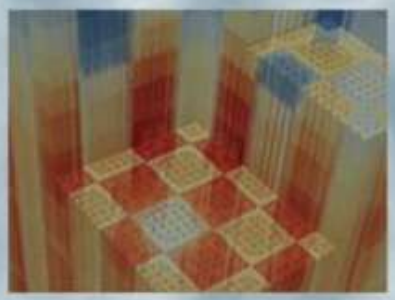
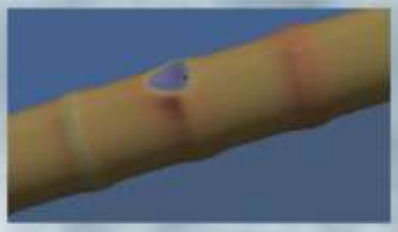
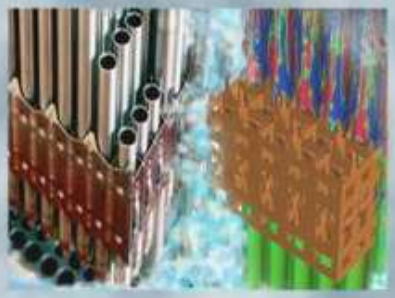
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

CASL-U-2016-1115-000







MPACT Performance Improvements in VERA-CS

Brendan Kochunas, University of Michigan
Shane Stimpson, Oak Ridge National Laboratory
Yuxuan Liu, University of Michigan
Ben Yee, University of Michigan
Ang Zhu, University of Michigan
Andrew Fitzgerald, University of Michigan
Aaron Graham, University of Michigan
Mike Jarrett, University of Michigan
Benjamin Collins, Oak Ridge National Laboratory
Kang Seog Kim, Oak Ridge National Laboratory
Kevin Clarno, Oak Ridge National Laboratory

6/30/2016

REVISION LOG

Revision	Date	Affected Pages	Revision Description
0		All	Initial Release

Document pages that are:

Export Controlled _____

IP/Proprietary/NDA Controlled_____

Sensitive Controlled_____

Requested Distribution:

To:

Copy:

EXECUTIVE SUMMARY

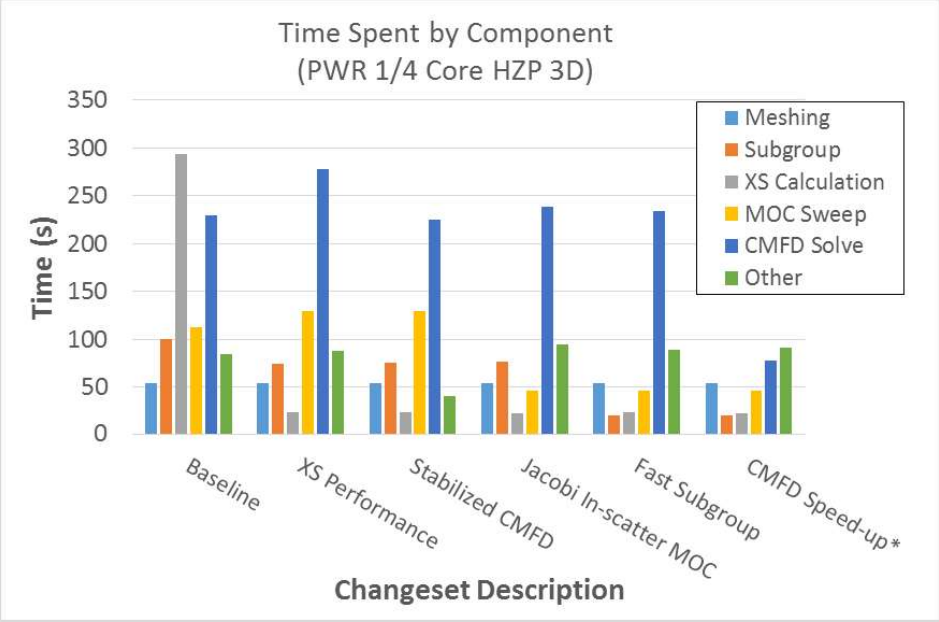
The report summarizes the initial profile of MPACT representative of the version that was distributed with VERA 3.4. It then describes several activities recently performed to increase the overall performance of the code. For the various improvements described in this report, several have been well known to the development team and reactor physics methods community for some time; while others proved novel in their combination of other existing ideas. Still, other improvements were the result of unanticipated breakthroughs in numerical methods research. These performance improvement activities focused on the following components of MPACT.

- Method of Characteristics Transport Kernel
- Iterative Methods for Coarse Mesh Finite Difference Linear System
- Macroscopic Cross Section and Effective Fission Spectrum Calculation
- Resonance self-shielding (Subgroup) calculation

A summary of the overall improvement from this work as tested on a few representative problems is provided in the table below. The incremental improvements for various changes are shown in the figure below.

Aggregate speed-up of MPACT for various test problems

Problem	Initial Time (s)	New Time (s)	Overall Speedup
PWR Lattice	41.55	13.10	3.17
PWR Subregion 2D	335.93	149.64	2.24
PWR ¼ Core-2D Depletion	11339.49	2353.52	4.82
PWR ¼ Core HZP 3D	875.07	312.40	2.80



Future work on performance improvements may focus on depletion, discretization optimization, miscellaneous improvements to the initialization, and continued improvements on the cross section calculation.

CONTENTS

FIGURES

TABLES

ACRONYMS

CASL Consortium for Advanced Simulation of Light Water Reactors
CP Challenge Problem
CZP Cold Zero Power
DOE US Department of Energy
DOE NE US Department of Energy Office of Nuclear Energy
DOE NR US Department of Energy Office of Naval Reactors
FA Focus Area
HFP Hot Full Power
HPC high-performance computing
HZP Hot Zero Power
INL Idaho National Laboratory
LANL Los Alamos National Laboratory
LWR light water reactor
MIT Massachusetts Institute of Technology
MOC method of characteristics
MPACT Michigan parallel characteristics transport code
OLCF Oak Ridge Leadership Computing Facility
ORNL Oak Ridge National Laboratory
PHI Physics Integration
PoR plan of record
PWR pressurized water reactor
QOI quantity of interest
R&D research and development
RTM Radiation Transport Methods
SNL Sandia National Laboratories
T/H thermal-hydraulics
TF 1012 floating point operations per second ("teraflop")
THM Thermal Hydraulics Methods
UM University of Michigan
VERA Virtual Environment for Reactor Applications

1. INTRODUCTION

For the past several years, MPACT has been undergoing aggressive development in CASL to meet challenge problem needs. During this time the focus of development was primarily for capability enhancement; with minimal effort being spent on usability, performance, and validation. However, with the recent successes of CASL to demonstrate simulation capability by modeling 12 cycles of Watts Bar [1], interest has grown in the CASL tools and so more analysis is being undertaken. Recommendations from this previous work also suggested improvements to run time. Given the current execution times for MPACT, breadth of analysis, and available compute resources, it was noticed that meeting the analysis goals for FY16 might be at risk given current code performance. Therefore much of the focus of FY16 has been on performance improvements. This milestone report discusses the performance improvements specifically within MPACT.

Much of the work discussed in this report is the result of supporting milestones. In many places the reader will be redirected to these reports for further details. For the various improvements described in this report, several have been well known to the development team and reactor physics methods community for some time; while others proved novel in their combination of other existing ideas. Still, other improvements were the result of unanticipated breakthroughs in numerical methods research. The objective of this report is to provide a complete and consistent description of the various activities completed thus far to improve the performance of MPACT.

We note that in the development of any software there is an inherent lifecycle. This lifecycle, if formally defined, describes policies for development activities as they relate to maturity. The lifecycle for MPACT is not formally defined within a CASL report, however, a generic lifecycle model for scientific computing software has been developed in [2], and is consequently used in TriBITS development. Using this model description as a guide one may note that MPACT is moving through its natural lifecycle and maturing. Since much of the required capability has been implemented, development efforts have now shifted focus. Performance of the code has come into the spotlight and based on feedback from users, CASL leadership, and science council members, code performance was determined to be the next most important area for improvement. However, numerous activities in other areas such as validation and usability have continued. Following the performance improvement activities, the development focus will again shift, and begin to focus more on usability and improved accuracy.

2. INITIAL PERFORMANCE PROFILE OF MPACT

Before embarking on the effort to optimize MPACT, some detailed profiling was performed to assess which areas of the codes were the relative “hot spots”. Naturally effort is more effectively spent focusing on those areas of the code that take the most time. To evaluate the performance of MPACT a series of typical problems were used to consistently quantify the performance of the code. The problems used throughout this report to measure the performance of MPACT are described in section 2.1. In section 2.2, a detailed initial profile is presented based on a subset of the problems described in section 2.1.

2.1 Description of Test Problems

The majority of the test problems for which results are presented in this document are described in detail in other references [3], [4], [5]. The purpose of this section is to briefly show what problems were used in testing and to provide some explanation as to the choice of problems and purpose of each.

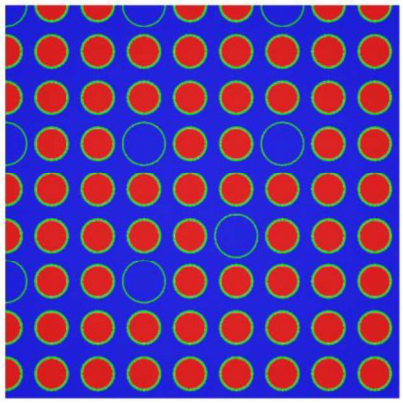


Figure 1. VERA problem 2a geometry

The VERA problem 2a, shown in Figure 1, is a 2D quarter assembly of a PWR. It generally represents the smallest spatial subdomain represented in MPACT. This is a useful problem for performance analysis because its small size generally means a several states can be run (e.g. fuel depletion) in a reasonable amount of time. Furthermore if something is shown to demonstrate serial performance improvements on this model, then the improvements generally scale proportionally with the domain size to larger problems.

The VERA problem 4a-2D, shown in Figure 2, is a 2D sub-region of a PWR that represents a quarter of a 3x3 block of assemblies. This is a useful problem for performance analysis because it introduces a little more non-uniformity compared to problem 2a by incorporating multiple fuel enrichments and burnable poisons. It is also of a relatively small size problem that still facilitates being able to run many states in a reasonable amount of time. Comparisons with results from problem 2a generally provide strong indicators of whether or not performance improvements will scale with the domain.

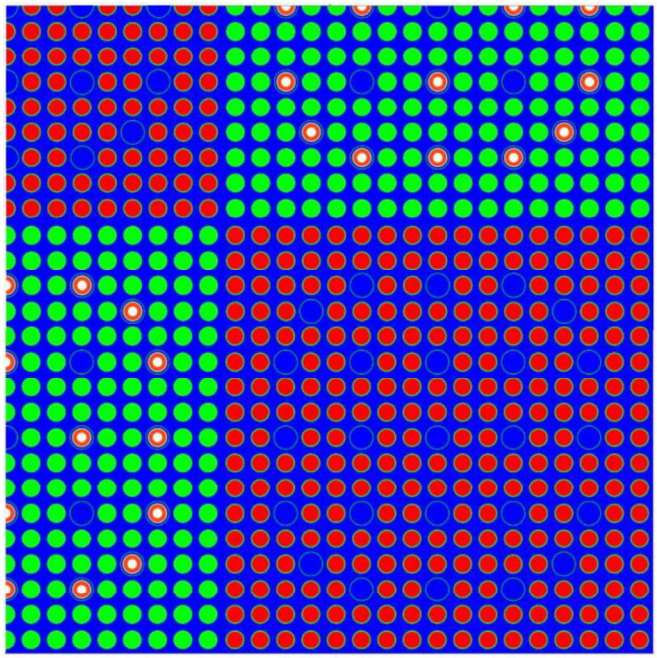


Figure 2. VERA problem 4a-2D geometry

The VERA problem 5a-2D, shown in Figure 3, is a 2D quarter core PWR with a baffle and small radial reflector. This problem is useful because it represents one plane of 3D model, which is effectively the maximum domain size for the 2D MOC problem. It provides estimates for the maximum overhead when run on a single processor, and also facilitates investigation into the parallel scaling, in particular for the MOC. It is also reasonably small, thus it does also provide some insight that is useful to assess the ability of the 3D model to compute multiple states. The 3D model is also used frequently in total profiles.

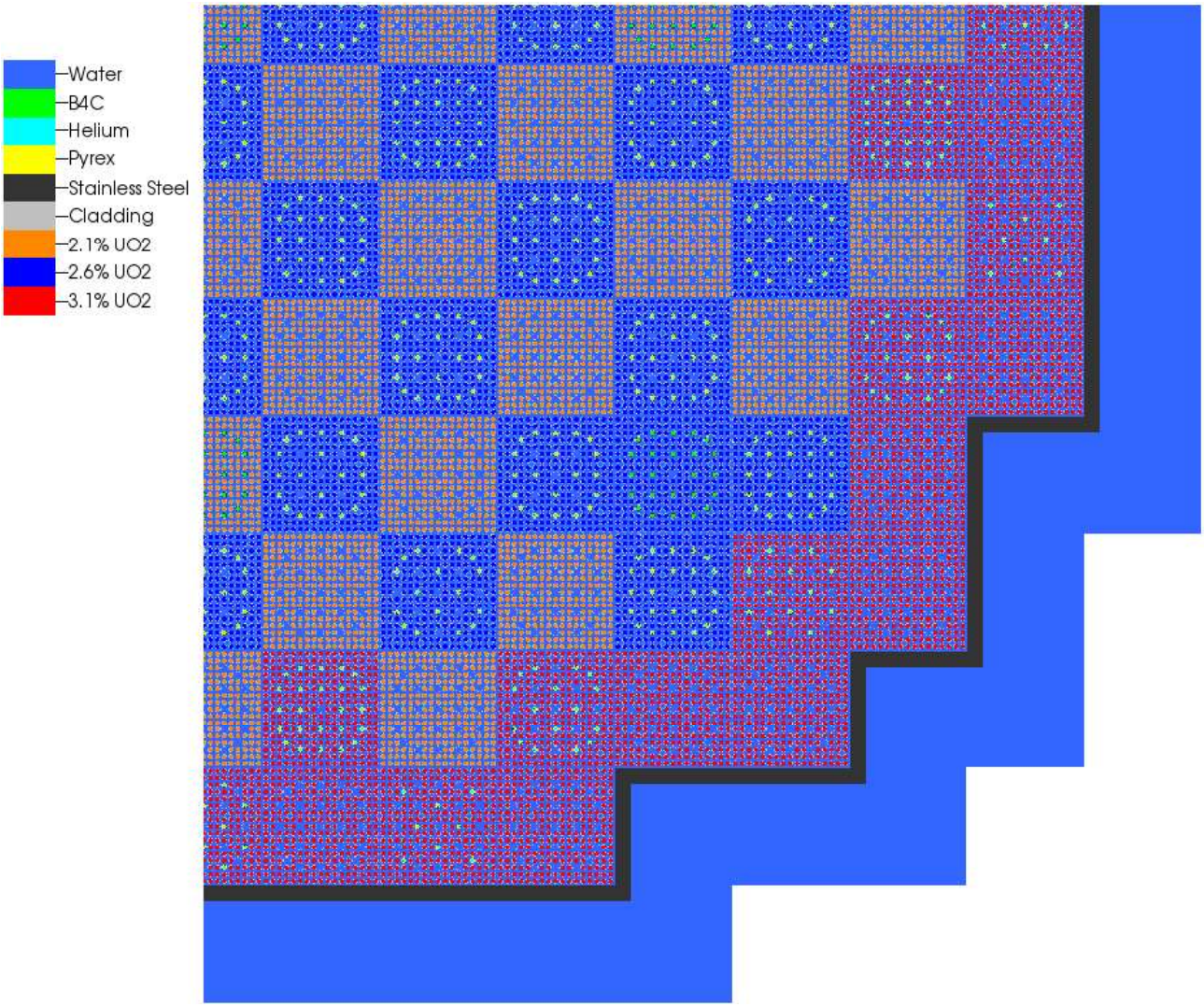
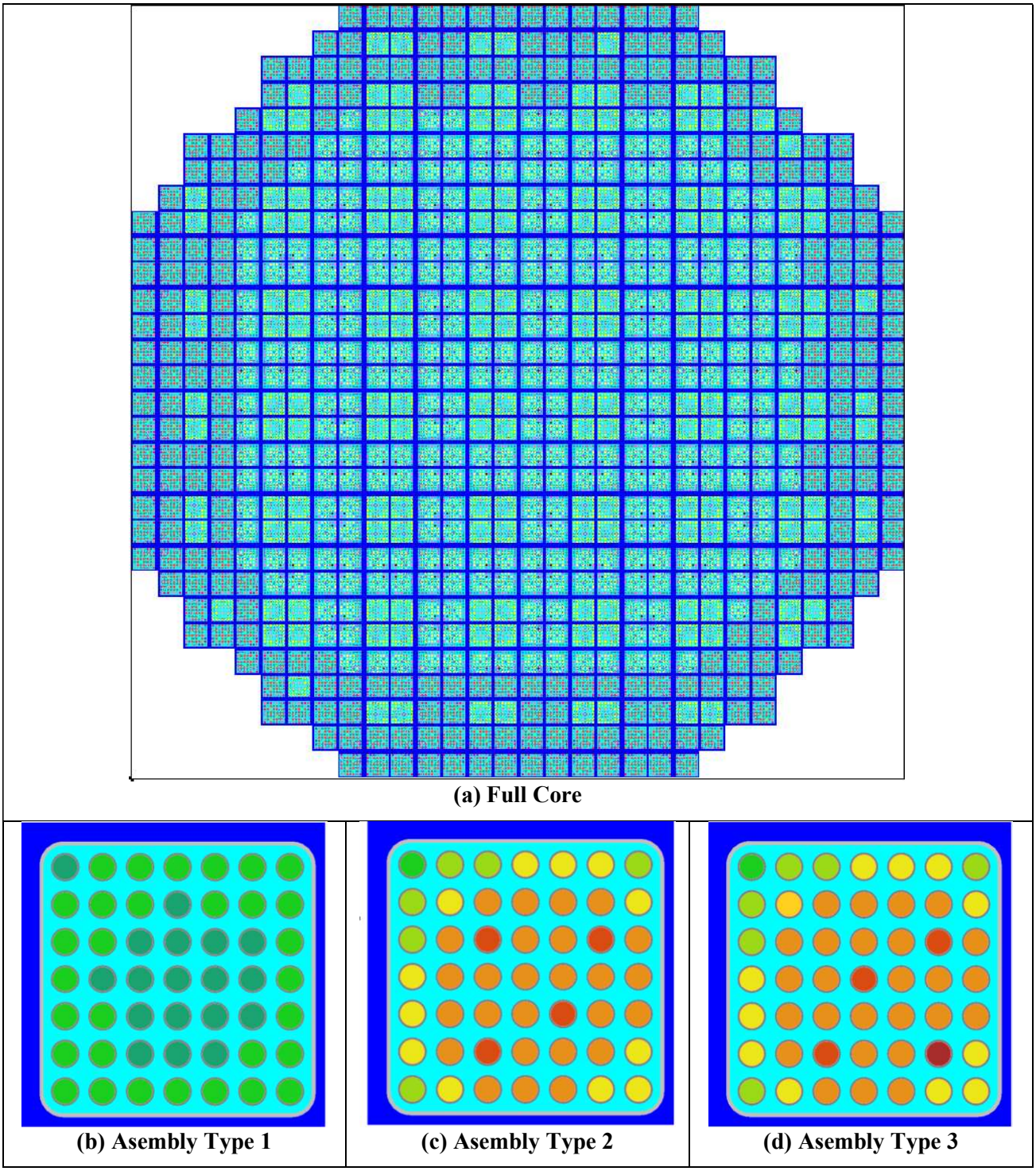


Figure 3. VERA problem 5a-2D geometry



(a) Full Core

(b) Asembly Type 1

(c) Asembly Type 2

(d) Asembly Type 3

Figure 4. Peach Bottom unit 2 geometry

The Peach Bottom model represents one of the most difficult models. It is a full core BWR in 2D. However, it is lacking the radial reflector, baffle, etc. The BWR assembly geometry is a bit more heterogeneous than the PWR problem, and the addition of the larger inter assembly gaps and fuel cans make this problem more challenging for the CMFD problem discussed in Section 3.

