Reference results computed via finite difference are shown along with values computed via differential tallies.

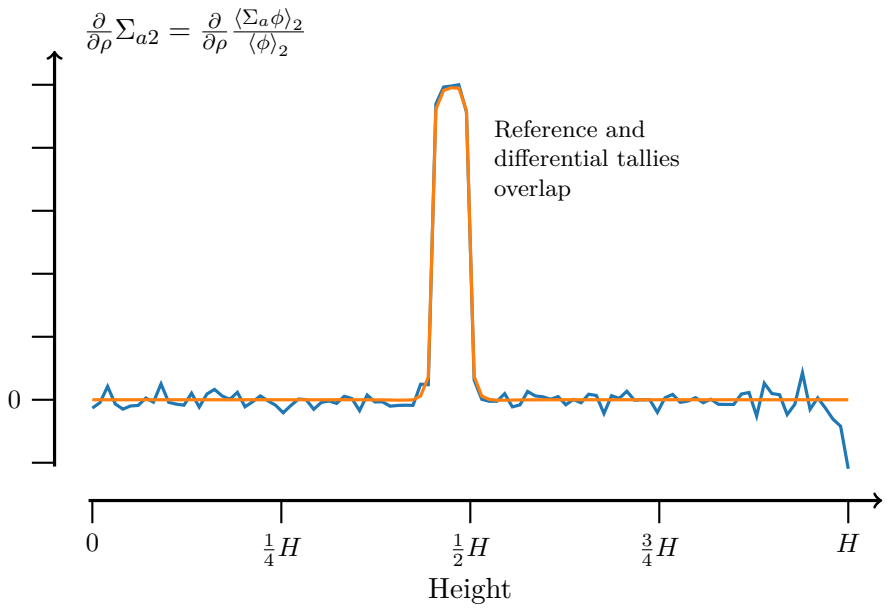


Figure 4-8: Derivative of the thermal absorption cross section. Reference results computed via finite difference are shown along with values computed via differential tallies.

physics. These simple models make the solver faster, but they also make it dependent on MC or other high fidelity methods for accuracy.

Details of the solver are provided in this section. First, a quick overview of the algorithmic structure is given. Next, the simplified and discretized form of the transport equation is discussed. Details are then provided on the numerical iteration procedures with some attention to their implementation in computer software. Finally, the linkage between the MC and diffusion solvers is discussed, including the novel techniques based on MC differential tallies.

4.5.1 Structure of the solver

To tackle the non-linearities in the neutron transport equation, the diffusion solver operates over three nested iteration loops as shown in Algorithm 3.

The outermost loop is formed by the multiphysics iteration procedure. The diffusion solver relies on parameters—multigroup cross sections and diffusion coefficients—which depend on the reactor temperature and density distributions, and these parameters are updated in the outer loop. Not shown in Algorithm 3 is that the other physics solvers for heat transfer and fluid dynamics must be included in the multiphysics iteration loop. That topic

is addressed further in Chapter 5.

The middle loop in Algorithm 3 focuses on the source term, $\boldsymbol{Q}$, of the transport equation. In an operating power reactor, fission and other neutron-induced reactions are the only significant source of neutrons. By neglecting all other sources (e.g. photon-induced reactions or α-induced reactions), the neutron transport equation becomes an eigenvalue equation. A detailed description of the methods used to solve the eigenvalue equation and why they work will not be provided here, but such descriptions can be found in reactor physics textbooks such as the one written by Bell and Glasstone [48].

For the inner loop of Algorithm 3, every variable of the transport equation except the flux, $\boldsymbol{\phi}$, is held constant. The transport equation can then be discretized into a linear system of equations. For this work, that linear system is solved using Gauss-Seidel iteration.

4.5.2 Fundamental conservation equation

This derivation will begin with the steady-state multigroup neutron transport equation integrated over all spatial directions. Derivations of this equation from a more general form of the transport equation can be found in [48]. For energy group, g, this equation can be written in the form,

$$\nabla \cdot \boldsymbol{J}_g(\boldsymbol{r}) + \Sigma_{r,g}(\boldsymbol{r})\phi_g(\boldsymbol{r}, E) = Q_g(\boldsymbol{r}) \tag{4.14}$$

where $\boldsymbol{r}$ is the position in 3D space, $\boldsymbol{J}$ is the neutron current, ϕ is the neutron flux (integrated over all directions), Σ_r is a macroscopic removal cross section, and Q is a source of neutrons.

This equation is discretized by integrating over an arbitrary finite volume V_i and applying the divergence theorem to find,

$$\iint_{\partial V_i} \boldsymbol{J}_g(\boldsymbol{r}) \cdot \hat{\boldsymbol{n}}(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} + \iiint_{V_i} \Sigma_{r,g}(\boldsymbol{r})\phi_g(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} = \iiint_{V_i} Q_g(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} \tag{4.15}$$

where ∂V_i is the surface enclosing V_i, and $\hat{\boldsymbol{n}}$ is the normal vector of that surface.

Algorithm 3 Basic structure of diffusion neutronics solver

```
for Multiphysics iteration do
    Update solver parameters (e.g. cross sections) based on temperature and density
    for Eigenvalue iteration do
        Compute the fission neutron source, Q
        for Linear iteration do
            Solve for the neutron flux φ
```

Shorthand variables, $\phi_{g,i}$ and $Q_{g,i}$, are introduced which indicate the average values of the flux and source over volume i. Additionally, the spatially homogenized removal cross section, $\Sigma_{r,g,i}$, is introduced. These variables can be defined by the relationships,

$$\iiint_{V_i} \phi_g(\boldsymbol{r})\mathrm{d}\boldsymbol{r} = \phi_{g,i}V_i$$
$$\iiint_{V_i} Q_g(\boldsymbol{r})\mathrm{d}\boldsymbol{r} = Q_{g,i}V_i$$
$$\iiint_{V_i} \Sigma_{r,g}(\boldsymbol{r})\phi_g(\boldsymbol{r})\mathrm{d}\boldsymbol{r} = \Sigma_{r,g,i}\phi_{g,i}V_i$$

Using these variables, Equation 4.15 can be written in the shorter form,

$$\iint_{\partial V_i} \boldsymbol{J}_g(\boldsymbol{r}) \cdot \hat{\boldsymbol{n}}(\boldsymbol{r})\mathrm{d}\boldsymbol{r} + \Sigma_{r,g,i}\phi_{g,i}V_i = Q_{g,i}V_i \tag{4.16}$$

Equation 4.16 is the fundamental conservation equation used for the diffusion neutronics solver. It expresses a balance of neutrons over volume i and energy group g. It is important that the diffusion solver models this equation in a way that is consistent with the MC solver. The following subsections will discuss how the diffusion solver is parameterized and how those parameters will be computed in a consistent fashion from MC tallies. As a result, both solvers will agree (to within statistical precision) on the final converged values for $\phi_{g,i}$.

4.5.3 Group-specific conservation equations

The Σ_r and the Q terms are general expressions of the loss and gain of neutrons due to collisions and nuclear reactions. These terms will now be expressed in a more specific form. For this purpose, the solver is restricted to the conventional 2-group structure.

In this structure, energy group 1 indicates fast neutrons, and group 2 indicates thermal. It is assumed that all fission neutrons are born in group 1. It is also assumed that the only source of thermal neutrons is from the downscattering of fast neutrons. With this structure, the specific form of Equation 4.16 for each of group is,

$$\iint_{\partial V_i} \boldsymbol{J}_1(\boldsymbol{r}) \cdot \hat{n}\mathrm{d}\boldsymbol{r} + \left(\Sigma_{a,1,i} + \Sigma_{d,i}\right)\phi_{1,i}V_i = Q_{1,i}V_i \tag{4.17a}$$
$$\iint_{\partial V_i} \boldsymbol{J}_2(\boldsymbol{r}) \cdot \hat{n}\mathrm{d}\boldsymbol{r} + \Sigma_{a,2,i}\phi_{2,i}V_i = \Sigma_{d,i}\phi_{1,i}V_i \tag{4.17b}$$

where Σ_a is an effective absorption cross section, Σ_d is an effective downscatter cross

section, and Q now includes just the source of neutrons due to fission. The cross sections will be computed using MC tallies as discussed in Section 4.5.8.

4.5.4 Neutron streaming

The first term in Equation 4.16 describes the change in neutron population due to neutrons traveling in and out of volume V_i, and it is often called the *streaming* term. It is analogous to similar terms seen in the fluid mass, momentum, and energy transport equations; and it raises equal difficulties in the development of numerical solvers. Its treatment is described in this subsection.

The streaming term can be computed very precisely with MC. However, it is helpful to introduce simple models for the finite difference solver that describe integrals of the current, $\boldsymbol{J}$, in terms of the flux, ϕ, in nearby regions.

Stated more formally, a model is desired where the net current between two adjacent volumes, i and j, can be expressed as some function of the fluxes in those volumes, $\phi_{g,i}$ and $\phi_{g,j}$,

$$\iint_{A_{i,j}} \boldsymbol{J}_g \cdot \hat{\boldsymbol{n}}(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} = f\left(\phi_{g,i}, \phi_{g,j}\right) \tag{4.18}$$

Note that more complex models that consider the flux in more than just two cells could be used. However, this solver must be very fast in order to be useful, and simple models are likely to be faster. Furthermore, this solver will operate on a coarse mesh (discussed in Section 4.5.5) where the size of each mesh cell is large in comparison to the typical distances that neutrons travel in a reactor. Thus, the flux in each mesh cell will have little direct effect on non-adjacent mesh surfaces.

CMFD (coarse mesh finite difference) is one popular model for the net current that has been used for decades in order to accelerate higher-fidelity methods. Smith described this technique briefly in a 1983 publication [6]. A more detailed description was later offered by Sutton and Aviles [7]. These early descriptions are focused on the use of CMFD with deterministic solvers, but it has also recently been used with MC as described by Lee et al., for example [8]. The same techniques are used for this thesis, and they will be described here for completeness.

The CMFD model parameterizes the net current with two terms expressing the sum

and difference of the adjacent cell fluxes,

$$\iint_{A_{i,j}} \boldsymbol{J}_g \cdot \hat{\boldsymbol{n}}(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} = \alpha_{i,j,g} \left(\phi_{g,j} - \phi_{g,i}\right) + \beta_{i,j,g} \left(\phi_{g,i} + \phi_{g,j}\right) \tag{4.19}$$

where α and β are model coefficients that will be discussed shortly.

The first term in Equation 4.19 is an intuitive contribution that expresses a tendency for neutrons to flow from high-flux regions to low-flux regions, consistent with diffusion theory. The second term can be used as a correction to the first, and it allows for a current even when there is no flux difference between the regions.

The form of α can be derived from diffusion theory beginning with Fick's law which states,

$$\boldsymbol{J}_g(\boldsymbol{r}) \approx -D_g(\boldsymbol{r}) \nabla \phi_g(\boldsymbol{r}) \tag{4.20}$$

where D is a diffusion coefficient defined by this relationship. This coefficient is discussed further in Section 4.5.8.

A numerical discretization scheme is also needed in order to apply Fick's law; the mesh-centered finite difference scheme is used here. Hébert (among others) describes this numerical discretization and derives an α model in the form [49],

$$\alpha_{i,j,g} = \frac{2 D_{g,j} D_{g,i}}{\Delta_j D_{g,i} + \Delta_i D_{g,j}} A_{i,j} \tag{4.21}$$

where Δ is the width of a region along the axis of interest and $A_{i,j}$ is the area of the surface between the regions.

The β coefficient is then defined in terms of α. Solving Equation 4.19 for β gives,

$$\beta_{i,j,g} = \frac{\iint_{A_{i,j}} \boldsymbol{J}_g \cdot \hat{\boldsymbol{n}}(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} - \alpha_{i,j,g} \left(\phi_{g,j} - \phi_{g,i}\right)}{\left(\phi_{g,i} + \phi_{g,j}\right)} \tag{4.22}$$

With the CMFD acceleration method, the finite difference solver is used in concert with a higher fidelity solver like MC. Flux and net current values are computed using the high-fidelity solver, and then the value of β is computed from Equation 4.22 using these fluxes and currents. Crucially, even unconverged results from a high-fidelity solver can be used to compute usefully-accurate values of β. This fact allows the finite difference method to be used in the early iterations of the high-fidelity solver thus making it useful for accelerating the high-fidelity solver.

Also note that because β is computed using the high-fidelity fluxes and currents, the net currents computed using Equation 4.19 will precisely reproduce the net currents from the high-fidelity solver, given the same flux values. If the other terms in the transport equation, Equation 4.16, are computed in a consistent manner, then the finite difference solver will faithfully reproduce converged results from its parent high-fidelity solver.

Unfortunately, different authors tend to use different notations and conventions for these finite difference relationships. This author is no different so β is not used directly, but it is instead replaced with $\widehat{D}$ which here is defined as,

$$\widehat{D}_{i,j,g} = \frac{\beta_{i,j,g}}{A_{i,j}} \tag{4.23}$$

Note that other works frequently use the symbol $\widehat{D}$ to represent what is here referred to with β.

This parameterization in terms of A is used because larger surfaces will lead to proportionally greater currents, all else being equal. Thus for different mesh interfaces with different surface areas, $\widehat{D}$ values are easier to compare than β values—a useful feature for debugging software.

In summary, the current between regions, i and j, is computed with the relationship,

$$\iint_{A_{i,j}} \boldsymbol{J}_g \cdot \hat{\boldsymbol{n}}(\boldsymbol{r}) \mathrm{d}\boldsymbol{r} = \frac{2D_{g,j}D_{g,i}}{\Delta_j D_{g,i} + \Delta_i D_{g,j}} A_{i,j} \left(\phi_{g,j} - \phi_{g,i}\right) + \widehat{D}_{g,i,j} A_{i,j} \left(\phi_{g,j} + \phi_{g,i}\right) \tag{4.24}$$

A model is also needed for the problem boundary conditions. For reflective boundary conditions, the net current is simply zero,

$$\iint_{A_{i,j}} \boldsymbol{J}_g \mathrm{d}\boldsymbol{r} = 0 \tag{4.25}$$

All other boundaries are treated with the model,

$$\iint_{A_{i,j}} \boldsymbol{J}_g \mathrm{d}\boldsymbol{r} = \frac{2D_{g,i}}{\Delta_i + 4D_{g,i}} A_{i,j}\phi_{g,i} + \widehat{D}_{g,i,j} A_{i,j}\phi_{g,i} \tag{4.26}$$

The first term in Equation 4.26 is a vacuum boundary condition derived from diffusion theory using mesh-centered finite differences as described by Hébert [49]. The choice to use a vacuum boundary condition is somewhat arbitrary; deficiencies in the diffusion theory

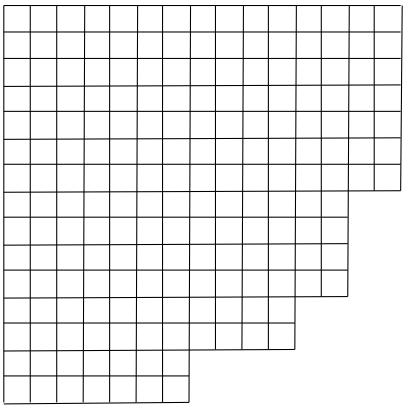


Figure 4-9: Diffusion solver mesh in the xy-plane for the BEAVRS quarter-core model. There are 4 mesh cells per assembly. Refer to Figure 7-1 for a similar image indicating fuel assemblies rather than mesh cells.

boundary condition are mitigated by the $\widehat{D}$ term. A better model may result in faster convergence for the overall solver, but this model will still be free of bias.

Note that there is just one flux value on the right-hand-side of Equation 4.26, and it can be factored out of both terms. This suggests that the solver could be simplified by ignoring the first term and using just the $\widehat{D}$ term to compute the boundary current. However, as discussed in Section 4.5.9, derivatives will be computed for the D values but not the $\widehat{D}$ values. Consequently, this two-term model is needed to make the boundary current feedback-dependent.

4.5.5 Solver mesh

The solver is restricted to 3D rectilinear meshes. An xy-slice of the mesh for a quarter-core reactor model is shown in Figure 4-9. The cells seen in this mesh are the volumes, V_i, referred to in the previous derivations.

Mesh size in x and y

The mesh uses 4 cells per fuel assembly in the xy-plane. A small number of mesh cells is preferable for a number of reasons. First, it reduces the number of flux and current values that must be computed so each iteration of the solver is faster. Second, it also reduces the required number of linear solver iterations. The reason for this is subtle, but it can be roughly explained by observing that if a mesh is N cells wide then a minimum of N linear iterations are needed for information to propagate from one side of the mesh to the

other. Third, cross sections are tallied from MC on this same mesh, and large mesh cells will collect more tally events which leads to less stochastic noise.

It is possible to use just one mesh cell per assembly. However, note that many PWRs have a row of assemblies that lie exactly on the centerline of core. A quarter-core model requires the use of reflective or periodic boundary conditions that slice through this row. As a result, some of these only lie halfway within the bounds of the model. Due to the boundary conditions, only one quarter of the very center assembly lies within the model.

If one mesh cell is used per assembly, then the cell for the center assembly will be 4 times smaller than most other mesh cells. The MC-tallied cross sections for this assembly will consequently have more stochastic noise which complicates the process of checking for solver convergence.

Furthermore, Keady and Larsen have demonstrated that overly large mesh cells can generate numerical instabilities with the MC-CMFD method [9]. It may be the case that 4 mesh cells provides better stability than 1 mesh cell.

Mesh boundaries in x and y

Note that the mesh shown in Figure 4-9 is ragged—some rows have fewer cells than others. This is because reactor designers have solved the classic "square peg in a round hole" problem by placing the square fuel assemblies in a roughly circular pattern. The boundaries of the mesh are set to match the resulting shape of the assemblies in the core.

In principle, the diffusion mesh can be made non-ragged by extending it beyond the fuel assemblies to include the core baffle, former, barrel, and so on. In fact, this non-ragged approach was used with early versions of the present solver. However, difficulties were encountered due to stochastic MC tally noise. The flux in these outer regions is relatively low, and MC tallies are consequently very noisy.

Despite the low flux, the solver proved very sensitive to the tallied cross sections and diffusion coefficients in the far-removed regions. The resulting diffusion solutions were highly inaccurate and effectively useless. Limiting the mesh to the actively-fueld core in the xy-plane reduced the stochastic noise issues and made the solver practical.

The resulting outer mesh boundary is concave which is generally not desirable for neutronics solvers. With this mesh, it is possible for neutrons traveling in a straight line to leave and then re-enter the mesh. This may raise issues with a general diffusion solver, but

here the effect will be resolved by MC and accounted for with the $\widehat{D}$ factor in Equation 4.26.

One detail not visible in Figure 4-9 is that the mesh is slightly irregular in the xy-plane. A perfectly regular mesh would result in many fuel pins precisely straddling the border between two different mesh cells. (This is true for fuel assemblies that have an odd number pins per side, such as the 17×17 assemblies in the BEAVRS model.)

Slicing fuel pins in two provides no inherent challenge to the diffusion solver, but it complicates the use of the solver in a multiphysics simulation. As discussed in Chapter 5, some model is needed that relates the power produced in each fuel pin to the coarse-mesh diffusion solution. If every pin fits into just one mesh cell, then a simple model can be used which relates one pin to one mesh cell. For this reason, the first mesh cell is wide enough to fit 9 fuel pins, the second fits 8 fuel pins, the third fits 9, and so on.[3]

Note that for more general diffusion solvers as used in the nuclear industry, this process is called *pin power reconstruction*. Likely, these solvers use more sophisticated methods for pin power reconstruction, and pins that straddle a border are no issue. However, sophisticated reconstruction methods are not necessary here given that the solver is used concurrently with a MC solver. Ultimately, the MC code is responsible for computing the fine details of the flux within each coarse mesh cell, and consequently the pin power distribution.

Mesh boundaries along z

The mesh spacing along the z-axis was similarly chosen in a manner that considers the numerics of the solver, the MC tally uncertainty, and the multiphysics feedback.

First, under the assumption that neutron diffusion is a roughly isotropic phenomenon, it is numerically beneficial for the solver to have a similar spacing along all three axes. Using 4 mesh cells per assembly in the xy plane gives a mesh size of about 10 cm so this suggests a roughly 10 cm spacing along z.

A mesh that resolves the assembly spacer grids is also desirable. These grids displace a significant amount of coolant which results in a non-smooth shape in the axial distribution of neutron cross sections. For this reason, mesh boundaries are placed at the top and bottom of each spacer grid. Mesh boundaries are also placed at the top and bottom of the

[3] Actually, the first mesh cell is 8.5 fuel pins wide. The first row of fuel pins are split by the boundary condition, but this is no issue as they do not straddle two mesh cells. The power of these split pins can be unambiguously assigned to the first mesh cell.

fuel. (Swelling of the fuel is neglected here so the top of the fuel is a well-defined constant.) These constrained boundaries necessitate an irregular mesh.

The same argument can be used to suggest placing mesh boundaries at the bottom of control rods as well as the tops and bottoms of the burnable absorbers. However, an overly constrained mesh may lead to small mesh sizes which pose numerical stability and tally noise issues. Consequently, the mesh is not constrained to match those geometry features.

The mesh is generated by first specifying each of these constrained boundaries. Then, each segment in between the boundaries is filled with the smallest number of mesh cells that yields a spacing less than 10 cm. For example, if the distance between two adjacent spacer grids is 46.5 cm, then that interval will be discretized with 5 mesh cells, each 9.3 cm high. The typical spacer grid in the BEAVRS model is about 6 cm high, so each spacer grid will be discretized with 1 mesh cell that is 6 cm high.

For the quarter-core BEAVRS model, this procedure results in 42 total mesh cells along the z-axis in the fueled region. This is similar in number to the 49-cell mesh used by Kelly et al. to simulate a quarter-core multiphysics problem [3].

There is one final detail of the z-mesh worth noting: as a legacy of the solver's development, the diffusion mesh includes one extra cell below the fuel and another above the fuel. Their intended purpose is only for simple debugging calculations—they allow for slightly more accurate solutions on single-assembly problems that do not include $\widehat{D}$ corrections. These extra mesh cells may have some impact on the convergence rate of the overall solver, but there is no known reason to expect this impact is large.

4.5.6 Linear system solver

Recall the basic structure of the diffusion solver outlined in Algorithm 3. For the innermost loop of the solver, the cross sections, diffusion coefficients, $\widehat{D}$ factors, and the fission source distribution can be considered constant. With these values fixed, Equations 4.17 and 4.24 form a system of linear equations with the neutron flux as the independent variable.

Solving this linear system quickly is crucial for the overall multiphysics solver performance, and parallel iterative solvers offer an easy way to achieve this performance. In particular, Gauss-Seidel iteration is a method that is known to be efficient for neutron diffusion problems, and this is the method chosen for this work.

Algorithm 4 shows a simple but inefficient linear iteration procedure for reference. This

algorithm includes just three nested loops that each step across consecutive mesh indices. Here the physics is packed into the simple statement, "Update $\phi_{1,i}$ and $\phi_{2,i}$," for brevity. Note that each mesh cell has 6 bounding surfaces, and the current across each surface must be computed using Equation 4.24, 4.25, or 4.26. This introduces a dependency between each cell and its neighbors.

Algorithm 4 could be used with a Jacobi iteration to make a parallel solver. Jacobi iteration means that the neutron current (Equation 4.24) is computed using flux values, ϕ, from the previous linear iteration. However, the solver will be faster if up-to-date values of ϕ are used when they are available, and this is referred to as Gauss-Seidel iteration. Gauss-Seidel slightly complicates a parallel solver as the results now depend on the order in which ϕ_i and ϕ_{i+1} are computed.

Because of this dependency, using Algorithm 4 with OpenMP parallelism will make the solver non-replicable—the same inputs may produce different results depending on the scheduling of parallel operations. It is also likely inefficient due to one thread frequently invalidating the cache lines of other threads. Furthermore, compiler optimizers will recognize the data dependency and will therefore not use SIMD operations.

For better parallel performance, Algorithm 5 is used instead. This is a red-black Gauss-Seidel scheme where data dependency is avoided by updating the flux in two disconnected passes. For example, ϕ_i and ϕ_{i+2} are updated in the first pass, and then ϕ_{i+1} and ϕ_{i+3} are updated in the second pass. Because the current in mesh cell i does not depend on the flux in cell $i+2$, the flux in both cells can be evaluated concurrently or in any order without changing the results.

Here the red-black iteration is applied to the loops over x and z with each loop targeting a different type of parallelism. The loops over z are explicitly parallelized with OpenMP shared-memory parallelism using the `#pragma omp parallel for` directive. No explicit parallelism is applied to the x loops, but pains are taken to make the lack of a data dependency clear so that the compiler optimizer uses SIMD operations for these loops. This

Algorithm 4 A simple linear update loop

```
1: for i_z in 0, 1, ..., n_z − 1 do
2:     for i_y in 0, 1, ..., n_y − 1 do
3:         for i_x in 0, 1, ..., n_{x,i_y} − 1 do
4:             i ← n_x n_y i_z + n_x i_y + i_x
5:             Update φ_{1,i} and φ_{2,i}
```

includes exposing the ϕ values as a simple C-style array within the solver and in-lining the software functions that implement the "Update $\phi_{1,i}$ and $\phi_{2,i}$" steps. The binary file produced by the compiler was disassembled and inspected to confirm that SIMD operations are used for these loops.

A further detail not shown in Algorithm 5 is that mesh cells next to the boundaries are handled separately from the interior mesh cells. This is due to the complexity of the "Update $\phi_{1,i}$ and $\phi_{2,i}$" introduced by the boundaries. This function includes several branch cases for when there is one, two, or more adjacent boundaries and whether they use vacuum or reflective boundary conditions. These branch cases slow the evaluation of this function and might prevent SIMD optimization, so it is beneficial to separate them from the primary loops.

The full Gauss-Seidel linear iteration requires repeatedly cycling through Algorithm 5 until the ϕ values are converged. After each loop, the new ϕ values are compared against their previous value, and the iteration ends when when the root-mean-square of the relative flux change is below a tolerance parameter.

The linear tolerance parameter will vary depending on the convergence of the eigenvalue iteration loop (the next largest loop in Algorithm 3). In the early iterations when the eigenvalue source is not well known, it is unnecessary to tightly converge the linear system.

Algorithm 5 A parallel-friendly red-black Gauss-Seidel linear update loop

```
1:  for i_z in 0, 2, ..., n_z − 1 do
2:      for i_y in 0, 1, ..., n_y − 1 do
3:          for i_x in 0, 2, ..., n_{x,i_y} − 1 do
4:              i ← n_x n_y i_z + n_x i_y + i_x
5:              Update φ_{1,i} and φ_{2,i}
6:          for i_x in 1, 3, ..., n_{x,i_y} − 1 do
7:              i ← n_x n_y i_z + n_x i_y + i_x
8:              Update φ_{1,i} and φ_{2,i}
9:  for i_z in 1, 3, ..., n_z − 1 do
10:     for i_y in 0, 1, ..., n_y − 1 do
11:         for i_x in 0, 2, ..., n_{x,i_y} − 1 do
12:             i ← n_x n_y i_z + n_x i_y + i_x
13:             Update φ_{1,i} and φ_{2,i}
14:         for i_x in 1, 3, ..., n_{x,i_y} − 1 do
15:             i ← n_x n_y i_z + n_x i_y + i_x
16:             Update φ_{1,i} and φ_{2,i}
```

The tolerance is set to

$$\text{Linear tolerance} = \max\left(\text{Err}_{\text{eig}} \times 10^{-7}, 10^{-12}\right)$$

where Err_{eig} is the root-mean-square of the relative flux change between the previous two eigenvalue iterations.

This tolerance is likely much smaller than needed, but the overall multiphysics solver runtime is insensitive to this parameter.

4.5.7 Eigenvalue iteration

As shown in Algorithm 4, the Gauss-Seidel linear iteration procedure occurs within a non-linear eigenvalue iteration loop. This loop uses power iteration to find the fundamental mode solution of the eigenvalue neutron transport problem. For each iteration, the neutron source is computed from the most up-to-date neutron flux as,

$$Q_{1,i} = C\left(\nu\Sigma_{f,1,i}\phi_{1,i} + \nu\Sigma_{f,2,i}\phi_{2,i}\right) \tag{4.27}$$

where $\nu\Sigma_f$ is the neutron production cross section and C is a normalization factor included so that the source integrates to the same value at the start of each iteration.

Like the linear solver, convergence is established based on the root-mean-square of the relative flux change between nonlinear iterations. The tolerance for convergence of this value is set to 10^{-7} for the simulations presented here.

4.5.8 Multigroup cross sections and diffusion coefficients

The diffusion solver relies on many parameters computed by the MC solver. The simplest of these parameters are the 7 cross sections and diffusion coefficients that are computed for each mesh cell. These 7 are listed in Table 4.4.

The cross sections are computed via Monte Carlo tallies on the same rectilinear Cartesian mesh used by the diffusion solver. The calculation of typical cross sections (using the thermal absorption cross section as an example) for each mesh cell can be expressed as,

$$\Sigma_{a,2,i} = \frac{\langle\Sigma_a\phi\rangle_{2,i}}{\langle\phi\rangle_{2,i}} \tag{4.28}$$

Table 4.4: The volumetric multigroup coefficients used in the diffusion solver and computed for each mesh cell.

D_1	Fast diffusion coefficient
D_2	Thermal diffusion coefficient
Σ_{a1}	Effective fast absorption cross section (slightly decreased to account for non-fission multiplying reactions like (n, 2n))
Σ_{a2}	Thermal absorption cross section
Σ_d	Effective fast-to-thermal downscatter cross section (slightly decreased to account for upscatter)
$\nu\Sigma_{f1}$	Fast fission production cross section
$\nu\Sigma_{f2}$	Thermal fission production cross section

where the brackets indicate a MC tally over the relevant mesh cell and the energy group, ϕ is the continuous-in-space and continuous-in-energy neutron flux, and Σ_a is the continuous reaction rate.

Note that there are non-fission reactions like $(\mathrm{n}, 2\mathrm{n})$ which multiply the number of neutrons in the system. In order to account for these reactions, a yield value, ν_s, is computed that gives the average number of outgoing neutrons from neutron scattering reactions. It is analogous to the fission yield, ν, and it is computed from the equation,

$$\nu_s = \frac{\langle \nu_s \Sigma_s \phi \rangle_{1,i}}{\langle \Sigma_s \phi \rangle_{1,i}}$$

where the numerator is computed from the OpenMC "nu-scatter" score. The vast majority of outgoing neutrons from these reactions are fast, so ν_s is only computed for the fast group.

The diffusion solver accounts for these non-fission multiplying reactions by using an effective fast absorption cross section computed from,

$$\Sigma_{a,1,i} = \frac{\langle \Sigma_a \phi \rangle_{1,i} - (\nu_s - 1) \langle \Sigma_s \phi \rangle_{1,i}}{\langle \phi \rangle_{1,i}}$$

Similarly, an effective downscatter cross section is computed from the equation,

$$\Sigma_{d,i} = \frac{\langle \Sigma_s \phi \rangle_{1\to 2,i} - \langle \Sigma_s \phi \rangle_{2\to 1,i}}{\langle \phi \rangle_{1,i}}$$

where the subscripts indicate tallies that are filtered for neutrons traveling from group 1 to

group 2 or vice-versa.

Diffusion coefficients are difficult to compute with a MC solver. Liu et al. discuss these difficulties and offer an accurate method for computing the diffusion coefficient, but unfortunately, this method cannot be used to compute the coefficient in multiple tally regions simultaneously [50]. However, an approximate method will suffice for this thesis. The effect of errors in the diffusion coefficient are mitigated by the $\widehat{D}$ correction terms used in the diffusion solver.

First, it is approximated that diffusion coefficients can be condensed in energy from the equation,

$$D_g(\boldsymbol{r}) \approx \frac{\int_{E_g}^{E_{g+1}} D_g(\boldsymbol{r}, E)\phi(\boldsymbol{r}, E)\mathrm{d}E}{\int_{E_g}^{E_{g+1}} \phi(\boldsymbol{r}, E)\mathrm{d}E} \tag{4.29}$$

This is known as the flux-limited approximation [50].

Diffusion coefficients can be related to another parameter, Σ_{tr}, known as the transport cross section by the equation [48],

$$D(\boldsymbol{r}, E) = \frac{1}{3\Sigma_{\mathrm{tr}}(\boldsymbol{r}, E)} \tag{4.30}$$

Using the transport cross section, Equation 4.29 can be expressed as,

$$D_g(\boldsymbol{r}) \approx \frac{\int_{E_g}^{E_{g+1}} \frac{1}{3\Sigma_{\mathrm{tr}}(\boldsymbol{r},E)}\phi(\boldsymbol{r}, E)\mathrm{d}E}{\int_{E_g}^{E_{g+1}} \phi(\boldsymbol{r}, E)\mathrm{d}E} \tag{4.31}$$

One difficulty with Equation 4.31 is that the transport cross section is equal to zero in voided regions of the geometry, leading to an undefined result. To circumvent this issue, the transport cross section in Equation 4.31 is replaced with its spatially homogenized value. All tally regions include some non-void material so the resulting cross section is always non-zero. The new expression for the diffusion coefficient is then given by,

$$D_{g,i} \approx \frac{\int_{E_g}^{E_{g+1}} \frac{1}{3\Sigma_{\mathrm{tr},i}(E)}\phi_i(E)\mathrm{d}E}{\int_{E_g}^{E_{g+1}} \phi_i(E)\mathrm{d}E} \tag{4.32}$$

The transport cross section in turn can be approximately defined as, [48]

$$\Sigma_{\mathrm{tr}}(\boldsymbol{r}, E) \approx \Sigma_t(\boldsymbol{r}, E) - \Sigma_{s1}(\boldsymbol{r}, E)$$

where Σ_{s1} is the first Legendre moment of double-differential scattering cross section.

The cross section defined by Equation 4.5.8 can be tallied in the usual fashion to arrive at values which are spatially homogenized and condensed in energy,

$$\Sigma_{\text{tr},g,i} = \frac{\langle \Sigma_t \phi \rangle_{g,i} - \langle \Sigma_{s1} \phi \rangle_{g,i}}{\langle \phi \rangle_{1,i}}$$

However, Equation 4.32 shows that a condensed $\frac{1}{\Sigma_{\text{tr}}}$ is required instead of Σ_{tr}. For this reason, the Σ_{tr} values are first condensed on a fine energy grid. The fine grid values are then used to compute the coefficient as,

$$D_{g,i} \approx \frac{\sum_{h \in g} \frac{1}{3\Sigma_{\text{tr},h,i}} \phi_{h,i}}{\sum_{h \in g} \phi_{h,i}} \tag{4.33}$$

where h is used to indicate energy groups on the fine grid.

The fine grid includes 20 energy groups. The first 10 groups of this fine grid are a subset of the fast group in the 2-group structure, and they spaced equally in lethargy. The last 10 fine groups are similarly equal-lethargy spaced in the thermal group.

To summarize: the Σ_t and Σ_{s1} cross sections are computed from tallies over a 20-group energy grid on the diffusion solver's spatial mesh. Transport cross sections are then computed from these values as, $\Sigma_{\text{tr}} = \Sigma_t - \Sigma_{s1}$. These fine-group transport cross sections are then used to compute 2-group diffusion coefficients per Equation 4.33. The resulting diffusion coefficients are approximate but useful.

Note that with the CMFD method (Equation 4.24), the net currents computed by the diffusion solver will exactly match those computed by a converged MC simulation (to within stochastic precision) regardless of how D is computed. It is nevertheless useful to compute D values with some accuracy as they make the diffusion-based CMFD solver better able to reduce errors in unconverged solutions.

In addition to the parameters listed in Table 4.4, $\widehat{D}$ correction factors must be computed for every mesh surface. These are computed by solving Equation 4.24 or Equation 4.26 for $\widehat{D}$ using the MC tallied values for the flux in each mesh cell and the net currents between them.

4.5.9 Multigroup cross section derivatives

The two feedback mechanisms considered here are through changes in the fuel temperature and the coolant density. Consequently, derivatives of the multigroup coefficients with respect to these quantities are computed. Typical multigroup cross section derivatives (using the example of thermal absorption and fuel temperature) are computed as,

$$\frac{\partial \Sigma_{a,2,i}}{\partial T} = \frac{\left\langle \frac{\partial \Sigma_a \psi}{\partial T} \right\rangle_{2,i} - \Sigma_{a2} \left\langle \frac{\partial \psi}{\partial T} \right\rangle_{2,i}}{\langle \psi \rangle_{2,i}} \tag{4.34}$$

For the effective fast absorption cross section, it is assumed that ν_s is approximately independent of temperature and density. Consequently, its derivative is computed as,

$$\frac{\partial \Sigma_{a1}}{\partial T} = \frac{\left\langle \frac{\partial \Sigma_a \psi}{\partial T} \right\rangle_{1,i} - (\nu_s - 1) \left\langle \frac{\partial \Sigma_s \psi}{\partial T} \right\rangle_{1,i} - \Sigma_{a1} \left\langle \frac{\partial \psi}{\partial T} \right\rangle_{1,i}}{\langle \psi \rangle_{1,i}}$$

Temperature derivatives are not implemented in OpenMC for group-to-group scattering cross sections. This is because the impact of temperature on the continuous-energy scattering kernel is difficult (and likely costly) to compute. Because of this, the temperature derivative of the downscatter cross section, $\frac{\partial \Sigma_d}{\partial T}$, is not computed. Downscatter is mostly dominated by hydrogen in the coolant rather than fuel isotopes so the expected impact of this derivative is small. Density has no impact on the scattering kernel so $\frac{\partial \Sigma_d}{\partial \rho}$ is computed explicitly.

A simplifying approximation is also made for the diffusion coefficient derivatives. Considering D_g as proportional to $1/\Sigma_{t,g}$, its derivative can be approximated as,

$$\frac{\partial D_g}{\partial \theta} \approx -\frac{D_g}{\Sigma_{t,g}} \frac{\partial \Sigma_{t,g}}{\partial \theta} \tag{4.35}$$

This approximation for the D derivatives is used to avoid the complexity of propagating derivatives through the fine-group flux-limited calculation. Due to details of the OpenMC implementation, it is also significantly more efficient to compute the 2-group Σ_t derivative along with the other needed 2-group derivatives rather than introducing derivatives on the fine group structure. (One OpenMC tally object can compute multiple derivatives as long as they use the same filters.)

Simulations were run to evaluate the accuracy of these derivatives. These simulations

model a simplified PWR fuel pin with a small perturbed region. One set of simulations focuses on a perturbation in coolant density. This set was described previously in Section 4.4.2. Another set of simulations focused on the fuel temperature derivative. These simulations are identical to those described in Section 4.4.2 except that the fuel temperature in the perturbed region is changed instead of the coolant density. The base temperature is 600 K and the perturbed temperature is 900 K. The computed 2-group cross section derivatives are shown in Figures 4-10 and 4-11.

Coolant density derivatives for the 2-group cross sections are shown in Figure 4-10. It is helpful to compare the magnitude of these derivatives to the underlying cross section. For this purpose, the figure also includes typical cross section values. (These values are taken from the midpoint of the unperturbed case.)

The magnitude of the derivatives can be better understood by multiplying them by a perturbation that is representative of those seen in reactors. For example, consider a coolant density perturbation of 0.1 g/cm^3. Figure 4-10 indicates that this would change D_2 by about 12%, Σ_{a2} by about 4%, and $\nu\Sigma_{f2}$ by about 2%. Those values indicate that most of the coolant density derivatives are relatively large and will have a significant impact on the system.

Stochastic noise is visible for most of the reference derivatives. Although the noise can hide an underlying signal, it also provides a useful indication of the magnitude of these derivatives. If a derivative cannot be resolved from this set of MC simulations totaling 600 million active neutrons, then it is likely small enough that it is unimportant for the convergence rate of the coupled solver.

Figure 4-10 generally shows agreement between the reference values and those computed using differential tallies. One notable exception is the D_1 derivative. The values computed using differential tallies and the Equation 4.35 approximation overestimates the magnitude of the true derivative. Note that this will limit the accuracy of the surrogate solver and possibly slow convergence, but it will not introduce a bias in the converged result.

Figure 4-11 shows the analogous dataset for the fuel temperature derivative. In contrast to the coolant density derivative, these fuel temperature derivatives are generally small. A signal can be seen for D_2, Σ_{a2}, and Σ_d; but the values are small relative to the noise. The data suggest that a 300 K perturbation would change each of those 3 parameters by less than 0.5%.

The most significant temperature derivative is for Σ_{a1}. Doppler broadening of resonant absorbers will have a large impact on this cross section. The Figure 4-11 data indicates that a 300 K perturbation would change the cross section by about 3%. The differential tallies capture this derivative well.

Derivatives are not computed for $\widehat{D}$, largely because there is no clear method for doing so that is expected to be both efficient and accurate. The values of $\widehat{D}$ seen in the fuel pin perturbation studies are shown in Figure 4-12, and they give some quantitative detail on the error underlying this approximation. Unlike multigroup cross sections, the derivatives of $\widehat{D}$ are large in regions far away from the perturbation—particularly near the problem boundaries. This implies that the perturbed source effect is strong for $\widehat{D}$.

Note that with the method presented here, these derivatives are only used to build an approximate surrogate model. This surrogate reduces the frequency at which the MC solver is used, but it is not used to replace MC entirely. For that reason, the approximations used here may affect the convergence rate or stability of the solver, but they will not introduce an error in the converged result.

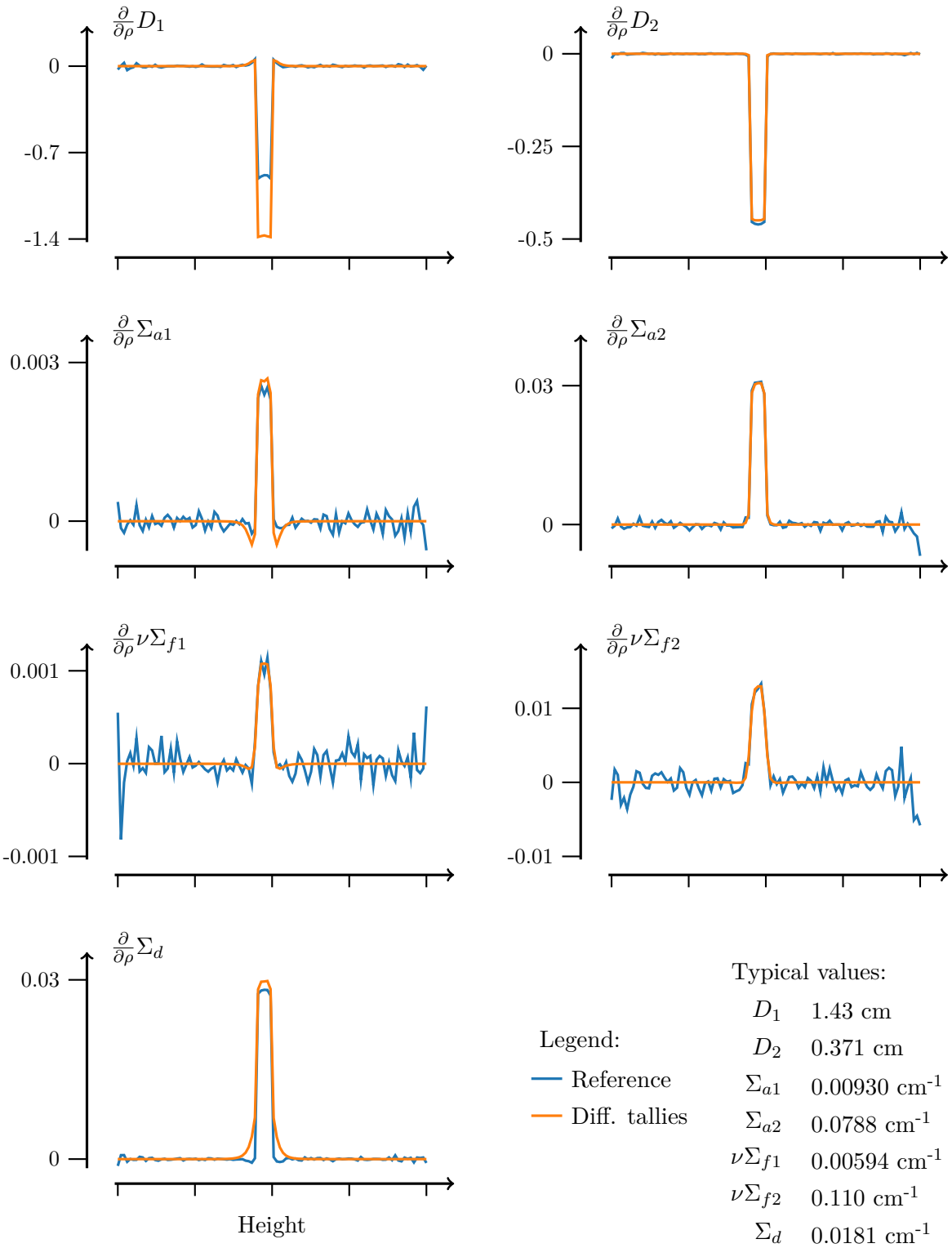


Figure 4-10: Coolant density derivatives for multigroup cross sections compared to a finite-difference reference. Derivative values are in units of [cm^4/g] or [cm^2/g].

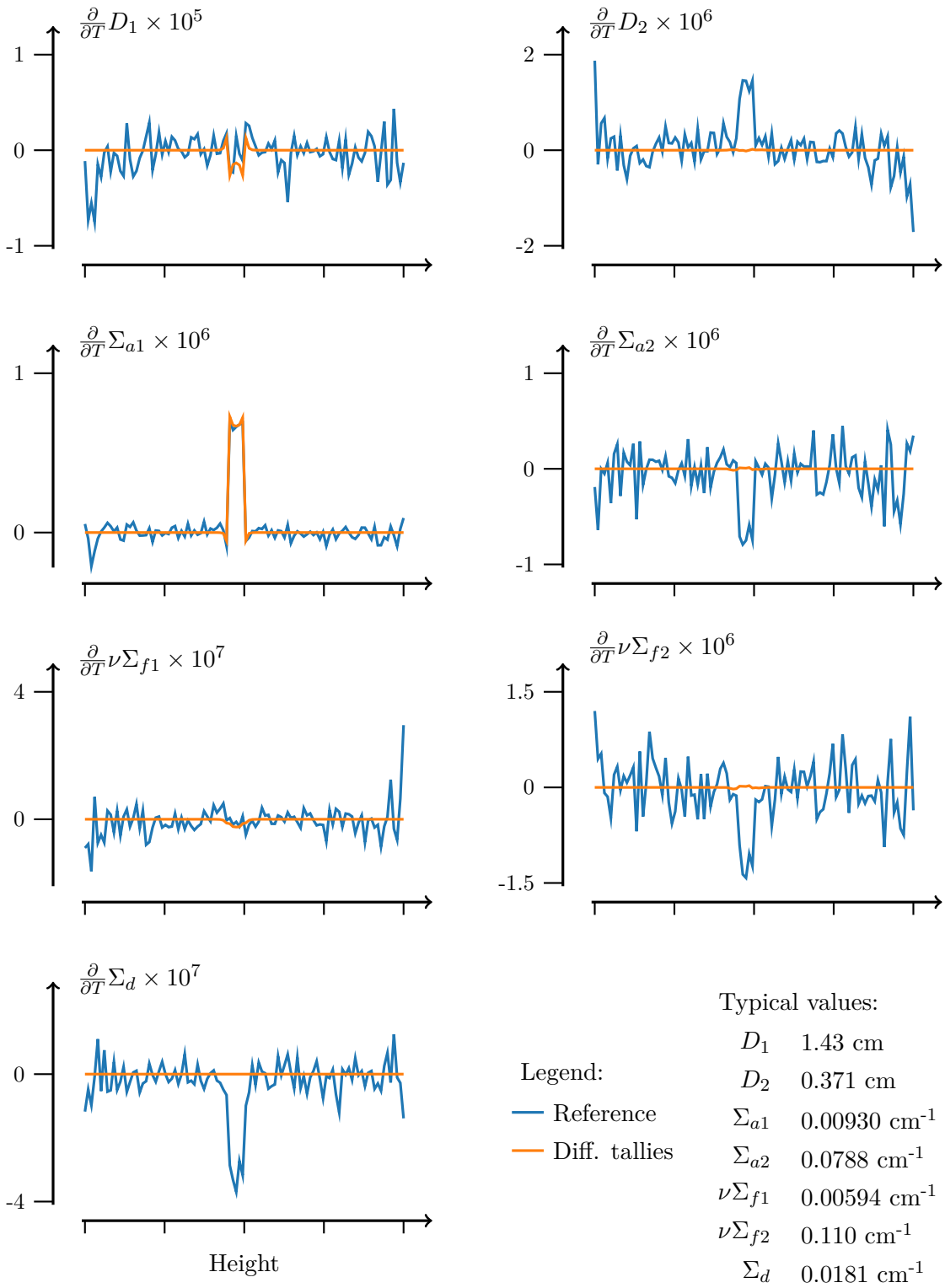


Figure 4-11: Fuel temperature derivatives for multigroup cross sections compared to a finite-difference reference. Derivative values are in units of [cm/K] or [cm^{-1}/K].

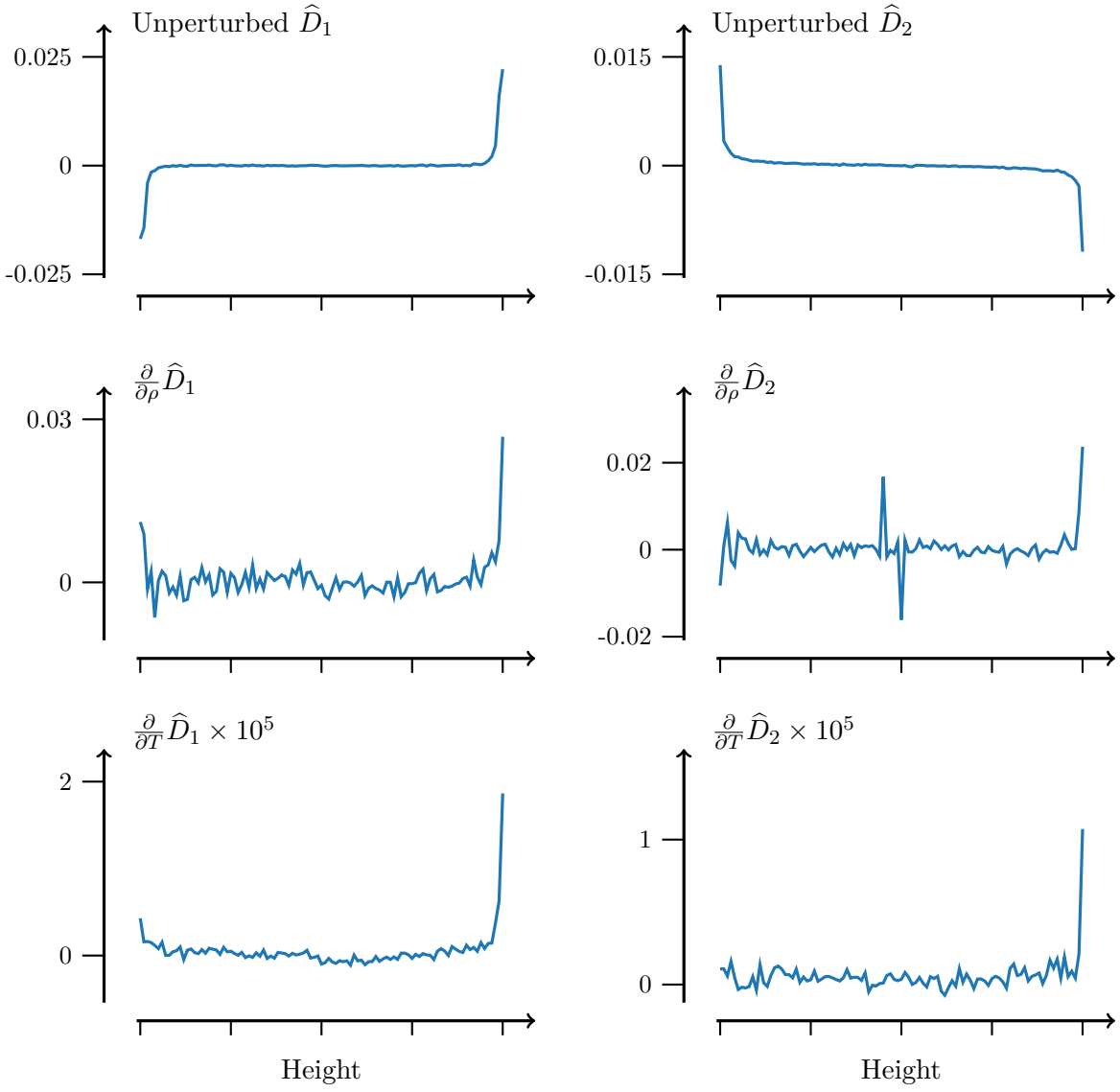


Figure 4-12: Current correction factors, $\widehat{D}$, and their derivatives as computed by finite-difference. The factor $\widehat{D}$ is unitless. Its derivatives are in units of [cm^3/g] or [K^{-1}]. These derivatives are not computed by the solver.

4.5.10 Uniform derivative approximation

The diffusion coefficients and multigroup cross sections used by the diffusion solver are computed using a first-order Taylor series in terms of fuel temperature and coolant density. Using matrix and vector notation, the Taylor series can be written as,

$$\boldsymbol{\Sigma}(\boldsymbol{T}, \boldsymbol{\rho}) \approx \boldsymbol{\Sigma}(\boldsymbol{T}_{\text{ref}}, \boldsymbol{\rho}_{\text{ref}}) + \boldsymbol{J}_T \left(\boldsymbol{T} - \boldsymbol{T}_{\text{ref}}\right) + \boldsymbol{J}_\rho \left(\boldsymbol{\rho} - \boldsymbol{\rho}_{\text{ref}}\right) \tag{4.36}$$

where $\boldsymbol{T}$ and $\boldsymbol{\rho}$ are vectors containing the temperature and density of each region; $\boldsymbol{T}_{\text{ref}}$ and $\boldsymbol{\rho}_{\text{ref}}$ are reference values at which cross sections and derivatives are computed; and $\boldsymbol{J}_T$ and $\boldsymbol{J}_\rho$ are Jacobian matrices containing cross section derivatives with respect to temperature and density.

An example temperature Jacobian with only three regions takes the form,

$$\boldsymbol{J}_T = \begin{pmatrix} \frac{\partial \Sigma_1}{\partial T_1} & \frac{\partial \Sigma_1}{\partial T_2} & \frac{\partial \Sigma_1}{\partial T_3} \\ \frac{\partial \Sigma_2}{\partial T_1} & \frac{\partial \Sigma_2}{\partial T_2} & \frac{\partial \Sigma_2}{\partial T_3} \\ \frac{\partial \Sigma_3}{\partial T_1} & \frac{\partial \Sigma_3}{\partial T_2} & \frac{\partial \Sigma_3}{\partial T_3} \end{pmatrix} \tag{4.37}$$

where subscripts indicate the index of a region.

However, these dense Jacobians are expected to be too expensive to use directly. The number of elements in the Jacobian scales with N^2 where N is the number of regions. Tallying N^2 values may lead to prohibitive runtime and memory costs in the MC solver, and using them may lead to prohibitive costs in the diffusion solver.

Instead a different approach is taken by modifying the process used to compute the differential tallies. Recall that differential tallies include a flux derivative term, discussed in section 4.3.2, that is unique to each particle and is updated every time the particle enters the perturbed region. This accumulated value can be thought of as the derivative of the particle's weight.

In order to compute a differential tally with respect to the temperature in region i, the MC solver must track each particle that enters region i and update that particle's weight derivative each time it moves in region i or undergoes a reaction in region i. Note that this is true even if the tally itself is limited to a different region (i.e. off-diagonal elements of the Equation 4.37 Jacobian). To compute each element in the full Jacobian, the MC solver must keep track of N perturbed weights for each particle—one weight per Jacobian column.

Instead, this work circumvents the issue by treating all of the fuel as a single perturbed region for the purposes of the fuel temperature derivatives. Instead of N weight derivatives, the solver just tracks one weight derivative for temperature. This one weight derivative is updated whenever a particle interacts with a fuel region, regardless of which fuel region it is. Similarly, all of the coolant is treated as one perturbed region in order to compute the coolant density derivatives.

These derivatives are then used to form an approximated diagonal Jacobian. The approximated temperature Jacobian for the three-region example is,

$$\tilde{\boldsymbol{J}}_T = \begin{pmatrix} \frac{\partial \Sigma_1}{\partial T_{\text{uni}}} & & \\ & \frac{\partial \Sigma_2}{\partial T_{\text{uni}}} & \\ & & \frac{\partial \Sigma_3}{\partial T_{\text{uni}}} \end{pmatrix} \tag{4.38}$$

where subscript "uni" is used to indicate that derivatives are taken over the unified fuel region.

The utility of this approximation will be demonstrated in Section 6.6.

4.6 Summary

This chapter has described much of methodology used in this thesis for neutron transport. The guiding principle is to create a coarse mesh diffusion-based solver that bears most of the burden of solving the neutron transport eigenvalue problem while Monte Carlo is used to capture the fine details of the neutron flux in terms of space, energy, and angle.

The use of coarse mesh diffusion and MC has been explored by other researchers, but a novel contribution outlined in this chapter is a method for making diffusion solver responsive to feedback using differential tallies. A series of approximations are used to make this method practical, but it is designed to introduce no bias in a converged solution. The utility of these methods will be demonstrated in Chapters 6 and 7.

Chapter 5

Coupled Solver

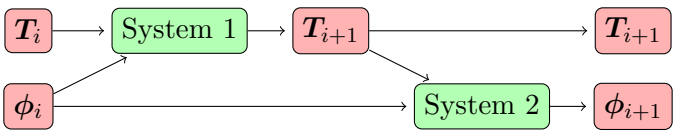
The previous chapters have described four very different physics solvers that must now be unified into one multiphysics tool. This chapter describes the methodology for coupling these individual solvers together.

First, some background is provided on multiphysics iteration schemes and associated terminology used for reactor modeling. Then the geometry model is discussed along with the methods used to map variables from one solver to another. A discussion is also included here on the related topics of computing heat generation and ^{135}Xe buildup. A description of the implemented iteration algorithms follows. The chapter concludes with some brief discussion of the software architecture which is crucial to reducing both the cpu-hours required for the final solver and the student-hours required to implement this project.

5.1 Background on multiphysics iteration methods

5.1.1 Picard iteration

Picard iteration is the most popular multiphysics coupling procedure in reactor physics, likely because it is simple and flexible. It is a divide-and-conquer approach where the problem is separated into distinct systems of equations, and each system is solved one-at-a-time. Consider a problem with the coupled variables of temperature, $\boldsymbol{T}$, and neutron flux, $\boldsymbol{\phi}$. A Picard iteration scheme for this problem can be represented as:



In iteration i, this example solver first assumes a fixed distribution of $\boldsymbol{\phi}_i$ then solves the first system of equations (probably heat transfer equations) for a new temperature distribution, $\boldsymbol{T}_{i+1}$. The solver then uses that new temperature distribution in the second system of equations (neutron transport) to find a new flux distribution, $\boldsymbol{\phi}_{i+1}$. This process is then repeated iteratively, hopefully converging to a fixed solution of $\boldsymbol{T}$ and $\boldsymbol{\phi}$. The trouble with Picard hinges on those words "hopefully converging to a fixed solution", and this trouble will be discussed in Section 5.1.2.

This iteration scheme is referred to as Picard iteration in this work, but other researchers call this scheme—or very similar ones—by the names Gauss-Siedel iteration, Jacobi iteration, or operator splitting.

Note that the System 1 and System 2 from the example can be solved with two completely different computer programs. For example, COBRA can be used to solve for the temperature distribution and OpenMC for for the flux distribution. The output from one code can then be used to update the input files for another code either manually or with the use of a computer script. This is one key advantage of Picard iteration: it easily allows for the use of two programs developed and validated independently by the experts in their respective domains, and the internal workings of both programs can be very different.

5.1.2 Under-relaxation

Picard iteration often suffers from convergence issues. The iteration might not converge at all or it might converge prohibitively slowly. One elegantly simple example of these convergence difficulties for reactor calculations was published by Gill et al. [51]. Their study analyzed a 3D PWR fuel pin and included neutronics, heat transfer, and fluid dynamics. Gill et al. showed that depending on input parameters (coolant flow rate and cladding resistivity), axial instabilities develop and the system oscillates between producing power predominately at the top of the core, then at the bottom of the core, then back to the top, and so on.

Picard instabilities such as these are typically eliminated by using under-relaxation which limits how much solution variables may change between iterations. The relaxation might be applied to any variable or multiple variables. For example, a temperature update

can be relaxed with the formula,

$$\boldsymbol{T}_{i+1} = (1-\alpha)\boldsymbol{T}_i + \alpha \boldsymbol{T}_{\text{solved}}$$

where $\boldsymbol{T}_{i+1}$ is the computed temperature for the next iteration, $\boldsymbol{T}_i$ is the temperature from the previous iteration, $\boldsymbol{T}_{\text{solved}}$ is the temperature computed directly by the solver, and α is the relaxation factor. A value of α below 1 is referred to as under-relaxation; above 1 is over-relaxation.

The results published by Gill et al. [51] show the classic relaxation optimization problem where a large relaxation factor results in a non-converging solution, and an excessively small relaxation factor results in slow convergence. The Gill et al. results and the clever way the authors presented them are enlightening so they are reproduced here.

Simulations using the solver described in this thesis were run on a single PWR fuel pin. For this problem, cross sections were pre-generated with MC and fitted to polynomials of fuel temperature and coolant density. The CMFD diffusion solver was then used (without $\widehat{D}$ correction factors) as the sole neutronics solver for this example problem. (For a similar example with a detailed description, see [51].) The solvers were coupled using Picard iteration. Figure 5.1.2 shows the result of these simulations using different under-relaxation factors. The upper portion of the figure shows the axial power profile computed in each iteration of the solver. The x-axis indicates the iteration number; the y-axis indicates the height in cm; and the color indicates the power density with high-power regions colored red.

The data show that with no relaxation, $\alpha = 1.0$, the solver oscillates between two different solutions. The oscillation is driven by the strong negative feedback. If the power is bottom-peaked in one iteration, then the temperature in the lower part of the core rises, the absorption cross section increases, and the neutronics solver will then predict a top-peaked power distribution in the next iteration.

As the relaxation factor is decreased to 0.5 and then 0.25, the oscillations are damped and the solver converges to one solution. However, the L_2 error metric shown in the lower part of Figure 5.1.2 shows that the error also decays more slowly with $\alpha = 0.25$ versus $\alpha = 0.5$. Hence, the under-relaxation factor is a parameter that must be tuned for optimal performance.

Although inelegant, the relaxation factor is fairly easy to configure in practice. It is the

sort of thing that you tune once for the system of interest, leave as an input parameter for any issues down the line, then forget about it.

5.1.3 Picard coupling with Monte Carlo

Thanks to its simplicity and flexibility, Picard iteration is an extremely popular choice for coupled reactor simulations. It is so popular that it is usually assumed rather than explicitly mentioned in publications. It is also frequently used with Monte Carlo neutronics solvers. Even full-core and quarter-core simulations have been computed with Picard-coupled MC [52, 3].

As demonstrated by these researchers, MC-coupled multiphysics simulations with Picard iteration are practical for occasional use in research work, but they can be extraordinarily expensive. The 2017 quarter-core coupled PWR study by Kelly et al. required 7 days of wall clock time with 1,024 cpu cores (170,000 cpu-hours) [3]. A 7 day wait on such a large cluster makes routine calculations infeasible.

There is also a concern specific to Monte Carlo that stochastic noise might activate feedback-driven oscillations and prevent the multiphysics solver from converging. To address this issue, some researchers make use of the Robbins-Monro algorithm [53]. Dufek and

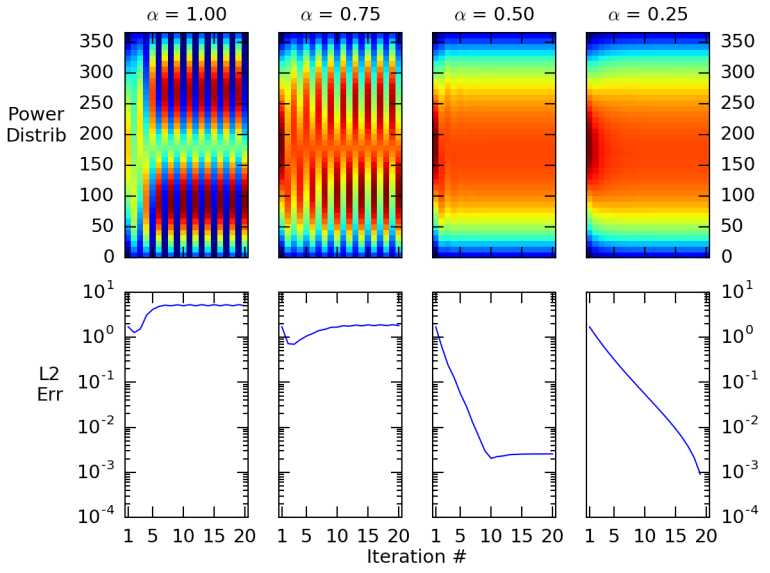


Figure 5-1: The effect of under-relaxation on axial Picard iteration instabilities for a fuel pin problem. The upper plots use a colormap to indicate the axial power distribution in each iteration. The lower plots show the L_2 norm of the error in the power distribution with each iteration.

Gudowski [54] are probably the first to apply this algorithm to multiphysics with MC neutron transport, and it has since been used by others such as Valtavirta et al. [55].

The Robbins-Monro algorithm can essentially be implemented as an under-relaxed Picard iteration where the relaxation factor decreases over the course of the computation. The relaxation factor must follow an infinite series where the values of the series α_n obey the rules,

$$\alpha_n > 0, \qquad \sum_{n=1}^{\infty} \alpha_n = \infty, \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty$$

An example of such a series is $\alpha_n = 1/n$. If the series follows those rules, then there are some guarantees of convergence [53, 54].

The Robbins-Monro algorithm was not found to be helpful for the simulations presented here. A large number of neutrons per Monte Carlo generation were used (the reasoning will be discussed in Chapter 6) which eliminated or significantly reduced the issue of stochastically-generated, feedback-driven oscillations. It was also found that using the Robbins-Monro algorithm complicates the issue of convergence checking. A rapidly decreasing relaxation factor can easily give a false sense of convergence despite the solution being far from its asymptotic value.

5.1.4 Physics-based under-relaxation

Without multiphysics, the accepted approach to solving eigenvalue MC problems is to use a certain number of inactive or discarded generations. During these inactive generations, the solver simulates the movement of neutrons and iterates on the spatial distribution of fission source sites without tallying quantities of interest. This process converges the fission source distribution before tallies are accumulated thus preventing biased results.

When combining a MC code with thermal-hydraulics solvers to make a multiphysics tool, it is tempting to use the same approach and fully converge the fission source in each Picard iteration by using many inactive generations. However, some researchers have shown that in many conditions it is actually preferable *not* to fully converge the fission source in each iteration.

Gill et al. show clearly that using a very small number of MC generations in each iteration leads to better performance [51]. They demonstrate that the solver is both faster to converge and more stable with fewer generations per iteration.

This technique can be referred to as *physics-based under-relaxation*. This terminology is often used to describe the practice of using a transient solver and a short time-step to solve a steady-state coupled problem. The transient physics essentially limits how much the solution may change in each iteration which provides the same benefits as traditional explicit under-relaxation. Limiting the number of generations run by a Monte Carlo eigenvalue solver has an analogous effect since the underlying fission source distribution changes slowly from generation to generation.

5.1.5 Newton-Based Iteration Methods

Newton-based methods like JFNK (Jacobian-Free Newton-Krylov) are a well-known alternative to Picard iteration. The inner workings of JFNK will not be described here because there are many good discussions available elsewhere written by more knowledgable authors (for example, [56]), but some observations on their use will be described in this section.

Because Newton-based methods explicitly consider a problem's Jacobian, they are expected to offer better stability and faster convergence rates relative to Picard iteration. Speedups have been observed for some multiphysics problems such as that discussed by Hales et al. who observed a factor of 2× runtime improvement by using JFNK over Picard for LWR thermo-mechanics simulations [57].

Note that the thermo-mechanic solver described by Hales et al. used the finite element method to discretize all of the underlying physics problems. The coupled problem is then essentially treated as a series of linear algebra problems. For solvers like this, JFNK is relatively straight-forward to apply. The conceptual solver matrix simply includes some rows that discretize the heat transfer problem and other rows that discretize the mechanical deformation problem. This means that the solver considers each of the coupled physics problems essentially simultaneously.

However, the process of developing a Newton-based solver is complicated when different discretization methods are used. Monte Carlo neutron transport is particularly difficult to include in such a solver as it cannot be naturally converted into a linear algebra problem.

Despite the difficulties, Mylonakis et al. have demonstrated a clever implementation of a Newton-based iteration with a MC neutronics solver [58]. This method—called approximate block Newton—allows MC or any other solver to be coupled to others with a Newton iteration scheme.

To describe this method, it is helpful to use the notation of Mylonakis et al. where $\boldsymbol{x}$ is a solution vector describing the power distribution, N is a neutronics solver, and T is a thermal-hydraulics solver. With this notation, Picard iteration without relaxation can be represented in the form,

$$\boldsymbol{x}_{i+1} = N\left(T\left(\boldsymbol{x}_i\right)\right)$$

Newton-Krylov iteration requires that the solver compute (or approximate) the action of the Jacobian on a set of Krylov vectors. Mylonakis et al. approximate the action of the Jacobian with the finite difference form,

$$\boldsymbol{J}\boldsymbol{\nu} \approx \boldsymbol{\nu} - \frac{N(T(\boldsymbol{x}_i + \epsilon\boldsymbol{\nu})) - N(T(\boldsymbol{x}_i))}{\epsilon}$$

where $\boldsymbol{J}$ is the Jacobian, $\boldsymbol{\nu}$ is a Krylov vector, and ϵ is a step-size parameter.

The vectors computed via this approximation then form the Krylov subspace used by the solver. The usual Newton-Krylov iteration is then used to compute a new solution vector. This step can be written in the form of a linear algebra problem as,

$$\boldsymbol{J}\left(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\right) = -\boldsymbol{F}$$

where $\boldsymbol{F}$ is a residual vector.

One downside to this method is that the MC solver must be run multiple times per update of the solution vector; each new vector in the Krylov subspace requires another MC solution. Also note that Mylonakis et al. used the traditional MC approach with many inactive generations (100) and active generations (400).

Since this solver does not operate on the neutronics and thermal-hydraulics problems simultaneously, it breaks the typical mold of JFNK solvers. This may have implications for the loosely-defined terms *tightly-coupled* and *loosely-coupled* that are frequently used to describe multiphysics algorithms.

The method outlined by Mylonakis et al. is promising, but it is not explored here. Instead, it is hoped that the gains found by including a fast deterministic neutronics solver are greater than those which can be found with clever Newton-based iteration algorithms.

5.2 Solver coupling

For this thesis, neutronics, fluid dynamics, and heat transfer are each treated with specialized solvers. These solvers operate on different meshes, have differing numbers of internal loops, and have different runtime costs. Consequently, Picard iteration is used to couple these solvers due to its simplicity and flexibility.

5.2.1 Overview

An overview of the coupling is shown in Figure 5-2. The fine details of the this coupling are described throughout the remainder of the chapter, but an introduction is given here.

The neutronics solvers compute the power distribution (expressed as a linear heat rate, q') which is in turn used by the subchannel and heat transfer solvers. Those solvers in turn provide the coolant density (ρ_{cool}), coolant temperature (T_{cool}), and fuel temperature (T_{fuel}) values needed to solve the neutronics problem.

As indicated by the figure, the Monte Carlo and the surrogate solvers are somewhat interchangeable in this regard. Either can provide a power distribution, and either will respond to changes in fuel and coolant conditions. (Differential tallies are computed for coolant density but not for coolant temperature. The surrogate will therefore respond to changes in ρ_{cool} but not T_{cool}.)

Data is also exchanged between these two neutronics solvers. Homogenized 2-group cross sections (Σ), diffusion coefficients (D), current correction factors ($\widehat{D}$), cross section

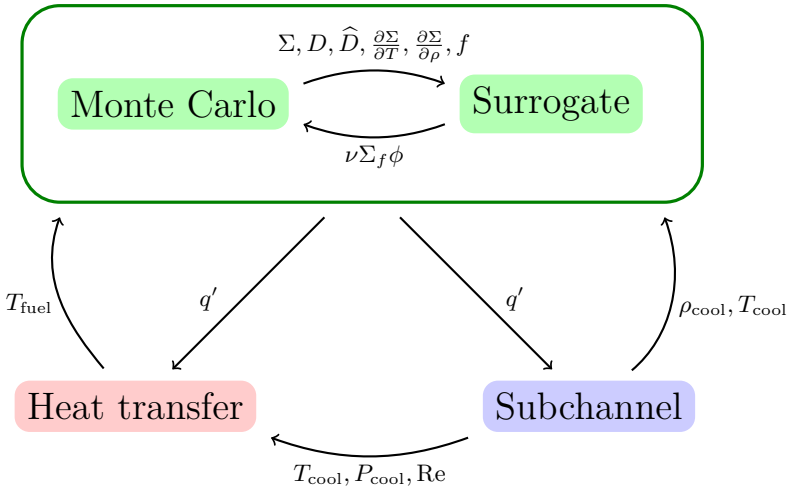


Figure 5-2: Overview of the data exchanges between solvers.

derivatives ($\frac{\partial\Sigma}{\partial T}$ and $\frac{\partial\Sigma}{\partial\rho}$), and pin power factors (f) are computed by Monte Carlo and used in the surrogate. The surrogate can provide a fission neutron source distribution ($\nu\Sigma_f\phi$) to MC.

Finally, the subchannel solver provides the coolant temperature (T_{cool}), coolant pressure (P_{cool}), and Reynolds number (Re) distributions to the heat transfer solver.

For computational efficiency, each of the data transfers shown in Figure 5-2 happens in-memory, without reading or writing files on the disk.

5.2.2 Common computational mesh

Coupling in this work is complicated by the fact that the four physics solvers operate internally on very different computational meshes. Despite the differences, there is a least-common-denominator model of the geometry used throughout the program that aids coupling.

This least-common-denominator geometry model conceptually separates the z-axis from the x and y axes and describes them with different data structures. The geometry in the xy-plane is largely described in terms of pins, channels[1], and the adjacency relationships between them. The z-axis is instead modeled with a simple mesh that that can be used to discretize pins or channels or even homogenized blocks of material.

The decoupling between z and xy appears throughout the program. Many variables are stored in 2D arrays matching these two different dimensions. For example, the linear heat rate is stored as a 2D array indexed by pin number and by axial height. The coolant temperature is a 2D array indexed by channel number and axial height.

Pin and channel adjacency maps

The geometry model for any problem begins with a description of the fuel assemblies and pins within them. From this, the solver assigns a unique index to every pin and a unique index to every channel in the active region of the core. During initialization it also computes maps that describe the adjacency relationships between these. There are pin-to-channel, channel-to-pin, and channel-to-channel maps.

Figure 5-3 shows an example pin-to-channel map for simple 9-pin, 4-channel geometry. Note that different pins can connect to a different number of channels so the mapping is

[1] In this chapter the terms *channel* and *subchannel* are used interchangeably for brevity.

not one-to-one. There are many data structures that could be used to implement the map in software, but a 2D array with some unused elements proves to be a simple and efficient choice.

As a note on implementation, the word *pins* throughout this thesis usually refers to fuel pins, but the program also treats RCCA guide tubes and instrument tubes as *pins*. This makes the channel-to-pin map more useful for tasks such as computing the flow area and wetted perimeter of each channel. These non-fuel pins are simply ignored by the heat transfer solver.

In principle, each guide tube interior could be considered as a channel and modeled by the fluids solver. However, the guide tube interiors are instead neglected by the fluids solver for development expediency. Accurately solving for the flow in these channels would likely require models for the fluid flow through the nozzles, the lower dashpots, and in the annuli around control rods and burnable poisons. These details are considered outside of the scope of this thesis.

Note that the regions outside of the active core—everything beyond the baffle, below the fuel, and above the fuel—are not modeled by the subchannel or heat transfer solvers. Consequently, the solver coupling routines do not transfer any information relating to those regions.

Unified axial mesh

One tool used here to simplify coupling is a unified coarse axial mesh that each solver operates with in some fashion. This axial mesh extends the length of the active fueled

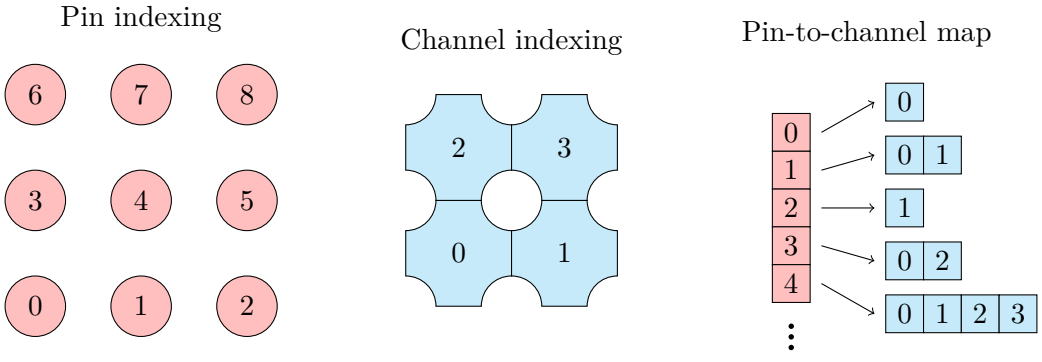


Figure 5-3: Illustration of the indexing schemes for pins and channels, and an example pin-to-channel map.

region. It is irregular and constrained so that there are always mesh boundaries at the top and bottom of each spacer grid. For the calculations presented here, the mesh cells are limited to a maximum size of 10 cm which results in a 42-cell mesh covering the 366 cm of active length.

Temperature and density fields in MC neutronics are resolved on the coarse axial mesh. This is implemented in the OpenMC model by placing z-planes at each of the mesh boundaries and using these planes to make different cells for each axial layer. Similarly, the OpenMC model uses rectilinear tally meshes that match this unified axial mesh in the z-direction.

The heat transfer solver—which essentially solves a large number of uncoupled 1D problems—also operates on the same axial mesh and solves one problem per axial mesh cell and per fuel pin. This provides a convenient conformation between heat transfer and neutronics. No interpolation or mapping is needed along the axial direction for the power distribution or temperature within the fuel.

The subchannel solver requires a finer axial mesh ($\sim$1 cm is used for all calculations here), but this fine mesh is also irregular and constrained so that each cell fits within just one coarse axial mesh cell. It is then assumed that the axial power distribution is flat within each of the coarse mesh cells; the linear heat rate in each fine mesh cell is equal to the linear heat rate of its corresponding coarse cell. Coarse mesh values for the coolant temperature and density are then computed using the average of the fine mesh values, and the coarse values are communicated to the neutronics and heat transfer solvers.

5.2.3 MC geometry and coupling

Of all the solvers, the MC geometry is the most detailed. It is fully 3D and explicitly models many fine structures (e.g. fuel-clad gaps and simplified grid spacers). The shapes and dimensions of geometry features closely match those specified in the BEAVRS benchmark [47], but they are also discretized with the multiphysics temperature and density distributions in mind.

Example plots of the MC geometry are shown in Figure 5-4. Note that the coolant subchannels are treated explicitly with 4 unique materials surrounding each fuel pin. This figure shows the corner between 4 different fuel assemblies and highlights that the MC geometry treats each subchannel with one material even when it crosses an assembly boundary.

OpenMC's *distributed materials* feature is used extensively so that one cell can have a different material for each instance of it appearing in different lattice locations. Note that these plots show the fuel discretized with a 3-region radial mesh, but only 1 region is used for almost all calculations in this thesis.

The OpenMC model uses a unique material for each axial level of each subchannel. It also uses a unique material for each radial region and each axial level of each fuel pin. In the quarter-core BEAVRS model there are 12 828 fuel pins. With the typical discretization of one radial fuel region and 42 axial regions, this leads to 538 776 fuel materials and 591 024 subchannel coolant materials.

With this many materials the default Python library provided with OpenMC to generate the input "materials.xml" file is quite slow. Instead, a simplified XML writer is used here. This XML writer calls the relevant OpenMC Python functions once for a water material and once for each fuel enrichment. The simplified writer then copies the text output from those functions and updates the material "id" values as needed. The usual OpenMC Python functions are then used for the remaining materials such as burnable poisons, steel, and zirconium alloy. Note that XML files are only written and read during solver initialization.

Here the mechanical deformation caused by radiation, temperature changes, and creep are neglected so the geometry is fixed over the course of the simulation.

During multiphysics iterations, the solver adjusts the density and temperature values of

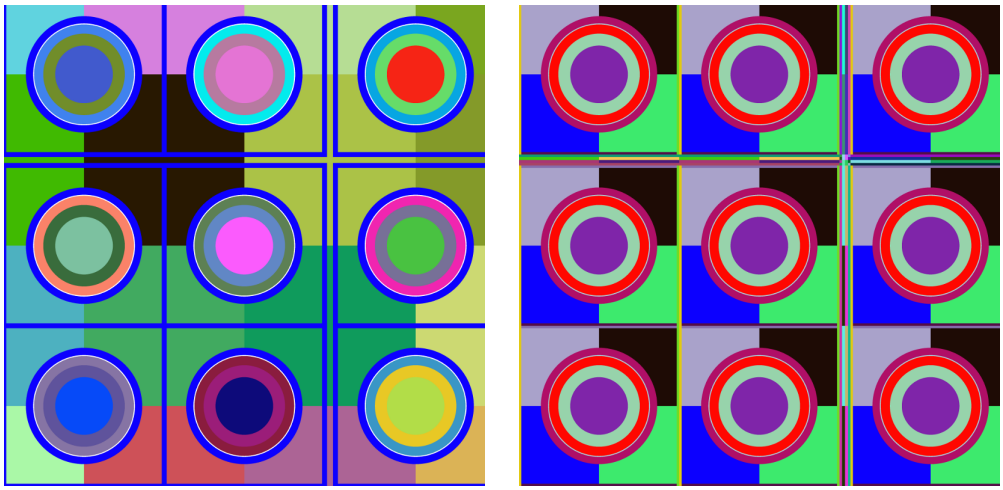


Figure 5-4: MC geometry details near the corner of four fuel assemblies. The left plot shows each material with a unique color; the right plot shows cells. Grid spacers can be seen in these plots.

the coolant materials, the temperature of the fuel materials, and the ^{135}Xe density of the fuel materials. This would be prohibitively slow to do through computer disk access so it is instead achieved directly in RAM. Using the in-memory Python API functions provided with OpenMC is also too slow, so these values are updated using custom implemented in C++ that can be called from Python. The OpenMC header files are included in that C++ code, and the OpenMC shared library is linked during runtime so that the values used by OpenMC can be directly modified during the simulation.

As a further simplification, the MC coolant density is only updated by multiphysics coupling in the actively fueled region. The coolant immediately above the active region is set to the average outlet conditions, and the coolant elsewhere is set to the inlet conditions. Because the coolant in the guide tubes is not modeled by the fluids solver, it is assumed to maintain the inlet conditions.

5.2.4 Coupling from heat transfer to Monte Carlo neutronics

Data models

The fuel temperature distribution is computed by the heat transfer solver and used by the MC solver. Figure 5-5 summarizes the mapping of this data from one solver to the other.

The heat transfer and MC neutronics solvers have very different characteristics which make it best to use a different radial mesh for each solver. It is practical to use a fine radial mesh in the heat transfer solver with a piecewise-linear discretization of the temperature distribution. However, MC codes are usually restricted to a constant temperature within each cell, i.e. a piecewise-constant discretization. (Some researchers have offered alternatives to the piecewise constant approach [37, 59], but they are not widely adopted.) A fine

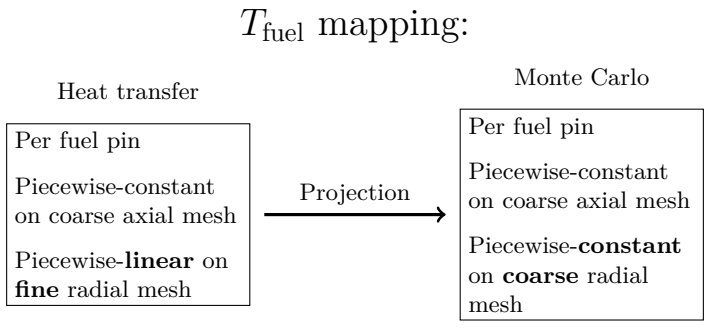


Figure 5-5: Summary of the process of mapping fuel temperature data from the heat transfer model to the MC model.

mesh can also greatly slow down a MC solver by imposing greater ray-tracing and cross section evaluation costs.

Therefore, different meshes are used to match the different solvers. The MC model uses a conventional discretization with mesh boundaries spaced proportional to $\sqrt{r}$ so that all regions have the same area. This has the benefit of more finely resolving the temperature near the edge of the fuel where the gradient is steeper. For simplicity, the heat transfer mesh uses a constant radius interval. An illustrative example of these meshes is shown in Figure 5-6.

When transferring temperature values from heat transfer to MC, they are projected onto the new mesh. The projection is implemented such that the area-weighted average temperature in each cell of the MC mesh matches the heat transfer mesh.