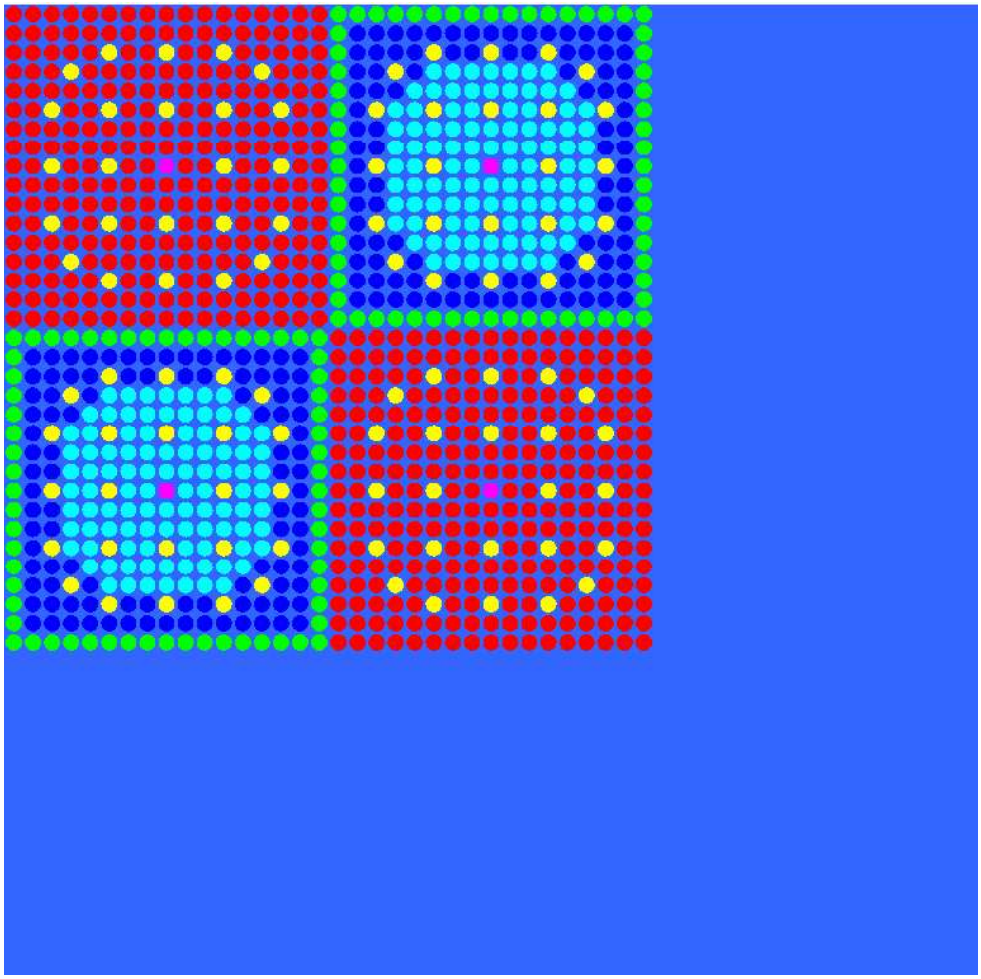


Figure 5. C5G7 2D geometry

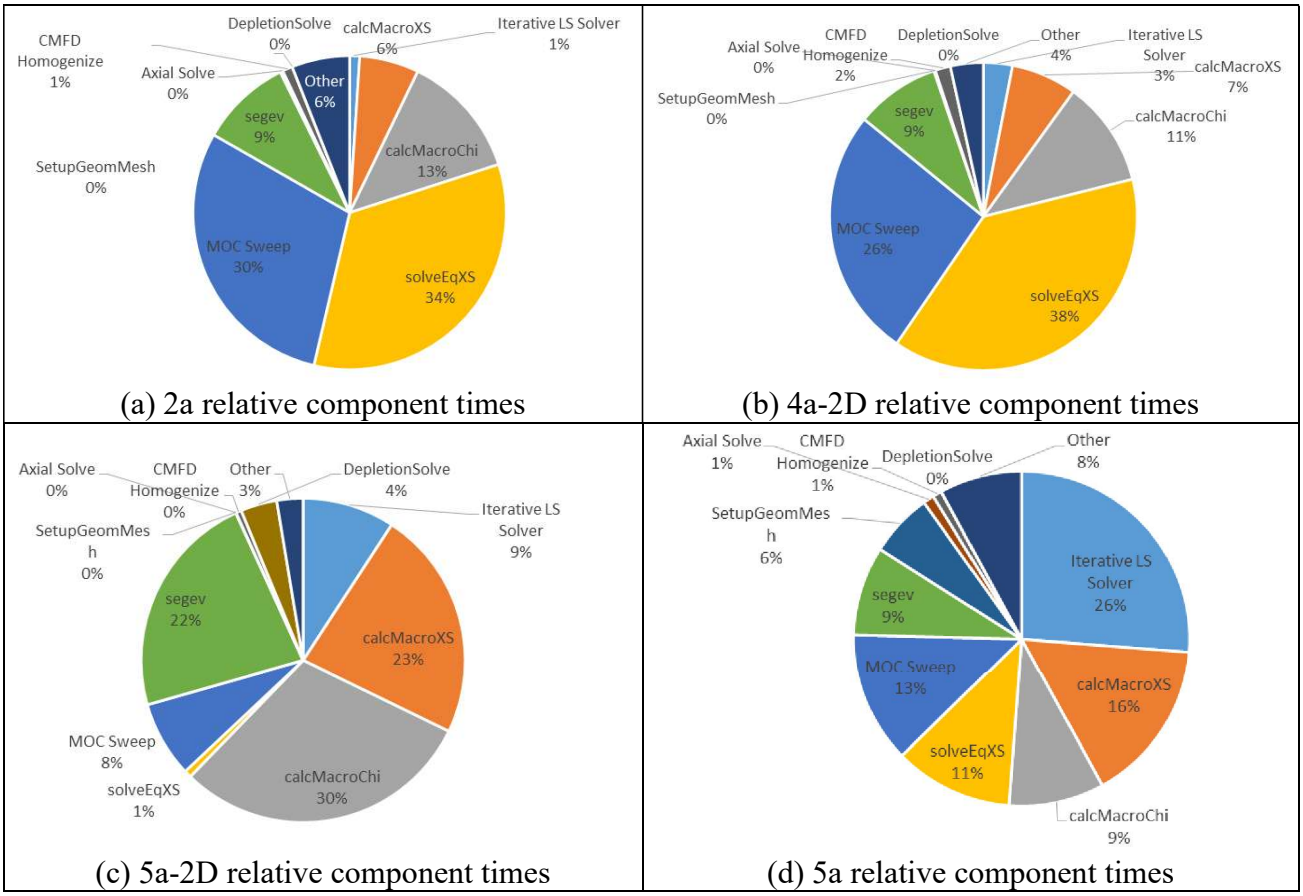
The C5G7 model is a public international benchmark. Therefore, data is available for comparison of results with other codes. The problem also poses some challenges to the stability and convergence of the iteration scheme, in particular when CMFD is used.

2.2 Initial Profiles of Test Problems

The initial profiling was performed using the Eos [6] cluster at the OLCF. It used a development version of MPACT from Feb. 2016 that was very close to v2.1.0 released with VERA-v3.4. Some instrumentation was added to the code to obtain profiling data. The main problems run here were VERA Problem 2a (with 1 process), VERA Problem 4a-2D (with 1 process), VERA problem 5a-2D (with 73 processes) and the 3D version of 5a (run with 4234 processes). In the 3D problem 5, critical boron search was added since this is more representative how the steady-state problems are being solved for practical applications. A variant for 5a-2D using depletion steps at a total of 10 states was also simulated. The total wall time for each problem is given in Table 1. Figure 6 shows the run times for various components of the code.

Table 1. Initial profile times

Problem	Total Wall Time (s)	CPU-hours
2a	41.5	0.01
4a-2D	335.9	0.09
5a-2D w/ depletion	11339.5	229.94
5a w/ critical boron	875.1	1029.17



(a) 2a relative component times

(b) 4a-2D relative component times

(c) 5a-2D relative component times

(d) 5a relative component times

Figure 6. Initial profile relative component run times

For the initial profiles instrumentation was added for the main computational components. These are described as follows.

- *Iterative LS Solver*: this component represents the time spent iterating on the linear system defined for the multi-group CMFD problem. It is the GMRES solver in PETSc with a block ILU(0) preconditioner.
- *calcMacroXS*: this component represents the time spent computing the macroscopic cross sections for all energy groups. This routine also applies the resonance self-shielding factors
- *calcMacroChi*: this component represents the time spent computing effective fission spectrum which is weighted by the fission source
- *solveEqXS*: this component represents the time spent solving the subgroup fixed source problem.
- *MOC sweep*: this component represents the time spent evaluating the MOC equations.

- *segev*: this component represents the time spent performing Segev interpolation.
- *SetupGeomMesh*: this component represents the time spent meshing the geometry
- *CMFD Homogenize*: this component represents the time spent computing the coefficients for the CMFD problem
- *Axial Solve*: this component represents the time performing the axial calculations for the 2D/1D solver
- *Depletion solve*: this component represents the time spent performing the point depletion solve
- *Other*: this is the remainder of the execution time not explicitly accounted for in the other components. This includes other parts of the initialization and editing of output.

From the data in Figure 6, several observations can be made. It is observed that the time spent solving the CMFD problem is negligible for small problems, but as the problems get larger and larger, the linear system becomes more and more difficult to solve. For smaller problems the time spent solving the subgroup fixed source problem tends to dominate. With depletion, this time is amortized over multiple state-points and many more iterations, so it appears as a smaller overall fraction. The time spent computing the cross sections is a significant part of the run time in all cases. The axial solve and CMFD homogenization are never significant contributors to the overall run time. The MOC time is a significant portion of the run time in smaller problems, but with larger problems, that utilize a high degree of parallelism, the MOC takes less of the overall time. With depletion when there are more nuclides in the fuel, the time spent computing the cross sections dominates by far. The time spent actually performing depletion is relatively small in problem 5a-2D.

This profile provides an initial picture of where the time is spent in MPACT as its executing. As it can be seen from this data, general statements about which component is taking the most time are not easily made. The component dominating the run time typically varies depending on the problem and the parallelism, although the effect of parallelism is not explicitly shown in Figure 6.

The following sections will systematically address the various components identified in from the initial profiling. Section 3 describes improvements made to the MOC sweep, Section 4 describes improvements made to the CMFD solve, and Section 5 describes improvements made to the cross section calculations.

3. MOC PERFORMANCE IMPROVEMENTS

The method of characteristics transport solution in general should be the most expensive part of any neutron transport calculation. Although, from the initial profiling results in Section 2.2, this was not always the case. Compared to other published MOC implementations it became known that approximately a factor of two in speed-up could be easily achieved by modifying the iteration scheme. This modification is discussed in Section 3.1 and represents the key factor in achieving the performance improvements reported in this section. Additional optimizations were investigated and are discussed in Section 3.2.

The other investigations discussed in this section have a more long term application. CASL uses many HPC resources, and the next evolution of this hardware is on the horizon. Many of the new HPC machines will include some heterogeneous architecture involving graphics processing units (GPUs) from NVIDIA or Intel's many-integrated-core (MIC) architecture. Additional research into developing MOC kernels for these architectures is the focus of the final section 3.3.

3.1 Jacobi In-scatter Source Iteration

Much of the description in this section comes from reference [7].

3.1.1 Algorithm Description

The traditional iteration scheme in MPACT for the multi-group transport problem has used a Gauss-Seidel iteration in energy. This iteration scheme is shown in Figure 7.

```
1: Loop over groups (g from 1 to G)
2:   Setup 1-group source for group g (fission + in-scatter)
3:   Loop over inner iterations (i from 1 to N_inners)
4:       Update self-scattering source
5:       Loop over azimuthal angles (a from 1 to N_azi)
6:           Loop over longrays in angle a (l from 1 to N_ray(a))
7:               Re-order modular ray data into flat arrays.
8:               Evaluate exponential tables for optical thickness.
9:               Loop over polar angles (p from 1 to N_pol)
10:                  Get incoming angular flux boundary condition
11:                  Loop over segments (s from 1 to N_seg(l))
12:                      Evaluate MOC equations for forward direction
13:                      Evaluate MOC equations for backward direction
14:                  Store outgoing angular flux boundary condition
15:              Loop over coarse mesh intersections (c from 1 to N_cm(l))
16:                  Accumulate surface flux and net current from ray l
17:     Reduce partials sum for scalar flux
18:     Compute scalar flux
```

Figure 7. Gauss-Seidel in-scatter source iteration

This iteration scheme has a few advantages, namely, it provides for faster convergence of the multi-group scattering source, and it requires minimal storage since the storage must only account for a 1-group problem. However, for the MOC it does have some drawbacks. The most obvious is that the ray and segment information is independent of the energy group, but this data is moved into and out of memory for every iteration over energy.

It has been known in industry for some time that a more computationally efficient algorithm to perform an MOC sweep is to use a Jacobi iteration in energy. This iteration scheme is shown in Figure 8.

```
1: Setup 1-group source for all groups (fission + in-scatter)
2: Loop over inner iterations (i from 1 to N_inners)
3:     Update self-scattering source
4:     Loop over azimuthal angles (a from 1 to N_azi)
5:         Loop over longrays in angle a (l from 1 to N_ray(a))
6:             Re-order modular ray data into flat arrays.
7:             Evaluate exponential tables for optical thickness.
8:             Loop over polar angles (p from 1 to N_pol)
9:                 Get incoming angular flux boundary condition
10:                Loop over segments (s from 1 to N_seg(l))
11:                    Loop over groups (g from 1 to G)
12:                        Evaluate MOC equations for forward direction
13:                        Evaluate MOC equations for backward direction
14:                Store outgoing angular flux boundary condition
15:                Loop over coarse mesh intersections (c from 1 to N_cm(l))
16:                    Accumulate surface flux and net current from ray l
17:    Reduce partial sums for scalar flux
18:    Compute scalar flux
```

Figure 8. Jacobi in-scatter source iteration

The efficiency of this algorithm and its associated speedup became more widely known when it was published [8]. The reason for the efficiency arises from two reasons. The first is that the loop over energy groups can be vectorized extremely efficiently, and second it amortizes the cost of accessing the ray tracing for all groups to effectively one access. However, this speed-up does not come without cost. Since the iteration scheme is modified many of the data structures for the transport sweep now increase by a factor of the number of groups. Presently MPACT uses a 47-group library which keeps this memory overhead manageable as will be seen in the following results. Although, if different group structures involving an order of magnitude or more groups are used, then the memory overhead from the Jacobi in-scatter source iteration will likely become prohibitive.

3.1.2 Numerical Results

In this section several of the test problems described in Section 1 are used to evaluate the performance of the Jacobi In-scatter source iteration. The problems tested include:

- a single quarter assembly lattice (VERA Problem 2a, Figure 1)
- a 2D slice of the 3x3 assembly cluster (VERA Problem 4a-2D, Figure 2)
- a 2D quarter core model (VERA Problem 5a-2D core layout in Figure 3)

All cases used a 0.05 cm ray spacing, 16 azimuthal angles per octant, and 2 polar angles in a Tabuchi-Yamamoto [9] quadrature with 3 radial rings in the fuel and 8 azimuthal divisions. Additionally, all cases used a 47-group cross section library generated by ORNL [10].

Two different machines were used to analyze the first two problems, one with AMD processors (Opteron™ Processor 6376, 2.3 GHz) and the other with Intel® (Xeon® CPU E5-1650v3, 3.2 GHz). Below are the results for Problem 2a (Table 2), showing the timing (in cycles per integration) for two kernels (one with and the other without current tallies). The number of integrations is determined by tallying the number of ray segments (across all angles and long-rays), multiplying by the number of groups and polar angles. There is also an additional 2x multiplier to account for the fact that forward and backward directions are swept simultaneously. For example, this case had 715,675 ray tracing segments or a total of ~134.5 million integrations). The timing values that are reported were averaged over all kernel instantiations over 10 iterations. Both 2a and 4a-2D were run on only one process.

On the AMD machine, modest performance improvements are observed in the kernel without currents (1.4x speedup), but the kernel with currents yields a speedup of over 2.7x. On the Intel machine, the speed ups are a bit lower, though still substantial.

Table 2. Comparison of Jacobi and Gauss-Seidel for problem 2a

	AMD (2.3 GHZ)			Intel (3.2 GHz)		
	cycles/integration		Ratio	cycles/integration		Ratio
	Gauss-Seidel	Jacobi		Gauss-Seidel	Jacobi	
Kernel w/ Currents	36.0	13.3	2.71	21.5	10.4	2.02
Kernel w/o Currents	17.0	12.4	1.37	11.3	9.3	1.21

Similar results are observed on Problem 4a-2D, which is not very surprising (Table 3). The case had 6.54 million ray tracing segments (1.23 billion integrations), which is slightly more than 9x the segments in Problem 2a because of the Pyrex inserts in the 2.6% enriched assemblies.

Table 3. Comparison of Jacobi and Gauss-Seidel for problem 4a-2D

	AMD (2.3 GHZ)			Intel (3.2 GHz)		
	cycles/integration		Ratio	cycles/integration		Ratio
	Gauss-Seidel	Jacobi		Gauss-Seidel	Jacobi	
Kernel w/ Currents	34.0	13.6	2.51	22.5	11.9	1.89
Kernel w/o Currents	17.5	12.3	1.43	12.6	10.8	1.18

5a-2D was run on the Titan supercomputer [11], which each 16-core node contains 2.2 GHz AMD Opteron™ 6274 (Interlagos) processors, using 1, 73, and 257 spatial parallelization processors. To determine if there are any differences in the micro-architectures the same 5a-2D cases were run on the Eos compute cluster which has 2.6 GHz Intel Xeon E5-2670 processors. Table 4 shows the clock cycles per integration. Since the goal with the Jacobi sweeper is to use only 1 inner iteration, the results here are basically a comparison of the performance of the kernels for a variety of domain sizes. Additionally, because the Gauss-Seidel algorithm demonstrates issues with TCP_0 when using only 1 inner iteration, P_0 scattering was used to ensure stability, while still enabling an equal comparison. It is also important to note that the Jacobi iteration strategy generally requires a few more iterations to achieve the same level of convergence as the Gauss-Seidel solver.

In Table 4 it is observed that the cycles/integration are not very sensitive to the problem size; the numbers are similar to those in Table 2 and Table 3. This trend indicates the implementation is effectively utilizing the conventional memory hierarchy. The other trend observed in the results in Table 4 show that the cycles per integration decrease rapidly as the number of parallel domains increases. This is indicative the strong scaling of the MOC kernel. Comparing the parallel cases with the serial case can be used to compute the parallel efficiency. In this problem the efficiency is around 70% on the AMD chip, and 50% on the Intel chip, and slightly better for the Jacobi sweep algorithm.