Test calculations on a quarter-core model found that the computed power distribution is insensitive to the number of radial mesh cells used in MC. These calculations are described in Section D.4. For this reason, only 1 mesh cell is used for the results presented everywhere except Section D.4, i.e. the temperature is averaged over the entire interior of the fuel for use in MC. For the calculations shown here, the heat transfer solver uses a mesh with 20 cells in the fuel and 2 in the cladding.

In order to compute the power produced by each pin, the MC solver uses a rectilinear tally mesh with one mesh cell in the xy-plane per fuel pin. This mesh matches the unified coarse mesh in the axial direction. The OpenMC "fission-Q-recoverable" tally is used which computes the recoverable energy (energy excluding neutrinos) released by fission reactions in that mesh cell. There is some nuance to this topic that is discussed further in Section 5.3.

It is approximated that the power distribution is radially flat within the fuel. Note that this is an approximation which is generally worse for depleted fuel due to the accumulation

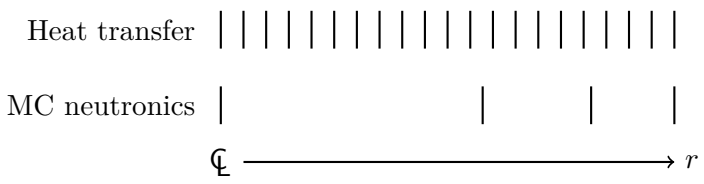


Figure 5-6: Illustration of the mesh spacing for the heat transfer and neutronics meshes. This is a 1D radial meshed used only for the fuel pin interior. The heat transfer mesh is fine with constant-width and the MC mesh is coarse with $\sqrt{r}$ spacing.

of plutonium near the outer edges of the fuel.

One upside to this approximation is that it simplifies the MC tallies and the data transfer software between the neutronics and heat transfer solvers. The only values communicated from neutronics to heat transfer are the linear heat rates in each pin as a function of height.

Software implementation

The heat transfer solver only runs on the root process so an MPI broadcast is used to synchronize fuel temperature values across all processes before updating OpenMC values.[2] The fuel temperature values used by OpenMC are updated directly in RAM without any overhead of disk access and solver finalizing/re-initializing.

Note that an in-memory Python API is provided with OpenMC (the `openmc.lib` module), and it could be used for this purpose. This API was used in the early stages of solver development, but looping over all fuel materials in Python via this API is too slow for efficient quarter-core simulations.

Instead, custom C++ software was written for this task. The OpenMC header files are included in this software, and the OpenMC shared library is linked at runtime. Consequently, the RAM locations used by OpenMC can be directly updated with no overhead. This C++ software also includes Python header files, uses the Python C API, and is compiled as a shared library. Consequently it can be imported into Python at runtime for use. The end result is that the fuel temperature can be represented as an array in Python then passed to C++ software which efficiently loops over each value and copies it to the appropriate memory address for OpenMC.

5.2.5 Coupling from subchannel fluids to Monte Carlo neutronics

The coolant density and temperature distributions are mapped from the subchannel solver to the MC solver as summarized in Figure 5-7. Both solvers use the same radial discretization for coolant variables: each subchannel is one uniform region. No projection or averaging is needed in the xy-plane.

As mentioned previously, the subchannel solver internally uses a finer axial mesh, but each fine mesh cell fits exactly within one coarse mesh cell. The coarse mesh values used

[2] Tricky synchronization errors can arise when updating values in-memory for distributed memory programs like OpenMC. If implemented incorrectly, it's possible for neutrons on one MPI process to see a different temperature than neutrons on another MPI process!

by MC are computed from the arithmetic average of the corresponding fine mesh values. With the meshes used in this work, the coolant properties do not vary greatly within one coarse mesh cell so an arithmetic average will be similar to other schemes.

Similar to the way fuel temperature is handled, custom C++ software was written to update the OpenMC coolant values in-memory from Python arrays. Both the density and temperature of each coolant material is updated by this procedure.

5.2.6 Coupling from heat transfer and subchannel fluids to the neutronics surrogate

The mapping of coolant density and fuel temperature to the surrogate model is summarized in Figure 5-8. Note that these values are only used for extrapolating from the conditions last seen in the MC solver. In a converged solution where the feedback variables are unchanging, the MC solver is ultimately responsible for converting the fuel and coolant conditions into accurate CMFD diffusion parameters. Consequently, simple models for fuel temperature and coolant density in the surrogate will suffice.

The diffusion-based surrogate solver uses the same axial mesh as MC so variables along the axial directions are mapped similarly. (No axial mapping of fuel temperature because the heat transfer solver uses the same mesh; coolant values are averaged over fine mesh cells.)

In the xy-plane this solver uses a very coarse mesh which homogenizes many fuel pins and subchannels. Here the simple approach is taken of computing the arithmetic average of the (area-averaged) fuel temperature in each pin that lies within the coarse mesh. Ideally, an adjoint-flux-weighted average (with the adjoint defined as the importance to each cross section in that mesh) would be used, but this is difficult and unnecessary to implement

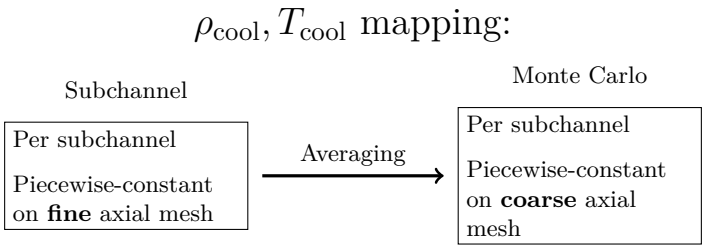


Figure 5-7: Summary of mapping coolant density and temperature from the subchannel model to the MC model.

in practice. Similarly, the arithmetic average of the coolant density is computed for the subchannels that are adjacent to those fuel pins.

5.2.7 Coupling Monte Carlo and surrogate neutronics

The spatial mapping of data from MC to the diffusion-based surrogate solver's mesh is relatively simple: the MC simply uses a Cartesian rectilinear tally mesh that exactly matches the surrogate's mesh.[3] The physics of computing useful parameters on this mesh is more complex, and this topic was discussed at length throughout Chapter 4. Some extra details particularly relevant to multiphysics feedback are described here.

Pin power reconstruction

The surrogate uses a very coarse mesh in the xy-plane that homogenizes many fuel pins, but pin-by-pin resolution of the power distribution is needed for use in the heat transfer and subchannel solvers. A simple scheme is used here to reconstruct pin-by-pin distributions from the surrogate's coarse mesh.

[3]Rectilinear tally meshes were added to OpenMC for this work and merged into the main branch with pull request #1246.

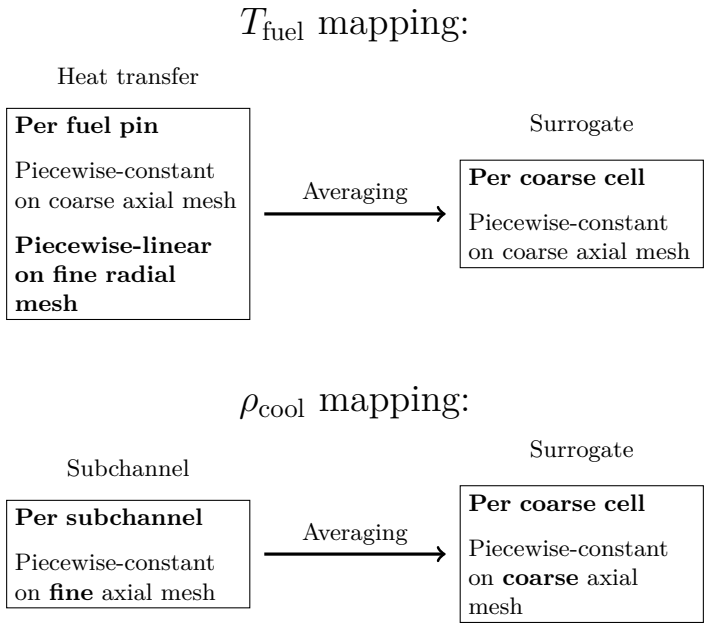


Figure 5-8: Summary of mapping fuel temperature and coolant density values to the neutronics surrogate model.

This reconstruction uses a factor, $f_{i,j}$, unique to each pin and each coarse-mesh axial level. The factor relates the linear heat rate in pin i at axial level j to the neutron production rate in the corresponding coarse mesh cell k,

$$f_{i,j} = \frac{q'_{i,j}}{\left(\nu\Sigma_{f,1}\phi_1 + \nu\Sigma_{f,2}\phi_2\right)_k}$$

The neutron production rate is a convenient denominator because this reaction rate is computed in each eigenvalue iteration of the surrogate. As described in Section 4.5.5, the surrogate mesh is slightly irregular so that no fuel pins straddle the boundaries between mesh cells. This means that each fuel pin can be assigned unambiguously to a single coarse mesh cell. (Some fuel pins are sliced in half by the quarter-core reflective boundary conditions, but they are still only adjacent to one mesh cell so they can be assigned to that one mesh cell for pin power reconstruction.)

The factors, $f_{i,j}$, are computed from MC tallies after each MC iteration (usually just 1 generation). In this sense, they are analogous to multigroup cross sections and are treated in a similar fashion except that they are not modeled with a Taylor series expansion of temperature and/or density.

Source site re-weighting

In order to achieve bi-directional MC-CMFD coupling, the solver must be able to update the MC source distribution using the coarse mesh neutronics solution. In MC, the fission source distribution is represented as a set of discrete points sampled from the transport of neutrons in the previous generation, and the procedure for adjusting the distribution of these discrete points is non-obvious.

Lee et al. adopted the clever approach of adjusting the *weights* of particles emitted from these source sites [8]. With this approach, the source sites themselves are not changed, but the impact each simulated neutron has on the output tallies (and the distribution of source sites for the next generation) is scaled.

A possible implementation of the technique described by Lee et al. is shown in Algorithm 6. This algorithm forms the basis for a more advanced algorithm that will be described shortly.

The goal of this algorithm is to make the MC source weight distribution match the

$Q_{\text{CMFD},i}$ values, the fission source intensity in each coarse mesh cell, i, as computed from the converged CMFD solution. This is achieved by multiplying the weight of each source site by a factor, m_i, that is determined from the ratio, $m_i = Q_{\text{CMFD},i}/Q_{\text{MC},i}$.

Note the final section of this algorithm which renormalizes the particle weights so that they sum to N_{part}, the number of particles per MC generation. A more elegant algorithm could modify the m_i values directly instead of renormalizing the particle weights, but the presented algorithm is preferred for its robustness. The intent of the re-weighting procedure is clear, and is unaffected by experimental modifications to the way m_i values are computed.

One detail not shown in Algorithm 6 is that MPI communication procedures are needed. In OpenMC, the array of source sites is a distributed array—each MPI process has a different set of source sites. For the re-weighting procedure, each MPI process computes its own $Q_{\text{MC},i}$ values and these are summed with a call to the `MPI_Allreduce` routine. (This is advantageous because it parallelizes the significant work of computing which coarse mesh cell each source site resides in.) The coarse mesh neutronics equations are only solved on the root MPI process so the $Q_{\text{CMFD,i}}$ values are sent to the other processes via `MPI_Bcast`.

Algorithm 6 A simple MC source site re-weighting procedure (not used here). This algorithm sets the MC particle weights so that the neutron source distribution matches the distribution computed by CMFD.

▷ Compute MC/CMFD source scaling factors
Declare $\boldsymbol{m}$ an array of scaling factors for each coarse mesh cell
for each coarse mesh cell, i **do**
    $Q_{\text{MC},i}$ = number of MC source sites in cell i
    $Q_{\text{CMFD},i} = \nu\Sigma_{f,1,i}\phi_{1,i} + \nu\Sigma_{f,2,i}\phi_{2,i}$ from the CMFD solution
    if $Q_{\text{MC},i} > 0$ **then**
        $m_i \leftarrow Q_{\text{CMFD},i}/Q_{\text{MC},i}$
    else
        $m_i \leftarrow 0$
▷ Update the source site weights
for each MC source site, j **do**
    $i \leftarrow$ coarse mesh cell containing site j
    w_j = weight of site j
    $w_j \leftarrow w_j \cdot m_i$
▷ Ensure the particle weights sum to the number of MC particles
$W = 0$
for each MC source site, j **do**
    $W \leftarrow W + w_j$
for each MC source site, j **do**
    $w_j \leftarrow w_j \cdot \frac{N_{\text{part}}}{W}$

Each process then computes the m_i values and adjusts the weights of its assigned source sites. Each process then computes its own portion of W and contributes to the total via `MPI_Allreduce` before renormalizing source weights.

The issue with this procedure is that it is a non-relaxed Picard iteration and thus susceptible to instabilities. Test calculations using the solver developed for this thesis find that the MC-CMFD coupling can be unstable even without multiphysics feedback. Hence some form of underrelaxation is needed.

Note that in prior publications, several MC generations are used before coupling to CMFD. For example, Lee et al. delayed MC-CMFD coupling until after the 10^{th} generation [8], and Herman delayed MC-CMFD coupling until after the 5^{th} generation [10]. These authors cite increased tally precision as the reason for the delay, but it also presumably reduces the Picard instability.

Here an alternate approach is taken of applying explicit relaxation to the m_i values. In particular, extreme values of m_i are clipped so that they lie over a smaller range. This procedure is shown in Algorithm 7.

No detailed parameter study for $m_{\max}$ was carried out in this work so no firm recommendations are provided here for setting $m_{\max}$. A value of 1.2 was used for this work which was found to be sufficiently small to prevent instabilities and large enough to allow sufficiently fast convergence. In the quarter-core simulations discussed in Chapter 7, m_i values can be as large as 4 and as small as 0.2 in the first MC generation so the clipping to 1.2 and 1/1.2 is significant. In a converged simulation, the m_i values usually fall within the 1.2 to 1/1.2 range meaning that no relaxation occurs. (Note that this is a unique and probably desirable feature of the nonlinearity of the clipping procedure; linear relaxation techniques would still apply relaxation even when the solution is converged.)

5.2.8 Coupling from neutronics to subchannel fluids

Both neutronics solvers—MC and the surrogate—output q', the linear heat rate distribution, for use in the other solvers. The neutronics solvers provide q' values for each pin discretized on the common coarse axial mesh.

The MC solver computes these values using a tally on a rectilinear mesh with a single fuel pin per mesh cell in the xy-plane. This mesh is separate from the mesh used for surrogate parameters. Note that some fuel pins and instrument tubes are cut in half by the quarter-

Algorithm 7 A MC source site re-weighting procedure with clipping.

▷ Compute MC/CMFD source scaling factors
Declare $\boldsymbol{m}$ an array of scaling factors for each coarse mesh cell
for each coarse mesh cell, i **do**
    $Q_{\mathrm{MC},i}$ = number of MC source sites in cell i
    $Q_{\mathrm{CMFD},i} = \nu\Sigma_{f,1,i}\phi_{1,i} + \nu\Sigma_{f,2,i}\phi_{2,i}$ from the CMFD solution
    if $Q_{\mathrm{MC},i} > 0$ **then**
        $m_i \leftarrow Q_{\mathrm{CMFD},i}/Q_{\mathrm{MC},i}$
    else
        $m_i \leftarrow 0$
▷ Renormalize the scaling factors so the non-zero values average to unity
$\overline{m} \leftarrow 0$
$N \leftarrow 0$
for each coarse mesh cell, i **do**
    if $m_i > 0$ **then**
        $\overline{m} \leftarrow \overline{m} + 1$
        $N \leftarrow N + 1$
$\overline{m} \leftarrow \overline{m}/N$
for each coarse mesh cell, i **do**
    $m_i \leftarrow m_i/\overline{m}$
▷ Clip the scaling factors
for each coarse mesh cell, i **do**
    if $m_i > m_{\max}$ **then**
        $m_i \leftarrow m_{\max}$
    else if $m_i < 1/m_{\max}$ **then**
        $m_i \leftarrow 1/m_{\max}$
▷ Update the source site weights
for each MC source site, j **do**
    $i \leftarrow$ coarse mesh cell containing site j
    w_j = weight of site j
    $w_j \leftarrow w_j \cdot m_i$
▷ Ensure the particle weights sum to the number of MC particles
$W = 0$
for each MC source site, j **do**
    $W \leftarrow W + w_j$
for each MC source site, j **do**
    $w_j \leftarrow w_j \cdot \frac{N_{\mathrm{part}}}{W}$

core reflective boundary conditions. The solver automatically doubles the tallied reaction rates (both the power production rates and the simulated fission detector responses) for these locations to account for the fact that only half of these pins/tubes lie within the model. The surrogate solver can then reconstruct these pin-by-pin q' values using the $f_{i,j}$ factors (which include the boundary condition adjustment).

The subchannel solver has no conventional "mesh" in the xy-plane, but instead uses the channel-to-pin and channel-to-channel adjacency maps. The channel-to-pin map is used to find how much heat is flowing into each channel from the surrounding pins, and the channel-to-channel map is then used to compute the turbulent exchange of enthalpy between channels.

The subchannel solver assumes that the heat flux in each fuel pin is uniform in the azimuthal direction. This means that 1/4 of the power produced by a pin flows into each of the pin's adjacent subchannels.

As a consequence of this approximation, the subchannel solver can directly use the power distribution computed by the neutronics solvers. Otherwise, the heat transfer solver would be needed to determine the azimuthal distribution of the heat flux and determine what fraction of the heat flows into each subchannel. (A transient solver would similarly need to solve the heat transfer problem in order to compute the pin heat flux.)

As mentioned previously, the subchannel solver internally uses a fine axial mesh. The input power distribution comes from the neutronics solvers in the form of an array expressing the linear heat rate of each pin on the coarse unified mesh. The subchannel solver then transfers values from the coarse mesh to each of the appropriate fine mesh cells. Note that this makes the linear heat rate a convenient quantity to use in coupling; values can be used interchangeably between coarse and fine meshes without any rescaling. A diagram of this mapping is provided in Figure 5-9.

5.2.9 Coupling from neutronics to heat transfer

The heat transfer solver essentially operates on a separate 1D problem per fuel pin and per axial level on the common mesh. The neutronics solvers provide exactly one q' value for each of those positions so the q' data can be used directly in the heat transfer solver with no mapping.

Internally, the heat transfer solver makes the approximation that the power distribution

is radially uniform. The distribution is roughly uniform for beginning-of-cycle fuel pins, but in depleted fuel it becomes strongly peaked towards the edge of the fuel due to the rim-effect buildup of Pu-239. Gadolinia-bearing fuel pins may also cause a similar effect due to their strong absorption cross section for thermal neutrons.

5.2.10 Coupling from subchannel fluids to heat transfer

The heat transfer solver must use various fluid properties in order to compute the boundary conditions on the each fuel pin. The mapping of these properties is summarized in Figure 5-10.

Each fuel pin is surrounded by 4 subchannels with different coolant temperatures. This poses a problem for the 1D heat transfer solver which has no means of accounting for this azimuthal variation. Here the simple approach is taken of computing the arithmetic average of the adjacent coolant temperatures and using this average for the heat transfer boundary condition. The coolant properties generally change very little from just one subchannel to the next so the solver is not expected to be sensitive to this averaging. Similarly, the average value in each coarse axial mesh cell is computed from the subchannel solver's fine mesh.

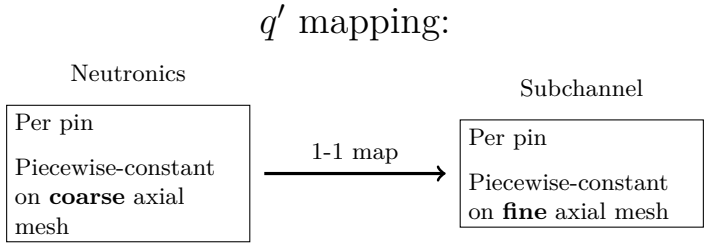


Figure 5-9: Summary of mapping linear heat rate values from the neutronics model to the subchannel model.

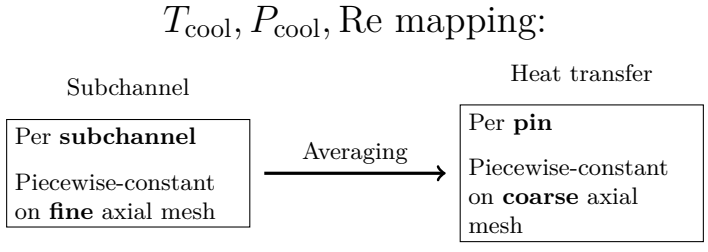


Figure 5-10: Summary of mapping fluid properties from the subchannel model to the heat transfer model.

5.3 Energy deposition and equilibrium xenon

5.3.1 Local energy deposition approximation

The heat energy generated by fission is computed using OpenMC's "fission-q-recoverable" score—a tally scoring function that was added to OpenMC in the course of this work.[4] This score computes the energy released from fission events using the Madland model which gives the energy release from each isotope as a function of incident neutron energy [60, 61].[5] This score includes the energy released promptly from fission in the form of fission fragments, neutrons, and photons. It also includes the delayed energy released from the radioactive decay of fission products in the form of neutrons, photons, and β particles. This score does not include the energy released in the form of anti-neutrinos as it is unrecoverable (practically all anti-neutrinos leak out of the reactor).

The neutrons and photons travel far from their original fission sites and will deposit their energy throughout the reactor materials. This effect can be computed with MC solvers, but it is neglected here for expediency. The energy is instead assumed to be deposited locally, at the fission sites. As a result, the power density in the interior of the fuel is over-estimated which will lead to an over-estimate of fuel temperature. To understand the impact of this approximation, note that the neutrons and photons carry about 10% of the recoverable energy.

5.3.2 Negligible neutron capture energy approximation

Energy is also produced in the reactor from capture reactions such as (n, γ) and (n, α). Accounting for this effect is a surprisingly complex topic.

A common approach of accounting for this energy is to artificially increase the fission Q-value by $(\nu - 1)\,\overline{Q}_c$ where $\overline{Q}_c$ is the average energy produced from neutron capture reactions [62]. This model is valid for a critical reactor where precisely $\nu - 1$ neutrons from each fission event are lost to a capture reaction. Rhodes et al. critique some dubious models for $\overline{Q}_c$ that have been widely used and instead advocate for computing this value directly with high-fidelity neutronics codes that can resolve the per-isotope capture rates [62].

The nuclear data files used by OpenMC include the direct Q-values for all the neutron

[4] OpenMC pull request #698.

[5] This data is listed under MT=1, MF=458 in nuclear data files with the ENDF format.

capture reactions of interest, and tallies could easily be used to compute the reaction-rate-weighted average over all isotopes. However, these Q-values do not consider the β-decays that often follow a capture reaction, and they are thus under-estimates of the value needed for this thesis. The issue is further complicated by the fact that capture products might undergo β-decay or they might instead capture a further neutron.

The total energy produced by these capture reactions is very roughly about 5% of the energy produced by fission reactions. Given the complexities of accounting for it accurately, the capture energy is neglected for this work. It is instead assumed that all energy comes from fission reactions (and the β-decays of their products). As a result, the total fission rate in the core will be over-estimated which leads to an over-estimate of ^{135}Xe concentration.

5.3.3 Equilibrium xenon

A general depletion calculation is outside the scope of this work, but the isotope ^{135}Xe is considered given that it builds up quickly in a reactor and has a large impact on the reactor power distribution.

^{135}Xe is produced directly by fission and through β-decay of other fission products. It is removed from a reactor either through β-decay or through neutron absorption. A reactor operating at stable power can quickly reach an equilibrium concentration where these source and loss rates are in balance. The concentration in this condition can be described by the equation,

$$\sum_j \gamma^j \Sigma_f^j(\boldsymbol{r})\phi(\boldsymbol{r}) = \sigma_{a,\mathrm{Xe}}(\boldsymbol{r})N_{\mathrm{Xe,eq}}(\boldsymbol{r})\phi(\boldsymbol{r}) + \lambda N_{\mathrm{Xe,eq}}(\boldsymbol{r})$$

where $N_{\mathrm{Xe,eq}}(\boldsymbol{r})$ is the equilibrium atomic density of ^{135}Xe at location $\boldsymbol{r}$; γ^j is the cumulative yield of ^{135}Xe from fissions of isotope j; Σ_f^j is the macroscopic fission cross section for isotope j; $\sigma_{a,\mathrm{Xe}}$ is the microscopic absorption cross section for ^{135}Xe; and λ is the β-decay rate of ^{135}Xe.

This equation can then be integrated over volume V_i. This volume will represent one axial segment of the fuel in a single fuel rod. For simplicity, the xenon density will modeled as uniform over this volume. The resulting integrated equation is,

$$\sum_j \gamma^j \iiint_{V_i} \Sigma_f^j(\boldsymbol{r})\phi(\boldsymbol{r})\mathrm{d}\boldsymbol{r} = N_{\mathrm{Xe,eq},i} \iiint_{V_i} \sigma_{a,\mathrm{Xe}}(\boldsymbol{r})\phi(\boldsymbol{r})\mathrm{d}\boldsymbol{r} + \lambda N_{\mathrm{Xe,eq},i} V_i$$

The integrals can be evaluated using MC tallies with the caveat that the MC-computed values must be scaled to match the appropriate intensity of the fission source distribution. Using the angle-bracket notation for MC tallies, the xenon equation can be written as,

$$C \sum_j \gamma^j \left\langle \Sigma_f^j \phi \right\rangle_i = C N_{\mathrm{Xe,eq},i} \left\langle \sigma_{a,\mathrm{Xe}} \phi \right\rangle_i + \lambda N_{\mathrm{Xe,eq},i} V_i$$

where the C scaling factor is computed as,

$$C = \frac{\text{Power}}{\sum_i \left\langle Q_{\mathrm{recov}} \Sigma_f \phi \right\rangle_i}$$

where the numerator is the total specified power for the problem, and the denominator uses the OpenMC "fission-q-recoverable" score. Note the discussion from the previous subsection that a more rigorous implementation requires consideration of the power produced by neutron capture reactions.

Solving for the equilibrium concentration gives,

$$N_{\mathrm{Xe,eq},i} = \frac{C \sum_j \gamma^j \left\langle \Sigma_f^j \phi \right\rangle_i}{C \left\langle \sigma_{a,\mathrm{Xe}} \phi \right\rangle_i + \lambda V_i}$$

This work focuses on problems that do not use depleted fuel so it is assumed that ^{235}U and ^{238}U are the dominant fissionable isotopes. Consequently, just these two isotopes are considered for the xenon production rate. As a further detail, OpenMC does not provide tallies for microscopic cross sections; instead, the macroscopic value is tallied and the result must be divided by the xenon density used in the MC simulation. Incorporating these details gives the final form used for computing the equilibrium concentration,

$$N_{\mathrm{Xe,eq},i} = \frac{C\gamma^{\mathrm{U235}} \left\langle \Sigma_f^{\mathrm{U235}} \phi \right\rangle_i + C\gamma^{\mathrm{U238}} \left\langle \Sigma_f^{\mathrm{U238}} \phi \right\rangle_i}{C \frac{1}{N_{\mathrm{Xe}}} \left\langle \Sigma_{a,\mathrm{Xe}} \phi \right\rangle_i + \lambda V_i} \tag{5.1}$$

The γ values from the ENDF/B-VII.1 nuclear data library are used here [63]. Specifically, the library values for cumulative fission yield from 0.0253 eV neutrons are used.

5.4 Multiphysics iteration

The focus of this thesis is on multiphysics iteration that combines the advantages of Monte Carlo and diffusion neutronics solvers. This section will discuss the details of software algorithms that realize this multiphysics iteration.

5.4.1 Unaccelerated

For a reference point it is helpful to begin with an algorithm that does not include a diffusion neutronics solver. Algorithm 8 shows the iteration procedure used here for simulations without diffusion. This is referred to as the *unaccelerated* iteration procedure. This algorithm references several variables that are represented in software as multidimensional arrays describing a 3D field: T_{fuel} is the fuel temperature, ρ_{cool} is the density of the coolant, N_{Xe} is the atomic density of ^{135}Xe in the fuel, q'_{MC} is the linear heat rate computed directly from Monte Carlo tallies, and q'_{relax} is the linear heat rate computed with under-relaxation. There are also other less important variables exchanged between the solvers (e.g. the coolant Reynolds number from the subchannel solver to the heat transfer solver) that are omitted here for brevity.

As implemented here, the MC tallies are "reset" at the beginning of each iteration. They do not accumulate any information from the previous iterations which used different temperature and density distributions.

Algorithm 8 refers to running N_{gen} MC generations in each of the main iterations. Prior works have frequently used a large number for N_{gen} (perhaps 100) and possibly included "inactive" generations that do not contribute to tallies. The results presented later demonstrate that small values of N_{gen}—even as small as 1—generally perform better. Similarly, no inactive generations are used here since the tallies are reset in each iteration which removes

Algorithm 8 Unaccelerated coupled iteration procedure

1: **for** Main iteration **do**
2: Update MC $T_{\text{fuel}}, \rho_{\text{cool}}, N_{\text{Xe}}$
3: Simulate N_{gen} MC generations
4: Compute q'_{MC} and $N_{\text{Xe,eq}}$
5: $q'_{\text{relax}} \leftarrow (1-\alpha)q'_{\text{relax}} + \alpha q'_{\text{MC}}$
6: $N_{\text{Xe}} \leftarrow (1-\alpha)N_{\text{Xe}} + \alpha N_{\text{Xe,eq}}$
7: Solve subchannel fluids with q'_{relax} for $T_{\text{cool}}, \rho_{\text{cool}}$
8: Solve heat transfer with $q'_{\text{relax}}, T_{\text{cool}}$ for T_{fuel}

the bias from early non-converged MC generations.

Algorithm 8 includes a relaxation step on the power distribution. After performing relaxation, the power distribution is also renormalized to ensure that the total desired power is produced. As demonstrated later in this thesis, using a small number of MC generations (e.g. 1) per multiphysics iteration provides a form of relaxation which makes the arithmetic under-relaxation unnecessary. Consequently, the presented unaccelerated simulations use a unity relaxation factor, $\alpha = 1$.

Relaxation is also applied to the xenon concentration. The same relaxation factor is used for both N_{Xe} and q'_{relax} to avoid the complexity of multi-parameter optimization.

The coupled solver is initialized with a uniform distribution of fuel temperature and coolant properties. The fuel is set to 600 K and the inlet conditions are used for the coolant. The initial MC source is axially cosine and radially flat. The initial ^{135}Xe is set to a value which is practically zero for the purposes of neutron transport, but an arbitrarily small non-zero value is needed so that OpenMC will be able to tally reaction rates.[6]

No automated convergence checking is used for the main iteration loop, although it could easily be added. Instead, a fixed number of iterations are run based on resource constraints, and the data is analyzed after the fact to establish convergence.

5.4.2 Accelerated

Algorithm 9 presents the accelerated iteration procedure. It is largely similar to Algorithm 8 but includes an inner loop where the power distribution is computed using the diffusion-based surrogate instead of MC. This power distribution is indicated with the variable, $q'_{\text{diffusion}}$.

Under-relaxation on the $q'_{\text{diffusion}}$ variable is required for stability of the inner loop, but this is not shown for brevity. This relaxation factor can be set to a small value without a large impact on the overall solver runtime since the MC solver is the most expensive.

Algorithm 9 also includes the computation of 2-group cross sections, diffusion coefficients, their derivatives, $\widehat{D}$ correction factors, and pin power factors. Like the power distribution, these are computed using MC tallies that are reset at the beginning of each primary iteration.

Note that the MC fuel temperature and coolant density is always set using an under-

[6]The initial xenon concentration is set so that it makes up 10^{-12} of the fuel nuclides.

relaxed power distribution that includes contributions from both the MC and diffusion solvers. Test calculations showed that relaxation of this quantity was important for solver stability. Surprisingly, oscillations in the power distribution can propagate through the solver to the multigroup cross sections and form an instability. This instability is mitigated by under-relaxation on the power distribution.

5.5 Software design

The bulk of this thesis discusses numerical methods, algorithms, and physics approximations; but the success of this work also hinges on software design. Poorly implemented software can easily destroy performance and turn a viable method into a useless program. Therefore, this section briefly discusses some of the programming practices and software details of the coupled solver.

The multiphysics solver is written in a combination of Python and C++. APIs and dynamic linkage are used extensively so that the solver transitions seamlessly between code written in each language. Python is used because it makes the software development process faster. Python code does not need to be compiled which allows for fast prototyping and debugging, and there is a large ecosystem of helpful third-party Python software that is trivial to acquire and use. C++ on the other hand is usually more difficult to use,

Algorithm 9 Accelerated coupled iteration procedure

for Main iteration **do**
    Update MC $T_{\text{fuel}}, \rho_{\text{cool}}, N_{\text{Xe}}$
    Simulate N MC generations
    Compute q'_{MC} and $N_{\text{Xe,eq}}$
    $q'_{\text{relax}} \leftarrow (1-\alpha)q'_{\text{relax}} + \alpha q'_{\text{MC}}$
    $N_{\text{Xe}} \leftarrow (1-\alpha)N_{\text{Xe}} + \alpha N_{\text{Xe,eq}}$
    Compute 2-group $D, \Sigma, \widehat{D}, \frac{\partial}{\partial T}, \frac{\partial}{\partial \rho}$ values and pin power factors, f
    while $q'_{\text{diffusion}}$ unconverged **do**
        Extrapolate cross sections (Equation 4.36) with $T_{\text{fuel}}, \rho_{\text{cool}}$
        Solve diffusion neutronics for $q'_{\text{diffusion}}$
        Solve subchannel fluids with $q'_{\text{diffusion}}$ for $T_{\text{cool}}, \rho_{\text{cool}}$
        Solve heat transfer with $q'_{\text{diffusion}}, T_{\text{cool}}$ for T_{fuel}
    Reweight MC source sites using the diffusion solution (Algorithm 7)
    $q'_{\text{relax}} \leftarrow (1-\alpha)q'_{\text{relax}} + \alpha q'_{\text{diffusion}}$
    Solve subchannel fluids with q'_{relax} for $T_{\text{cool}}, \rho_{\text{cool}}$
    Solve heat transfer with $q'_{\text{relax}}, T_{\text{cool}}$ for T_{fuel}

but the resulting software is often orders-of-magnitude faster thanks the optimization and compilation process. It is also easier to develop parallel code in C++ using MPI and OpenMP.

The software was developed in an iterative cycle of prototyping, testing, profiling, and optimizing. All novel software (everything except OpenMC) was first written in Python. After implementing new code and testing for accuracy, it would then be instrumented to identify runtime hotspots. Code hotspots were then rewritten in C++ as needed.

This software makes extensive use of the Python C API. C++ software that uses this API can use the native Python data structures such as the `list`, `dict`, and `tuple` so that the C++ components interface well with the pure Python components. This software also uses the C API provided for the numpy Python package which allows for numpy arrays to be used interchangeably between Python and C++.

The Python C API and numpy C API are somewhat challenging to learn and use, but they offer important performance gains. Critically, no copying is needed for one language to access the data allocated in another language. Only memory addresses are exchanged, and the software written in either language can interpret the data at that address.

The C APIs allow the software to switch between code written in either language without any overhead. The solver takes advantage of this and frequently switches between languages. For example, the diffusion neutronics eigenvalue loop is written in Python, but it calls a Gauss-Seidel loop written in C++. The pin-to-channel map is allocated in Python, and the unified axial grid is allocated in C++. The subchannel loop over axial height is written in Python, but it calls a C++ loop over the channels.

5.6 Summary

This chapter has described the linkage between the individual physics solvers. Most of the methodology was chosen with simplicity in mind, but the focus here is on Algorithm 9 which shows how the diffusion-based surrogate is used within the multiphysics iteration procedure. The following chapters are focused on evaluating the performance of this algorithm.

Chapter 6

Fuel Pin Simulations

The focus of this thesis is on quarter-core sized calculations which prove challenging for a multiphysics solver. However, smaller simulations on just a single fuel pin are also useful for characterizing the convergence behavior of the coupled solver.

This chapter covers an array of calculations on a single fuel pin, and discusses the issues of instability, convergence precision, and convergence rate. The data presented here suggest that simulations using a small number of MC generations per multiphysics iteration (e.g. 1) and a large number of particles per generation are more stable and less noisy. The data also show that the proposed acceleration algorithm can reduce the number of generations needed for convergence by more than 30×.

For all simulations shown here, the geometry is of a fuel pin from the BEAVRS benchmark [47]. This model uses the same 42-cell axial mesh that is later used for quarter-core calculations. The pin power is set at 68 kW, the average value for full-power BEAVRS. Fuel temperature and coolant density feedback are included. No depletion or equilibrium xenon calculations are used.

6.1 Convergence analysis

The typical approach for evaluating Monte Carlo source convergence is to use a Shannon entropy metric. Specifically, this metric measures the entropy of the fission source distribution on some specified mesh. Unlike typical tallies, the entropy is computed for each generation rather than accumulated over the course of the simulation. Plotting the entropy as a function of generation number usually reveals a signal with a bias that decays away in

the early generations.

The fission site Shannon entropy is convenient in the general case because it is applicable to a wide variety of systems and condenses all the complexity of the fission source distribution into one number per generation. However, the value of entropy itself is essentially meaningless and it cannot be easily related to the uncertainty or error in the quantities of interest.

For convergence checking, this work can take advantage of the fact that tallies of the power distribution are needed and these tallies must be frequently reset for multiphysics purposes. This means they can be plotted as a function of generation number to reveal convergence trends.

One particularly useful metric of the power distribution is the axial offset. It measures the relative imbalance of power between the top and bottom of the core and is defined as,

$$\mathrm{AO} = \frac{Q_{\mathrm{top}} - Q_{\mathrm{bottom}}}{Q_{\mathrm{top}} + Q_{\mathrm{bottom}}} \tag{6.1}$$

where Q_{bottom} is the total power produced in the lower half of the core and Q_{top} is of the upper half.

Note that in a single fuel pin problem, the axial offset will covary strongly with the second-most-dominant eigenmode. Consequently, it can be expected to converge slowly relative to other metrics which makes it a good measure to evaluate solution stationarity. As will be demonstrated below, this metric also captures the oscillations seen with unstable algorithms.

6.2 Active and inactive generations

Note that this work will also adopt the atypical practice of averaging tallies over multiple generations after-the-fact rather than during the calculation. Usually a MC solver internally accumulates tally values over the course of some number of active generations and then reports the average rather than the generation-by-generation value. This is advantageous in the general case because it requires less memory, and the generation-by-generation values are usually more information than needed.

However, the alternative approach is practical and useful here. This solver stores the power distribution computed in each MC Picard iteration. In the usual case, there is 1

generation per Picard iteration so the generation-by-generation tallied power distribution is stored. The memory required is small enough that this approach is practical, and it provides much more useful data for post-processing.

In particular, this removes the need to determine before-the-fact how many MC generations are needed to converge the source distribution. The data can be analyzed after the simulation to determine the point of stationarity, and then the average value of the stationary region can be computed. This means that the solver does not use any *inactive* generations, at least not as they are usually defined for MC simulations. Even the first MC generation provides valuable data used by the coupled solver.

Storing the values from each Picard iteration also allows for a better understanding of the statistical noise in the solver. Unlike the traditional MC approach, autocorrelation can be seen and measured which protects against an underestimation of statistical uncertainty.

6.3 MC generations and multiphysics instability

One important parameter for MC-based multiphysics solvers is how many particle generations are used per multiphysics iteration. Using multiple generations per iteration can theoretically reduce statistical noise and better converge the fission source distribution within each iteration, but it also reduces the frequency at which the temperature distribution is updated. Gill et al. have previously demonstrated the surprising conclusion that using very few generations per iteration leads to better stability and faster convergence, even when it increases the stochastic noise seen by the thermal-hydraulics solvers [51]. Those conclusions are further reinforced by the data presented here.

Simulations were run with a varying number of MC generations per multiphysics Picard iteration. These simulations were run without acceleration using Algorithm 8. Figure 6-1 shows the axial offset behavior with 1, 20, 50, and 100 generations per iteration. Figure 6-2 shows a set of longer simulations with 200, 500, and 1000 generations per iteration. All simulations use 1 million particles per generation, and no explicit relaxation is applied ($\alpha = 1$). Note that these generation numbers indicate both how frequently a Picard iteration is executed and how frequently tallies are reset. This means that the 20-generation data points are averaged from 20× as many simulated neutrons as the 1-generation data points.

These figures show that the number of generations per iteration acts essentially as a

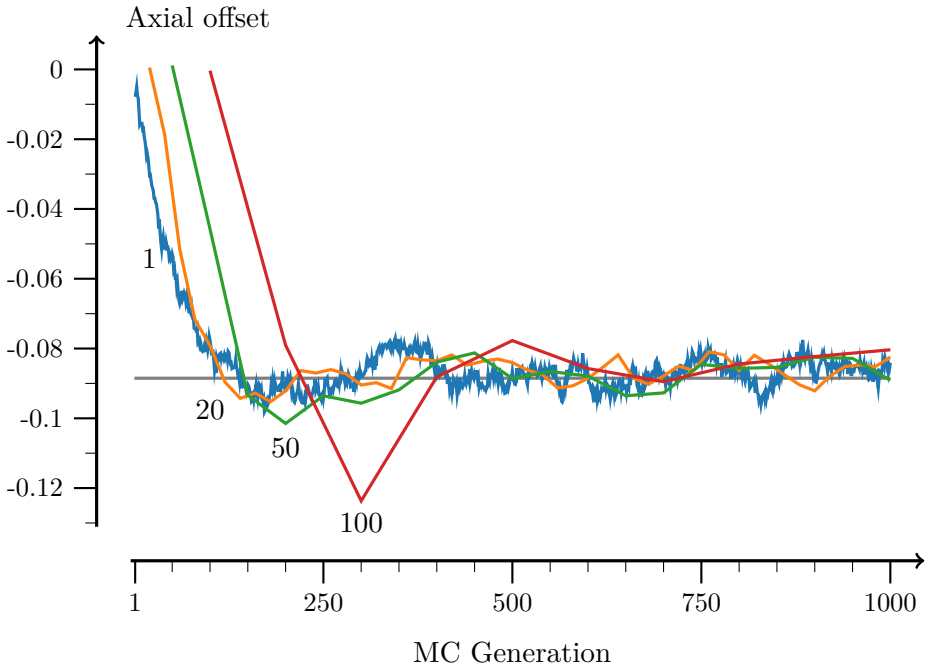


Figure 6-1: Convergence of the axial offset (Equation 6.1) from unaccelerated simulations of a fuel pin with varying numbers of MC generations per Picard iteration (indicated by the number next to each series).