Table 4. Comparison of Jacobi and Gauss-Seidel for problem 5a-2D

MOC Kernel	Spatial Domains	AMD (2.3 GHZ)			Intel (3.2 GHz)		
		cycles/integration		Ratio	cycles/integration		Ratio
		Gauss-Seidel	Jacobi		Gauss-Seidel	Jacobi	
w/ Currents	1	40.6	20.1	2.0	22.7	10.2	2.2
	73	0.88	0.41	2.2	0.60	0.35	1.7
	257	0.23	0.10	2.2	0.15	0.09	1.7
w/o Currents	1	30.7	17.3	1.8	11.4	8.7	1.3
	73	0.48	0.33	1.5	0.37	0.27	1.3
	257	0.12	0.08	1.4	0.09	0.07	1.4

However, these notable speedups do incur some cost in terms of the memory burden. Table 5 shows the total memory required for each of the problems presented (in gigabytes, GB). With the Gauss-Seidel sweep algorithm, only one group of source data is necessary at a time and only the incoming angular flux needs to be stored for all groups. For the Jacobi sweep algorithm, though, the source data for all groups needs to be set up at once and both the incoming and outgoing angular flux variables need to be stored for all groups. As a result, the Jacobi sweep algorithm requires about 10% more overall storage than the Gauss-Seidel sweep. For our target applications, this is likely acceptable, but something to consider when deploying to new machines, or as spatial domain sizes increase in some applications. It is also worth noting that these memory requirements were collected where the subgroup fixed source solver and eigenvalue transport solvers are completely separate, so two transport sweepers are initialized.

Table 5. Total memory requirements (GB) for test problems

Problem	Gauss-Seidel	Jacobi	Ratio
2a	0.215	0.236	1.10
4a-2D	0.865	0.938	1.08
5a-2D (1 domain)	12.434	13.459	1.08
5a-2D (73 domains)	23.527	25.929	1.10
5a-2D (257 domains)	37.221	40.737	1.09

3.1.3 Effect on Stability and convergence

By changing the iteration scheme there are some very important effects to consider with regard to stability of the overall iteration scheme and the rate of convergence. In general Jacobi is well known not to converge as quickly as Gauss-Seidel. Therefore, the expectation of switching from Gauss-Seidel to Jacobi is to observe potentially more iterations to meet the convergence criteria. However, this is not necessarily observed when using CMFD acceleration.

Another, somewhat unexpected outcome from switching iteration schemes was increased stability of the iterations when transport corrected P0 cross sections are used. The transport correction [12] modifies the total cross section of H-1 to account for P1 scattering; to preserve balance, another modification is required. Typically, to preserve balance the self-scattering cross section is adjusted by the same factor as the total cross section. This adjustment often results in a negative self-scattering cross section, and in particular this problem worsens with refinement in the energy discretization. Because the TCP0 approximation frequently introduces a negative self-scatter cross section, the scattering ratio ($\Sigma_{s0,g\to g}/\Sigma_{t,g}$), is no longer guaranteed positive. Thus, even the conventional source iteration method becomes conditionally stable. Some work has been done to identify ways to stabilize iteration schemes using TCP0, but this is still largely an open problem.

In practice, instability is frequently observed with the TCP0 approximation in problems that have large water reflectors when the Gauss-Seidel iteration scheme is used. However, when using the Jacobi iteration scheme, the iterations are observed to be more stable in some cases. No clear theoretical explanation currently exists for why this behavior is observed, and this is an area of ongoing research. The DIMPLE benchmark [13], demonstrates the behavior of being unstable with Gauss-Seidel and stable with Jacobi.

3.2 Miscellaneous Performance Improvements

This section looks at additional techniques to further improve the performance of MOC kernel represented by Figure 8. In particular, the operations encompassed by lines (or steps) 6, 7, 12, 13, and 16-19 in Figure 8 were explored in detail. Prior to this work the various components of the kernel were profiled in a differential fashion to estimate their relative fractions of execution time. The relative run times for a single PWR lattice based on problem 2a are given in Table 6. The line numbers given in the table correspond to those in Figure 8.

Table 6. MOC kernel component relative execution times for problem 2a

Label	Lines	[1]Time (s)	Fraction
Total	4-18	18.518	100%
Longray loop	5-16	18.073	98%
[2]Re-order Modular Ray	6	1.955	11%
[2]Exponential Table Eval.	7	4.973	27%
[2]MOC Equation Eval.	9-14	11.417	62%
[2]Coarse Mesh Current Calc.	15-16	1.574	8%
Scalar Flux calc	17-18	0.300	2%
1 Times are not corrected for measurement overhead. Therefore, fractions are estimates and sum may not equal total. 2 This component is included in the "Longray loop" time.			

From this limited profile it is observed that the majority of the time of the kernel is spent evaluating the MOC equations (62%), and currents (8%). The next largest fractions belong to evaluating the exponential tables and re-ordering the ray data, respectively. These areas represent potential room for improvement. One change in particular that has been previously suggested is to store the long ray ordered ray tracing data so that step 6 may be completely eliminated. Looking at the values in Table 6, we can conclude that the theoretical limit to these speed-up would be to reduce the execution time of the kernel by 11%. Naturally, achieving this theoretical limit this should not be realistically expected, since some operations on the data are still needed. Also, from examining Table 6, the data

suggests that reducing the time spent evaluating the exponential tables could potentially provide more savings than the long ray data.

Preliminary refactoring of the ray tracing data structures was performed to eliminate line 6 from Figure 8. The changes in performance based on this refactor are summarized in Table 7. The measurements were performed on the Flux cluster at UM. The nodes on the cluster used for this work contain two 2.5 GHz Intel Xeon E5-2680 processors, each having 12 cores.

Table 7. Longray refactor performance data

Test Problem	Nominal		Refactored		Incremental Change	
	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Speed-up	Usage
2a (full symmetry)	18.518	477.9	17.866	564.9	3.6%	+18.2%
4a-2D	44.151	1061.5	43.02	1241.5	2.6%	+17.0%
5a-2D	1295	14401.6	1245	18793.0	4.0%	+30.5%

These results show a significant memory increase for only a marginal speed-up so it is not necessarily recommended. However, the memory burden could be mitigated by reducing the precision of the segment data from double precision to single precision. To assess the accuracy of this modification, the MPACT regression suite was run with single precision segment lengths. The results are summarized in the figure below.

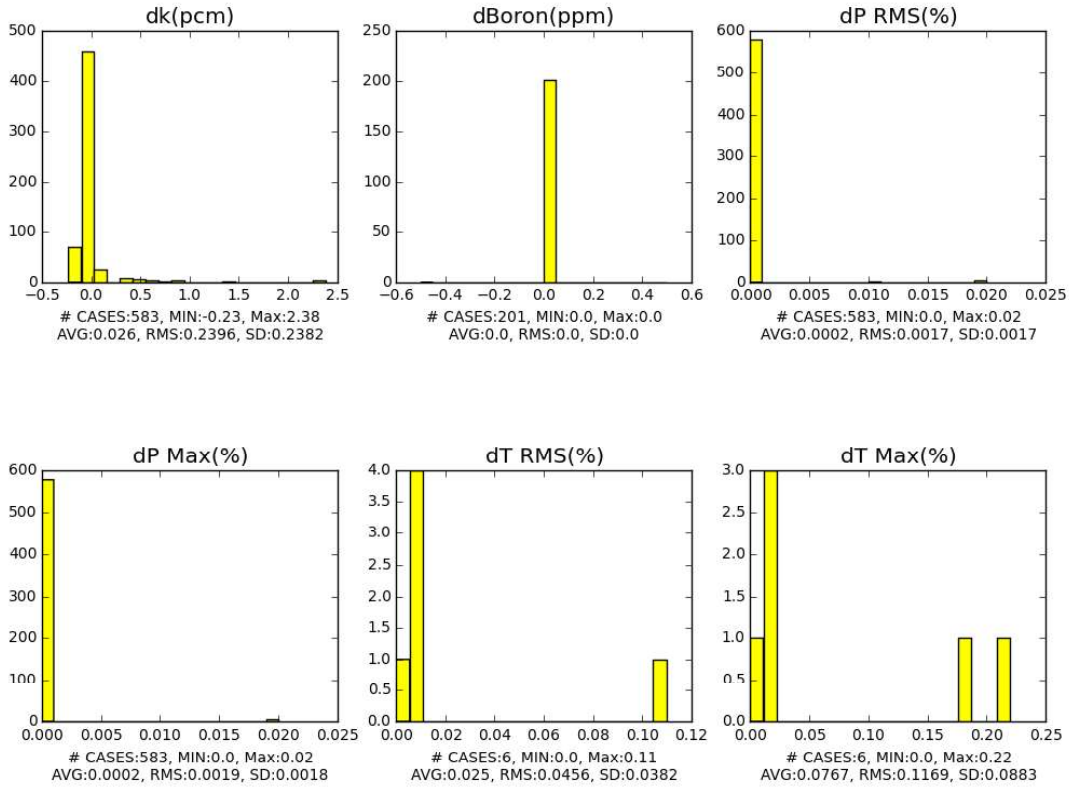


Figure 9. Regression suite differences for single precision segment data

The results of Figure 9 show that except for a few cases the change in Δkeff for the +400 tests is between -0.23 and +2.38 pcm, with an average of 0.03 pcm. The effect on power is similarly negligible being less than 0.02% for the max and RMS. Therefore, the precision of the segment length data can be safely reduced from double precision to single precision. Part of the reason this effect is so negligible is because of the use of the exponential tables, which introduce truncation error in the evaluation of the exponential on the order of the epsilon for single precision floating point variables. By reducing the precision of the segment data, the performance comparisons can be improved to those in Table 8.

Table 8. Longray refactor performance data for single precision segments

Test Problem	Nominal		Refactored		Incremental Change	
	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Speed-up	Usage
2a (full symmetry)	18.518	477.9	17.835	543.2	3.8%	+13.6%
4a-2D	44.151	1061.5	42.503	1191.6	3.9%	+12.3%
5a-2D	1295	14401.6	1224	17525.8	5.8%	+22%

The implementation of the Jacobi in-scatter kernel was also modified and investigated for its performance with threading for shared memory parallelism. The threading is implemented with OpenMP directives and the loofp over long rays is parallelized. The scaling was tested on the thread scaling of the longray loop. The results are reported for problems 2a, 4a-2D and 5a-2D in Table 9 below.

Table 9. MOC shared memory parallel efficiency

Threads	2a (full symmetry)		4a-2D		5a-2D	
	Nominal	Refactored	Nominal	Refactored	Nominal	Refactored
2	95%	95%	93%	95%	91%	95%
4	83%	87%	82%	89%	79%	86%
8	61%	70%	59%	68%	52%	55%
12	45%	57%	37%	47%	N/A	
16	33%	46%	25%	33%		
24	15%	22%	13%	19%		

The improvements to the kernel provide slightly better parallel efficiency for the thread scaling. Essentially there is an improvement in efficiency from 5% to 10% for 4 threads or greater. This means that it would now be feasible for users to use up to 4 (or even 8) threads efficiently, where the previous implementation was not really attractive beyond 2 (or in some cases 4) threads. Presently the threading primarily helps the transport kernels and not the other kernels (e.g. CMFD, macroscopic cross section calculation, and depletion). Therefore, the overall speed-up in wall time is not as good as the MOC. Future work, should focus on adding threading to these other kernels to allow the code to take better advantage of shared memory parallelism. The present alternative to the shared memory parallelism is the angle decomposition, which is better utilized by the CMFD, but not any of the other kernels. The drawback of the angle decomposition is the extremely high memory overhead from duplicating spatial domains.

3.3 Investigation of Advanced Architectures

3.3.1 MOC kernel on Intel many-integrated-core

The Intel many-integrated-core (MIC) architecture is shown for their first generation Xeon Phi coprocessor in Figure 10. The primary production machine utilizing this architecture within the U.S. is the Stampeded cluster at the Texas Advanced Computing Center (TACC) [14]. This work was performed as a part of class project in 2015.

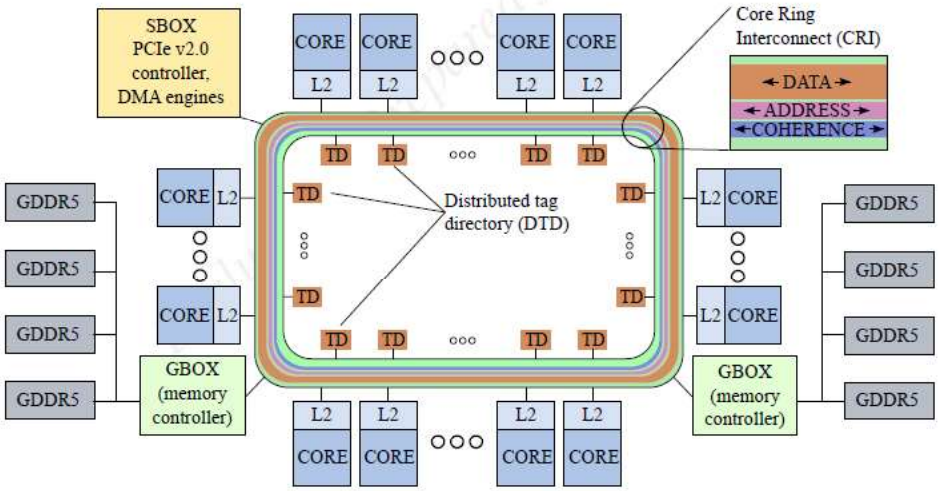


Figure 10. Intel MIC architecture (Knight's Corner)

The Xeon Phi has 61 cores that are more "lightweight" than the conventional CPU. However, unlike the programming models for the GPU, the Xeon Phi coprocessor runs the same x86 architecture as the host processor. Each core comes with its own dedicated L1 and L2 cache and has 4 hardware threads. The processors also have 512-bit wide registers allowing for up to 8 floating point operations per cycle. The cores on the Xeon Phi are connected by a ring bus that facilitates communication of application data between the cores. Memory traffic over the bus can become a bottleneck as the latency and bandwidth for point to point communication does depend on distance between the cores on the bus. Similar to other architecture models with a host processor and a coprocessor, the data transfer from the host to the coprocessor occurs over the PCIe bus

From a programming perspective, the Intel MIC, is a bit more non-uniform than the conventional symmetric multi-processor (SMP). However, the support for the x86 instructions simplifies the programming model greatly compared to the GPU. The MIC supports the following executi models:

- *multi-core hosted*: program executes on the host, with some operations executed on the coprocessor
- *offload*: target specific highly parallel routines to be run on the coprocessor only
- *symmetric*: employ the host and coprocessor on a nearly equal basis
- *manycore hosted*: executing exclusively on the coprocessor

All of these models are effectively controlled by compiler directives in the source code and compiler options. Very little, if any, modification is required for the source code. However, the transferring data to the MIC does have some restriction. Effectively, any variable that is a derived type with allocatable components cannot be transferred. This does not limit the use of any intrinsic data type,

unfortunately, in MPACT several key data structures in the computational kernels fall into the category of those that cannot be transferred to the coprocessor. Therefore, the work performed in this section used a mini-app that emulated the MOC kernel in MPACT.

Several implementations of the Jacobi-Inscatter MOC algorithm described in Figure 8 were developed for the Intel MIC. First was a naive offload approach that simply took the MOC kernel and had it run on the MIC. The next approach was more general and used what is typically referred to as the MPI+X model. This model is already utilized in MPACT with MPI+OpenMP, but now OpenMP is replaced by the Xeon Phi. This model provides more flexibility so that both the symmetric application MIC hosted applications can be evaluated with the same executable.

The preliminary results actually focus on a single test problem that is representative of a 4x4x4 block of pin cells and for a 3D MOC kernel, rather than 2D. The results are given in Table 10.

Table 10. 3D MOC sweep performance on Intel MIC

Case	Sweep Time (s)	Speed-up
1x1 Host	269	---
1x16 Host	24.7	10.9
Offload	67.7	4.0
8x30 MIC	24.8	10.9
8x2 Host + 9x25 MIC	14.3	18.8
16x1 Host + 17x14 MIC	14.5	18.6
4x4 Host + 17x10 MIC	16.3	16.5
<MPI Proc> x <Threads>		

3.3.2 Investigation of MOC kernel on GPU

The GPU architecture for the NVIDIA Kepler K20X, that is the accelerator on Titan, is shown in Figure 11. This work was performed on Titan at the OLCF as a part of a class project in 2014. The GPU architecture consists of a more rigid hierarchy that reflects the hardware structure. The GPU consists of 14 streaming multiprocessors (SMX). Each multiprocessor has 192 CUDA cores, that are single precision and integer arithmetic cores. They also have 64 double precision cores and 32 load/store units. Therefore, three CUDA cores share one double precision arithmetic unit and 6 cores share a load/store unit. The streaming multiprocessors are shown in more detail in Figure 12.

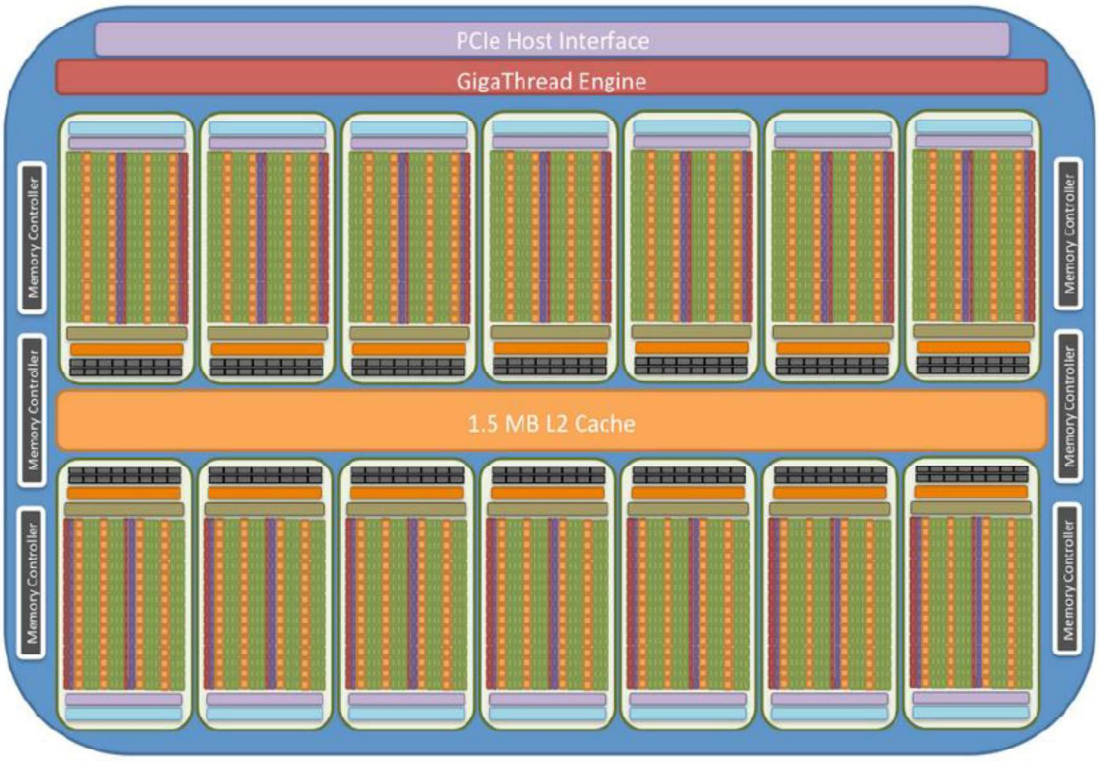


Figure 11. Architecture of NVIDIA Kepler K20X GPU

In terms of the execution of the threads on the hardware, the GPU organizes groups of 32 threads into warps. Each SMX has a total of 2048 threads that are controlled as 64 threadblocks, or warps. All threads in a warp execute the same instruction set. In addition to the threads topology, the memory hierarchy is also somewhat complex. The memory consists of 6GB global memory, a 1.5 MB L2 cache that sits between the global memory and SMX's. The global memory is separate from the CPU memory, and similar to the Intel MIC, the data from main RAM to the coprocessor is handled by the PCIe bus. On each SMX there are 65,536 32-bit registers, 64 KB of shared memory L1 cache and 48KB read only cache. Additionally, there is a constant cache of 8 KB optimized for warp level broadcasts.

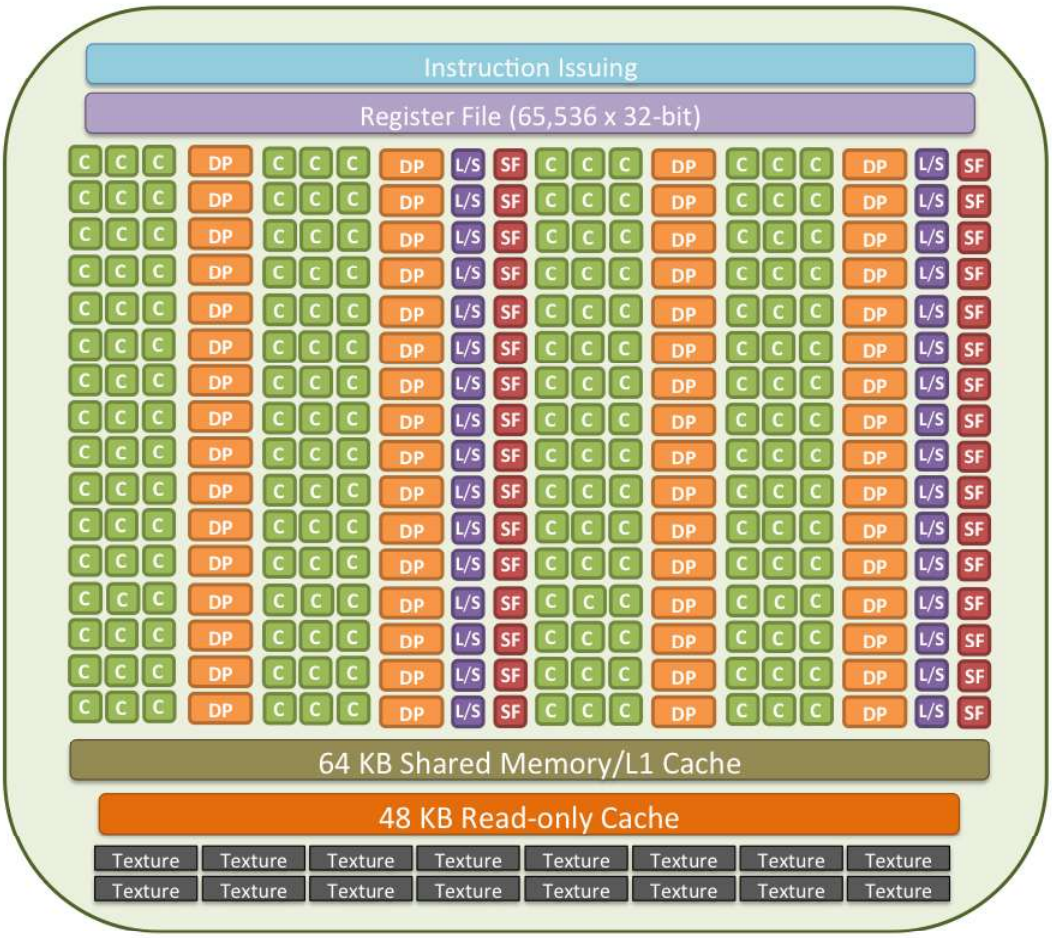


Figure 12. Architecture of NVIDIA Kepler K20X streaming multiprocessor

The programming models to utilize a GPU are somewhat more complex than the Intel MIC, however, the performance gains shown by other researchers suggest the potential for a lot of speedup [8]. In addition to specialized scientific libraries, programming the GPU may also be done using compiler directives, in particular the directives defined by the OpenACC initiative. The newest OpenMP 4.0 standard provides new directives for offloading to a device other than the host also, however there is little compiler support for OpenMP 4.0 at the moment. There are also GPU specific languages that may be utilized to obtain the greatest control over the GPU. Some examples of these languages include CUDA and OpenCL.

In the evaluation of the GPU the test code implemented the original MPACT sweep algorithm outlined in Figure 7, and then investigated Jacobi kernel described in Section 3.1 For the results reported, the reference case was taken to be the original algorithm running on a single core. The other algorithms included the Jacobi algorithm from 3.1 in serial, albeit slightly different in that not all of the arrays were transposed. Then this same algorithm executed on the GPU utilizing OpenACC directives. Four test cases were examined and developed based on typical discretization parameters for PWR type problems. The discretization parameters for these test cases are given in Table 11.

Table 11. GPU test problem discretization

Problem	Fuel Assemblies	Fuel Pins Per Assembly	Energy Groups	Long Rays	Regions
1	1	1	1	584	64
2	324	49	1	65664	1,016,064
3	9	3	56	4824	5,184
4	52 (quarter core)	289	49	70272	961,792

All the numerical tests were performed on Titan. The host process on Titan is the 2.2 GHz AMD Opteron 6274 CPU. The results of the numerical tests are given in Table 12.

Table 12. GPU test problem performance

Problem	Algorithm	Sweep Time (s)	Total Time (s)	Sweep Speedup	Total Speedup
1	Gauss-Seidel, serial	0.0079	0.0108	---	---
	Jacobi, serial	0.0133	0.0179	0.59	0.60
	Jacobi, GPU	0.0459	0.3913	0.17	0.03
2	Gauss-Seidel, serial	100.9	102.1	---	---
	Jacobi, serial	168.9	185.0	0.60	0.55
	Jacobi, GPU	3.2	19.6	31.24	5.21
3	Gauss-Seidel, serial	28.67	30.30	---	---
	Jacobi, serial	48.83	49.88	0.59	0.61
	Jacobi, GPU	1.14	2.51	25.22	12.07
4	Gauss-Seidel, serial	4582	4799	---	---
	Jacobi, serial	8703	8918	0.53	0.54
	Jacobi, GPU	145	368	31.53	13.00

From the results several observations are made. The first is that the serial Jacobi algorithm performs worse than the original algorithm. This is because not all of the data structures were transposed to reflect the looping structure and provide stride-1 memory access. The next observation is that the GPU does provide speed-up, in particular in the sweep. However, the speed-up of the sweep-time does not translate well to the overall speed-up because of the memory transfer overhead. Although some of this overhead may be reduced in real problems since not all of the data needs to be copied every sweep, and typically many sweeps will be done, anywhere from 10 to 100, depending on the simulation being performed. The last observation to be made is that for the first test problem representing a pin cell, the GPU has terrible performance. This implies that there is a minimum problem size that is larger than a pin cell that could be executed on the GPU that would observe good speed-up.

3.3.3 Summary

At this point the investigations into next generation architectures are only preliminary. It would seem that a proof of principle for improvement has been demonstrated that utilization of these new architectures is possible. However, aggressively pursuing these architectures at this point may be premature. In the previous sections, what was not discussed in great detail was the difficulty in working with the compilers. Many of the features of the compilers that were needed to utilize the new hardware often had some issues. Furthermore, with MPACT being written in Fortran 2003, compiler support for this standard is still lacking in several compilers, in particular PGI which has the most mature OpenACC implementation.

4. CMFD PERFORMANCE IMPROVEMENTS

The CMFD performance improvements in this section cover primarily some newly developed ideas that are still considered to be, in part, works in progress. The first of which, Optimally Diffusive CMFD, discussed in Section 4.1 was developed more to address stability and convergence than to directly address performance. Its effect on performance appears only as a minimized number of iterations, at least theoretically. The next section, 4.2, discusses some preliminary research to accelerate convergence of the multi-group diffusion eigenvalue problem through the use of a novel shifted iteration. This work is also still ongoing.

The last effort, that is also still ongoing at the time of this writing, to address the CMFD performance has been the integration of Trilinos so that more advanced iterative techniques for solving eigenvalue problems, such as Generalized Davidson methods, can be used. Since this work is also still ongoing it will not be discussed in this report. The details of this work can be found in [15].

Therefore, the conclusions to be drawn from Section 4 should be that, while progress has been made on improving the CMFD performance, most of the activities are still ongoing and results at this point are preliminary.

4.1 Optimally Diffusive CMFD

It is well known that CMFD acceleration of the transport equation can become unstable if the optical thickness of the coarse cells becomes too large [16], [17]. Some research has been done to address this stability issue with CMFD and multiple solutions have been developed: including: partial current based CMFD [18], performing multiple transport sweeps between CMFD updates [19], higher spatial order prolongation [20], applying relaxation to the flux [19], applying relaxation to the non-linear coupling coefficient $\hat{D}$, and including an artificial diffusion term. One open question has been which of these approaches is most optimal. Some recent work [21] attempted to answer this question and obtained suitable results suggesting that optimization of the artificial diffusion term is a nearly optimal approach.

Traditionally, MPACT has used multiple transport sweeps to stabilize the CMFD acceleration. The stability curves from a Fourier analysis of performing multiple sweeps are shown in Figure 13.

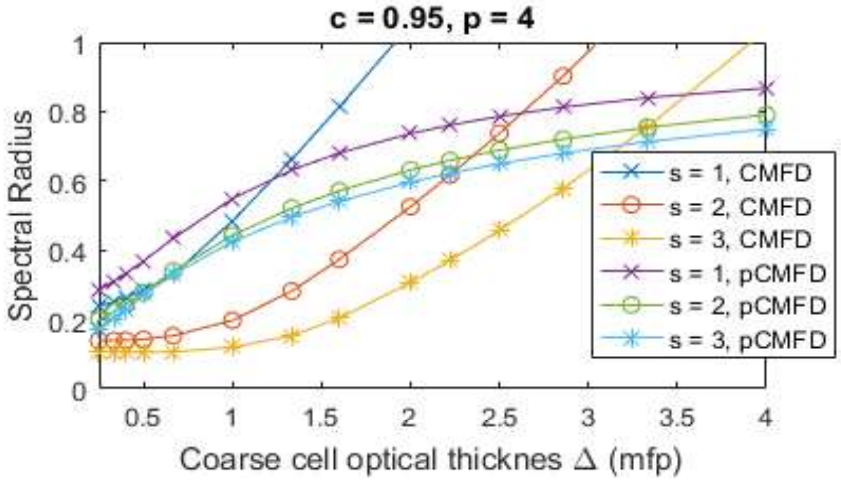


Figure 13. Stability of CMFD and pCMFD with multiple transport sweeps (*s*)

Using multiple transport sweeps to stabilize CMFD is generally not a desirable approach since the transport sweep is usually the most expensive part of the calculation. With effective stabilization of CMFD, the number of transport sweeps may be minimized to just a single sweep per iteration. Therefore, the benefits of using optimally diffusive CMFD (odCMFD) are summarized as:

- Guarantees nearly optimal convergence
- Guarantees stability of the CMFD acceleration
- Minimizes the number of transport sweeps

4.1.1 Description of the odCMFD method

This section provides a summary of the work in [19], [21], and [22]. The optimally diffusive CMFD method (odCMFD) first introduces an extra parameter, θ, to be added to the diffusion coefficient of CMFD.

$$\overline{D}_{j,g} = \frac{1}{3\overline{\Sigma}_{tr,j,g}} + \theta_{j,g}\Delta_j, \tag{1}$$

where the subscript g is the neutron energy group index, j is the coarse cell index, and h is the width of the coarse cell.

The idea to add this term arises from the Fourier analysis of CMFD and pCMFD, where one can observe that the result of the Fourier analysis of CMFD can be made algebraically equivalent to the result from pCMFD by simply adding ¼ to the diffusion coefficient. Figure 14 shows the rate of convergence for several problems using CMFD, pCMFD, and CMFD with an additional artificial diffusion term of ¼ added to the diffusion coefficient.

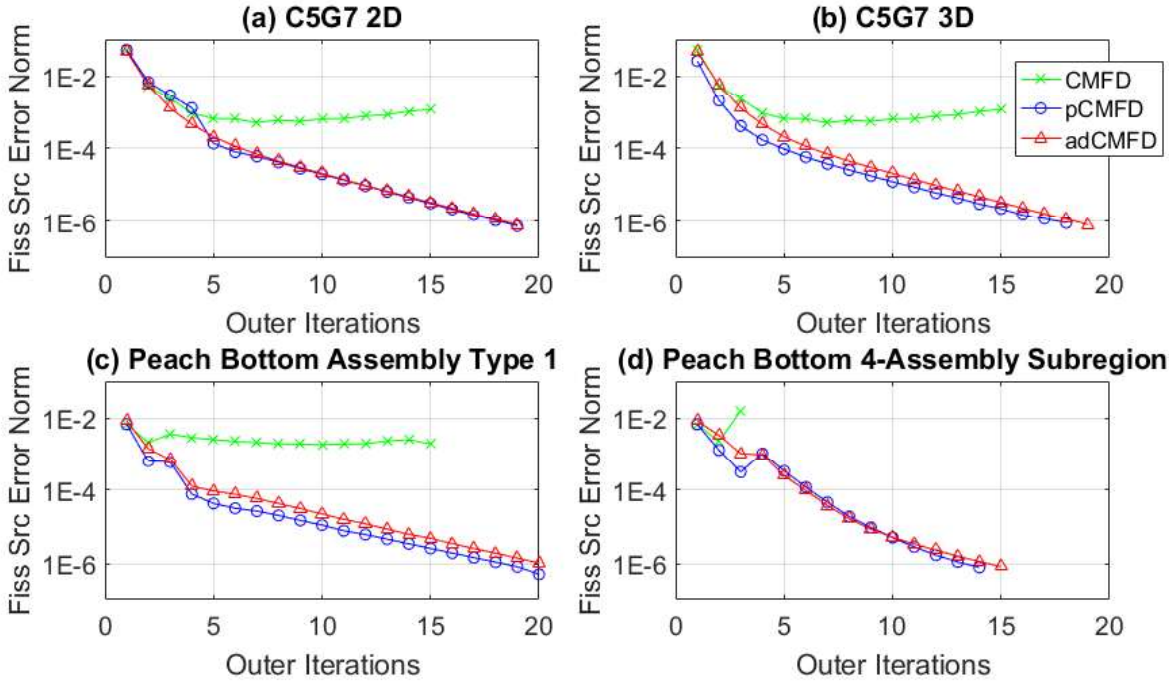


Figure 14. Convergence of CMFD, pCMFD, and adCMFD with $\theta = ¼$

However, in comparing the Fourier analysis result of CMFD and pCMFD it is readily observable that for a certain range of problems where both methods are stable, CMFD will outperform pCMFD. This result has also been observed in practice [21].

Through a generalized Fourier analysis it has been shown that several of the stabilization techniques mentioned previously share some similarity; namely, pCMFD, higher order spatial prolongation, flux relaxation, and artificial diffusion [22]. The pCMFD and artificial diffusion (adCMFD) approaches are demonstrated to behave similarly in Figure 14. It was discovered from the generalized Fourier analysis that flux relation can also be made algebraically equivalent to artificially diffusive CMFD.

Since the Fourier analysis result has been previously shown to be predictive of the convergence behavior of CMFD, it was hypothesized that the generalized Fourier analysis result presented in [22] could be used to derive an optimal artificial diffusion term as a function of the coarse cell optical thickness and scattering ratio. The optimal value was determined numerically by a brute force evaluation of the CMFD Fourier analysis result, then this result was fitted with a polynomial. The optimal artificial diffusion parameter and polynomial fit are shown in Figure 15 and Eq. (2).

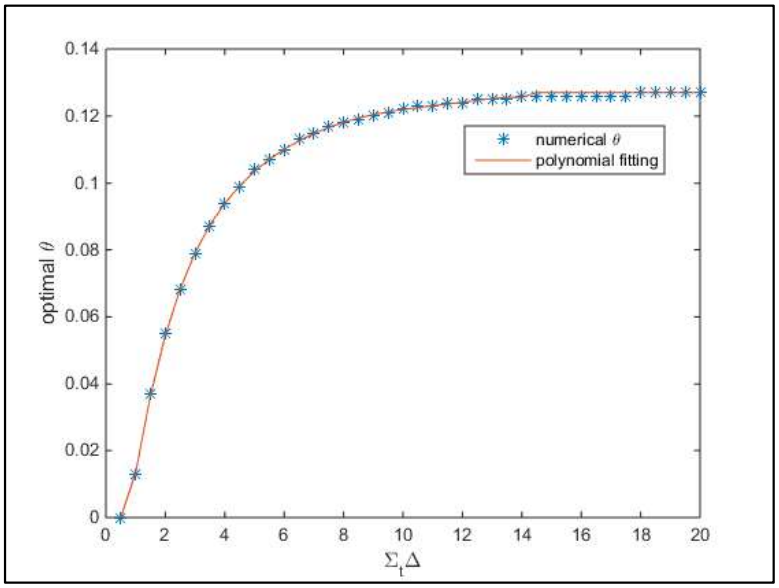


Figure 15. Optimal artificial diffusion parameter and polynomial fit

$$\theta_{g,j}^{opt} = \begin{cases} 0, & \Sigma_{t,g,j}\Delta_j < 1 \\ \sum_{i=0}^{6} a_i \left(\Sigma_{t,g,j}\Delta_j\right)^i, & 1 \le \Sigma_{t,g,j}\Delta_j < 14, \\ 0.127, & \Sigma_{t,g,j}\Delta_j \ge 14 \end{cases} \quad \textbf{(2)}$$

Table 13. Coefficients of optimal artificial diffusion parameter Eq. (2)

Order i	Coefficient ai
0	-5.542780E-02
1	8.740501E-02
2	-2.152599E-02
3	3.145553E-03
4	-2.683648E-04
5	1.222516E-05
6	-2.284879E-07

Additionally, to evaluate the quality of the optimal artificial diffusion parameter, the Fourier analysis result was optimized in a more general way [22]. This result, shown in Figure 16, produces a theoretical lower bound for the asymptotic convergence of CMFD for the variants mentioned in this section. This result also shows that the odCMFD method is very close to the optimal.

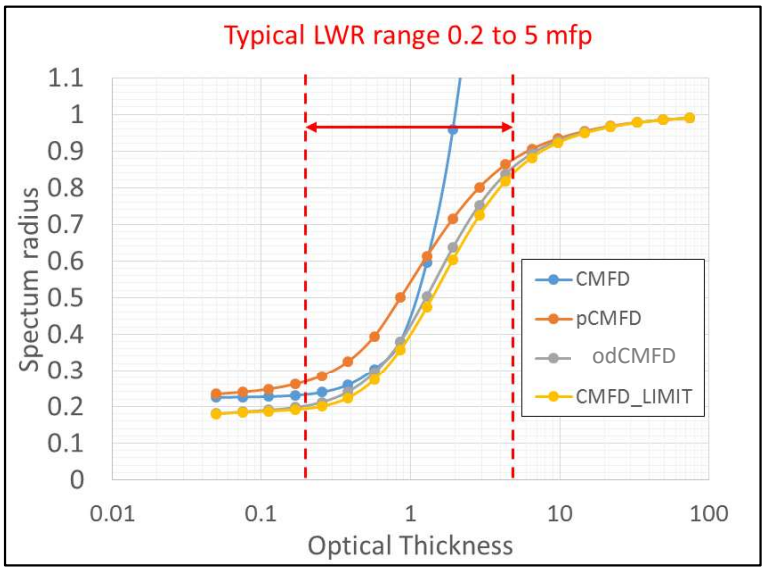


Figure 16. Comparison of theoretical spectral radius of several CMFD variants

4.1.2 Numerical results

Three 2D problems with increasing complexity were tested to evaluate the odCMFD method. The problems are:

- C5G7 (Figure 5)
- VERA Benchmark Problem 5a-2D (Figure 3)
- Peach Bottom Unit 2 BOL at CZP (Figure 4)

The number of iterations for each problem are shown for each method in Table 14. All problems were run with the default discretization parameters in MPACT.