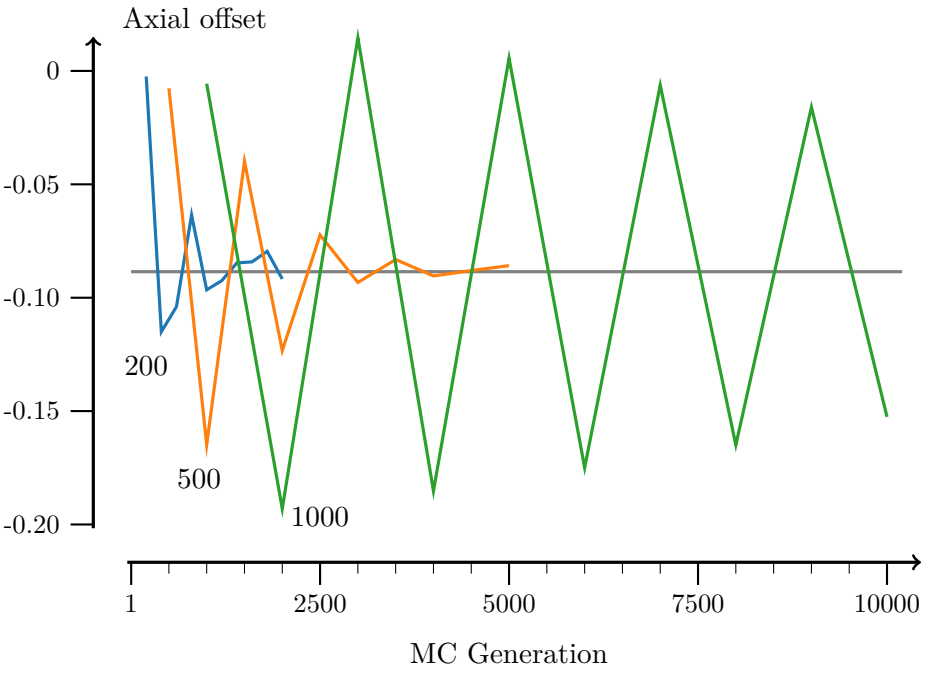


Figure 6-2: Convergence of the axial offset for unaccelerated simulations that use many generations per Picard iteration. Note the change in x- and y-axis scales from Figure 6-1

relaxation parameter. Too many generations per iteration leads to a wildly oscillating solution that is driven by the strong negative feedback in the system. The impact of the number of generations seen here is analogous to the impact of the relaxation factor shown in Figure 5.1.2.

The reason for this relaxation effect is due to the dynamics of the MC fission source. Simulated neutrons do not travel very far from their birth location so the fission source sites that they sample will be relatively near to their own source site. Thus the overall change in the fission source distribution from generation-to-generation is limited.

Using a small number of generations to damp oscillations is somewhat similar to the technique of using a transient solver in a steady-state multiphysics calculation. If the time step parameter for the transient solver is sufficiently small, then the time derivative terms act like relaxation factors because they limit how much a solution can change between iterations. This practice is often referred to as *physics-based under-relaxation* [64], and these results suggest that limiting the number of MC generations per Picard iteration is another form of physics-based under-relaxation.

Note that this under-relaxation effect is limited to simulations without acceleration. The entire purpose of acceleration is to make large changes in the fission source distribution between MC generations so other forms of relaxation are needed.

6.4 MC generations and tally noise

Another finding from the Figure 6-1 data is the surprising fact that that increasing the number of generations does not decrease the size of the noise in the stationary region.

To highlight this point, Figure 6-3 focuses on the computed axial offset for generation 250 and beyond. This figure shows simulation results using 1, 2, 5, 10, and 20 generations per iteration. Note that the subplots use identical scales. Horizontal lines on this figure indicate the $\pm 2\sigma$ envelope where σ is the standard deviation of the 1 generation-per-iteration data.

The data in this figure indicate that the number of MC generations per iteration has a very small effect on the tally noise. Simulations that use 20 generations for each data point are no more precise than simulations that use 1 generation per data point.

This finding is somewhat counter-intuitive. It is generally expected that a larger number of of generations leads to a larger number of tally events which significantly reduces the

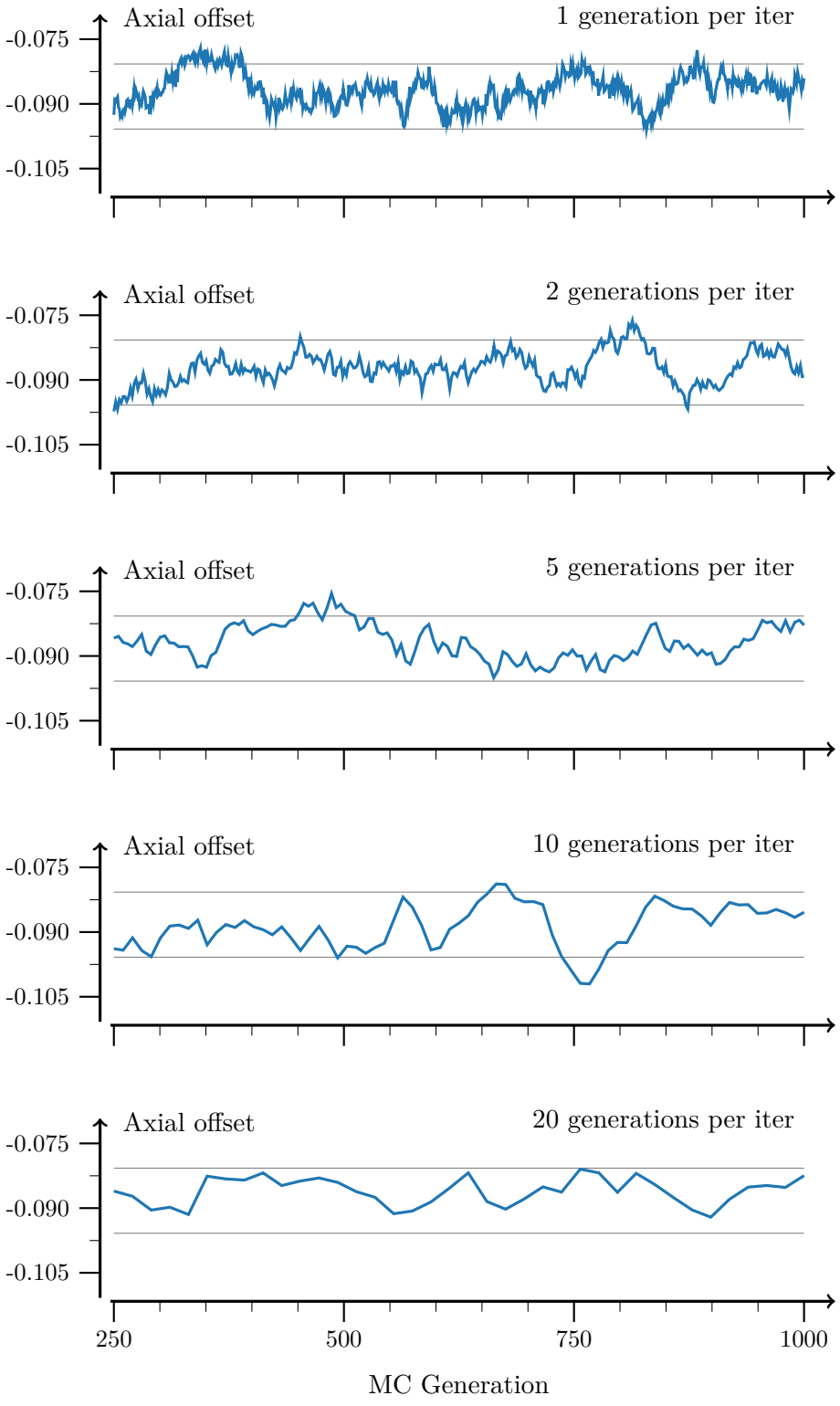


Figure 6-3: Tallied axial offset as a function of MC generations. Subplots show a different number of generations per iteration. Horizontal lines indicate the uniform noise envelope. All simulations use 1 million neutrons per generation.

Figure 6-4: Standard deviation of the axial offset (generations 250 onward) versus the number of generations per iteration. For reference, the last column indicates the expected standard deviation if tally results were independent.

N_{gen}	Std. dev.	$\propto 1/\sqrt{N_{\text{gen}}}$
1	0.0038	—
2	0.0037	0.0027
5	0.0041	0.0017
10	0.0051	0.0012
20	0.0031	0.0008
50	0.0042	0.0005

statistical uncertainty of tallied results.

More rigorously, if tally results from each generation were independent, then the central limit theorem suggests that the tally standard deviation would scale with $1/\sqrt{N_{\text{gen}}}$ where N_{gen} is the number of generations per iteration. Table 6-4 shows the standard deviation of the Figure 6-3 data and compares it to $1/\sqrt{N_{\text{gen}}}$ scaling. The observed standard deviation is nearly constant, and does not follow the $1/\sqrt{N_{\text{gen}}}$ scaling.

The noise does not scale with $1/\sqrt{N_{\text{gen}}}$ because tally values from consecutive generations are not statistically independent. Tallied quantities like the axial offset strongly depend on the MC fission source distribution, and this distribution does not change by a large amount from one generation to the next.

Because consecutive tally values depend significantly on the slowly-changing source distribution, these consecutive values will be statistically correlated. This is referred to as autocorrelation—the signal is correlated with itself—and it can be seen in the Figure 6-3 plots. The $N_{\text{gen}} = 1$ and $N_{\text{gen}} = 2$ cases in particular show that each tallied value is very close to the previous tallied value. As a result, the tally values slowly meander upwards and downwards within the noise envelope. Contrast this behavior with Figure 6-5. This figure shows random noise sampled from a normal distribution, and it demonstrates a signal with little autocorrelation.

This behavior can be quantified in terms of an autocorrelation coefficient which is defined as,

$$R(\tau) = \frac{\text{Cov}(X_t, X_{t+\tau})}{\text{Var}(X_t)} \tag{6.2}$$

where X_t is the set of tally values and $X_{t+\tau}$ is the set of tally values shifted by τ generations.

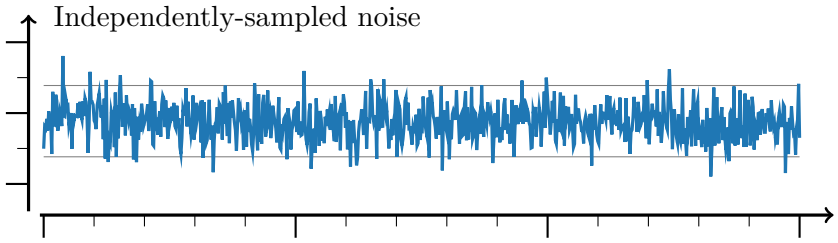


Figure 6-5: Random noise. Values were sampled independently from a normal distribution with a mean and standard deviation that matches the data shown in Figure 6-3 for 1 generation per iteration.

This value can be computed as,

$$R(\tau) = \frac{\frac{1}{n-\tau}\sum_{t=0}^{n-\tau-1}(X_t-\mu)(X_{t+\tau}-\mu)}{\frac{1}{n}\sum_{t=0}^{n-1}(X_t-\mu)^2} \tag{6.3}$$

where μ is the mean value of X, and n is the number of X values. For the present discussion, X is the set of computed axial offset values.

Figure 6-6 shows the autocorrelation coefficient for the axial offset data up to $\tau = 20$. Even with a lag of 20 generations, the coefficient only decays to $R(20) = 0.49$. Note that the autocorrelation coefficient is a specific case of the Pearson correlation coefficient. A Pearson coefficient of 0.49 indicates a relatively strong relationship between two variables. This means that error of a tallied value (the value minus the mean) at generation $t + 20$ is strongly related the error at generation t, and the two samples cannot be considered independent.

Bayley and Hammersley offer a way to compute an *effective* number of independent samples given a known $R(\tau)$ [65]. The uncertainty of N correlated samples is equal to the uncertainty that would be expected with N_{eff} independent samples. For this discussion, the number of samples equals N_{gen}, the number of MC generations per iteration. The effective number of generations is then [65],

$$N_{\text{gen,eff}} = N_{\text{gen}}\frac{1}{\sum_{\tau=0}^{N_{\text{gen}}-1}R(\tau)} \tag{6.4}$$

These values were computed for simulations with N_{gen} up to 50, and the results are shown in Table 6-7. This table also shows the expected standard deviation computed using N_{gen} and $N_{\text{gen, eff}}$. These numbers indicate that even 20 generations can barely be expected

to give a more precise answer than 1—a prediction which matches well with the observed standard deviations. To put it bluntly, a tally over 20 MC generations mostly gives the same number 20 times in a row, and averaging the result does not give a noticeably more precise answer.

One practical implication of these observations is that different generations cannot be considered independent for the purpose of estimating uncertainty on tally mean values. Like most MC codes, OpenMC reports uncertainty figures that make this erroneous assumption. Due to the high autocorrelation, those uncertainty estimates are not used anywhere in this thesis.

Another practical implication is that increasing N_{gen} is a generally inefficient means of reducing tally error. Increasing the number of particles per generation or running multiple independent simulations is a much more efficient choice as will be discussed in Section 6.5.

There is one caveat to mention before leaving this discussion. Increasing the number of particles per generation is more effective than increasing the number of generations, but there are sometimes practical computer limitations on the total number of particles per generation.[1] When these limits are reached, increasing N_{gen} can still provide some benefits to the solver even though the overall sample standard deviation is largely unaffected.

Notice from Figure 6-3 that all results show a similar noise envelop, but increasing N_{gen} does reduce the high-frequency components of the noise. The high-frequency components are smaller than the low-frequency components and thus largely unimportant for the overall standard deviation. However, the high-frequency error does manifest in other ways that can be occasionally troublesome for the solver.

For example, this high-frequency error leads to less smooth spatial distributions of tally results. This effect on spatial smoothness is demonstrated in Figure 6-8. Each plot in this figure shows the q' distribution sampled from just one solver iteration (an arbitrarily chosen iteration in the stationary region). The two plots show simulations with differing N_{gen}, and the greater N_{gen} results in a smoother curve. Test calculations (not shown here) find that values such as $N_{\text{gen}} = 2$ are occasionally helpful for diffusion solver stability which suggests the importance of spatially-smooth tally results.

[1] Because of a limitation in the version of the numpy software used here, some data arrays passed through C APIs are limited to a maximum size of 2 GB. This in turn limits the simulations in this thesis to 24 million particles per generation per MPI process.

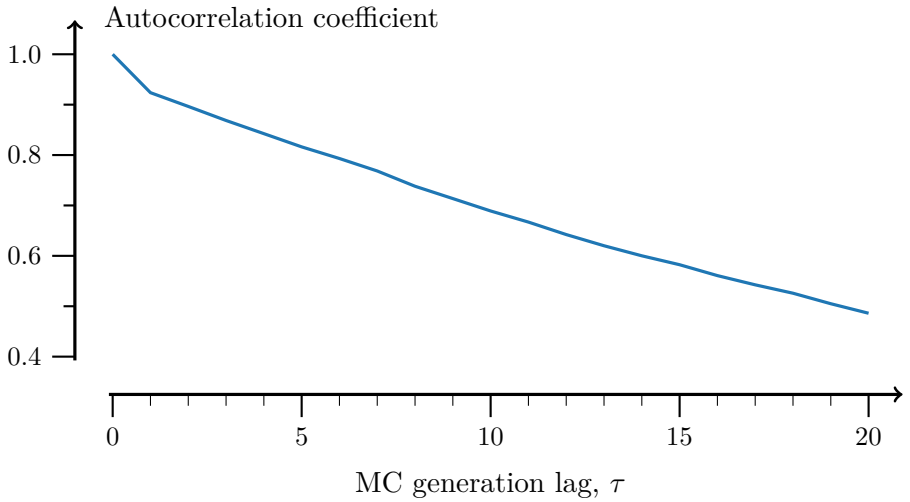


Figure 6-6: Autocorrelation coefficient of the axial offset tally. The values are computed from a simulation with 1 million particles per generation and 1 generation per iteration. The top subplot of Figure 6-3 shows the dataset used to compute these coefficients.

Figure 6-7: Effective number of MC generations for AO tally statistics. The middle column shows the observed standard deviation. The last two columns show the predicted standard deviations using N_{gen} and $N_{\text{gen, eff}}$.

N_{gen}	$N_{\text{gen,eff}}$	Std. dev.	$\propto 1/\sqrt{N_{\text{gen}}}$	$\propto 1/\sqrt{N_{\text{gen, eff}}}$
1	1.00	0.0038	—	—
2	1.04	0.0037	0.0027	0.0037
5	1.10	0.0041	0.0017	0.0036
10	1.20	0.0051	0.0012	0.0034
20	1.40	0.0031	0.0008	0.0032
50	2.17	0.0042	0.0005	0.0026

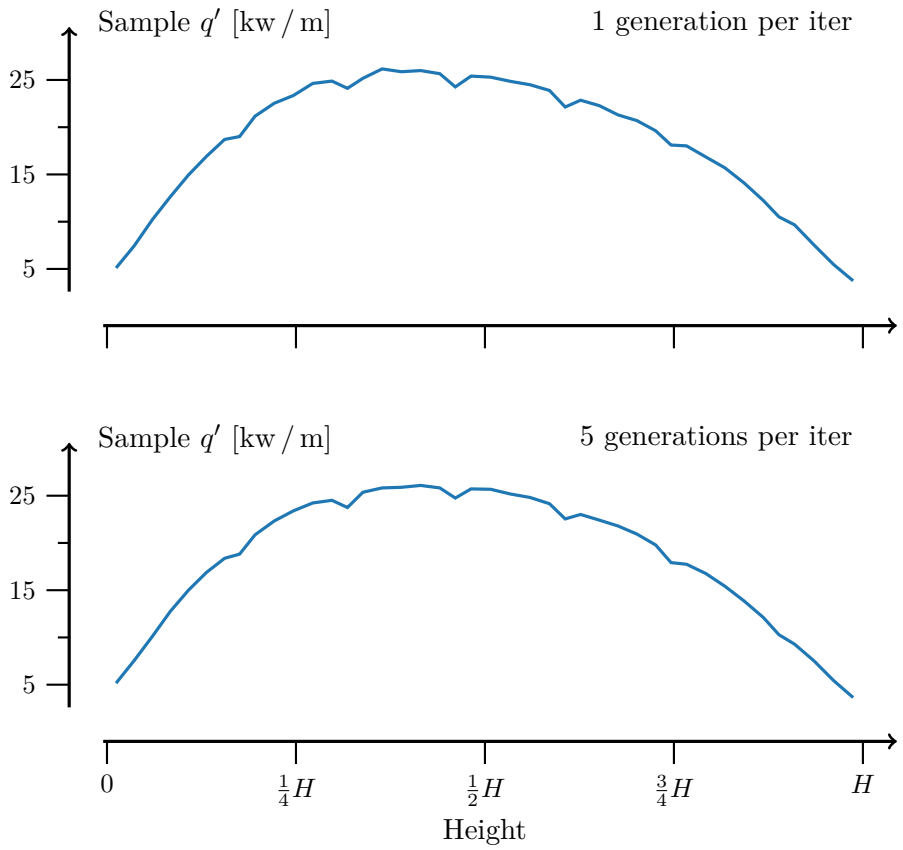


Figure 6-8: Linear heat rate distribution computed from 1 solver iteration. Subplots show results using 1 or 5 MC generations per iteration. Both simulations use 1 M particles per generation.

6.5 Effect of particles per generation

Another important parameter for this solver is how many particles are simulated in each MC generation. Fuel pin simulations without acceleration were run with 1 M (1 million) and 10 M (10 million) particles per generation to study this parameter. Based on the previously discussed findings, these simulations use $N_{\text{gen}} = 1$ which means that a multiphysics iteration is performed and tallies are reset after each MC generation.

Figure 6-9 shows the time series of the axial offset from two simulations with 1 M particles per batch. (One of these series was shown previously in Figure 6-1). These two simulations use different seeds for the MC random number generator to make them statistically independent. Figure 6-10 plots a similar dataset from simulations using 10 M particles per batch.

These plots show that unlike the generations-per-iteration parameter, increasing the number of particles per generation significantly decreases the noise in the stationary region. Likely, this is because the uncertainty in the fission source distribution for any given generation depends on the number of particles per generation but it does not depend on the generations per iteration. In other words, increasing N_{gen} helps with statistical undersampling but does not improve the source distribution; increasing the number of particles tackles both issues which leads to a stronger effect on tally uncertainty.

This discussion has focused on tallies of the axial offset because it is a clear measure of solver convergence, but the same findings about the number of particles and generations are also true for the tallies of multigroup cross sections and other diffusion solver parameters. Accurate tallies of these parameters are important for the performance of the accelerated solver so it is important that simulations use many neutrons per generation.

A downside to increasing the number of particles per generation is that it makes the non-stationary portion of the simulation more expensive. The number of generations required to reach stationarity does not decrease if more particles are used, and those generations become more computationally expensive to run. Thus there is a trade-off between runtime spent in the non-stationary region versus noise in the stationary region. Note that an acceleration method can shift the balance of this trade-off by decreasing the number of generations needed to reach stationarity, as will be discussed in Section 6.6.

One further observation to take from Figure 6-10 is the possibility for deceptive plateaus

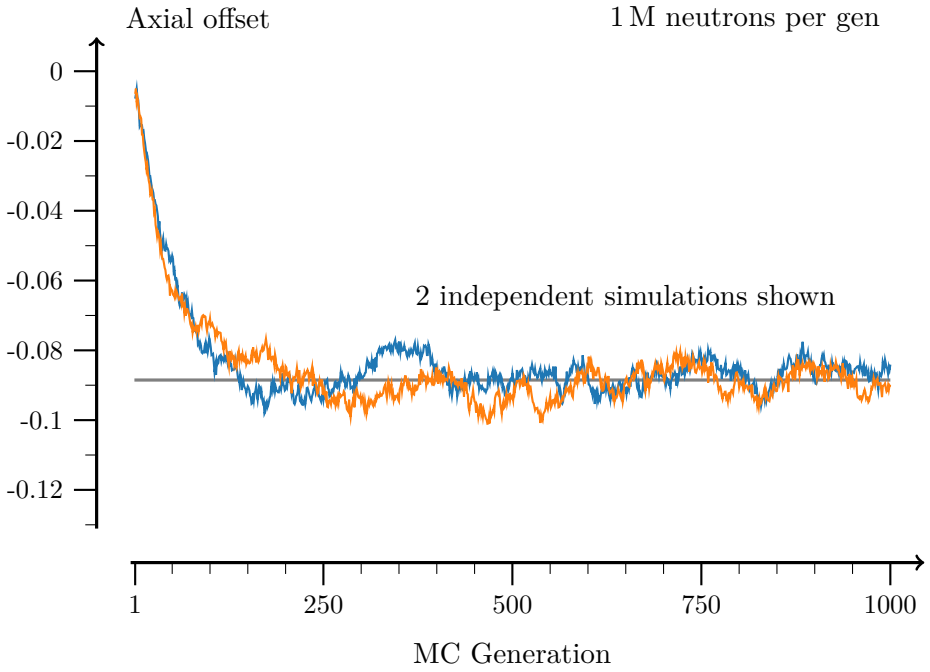


Figure 6-9: Convergence of the axial offset from two unaccelerated multiphysics fuel pin simulations with different random number seeds. Simulations used 1 M neutrons per generation with multiphysics coupling after each generation.

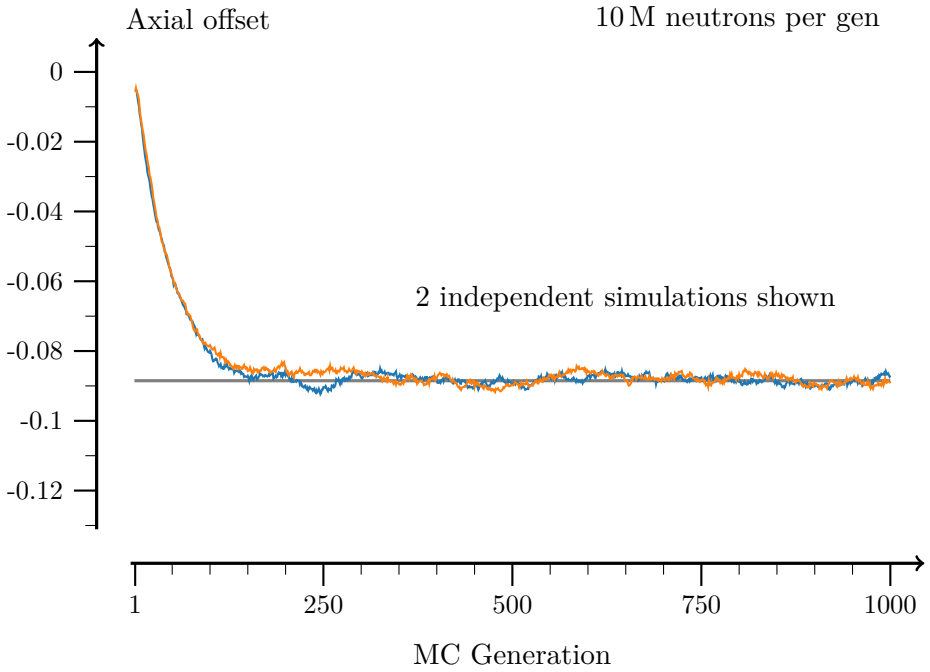


Figure 6-10: Convergence of the axial offset from two unaccelerated simulations with different random number seeds using 10 M neutrons per generation.

in the data. One simulation shown in this figure (plotted by the orange line) reaches an early plateau that lasts for 150 generations, and every value in this plateau lies well below the mean. If the simulation halted before the end of the plateau, the observed mean would be biased, and it would be easy to underestimate the uncertainty of that mean. This is another manifestation of the autocorrelation discussed in the previous section. (In fact, simulations with more neutrons per generation also have greater autocorrelation.)

The implication is that in order to confidently establish convergence either multiple independent simulations should be run or the single simulation should be run for an extremely large number of generations. With acceleration, running multiple independent simulations becomes a more practical option.

6.6 Acceleration

This section discusses the performance of the acceleration multiphysics algorithm (Algorithm 9) on the fuel pin problem. These simulations use 10 million neutrons per generation and 1 generation per iteration. The accelerated algorithm is somewhat complex so the following paragraphs will walk through the first few iterations of the solver in detail, showing the computed power distributions.

The multiphysics solver is initialized with a uniform fuel temperature distribution of 600 K and uniform coolant density of $0.740\,\mathrm{g\,/\,cm^3}$. The MC solver is initialized with a source distribution that is flat in the radial direction and cosine-shaped in the axial direction. The MC solver is then the first of the coupled solvers to be called. It computes the initial distribution of q'. It also computes a set of multigroup cross sections, diffusion coefficients, $\widehat{D}$ correction factors, and derivatives for use in the diffusion-based surrogate solver.[2]

The surrogate solver (which includes the diffusion neutronics, subchannel fluids, and heat transfer solvers) is called immediately after the 1st MC generation. The surrogate first converges the eigenvalue neutronics problem using the diffusion solver. The diffusion solution then provides the second q' distribution computed by the overall solver.

Figure 6-11 shows these two initial q' distributions—one computed via MC and one via the diffusion solver. At this point in the solver, multiphysics feedback has not yet

[2]The MC solver also computes the pin power factors, f. There is only one pin in this problem so the only purpose of f here is to relate the neutron production rate to the heat generation rate. There is significant axial variation in this quantity due to the differences in the neutron energy spectra.

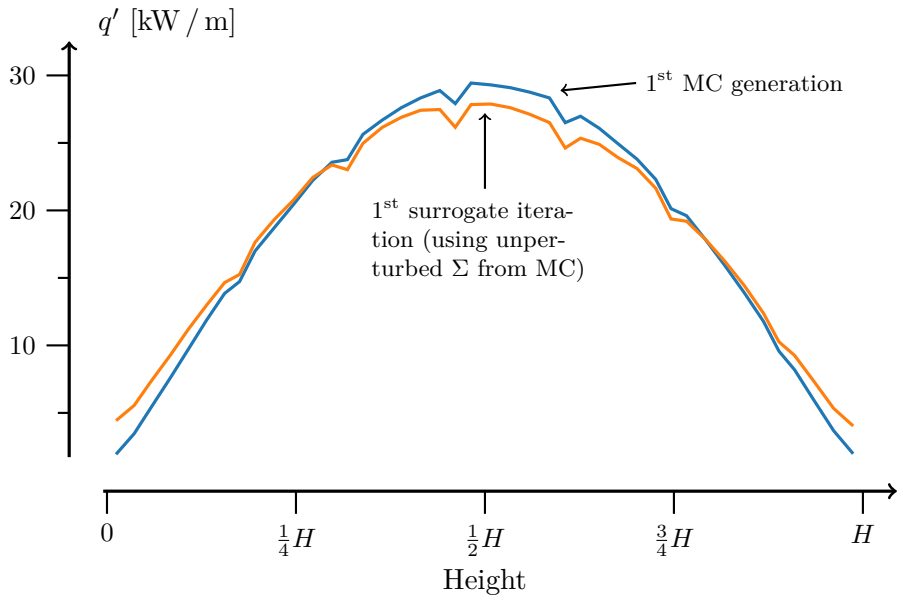


Figure 6-11: Linear heat rate computed by the first iterations of the MC solver and the surrogate solver. Neither solution yet includes temperature or density feedback.

been included. Both the MC and diffusion solvers are operating on the initial guess of a flat temperature and density so the resulting power distribution is approximately cosine-shaped.

The surrogate solver then computes an under-relaxed q' distribution (using the MC q' as the "old" distribution and the diffusion q' as the "new" distribution). The resulting q' is then passed to the subchannel and heat transfer solvers. Note that this is the first time the subchannel and heat transfer solvers are used in the simulation.

New multigroup cross sections and diffusion coefficients are then computed using the MC differential tallies. The diffusion solver is called again with the updated parameters in order to compute a new q'. The process then repeats, cycling through the diffusion neutronics, subchannel fluids, and heat transfer solvers.

The first three q' distributions computed in this surrogate iteration procedure are shown in Figure 6-12. The q' from the second iteration is heavily bottom-peaked, showing the impact of axial coolant density distribution. Fuel temperature feedback then leads to a third q' which is more flat.

After 24 iterations, the surrogate solver converges to the distribution shown in Figure 6-13. For reference, this figure also shows the final converged solution computed after many MC generations (the average q' from the stationary region).

Figure 6-13 shows that surrogate solver is able to very nearly find the converged q' using

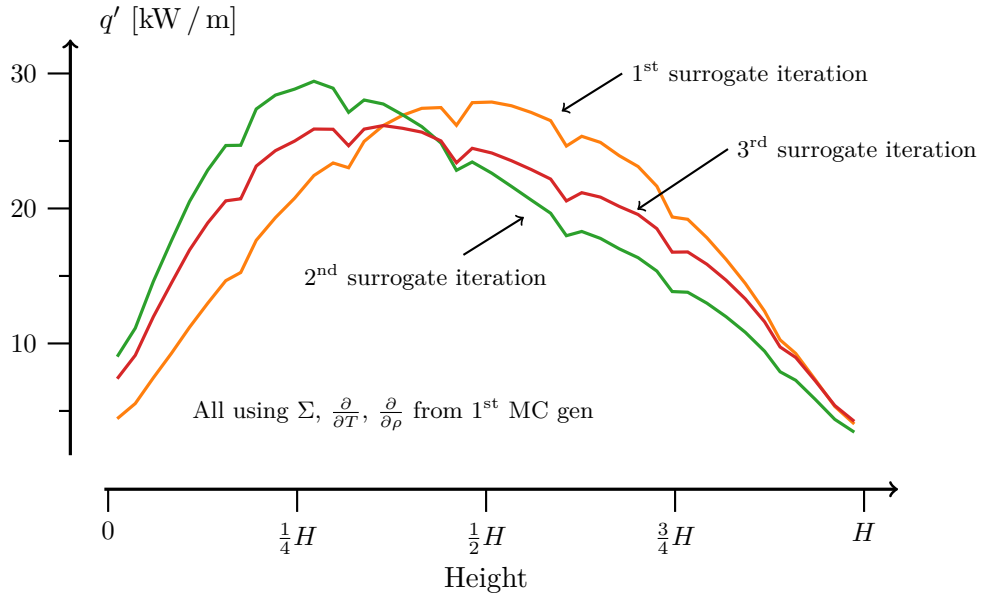


Figure 6-12: Linear heat rate computed by the first three surrogate solver iterations. The non-relaxed values are shown.

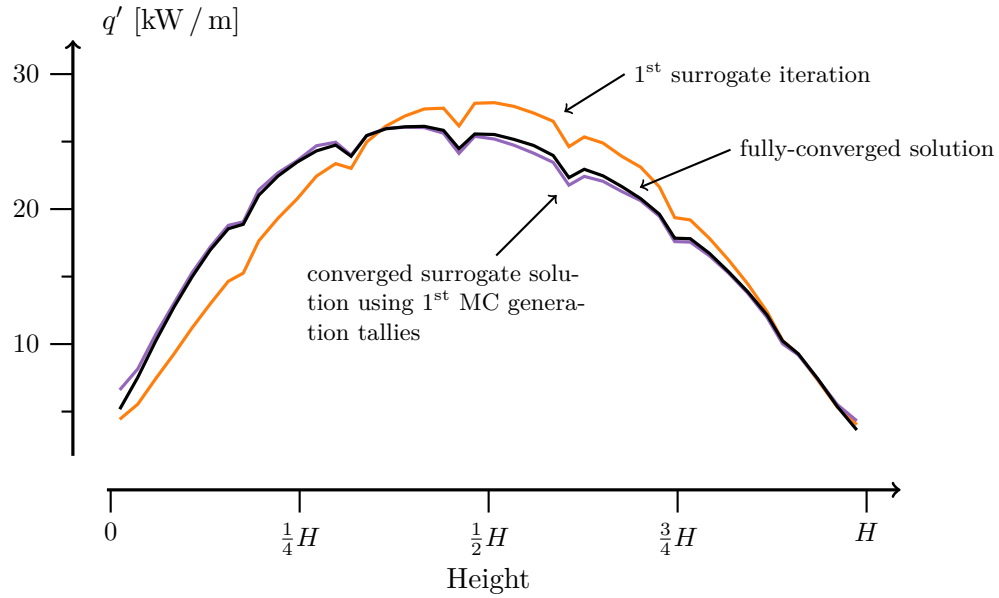


Figure 6-13: Linear heat rate computed by the first three surrogate solver iterations. The non-relaxed values are shown.

tallied values from only the 1st MC generation. The largest error seen in Figure 6-13 occurs at the very bottom of the problem. (It is likely due to the assumption of zero derivatives for $\widehat{D}$). Except for the top and bottom mesh cells, the maximum error is $0.57\,\mathrm{kW}\,/\,\mathrm{m}$—2% of the peak q'. This is impressive given that those initial tallies were computed from a uniform distribution of temperature and density.

The fission source distribution computed by the converged surrogate is then used to re-weight MC source sites. Note that the MC particle weights are clipped at 1.2 and 1/1.2 (then renormalized) as a form of under-relaxation. As indicated in Algorithm 9 the converged linear heat rate computed by the surrogate, $q'_{\text{diffusion}}$, is used with the MC linear heat rate, q'_{MC}, to compute an under-relaxed value, q'_{relax}. The under-relaxation factor (used both to compute q'_{relax} and to compute $q'_{\text{diffusion}}$ within the surrogate) is $\alpha = 0.3$.[3]

The q'_{relax} distribution is then used to compute a new distribution of temperature and density. These values are incorporated in the MC solver, and then the second MC generation is computed. The solver then proceeds in the same fashion until the specified number of iterations have been performed.

Figure 6-14 shows the resulting trend of the axial offset (computed from q'_{relax}) computed after each of the main solver iterations. Two independent accelerated simulations (using different random number seeds) are shown. An unaccelerated calculation is also shown for reference.

This figure shows that the accelerated simulations reach stationarity with many fewer generations and arrive at the same result as the unaccelerated simulation. (The unaccelerated simulation does not reach the final asymptotic value in these first 200 generations, but it eventually will as seen in Figure 6-10).

The signal is noisier with acceleration which would suggest that values must be averaged over a larger number of stationary generations in order to resolve precise answers. However, the accelerated simulations also show less autocorrelation which counteracts the impact of the noise. Thus it is not obvious which method requires more total generations after reaching stationarity.

[3]This under-relaxation factor is probably far smaller than necessary for this problem. Such a small relaxation factor was found to be necessary for small, high-peaking reactor cores that were modeled in the testing and development phases of the solver. A small relaxation factor prevents an intermediate solution from sampling unusually high local powers that cause bulk boiling in subchannels or excessively large cladding temperatures, both of which are error conditions for the solver to prevent it from extrapolating fluid properties outside of the IAPWS region 1. The relaxation factor was not tuned for this fuel pin simulation because it is computationally cheap regardless of relaxation.

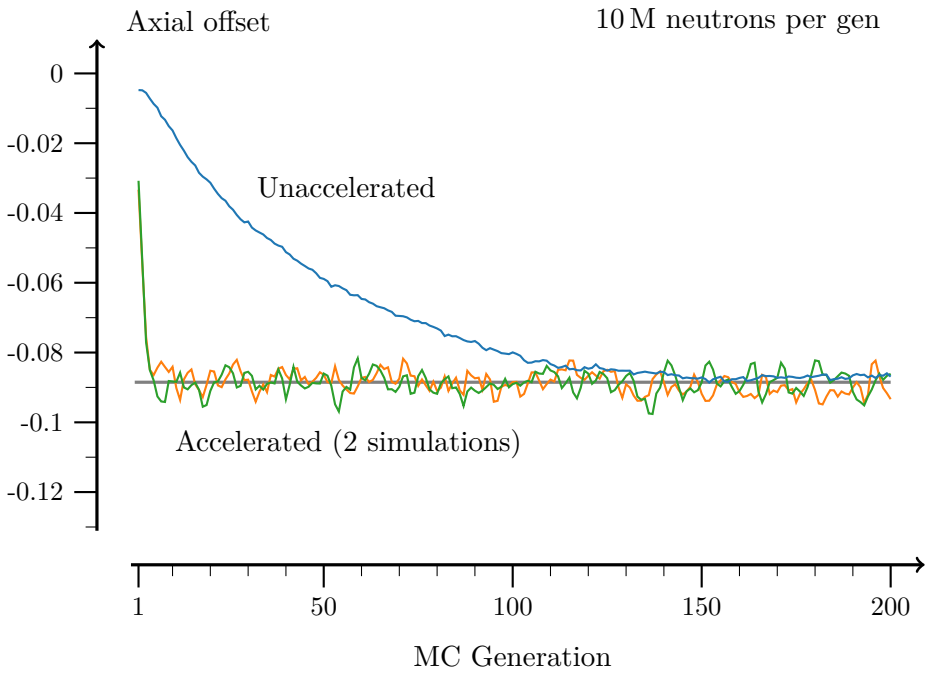


Figure 6-14: Convergence of the axial offset of fuel pin calculations with and without acceleration. Two accelerated simulations with different random number seeds are shown. All simulations use 10 M neutrons per generation.

It is however obvious which method reaches stationarity more quickly. Due to autocorrelation, it is difficult to say exactly when the unaccelerated algorithm reaches stationarity, but it is probably between 150 and 250 generations. The accelerated algorithm requires only requires about 5. The tallies required for acceleration slow each MC generation but only by a factor of 50% so the overall runtime gain is still 20× to 30×.

Lowering the cost to reach stationarity is important not just because it makes any given simulation faster but also because it makes running multiple independent simulations a more practical option. With redundant simulations, the time spent converging the fission source is essentially wasted, but acceleration reduces that wasted time. As discussed previously, independent simulations are important when using a MC solver to guard against false indications of convergence. Similarly, acceleration also makes it more practical to run simulations with more neutrons per generation which results in a less noisy fission source site distribution.

6.7 Summary

Accurate MC tallies are important for the performance of the solver presented in this thesis. At a minimum, each solver iteration must be able to sufficiently resolve the power distribution for multiphysics feedback. With acceleration, the solver must also be able to resolve multigroup cross sections and other diffusion parameters.

The simulations shown in this chapter demonstrate that the common practice of averaging tallies of multiple MC generations is an ineffective way to reduce tally uncertainty. Instead, it is better to increase the number of particles per generation (to improve the accuracy of a given simulation) and run multiple independent simulations (to establish uncertainty on the final solution).

This chapter also demonstrates that the surrogate solver (using diffusion-based neutronics) can find accurate solutions to the multiphysics problem from the very first MC generation. It is consequently useful for acceleration purposes, and it can reduce the total number of MC generations needed to reach stationarity by an order-of-magnitude.

Note that these arguments relating to acceleration, particles per generation, and generations per iteration are tightly inter-related. Acceleration both enables and necessitates simulations that use many particles per generation; The autocorrelation issues due to using multiple MC generations are made more acute by the tallies needed for acceleration. Making a fast solver entails considering all of these variables simultaneously.

Chapter 7

Quarter-Core Simulations

This chapter presents simulations on a quarter-core pressurized water reactor problem. A benchmark of a real operating reactor is used which allows for comparison of the simulated results against empirical measurements. High accuracy is not a goal of this thesis, but seeking some degree of accuracy is helpful in ensuring that the presented methods are useful for works beyond this thesis.

This chapter will also discuss in detail the convergence behavior of the accelerated solver. Comparisons are made against unaccelerated calculations, and against other published works.

7.1 Model description

The geometry and material definitions used here closely match those specified in the BEAVRS benchmark [47]. This benchmark describes a typical large Westinghouse reactor with a nominal thermal power of 3411 MW produced by 193 fuel assemblies using a 17×17 rod-pattern. Figure 7-1 shows a map of one quadrant of the core and the checkerboard enrichment pattern.

Some minor deviations are made from the benchmark geometry. The sleeves attached to grid spacers have been neglected for simplicity. The region immediately below the fuel rods is modeled using the VERA benchmark definitions instead of the BEAVRS definitions because they are expected to be better representations of the geometry [66]. With the VERA model, there is a 4.2 cm water-filled gap between the top of the lower nozzle and the bottom of the fuel end plugs. Like the VERA benchmark, the nozzle here is modeled

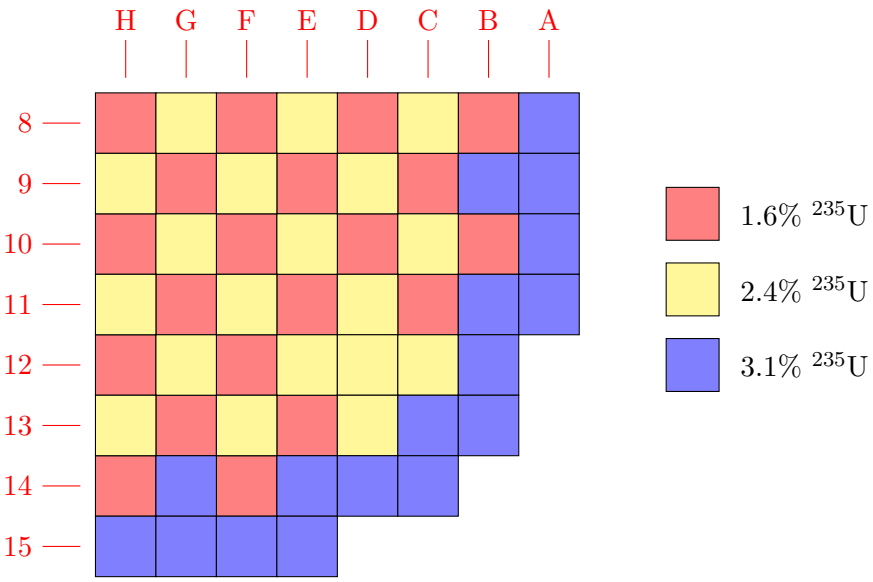


Figure 7-1: Map of a quadrant of the BEAVRS cycle 1 core with color indicating assembly enrichment.

simply as a homogeneous region with a material that is 50% steel and 50% water by volume. In contrast, BEAVRS uses a discrete steel block with fuel-pin-sized holes, and the nozzle is in contact with the fuel. Radially, the model used here includes the baffle, barrel, neutron pads, and downcomer. The reactor pressure vessel is not included.

A simplified approach is also used for the instrumentation tubes. The voided instrument thimbles are not modeled and each instrument guide tube is instead filled with water. A trace amount of ^{235}U is added to the water inside the instrument tubes which facilitates OpenMC tallies of ^{235}U fission rate in order to simulate measurements with fission detectors. Note that replacing the instrument thimbles with a moderator will introduce an error, but this approximation has the advantage of making the geometry completely octant-symmetric. Asymmetry in these quarter-core simulations can then be used as an indicator of convergence errors.

The BEAVRS benchmark includes measurements taken at hot-zero-power conditions. Checking against these measurements helps to ensure that the there are no large errors in the geometry and material definitions. Section D.1 of the appendix shows how the computed distributions compare with the measurements.

7.2 Comparison to full power measurements

Most of this chapter is devoted to discussing the fine details of the solver convergence, but first, this section will discuss how well the final converged results match measurements in the reactor at full power conditions.

The empirical data comes from day 169 of the BEAVRS first cycle, the first day where in-core flux measurements were made at full-power. Although core burnup would be quite high for 169 days of typical operation, the BEAVRS first cycle has many days of low power operation or even shutdown so the core burnup on this day is only 1.5 MWd/kg. An equilibrium xenon calculation is used to account for ^{135}Xe buildup, but no other depletion calculations are used. The benchmark specifies the conditions for the power level, boron concentration, inlet coolant temperature, and RCCA positions matching these measurements.

Two independent simulations were run to guard against statistical outliers, and each simulation used 30 MC generations with 200 million neutrons per generation. The diffusion neutronics, subchannel, and heat transfer solvers are run after each generation per Algorithm 9, and all tallies are reset after each generation.

Starting from an initial fission source that is radially-uniform and axially-cosine, the accelerated solver reaches stationarity by the 6$^{\text{th}}$ MC generation. The details of this convergence are discussed in the remainder of this chapter. The distributions computed by each of the two simulation are averaged over generations 6 through 30, and the resulting two distributions are again averaged to arrive at the results presented here. In total, the two simulations used for this section required 2 700 cpu core-hours.[1]

Figure 7-2 focuses on the radial distribution of detector signal and compares the computed and measured values. The reactor core is very nearly octant-symmetric so measured values from across the entire core have been collapsed down into the one plotted octant. The axial distributions are integrated to arrive at these radial values. The plotted data show an error in the calculated results in the form of an in-out tilt that peaks at about 8% relative error.

The source of the in-out tilt error is unclear. Several physics approximations have been made, and each can be imagined to play a role (no depletion, fixed fuel-clad gap width, no axial swelling, local energy deposition, neglected neutron capture energy, radially-uniform

[1]These simulations used 12 Intel Xeon E5-2683 v4 processors totaling 192 processor cores.

fuel heat depostion). The two simulations find eigenvalues of $k = 0.99822$ and $k = 0.99824$ so the net affect of these approximations appears to be too much negative feedback.[2]

Note that an 8% radial error is probably unacceptable for industrial use, but it is small enough that it is not expected to significantly impact the conclusions of this thesis. A more accurate solver would likely require a very similar number of multiphysics iterations and neutron eigenvalue iterations, and it would likely face similar difficulties with multiphysics instabilities.

Figure 7-3 shows the computed and measured axial distributions for assembly D12. The data are normalized so that they integrate to the same value. Assembly D12 is noteworthy because it is the highest-power assembly and it clearly shows the effect of the partially inserted RCCA D-bank. A tilt error can be seen in this data, with the values under-predicted near the bottom of the core and over-predicted in the upper parts of the core. This figure also shows that the depressions caused by spacer grids are deeper for the computed data relative to the measured. It is unclear if this is due to the simple model used for

[2]This is a 180 pcm error. How big is a pcm? My personal reference point is appended to this thesis in Section D.2.

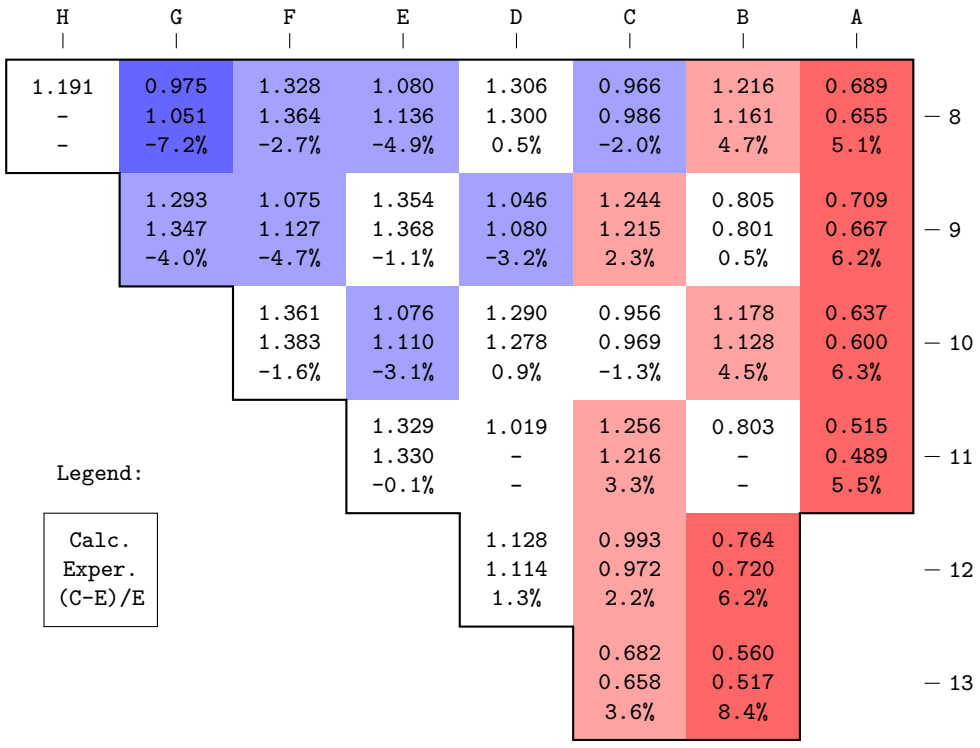


Figure 7-2: Calculated and measured detector response in each fuel assembly on day 169 of operation (hot full power).

the grid spacers, the instrument tube model which is filled with coolant, or some other approximation.

Figures 7-4 and 7-5 show the measurements for assemblies J08 and B03. Assuming octant symmetry, these correspond to assemblies G03 and B13 in the plotted octant, and these assemblies are notable because they show the peak radial errors as indicated by Figure 7-2. (These curves have been renormalized so the integral error is not shown in these plots.) The computed data for G08 shows greater stochastic noise because this instrument tube is cut in half by the reflective boundary condition (which leads to fewer MC samples).

The B03 position is also particularly useful because measurements were taken in two other assemblies that lie in symmetric positions: N02, and N14. All three of these measurements are plotted in Figure 7-6 to give an idea of the uncertainties in the data. The computed distribution for this location is also shown. Again, the flux depressions due to the spacer grids are clearly overestimated. Otherwise, a small tilt towards the top of the core can be seen, but the calculations for this assembly lie just at the edge of the measured spread.

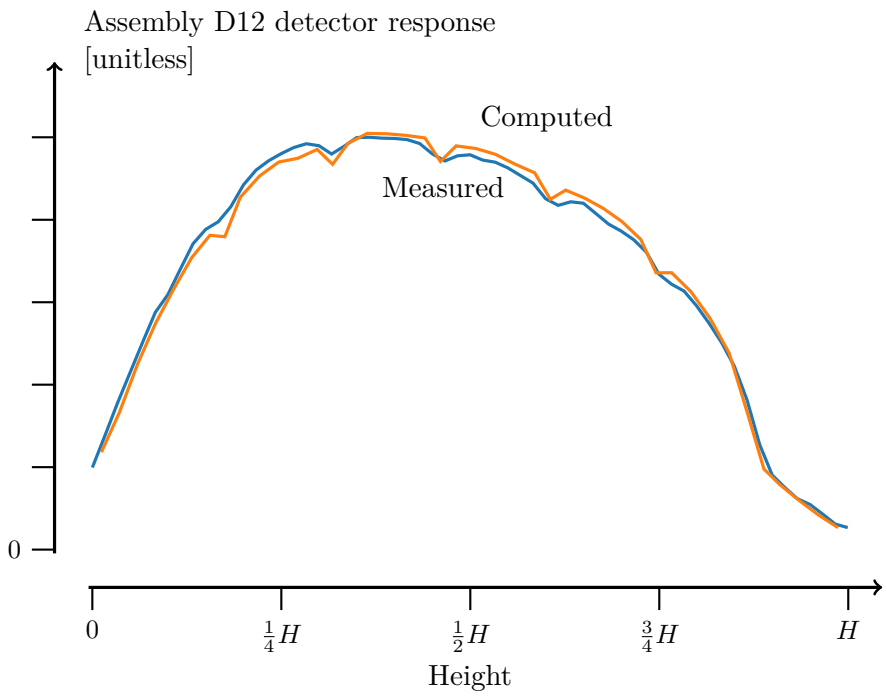


Figure 7-3: Measured and computed axial detector profile in assembly D12 on day 169 of operation. Both curves have been normalized to the same integral.

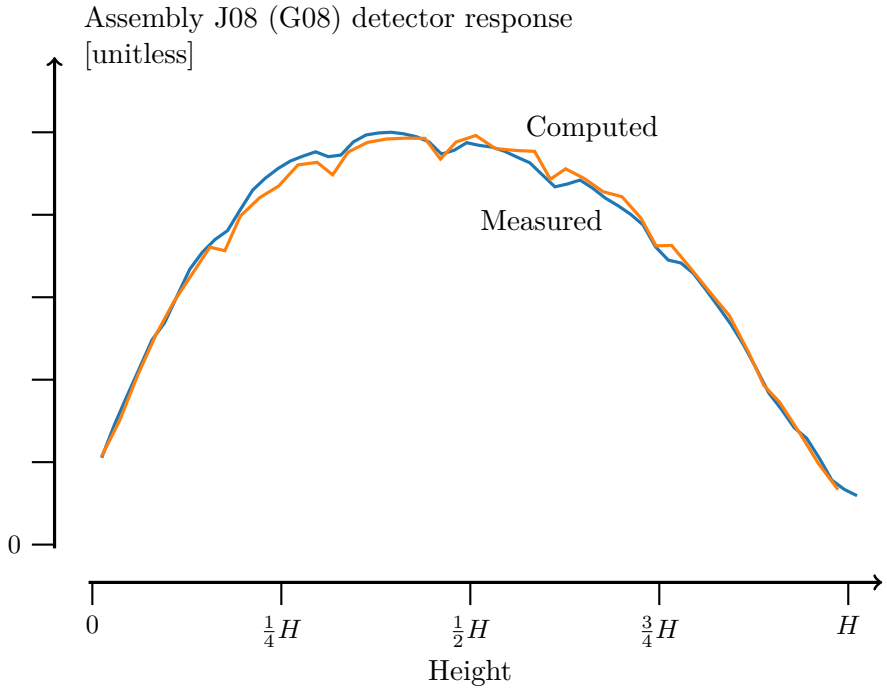


Figure 7-4: Measured and computed axial detector profile in assembly J08 (same position as G08 with octant symmetry) on day 169 of operation. Both curves have been normalized to the same integral.

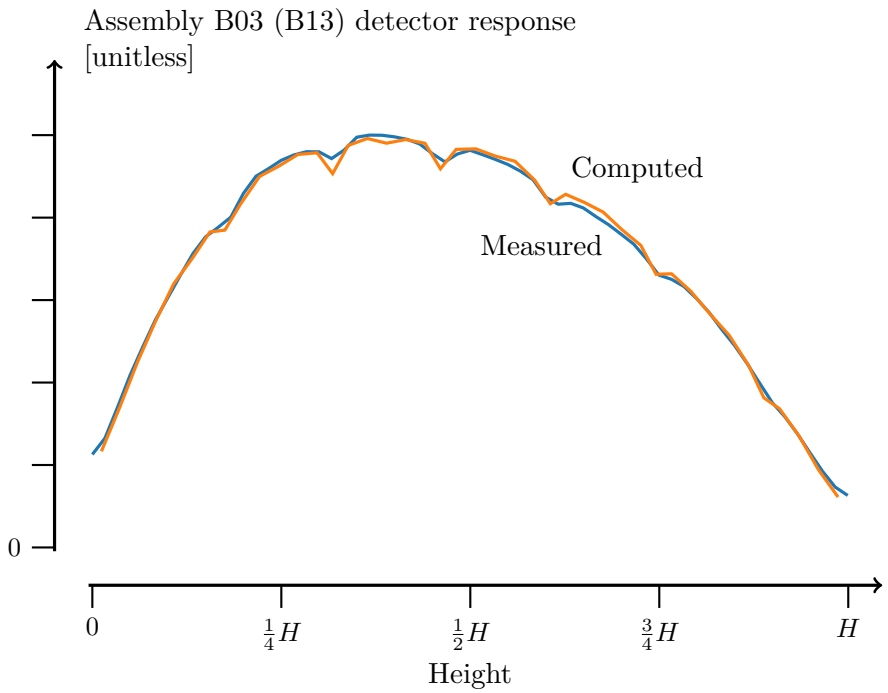


Figure 7-5: Measured and computed axial detector profile in assembly B03 (same position as B13 with octant symmetry) on day 169 of operation. Both curves have been normalized to the same integral.

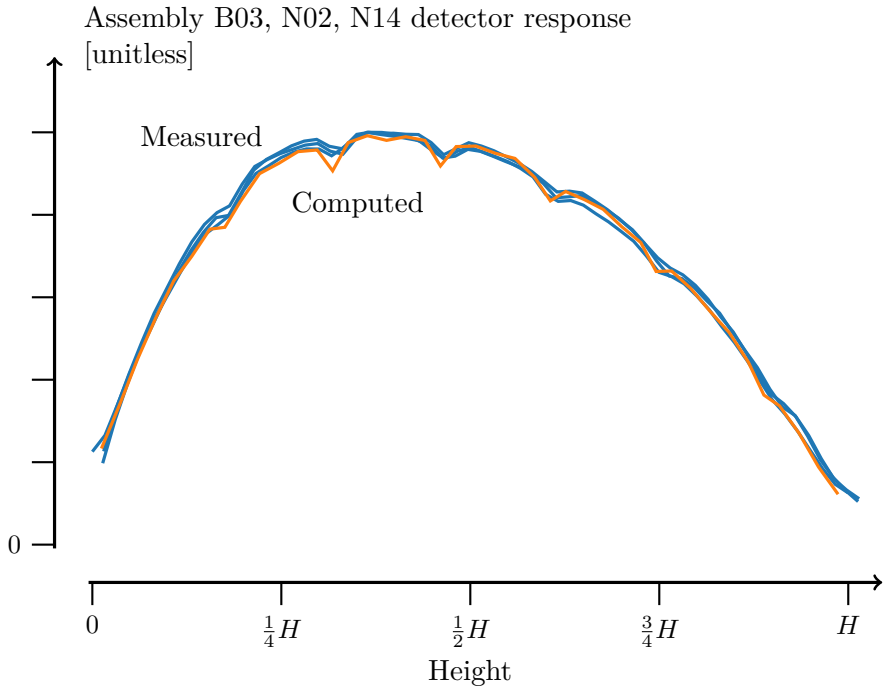


Figure 7-6: Measured axial detector profiles in assemblies B03, N02, and N14 are shown in blue. Each of these assemblies occupies the same position assuming octant symmetry. The computed profile is shown in orange.

7.3 Details of the coupled solution

The previous section focused on the instrument fission rate because this value can be compared against measurements published in the benchmark. This section explores other computed results in order to highlight some features of the coupled physics system.

Figure 7-7 shows the computed radial distribution of pin power. This dataset clearly shows the effects of burnable poison rods and the checkerboard pattern of fuel assembly enrichment. Sharp gradients in the power distribution appear at assembly boundaries where assemblies with higher enrichment experience an incoming current of thermal neutrons from the lower enriched assemblies

There are also strong gradients across assemblies at the core periphery where power across an assembly can vary by more than a factor of $2\times$. These assemblies highlight the advantage of quarter-core and full-core Monte Carlo neutronics. In the traditional toolchain with lattice physics solvers, the lattice physics parameters are usually computed from a simulation of an assembly with reflective boundary conditions which leads to a roughly flat power distribution. Methods developers must then take great care when using these parameters in a core-sized calculation which shows a more complex flux distribution. Full-core MC eliminates the lattice physics step which can simplify software development and validation at the cost of computational expense.

Figure 7-8 shows the distribution of outlet coolant temperature. The subchannel mixing model used here results in a very smooth outlet temperature distribution. The mixing is so strong that it is difficult to discern from this plot which subchannels border a guide tube and which are surrounded by fuel pins. However, steep gradients in temperature can still be found at the core periphery.

Figure 7-9 shows the inlet velocity distribution computed by the subchannel solver. With the simple models used here the distribution is mostly determined by the hydraulic diameter and consequently the frictional force of each subchannel. This results in less velocity in the subchannels adjacent to guide tubes which have a smaller hydraulic diameter and conversely greater velocity in the subchannels between fuel assemblies.

Note that the solver misses some important details such as the fact that grid spacers likely have different pressure loss coefficients for interior subchannels versus the subchannels that bridge the space between two fuel assemblies. Furthermore, no consideration is made

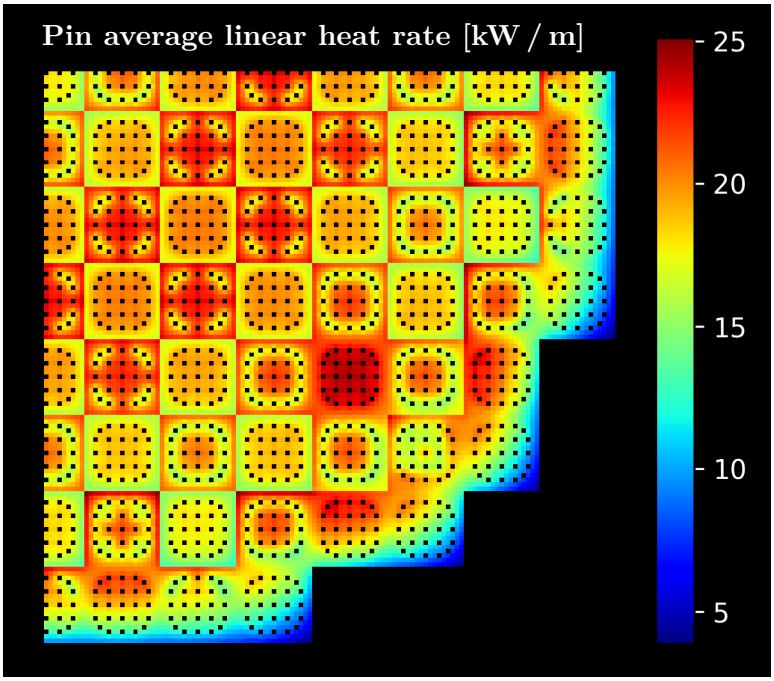


Figure 7-7: Calculated average q' in each fuel pin at full-power.

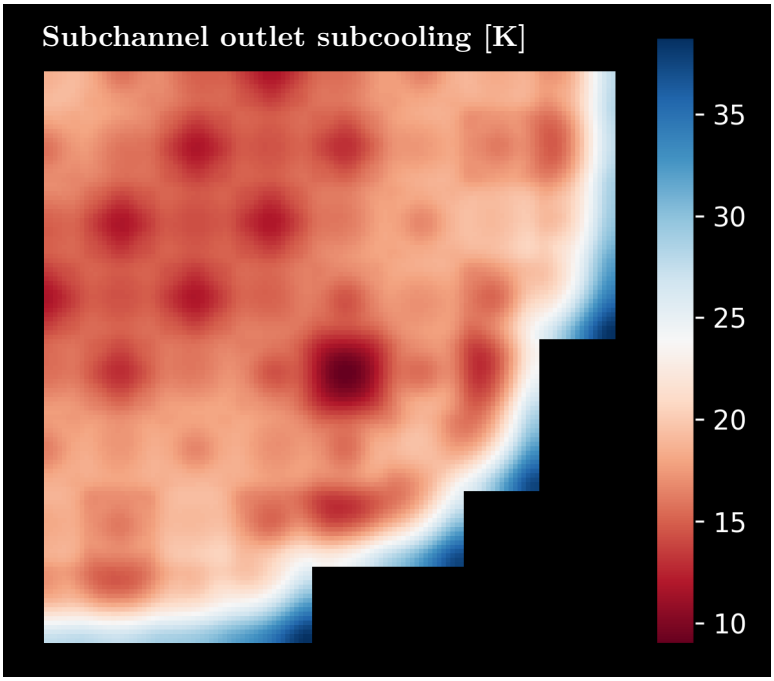


Figure 7-8: Outlet subcooling ($T_{\text{sat}} - T$) at full-power.

for the geometry of the lower nozzle. Hence large errors in the inlet velocity distribution are expected, but the impact of this error on the temperature distribution is partly mitigated by the large amount of turbulent diffusion in the system.

The solver also finds that the power distribution has a small impact on the inlet velocity. This is due to additional friction that results from the low-density, high-velocity flow in warm subchannels. The impact of the power distribution can most clearly been seen for the peripheral assemblies in Figure 7-9. Assembly D12 also shows a noticeably lower inlet velocity due to its high power.

A combination of the axial and radial variation is shown in Figures 7-10 and 7-11. These figures show a diagonal slice through the core from the upper-left of assembly E11, through assembly D12, and to the lower-right of assembly C13. The figures respectively show the fuel pin linear heat rates and the subchannel coolant temperatures along this line.

This diagonal slice is particularly interesting because it samples three very different fuel assemblies. Assembly E11 is a typical assembly with 1.6% ^{235}U enrichment. D12 is a 2.4% assembly that contains partially-inserted control rods from the D-bank, and it is the highest power assembly in the core. C13 is a 3.1% assembly that lies along the core boundary and includes pyrex rods in some guide tubes. These slices highlight the 3D nature of the data

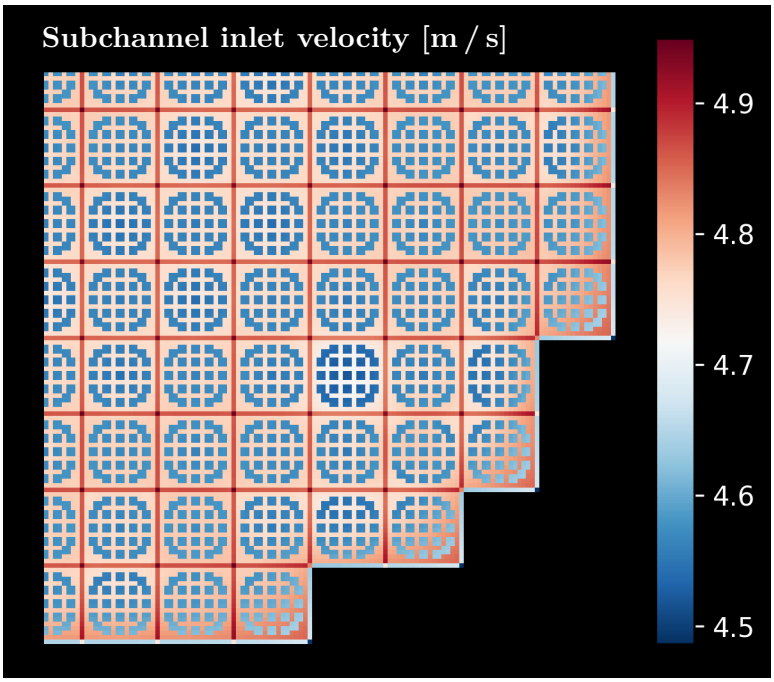


Figure 7-9: Inlet velocity distribution at full-power.

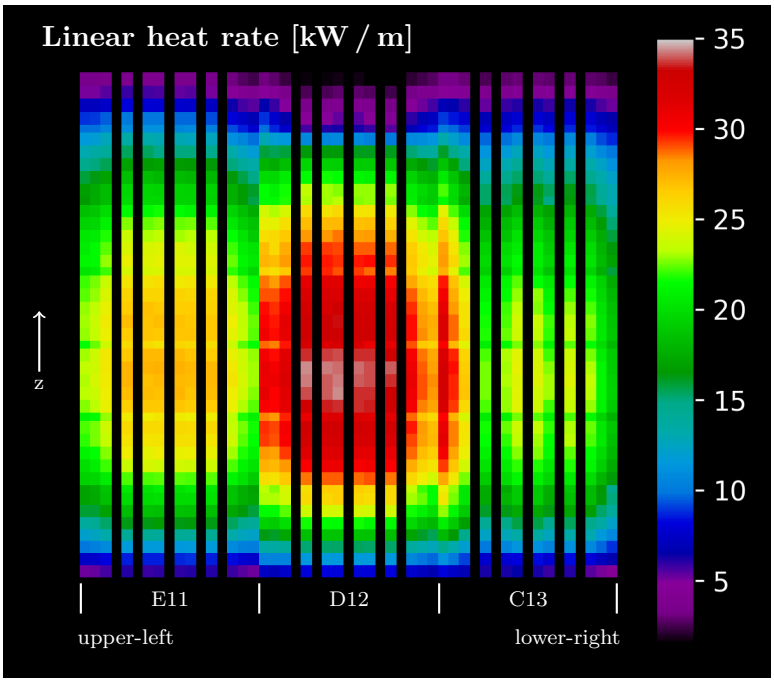


Figure 7-10: Calculated q' in a diagonal slice through the reactor from the upper-left of assembly E11 to the lower-right of assembly C13.

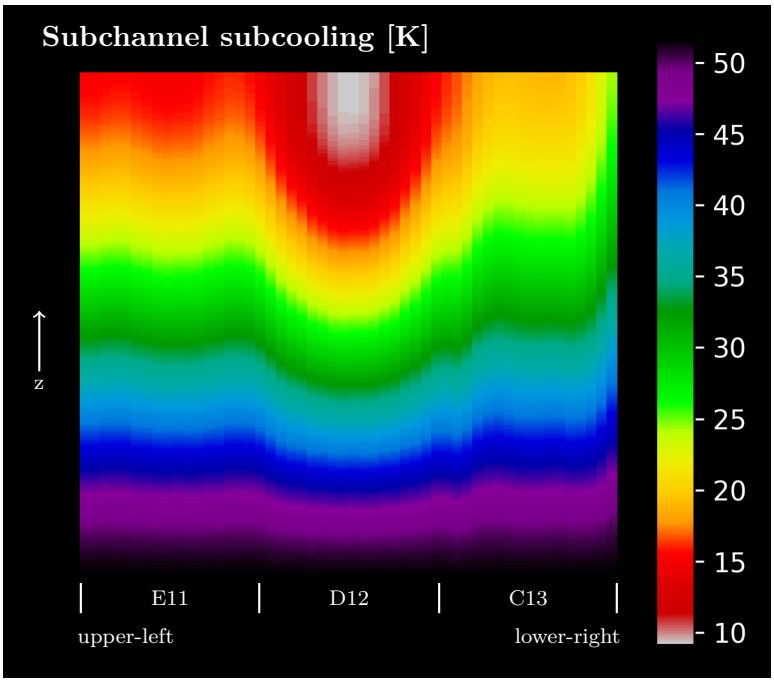


Figure 7-11: Coolant subcooling in a diagonal slice through the reactor from the upper-left of assembly E11 to the lower-right of assembly C13.

and the importance of tools that can resolve it.

Figure 7-12 shows the axial temperature distribution of the cladding and surrounding coolant for the hottest pin in assembly D12. These quantities would be important to resolve for safety analysis as the reactor power is limited by the highest fuel cladding temperatures which can cause thin margins to DNB, accelerated corrosion, and loss of mechanical strength.

The surface temperature of this fuel rod exceeds the saturation temperature for a significant portion of its length, indicating a large region of subcooled nucleate boiling. The nucleate boiling limits the maximum temperature of this rod to 353 °C. Note that the Watts Bar Unit 2 FSAR cites a hot spot temperature of about 350 °C for normal operation [24]. This agreement with the FSAR provides some confidence in the correctness of the implementation of the coupled solver.

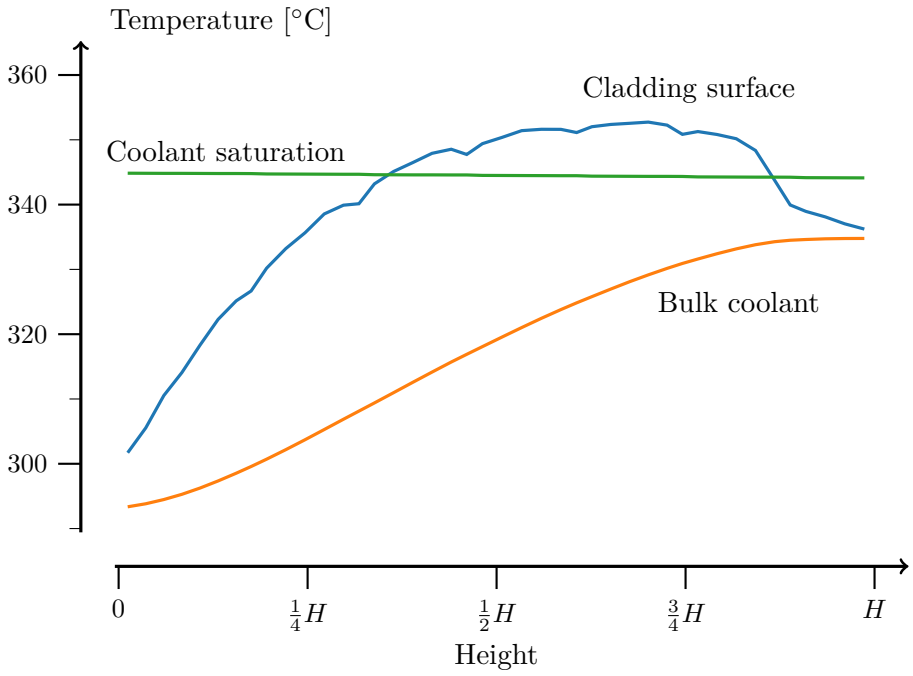


Figure 7-12: Temperatures for the hottest pin in assembly D12.

7.4 Convergence behavior of the q' distribution

The linear heat rate values computed by the solver in each of the main iterations are stored in an output file. These values can then be analyzed after-the-fact to understand the convergence of the solver.

Note from Algorithm 9 that in each main iteration the solver first executes 1 MC generation, and the resulting tallied linear heat rate is labeled q'_{MC}. Then, the surrogate system using the diffusion solver is converged, and the final linear heat rate from this step is labeled $q'_{\text{diffusion}}$. Both of these variables are then used to update the under-relaxed value which is labeled q'_{relax}.

7.4.1 Axial variation

All three linear heat rate distributions are stored in the output file. Figure 7-13 shows the convergence behavior of these three variables from one simulation. For a holistic picture, the heat rate at three different axial locations is shown. Values are taken from the axial mesh cell nearest the indicated height, e.g. cell 0 for $0H$ and cell 21 for $\frac{1}{2}H$. All values are averaged radially across the entire core.

In the early iterations, all three q' variables can disagree significantly, but they converge as the simulation progresses. The diffusion-based surrogate solver generally shows an overshoot in its convergence trend. If the asymptotic q' is above the initial q' then the diffusion solver will predict an overly large value in the early iterations. The overshoot is partly due to xenon feedback (the surrogate does not attempt to predict xenon changes), but it also occurs in simulations without xenon feedback. This overshooting is damped by the explicit under-relaxation on q' and by the source site weight clipping.

As the simulation nears convergence, the MC solver begins to closely lag the diffusion solver. This is particularly apparent for the middle subplot in Figure 7-13 after generation 4. At that point, q'_{MC}, matches the previous $q'_{\text{diffusion}}$ almost exactly. At this stage in the solver, the temperature and density distribution used by the MC solver are relatively stable (they are computed from q'_{relax}) so this lagging effect must be caused by the source site reweighting procedure. In other words, the diffusion solver and MC solver are using the same fission source distribution (lagged by 1 iteration) so they so they agree on the resulting q' distribution (lagged by 1 iteration). This suggests that the coupled solver could rely solely

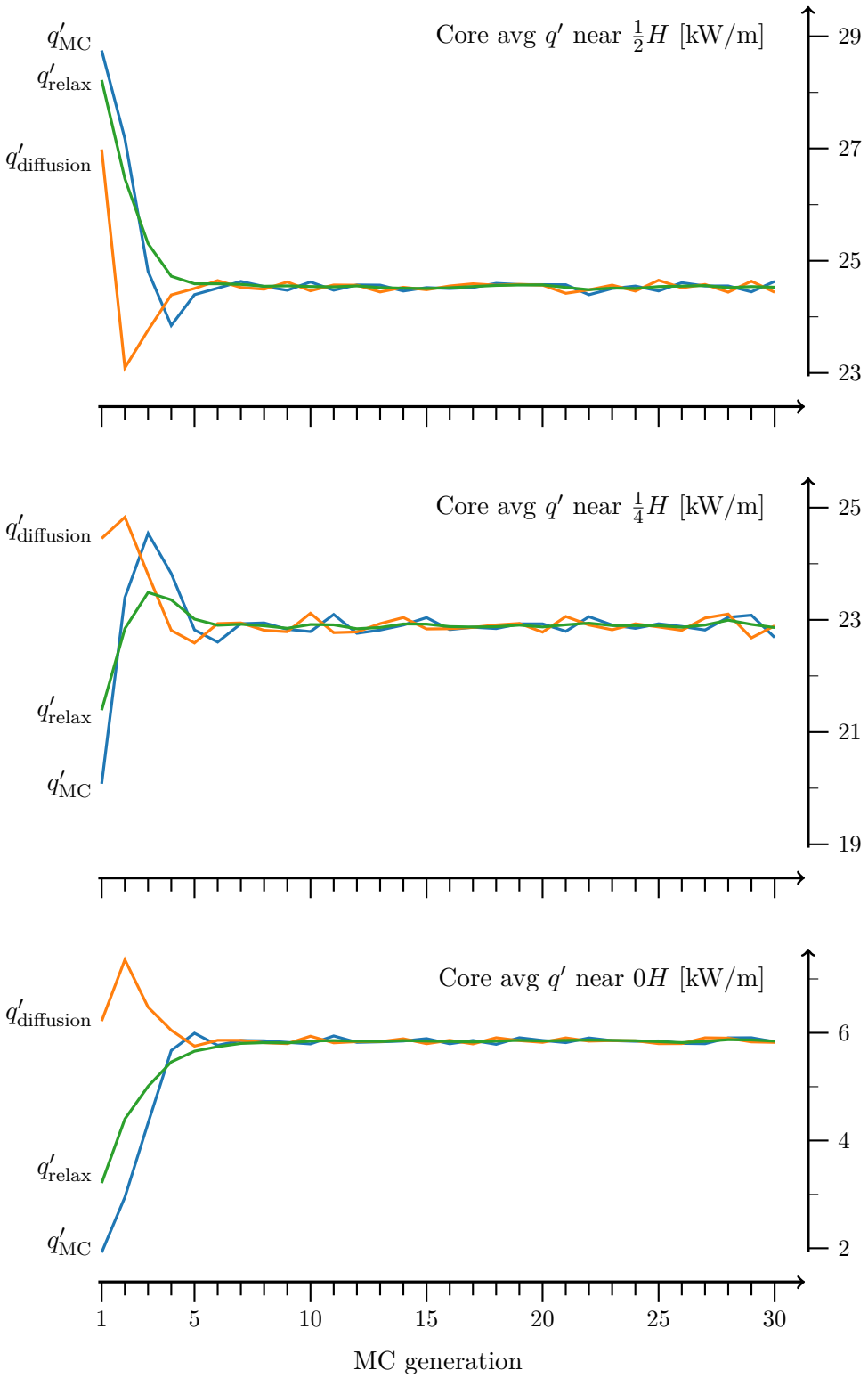


Figure 7-13: Convergence of the linear heat rate variables. The subplots show the core-average linear heat rate at different axial locations.

on $q'_{\text{diffusion}}$ and ignore q'_{MC} when computing the under-relaxed heat rate.

Another observation from Figure 7-13 is that the data from all three heights appear to reach stationarity at the same generation. This is helpful because it simplifies the process of convergence checking—only values from one or two heights must be considered.

A final take-away from Figure 7-13 relates to the stochastic noise seen in the stationary region. There are occasional small jumps in the q' data at $\frac{1}{4}H$. These jumps are also present at $\frac{1}{2}H$, but they are much smaller. This difference in noise magnitude between $\frac{1}{4}H$ and $\frac{1}{2}H$ cannot be explained by a difference in the power level at these locations; they have a very similar q'.

To explore this phenomenon further, Figure 7-14 shows the standard deviation of the stationary q' values for every axial location. There are clear peaks just below $\frac{1}{4}H$ and near $\frac{3}{4}H$. The location of these peaks could plausibly correspond to the extrema of an important eigenmode in the system. One-speed diffusion theory applied to a 1D bare homogeneous slab predicts that the second-most-dominant eigenmode is sinusoidal with extrema near $\frac{1}{4}H$ and $\frac{3}{4}H$. Perhaps noise in the MC-tallied cross sections cause oscillations in the diffusion solver that correspond to the eigenmodes.

Because the noise peaks near $\frac{1}{4}H$, other convergence plots in this chapter focus on the $\frac{1}{4}H$ axial location.

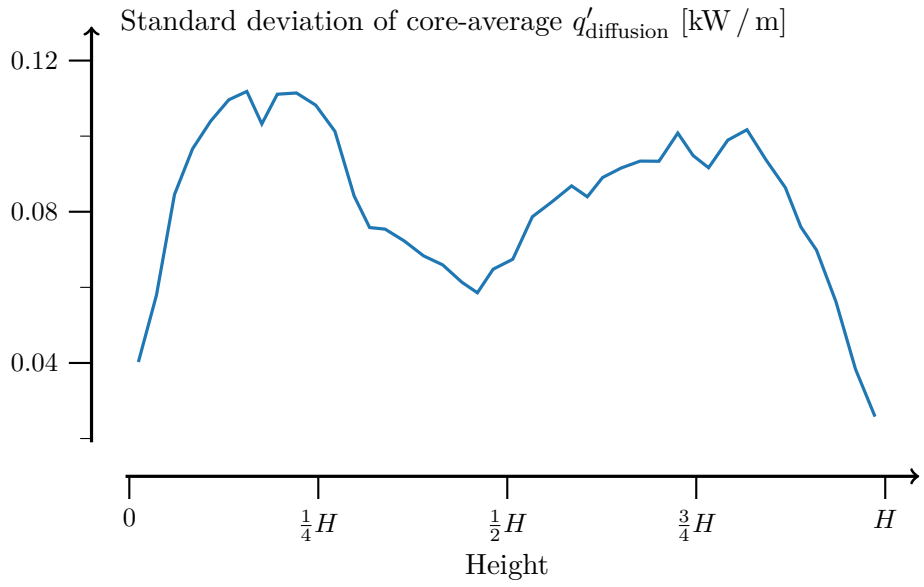


Figure 7-14: Axial variation in the standard deviation of $q'_{\text{diffusion}}$ after reaching stationarity. This data quantifies the noise seen in Figure 7-13 at all axial heights.

7.4.2 Radial variation

The solution to this problem is 3D, so it is important to also check values that have not been averaged over the entire xy-plane when establishing convergence. For this reason, Figure 7-15 shows the convergence of q' at $\frac{1}{4}H$ in three different assemblies. (Refer to Figure 7-1 for the location of these assemblies.)

The three assemblies shown in Figure 7-15 are spread across different locations in the core. The total change in q' from the initial generation to the final varies significantly across these assemblies—about 3 kW / m for D12 but less than 0.5 kW / m for B13. Despite the differences, q' reaches its asymptotic value practically simultaneously in all 3 cases (at about generation 6).

Again, interesting behavior can be found in the noise of the stationary region. Notice that the noise is similar for each assembly. This is somewhat surprising given that the tally region for assembly G08 is only half the size of the others—G08 is cut in half by the quarter-core boundary conditions. This reinforces a notion touched on in Chapter 6 that simply increasing the number of MC samples in a tally region does not guarantee a decrease in the stochastic noise. When the number of samples in a tally region is sufficiently high, it appears that oscillations in the fission source distribution are the dominant driver of noise.

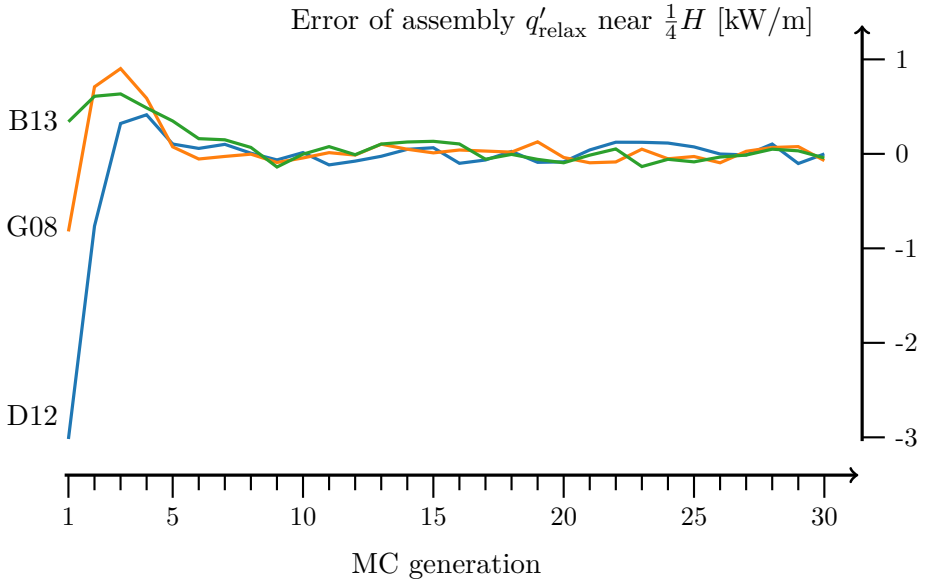


Figure 7-15: Convergence of the linear heat rate in different assemblies. Values are taken from the $\frac{1}{4}H$ location and averaged over all fuel pins in each assembly. Values are expressed as an absolute error by subtracting the asymptotic solution from each assembly.