Table 14. Number of iterations for CMFD variants for test problems

Problem	CMFD	pCMFD	odCMFD
C5G7	Diverged	19	18
VERA Problem 5 (2D)	15	16	14
Peach Bottom (2D)	Diverged	33	30

4.1.3 Summary and continuing work

By using the odCMFD method, MPACT has a sound theoretical basis to which provides confidence that the accelerated iteration scheme is unconditionally stable and will converge as fast or faster than CMFD and pCMFD. However, in recent applications of this method to the Cawtaba reactor, the iteration scheme was observed to become unstable [23]. This result will require further investigation, but a likely explanation is that the theory behind odCMFD does not account for the 2D/1D iteration scheme, just a conventional transport sweep. 2D/1D is also known to have stability issues [24]. The stability analyses performed for each of these iteration schemes should be revisited together in order to determine if this is in fact the cause of the observed instabilities or if the cause lies elsewhere.

4.2 Space Dependent Wielandt Shift

In the calculation of neutronic eigenfunctions and eigenvalues, it is often desirable to approximate the neutron transport equation by a diffusion equation that requires significantly less computational resources to solve. Even when a full transport solution is needed, the solution of the diffusion system can be used to accelerate iterative methods for calculating the transport eigenfunction and eigenvalue [25].

The convergence rate of the basic power iteration method for eigenvalue problems is governed by the dominance ratio; realistic reactor core problems often have high dominance ratios and the power iteration method can converge slowly as a result. A common technique for accelerating the convergence of power iteration is the Wielandt shift (WS) method [26], which reduces the dominance ratio by "shifting" the eigenvalue spectrum using an estimate of the desired eigenvalue. WS provides a significant speedup over power iteration if the eigenvalue estimate is close to the true eigenvalue, but it can be difficult to obtain such an estimate. In practice, the shift in the WS method can be dynamically updated in the iteration scheme, in a way that depends on the eigenvalue estimates from the most recent iterations. In MPACT, a fixed Wielandt shift value of 2/3 is typically used.

This section describes the preliminary research of a physically-motivated Space-Dependent Wielandt Shift (SDWS) for solving multi-group diffusion eigenvalue problems. Much of the information is reproduced from reference [27]. The space-dependence of the eigenvalue shift in SDWS enables a more effective acceleration of the power iteration scheme than WS at the beginning of the iteration scheme, when the current estimate of the eigenvalue is not close to the actual eigenvalue. Several variants of SDWS are discussed here, including an SDWS method that combines the strengths of SDWS (in the early iterations) with those of standard iteration-dependent WS (in the later iterations). Numerical results from a 1D multi-group diffusion problem demonstrate an order of magnitude speedup over power iteration and a 41% speedup over a standard WS technique. This speedup is highly problem-dependent, however.

4.2.1 Theory

To simplify the discussion and notation, the theory is described in 1D, however its application to 3D is straightforward. Given the 1D multi-group diffusion eigenvalue problem the power iteration scheme is:

$$-\frac{d}{dx}D_g(x)\frac{d\phi_g^{(n+1/2)}(x)}{dx}+\Sigma_{t,g}(x)\phi_g^{(n+1/2)}(x)-\sum_{g'=1}^{G}\Sigma_{s0,g'\to g}(x)\phi_{g'}^{(n+1/2)}(x) = \lambda^{(n)}\chi_g(x)\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(x)\phi_{g'}^{(n)}(x), \tag{3}$$

$$\phi_g^{(n+1)}(x)=\phi_g^{(n+1/2)}(x)\left[\sum_{g'=1}^{G}\int_0^X\phi_g^{(n+1/2)}(x')dx'\right]^{-1}, \tag{4}$$

$$\lambda^{(n+1)}=\frac{\sum_{g=1}^{G}\left[\int_0^X\Sigma_{a,g}(x)\phi_g^{(n+1)}(x)dx-D_g(x)\frac{d\phi_g^{(n+1)}}{dx}\Big|_{x=0}^{x=X}\right]}{\sum_{g=1}^{G}\int_0^X\nu\Sigma_{f,g}(x)\phi_g^{(n+1)}(x)dx}. \tag{5}$$

where, n is the iteration index. When adding the Wielandt shift, the shifted power iteration scheme becomes:

$$-\frac{d}{dx}D_g(x)\frac{d\phi_g^{(n+1/2)}(x)}{dx}+\Sigma_{t,g}(x)\phi_g^{(n+1/2)}(x)-\sum_{g'=1}^{G}\Sigma_{s0,g'\to g}(x)\phi_{g'}^{(n+1/2)}(x) - \lambda'\chi_g(x)\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(x)\phi_{g'}^{(n+1/2)}(x)=\left(\lambda^{(n)}-\lambda'\right)\chi_g(x)\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(x)\phi_{g'}^{(n)}(x), \tag{6}$$

$$\phi_g^{(n+1)}(x)=\phi_g^{(n+1/2)}(x)\left[\sum_{g'=1}^{G}\int_0^X\phi_g^{(n+1/2)}(x')dx'\right]^{-1}, \tag{7}$$

$$\lambda^{(n+1)}=\frac{\sum_{g=1}^{G}\left[\int_0^X\Sigma_{a,g}(x)\phi_g^{(n+1)}(x)dx-D_g(x)\frac{d\phi_g^{(n+1)}}{dx}\Big|_{x=0}^{x=X}\right]}{\sum_{g=1}^{G}\int_0^X\nu\Sigma_{f,g}(x)\phi_g^{(n+1)}(x)dx}. \tag{8}$$

Using a Wielandt shift introduces the potential for several things to go wrong. For example, if one uses a shift that is exactly the fundamental eigenvalue, then the system becomes singular, which means the linear system cannot be solved by conventional means. Furthermore, if the shift does not lie on the interval $[0,\lambda)$, then iterations may not converge to the fundamental mode due to the source in Eq. (6) being negative. Finally, one must be careful in their choice of methods by which Eq. (6) is solved when a direct solve is computationally prohibitive. When the shift value approaches the true eigenvalue, the linear system in Eq. (6) becomes approaches a singular linear system. Many iterative solvers struggle with ill-conditioned or singular systems and, if the iterative solver is not carefully chosen, the benefits of reducing the number of power iterations required to converge may be offset by the extra time required to solve the linear system at each power iteration step.

In addition to using a fixed shift, which is what is presently done in MPACT, one may also use a dynamic or adaptive shift that is iteration dependent (e.g. $\lambda' \rightarrow \lambda'^{(n)}$). One such shift that is used by the PARCS code is:

$$\lambda_P'^{(n)} = \max\left\{\lambda^{(n)} - c_1\left|\lambda^{(n)} - \lambda^{(n-1)}\right| - c_0, \lambda_{\min}\right\}, \tag{9}$$

where $\lambda_{\min}$, c_1, and c_0 are user provided values. The idea in this equation is that $\lambda_P'^{(n)}$ is always smaller than $\lambda^{(n)}$ for each iteration, thus leading to a subcritical system on the left-hand side of Eq. (6) and a positive source on the right hand side of Eq. (6). $\lambda_{\min}$, c_1, and c_0 should be chosen such that the above is true. In PARCS (and in the results shown in this document), these values are: $\lambda_{\min}$=1/3, c_0=0.02, and c_1=10.

The subsequent sections describe three variants of the shifted power iteration that differ only in their definition of the eigenvalue shift.

4.2.1.1 SDWS-LE

The first SDWS method solves a Local Eigenvalue (LE) problem in each cell, and then uses this value as the shift. In this method the shift parameter is iteration-independent, and is determined by Eq. (10) which is the solution of the local infinite medium eigenvalue.

$$\lambda_{LE}'^{(n)}(x) = \lambda_\infty(x) = \left[\underline{\nu\Sigma}_f(x)\right]^T \left[\underline{\underline{\Sigma}}_t(x) - \underline{\underline{\Sigma}}_{s0}(x)\right]^{-1} \underline{\chi}(x). \tag{10}$$

Here the "_" notation indicates G x 1 column vector and the double "=" indicates a G x G matrix. This system can typically be solved directly without much effort, provided the number of groups is not extremely large.

The difficulty with the LE method is that, in a heterogeneous media, the right side of Eq. (6) can change sign, and this can sometimes cause the method to converge to a spurious solution. This is an obvious deficiency in the SDWS-LE method, which motivates the variant discussed next, SDWS-PS.

4.2.1.2 SDWS-PS

The Positive Source (PS) variant of the SDWS approach is designed to guarantee that the right side of Eq. (6) remains positive throughout the iterations. The SDWS-PS method attempts to use the LE shift everywhere except where it would drive the source negative in a cell. In the case where the LE shift would cause a negative source, the current estimate of the eigenvalue is used instead. Thus the equation for the shift used by the SDWS-PS method is defined quite simply as:

$$\lambda_{PS}'^{(n)}(x) = \min\left\{\lambda_\infty(x), \lambda^{(n)}\right\}. \tag{11}$$

The PS method always (from our experience) converges to the correct eigenfunction and eigenvalue. In particular the PS shift provides an effective shift in the early iterations, when the estimate of the eigenvalue is often not very good. However, in later iterations, compared to an adaptive shift, such as the one used by PARCS, the adaptive shift outperforms the PS shift. Thus once a certain threshold in the error of the eigenvalue is passed, convergence of an adaptive shift will outperform the PS shift. This motivates the creation of the yet another SDWS variant: SDWS-IPS.

4.2.1.3 SDWS-IPS

The Improved Positive Source (IPS) variant of SDWS attempts to improve upon the PS variant without introducing a negative component to the source. Essentially the modification to Eq. (11) is to introduce a logic to use the PARCS shift if it is more aggressive than the PS shift. Thus the equation for the SDWS-IPS shift is:

$$\lambda_{IPS}^{(n)}(x) = \max\left\{\lambda_{PS}^{(n)}(x), \lambda_{P}'^{(n)}\right\}. \quad \textbf{(12)}$$

The IPS variant has the following desirable properties: (i) it provides a more aggressive shift than the PARCS shift in early iterations when the eigenvalue estimate is far from the solution, (ii) it provides a shift that is closer to the true eigenvalue in later portions of the iteration scheme than the PS method, and (iii) it ensures that the terms on the right hand side of Eq. (6) are non-negative.

4.2.2 Numerical results

In this section numerical results of the SDWS variants discussed previously are reported for two 1D test problems using a research code developed to explore the method. The test problems that were investigated are:

- 1D 1-group Joo-Lee problem [28], computational cell = 1 pin cell
- 1D 47-group "Watts Bar" problem, computational cell = 5 pin cells

The "Watts Bar" problem uses pin cell homogenized cross sections for the centerline of pins in the VERA Problem 5a-2D benchmark shown in Figure 3. Each of the SDWS variants are compared to cases that do not use a shift, use a fixed shift, and use the PARCS shift. The first set of results solve the multi-group linear system directly using Gaussian elimination. The number of iterations for each method are given in Table 15.

Table 15. Iterations required of various shift methods of various shift methods for 1D test problems using Gaussian elimination

Method	Joo-Lee	"Watts Bar"
$\lambda'=0$	431	158
$\lambda'=2/3$	106	77
PARCS	26	17
SDWS-PS	59	16
SDWS-IPS	24	10

The results in Table 15 essentially represent the number of power iterations required to converge -- only one Gaussian elimination is required per power iteration since the system is solved exactly. We see that the adaptive and SDWS shifts provide an order-of-magnitude reduction in the number of power iterations required compared to a fixed shift (or no shift). We also see that the relative performances of SDWS-PS, SDWS-IPS, and the PARCS shifts are very much problem-dependent. Nonetheless, the SDWS-IPS shift, by design, will always converge as fast as or faster than either the PARCS or SDWS-PS shifts alone.

The number of power iterations required, however, is not the only quantity that should be observed. As noted earlier, an aggressive Wielandt shift makes the diffusion system (which has to be solved at each power iteration step) nearly singular and more ill-conditioned. Direct solvers are not feasible for realistic problems, so iterative solvers must be considered. The performance of these iterative

solvers is generally dependent on the conditioning of the matrix, so one has to be careful in picking an iterative solver if one wishes to take advantage of Wielandt shift methods.

In Tables 11-13 below, we consider the use of various iterative solvers and preconditioners in place of the direct solver used for the results in Table 15. One surprising result is that, although the SDWS-IPS shift always requires fewer power iterations than the SDWS-PS shift, the SDWS-PS shift can actually outperform the SDWS-IPS shift for certain choices of iterative solvers and preconditioners – namely those that do not perform well with the ill-conditioning caused by the Wielandt shift. For example, in Table 16 SDWS-PS requires significantly fewer iterations to converge than the SDWS-IPS method when red-block block Jacobi is used. We see the same phenomenon in Table 17 when unpreconditioned GMRES is used for the Watts bar problem.

One interesting result is that the ILU preconditioner seems to help GMRES and BiCGSTAB become agnostic to the ill-conditioning caused by the Wielandt shift. (This appears to be more true for BiCGSTAB than it is for GMRES.) In Tables 12-13, we see that the columns most proportional to the number of power iterations required (i.e., the results from Table 15) are those for the ILU preconditioner.

Table 16. Iterations required of various shift methods for 1D test problems using Red-Black Block Jacobi

Method	Joo-Lee Problem	"Watts Bar"
λ'=0	N/A	1056
λ'=2/3		870
PARCS		986
SDWS-PS		359
SDWS-IPS		570

Table 17. Iterations required of various shift methods for 1D test problems using GMRES

Method	No Preconditioner		Block Jacobi		ILU	
	Joo-Lee	"Watts Bar"	Joo-Lee	"Watts Bar"	Joo-Lee	"Watts Bar"
λ'=0	336630	10995	69516	2418	402	400
λ'=2/3	119000	13461	54716	1664	106	256
PARCS	62000	28275	29838	1075	24	79
SDWS-PS	95000	10235	49765	682	70	65
SDWS-IPS	62000	21926	30841	797	23	50

Table 18. Iterations required of various shift methods for 1D test problems using BiCGSTAB

Method	No Preconditioner		Block Jacobi		ILU	
	Joo-Lee	"Watts Bar"	Joo-Lee	"Watts Bar"	Joo-Lee	"Watts Bar"
λ'=0	69506	5179	59060	1270	402	220
λ'=2/3	28298	4695	24000	858	106	133
PARCS	12378	2405	9262	401	24	42
SDWS-PS	24637	2140	19268	308	70	32
SDWS-IPS	11182	2045	8578	314	23	23

Finally, because the time per iteration is not necessarily equal for each method, the run times of each of the solution methods for the "Watts Bar" problem are presented in Table 19. The reader is reminded that these results are from a 1D test code which has not undergone optimization, so these results could vary significantly for optimized implementations.

Table 19. Total serial run times [s] for various solutions methods of the "Watts Bar" test problem

Method	Direct Solve	Red-Black Jacobi	GMRES			BiCGSTAB		
Preconditioner	---	None	None	Block Jacobi	ILU	None	Block Jacobi	ILU
λ'=0	306.9	14.7	40.22	24.6	27.4	38.2	25.0	26.7
λ'=2/3	637.7	25.5	34.9	12.6	13.0	26.0	12.7	12.9
PARCS	69.8	7.9	56.5	4.4	3.1	10.9	3.8	3.1
SDWS-PS	66.9	4.1	22.0	3.4	2.8	9.7	3.2	2.7
SDWS-IPS	40.8	4.6	45.0	3.0	1.8	8.7	2.6	1.8

From the results, one sees that the ILU-preconditioned methods converged the fastest in terms of serial run time. However, it should be noted that certain methods (such as block Jacobi) are more readily parallelized than other methods (such as ILU-preconditioned solvers), and more work needs to be done to study the performance of the methods as we scale to larger and more realistic problems.

As it can be seen from these results, it is not necessarily optimal to use a red-black Jacobi scheme or GMRES. Presently in MPACT these are the main linear solvers for the CMFD problem. However, the Red-Black Jacobi solver in MPACT also includes successive over-relaxation, which should provide faster convergence. Preliminary tests in MPACT using the SDWS-IPS method seem to confirm this, as significantly more time may spent attempting to converge the linear system for a particular shift. Thus, this work is still ongoing because there is a clear need to use an efficient and robust linear solver with the SDWS-IPS shift.

In Table 20, we have some preliminary results from our implementation of SDWS-IPS in MPACT. First, the results indicate that SDWS-IPS is incompatible with restart-GMRES – this approach does not converge for this problem. Second, we observe a significant improvement over the default approach in MPACT when BiCGSTAB is used instead with SDWS-IPS; the runtime for SDWS-IPS with BiCGSTAB is approximately 2/3 of that required for the default MPACT approach. One oddity in the results, however, is that SDWS-IPS requires significantly more transport sweeps than the fixed 2/3 shift. This is an unexpected result since the shift method should only affect the CMFD solver, and indicates that there is a suboptimal convergence criterion being used to determine the convergence of the SDWS-IPS method with BiCGSTAB in the CMFD solver. It is likely that improving this convergence criterion (i.e., requiring SDWS-IPS to spend more time in the CMFD solver) will lead to a reduction in transport sweeps and a significant reduction in the total solve time. Again, more work needs to be done to determine the optimal iterative solver and implementation for SDWS-IPS, particularly in the context of solving the CMFD system in a transport code.

Table 20. Preliminary run times from MPACT for various solution methods for quarter core Problem 5.

Method[1]	Transport Sweeps	CMFD Solve Time (s)	Total Solve Time (s)
λ'=2/3 with restart GMRES	13	1092	1695
SDWS-IPS with restart GMRES	DNC	DNC	DNC
SDWS-IPS with BiCGSTAB	21	304	1080

[1] – All linear solvers used a block Jacobi preconditioner
DNC – Did Not Converge

4.2.3 Summary and continuing work

Several complementary efforts are underway to address the performance of the CMFD. This includes the development of new theory to optimize convergence and leveraging of existing scientific software libraries. The work discussed in this section only related to the former, and as results have shown, there is some promising capability for improvement, but important details for the iteration scheme, such as a robust and efficient linear solver still need to be determined.

Additionally, the efforts to leverage existing scientific software libraries appear to be quite promising as well [15], but these are presently undergoing testing and quality assurance, and so have not been deployed at the time of this writing.

5. CROSS SECTION PERFORMANCE IMPROVEMENTS

The performance improvements for the cross section calculation in MPACT were performed under the supporting milestone L3:RTM.XS.P13.01. The information presented in this section is largely a condensation of the information from this milestone report [29].

The cross section calculation in MPACT includes the resonance self-shielding calculation to obtain the equivalence cross section in the resonance groups, use of the equivalence cross section to compute the microscopic cross section in resonance groups, and calculation of region-wise macroscopic cross sections and fission spectra for all energy groups. As shown in Section 1, it has been identified that these cross section calculations account for more than 70% of total computing time in a MPACT steady state 2D quarter-core depletion calculation. This fraction of time spent in computing the cross sections is larger than other direct neutron transport codes, such as DeCART [30] or nTRACER [31].

To improve the efficiency of cross section calculation in MPACT, three tasks have been planned in this milestone:

- Improve calculation of macroscopic cross section and fission spectrum
- Develop a MOC sweeper solely for subgroup calculation
- Investigate and implement the 1-group subgroup method

Section 5.1 provides the technical details on the performance improvements of the macroscopic cross section and fission spectrum calculation. Section 5.2 discusses the details of a new transport kernel for solving the subgroup resonance self-shielding fixed source problem. Section 5.3 describes a new approach to treating some resonant material with a one-group model. Finally a summary of the speed-up from this work is given in Section 5.4.

5.1 Macroscopic Cross Section Calculation

This subsection discusses the performance improvements in calculation of the macroscopic cross section and fission spectrum. The related routines in MPACT include *calcMacroXS*, *calcMacroChi* and *segev*.