7.5 Verifying an unbiased surrogate

It is important that the surrogate solver does not introduce a bias in the converged solution. The CMFD conservation equations are designed to be consistent with the MC solver, but very subtle errors in implementation can lead to a biased surrogate.[3]

To check for a bias, a simulation was run that uses acceleration to reach a converged solution, and then acceleration is turned off. The resulting q'_{relax} data for one region of the core is shown in Figure 7-16. The first 15 generations of this simulation use acceleration. For the remaining 44 generations, the diffusion solver is not run, MC source sites are not reweighted, and only q'_{MC} is used to update q'_{relax}.[4]

There is less noise in the unaccelerated portion of the simulation. This should be expected given that simulations without acceleration show extreme autocorrelation (as demonstrated in Chapter 6). More importantly, there is no significant trend in the unaccelerated

[3] For example, an early implementation of this solver used a used a core-average Q-value for fission when computing the pin power factors. This lead to a significant bias in the converged solution as the Q-value differs greatly between different pins and axial locations, likely due to differences in the ratio of ^{235}U to ^{238}U fission.

[4] Test calculations such as this one are the primary reason why q'_{MC} is used to update q'_{relax} in addition to $q'_{\text{diffusion}}$.

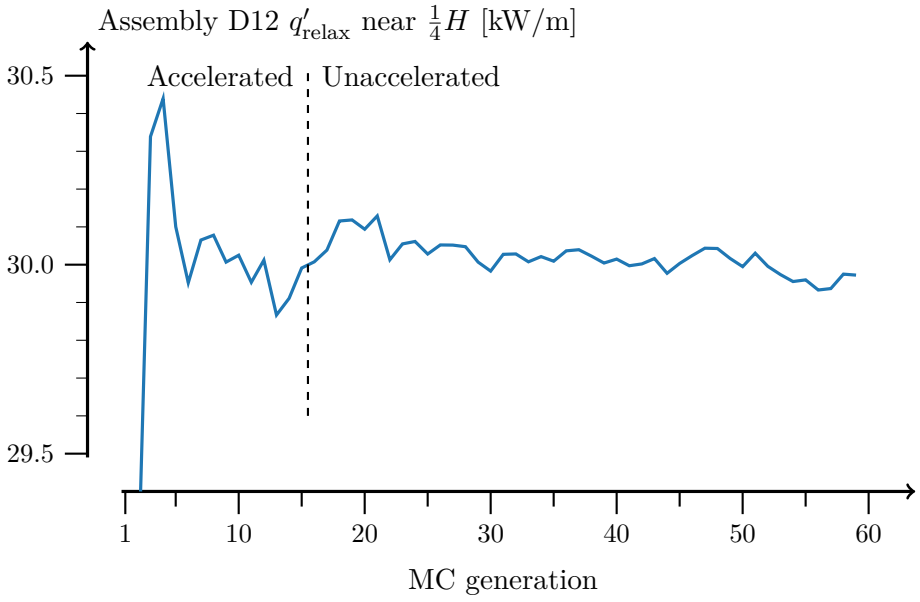


Figure 7-16: Behavior of a partially accelerated simulation. Acceleration is used for the first 15 generations, then MC alone is used without the CMFD diffusion surrogate. Values are taken from assembly D12 at $\frac{1}{4}H$ as a representative example. Note the narrow range used for the q' scale.

data which suggests that it is converged or at least nearly converged. This in turn suggests that the surrogate solver does not add any bias to the final result.

Note that autocorrelation might hide a small bias over the course of just 44 generations, but a large bias would likely show some trend. This assertion will be supported by the following section which shows measurable trends in unaccelerated simulations.

7.6 Convergence with and without acceleration

In an effort to quantify the gains offered by acceleration, another simulation was run without it. Figure 7-17 compares the convergence behavior of accelerated versus unaccelerated simulations for one representative region of the core. For the accelerated case, the figure shows two independent simulations with different random number seeds to give an idea of the repeatability and statistical uncertainty of the simulations.

Each accelerated simulation is run for 30 generations, and both appear to reach stationarity by about generation 6, consistent with the data presented previously. The unaccelerated simulation is limited to 60 generations to prevent excessive runtime, and it does not reach stationarity. However, the number of generations required to reach stationarity can be roughly estimated from the trend in the data.

Figure 7-18 expresses this same dataset in terms of a relative error, referenced against the converged value. The error is computed from the equation,

$$\text{rel err} = \left| \frac{q' - q'_{\text{avg}}}{q'_{\text{avg}}} \right|$$

where q'_{avg} is the average value from generation 15 onwards of the two accelerated simulations.

Figure 7-18 shows that the error error in the accelerated simulations is uniformly below 0.7% in the stationary region. Therefore, projecting the point at which the unaccelerated simulation reaches 0.7% error will provide a useful indication of the speedup offered by acceleration.

The error from the unaccelerated simulation appears to decrease linearly on this logarithmic plot (indicating exponential decay) from about generation 20 onwards. By fitting a line to this data and extrapolating, the unaccelerated simulation is estimated to reach 0.7% error after 110 generations. Thus a ratio of 110-to-6 can be used as an estimate of the

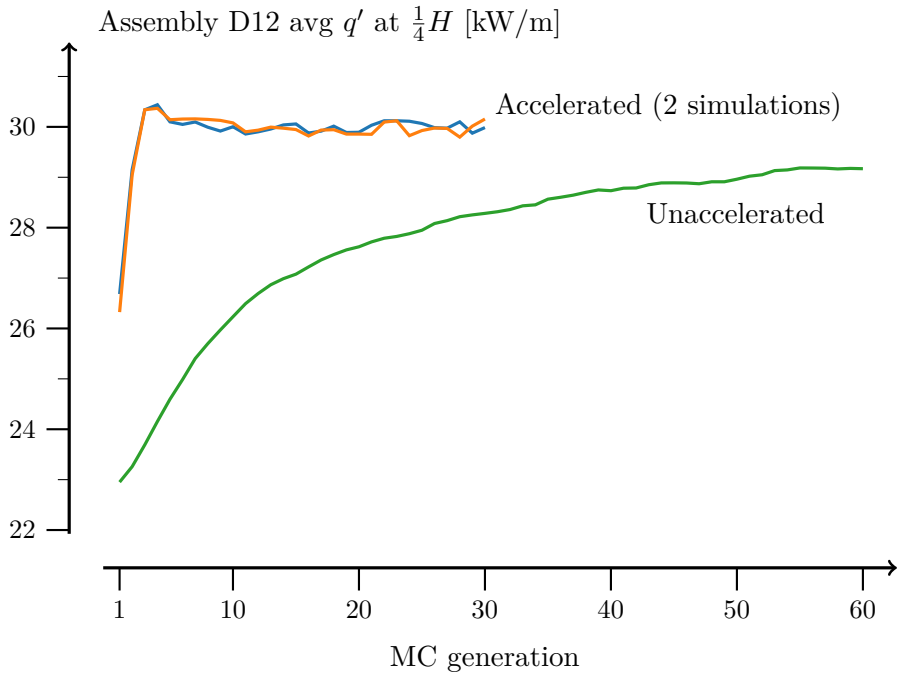


Figure 7-17: Convergence in a high-power region of the core. The linear heat rate at 1/4 of the active height averaged over fuel pins in assembly D12 is plotted. One unaccelerated simulation and two accelerated simulations with different seeds are shown.

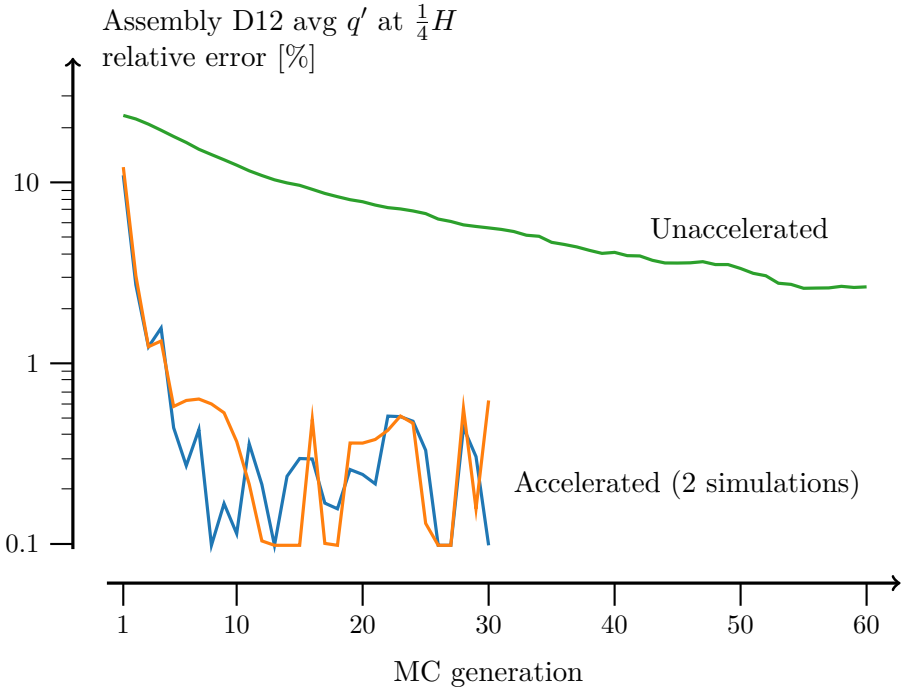


Figure 7-18: Relative error of Figure 7-17 data. Values below 0.1% are clipped.

number of generations needed to reach stationarity. This analysis was repeated for a few other assemblies and axial heights, and it consistently yielded similar numbers in the range of 80 to 110 generations for stationary.

Section 7.7 will discuss the runtime in detail, but for this discussion it is worth noting that on average, a simulation required 30 cpu core-hours per MC generation without acceleration and 45 core-hours per MC generation with acceleration. (This includes the cost of the heat transfer, fluids, and CMFD neutronics solvers as well as the MC solver; the runtime difference is mostly due to the extra tallies needed for the surrogate solver parameters.) With these values, the computational cost to reach stationarity can be estimated as $6 \times 45 = 270$ core-hours with acceleration and $110 \times 30 = 3300$ core-hours without.

7.7 Runtime breakdown

The simulations presented in this section were run on a computer cluster with Intel Xeon E5-2683 v4 processors. Each cluster node has two processor sockets, and each processor has 16 cores. These simulations used 6 nodes of the cluster for a total of 192 processor cores. The accelerated simulations with 30 total MC generations required 7.0 wall-clock hours, and the unaccelerated simulation with 60 generations required 11.9 wall-clock hours.

Table 7.1 shows the fraction of runtime spent in various components of the coupled solver. The bulk of the runtime is spent in Monte Carlo neutron transport regardless of whether or not acceleration is used. Acceleration introduces some additional cost to generate multigroup cross sections (MGXS in the table) and solve the CMFD diffusion problem, but these costs are small compared to the total runtime. The accelerated solver also makes use of many more fluid and heat transfer iterations per MC generation so the fraction of runtime spent in those solvers increases significantly.

Note that the accelerated solver requires more MC tallies so each MC generation is

Table 7.1: Runtime spent in various solver components with and without acceleration

Component	Accelerated	Unaccelerated
MC transport	78%	91%
Heat transfer	6%	0.6%
Fluid dynamics	1%	0.1%
MGXS	2%	-
CMFD	2%	-

also slower. Here it was found that that each MC generation runs about 24% slower with the tallies needed for the accelerated solver. Note that a solver for industrial use would likely need additional tallies for energy deposition and the non-fission heating which are not included here.

Another detail worth noting is that the Monte Carlo solver is the only solver which takes full advantage of the available cpu cores. Simulations were run with OpenMP shared-memory parallelism over the 16 cores in each socket and MPI distributed-memory parallelism over the 12 sockets. The fluid dynamics solver only uses OpenMP parallelism and so it uses just 16 cores. The same is true for the diffusion solver. The heat transfer solver is not parallelized and was run on a single core. Further parallelizing these solvers is of course possible, but was not considered worth the development time for this work.

Although the heat transfer, fluid dynamics, and CMFD solvers were not set up for distributed memory parallelism, significant effort was spent making them fast for single-core calculations. For example, using SIMD-friendly loops in the CMFD solver and implementing custom water property tables led to major single-core runtime gains.

Designing each of these solver components for speed without parallelism led to crucial usability gains which are not well captured by the raw numbers presented here. For example, fast heat-transfer and fluid dynamics solvers means that the output data files need only store the power distribution—temperature distributions can easily be reconstructed after-the-fact. To generate the each temperature plot presented in this thesis, the heat transfer and fluid dynamics solvers were run on a single core of a laptop computer using the power distributions computed on the cluster. For a fixed power distribution, solving the heat transfer and fluid dynamics problems on the quarter-core problem takes about 30 seconds. If CMFD parameters are saved as well, then fully-coupled simulations without MC can be run on a laptop which greatly speeds the process of development and debugging.

7.8 Uncertainty in the stationary solution

There are two different notions of convergence for this simulation: stationarity and stochastic uncertainty. Rigorously defining stationarity and devising efficient methods to measure it is a non-trivial task. Instead, a "you know it when you see it" approach is taken here; a simulation is considered stationary when there is no visible trend in any of the computed

quantities. From the analysis in this chapter, these simulations are considered stationary from about generation 6 onwards.

However, the solution still oscillates in the stationary region within some uncertainty bounds. Figure 7-15, for example, shows that assembly average q' values near $\frac{1}{4}H$ oscillate between ± 0.15 kW / m of the mean value.

The volume of a tally region sometimes plays a role in the uncertainty, and sometimes does not. The tally volume for assembly G08 is $2\times$ smaller than assemblies B13 and D12 (because G08 is cut by a boundary condition), but Figure 7-15 shows no visible difference in the uncertainties for these assemblies. In contrast, tallies over one axial region of one fuel pin or one instrument tube are small enough that the volume effect is significant. This can be seen by comparing the instrument fission rate tallies shown in Figures 7-4 and 7-3; here the instrument that is cut by the boundary condition shows more visible noise.

To further illuminate the noise at this fine scale, Figure 7-19 shows an example of the iteration-by-iteration change in the q' distribution of a single fuel pin. Values at each axial level can vary by as much as ± 1.5 kW / m in the stationary generations. These large changes in q' will come with correspondingly large changes in the fuel temperature and xenon concentration which underscores the importance of relaxation.

To reduce the noise and facilitate an analysis like the one shown in Figure 7-12, the

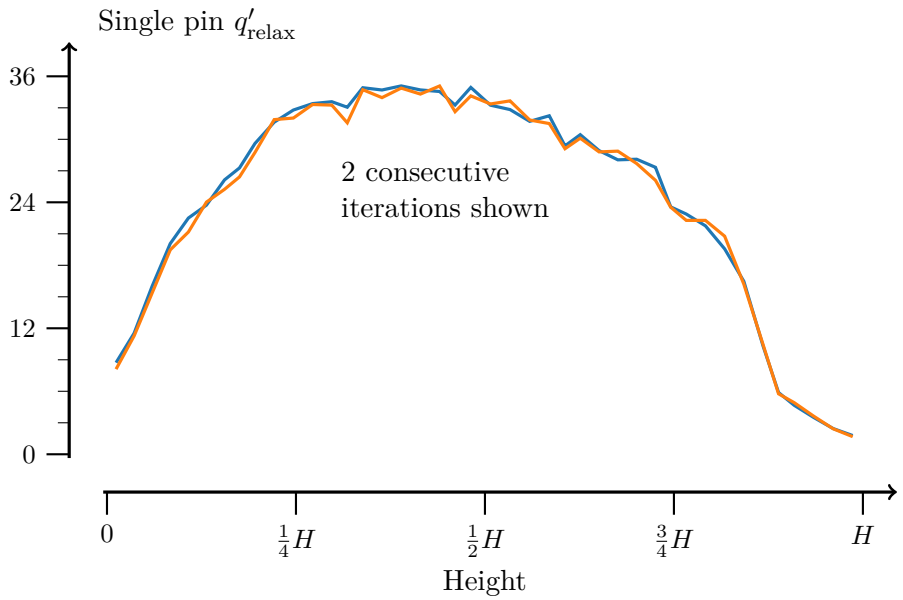


Figure 7-19: Linear heat rate for the hottest pin in assembly D12. The plotted lines show the values from 2 consecutive iterations taken from the stationary region of one simulation.

results should be averaged over multiple stationary generations and/or multiple independent simulations.

Another set of simulations was run to quantify the computational cost of very finely resolving these local uncertainties. This set consists of 10 independent simulations that each run for 15 iterations. The q'_{relax} distribution is averaged over iterations 6 through 15 of each simulation to give 10 different power distributions.

Because these 10 distributions are computed from independent simulations, the uncertainty can be quantified using Student's t-distribution. A mean and 95% confidence interval were computed for the q' value of each pin at each axial height. The distribution of these confidence intervals (relative to the local power) is shown in Figure 7-20.

Using these 10 simulations, the computed 95% confidence interval is less than 2% of the local power in 95% of the fine regions. These uncertainties can be roughly compared to Kelly et al. who also used MC to simulate a quarter-core of an equally-sized full-power reactor [3]. Note that they used a 49-cell axial mesh, slightly finer than the 42-cell axial mesh used here. Using one simulation with a total of 30 billion active neutrons (4 million per generation), they reduced the uncertainty in 95% of the regions to under 1%.

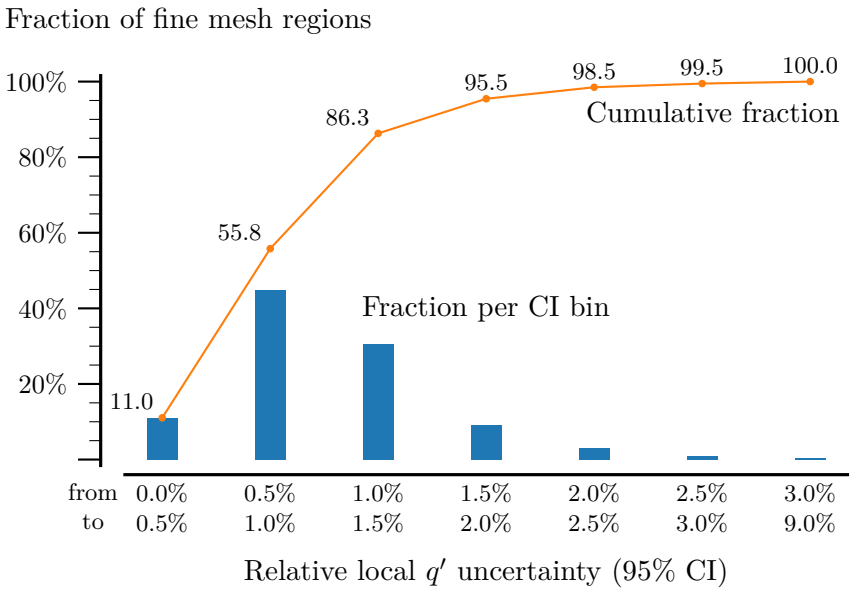


Figure 7-20: Distribution in the relative uncertainty of linear heat rate values. This data covers the 591 024 regions describing each axial level of each pin. Uncertainties are computed from 10 independent simulations. Each simulation uses 200 million neutrons per generation and results are averaged over generations 6 through 15.

Assuming that the uncertainty for this work scales with $\sqrt{N}$ where N is the number of independent simulations, then the same 1% error target used by Kelly et al. could probably be achieved with 40 of the 15-generation simulations. Note that this would entail averaging results over a total of 80 billion neutrons, a larger number than the 30 billion used by Kelly et al. Possibly, this is due to the use of independent simulations which leads to more accurate confidence intervals.

Despite the increased number of needed neutrons, the solver presented here still leads to a runtime performance gain over the Kelly et al. example. Each of the 15-generation simulations requires 690 cpu core-hours so the total cost of 40 simulations would be 28 000 core-hours which is 6× smaller than the 170 000 core-hours reported by Kelly et al. [3]. The calculation rate per-neutron is faster for this solver so the runtime difference must be due to the performance of the MC solver itself rather than performance gains offered by the surrogate solver. The reason for this runtime difference is unclear; it could be attributed to differences in cross section representation, ray tracing implementation, or a dozen other subtleties.

It should also be noted that 1% convergence on the local power density is likely a much tighter tolerance than needed for many analyses. For example, the 10 15-generation simulations do not significantly change the results shown in Figure 7-2 (which used 2 30-generation simulations).

7.9 Summary

This chapter has demonstrated the proposed MC-based multiphysics solver methodology on quarter-core full-power simulations of a large pressurized water reactor at the beginning-of-cycle. The calculated results match the experimental measurements relatively well, although some significant errors persist due to the simplifying approximations made for this work.

This chapter shows that measurable quantities (the fission detector profiles) and some safety-relevant quantities (the peak cladding surface temperature) can be resolved with a computational cost of 2 700 cpu core-hours. For high-precision work, it is predicted that 95% of the fine-mesh regions (42 axial cells and 1 cell per fuel pin) could be resolved to 1% uncertainty (95% confidence interval) with a cost of 28 000 core-hours. A caveat to these figures is that the runtime cost will likely be much larger with depleted fuel.

The acceleration procedure described in previous chapters contributes greatly to this runtime performance as only 6 MC generations are needed to reach stationarity as opposed to the 110 generations expected without acceleration.

Chapter 8

Conclusion

8.1 Review of this work

This thesis is aimed at reducing the cost of multiphysics reactor simulations using Monte Carlo neutron transport. To that end, this work builds upon the recently developed techniques for accelerating MC with a CMFD neutronics solver.

Several atypical neutronics practices are adopted in this thesis. These practices should not be considered in isolation as each builds upon the others. CMFD-based acceleration is used, and a source weight clipping procedure is introduced so that acceleration can be performed as early as the first MC generation. Furthermore, the CMFD solver is augmented with differential tallies and pin power reconstruction which enables bi-directional multiphysics coupling. Consequently, the CMFD solver with these augmentations can be used as an approximate surrogate for Monte Carlo, suitable for converging the multiphysics problem in addition to the neutronics eigenvalue problem.

Due to this efficient surrogate, very few MC generations are needed to reach stationarity with the multiphysics system. This in turn enables the use of MC simulations that use a very large number of particles per generation, a practice which is far more effective at improving tally accuracy than accumulating tallies over many generations. Likely, these high-accuracy tallies also contribute to the accuracy and stability of the coarse mesh solver. Note that these methods build upon one-another: an effective coarse mesh surrogate enables large MC generations which contributes to the effectiveness of the coarse mesh surrogate.

A parallel line of work in this thesis is a collection of software-focused improvements. A pair of ray tracing improvements for the OpenMC code are described: cell-based shared-

memory neighbor lists and geometry search partitioning. These ray tracing improvements help to maintain solver performance on geometries that are discretized for multiphysics purposes. Other improvements and best practices include: using collision-estimated tallies in place of tracklength-estimated, axially discretizing most MC geometry features to ensure short neighbor lists, using rectilinear tally meshes that conform to geometry features, and using a ragged coarse mesh that excludes low-flux, high-uncertainty regions.

Using these methods, quarter-core PWR multiphysics simulations can be run in a matter of hours on relatively small computer clusters. For example, a 7-hour calculation using 192 processor cores is sufficient to converge the quarter-core multiphysics problem and resolve important details such as simulated flux map measurements.

This is a significant reduction in computational cost over prior methods, and it makes Monte Carlo neutron transport a more viable tool for both research and industrial use. Because MC codes can relax many approximations used by deterministic solvers, this opens possibilities for more general and predictive nuclear reactor modeling and simulation tools.

A further finding of this work that is difficult to quantify, but should not be overlooked, is the utility of good software development practices. This work made extensive use of in-memory C-APIs for Python and its third-party packages which allows for both rapid software prototyping and fast solver runtime. Similarly, the clean C++ codebase and Python API of OpenMC enabled much of this work. This speaks to the value of projects such as the 2018 effort to translate the core of OpenMC from Fortran to C++, a project that was difficult and distracted from short-term research work but paid significant dividends for the methodology improvements presented in this thesis.

8.2 Future work

A priority for future extensions of this work is to account for isotopic depletion. Depletion calculations are necessary for the ultimate goal of creating useful tools for power reactor modeling, but it also contributes to a more immediate goal of making tools that can be compared against measured plant data (like the BEAVRS benchmark) for validation purposes.

It is also not obvious how best to incorporate depletion. This work found significant gains by performing multiphysics coupling after each MC generation, and there may be

analogous benefits to performing many small depletion steps that each occur after one MC generation. This would be counter to the established practice of accumulating tallies over many generations and then executing a large depletion time step.

Another priority for future work would be to improve upon the fluid dynamics methods used here. It is not clear if the approximate methods used in this work introduce a significant error for the BEAVRS benchmark calculations (such an error would be hidden by the larger error of neglecting depletion effects), but methods which resolve finer fluid details are desirable for the same reasons that it is desirable to use high-fidelity neutronics methods. Using higher-fidelity fluid dynamics methods will no doubt prove difficult because—as discussed in Chapter 2—the common approach of RANS CFD is more computationally expensive than MC neutronics (based on the observed performance of MC in this work) and it introduces modeling challenges for two-phase fluids and anisotropic eddy viscosity.

Furthermore, there is some room for improvement of the differential tally methods used here. No attempt was made in this work to compute derivatives of the $\widehat{D}$ factors, but these likely have a significant impact on the solver. As seen in Chapter 4, there are also noticeable errors in the temperature derivatives for thermal group cross sections likely due to OpenMC's approximation of the scattering kernel derivative. Improving those derivatives may lead to faster solver convergence. However, note that improvements to the derivatives and their usage will only reduce the number of MC generations required to reach stationarity, but the solver runtime is likely dominated by the stationary generations for most applications of interest. Consequently, research aimed at reducing the cost of or increasing the precision of the stationary generations will likely prove more fruitful.

Appendix A

Subchannel Equation Derivations

The numerical system used here to describe the coolant thermal-hydraulics is derived from the conservation of mass, momentum, and energy with a finite volume discretization. Figure A-1 shows a prototypical volume used by the solver along with the labeling convention for the fluid interface surfaces.

The particular volume shown in Figure A-1 sits between four fuel pins of a rectangular lattice, but the derivations in this appendix are general to allow for subchannels at the core boundary (which border fewer than four pins) or subchannels in a non-rectangular lattice (such as a VVER or CANDU reactor).

However, for convenience it is assumed that the volume has a top and bottom face which are perpendicular to the z-axis. The volume Ω_{ik} sits between $z = z_k$ and $z = z_{k+1}$. The index, k, specifies the axial location and the other index, i, is for the radial location. The index, j, indicates a neighboring volume.

It will also be assumed that none of the sides of the volume slope inwards or outwards. This gives a uniform cross-section which does not allow modeling of effects like rod bowing or ballooning. However, this assumption is made late in the derivation which makes it easier to relax in future work. For the same reason, the assumption of steady state is left until near the end of the derivations.

A.1 General conservation equation

The derivations here closely follow those found in Chapter 4 of Todreas and Kazimi [67].

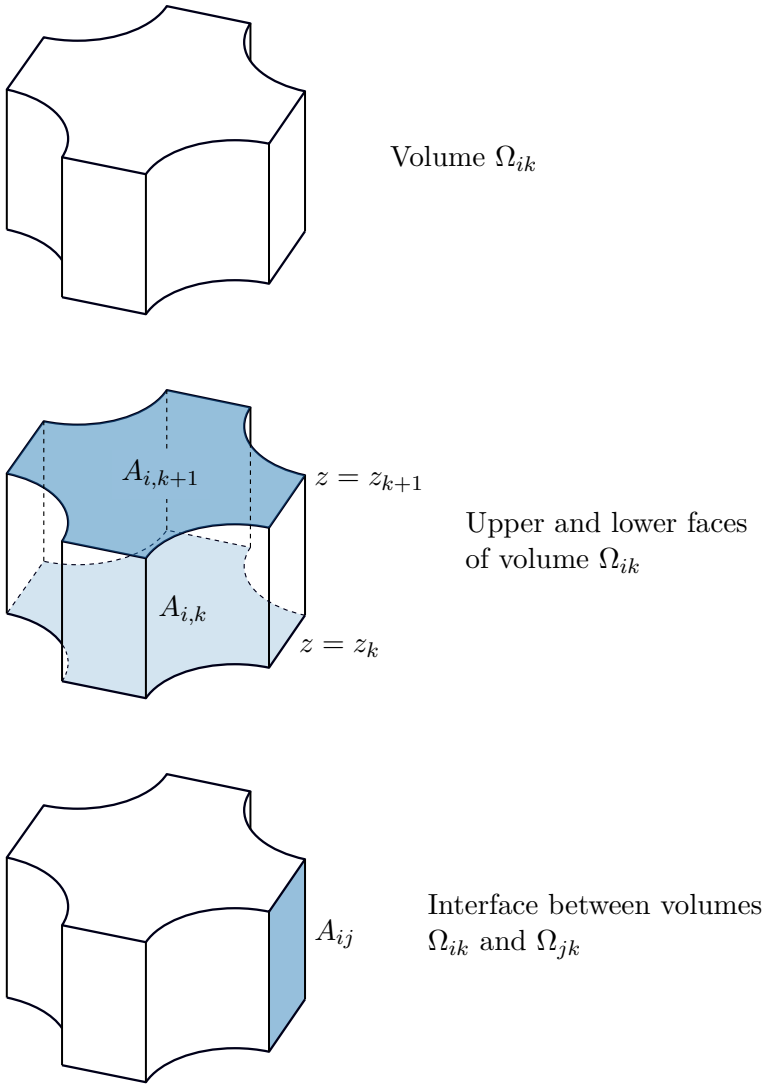


Figure A-1: Isometric views of an example subchannel volume, $\Omega_{i,k}$, and the labels used for various surfaces.

All conservation equations considered here take the form,

$$\frac{\partial \rho\phi}{\partial t} + \nabla \cdot \rho\phi\boldsymbol{v} = S \tag{A.1}$$

where ϕ is some conserved quantity per unit mass, ρ is the fluid density, $\boldsymbol{v}$ is the fluid velocity vector, and S accounts for sources and losses.

Three conservation equations will be considered here for mass, axial momentum, and enthalpy. For each of these, ϕ should be replaced with,

$\phi \leftarrow 1$	Conservation of mass
$\phi \leftarrow v_z$	Conservation of z-momentum
$\phi \leftarrow h$	Conservation of enthalpy

where v_z is the z-component of the fluid velocity and h is the specific enthalpy. Each conservation equation also has a special form of the source, S.

To form a numerical system, the conservation equation is first integrated over a finite subchannel volume, $\Omega_{i,k}$,

$$\iiint_{\Omega_{i,k}} \frac{\partial \rho\phi}{\partial t} \mathrm{d}\boldsymbol{r} + \iiint_{\Omega_{i,k}} \nabla \cdot \rho\phi\boldsymbol{v} \mathrm{d}\boldsymbol{r} = \iiint_{\Omega_{i,k}} S \mathrm{d}\boldsymbol{r} \tag{A.2}$$

where the subscript i indicates a radial position and k indicates an axial position in the model.

Next, the divergence theorem is applied to the second term which gives,

$$\iiint_{\Omega_{i,k}} \frac{\partial \rho\phi}{\partial t} \mathrm{d}\boldsymbol{r} + \iint_{\partial\Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho\phi\boldsymbol{v} \mathrm{d}\boldsymbol{r} = \iiint_{\Omega_{i,k}} S \mathrm{d}\boldsymbol{r} \tag{A.3}$$

where $\partial\Omega_{i,k}$ is the surface enclosing the volume $\Omega_{i,k}$ and $\hat{\boldsymbol{n}}$ is the normal vector of that surface.

Assuming that the top and bottom faces of each volume are perpendicular to the z-axis allows the integration over z to be considered separately from the other axes,

$$\int_{\Delta z_k} \iint_{A_i} \frac{\partial \rho\phi}{\partial t} \mathrm{d}x\mathrm{d}y\mathrm{d}z + \iint_{\partial\Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho\phi\boldsymbol{v} \mathrm{d}\boldsymbol{r} = \int_{\Delta z_k} \iint_{A_i} S \mathrm{d}x\mathrm{d}y\mathrm{d}z \tag{A.4}$$

It will be convenient to express the integrals over the xy-plane in terms of area averaged quantities. Here angle brackets are used to indicate averages as defined by,

$$\langle \psi \rangle_i = \frac{1}{A_i} \iint_{A_i} \psi \mathrm{d}x\mathrm{d}y$$

where ψ is some arbitrary quantity and A_i is the cross-sectional area of the subchannel.

The conservation equation can be written using these area-averaged quantities as,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho\phi}{\partial t} \right\rangle_i \mathrm{d}z + \iint_{\partial\Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho\phi \boldsymbol{v} \mathrm{d}\boldsymbol{r} = \int_{\Delta z_k} A_i \langle S \rangle_i \, \mathrm{d}z \tag{A.5}$$

A.2 Specific conservation equations

A.2.1 Mass

Mass is the simplest of the conservation equations. The variable ψ is unity and there are no source or loss terms,

$$\psi \leftarrow 1 \qquad S \leftarrow 0$$

Substituting this into Equation A.5 gives,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho}{\partial t} \right\rangle_i \mathrm{d}z + \iint_{\partial\Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho \boldsymbol{v} \mathrm{d}\boldsymbol{r} = 0 \tag{A.6}$$

A.2.2 Axial momentum

For the axial momentum, the specific conservation terms are,

$$\psi \leftarrow v_z \qquad S \leftarrow -\frac{\partial P}{\partial z} + \nabla \cdot \boldsymbol{\tau_z} - \rho g$$

where v_z is the z-component of the velocity, P is the fluid pressure, $\boldsymbol{\tau_z}$ is a vector of the normal and shear stresses that act in the z-direction, and g is the acceleration due to gravity (assumed to point in the negative z direction).

Substituting this into Equation A.5 gives,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho v_z}{\partial t} \right\rangle_i \mathrm{d}z + \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho v_z \boldsymbol{v} \mathrm{d}\boldsymbol{r} = -\int_{\Delta z_k} A_i \left\langle \frac{\partial P}{\partial z} \right\rangle_i \mathrm{d}z + \int_{\Delta z_k} A_i \left\langle \nabla \cdot \boldsymbol{\tau_z} \right\rangle_i \mathrm{d}z - g \int_{\Delta z_k} A_i \left\langle \rho \right\rangle \mathrm{d}z \quad \text{(A.7)}$$

Next, it is approximated that the shear due to the solid walls surrounding the volume is much larger than the shear due to adjacent fluid volumes. The stress is then modeled as,

$$\left\langle \nabla \cdot \boldsymbol{\tau_z} \right\rangle_i = -\left(\frac{f}{D_e} + K \delta_K \right) \frac{\left\langle \rho \right\rangle_i \left\langle v_z \right\rangle_i^2}{2} \quad \text{(A.8)}$$

where f is the Darcy friction factor, D_e is the equivalent diameter of the channel, K is the form loss coefficient, and δ_K is a delta function which indicates that the form loss should only be applied where there is a restriction in the flow. Empirical correlations will be used for the values of f and K.

The conservation of axial momentum is now,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho v_z}{\partial t} \right\rangle_i \mathrm{d}z + \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho v_z \boldsymbol{v} \mathrm{d}\boldsymbol{r} = -\int_{\Delta z_k} A_i \left\langle \frac{\partial P}{\partial z} \right\rangle_i \mathrm{d}z - \int_{\Delta z_k} A_i \left(\frac{f}{D_e} + K \delta_K \right) \frac{\left\langle \rho \right\rangle_i \left\langle v_z \right\rangle_i^2}{2} \mathrm{d}z - g \int_{\Delta z_k} A_i \left\langle \rho \right\rangle \mathrm{d}z \quad \text{(A.9)}$$

A.2.3 Energy

The conservation of energy (in terms of enthalpy) can be found with the specific terms,

$$\psi \leftarrow h \qquad S \leftarrow -\nabla \cdot \boldsymbol{q}'' + q''' + \frac{\partial P}{\partial t} + \boldsymbol{v} \cdot \nabla P + \Phi$$

where $\boldsymbol{q}''$ is the heat flux due to thermal conduction, q''' is the volumetric heat generation (e.g. due to neutron scattering), and Φ is heat generated from internal friction.

For the core of a reactor, the heat flux from the fuel pins is much larger than the generation of heat via friction or pressure work so those terms will be neglected. Removing those terms and substituting into Equation A.5 gives,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho h}{\partial t} \right\rangle_i \mathrm{d}z + \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho h \boldsymbol{v} \mathrm{d}\boldsymbol{r} = -\int_{\Delta z_k} A_i \left\langle \nabla \cdot \boldsymbol{q}'' \right\rangle_i \mathrm{d}z + \int_{\Delta z_k} A_i \left\langle q''' \right\rangle_i \mathrm{d}z \quad \text{(A.10)}$$

The divergence theorem can be applied to the heat flux term to find,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho h}{\partial t} \right\rangle_i \mathrm{d}z + \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho h \boldsymbol{v} \mathrm{d}\boldsymbol{r} = - \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \boldsymbol{q}'' \mathrm{d}\boldsymbol{r} + \int_{\Delta z_k} A_i \left\langle q''' \right\rangle_i \mathrm{d}z \quad \text{(A.11)}$$

A.3 Crossflow terms

Equation A.5, the general conservation equation, includes a term on the left-hand-side for the net flux of the conserved quantity. The flux term can be expanded as,

$$\iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \rho \phi \boldsymbol{v} \mathrm{d}\boldsymbol{r} = A_i \left\langle \rho \phi v_z \right\rangle_{i,k+1} - A_i \left\langle \rho \phi v_z \right\rangle_{i,k} + \sum_j \iint_{A_{ijk}} \hat{\boldsymbol{n}} \cdot \rho \phi \boldsymbol{v} \mathrm{d}\boldsymbol{r} \quad \text{(A.12)}$$

where the subscript $k+1$ indicates an average over the upper face of the volume, k indicates the lower face, and A_{ijk} is the interface between channels i and j at axial level k. Each term on the right-hand-side of this equation gives the flux over one of the faces.

It is helpful to handle the integration over z separately from the other axes. Consequently, the variables W, M, and H are introduced which are defined as,

$$\int_{\Delta z_k} W_{ij} \mathrm{d}z = \iint_{A_{ij,k}} \hat{\boldsymbol{n}} \cdot \rho \boldsymbol{v} \mathrm{d}\boldsymbol{r} \quad \text{(A.13)}$$

$$\int_{\Delta z_k} M_{ij} \mathrm{d}z = \iint_{A_{ij,k}} \hat{\boldsymbol{n}} \cdot \rho v_z \boldsymbol{v} \mathrm{d}\boldsymbol{r} \quad \text{(A.14)}$$

$$\int_{\Delta z_k} H_{ij} \mathrm{d}z = \iint_{A_{ij,k}} \hat{\boldsymbol{n}} \cdot \rho h \boldsymbol{v} \mathrm{d}\boldsymbol{r} \quad \text{(A.15)}$$

These variables describe the amount of mass, momentum, and enthalpy transferred per unit length of the subchannel.

It is important to consider the impact of turbulence on the surface fluxes. The general integrand can be expanded with the notation of Reynolds decomposition as,

$$\rho \phi \boldsymbol{v} = \left(\overline{\rho \boldsymbol{v}} + (\rho \boldsymbol{v})' \right) \left(\overline{\phi} + \phi' \right) \quad \text{(A.16)}$$

where overlines indicate the component of the variable that is steady or at least changing slowly in time and tic marks indicate the remaining fluctuating component.

The steady part of this integrand is,

$$\overline{\rho\phi\boldsymbol{v}} = \underbrace{\overline{\rho\boldsymbol{v}}\,\overline{\phi}}_{\text{diversion}} + \underbrace{\overline{(\rho\boldsymbol{v})'\,\phi'}}_{\text{turbulent}} \tag{A.17}$$

In subchannel literature, the first term on the right is frequently referred to as the diversion term and the second as the turbulent diffusion term. The turbulent term accounts for the covariance between $(\rho\boldsymbol{v})'$ and ϕ'.

Applying this Reynolds averaging to the W, M, and H variables leads to,

$$\overline{W}_{ij} = W_{ij,\text{div}} \tag{A.18}$$

$$\overline{M}_{ij} = M_{ij,\text{div}} + M_{ij,\text{turb}} \tag{A.19}$$

$$\overline{H}_{ij} = H_{ij,\text{div}} + H_{ij,\text{turb}} \tag{A.20}$$

Note that there is no turbulent term for mass transfer since $\phi \leftarrow 1$. Subchannel literature frequently uses the W' variable, but it does not mean the value of the fluctuating term as would be expected by the Reynolds decomposition conventions. Instead it refers to the magnitude of the fluctuating term. To avoid confusion, it is not used here.

From here on, the fluctuating components of each variable will not be considered explicitly and every variable should be assumed to represent the steady component. The overlines are omitted as they clutter the equations.

Substituting the new flux notation into Equation A.6 gives the conservation of mass as,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho}{\partial t} \right\rangle_i \mathrm{d}z + A_i \left\langle \rho v_z \right\rangle_{i,k+1} - A_i \left\langle \rho v_z \right\rangle_{i,k} + \sum_j \int_{\Delta z_k} W_{ij} \mathrm{d}z = 0 \tag{A.21}$$

Similarly for Equation A.9 and momentum,

$$\begin{aligned} &\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho v_z}{\partial t} \right\rangle_i \mathrm{d}z + A_i \left\langle \rho v_z^2 \right\rangle_{i,k+1} - A_i \left\langle \rho v_z^2 \right\rangle_{i,k} + \sum_j \int_{\Delta z_k} M_{ij} \mathrm{d}z \\ &\quad = -\int_{\Delta z_k} A_i \left\langle \frac{\partial P}{\partial z} \right\rangle_i \mathrm{d}z - \int_{\Delta z_k} A_i \left(\frac{f}{D_e} + K\delta_K \right) \frac{\langle \rho \rangle_i \langle v_z \rangle_i^2}{2} \mathrm{d}z - g \int_{\Delta z_k} A_i \langle \rho \rangle \, \mathrm{d}z \end{aligned} \tag{A.22}$$

And Equation A.11 for energy becomes,

$$\int_{\Delta z_k} A_i \left\langle \frac{\partial \rho h}{\partial t} \right\rangle_i \mathrm{d}z + A_i \left\langle \rho h v_z \right\rangle_{i,k+1} - A_i \left\langle \rho h v_z \right\rangle_{i,k} + \sum_j \int_{\Delta z_k} H_{ij} \mathrm{d}z = - \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \boldsymbol{q}'' \mathrm{d}\boldsymbol{r} + \int_{\Delta z_k} A_i \left\langle q''' \right\rangle_i \mathrm{d}z \quad \text{(A.23)}$$

A.4 Simplification

At this point several approximations and simplifications will be applied. Steady-state flow is assumed which removes all time derivative terms. It is also assumed that the cross section, A_i, does not vary with z which allows it to be factored outside of integrals over z.

With these simplifications, the conservation of mass from Equation A.21 can be rewritten as,

$$\left\langle \rho v_z \right\rangle_{i,k+1} - \left\langle \rho v_z \right\rangle_{i,k} + \frac{1}{A_i} \sum_j \int_{\Delta z_k} W_{ij} \mathrm{d}z = 0 \quad \text{(A.24)}$$

Similarly, the conservation of momentum from Equation A.22 becomes,

$$\left\langle \rho v_z^2 \right\rangle_{i,k+1} - \left\langle \rho v_z^2 \right\rangle_{i,k} + \frac{1}{A_i} \sum_j \int_{\Delta z_k} M_{ij} \mathrm{d}z = - \int_{\Delta z_k} \left\langle \frac{\partial P}{\partial z} \right\rangle_i \mathrm{d}z - \int_{\Delta z_k} \left(\frac{f}{D_e} + K \delta_K \right) \frac{\left\langle \rho \right\rangle_i \left\langle v_z \right\rangle_i^2}{2} \mathrm{d}z - g \int_{\Delta z_k} \left\langle \rho \right\rangle \mathrm{d}z \quad \text{(A.25)}$$

The fundamental theorem of calculus may now be applied to the pressure derivative term to find,

$$\left\langle \rho v_z^2 \right\rangle_{i,k+1} - \left\langle \rho v_z^2 \right\rangle_{i,k} + \frac{1}{A_i} \sum_j \int_{\Delta z_k} M_{ij} \mathrm{d}z = - \left\langle P \right\rangle_{i,k+1} + \left\langle P \right\rangle_{i,k} - \int_{\Delta z_k} \left(\frac{f}{D_e} + K \delta_K \right) \frac{\left\langle \rho \right\rangle_i \left\langle v_z \right\rangle_i^2}{2} \mathrm{d}z - g \int_{\Delta z_k} \left\langle \rho \right\rangle \mathrm{d}z \quad \text{(A.26)}$$

The conservation of enthalpy, Equation A.23, can be simplified to,

$$\left\langle \rho h v_z \right\rangle_{i,k+1} - \left\langle \rho h v_z \right\rangle_{i,k} + \frac{1}{A_i} \sum_j \int_{\Delta z_k} H_{ij} \mathrm{d}z = -\frac{1}{A_i} \iint_{\partial \Omega_{i,k}} \hat{\boldsymbol{n}} \cdot \boldsymbol{q}'' \mathrm{d}\boldsymbol{r} + \int_{\Delta z_k} \left\langle q''' \right\rangle_i \mathrm{d}z \quad \text{(A.27)}$$

It will also be assumed that all of the energy generated by nuclear reactions is deposited

directly in the fuel. This essentially moves energy from the volumetric q''' term to the q'' surface flux term so the q''' term is removed. Heat conduction within the coolant is assumed to be small relative to the convective transfer so the integral over q'' will be replaced with q_{wall} to indicate the integrated heat flux from the solid walls surrounding the volume,

$$\langle \rho h v_z \rangle_{i,k+1} - \langle \rho h v_z \rangle_{i,k} + \frac{1}{A_i} \sum_j \int_{\Delta z_k} H_{ij} \mathrm{d}z = \frac{1}{A_i} q_{\text{wall},i,k} \tag{A.28}$$

A.5 Upwind discretization

A simple zeroth-order accurate discretization scheme is now applied in order to evaluate the integrals. For example, the integration of the mass exchange becomes,

$$\int_{\Delta z_k} W_{ij} \mathrm{d}z \approx \Delta z_k W_{ij,k} \tag{A.29}$$

where $W_{ij,k}$ is the value of W_{ij} evaluated at $z = z_k$, i.e. the upwind face of the subchannel volume.

With this discretization, the conservation of mass may be solved for the downwind mass flux as,

$$\langle \rho v_z \rangle_{i,k+1} = \langle \rho v_z \rangle_{i,k} - \frac{\Delta z_k}{A_i} \sum_j W_{ij,k} \tag{A.30}$$

This choice of discretization is advantageous because all variables on the right-hand-side of Equation A.30 are known when performing a sweep of the solution from inlet to outlet. Given fixed values at the inlet, no iteration is required to evaluate the mass fluxes via Equation A.30.

Similarly discretizing the integrals for the conservation of momentum and solving for the downwind pressure gives,

$$\begin{aligned} \langle P \rangle_{i,k+1} = \langle P \rangle_{i,k} - \left\langle \rho v_z^2 \right\rangle_{i,k+1} + \left\langle \rho v_z^2 \right\rangle_{i,k} - \frac{\Delta z_k}{A_i} \sum_j M_{ij,k} \\ - \left(\frac{f_k \Delta z_k}{D_e} + K_k \right) \frac{\langle \rho \rangle_{i,k} \langle v_z \rangle_{i,k}^2}{2} - g \Delta z_k \langle \rho \rangle_k \end{aligned} \tag{A.31}$$

The acceleration terms, those involving v^2, provide some difficulty as the outlet momentum flux, $\langle \rho v_z^2 \rangle_{i,k+1}$, cannot be known without some iteration. However the contribution of this term is small and the velocity will not vary greatly over each element so this equation

is replaced with the approximation,

$$\langle P\rangle_{i,k+1} = \langle P\rangle_{i,k} - \left(\langle \rho v_z\rangle_{i,k+1} - \langle \rho v_z\rangle_{i,k}\right)\langle v_z\rangle_{i,k} - \frac{\Delta z_k}{A_i}\sum_j M_{ij,k} - \frac{1}{A_i}\left(\frac{f_k \Delta z_k}{D_e} + K_k\right)\frac{\langle\rho\rangle_{i,k}\langle v_z\rangle_{i,k}^2}{2} - g\Delta z_k \langle\rho\rangle_k \quad \text{(A.32)}$$

Using the upwind discretization, the downwind enthalpy flux can be solved for as,

$$\langle \rho h v_z\rangle_{i,k+1} = \langle \rho h v_z\rangle_{i,k} - \frac{\Delta z_k}{A_i}\sum_j H_{ij,k} + \frac{1}{A_i} q_{\text{wall},i,k} \quad \text{(A.33)}$$

The average specific enthalpy may then be computed by approximating the enthalpy and mass flux as separable,

$$\langle h\rangle_{i,k+1} = \frac{\langle \rho h v_z\rangle_{i,k+1}}{\langle \rho v_z\rangle_{i,k+1}} \quad \text{(A.34)}$$

From the specific enthalpy and pressure, the equations of state can be solved for the fluid temperature and density. The velocity can then be approximated as,

$$\langle v_z\rangle_{i,k+1} = \frac{\langle \rho v_z\rangle_{i,k+1}}{\langle\rho\rangle_{i,k+1}} \quad \text{(A.35)}$$

At this point, all equations for the necessary values at $z = z_{k+1}$ have been specified. Given the inlet mass flux, pressure, and temperature the solver can sweep upwards through the mesh and to the outlet using these equations to compute the fluid properties.

A.6 Constitutive relations

The friction factor is computed from the McAdams relation for smooth tubes with Reynolds number between 3×10^4 and 10^6 (which can be found in chapter 9 of [67]),

$$f = 0.184\text{Re}^{-0.20} \quad \text{(A.36)}$$

For the general case, subchannel codes require models for the diversion crossflow terms, $W_{ij,\text{div}}$, $M_{ij,\text{div}}$, and $H_{ij,\text{div}}$, but here it is assumed that these terms are negligible. More discussion of this can be found in Chapter 2. See [68, 69, 70] for modeling of these terms.

The turbulent crossflow terms are computed from,

$$H_{ij,\text{turb}} = -\beta s \frac{G_i A_i + G_j A_j}{A_i + A_j} (h_i - h_j) \tag{A.37}$$

$$M_{ij,\text{turb}} = -\beta s \frac{G_i A_i + G_j A_j}{A_i + A_j} (v_{z,i} - v_{z,j}) \tag{A.38}$$

as discussed in Chapter 2.

Appendix B

Heat Transfer Equation Derivations

B.1 Discretization of the heat equation

The general heat equation can be written as,

$$\rho c_p \frac{\partial T}{\partial t} + \nabla \cdot \boldsymbol{q}'' - q''' = 0 \tag{B.1}$$

where ρ is the material density, c_p is the specific heat capacity, T is the temperature, $\boldsymbol{q}''$ is the heat flux vector, and q''' is the volumetric heat generation.

Expressing Equation B.1 in cylindrical coordinates and neglecting the terms for axial heat transfer, azimuthal heat transfer, and the time derivative gives,

$$\frac{1}{r}\frac{\partial}{\partial r}\left(rq''\right) - q''' = 0 \tag{B.2}$$

where that q'' is now a scalar value—the radial component of $\boldsymbol{q}''$.

Radial derivatives are approximated via a linear finite difference,

$$\left.\frac{\partial y(r)}{\partial r}\right|_{r_i} \approx \frac{y(r_{i+1/2}) - y(r_{i-1/2})}{\Delta r_i}$$

where $\Delta r_i = r_{i+1/2} - r_{i-1/2}$.

Applying this finite difference scheme to Equation B.2 gives,

$$\frac{r_{i+1/2}}{r_i \Delta r_i} q''_{i+1/2} - \frac{r_{i-1/2}}{r_i \Delta r_i} q''_{i-1/2} - q'''_i = 0 \tag{B.3}$$

B.2 Heat flux models

B.2.1 Simple conduction

Inside the fuel and cladding, heat flux can be modeled by Fourier's law of conduction. In the general 3D case,

$$\mathbf{q}'' = -k \nabla T$$

Simplifying to the conduction only in the radial direction of cylinderical coordinates gives,

$$q'' = -k \frac{\partial T}{\partial r}$$

Then discretizing for the outer half of ring i gives,

$$q''_{i+1/2} = -k_i \frac{T_{i+1/2} - T_i}{r_{i+1/2} - r_i}$$

$$q''_{i+1/2} = -2k_i \frac{T_{i+1/2} - T_i}{\Delta r_i}$$

$$T_{i+1/2} = T_i - \frac{q''_{i+1/2} \Delta r_i}{2k_i} \tag{B.4}$$

Doing the same for the inner half of ring $i+1$ gives,

$$q''_{i+1/2} = -2k_{i+1} \frac{T_{i+1} - T_{i+1/2}}{\Delta r_{i+1}}$$

$$T_{i+1/2} = T_{i+1} + \frac{q''_{i+1/2} \Delta r_{i+1}}{2k_{i+1}} \tag{B.5}$$

Enforcing continuity for $T_{i+1/2}$ from Equations B.4 and B.5,

$$T_i - \frac{q''_{i+1/2} \Delta r_i}{2k_i} = T_{i+1} + \frac{q''_{i+1/2} \Delta r_{i+1}}{2k_{i+1}}$$

$$-q''_{i+1/2} \left(\frac{\Delta r_i}{2k_i} + \frac{\Delta r_{i+1}}{2k_{i+1}} \right) = T_{i+1} - T_i$$

$$q''_{i+1/2} = -\frac{2k_i k_{i+1}}{\Delta r_i k_{i+1} + \Delta r_{i+1} k_i} \left(T_{i+1} - T_i\right) \tag{B.6}$$

Introducing the new variable, $\tilde{k}_{i,i+1}$, shortens Equation B.6 to

$$q''_{i+1/2} = -\tilde{k}_{i,i+1} \left(T_{i+1} - T_i\right) \tag{B.7}$$

where

$$\tilde{k}_{i,i+1} = \frac{2k_i k_{i+1}}{\Delta r_i k_{i+1} + \Delta r_{i+1} k_i}$$

B.2.2 Gap conduction

Heat transfer across the fuel-cladding gap is usually modeled with an equation of the form,

$$q''_g = h_g \left(T_{\text{fo}} - T_{\text{ci}}\right) \tag{B.8}$$

where T_{fo} is the temperature on the outer edge of the fuel and T_{ci} is the temperature on the inner wall of the cladding.

Let ring j be the last discretization ring in the fuel. There are no rings for the gap so ring $j+1$ is the first cladding ring. Applying Equations B.4 and B.5 to $i = j$ where $T_{j+1/2} = T_{\text{fo}}$ and $T_{j+1-1/2} = T_{\text{ci}}$,

$$T_{\text{fo}} = T_j - \frac{q''_{\text{fo}} \Delta r_j}{2k_j}$$

$$T_{\text{ci}} = T_{j+1} + \frac{q''_{\text{ci}} \Delta r_{j+1}}{2k_{j+1}} \tag{B.9}$$

Note that with an open gap, q''_{fo} and q''_{ci} will differ because they are at different radii. Assuming there is no significant heat generation within the gap, then the heat fluxes can be related geometrically,

$$q''_{\text{fo}} = q''_{\text{ci}} \frac{r_{\text{ci}}}{r_{\text{fo}}}$$

$$T_{\text{fo}} = T_j - \frac{q''_{\text{ci}} \Delta r_j}{2k_j} \frac{r_{\text{ci}}}{r_{\text{fo}}} \tag{B.10}$$

Here it is assumed that $q''_g = q''_{\text{ci}}$. This assumption may need to be revisited. Substituting Equations B.9 and B.10 into Equation B.8 gives,

$$q''_{\text{ci}} = h_g \left(T_j - \frac{q''_{\text{ci}} \Delta r_j}{2k_j} \frac{r_{\text{ci}}}{r_{\text{fo}}} - T_{j+1} - \frac{q''_{\text{ci}} \Delta r_{j+1}}{2k_{j+1}} \right)$$

$$q''_{\text{ci}} \left(\frac{1}{h_g} + \frac{\Delta r_j r_{\text{ci}}}{2k_j r_{\text{fo}}} + \frac{\Delta r_{j+1}}{2k_{j+1}} \right) = T_j - T_{j+1}$$

Or in a form similar to Equation B.7

$$q''_{\text{ci}} = \tilde{k}_g \left(T_j - T_{j+1} \right)$$

where

$$\tilde{k}_g = \frac{1}{\frac{1}{h_g} + \frac{\Delta r_j r_{\text{ci}}}{2k_j r_{\text{fo}}} + \frac{\Delta r_{j+1}}{2k_{j+1}}}$$

B.2.3 Clad-coolant Dirichlet BC

Heat transfer across the cladding-coolant boundary is usually modeled in the form,

$$q''_{\text{co}} = h_{\text{conv}} \left(T_{\text{co}} - T_b \right) + h_{\text{nb}} \left(T_{\text{co}} - T_{\text{sat}} \right) \tag{B.11}$$

where h_{conv} is a convective (single phase) heat transfer coefficient, h_{nb} is a nucleate boiling heat transfer coefficient, T_{co} is the temperature on the outer wall of the cladding, T_b is the bulk temperature of the coolant, and T_{sat} is the saturation temperature of the coolant.

Solving for T_{co} gives,

$$T_{\text{co}} = \frac{q''_{\text{co}} + h_{\text{conv}} T_b + h_{\text{nb}} T_{\text{sat}}}{h_{\text{conv}} + h_{\text{nb}}} \tag{B.12}$$

Using this expression for T_{co}, the heat transfer system can be solved with a Dirichlet boundary condition. Evaluating Equation B.4 at the outermost cladding ring, $i = N - 1$, gives,

$$T_{N-1} = T_{\text{co}} - \frac{q''_{\text{co}} \Delta r_{N-1}}{2k_{N-1}} \tag{B.13}$$

B.2.4 Clad-coolant convective BC

The Dirichlet BC works fine in steady-state calculations, but sometimes the performance of a transient solver can be improved by using a BC where T_{co} is not fixed. The following convective BC can be used in such cases.

Equation B.11 can be expressed as,

$$q''_{\text{co}} = \left(h_{\text{conv}} + h_{\text{nb}} \right) T_{\text{co}} - h_{\text{conv}} T_b - h_{\text{nb}} T_{\text{sat}}$$

T_{co} can then be eliminated with Equation B.13 to find,

$$q''_{\text{co}} = (h_{\text{conv}} + h_{\text{nb}}) \left(T_{N-1} - \frac{q''_{\text{co}} \Delta r_{N-1}}{2k_{N-1}} \right) - h_{\text{conv}} T_b - h_{\text{nb}} T_{\text{sat}}$$

$$q''_{\text{co}} \left[1 + (h_{\text{conv}} + h_{\text{nb}}) \frac{\Delta r_{N-1}}{2k_{N-1}} \right] = (h_{\text{conv}} + h_{\text{nb}}) T_{N-1} - h_{\text{conv}} T_b - h_{\text{nb}} T_{\text{sat}}$$

$$q''_{\text{co}} = \frac{(h_{\text{conv}} + h_{\text{nb}}) T_{N-1} - h_{\text{conv}} T_b - h_{\text{nb}} T_{\text{sat}}}{1 + (h_{\text{conv}} + h_{\text{nb}}) \frac{\Delta r_{N-1}}{2k_{N-1}}}$$

B.3 Material properties

The fuel thermal conductivity is computed via a model from section 8.3 of [67]. Here it has been simplified with the assumption that the fuel is fresh UO_2 at 95% theoretical density,

$$k_f = \frac{1}{A + BT} + \frac{E}{T^2} \exp\left(-\frac{F}{T} \right)$$

$$A = 4.52 \text{ cm K/W} \qquad B = 2.46 \times 10^{-2} \text{ cm/W}$$

$$E = 3.5 \times 10^7 \text{ W K/cm} \qquad F = 16361 \text{ K}$$

The specific heat capacity of the fuel is modeled from [67] as,

$$c_{pf} = c_2 + 2c_3 t + 3c_4 t^2 + 4c_5 t^3 + 5c_6 t^4 - c_7 t^{-2}$$

$$t = \frac{T}{1000 \text{ K}} \qquad c_2 = 193.238 \qquad c_3 = 162.8647$$

$$c_4 = -104.0014 \qquad c_5 = 29.2056 \qquad c_6 = -1.9507$$

$$c_7 = 2.6441$$

The density, thermal conductivity, and specific heat capacity of the cladding are set at the values listed for Zircaloy-2 in [67]:

$$\rho_c = 6.5 \times 10^3 \ \frac{\text{kg}}{\text{m}^3} \qquad k_c = 13 \ \frac{\text{W}}{\text{m K}} \qquad c_{pc} = 330 \ \frac{\text{J}}{\text{kg K}}$$

B.3.1 Gap heat transfer

Different models are used for heat transfer across an open gap versus a closed gap—the gap when the fuel and cladding come into full contact. For simplicity the scope here is limited to fresh fuel where the gap is open. Similarly, it is assumed here that the gas in the gap is pure helium.

The heat transfer coefficient for the fuel-cladding gap is the sum of two components, one for the thermal conductivity of the gas and another for the radiative heat transfer between the fuel and cladding,

$$h_g = h_{g,\text{gas}} + h_{g,\text{rad}}$$

Finding the thermal conductivity of the gas in the gap is a fairly involved calculation. It depends quite a bit on the gas mixture, and it is complicated by effects that arise when the size of the gap is comparable to the mean free path of the gas in the gap. For more details, see [71].

An approximate model is acceptable for these purposes so the conductivity portion will use this simple model from section 8.7.1 of [67],

$$h_{g,\text{gas}} = \frac{k_g}{\delta_{g,\text{eff}}}$$

where k_g is the conductivity of the gap gas material and δ_{eff} is the effective thickness of the gap. This effective thickness approximates the previously mentioned effects of a small gap. For pure helium at atmospheric pressure, [67] gives the effective thickness as the true thickness plus 10 μm,

$$\delta_{g,\text{eff}} = \delta_g + 10\ \mu\text{m}$$

This is the model adopted in this work.

The thermal conductivity of helium is given by,

$$k_g = 3.366 \times 10^{-3} T^{0.668}$$

where k_g is in units of W/m K and T is the gas temperature in units of K [71].

The radiative component of gap heat transfer is given by,

$$h_{g,\text{rad}} = \sigma \left[\frac{1}{\epsilon_f} + \frac{r_{\text{fo}}}{r_{\text{ci}}} \left(\frac{1}{\epsilon_c} - 1 \right) \right]^{-1} \left(T_{\text{fo}}^2 + T_{\text{ci}}^2 \right) \left(T_{\text{fo}} + T_{\text{ci}} \right)$$

where σ is the Stefan-Boltzman constant; ϵ_f and ϵ_c are the emissivities of the fuel and cladding; r_{fo} and T_{fo} are the radius and temperature of the fuel outer surface; r_{ci} and T_{ci} are the radius and temperature of the cladding inner surface [72]. (Note that reference [72] has a typo: it uses ϵ_f where it should use $\frac{1}{\epsilon_f}$.)

Using a model from MATPRO-11 [71], the emissivity of the fuel is given by

$$\epsilon_f = \begin{cases} 0.8707 & T < 1000 \text{ K} \\ 1.311 - 4.404 \times 10^{-4} T & 1000 \text{ K} \leq T \leq 2050 \text{ K} \\ 0.4083 & T > 2050 \text{ K} \end{cases}$$

where T is in units of K.

Section B3 of MATPRO-11 discusses cladding emissivity [71]. Note that these models are extremely dependent on the cladding oxide thickness. Cited values range from 0.2 to 0.9. Evaluating the cladding oxide thickness may require details of the manufacturing process that are hard to come by, and the resulting accuracy is probably unnecessary for the purposes of this work. So the simplifying assumption is made that the cladding emissivity is always,

$$\epsilon_c = 0.5$$

This value is plausible for thin oxide layers.

B.3.2 Nucleate boiling heat transfer coefficient

The Chen correlation is used for the nucleate boliing portion of coolant heat transfer coefficient. An original publication of this coefficient by Chen in Imperial units can be found in [73]. Beware the pounds and pounds-force! An SI verison can be found in section 13.3.1.2 of [67] and is reproduced here:

$$h_{\text{nb}} = S \cdot 0.00122 \frac{k_f^{0.79} c_{pf}^{0.45} \rho_f^{0.49}}{\sigma^{0.5} \mu_f^{0.29} h_{fg}^{0.24} \rho_g^{0.24}} \Delta T_{\text{sat}}^{0.24} \Delta P^{0.75}$$

$$S = \frac{1}{1 + 2.53 \times 10^{-6} \mathrm{Re}^{1.17}}$$

where k_f, c_{pf}, ρ_f, and μ_f are respectively the thermal conductivity, specific isobaric heat capacity, density, and shear viscosity of saturated liquid water; σ is the surface tension; h_{fg} is the difference in specific enthalpy between saturated vapor and saturated liquid; ρ_g is the density of saturated vapor; ΔT_{sat} is the difference between wall temperature and saturation temperature; and ΔP is the difference between saturation pressure at the wall temperature and the coolant pressure.

Appendix C

Dynamic Shared-Memory MC Neighbor Lists

This appendix chapter describes various neighbor list implementations considered for use in OpenMC, and evaluates their performance for a simple 3D pincell problem. The text and data presented here closely match those presented in a separate publication [41]. An addendum is provided at the end of this chapter to discuss the impact of these neighbor lists on a quarter-core problem.

C.1 Background on neighbor lists

MC codes comprise many distinct parts that can each become a bottleneck depending on the application. MC researchers generally focus their attention on the tally and cross section evaluation components of the software, but the ray tracing code can also become a bottleneck if it is overlooked. Here the term *ray tracing* refers to the components of the MC code which compute where a simulated particle is located in the geometry and where it is going next.

Each time a particle crosses a boundary in a MC solver, the software must determine which geometric cell the particle is entering.[1] The simplest approach would be to computationally search through every cell in the geometry and check if each one is on the other side of the surface, but this approach would be prohibitively slow. Instead, MC codes build

[1]Note that resolving surface crossings is not necessary with the delta-tracking method. This chapter is focused on simulations that do not use delta-tracking.

neighbor lists so that only a subset of the cells must be searched when a surface is crossed.

There are several different ways to implement neighbor lists. The most fundamental design choice for neighbor lists is what type of geometric entity their adjacency relationship is based on. One option is surface-based adjacency, where there is a neighbor list for each surface in the geometry (or one for each side of each surface). Another option is cell-based adjacency where there is one neighbor list for each cell. Figure C-1 provides a diagram to highlight the distinction.

These approaches can also be combined: the code can track multiple neighbor lists for each cell, one for each surface that the cell is in contact with. This combined approach results in the shortest neighbor lists, and shorter neighbor lists should provide better performance. The combined approach will be referred to as *side-based* neighbor lists here, as in there is a list for each side of each cell.

Neighbor lists are rarely discussed in MC code manuals or publications. Two exceptions to this are Sutton et al. who described the RACER implementation in 1999 [74] and She et al. who described the RMC implementation in 2011 [75]. RACER uses the side-based approach for up to 4 neighbors per side, and then falls back to the surface-based approach for further neighbors [74]. RMC uses an advanced side-based approach where the neighbor list is re-sorted during the simulation so that the more significant neighbors appear earlier in the neighbor lists [75]. She et al. also suggest that the side-based approach is used in MCNP5 [75]. Private communication with Griesheimer indicates that MC21 also uses the side-based approach [76].

Until recently, OpenMC used the surface-based approach. As this chapter will demonstrate, surface-based lists can lead to poor performance for multiphysics geometries. It is an interesting historical note that OpenMC is perhaps the only widely-used MC code to

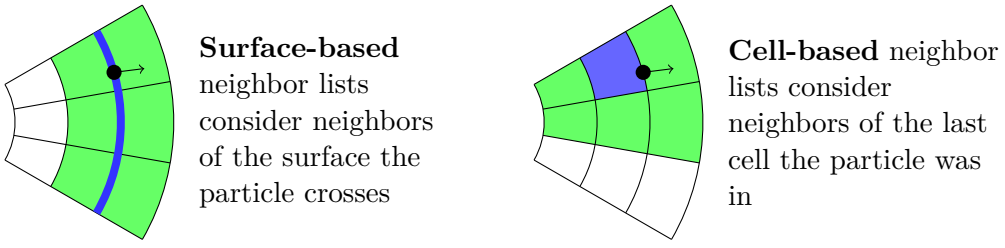


Figure C-1: Diagram to indicate the difference between surface-based and cell-based neighbor lists on a hypothetical geometry.

have relied on surface-based lists. These sorts of inefficiencies in newly-developed tools are lamentable, but they are a natural consequence of independent development. These inefficiencies are also easily corrected by the careful attention of developers over time.[2]

C.2 Surface-based versus cell-based neighbor lists

In the surface-based scheme every surface object has an associated list of neighboring cells. (In most implementations, each surface actually has two lists—one for each side of the surface.) When the MC code simulates a particle crossing the surface, the neighbor list for that particular surface is used to find the next cell.

A major advantage of the surface-based neighbor list scheme is that the lists are easy to construct. At initialization, the program can iterate through each cell and add it to the appropriate neighbor lists of its bounding surfaces. Note that iterating over a cell's bounding surfaces is trivial because the bounding surfaces are an inherent part of the cell's definition. (The bounding surfaces are typically explicitly specified by the user.)

Surface-based neighbor lists can greatly accelerate ray tracing in some cases, but their efficiency is degraded by the fact that a single surface can bound many cells in reactor geometries. Consider the cylindrical surfaces in a fuel pin model that is divided into axial regions. A sketch of such a geometry is shown in Figure C-2. The cylinder neighbors many cells in the geometry, and the number of neighbors scales proportionally with the number of axial regions. Axial discretization on the order of a centimeter would lead to hundreds of neighbors per cylinder in a typical PWR geometry. These overly large neighbor lists can easily cause a majority of the program's runtime to be spent performing cell searches after boundary crossings.

In order to address this issue, neighbor lists can instead be set up to represent cell-to-cell adjacency rather than surface-to-cell. Cells in a discretized fuel pin model have a roughly constant number of neighbors regardless of the axial discretization. See Figure C-3 for a sketch of this geometry.

Cell-based neighbor lists are shorter in such geometries and therefore faster to search, but they are more difficult to generate during the program initialization. The reason is that

[2]The decision to use surface-based neighbor lists in OpenMC can be partly traced to a publication of MC21 data structures [77]. A few sentences in this publication describe surface-based neighbor lists with no mention of the side-based lists. Likely, these described surface-based lists are a fall-back to the side-based neighbor lists as they are in RACER.

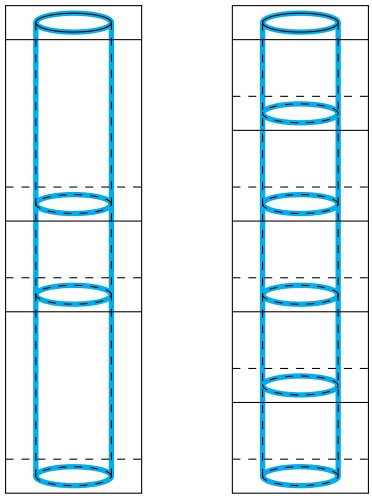


Figure C-2: A simplified fuel pin geometry with two different levels of axial discretization. The highlighted cylindrical surface divides fuel cells from moderator cells. The number of cells neighboring the highlighted surface scales proportionally with the axial discretization.

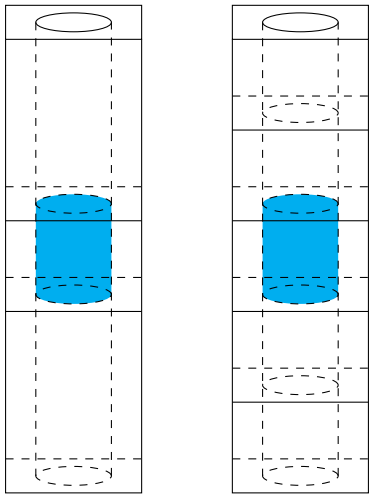


Figure C-3: A simplified fuel pin geometry with two different levels of axial discretization. The number of cells neighboring the highlighted cell does not scale with the discretization.

it is hard to devise a general algorithm for indicating whether two 3D constructive solid geometry cells neighbor each other. Even if a suitable algorithm can be found for analytical geometries, some difficulties will probably be encountered in a real implementation because of finite-precision numerical errors.

Cell-based neighbor lists are instead generated dynamically during the simulation, essentially using the ray tracer to determine neighboring cells. When a particle crosses a surface in this scheme, the program first searches through cells from the neighbor list of the previous cell. If no matching cell is found, then an exhaustive search through all cells in the geometry universe is performed. Next, the resulting cell from that search is added to the neighbor list. The lists are empty at the beginning of the simulation. Then they are quickly populated, and exhaustive searches become rarer as the simulation proceeds.

Cell-based neighbor lists require dynamic data structures so that cells can be added to them during transport. In a code that utilizes shared-memory parallelism, maintaining efficiency with dynamic data structures is difficult so exploring several implementation schemes is important.

Side-based neighbor lists are a further improvement on cell-based neighbor lists. Instead of one neighbor list per cell, each cell has a neighbor list for each of the surfaces that it is bound by. They are similar to cell-based lists in terms of the axial discretization performance and the need for a dynamic data structure. They differ in that side-based neighbor lists will be shorter and therefore faster to search through.

Side-based neighbor lists are expected to perform better than or similarly to cell-based lists, but they have not yet been implemented and tested in OpenMC. Consequently discussion of them is limited here. However, note that a dynamic data structure that can be used for cell-based lists can also be used for side-based lists, and the performance implications are expected to be similar.

C.3 Dynamic cell-based data structures

In single-threaded computing, the obvious approach to implement dynamically growing neighbor lists is to use a data structure such as the C++ standard library `std::vector` that is a contiguous, resizable array allocated with dynamic memory. However, vectors raise thread contention issues. Writing to the array may require deallocating and then

reallocating memory for it. If one thread deallocates the array in this process while another thread attempts to read it, errors occur. Protecting all read and write access to the array with locks or the `#pragma omp critical` directive would prevent these shared-access errors, but doing so is prohibitively slow for this application. Some other threadsafe data structure must be used.

Three threadsafe data structures are considered here:

1. duplicated threadprivate vectors,
2. a shared singly linked list, and
3. a shared hybrid structure with a vector and a singly linked list.

The first approach is to represent the neighbor list with a vector modified by the OpenMP `#pragma omp threadprivate` directive. Use of this directive means that the vector will be duplicated and each thread will use its own private copy.

Threadprivate neighbor lists eliminate the thread contention issues, but they incur a memory cost due to the duplication. Threadprivate variables can also bring unexpected runtime costs due to added overhead such as accessing a thread local storage (TLS) lookup table. For neighbor lists, this approach will bring an additional runtime cost since information is not shared between threads and each thread must build up its own lists with exhaustive geometry searches.

The second approach uses a singly linked list (C++ standard library `std::forward_list`) that is shared across all threads. For most cases of interest, MC neighbor lists are frequently read but rarely modified, and singly linked lists offer inherent threadsafe read access. Adding an element to the end of a linked list does not require reallocating or moving any of the other elements in the list, so one thread can safely write to the list while other threads read from the same list. Only one thread at a time can safely add elements to the list, so a locking mechanism is needed for write access, but this is a small cost given how infrequently the neighbor list must be modified. Also note that pointers in the list must be read and written atomically to prevent one thread from reading an invalid pointer that is being concurrently written to from another thread.

One small detail worth highlighting is that new elements should always be added to the end of the linked list rather than the beginning. Adding elements to the beginning is tempting because the resulting code is simpler. Adding to the end first requires traversing

the list but will result in a list that is roughly sorted in a favorable order. The neighbor list for each cell is ideally sorted by how frequently particles enter each neighbor when leaving the base cell, and this frequency will roughly correspond to how early each neighbor is encountered. For the example problem presented here, adding new elements to the end rather than the beginning of the list results in about a 5% speedup. Note that other work on the RMC code showed that re-sorting neighbor lists on the fly can lead to factors of about 2× speedup in some cases [75].

Linked lists come with some disadvantages over vectors. Linked lists require more memory because each element in the list must hold a pointer to the next element. The elements are also not contiguous in memory which can hurt runtime performance because of poor spatial locality resulting in cache misses.

The third approach aims to harness both the shared-memory threadsafe linked lists and the contiguous memory layout of vectors. The data structure includes a vector "prefix" and a linked list "suffix". When cells are added, they are added to the end of the suffix. Reading from the structure requires reading both the prefix and the suffix. The MC code can then periodically empty the suffix and move all of its elements to the prefix to make the data contiguous. In the OpenMC implementation used for this test, this reorganization occurs between generations of particles. The neighbor lists are not used for ray tracing at that time so it is easy to ensure that only one thread at a time interacts with any given neighbor list.

C.4 Performance tests

In order to compare the performance of the old and new neighbor list schemes, OpenMC calculations of a PWR pincell model were run using each scheme and varying levels of axial discretization. In the xy-plane, the model includes fuel, gap, cladding, and coolant with reflective boundary conditions. The pin is 366 cm long with vacuum boundary conditions at either end. All four materials (fuel, gap, cladding, and coolant) are discretized axially into equal-height segments. No tallies are included in this model. Fresh UO_2 is used for the fuel material.

Calculations were run on a single dual-socket computer node with Intel Xeon E5-2683 v4 processors. Each of the two processor packages has 16 cores, so calculations were run

with up to 32 OpenMP threads. Simulations used 10 generations with 20,000 particles per generation. All calculations were repeated 20 times with different random number seeds to establish a mean and confidence interval. The software was compiled with GCC version 6.2.0.

One set of calculations was aimed at scalability in terms of axial discretization. Simulations were run with 32 OpenMP threads using each neighbor list scheme, and the number of axial regions was varied between 1 and 200. The runtime performance is plotted in Figure C-4. A 95% confidence interval was established, but it is very small and not indicated on the plots.

The first major finding from the data in Figure C-4 is that each cell-based implementation significantly outperforms surface-based neighbor lists. With only a single axial region, cell-based and surface-based lists show a similar performance of about 130,000 particles per second. With 200 axial regions the surface-based rate slows to about 8,000 s^{-1}, but the cell-based lists maintain a rate between 80,000 s^{-1} and 100,000 s^{-1}.

Code profiling confirms that the boundary-crossing algorithm is the primary source of the cell-based versus surface-based runtime differences seen in Figure C-4. Using the surface-based lists, the fraction of runtime spent in OpenMC's `cross_surface` subroutine grows from 21% to 92% with 1 and 200 axial regions, respectively. With the cell-based lists, the fraction grows from 19% to 21%.

The differences between the cell-based implementations are smaller. The hybrid method is consistently faster than the linked list, but only by about 1–2%. The duplicated threadprivate neighbor lists perform about as well as the other cell-based lists for moderate axial discretization, but they scale poorly with finer discretization and are 20% slower than the others with 200 axial regions. This performance is likely due to the loss of shared information between threads, that is, the fact that neighbor list entries learned by one thread are not shared with the others.

Another set of calculations was aimed at scalability in terms of threading. Simulations were run with 200 axial regions and a varying number of OpenMP threads between 1 and 32. The results for the three cell-based schemes are plotted in Figure C-5. The data in Figure C-5 again show that the hybrid method and the linked list have similar performance. The duplicated threadprivate vectors show poorer thread scaling. Because the CPU has a shared L3 cache, as the number of threads increases, so does the probability that a memory

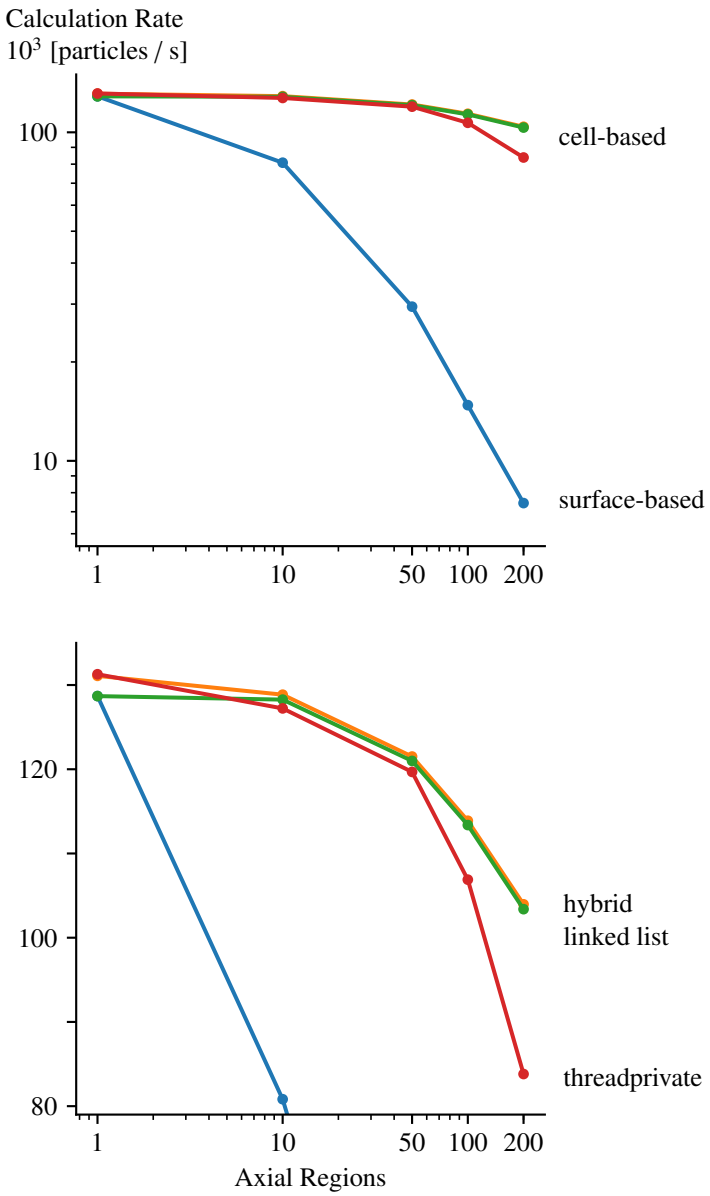


Figure C-4: Runtime performance of the different neighbor list schemes as a function of axial discretization. The same data are used in both subplots. The upper subplot shows the large difference between cell-based and surface-based neighbor lists, and the lower focuses on the different cell-based implementations.

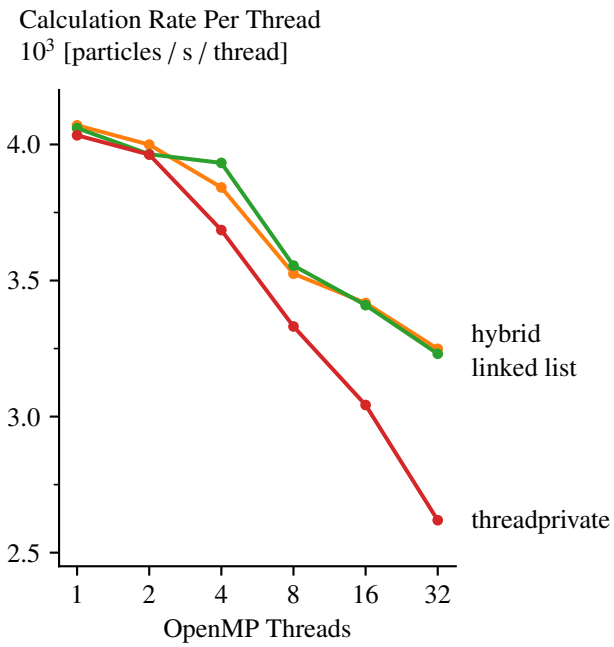


Figure C-5: Runtime performance as a function of thread count for each cell-based implementation on a model with 200 axial regions.

request will result in an L3 cache miss (hence the decreasing performance per thread). This effect is exacerbated with the duplicated threadprivate vectors wherein each thread has its own copy of the neighbor lists.

One caveat of these results worth noting is that the different neighbor list schemes have a particularly large impact for this problem, but they will not have such a large impact for many other problems of interest. If a large suite of tallies is added, they will consume a significant fraction of the runtime and decrease the relative impact of different neighbor lists. Using depleted fuel materials with many isotopes will increase the cost of cross section calculations which would also reduce neighbor list impact. Different geometries like reactors with TRISO fuels or tokamak fusion reactors will see different results.

C.5 Conclusions

Neighbor list implementation details can have a significant impact on the performance of a MC code. These details are particularly important for extruded geometries (e.g. most light water reactors) that are finely discretized for multiphysics or depletion purposes. For these models, cell-based neighbor lists offer a substantial runtime improvement over surface-based

neighbor lists. Dynamic data structures are needed so that cell-based neighbor lists can be built during the transport simulation, and care must be taken in their implementation to provide efficient, threadsafe performance.

A singly linked list is a simple data structure that performs well in the tests presented here. The list is shared among all threads, which means that memory costs do not scale with the number of threads, and each thread benefits from the neighbor list entries discovered by other threads. The presented results demonstrate that runtime for a program using linked lists scales well in terms of both geometry discretization and thread count.

A hybrid data structure is presented that combines a linked list with a vector to gain a small runtime performance increase over the simple linked list. The gain is so small, however, that many MC code developers will likely not consider it worth the slightly increased software complexity.