5.1.1 Speed-up of the fission spectrum calculation

The initial profiling of MPACT showed that for a 2D full-core depletion case, about one third of the computing time was being spent on computing the fission spectrum. In the multi-group transport equation with multiplying media the fission source is written as:

$$F_g(r) = \sum_{iso} \chi_{g,iso}(r) \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \phi_{g'}(r). \quad (13)$$

Although, the fission spectrum is dependent on the fissionable isotope, to facilitate the calculation of the fission source an effective fission spectrum, that is independent of the isotope, is used. The fission source calculated from the effective fission spectrum, $\chi_g^{eff}(r)$, is given by:

$$F_g(r) = \chi_g^{eff}(r) \sum_{iso} \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \phi_{g'}(r). \quad (14)$$

This leads to the following definition of the effective fission spectrum:

$$\chi_g(r) = \frac{\sum_{iso} \chi_{g,iso}(r) \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \phi_{g'}(r)}{\sum_{iso} \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \phi_{g'}(r)}. \quad (15)$$

In MPACT, the routine *calcMacroChi* is called when the effective fission spectrum of a cross section mesh region (associated with a material) is requested. Specifically, *calcMacroChi* takes the arguments of atomic number densities, cross section library, resonance parameters (equivalence cross section) and scalar fluxes of this fissionable material region. In general the fission spectrum should be updated every time the eigenvalue (e.g. k_{eff}) is updated so that one has a consistent iterative method for the fission source.

Previously, every time *calcMacroChi* was called the isotopic fission cross sections were computed even though these values would not change between successive eigenvalue iterations. It was determined that the repeated calculation of the isotopic fission cross sections was the most time consuming part of *calcMacroChi*.

Therefore, to avoid this time consuming repeated calculation, the decision was made to store the isotopic fission cross sections on the cross section mesh, and only update these values when the material changes (e.g. from depletion or T/H feedback).

The following table shows the speed-up for this approach on VERA Problem 2a, which has been modified to deplete with 10 time steps. The case used a 0.05 cm ray spacing, 16 azimuthal angles per octant, and 3 polar angles. Figure 1 shows the geometry of the Problem 2a, which is a simple 17x17 lattice of 2.1% enriched pins using quarter symmetry, but the test run was performed for the full lattice without using symmetry.

As shown in Table 2.1.1, by storing the fission cross section instead of computing them on-the-fly, the runtime of *calcMacroChi* is reduced by a factor of 55. The memory increase per 2D assembly is less than 10MB, a moderate increase as compared to the efficiency gains. Also, no accuracy is expected to be sacrificed by this update.

Table 21. Comparison of computational resources for storing the fission cross section

	Total Time (min)	*calcMacroChi* Time (min)	Total Memory (MB)
Computed	89.2	18.8	157
Stored in Memory	70.3	0.335	165

Based on the data in Table 21, we may extrapolate the memory overhead for a 2D quarter core. In this problem the associated memory overhead is predicted to be 514 MB. However, the VERA

problem 5 2D problem was run to assess this and a difference of approximately 174 MB was observed, which is quite reasonable for a 2D core.

5.1.2 Speed-up of Segev interpolation

The initial MPACT profiling results show that for a 2D full-core depletion case, about 20% of the total computing time is spent on an interpolation subroutine *segev*, which is primarily called in *calcMacroXS* and *calcMacroChi*. The Segev scheme [32] interpolates the RI at a specific background cross section from a RI table using the following expression:

$$f(\sigma_b) = \left(\frac{\sigma_b}{\sigma_b + \eta} \right)^P . \tag{16}$$

The whole interpolation procedure is to determine the coefficients P and η so that the self-shielding factor can be computed given a specific σ_b. These coefficients are determined iteratively using a bisection method, and each iteration of the Segev interpolation involves the evaluation of two exponentiations.

On a computer it is known that the exponentiation operator is very expensive to evaluate, more so than most other mathematical operations and intrinsic functions. The exponentiation can be eliminated from the above equation by applying the following identity:

$$x^a = \exp\left[a \log(x)\right]. \tag{17}$$

Making use of this identity and using other techniques to minimize the number of LOG() and EXP() functions that must be evaluated reduces the number of FLOPs per call to *segev* from 4425 to 2047.

This change leads to a speed-up of approximately 2x in the *segev* routine.

However, even with this speed-up the time spent in *segev* is still significant. This is due to the sheer number of times the routine is called. Given the changes described in Section 5.1.1, the *segev* calls in *calcMacroChi* are eliminated, thus the focus becomes minimizing the calls to *segev* from *calcMacroXS*.

For the subgroup method, a fresh pin cell test shows that previously, the *segev* routine was called 15,232 times. More than 90% of these calls are used in the non-uniform fuel temperature treatment and the remainder are used in computing the resonance self-shield scattering cross section. The number of calls for the resonance scattering components are reasonable (proportional to the number of resonance groups, isotopes, and regions). It was discovered that the massive number of calls to *segev* for the non-uniform temperature treatment was due to an inefficient looping structure. In this looping structure, the *segev* routine was being called many more times than were actually needed.

In fact for the non-uniform temperature treatment, the *segev* interpolation is used to compute an approximate effective cross section in order to determine the temperature adjustment ratio for the subgroup method (refer to [33] for details). In this approach, the background cross section is approximated to be $\lambda\sigma_p$, because an accurate background cross section cannot be obtained before performing the subgroup calculation. However, there is no justification for using such an expensive interpolation scheme given the argument is far away from the true value. Therefore, the *segev* interpolation to obtain the effective background cross section was replaced by directly using the base effective absorption provided by the MG library. So, the value used to determine the temperature adjustment ratio was changed to the unshielded absorption cross section. Physically, these cross sections should be much better than the current *segev* interpolated values, especially for LWR applications.

To assess the speed-up from this change in removing the *segev* interpolation for the non-uniform temperature treatment the VERA 2a problem shown in Figure 1 was run again. Effectively, the time spent in *calcMacroXS* was reduced by a factor of 2 from this change. The results are given in Table 22 below.

Table 22. Comparison of computational resources for replacing *segev*

	Total Time (min)	*calcMacroXS* Time (min)	Total Memory (MB)
segev	37.8	7.6	165
Base eff. XS	33.9	4.2	165

Since this change represents a change in the non-uniform temperature treatment methodology, a detailed study was performed to verify the results from using the unshielded absorption cross section. Overall, the computed value of the self-shielded absorption cross section showed better agreement with a Monte Carlo reference. The details for this study are given in [29].

5.1.3 Miscellaneous optimizations

In addition to the major improvements discussed in Sections 5.1.1 and 5.1.2, a few minor improvements have been implemented in *calcMacroXS*:

- Non-uniform temperature treatment is turned off for uniform-temperature case
- Skip the calculation of P_2-P_3 scattering matrices when transport-corrected scattering (TCP_0) is requested
- Unify the range of incoming scattering cross sections at a group for different temperatures
- A few other minor optimizations

These improvements give another 40% speed-up of *calcMacroXS* without changing any results. The most recent time distribution of *calcMacro* (*calcMacroXS* and *calcMacroChi*) for VERA-2a with depletion is shown in Table 23. The computation of scattering matrix is still a significant part, although we have done a few things to optimize it. It is a little surprising that the "overhead" part is non-trivial. This is obtained by comparing the timing of *calcMacroXS* inside the library and *calcMacroXS* inside the *XSMesh* (the difference of these two). The reason of it can only be attributed to the overhead of the subroutine call and those parts of *calcMacroXS* that were not explicitly profiled.

Table 23. Fractional time distribution in macroscopic cross section calculation

calcMacroXS				*calcMacroChi*
94%				6%
Scat. Matrix	***Resonance***	***Other***	***Overhead***	
45%	24%	7%	18%	

To further improve the efficiency of the cross section calculation, neglecting the non-important isotopes when computing the scattering matrices is a potential time saving approach. However, a detailed sensitivity study is required to determine the limit of isotopic number density.

5.2 Development of Fast Subgroup Fixed Source Problem Transport Kernel

Previously, the same MOC sweeper was used in both eigenvalue calculation and subgroup cross section calculation. Although the subgroup calculation solves a purely absorbing fixed source problem using IR approximation, where the standard MOC sweeper is likely excessive for this problem.

This section discusses the development of a new MOC sweeper for exclusive use in the subgroup calculation. Two improvements have been made to speed up the subgroup fixed source calculation:

- Vectorization of the subgroup categories and levels.
- Using pre-computed MOC transmission parameters for efficient convergence of the angular flux boundary condition.

Most contents in this section are extracted from a technical report [34]

5.2.1 Vectorization of subgroup categories and levels

In the subgroup method, the resonances are discretized into several levels. Additionally, the resonant isotopes are grouped into categories to facilitate treatment of resonance interference. To obtain the equivalence cross sections from subgroup to perform the resonance self-shielding, a different purely absorbing fixed source problem must be solved for each resonance energy group, category, and subgroup level.

In the cross section library for MPACT there are 17 resonance energy groups, 4 subgroup levels, and for typical PWR analysis, 4 subgroup categories. Provided one starts with an MOC kernel formulated to solve a 1-group fixed source problem with scattering, a natural looping structure to solve these fixed source problems looks like:

```
Loop over resonant groups (g from g_res,beg to g_res,end)
    Loop over subgroup categories (c from 1 to N_cat(g))
        Loop over subgroup levels (l from 1 to N_levels)
            Compute Σ_t for this group/category/level from Σ_a,g,c,l and λΣ_p.
            Compute source q for this group/category/level from λΣ_p.
            Iterate to solve subgroup fixed source problem
                Perform transport sweep
                Check for convergence
```

Figure 17. Pseudocode for subgroup fixed source problem looping structure

Since each of the subgroup fixed source problems is independent of the others, they can be solved concurrently, or in other words the different fixed source problems can be *vectorized*. This is effectively consistent with the transformations made to develop the new multi-group MOC kernel described in Section 3.1, where each subgroup fixed source problem is treated like a different energy group, or as if it were a pseudo-group.

For typical applications MPACT must solve 272 subgroup fixed source problems. Since it may be too burdensome to store all the parameters for all the fixed source problems simultaneously, an alternative is to solve a set, or batch, of the fixed source problems simultaneously. This is advantageous for a couple reasons. First it provides some flexibility in balancing memory overhead with reduction in runtime. Furthermore, it provides some mechanism for tuning the batch size to align with cache boundaries in the computer's memory hierarchy to get the most efficient computational performance. The pseudocode for this batched looping structure is given as:

```
Loop over batches (b from 1 to N_batch)
    Loop over subgroup fixed source problems ( pg from 1 to N_FSP(b))
        Compute Σ_t,pg for all FSP's from Σ_a,g,c,l and λΣ_p ( pg ← (g,c,l)).
        Compute source q_pg for this group/category/level from λΣ_p.
    Iterate to solve subgroup fixed source problems
        Perform transport sweep for all pg
        Check for convergence
    Loop over subgroup fixed source problems ( pg from 1 to N_FSP(b))
```

$\Sigma_{eq,g,c,l}$

Figure 18. Pseudocode for vectorized subgroup fixed source problem looping structure

The performance of the fixed source problem vectorization is not reported. However, based on the results from Section 3.1, which observed a speedup of roughly 1.4x using 47-group kernels without current tallies, it would not be unreasonable to expect a similar performance increases, especially if the number of fixed source problems in a batch is roughly 40-50. Evaluating the performance of subgroup fixed source problem vectorization, in addition to the next improvement, is the focus of the results in Section 5.2.3.

5.2.2 Fast MOC kernel for purely absorbing fixed source problems

For any fixed source problem in a purely absorbing medium, if the boundary condition is known, the angular flux solution can be obtained in a single transport sweep. However, typical reactor problems in MPACT are run with symmetry, thus introducing an iteration on the boundary condition. Furthermore, when spatial decomposition is used with MPACT the boundary condition between the different spatial subdomains also becomes an iterative quantity. Thus, the subgroup fixed source problem is primarily iterating on the angular flux boundary condition.

Since, the typical subgroup fixed source problem in MPACT is focused on converging the angular flux boundary condition, the decision was made to implement a different MOC transport kernel that would more efficiently converge the boundary condition. This efficiency is achieved, in part, by not computing the multi-group scalar flux until after the boundary condition has converged.

The other development leading to a more efficient calculation arises from recognizing that the way in which the outgoing angular flux at a boundary depends on the incoming angular flux does not change during the iterations. For a given characteristic ray, the attenuation and production of neutrons along that ray through the domain is constant; this is only true because the problem has a fixed source and no scattering. Therefore, one may pre-compute and store the coefficients that relate

the outgoing angular flux to the incoming angular flux on the boundaries, rather than compute these explicitly on every sweep.

This idea relates to other work that developed the Characteristic Direction Probabilities (CDP) [35] method. In fact this method has existed in MPACT for some time [36]. However, its implementation was to be used for a general transport solver with scattering. Traditionally, the CDP kernel has not been used because it has a high associated storage cost, and for the realistic problem with feedback the run time savings are minimal because the cross sections are changing during the iteration. Since this is not the case for the subgroup fixed source problem, the CDP method can be adapted to provide reduced run times.

Figure 19 below illustrates how the computation of the transmission probabilities is performed for a pin cell. On the left is a discretized ray showing 5 segments (blue) with the incoming and outgoing angular fluxes at the ends of the ray on the domain boundary. In this illustration, the outgoing angular flux would be obtained by sequentially evaluating the MOC transmission equation along each segment until the opposite end of the ray is reached. On the right of the figure below is the illustration of how that process can be condensed into evaluating a much simpler equation with a few directional transmission probabilities, $T_{g,m,r}$, $\hat{T}^{for}_{g,m,r}$, and $\hat{T}^{back}_{g,m,r}$. The CDP transmission equations are given as:

$$\varphi^{out,for}_{g,m,r} = T_{g,m,r}\varphi^{in,for}_{g,m,r} + \hat{T}^{for}_{g,m,r}, \tag{18}$$

$$\varphi^{out,back}_{g,m,r} = T_{g,m,r}\varphi^{in,back}_{g,m,r} + \hat{T}^{back}_{g,m,r}, \tag{19}$$

where the subscripts, g, m, and r refer to the group index, angle index and ray index, respectively.

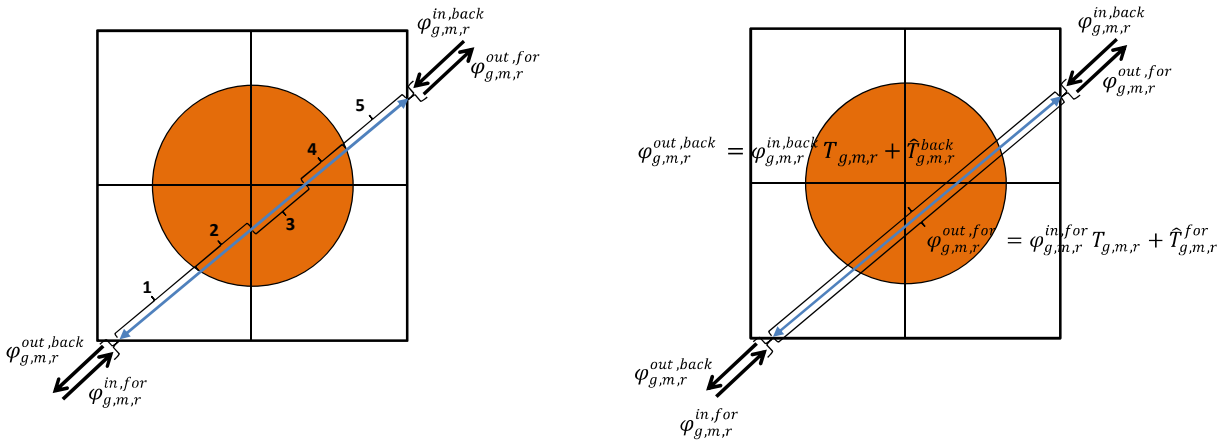


Figure 19. Illustration of conventional MOC evaluating segments along a ray (left) and lumped parameter representation along a ray (right) for a pin cell.

The directional transmission probabilities, $T_{g,m,r}$, $\hat{T}^{for}_{g,m,r}$, and $\hat{T}^{back}_{g,m,r}$ may be derived in a straightforward manner through recursive substitution of the MOC transmission equation for a finite number of segments, N_{seg}. The MOC transmission equation for an isotropic flat source is given as:

$$\varphi^{out}_{g,m,r,i} = \varphi^{in}_{g,m,r,i}\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right) + \frac{q_{g,i}}{\Sigma_{t,g,i}}\left[1-\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\right], \tag{20}$$

where the subscript i denotes the segment within ray r. s is the geometric segment length, and q is the isotropic fixed source. Since continuity of angular flux demands that: $\varphi^{in}_{g,m,r,i+1} = \varphi^{out}_{g,m,r,i}$, performing recursive substitution of Eq. (20) yields:

$$\varphi^{out}_{g,m,r,i} = \varphi^{in}_{g,m,r,i}\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)+\frac{q_{g,i}}{\Sigma_{t,g,i}}\left[1-\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\right],$$

$$\varphi^{out}_{g,m,r,i+1} = \left(\varphi^{in}_{g,m,r,i}\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)+\frac{q_{g,i}}{\Sigma_{t,g,i}}\left[1-\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\right]\right)\exp\left(-\Sigma_{t,g,i+1}s_{m,r,i+1}\right) + \frac{q_{g,i+1}}{\Sigma_{t,g,i+1}}\left[1-\exp\left(-\Sigma_{t,g,i+1}s_{m,r,i+1}\right)\right],$$

$$\varphi^{out}_{g,m,r,i+2} = \left(\begin{array}{l}\left(\begin{array}{l}\varphi^{in}_{g,m,r,i}\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\\ \quad+\frac{q_{g,i}}{\Sigma_{t,g,i}}\left[1-\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\right]\end{array}\right)\exp\left(-\Sigma_{t,g,i+1}s_{m,r,i+1}\right)\\ \quad+\frac{q_{g,i+1}}{\Sigma_{t,g,i+1}}\left[1-\exp\left(-\Sigma_{t,g,i+1}s_{m,r,i+1}\right)\right]\end{array}\right)\exp\left(-\Sigma_{t,g,i+2}s_{m,r,i+2}\right) + \frac{q_{g,i+2}}{\Sigma_{t,g,i+2}}\left[1-\exp\left(-\Sigma_{t,g,i+2}s_{m,r,i+2}\right)\right].$$

…

Performing some algebra on the above and rearranging into form of Eq. (18) or (19) yields the following for the lumped parameters $T_{g,m,r}$, $\hat{T}^{for}_{g,m,r}$, and $\hat{T}^{back}_{g,m,r}$:

$$T_{g,m,r} = \exp\left(-\sum_{i=1}^{N_{seg}}\Sigma_{t,g,i}s_{m,r,i}\right), \tag{21}$$

$$\hat{T}^{for}_{g,m,r} = \sum_{i=1}^{N_{seg}}\left(\frac{q_{g,i}}{\Sigma_{t,g,i}}\left[1-\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\right]\exp\left(-\sum_{j=i+1}^{N_{seg}}\Sigma_{t,g,i}s_{m,r,i}\right)\right). \tag{22}$$

If traversing the reverse direction along the ray, then the product in Eq. (22), changes. Therefore, the expression for $\hat{T}^{back}_{g,m,r}$ is given by:

$$\hat{T}^{back}_{g,m,r} = \sum_{i=1}^{N_{seg}}\left(\frac{q_{g,i}}{\Sigma_{t,g,i}}\left[1-\exp\left(-\Sigma_{t,g,i}s_{m,r,i}\right)\right]\exp\left(-\sum_{j=1}^{i-1}\Sigma_{t,g,i}s_{m,r,i}\right)\right). \tag{23}$$

The expressions for Eq. (22) and (23) may be further simplified using Eq. (18) and (19), provided $T_{g,m,r}$ has been computed. Thus, their final form is given by:

$$\hat{T}^{for}_{g,m,r\,g,m,r} = \varphi^{out,for}_{g,m,r} - T_{g,m,r}\varphi^{in,for}_{g,m,r}, \tag{24}$$

$$\hat{T}^{back}_{g,m,r\,g,m,r} = \varphi^{out,back}_{g,m,r} - T_{g,m,r}\varphi^{in,back}_{g,m,r}. \tag{25}$$

In Eq. (18) and (19), $T_{g,m,r}$ corresponds to the effective attenuation along the ray, and $\hat{T}^{for}_{g,m,r}$ and $\hat{T}^{back}_{g,m,r}$ correspond to the contribution of the outgoing angular flux from the fixed source for the two

directions that may be traversed along the ray, effectively making it similar to an escape probability for the characteristic ray.