Dynamic neighbor lists can also be implemented with threadprivate duplicated vectors. This approach is comparable to the linked list and hybrid approaches for coarsely discretized problems and few threads, but it is not recommended because its performance degrades with finer discretization and a higher thread count. Extrapolating from the results plotted in Figure C-4, a user who desires even finer discretization than that studied here can expect a precipitous drop in performance of threadprivate vectors versus the other data structures. Furthermore, if CPUs continue to reach incrementally higher core counts, good shared-memory scaling will become increasingly important in the future.

Based on the conclusions of this work, the neighbor list implementation used in OpenMC was updated with pull request #1140. OpenMC switched from surface-based adjacency to cell-based adjacency using the threadsafe linked-list data structure.

C.6 Addendum

C.6.1 Quarter-core performance

The preceding text of this appendix chapter closely matches a separate publication [41], and it describes a test problem that was relevant to the early stages of this thesis work. The finely-discretized pincell problem is also uniquely sensitive to the neighbor list implementation so the performance differences are not as extreme for most problems.

Simulations were run to test the impact of the neighbor list changes on the quarter-core

PWR multiphysics problem described in Chapter 7. These simulations used the same accelerated multiphysics solver discussed in Chapter 7, and the same geometric discretization. This problem uses a 42-region axial mesh. Unlike the pincell study just discussed, this problem uses a single radial region for the fuel interior. Simulations ran for 5 MC generations and used 200 million neutrons per generation. These simulations were run using Google Cloud Platform on a single "n1-standard-64" instance. Exact details of the machine architecture are unknown, but it presents as a machine with 64 generic Intel Xeon cores.

For this study, surface-based neighbor lists were reimplemented in a recent version of OpenMC. Consequently, the only software difference between the two simulations is the neighbor list implementation. (Both simulations include universe partitioning which is discussed in Section 4.2.2.)

With surface-based neighbor lists, the MC portion of the solver has a wall time cost of 5.3 hours (340 cpu core hours). With cell-based linked lists, the wall time falls to 4.0 hours (260 cpu core hours) which is a 25% reduction. The MC solver is the most expensive of each of the coupled physics solvers so the overall multiphysics solver runtime reduction is 23%.

C.6.2 Limitations of the neighbor lists

It is important to note that the performance differences shown here between surface-based and cell-based neighbor lists can be squandered by inefficient geometry discretizations that result in cells with many neighbors. For a concrete example, consider a cell which describes the gap between the fuel and cladding. In this thesis, the gap has a constant width over the entire axial dimension, and it is filled with a void instead of a material. This means it has axially uniform properties and could be represented with a single OpenMC cell. However, this single gap cell would neighbor a large number of cells that are used to axially discretize the fuel. As a result, the performance of cell-based neighbor lists would approach the performance of surface-based neighbor lists.

For this reason, it is important to axially discretize all regions of the geometry in the active core. In this work, the same z-planes that are used to slice the fuel are also used to slice the gap, cladding, coolant, control rods, and burnable absorbers.

A further detail: the pincell study presented in this chapter focused on simulations that used only one *universe*. In OpenMC (and other MC codes), a universe is a concept that

is used to compose complex geometries. Each tile in a lattice, for example, is filled with a universe. One universe might be used for a fuel pin, and another for a guide tube, and then both of these universes can be tiled repeatedly in a lattice to form a fuel assembly.

Because of various implementation details, neighbor lists are not used in OpenMC when a particle crosses a boundary between different universes (or even two different instances of the same universe). Consequently, the neighbor lists changes discussed in this chapter are irrelevant for particles that cross a boundary within a lattice. Section 4.2.2 discusses a performance improvement targeted at these universe crossings.

Appendix D

Miscellaneous notes

D.1 Comparison to BEAVRS hot zero power measurements

The BEAVRS benchmark includes measurements made on the first day of operations at low power (24 MW) conditions [47]. These measurements offer a way to check for geometry/material input errors without the confounding effects of thermal feedback.

Note that the raw measurements show an unexplained asymmetry, and the benchmark provides a "corrected" dataset which reduces the asymmetry by subtracting a linear fit. This tilt-corrected dataset is used as the reference here.

Simulations were run with the full coupled solver even though thermal feedback is not significant for this problem. The simulations used 30 Picard iterations with one MC generation per iteration and 200 million neutrons per generation. With acceleration, the simulation is stationary after 15 iterations. Stationarity was established by visually inspecting plots of the power distribution in various parts of the reactor as a function of iteration number. The presented results are the mean values from iterations 15 through 30 of one simulation. Two simulations were run with different random number seeds to check for statistical anomalies.

As one measure of convergence, assemblies with an octant-symmetric neighbor were compared to each other. The axially-integrated detector response in each of these assemblies differs by an average of 0.2% from its symmetric partner.

Figures D-1 and D-2 show how the simulated fission detector response distribution compares with the tilt-corrected measurements of the real reactor core. Figure D-1 shows the radial distribution (axially-integrated) throughout the core. Note that this dataset shows one octant, but measurements from the entire core are used under the assumption

that they should be roughly octant-symmetric.

Figure D-2 shows the axial profile of the detector response in assembly D12. This assembly is chosen because it is the highest power assembly in the full-power conditions. The error shown in Figure D-2 for D12 is similar to the error seen in other assemblies.

These results provide some confidence that there are no large errors in the geometry and material inputs. Errors seen with earlier simulations generally resulted in radial distributions with large in-out tilts (e.g. positive errors clustering in the center of the reactor and negative errors clustering on the periphery) or a checkerboard pattern matching the distribution of assembly enrichments. But none of these tell-tale signs are present in Figure D-1.

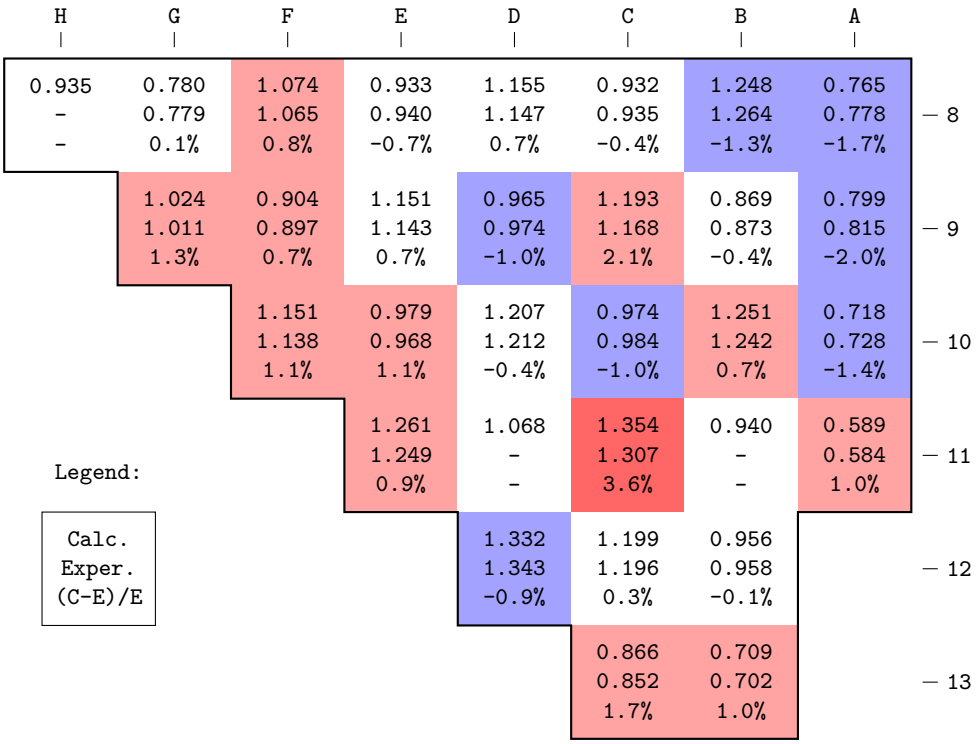


Figure D-1: Calculated and measured detector response in each fuel assembly on day 1 of operation (hot zero power).

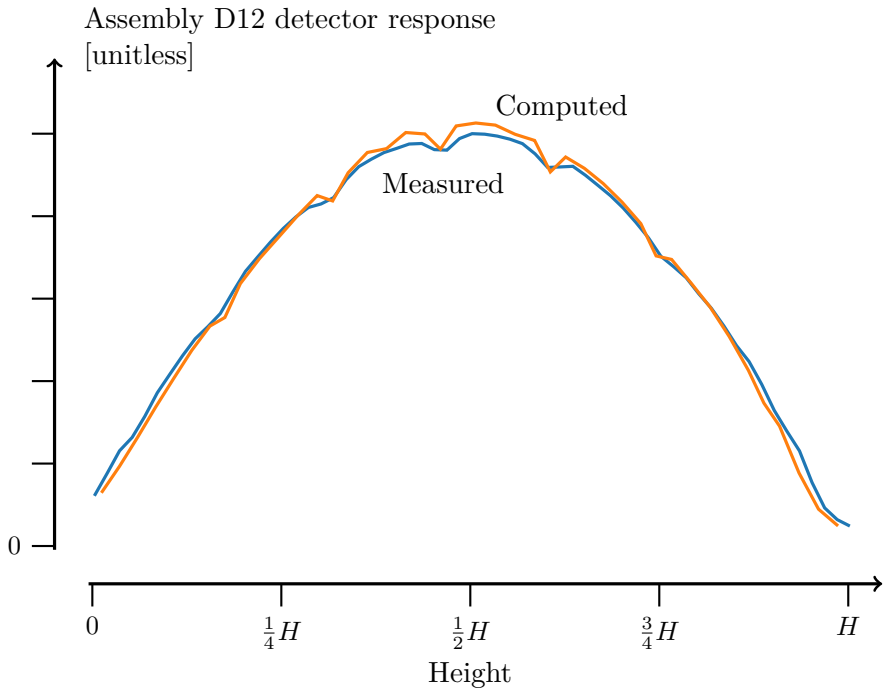


Figure D-2: Measured and computed axial detector profile in assembly D12 on day 1 of operation. Both curves have been normalized to the same integral.

D.2 How big is a pcm?

A pcm is a one-in-one-hundred-thousand error on the neutron multiplication factor. But how can that be related to intuitive physical quantities? This section describes a reference point that I often use.

Figure D.2 shows two images of the fuel used in LR-0, a zero-power research reactor. The first image in this figure shows an entire fuel assembly which is a shortened version of the fuel assemblies used in VVER reactors. The second image shows the very bottom of two different fuel pin designs that can be used in this assembly.

These pins are nearly identical, except that one has a smaller diameter on the lower end, just beneath the region filled with active fissile material. The diameter of one pin is 4 mm smaller than the other over a length of about 1.7 cm. The reactor is filled with a water moderator so swapping from the thick design to the thin one leads to an extra 0.7 g of water here at the bottom of each fuel pin. This difference seems negligible given the size of the full fuel assembly; there is roughly 130 cm of active fuel above the pictured feature. But the difference is nevertheless measurable in this experimental facility.

For the cores that use 6 of these assemblies, swapping out all of the fuel pins from one design to another leads to a 70 pcm reactivity change. This value is small, but it is non-negligible compared to the experimental perturbations tested in this facility. The researchers who use this reactor are therefore careful to model these fuel-pin-bottom details.

Figure D-3: Left: a fuel assembly for the LR-0 reactor at CVŘež. For scale, note the workers handling the assembly. Right: the lower end of two different fuel pin designs for the LR-0 assemblies. The upper of the two lines in the background indicates the bottom of the UO_2 fuel in the interior.

D.3 Estimating the uncertainty of a single simulation

In Chapter 7, conclusions were generally established after considering data from at least two independent simulations. This level of rigor is not always necessary so some conclusions in this appendix are made from single-simulation calculations. For this purpose, it is necessary to estimate the stochastic uncertainty of single-simulation results.

D.3.1 Eigenvalue

Chapter 7 described a study using many 15-generation calculations. For this study, 11 of the 15-generation simulations were run. (Only 10 are used in the Chapter 7 study as this makes the fraction of fine-mesh regions with $< 2\%$ error conveniently close to 95%.)

The average eigenvalue for each simulation is computed over generations 6 through 15. The standard deviation—standard deviation of eigenvalues, *not* the standard error of the mean of the eigenvalues—is 3 pcm. When listing an uncertainty, it is better to list a two-standard-deviation figure as this roughly corresponds to the 95% confidence interval for normal data. In this case, that corresponds to 6 pcm so the eigenvalue uncertainty for an individual simulation in this appendix will be given as ± 6 pcm.

This uncertainty will be applied to 30-generation simulations which are likely more accurate than the 15-generation simulations used to compute that 6 pcm value, but 6 pcm is small enough that its not worth worrying about.

D.3.2 Radial instrument distribution

Figure D-4 shows the radial distribution of errors for two 30-generation simulations at full power conditions. The largest difference between these is 0.3% (absolute difference, not relative) so 0.3% is considered the uncertainty for these results.

```
Error in radial distribution from two simulations:

(C-E) / E Error [%]:
     -7.5 -3.0 -5.0  0.3 -2.1  4.4  5.0
     -4.3 -4.8 -1.3 -3.3  2.0  0.2  5.8
          -1.9 -3.2  0.6 -1.7  4.3  6.0
               -0.3       3.1       5.1
                     1.1  2.0  5.8
                          3.4  8.1

(C-E) / E Error [%]:
     -7.5 -2.7 -5.2  0.2 -2.4  4.4  4.7
     -4.3 -4.9 -1.3 -3.5  2.1  0.3  6.1
          -1.8 -3.3  0.7 -1.6  4.2  6.0
               -0.3       3.0       5.2
                     1.1  1.9  6.0
                          3.4  8.2
```

Figure D-4: Relative error for axially-integrated detector signals from BEAVRS day 169 (hot full power). Results from two independent simulations are shown. Each simulation used 30 generations with 200 million neutrons per generation. Results are averaged from generations 5 through 30.

D.4 Number of fuel rings

A simple study was run to quantify the impact of discretizing the MC geometry with multiple radial regions in the fuel. Simulations were run on the BEAVRS day 169 (hot full power) conditions using the full solver (including subchannel, heat transfer, and equilibrium xenon). Each simulation used 30 generations with 200 million particles per generation. Results were averaged over generations 6 through 30. These simulations used either 1 or 3 rings in the fuel. Note that even with 3 rings, the concentration of xenon and the fuel power density are still considered radially uniform.

The eigenvalue with 3 rings is 0.99853, which is 27 pcm greater than the 1 ring case. Per Section D.3, the eigenvalue uncertainty is 6 pcm. Figure D-5 shows the radial error distribution from the two cases. Figure D-6 shows the axial profile in assembly D12.

The 3-ring case seems to slightly reduce the in-out tilt error, but the effect is not large. The difference in the axial distribution is quite small, and note that D12 (the highest power assembly) is likely the most sensitive to spatial self-shielding errors. Overall, the difference here is considered negligible so 1-ring calculations are used elsewhere in the thesis.

```
1 radial ring in the fuel:
(C-E) / E Error [%]:
     -7.5 -2.7 -5.2  0.2 -2.4  4.4  4.7
     -4.3 -4.9 -1.3 -3.5  2.1  0.3  6.1
          -1.8 -3.3  0.7 -1.6  4.2  6.0
               -0.3       3.0       5.2
                     1.1  1.9  6.0
                          3.4  8.2

3 radial rings in the fuel:
(C-E) / E Error [%]:
     -7.2 -2.9 -5.2  0.2 -2.8  4.2  4.9
     -4.2 -4.9 -1.4 -3.4  2.3  0.3  5.8
          -1.9 -3.3  0.7 -1.5  4.3  5.8
               -0.2       3.2       5.1
                     1.2  1.9  5.8
                          3.4  8.3
```

Figure D-5: Relative error for axially-integrated detector signals from BEAVRS day 169 (hot full power). Results using 1 ring and 3 rings in the fuel are shown. The stochastic uncertainty on each value is ±0.3% per Section D.3.

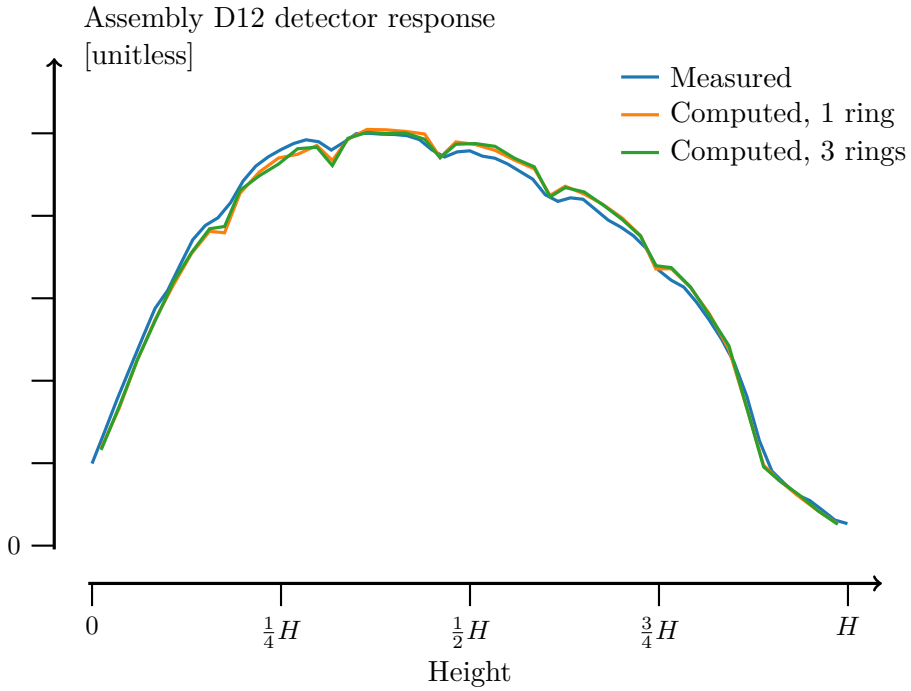


Figure D-6: Measured and computed axial detector profile in assembly D12 on day 169 of operation. All curves have been normalized to the same integral.

These conclusions are not expected to hold in general. Depletion will cause a significant non-uniform radial distribution of pin power to develop, which may make the radial temperature distribution more important to resolve. A gadolinia-bearing fuel pin may similarly require further discretization as thermal neutrons will only be sensitive to the temperature near the rim of the fuel.

Note that there is a significant computational cost to using a finer radial discretization. The solver runtime increases by about 30% after switching from 1 to 3 rings. This is expected to be largely due to the increased number of cross section evaluations in the MC solver; if a neutron passes directly through the center of a fuel rod then OpenMC will evaluate the fuel cross section 5 times versus 1 time.[1] Ray tracing costs may also contribute.

D.5 Sensitivity to fuel temperature

A number of approximations made in this work will lead to an over-estimate of the reactor fuel temperature. The power density within a pin is assumed uniform, but slightly more power is produced near the edge of the pin. All heat from fission energy is assumed to be deposited locally in the fuel, including the heat from neutrons and photons which in fact deposit their energy throughout the reactor materials. All heat is assumed to come from fission, and the heat from capture reactions (which can occur in non-fuel regions) is ignored. The fuel-clad gap is assumed to have a constant thickness.

To roughly understand how sensitive the results might be to these assumptions, a perturbed simulation was run. For this simulation, the q' values used by the heat transfer solver are reduced to $0.9\times$ their nominal value. The q' values used by the subchannel solver are unperturbed.

This perturbation plausibly counteracts the error of assuming local energy deposition. About 10% of the energy generated by fission is released in the form of neutrons and photons, and most of the neutron energy will be deposited in the moderator instead of the fuel.

As a result of this perturbation, the solver computes a slightly lower fuel temperature. For example, the average fuel temperature in assembly D12 near the core midplane drops from about 1 480 K to 1 390 K.

[1] Perhaps multipole cross sections at multiple temperatures can be evaluated concurrently with SIMD operations to reduce this cost. It may also be helpful to cache cross sections from recently-evaluated temperatures.

Consequently, the eigenvalue increases to 0.99954 which is 138 pcm higher than the unperturbed value. Figure D-7 shows the resulting radial error, and Figure D-8 shows the axial error.

The axial effect of this perturbation is very small. The radial effect is small, but it noticeably reduces the in-out tilt error. This suggests that the radial error is fairly sensitive to the fuel temperature—the perturbation is only about 100 K in the high-power areas of the core. Given that changes in the fuel-clad gap are expected to have a very nonlinear impact on fuel temperature, they are probably important to resolve when accuracy is the goal.

```
Unperturbed simulation:
(C-E) / E Error [%]:
     -7.5 -2.7 -5.2  0.2 -2.4  4.4  4.7
     -4.3 -4.9 -1.3 -3.5  2.1  0.3  6.1
          -1.8 -3.3  0.7 -1.6  4.2  6.0
               -0.3       3.0       5.2
                     1.1  1.9  6.0
                          3.4  8.2

90% power to heat transfer solver:
(C-E) / E Error [%]:
     -7.2 -2.5 -5.1  0.4 -2.3  4.3  4.7
     -4.0 -4.6 -1.3 -3.3  2.0 -0.0  5.7
          -1.8 -3.3  0.7 -1.8  4.1  5.5
               -0.2       3.0       4.9
                     1.0  1.9  5.6
                          3.3  8.0
```

Figure D-7: Relative error for axially-integrated detector signals from BEAVRS day 169 (hot full power). Results are shown for a simulation where the power seen by the heat transfer solver has been artificially reduced. The stochastic uncertainty on each value is ±0.3% per Section D.3.

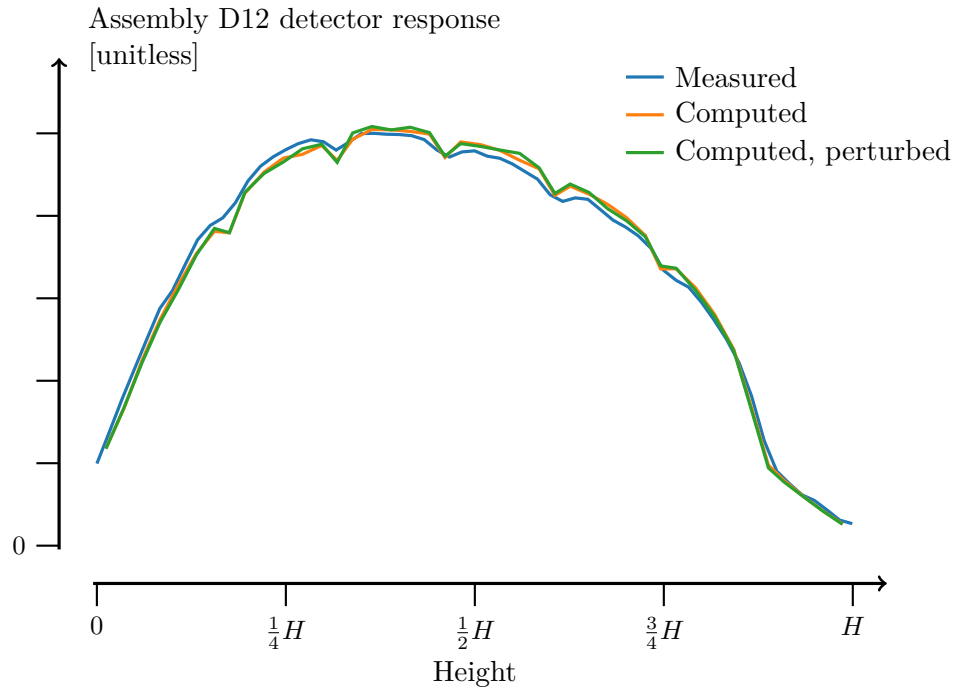


Figure D-8: Measured and computed axial detector profile in assembly D12 on day 169 of operation. All curves have been normalized to the same integral.

Appendix E

Bibliography

[1] "History, Development and Future of TRIGA Research Reactors," , IAEA (2016)URL https://www-pub.iaea.org/MTCD/Publications/PDF/trs482web-68751096.pdf.

[2] D. Knott and A. Yamamoto, "Lattice Physics Computations," D. G. Cacuci (Editor), *Handbook of Nuclear Engineering*, chap. 9, Springer; https://dx.doi.org/10.1007/978-0-387-98149-9_9.

[3] D. J. Kelly, A. E. Kelly, B. N. Aviles, A. T. Godfrey, R. K. Salko, and B. S. Collins, "MC21/CTF and VERA multiphysics solutions to VERA core physics benchmark progression problems 6 and 7," *Nuclear Engineering and Technology*, **49**, *6*, 1326 (2017); https://dx.doi.org/10.1016/j.net.2017.07.016.

[4] J. Leppänen, R. Mattila, and M. Pusa, "Validation of the Serpent-ARES code sequence using the MIT BEAVRS benchmark – Initial core at HZP conditions," *Annals of Nuclear Energy*, **69**, 212 (2014); https://dx.doi.org/10.1016/j.anucene.2014.02.014.

[5] J. Kuopanportti, S. Saarinen, and J. Leppänen, "Assessment Of Serpent Cross Sections Of The Second Generation Fuel Of TVEL For Loviisa Npp," *PHYSOR 2018, Cancun, Mexico* (2018).

[6] K. S. Smith, "Nodal Method Storage Reduction by Nonlinear Iteration," *Transactions of the American Nuclear Society*, **44**, 265 (1983).

[7] T. Sutton and B. Aviles, "Diffusion theory methods for spatial kinetics calculations," *Progress in Nuclear Energy*, **30**, *2*, 119 (1996); https://dx.doi.org/10.1016/0149-1970(95)00082-U.

[8] M.-J. Lee, H. G. Joo, D. Lee, and K. Smith, "Multigroup Monte Carlo reactor calculation with coarse mesh finite difference formulation for real variance reduction," *Joint International Conference on Supercomputing in nuclear Applications and Monte Carlo*, 17–21 (2010).

[9] K. P. Keady and E. W. Larsen, "Stability of Monte Carlo k-eigenvalue simulations with CMFD feedback," *Journal of Computational Physics*, **321**, 947 (2016); https://dx.doi.org/10.1016/j.jcp.2016.06.002.

[10] B. R. Herman, "Monte Carlo and Thermal Hydraulic Coupling using Low-Order Nonlinear Diffusion Acceleration," PhD Thesis, Massachusetts Institute of Technology (2014)URL http://hdl.handle.net/1721.1/95525.

[11] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, "OpenMC: A State-of-the-Art Monte Carlo Code for Research and Development," *Annals of Nuclear Energy*, **82**, 90 (2015); https://dx.doi.org/10.1016/j.anucene.2014.07.048.

[12] B. Kochunas, S. Stimpson, B. Collins, T. Downar, R. Brewster, and E. Baglietto, "COUPLED FULL CORE NEUTRON TRANSPORT / CFD SIMULATIONS OF PRESSURIZED WATER REACTORS," *PHYSOR 2012, Knoxville, TN, USA* (2012).

[13] D. S. Rowe, "Cross-Flow Mixing Between Parallel Flow Channels During Boiling. Part I. COBRA: Computer Program For Coolant Boiling In Rod Arrays," BNWL-371 PT 1, Battelle-Northwest (1967); https://dx.doi.org/10.2172/4410375.

[14] U. Bieder, F. Falk, and G. Fauchet, "CFD analysis of the flow in the near wake of a generic PWR mixing grid," *Annals of Nuclear Energy*, **82**, 169 (2015); https://dx.doi.org/10.1016/j.anucene.2014.08.004.

[15] F. Alexander, A. Almgren, J. Bell, A. Bhattacharjee, J. Chen, P. Colella, D. Daniel, J. DeSlippe, L. Diachin, E. Draeger et al., "Exascale applications: skin in the game," *Philosophical Transactions of the Royal Society A*, **378**, *2166* (2020); https://dx.doi.org/10.1098/rsta.2019.0056.

[16] E. Baglietto, "Anisotropic Turbulence Modeling for Accurate Rod Bundle Simulations," vol. Volume 4: Computational Fluid Dynamics, Neutronics Methods and Coupled Codes; Student Paper Competition of *International Conference on Nuclear Engineering*, 343–352 (2006); https://dx.doi.org/10.1115/ICONE14-89646.

[17] E. Merzari, H. Ninokata, and E. Baglietto, "Numerical simulation of flows in tight-lattice fuel bundles," *Nuclear Engineering and Design*, **238**, *7*, 1703 (2008); https://dx.doi.org/10.1016/j.nucengdes.2008.01.001.

[18] E. Baglietto, E. Demarly, R. Kommajosyula, N. Lubchenko, B. Magolan, and R. Sugrue, "A Second Generation Multiphase-CFD Framework Toward Predictive Modeling of DNB," *Nuclear Technology*, **205**, *1-2*, 1 (2019); https://dx.doi.org/10.1080/00295450.2018.1517528.

[19] D. S. Rowe and C. W. Angle, "Cross-Flow Mixing Between Parallel Flow Channels During Boiling. Part II. Measurement Of Flow And Enthalpy In Two Parallel Channels," BNWL-371 PT 2, Battelle-Northwest (1967); https://dx.doi.org/10.2172/4504525.

[20] T. Van der Ros and M. Bogaardt, "Mass and heat exchange between adjacent channels in liquid-cooled rod bundles," *Nuclear Engineering and Design*, **12**, *2*, 259 (1970); https://dx.doi.org/10.1016/0029-5493(70)90012-9.

[21] L. Meyer, "From discovery to recognition of periodic large scale vortices in rod bundles as source of natural mixing between subchannels—A review," *Nuclear Engineering and*

Design, **240**, *6*, 1575 (2010); https://dx.doi.org/10.1016/j.nucengdes.2010.03.014.

[22] S. V. Möller, "Single-phase turbulent mixing in rod bundles," *Experimental Thermal and Fluid Science*, **5**, *1*, 26 (1992); https://dx.doi.org/10.1016/0894-1777(92)90053-8.

[23] D. S. Rowe, "Measurement of Turbulent Velocity, Intensity and Scale in Rod Bundle Flow Channels," PhD Thesis, Oregon State University (1973)URL https://ir.library.oregonstate.edu/concern/graduate_thesis_or_dissertations/3197xq109.

[24] "Watts Bar, Unit 2 - Final Safety Analysis Report (FSAR), Amendment 93, Section 4.0, Reactor," URL https://www.nrc.gov/docs/ML0914/ML091400651.pdf.

[25] H. Chelemer, J. Weisman, and L. Tong, "Subchannel thermal analysis of rod bundle cores," *Nuclear Engineering and Design*, **21**, *1*, 35 (1972); https://dx.doi.org/10.1016/0029-5493(72)90084-2.

[26] IAPWS, "Revised Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam," IAPWS R7-97 (2007).

[27] IAPWS, "Release on the IAPWS Formulation 2011 for the Thermal Conductivity of Ordinary Water Substance," IAPWS R15-11 (2011).

[28] IAPWS, "Release on the IAPWS Formulation 2008 for the Viscosity of Ordinary Water Substance," IAPWS R12-08 (2008).

[29] A. Rubin, A. Schoedel, M. Avramova, H. Utsuno, S. Bajorek, and A. Velazquez-Lozada, "OECD/NRC Benchmark Based On NUPEC PWR Subchannel And Bundle Tests (PSBT); Volume I: Experimental Database and Final Problem Specifications," , OECD/NRC (2010).

[30] M. Valette, "Analysis of Subchannel and Rod Bundle PSBT Experiments with CATHARE 3," *Science and Technology of Nuclear Installations* (2012); https://dx.doi.org/10.1155/2012/123426.

[31] W. R. Martin, "Challenges and Prospects for Whole-Core Monte Carlo Analysis," *Nuclear Engineering and Technology*, **44**, 151 (2012); https://dx.doi.org/10.5516/NET.01.2012.502.

[32] J. L. Conlin, W. Ji, J. C. Lee, and W. R. Martin, "Pseudo material construct for coupled neutronic-thermal-hydraulic analysis of VHTGR," *Trans. Am. Nucl. Soc*, **92**, 225 (2005).

[33] M. Daeubler, A. Ivanov, B. L. Sjenitzer, V. Sanchez, R. Stieglitz, and R. Macian-Juan, "High-fidelity coupled Monte Carlo neutron transport and thermal-hydraulic simulations using Serpent 2/SUBCHANFLOW," *Annals of Nuclear Energy*, **83**, 352 (2015); https://dx.doi.org/10.1016/j.anucene.2015.03.040.

[34] T. H. Trumbull, "Treatment of Nuclear Data for Transport Problems Containing Detailed Temperature Distributions," *Nuclear Technology*, **156**, *1*, 75 (2006); https://dx.doi.org/10.13182/NT156-75.

[35] P. Ducru, C. Josey, K. Dibert, V. Sobes, B. Forget, and K. Smith, "Kernel reconstruction methods for Doppler broadening — Temperature interpolation by linear combination of reference cross sections at optimally chosen temperatures," *Journal of Computational Physics*, **335**, 535 (2017); https://dx.doi.org/10.1016/j.jcp.2017.01.039.

[36] W. R. Martin, "Implementation of On-the-Fly Doppler Broadening in MCNP for Multiphysics Simulation of Nuclear Reactors," Project 10-897, DOE NEUP (2012); https://dx.doi.org/10.2172/1058919.

[37] T. Viitanen and J. Leppänen, "Explicit Treatment of Thermal Motion in Continuous-Energy Monte Carlo Tracking Routines," *Nuclear Science and Engineering*, **171**, *2*, 165 (2012); https://dx.doi.org/10.13182/NSE11-36.

[38] T. Viitanen and J. Leppänen, "Target Motion Sampling Temperature Treatment Technique with Elevated Basis Cross-Section Temperatures," *Nuclear Science and Engineering*, **177**, *1*, 77 (2014); https://dx.doi.org/10.13182/NSE13-37.

[39] B. Forget, S. Xu, and K. Smith, "Direct Doppler broadening in Monte Carlo simulations using the multipole representation," *Annals of Nuclear Energy*, **64**, 78 (2014); https://dx.doi.org/10.1016/j.anucene.2013.09.043.

[40] C. Josey, P. Ducru, B. Forget, and K. Smith, "Windowed multipole for cross section Doppler broadening," *Journal of Computational Physics*, **307**, 715 (2016); https://dx.doi.org/10.1016/j.jcp.2015.08.013.

[41] S. M. Harper, P. K. Romano, B. Forget, and K. S. Smith, "Threadsafe Dynamic Neighbor Lists for Monte Carlo Ray Tracing," *Nuclear Science and Engineering* (2020); https://dx.doi.org/10.1080/00295639.2020.1719765.

[42] H. Rief, "Generalized Monte Carlo Perturbation Algorithms for Correlated Sampling and a Second-Order Taylor Series Approach," *Annals of Nuclear Energy*, **11**, *9*, 255 (1984).

[43] D. E. Peplow and K. Verghese, "Differential Sampling for the Monte Carlo Practitioner," *Progress in Nuclear Energy*, **36**, *1*, 39 (2000).

[44] Y. Nagaya and T. Mori, "Impact of Perturbed Fission Source on the Effective Multiplication Factor in Monte Carlo Perturbation Calculations," *Nuclear Science and Technology*, **42**, *5*, 428 (2005).

[45] S. M. Harper, "Calculating Reaction Rate Derivatives in Monte Carlo Neutron Transport," Master's Thesis, Massachusetts Institute of Technology (2016)URL http://hdl.handle.net/1721.1/106690.

[46] Y. Nagaya and T. Mori, "Estimation of Sample Reactivity Worth with Differential Operator Sampling Method," *Nuclear Science and Technology*, **2**, 842 (2011).

[47] N. Horelik, B. Herman, M. Ellis, S. Kumar, J. Liang, B. Forget, and K. Smith, "Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS) rev. 2.0," , MIT CRPG (2016)URL https://crpg.mit.edu/research/beavrs.

[48] G. I. Bell and S. Glasstone, *Nuclear Reactor Theory*, Van Nostrand Reinhold Company (1970).

[49] A. Hébert, "Multigroup Neutron Transport and Diffusion Computations," D. G. Cacuci (Editor), *Handbook of Nuclear Engineering*, chap. 8, Springer; https://dx.doi.org/10.1007/978-0-387-98149-9_8.

[50] Z. Liu, K. Smith, B. Forget, and J. Ortensi, "Cumulative migration method for computing rigorous diffusion coefficients and transport cross sections from Monte Carlo," *Annals of Nuclear Energy*, **112**, 507 (2018); https://dx.doi.org/10.1016/j.anucene.2017.10.039.

[51] D. F. Gill, D. P. Griesheimer, and D. L. Aumiller, "Numerical Methods in Coupled Monte Carlo and Thermal-Hydraulic Calculations," *Nuclear Science and Engineering*, **185**, *1*, 194 (2017); https://dx.doi.org/10.13182/NSE16-3.

[52] B. R. Hanna, D. F. Gill, and D. P. Griesheimer, "Reactivity Effects of Spatial Homogenization of Thermal Feedback Regions in Monte Carlo Reactor Calculations," *Nuclear Technology*, **183**, *3*, 367 (2013); https://dx.doi.org/10.13182/NT13-A19425.

[53] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, **22**, *3*, 400 (1951).

[54] J. Dufek and W. Gudowski, "Stochastic Approximation for Monte Carlo Calculation of Steady-State Conditions in Thermal Reactors," *Nuclear Science and Engineering*, **152**, *3*, 274 (2006); https://dx.doi.org/10.13182/NSE06-2.

[55] V. Valtavirta, J. Leppänen, and T. Viitanen, "Coupled neutronics–fuel behavior calculations in steady state using the Serpent 2 Monte Carlo code," *Annals of Nuclear Energy*, **100**, 50 (2017); https://dx.doi.org/10.1016/j.anucene.2016.10.015.

[56] D. Knoll and D. Keyes, "Jacobian-free Newton–Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, **193**, *2*, 357 (2004); https://dx.doi.org/10.1016/j.jcp.2003.08.010.

[57] J. Hales, M. Tonks, F. Gleicher, B. Spencer, S. Novascone, R. Williamson, G. Pastore, and D. Perez, "Advanced multiphysics coupling for LWR fuel performance analysis," *Annals of Nuclear Energy*, **84**, 98 (2015); https://dx.doi.org/10.1016/j.anucene.2014.11.003.

[58] A. G. Mylonakis, M. Varvayanni, and N. Catsaros, "A Newton-based Jacobian-free approach for neutronic-Monte Carlo/thermal-hydraulic static coupled analysis," *Annals of Nuclear Energy*, **110**, 709 (2017); https://dx.doi.org/10.1016/j.anucene.2017.07.014.

[59] M. Ellis, B. Forget, K. Smith, and D. Gaston, "Continuous Temperature Representation In Coupled OpenMC/MOOSE Simulations," *PHYSOR 2016* (2016).

[60] D. Madland, "Total prompt energy release in the neutron-induced fission of 235U, 238U, and 239Pu," *Nuclear Physics A*, **772**, *3*, 113 (2006); https://dx.doi.org/10.1016/j.nuclphysa.2006.03.013.

[61] R. Vogt, "Energy-Dependent Fission Q Values Generalized for All Actinides," LLNL-TR-407620, Lawrence Livermore National Laboratory (2008); https://dx.doi.org/10.2172/945803., URL https://www.osti.gov/biblio/945803.

[62] J. Rhodes, K. Smith, , and Z. Xu, "CASMO-5 Energy Release per Fission Model," *International Conference on Reactor Physics* (2008).

[63] M. Chadwick, M. Herman, P. Obložinský et al., "ENDF/B-VII.1 Nuclear Data for Science and Technology: Cross Sections, Covariances, Fission Product Yields and Decay Data," *Nuclear Data Sheets*, **112**, *12*, 2887 (2011); https://dx.doi.org/10.1016/j.nds.2011.11.002., special Issue on ENDF/B-VII.1 Library.

[64] B. N. Aviles, D. J. Kelly, D. L. Aumiller, D. F. Gill, and B. W. Siebert, "Coupled MC21 And COBRA-IE Solution To Vera Core Physics Benchmark Problem #6," *PHYSOR 2016, Sun Valley, ID.* (2016).

[65] G. V. Bayley and J. M. Hammersley, "The "Effective" Number of Independent Observations in an Autocorrelated Time Series," *Supplement to the Journal of the Royal Statistical Society*, **8**, *2*, 184 (1946); https://dx.doi.org/10.2307/2983560.

[66] A. T. Godfrey, "VERA Core Physics Benchmark Progression Problem Specifications rev. 4," CASL-U-2012-0131-004, ORNL (2014).

[67] N. E. Todreas and M. S. Kazimi, *Nuclear Systems*, vol. 1, 2nd ed., CRC Press (2011).

[68] S. Z. Rouhani, "Axial and transverse momentum balance in subchannel analysis," Topical meeting on requirements and status of the prediction of physics parameters for thermal and fast power reactors; Juelich, West Germany (1973).

[69] D. S. Rowe, C. L. Wheeler, and D. E. Fitzsimmons, "Experimental study of flow and pressure in rod bundle subchannels containing blockages," BNWL-1771, Pacific Northwest Laboratory (1973); https://dx.doi.org/10.2172/4373591.

[70] J. G. Bartzis, "Axial and transverse momentum balance in subchannel analysis," PhD Thesis, Massachusetts Institute of Technology (1975)URL http://hdl.handle.net/1721.1/16431.

[71] D. L. Hagrman and G. A. Reymann (Editors), *Matpro-Version 11: A Handbook of Materials Properties for Use in the Analysis of Light Water Reactor Fule Rod Behavior* (1979); https://dx.doi.org/10.2172/6442256.

[72] K. J. Geelhood and W. G. Luscher, "FRAPCON-3.5: A Computer Code for the Calculation of Steady-State, Thermal-Mechanical Behavior of Oxide Fuel Rods for High Burnup," NUREG/CR-7022, PNNL-19418, US NRC (2014)URL https://www.nrc.gov/docs/ML1429/ML14295A539.pdf.

[73] J. C. Chen, "Correlation for Boiling Heat Transfer to Saturated Fluids in Convective Flow," *Industrial & Engineering Chemistry Process Design and Development*, **5**, *3*, 322 (1966); https://dx.doi.org/10.1021/i260019a023.

[74] T. M. Sutton, F. B. Brown, F. G. Bischoff, D. B. MacMillan, C. L. Ellis, J. T. Ward, C. T. Ballinger, D. J. Kelly, and L. Schindler, "The Physical Models and Statistical Procedures Used in the RACER Monte Carlo Code," KAPL-4840, Knolls Atomic Power Laboratory (1999); https://dx.doi.org/https://doi.org/10.2172/767449.

[75] D. She, Z. Li, Q. Xu, K. Wang, and G. Yu, "Probability-Neighbor Method of Accelerating Geometry Treatment in Reactor Monte Carlo Code RMC," *M&C 2011, Rio de Janeiro, RJ, Brazil* (2011).

[76] D. Griesheimer, Naval Nuclear Laboratory, MC21 Neighbor Lists, Personal Communication (2020).

[77] T. Donovan and L. Tyburski, "Geometric Representations in the Developmental Monte Carlo Transport Code MC21," *PHYSOR 2006, Vancouver, BC, Canada* (2006).