In practice to calculate the lumped parameters, one must effectively perform a transport sweep. Once calculated, the lumped parameters must then be stored for each ray in each angle and group; depending on the domain size and ray discretization this memory overhead could become limiting. However, the savings in FLOPs is significant. The MOC transmission equation requires at least 3 FLOPs and 6 data accesses per segment. Considering there are $O(100)$ segments along a ray for an assembly sized domain, and that it typically takes 3 to 5 iterations, one is saving $O(1000)$ FLOPs and data accesses.

The pseudo-code for performing a sweep based on the lumped parameters is given in Figure 20.

```
Loop over batches (b from 1 to N_batch)
    Loop over subgroup fixed source problems ( pg from 1 to N_FSP(b))
        Compute Σ_t,pg for all FSP's from Σ_a,g,c,l and λΣ_p ( pg ← (g,c,l)).
        Compute source q_pg for this group/category/level from λΣ_p.
    Perform first sweep to compute lumped parameters A, B, and C.
    Iterate to converge boundary condition
        Perform "fast" transport sweep for all pg using lumped parameters
        Check for convergence
    Perform final transport sweep accumulating scalar flux
    Loop over subgroup fixed source problems ( pg from 1 to N_FSP(b))
```

Figure 20. Pseudocode for subgroup fixed source problem looping structure with "fast" sweeper

5.2.3 Measured runtime improvements of fast subgroup transport kernel

The improvements covered in the previous sections have been applied to several tests problems . The problems examined were problem 2a, problem 4a-2D, and problem 5a-2D. In each case a 0.05 cm ray spacing, 16 azimuthal angles per octant, and 2 polar angles in a Tabuchi-Yamamoto (TY) [9] quadrature with 3 radial rings in the fuel and 8 azimuthal divisions were used. Each case was run on Titan [11].The number of batches used to partition the fixed source problems is varied between 1 and 10, with the average number of fixed source problems in each batch reported along with the total time spent performing the subgroup calculation (sec), the observed speedup, and the total memory for the problem (GB). Additionally, results from the pre-existing one-group sweeping scheme (1G) are reported to provide a reference for speed-up. It is also noted that the new scheme has a trivial effect on the solution since the convergence of the fixed source problem may be slightly different. The overall impact on the solution was trivial in each case, affecting the eigenvalue by <0.1pcm and pin power results by <0.01%. The results from Problem 2a are given in Table 24. The results from Problem 4a-2D are given in Table 25, and Table 26 shows the results for 5a-2D, which was run in parallel on 73 processors.

Table 24. Speed-up subgroup time for problem 2a

Number of Batches	Average Batch Size	Subgroup Calculation Time (sec)	Speed-up	Total Memory (GB)
1	272.0	8.17	2.98	0.50
2	136.0	8.25	2.95	0.35
3	90.7	8.26	2.95	0.30
4	68.0	8.42	2.89	0.27
5	54.4	8.70	2.80	0.26
6	45.3	8.78	2.77	0.25
7	38.9	8.87	2.75	0.25
8	34.0	8.92	2.73	0.25
9	30.2	9.52	2.56	0.25
10	27.2	9.58	2.54	0.25
1G (272)	1	24.36	---	0.20

Table 25. Speed-up subgroup time for problem 4a-2D

Number of Batches	Average Batch Size	Subgroup Calculation Time (sec)	Speed-up	Total Memory (GB)
1	272.0	80.22	2.78	1.99
2	136.0	78.25	2.85	1.43
3	90.7	77.55	2.87	1.25
4	68.0	79.52	2.80	1.16
5	54.4	80.59	2.77	1.11
6	45.3	77.62	2.87	1.07
7	38.9	84.96	2.62	1.04
8	34.0	79.43	2.81	1.04
9	30.2	81.97	2.72	1.04
10	27.2	84.11	2.65	1.04
1G (272)	1	222.91	---	0.88

Table 26. Speed-up subgroup time for problem 5a-2D (73 procs)

Number of Batches	Average Batch Size	Subgroup Calculation Time (sec)	Speed-up	Total Memory (GB)
1	272.0	76.32	2.48	67.53
2	136.0	56.14	3.37	44.98
3	90.7	52.10	3.63	37.98
4	68.0	48.97	3.86	34.41
5	54.4	48.82	3.87	33.03
6	45.3	49.58	3.81	32.89
7	38.9	50.97	3.71	32.87
8	34.0	49.38	3.83	32.87
9	30.2	52.31	3.61	32.87
10	27.2	53.96	3.50	32.87
1G (272)	1	188.95	---	23.74

The trends here indicate the fundamental issue of optimization: trading run time for memory. In the worst case of a single batch, the memory overhead increases by roughly a factor of 2.84. Additionally, this case takes the most time, although still provides speedup. The hypothesis to explain the degraded speedup is the extra overhead introduced by the spatial decomposition data passing, which now is passing buffers of data that contain a very substantial number of more data entries than the others. It is also possible that the reason this case has less speedup is the vector

length on the inner most loop becomes too large and cannot fit in the L1 cache. These effects however can be easily mitigated when increasing the batches, where an optimum point is reached around 5 batches. For this case the average memory increase per process compared to the 1G case is 130 MB.

In terms of overall speedup, a nearly 3.9x speedup is observed in the 5 batch case. While a large portion of this is attributable to the characteristic direction transmission probability approach, it is likely that this speedup is increased over what was observed in smaller problems because the data passing is using larger buffers. In the 1G scheme, the angular flux data is passed using one-group buffers, whereas the new scheme uses buffers of several groups.

5.3 One-group Subgroup Method

The previous section focused on solving a set of subgroup fixed source problems as efficiently as possible. The focus in this section is to investigate the possibility of reducing the number of fixed source problems that must be solved.

As noted in the previous sections, the standard subgroup method solves a fixed source problem for each resonant energy group, resonance category, and subgroup level. A typical set of resonance categories used for most PWR applications in MPACT is shown in Table 27.

Table 27. A typical set of resonance categories in the ORNL 47-group library

Category	Isotopes
1	U-238
2	U-235, other actinides and important FPs
3	Clad isotopes
4	Poison Isotopes (AIC, Gd, Hf, etc.)
5	Tungsten

Regardless of the category, fixed source problems are solved for each level in each resonant energy group. However, the cross section resonances of the various isotopes in a particular category can be very different from those isotopes in a different category. In the case of Zr-91 and Zr-96, only two groups, ranging from 130.1-9118.8eV, of the 17 resonance groups have resonances for these isotopes, while U-238 has significant resonances in 5 resonant groups. If a resonance group has no resonances for a particular isotope or category, it is unnecessary to perform the FSP calculations. Therefore, the majority of the fixed source problems for the clad isotopes to not need to be solve.

Instead of solving the MG subgroup problem, a new 1G subgroup method is developed and implemented. This is developed in a way so that it may be generally applied to any category. The detailed theory of this development is omitted here for brevity, but the complete description may be found in [29].

Theoretically, 1G subgroup can be applied to all the resonance categories. Due to the two approximations made in formulating the 1G subgroup equations, we investigate the applicability of the 1G subgroup with regard to the different resonance categories to assess the overall accuracy.

Four options were tested:

1. MG: regular MG subgroup calculation for all categories (reference)
2. 1G: 1G subgroup calculation for all categories
3. 1G-Clad: 1G subgroup calculation only for clad category
4. 1G-AllButU: 1G subgroup calculation used for non-uranium categories

Table 28 summarizes the results for VERA Problem 2. The results indicate that:

- Enabling 1G subgroup for clad calculation is sufficiently good as compared to MG subgroup.
- For some cases with resonance absorbers in category 4 (AIC in 2g, Gd in 2o and 2p), we will have small eigenvalue difference of ~20 pcm if using 1G-AllButU.
- 1G subgroup for all categories has an eigenvalue bias of -40 to -90pcm.
- The last two rows show the maximum difference of pin power for all the above cases. Compared to the reference, all three methods result in trivial difference.

Table 28. Results of 1G subgroup compared to MG subgroup

Case	MG(keff)	Eigenvalue difference (pcm) compared with MG		
		1G	1G-Clad	1G-AllButU
1a	1.186242	-71.6	0.1	0.1
1b	1.181856	-79.5	0.1	0.1
1c	1.171485	-85.5	0.1	0.1
1d	1.162268	-79.1	0.1	0.1
2a	1.180841	-60.3	0.1	0.0
2b	1.182547	-67.9	0.1	0.1
2c	1.173164	-73.4	0.1	0.1
2d	1.164766	-67.3	0.1	0.1
2e	1.069116	-55.0	0.1	0.1
2f	0.975980	-51.1	0.1	0.1
2g	0.849946	-22.0	0.1	22.7
2h	0.791799	-37.2	0.1	0.0
2i	1.178756	-59.8	0.1	0.0
2j	0.975239	-51.1	0.1	0.1
2k	1.020060	-53.1	0.1	0.1
2l	1.017051	-49.8	0.1	0.0
2m	0.936860	-44.7	0.1	0.0
2n	0.868073	-42.7	0.1	0.0
2o	1.047530	-61.6	0.1	-9.6
2p	0.927848	-62.0	0.1	-17.0
2q	1.171205	-85.0	0.0	-0.1
		RMS: 0.01%	RMS: 0.00%	RMS: 0.01%
		Max: 0.03%	Max: 0.00%	Max: 0.02%

The computing time of subgroup calculation is very small for most of the test cases above. Therefore problem 4a-2D is analyzed to provide a better estimate of the speed-up of the subgroup calculation by incorporating the 1-group formulation. In this problem it is observed that the 1G-clad case which provides the most accurate results gives a 27% speed-up in the subgroup. This is close to what is expected since nearly a quarter of the subgroup fixed source problems are no longer needed. All of the cases in Table 29 utilizing the 1-group subgroup have a reduction in memory. For the nominal

case of 1G-clad the memory reduction is just over 6%. Naturally, if additional speed-up is desired then there is the potential to run with the 1G-AllButU case to get a total of 62% speedup in the subgroup and a 12% memory reduction.

Table 29. Speed-up 1G-subgroup for problem 4a-2D

Method	Subgroup Calculation Time (sec)	Speed-up	Total Memory (MB)
MG	68.2	-	523.6
1G	11.7	5.83	395.0
1G-Clad	53.6	1.27	490.8
1G-AllButU	42.2	1.62	458.0

5.4 Summary of Cross Section Calculation Improvements

Significant performance improvements have been made to the cross section calculation in MPACT including the subgroup calculation and the calculation of macroscopic cross sections and fission spectrum. Table 30 summarizes the information from these improvements. For the macroscopic data calculation, the speedups are all independent for each routine. For the subgroup FSP calculation, the multiplication of the speedups gives the overall efficiency gains of subgroup calculation. Note the overall memory increase (34MB per assembly) is well acceptable given the significant time savings. Most of the improvements have no (or trivial) effect on the accuracy. Some of the changes even achieve better results, as noted in [29].

Table 30. Speed-up subgroup time for problem 5a-2D

	Item	Speedup[a]	Change in Memory usage[b]	Effect on results
Macroscopic Data Calculationa	calcMacroChi	84x	+8 MB	No
	calcMacroXS	3x	+1 MB	No
	Segev	33x	0 MB	Better
Subgroup FSP	New Sweeper	3.9x	+63 MB	Trivial[c]
	1-group subgroup	1.3x	-13 MB	Trivial[c]
Overall	Estimated	2.5x	+59 MB	Trivial

[a] - Speedup observed by 2D core depletion.
[b] – The additional memory usage per processor due to the improvement, assuming the geometry is decomposed to one 2D assembly per processor.
[c] - 'Trivial' means the difference less than 1pcm k_{eff} and 0.1% max pin power.

The continuing work relating to the cross section performance improvement includes,

- Optimize Subgroup FSP discretization
- Further improve the efficiency of *calcMacroXS* (still more than 10% of overall computing time).
- Improve the efficiency of ESSM and ESSM-X. The new subgroup sweeper can also be used in ESSM to speed up the FSP. In addition, optimizations are needed for ESSM-X to efficiently solve the quasi-1D slowing-down equation.
- We expect a transition to the AMPX MG library in the near future, which means many approaches developed in the L3:RTM.XSNP13.01 milestone need to be reexamined. For example, one of the issues is the *Segev* interpolation, which will be more used in the AMPX

MG library. If it becomes an efficiency concern again, a new scheme should be developed to speed up the interpolation.

6. SUMMARY & ONGOING WORK

The summary of the overall speed-up for some of the selected test problems is given in Table 31. The incremental improvement from each major change-set described in this report is shown chronologically for each of these problems in Figure 21 through Figure 24.

Table 31. Aggregate speed-up of MPACT for various test problems

Problem	Initial Time (s)	New Time (s)	Overall Speedup
2a	41.55	13.10	3.17
4a-2D	335.93	149.64	2.24
5a-2D	11339.49	2353.52	4.82
5a	875.07	312.40	2.80

The change-set labels are described as follows:

- *Baseline*: This corresponds to the initial performance. Results are consistent with those presented in Section 2.2 and should closely resemble the performance of MPACT v2.2.0 released with VERA 3.4.
- *XS Performance*: These changes correspond to the work discussed in Section 5.1 and 5.3. The goal of this change-set was minimize the time spent computing cross sections. The changes in Section 5.1 likely provided the greatest speed-up to MPACT of all work described here in. This is particularly evident with depletion and problem 5a-2D (FIGUREX).
- *Stabilized CMFD*: This change-set corresponds to the work discussed in Section 4.1. The goal of this change-set was to guarantee stability and optimum convergence of CMFD with larger coarse mesh.
- *Jacobi In-scatter MOC*: This change-set corresponds to the work discussed in Section 3.1. The goal of this change-set was to implement an algorithm that was well known in the reactor physics community with the expectation that the MOC sweep would be sped up by roughly a factor of 2x.
- *Fast Subgroup*: This change-set corresponds to the work described in Section 5.2. The goal of this change-set was to provide a "lightweight" transport kernel optimized for iterating on the boundary condition in a purely absorbing media, and to take advantage of the speed-up provided by Section 3.1 by "vectorizing" the subgroup fixed source problems.
- *CMFD Speed-up**: This change-set corresponds to the work described in Section 4.2. Results are still preliminary, and an alternative approach described in [15] has shown similar speed-up.

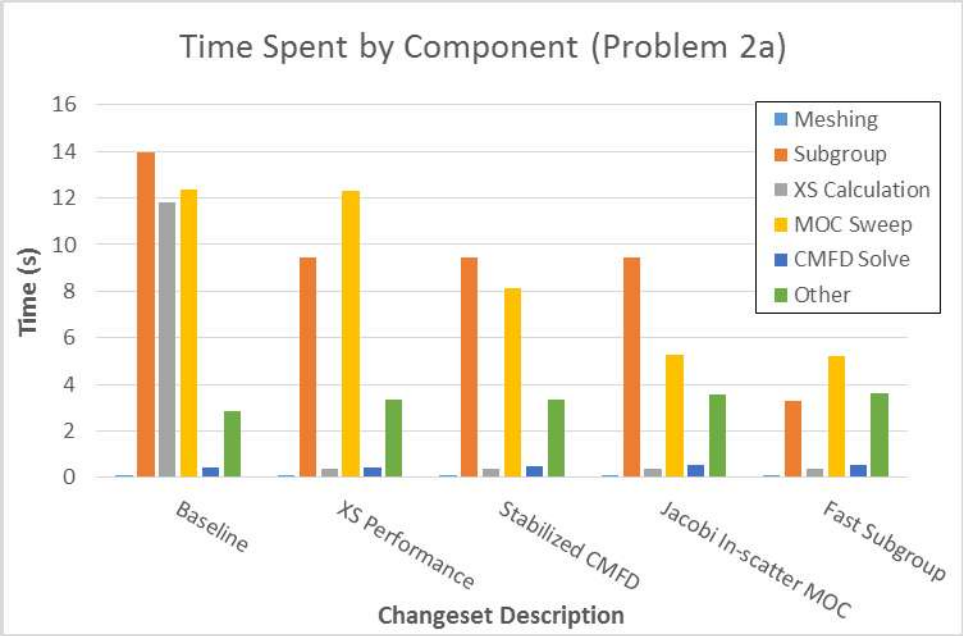


Figure 21. Incremental performance improvements for problem 2a

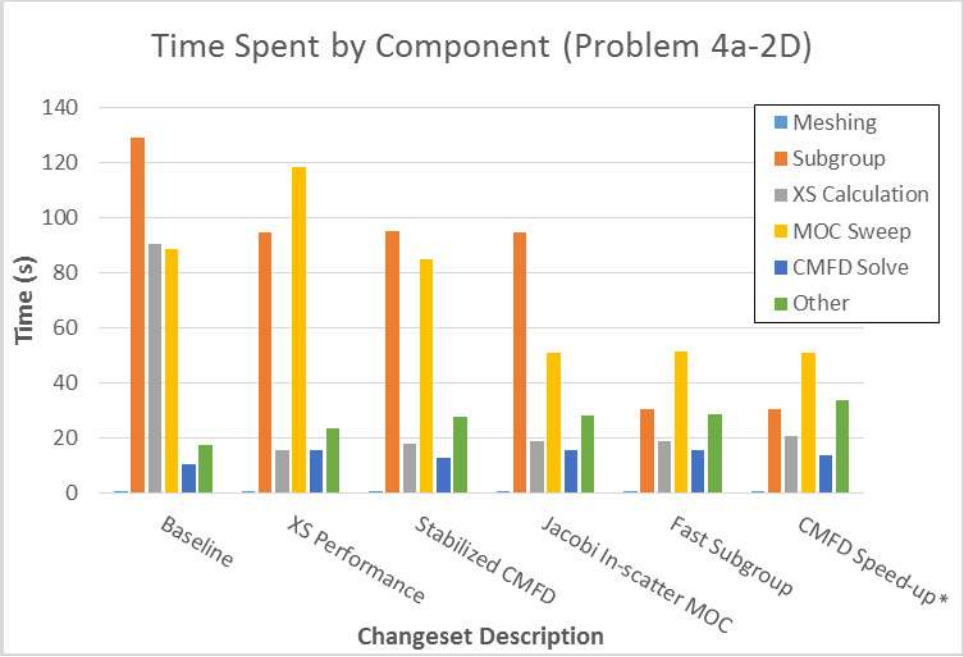


Figure 22. Incremental performance improvements for problem 4a-2D

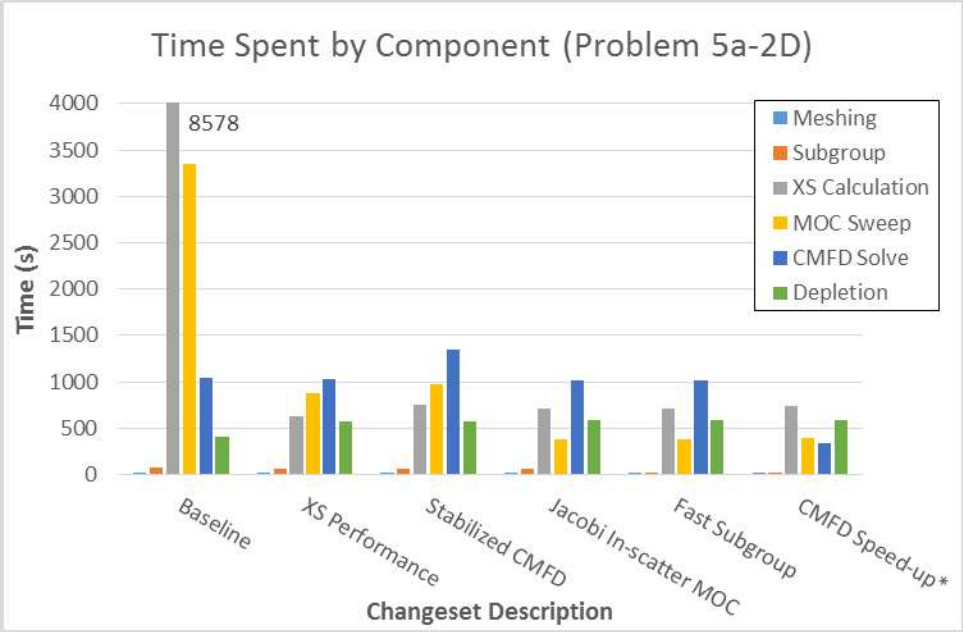


Figure 23. Incremental performance improvements for problem 5a-2D

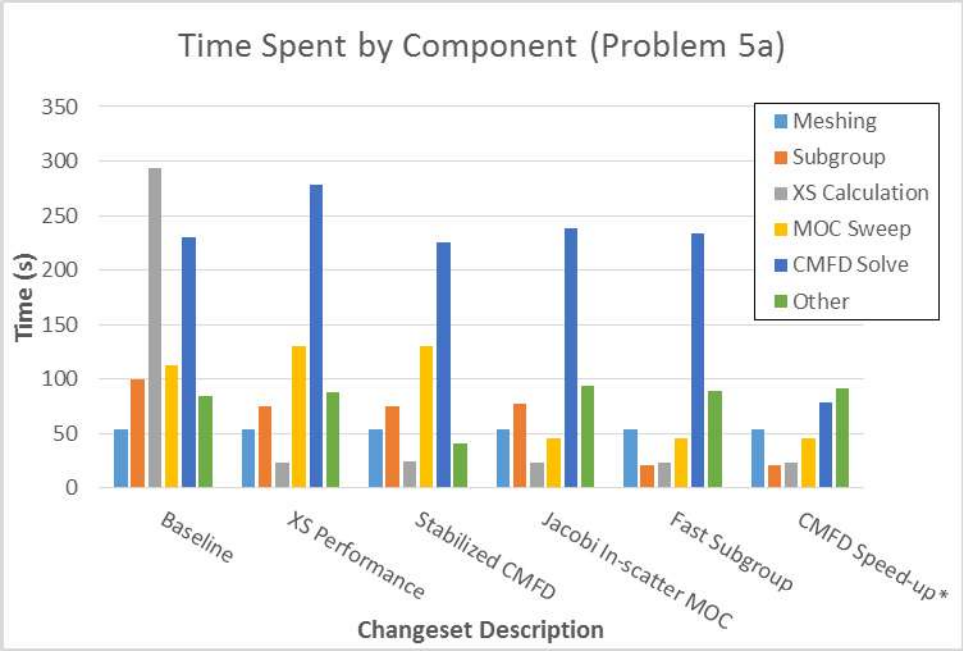


Figure 24. Incremental performance improvements for problem 5a

These results indicate that MPACT has achieved an overall nominal speed-up of ~3x. As discussed in Section 4, the last change-set corresponding to this section is not quite finalized yet. Thus the "*" indicates preliminary results. It is certainly expected that this amount of speed-up will be achievable with a stable and robust iteration scheme.

Figure 25 shows the changes in the number of iterations and Figure 26 shows the total wall time and memory usage for problem 5a from each incremental change. This illustrates the trade-off of run time and memory that was made for several of the improvements.

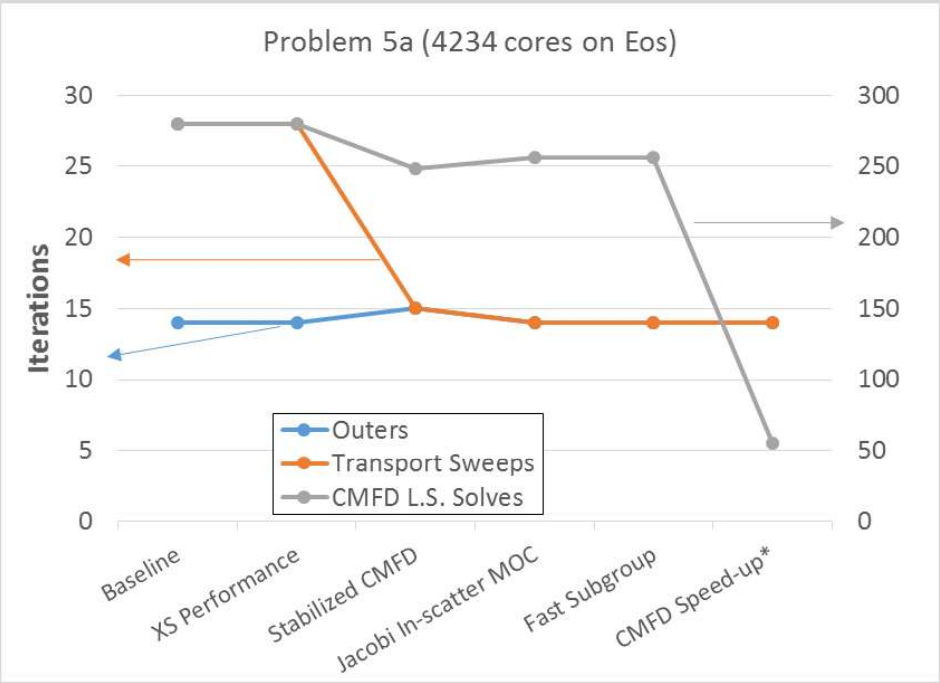


Figure 25. Incremental changes in iterations for problem 5a

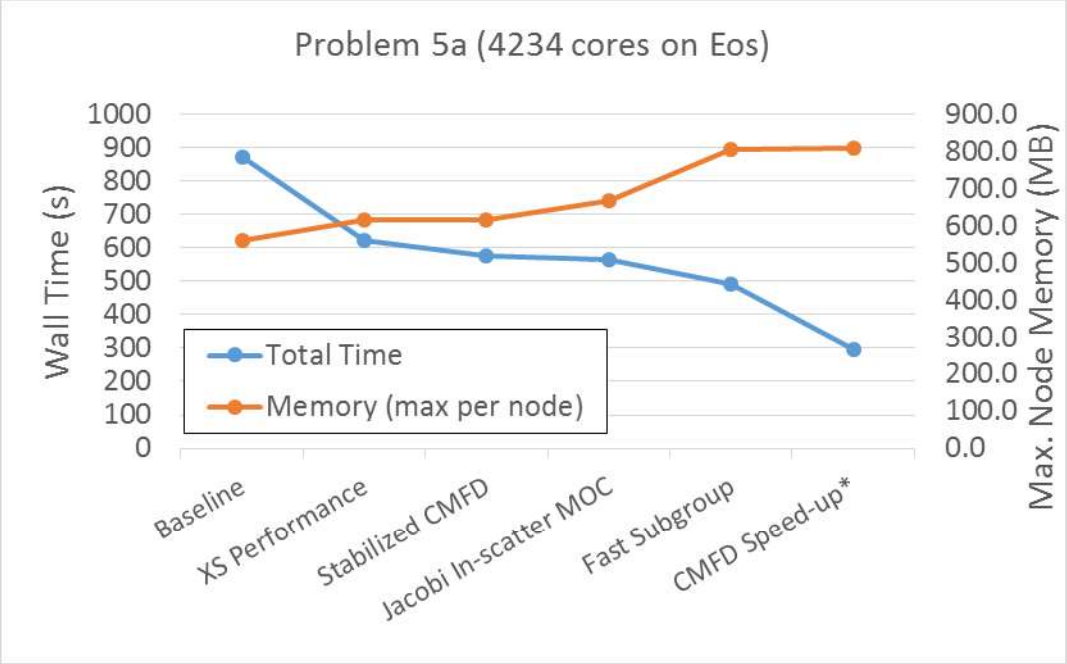


Figure 26. Incremental changes in compute resources for problem 5a

To continue to improve the overall speed-up will be challenging. The work thus far had to find optimizations that gave approximately the same amount of speed-up across a breadth of numerical methods. This is not a simple task. It is likely that continued improvements would become more costly in terms of developer time. However, there are some other areas within MPACT that may still

be optimized. The future work for performance improvements in MPACT should focus on the following:

- *Depletion*
  One major computational component not discussed in detail in this report has been performance improvements related to depletion. Several other time stepping algorithms have been implemented besides the one tested in these results. The other algorithms: semi-predictor-corrector and post-corrector are still undergoing testing. Each of these algorithms eliminate the steady-state eigenvalue solve of the corrector step, and thus have the opportunity to provide an overall savings of nearly 1.5x for cycle long depletions. Further optimization of the depletion chains and reactions may also be possible, as well as optimization of the matrix exponential calculation.

- *Improve Shared Memory Parallelism*
  Presently it is mostly the MOC routines that make use of shared memory parallelism through OpenMP. It is possible that better overall utilization of shared memory parallelism could be achieved. The relevant areas where shared memory parallelism could be implemented include the linear system solve for CMFD, computation of the macroscopic cross sections, and the point depletion calculation. In particular, for the macroscopic cross section calculation and point depletion calculation, the loop over cross section mesh regions can be parallelized offering a fairly simple and coarse grained approach that should scale well.

- *Discretization Optimization*
  This category represents any number of ways the model discretization may be "optimized". This is currently an activity for the subgroup fixed source problems to reduce the number of angles required for this calculation. Other approaches might include optimization of the group structure, optimization of the spatial mesh, and utilization of subplane. For the spatial mesh in particular it may be possible to improve performance through the use of a linear source [37]. Even outside of it may be possible to shed mesh regions elsewhere in the problem. Effective use of subplane will reduce the overall number of processors and MOC planes in 3D models, and push more of the computational burden to the CMFD.

- *Continued improvement of macroscopic cross section calculation*
  As noted in Figure 23, the cross section calculation is still an extremely burdensome part of the overall calculation in MPACT. This is especially true with depletion when there are many more isotopes present that must be evaluated.

- *Improvements in model setup and code initialization*
  This activity would fall under the category of miscellaneous improvements. For cycle depletion type calculations, this component of the simulation is a very small fraction of the overall run time as seen Figure 23. However for single state calculations, as in Figure 24, it is observed that this part of the simulation may be a nontrivial fraction. Improvements to the model setup would effectively mean reducing the time searching for and identifying duplicate components of the mesh.

- *Maturation of algorithms for next generation hardware*
  Thus far the work looking into next generation hardware that is typified by GPUs and the Intel MIC has been exploratory. The current results suggest it may be possible to achieve some speed-up, but significant effort will be needed to get something to the production level. Waiting on this activity may also be prudent to reduce developer burden while programming models and compilers are still maturing.

REFERENCES

[1] A. Godfrey, et al., "VERA Benchmarking Results For Watts Bar Nuclear Plant," CASL Technical Report: CASL-U-2015-0206-000, June 26 (2015). http://www.casl.gov/docs/CASL-U-2015-0206-000.pdf

[2] R.A. Bartlett, M.A. Heroux, and J.M. Willenbring, "TriBITS Lifecyle Model Version 1.0," Sandia Report: SAND2012-0561, Feb. (2012). http://web.ornl.gov/~8vt/TribitsLifecycleModel_v1.0.pdf

[3] A. Godfrey, "VERA Core Physics Benchmark Progression Problem Specifications," Revision 4, CASL Technical Report: CASL-U-2012-0131-004, August 29 (2014). http://www.casl.gov/docs/CASL-U-2012-0131-004.pdf

[4] N.H. Larsen, "Core Design and Operating Data for Cycles 1 and 2 of Peach Bottom 2," EPRI Technical Report, NP-563, June, (1978). http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=NP-563

[5] M.A. Smith, E.E. Lewis, and B.C. Na, "Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation - A 2-D/3-D MOX Fuel Assembly Benchmark (C5G7 MOX Benchmark)," Tech. Rep. NEA/NSC/DOC(2003)16, Organisation for Economic Co-operation and Development Nuclear Energy Agency (2003). https://www.oecd-nea.org/science/docs/2003/nsc-doc2003-16.pdf

[6] Oak Ridge Leadership Computing Facility. "Eos User Guide" (2014). https://www.olcf.ornl.gov/support/system-user-guides/eos-user-guide/

[7] S. Stimpson, B. Collins, and B. Kochunas, "MOC Efficiency Improvements Using a Jacobi Inscatter Approximation," CASL Technical Report: CASL-I-2-2016-1056-001, March 30 (2016).

[8] W.R. Boyd, K.S. Smith and B. Forget, "A Massively Parallel Method of Characteristics Neutral Particle Transport Code for GPUs," *Proc. M&C 2013*, Sun Valley, ID, USA, May 5-9 (2013).

[9] A. Yamamoto et al., "Derivation of Optimum Polar Angle Quadrature Set for the Method of Characteristics Based On Approximation Error for the Bickley Function," *Journal of Nuclear Science and Technology*, **4**, No. 2, p. 129-136 (2007). http://dx.doi.org/10.1080/18811248.2007.9711266

[10] K. S. Kim et al., "Development of a New 47-Group Library for the CASL Neutronics Simulators," *Proc. M&C 2015*, Nashville, Tennessee, April 19–23 (2015).

[11] Oak Ridge Leadership Computing Facility. "Titan User Guide" (2014). https://www.olcf.ornl.gov/support/system-user-guides/titan-user-guide/

[12] B. Herman, et al., "Improved diffusion coefficients generated from Monte Carlo codes," *Proc. M&C 2013*, Sun Valley, ID, USA, May 5-9 (2013).

[13] B.C. Yee, B. Kochunas, and T. Downar, "MPACT Modeling of DIMPLE Benchmark Problems," CASL Technical Report: CASL-U-2016-1045-000, Feb. 2nd (2016).

[14] Texas Advanced Computing Center, "TACC Stampede User Guide" (2016). https://portal.tacc.utexas.edu/user-guides/stampede

[15] B. Collins, et al., "Performance Improvements for Coarse Mesh Finite Difference Acceleration," CASL Technical Report: CASL-X-2016-1106-000, May 30 (2016).

[16]N.Z. Cho and C.J. Park, "A Comparison of Coarse Mesh Rebalance and Coarse Mesh Finite Difference Accelerations for the Neutron Transport Calculations," *Proc. of M&C 2003*, Gatlinburg, USA, April 6-11, (2003).

[17]S.G. Hong, K.S. Kim, and J.S. Song, "Fourier Convergence Analysis of the Rebalance Methods for Discrete Ordinates Transport Equations in Eigenvalue Problems," Nucl. Sci. Eng., 164, pp. 33-52 (2010). http://dx.doi.org/10.13182/NSE09-18

[18]N.Z. Cho, G.S. Lee, and C.J. Park, "Partial Current-Based CMFD Acceleration of the 2D/1D Fusion method for 3D Whole-Core Transport Calculations," *Trans. Am. Nucl. Soc.*, **88**, 594 (2003).

[19]M. Jarrett, et al., "Analysis of Stabilization Techniques for CMFD Acceleration of Neutron Transport Problems," *Nucl Sci. Eng.*, (in preparation).

[20]L. Li, K. Smith, and B. Forget, "Techniques for Stabilizing CMFD in MOC," *Proc. M&C 2015,*Nashville, TN, USA. April 19-23 (2015).

[21]A. Zhu, et al., "An Optimized Artificial Diffusion Coarse Mesh Finite Difference Method to Accelerate Neutron Transport Calculations," *Annals of Nuclear Energy*, **95**, pp. 116-124 (2016). http://dx.doi.org/10.1016/j.anucene.2016.05.004

[22]A. Zhu, et al., "Theoretical Convergence Rate Lower Bounds for Variants of Coarse Mesh Finite Difference to Accelerate the Neutron Transport Calculations," *in preparation*, (2016).

[23]A. Godfrey, personal communication (2016).

[24]B.W. Kelley and. E.W. Larsen, "A consistent 2D/1D approximation to the 3D neutron transport equation," *Nuclear Engineering Design*, **295**, pp. 598-614 (2015). http://dx.doi.org/10.1016/j.nucengdes.2015.07.026

[25]K.S. Smith and J.D. Rhodes III, "Full-core 2-D, LWR core calculations with CASMO-4E," *Proc. PHYSOR 2002*, Seoul, ROK, October 7-10 (2002).

[26]E.L. Wachspress, *Iterative Solution of Elliptical Systems*, Prentice Hall, Inc., Englewood Cliffs, New Jersey (1966).

[27]B.C. Yee, et al., "Space-Dependent Wielandt Shift Methods for Multigroup Diffusion Eigenvalue Problems," *Trans. Am. Nucl. Soc.*, **114**, 1, pp. 753-756 (2016). http://epubs.ans.org/download/?a=38668

[28]E.R. Wolters, E.W. Larsen, and W.R. Martin, "Hybrid Monte Carlo-CMFD Methods for Accelerating Fission Source Convergence," *Nucl. Sci. Eng.,***174**, pp. 286-299 (2013). http://dx.doi.org/10.13182/NSE12-72.

[29]Y. Liu, et al., "Runtime Improvements to the Cross Section Calculation in MPACT," CASL Technical Report: CASL-X-2016-1105-000, May 30 (2016).

[30]H.G. Joo, J.Y. Cho, K.S. KIM, C.C. Lee, and S.Q. Zee, "Methods and Performance of a Three-Dimensional Whole-Core Transport Code DeCART," *Proc. of PHYSOR 2004*, Chicago, IL, USA, April 25-29 (2004).

[31]Y.S. Jung, et al., "Practical numerical reactor employing direct whole core neutron transport and subchannel thermal/hydraulic solvers," *Annals of Nuclear Energy*, 62, pp. 357-374 (2013).

[32]M. Segev, "Interpolation of Resonance Integral," *Nucl. Sci. Eng.,***79**, 113 (1981). http://dx.doi.org/10.13182/NSE81-2

[33] Y.S. Jung and H.G. Joo, "Investigation of Intra-Pellet Temperature Profile Treatment in nTRACER Resonance Calculation," The 4th I-NERI meeting, South Korea, Nov. 19 (2015).

[34] S. Stimpson, et al., "Subgroup Self-Shielding Efficiency Improvements," CASL Technical Report: CASL-I-2016-1063-000, April 1 (2016).

[35];S.G. Hong and N.Z. Cho, "Method of Characteristic Direction Probabilities for Heterogeneous Lattice Calculation," *Nucl. Sci. Eng.,* **132**, pp. 65-77 (1999). http://epubs.ans.org/download/?a=2049

[36] Z. Liu, et al., "Theory and analysis of accuracy for the method of characteristics direction probabilities with boundary averaging," *Annals of Nuclear Energy*, **77**, pp. 212-222 (2015). http://dx.doi.org/10.1016/j.anucene.2014.11.016

[37] R.M. Ferrer and J.D. Rhodes III, "A Linear Source Approximation Scheme for the Method of Characteristics," *Nucl. Sci. Eng.,* **182**, pp. 151-165 (2016). http://dx.doi.org/10.13182/NSE15-6