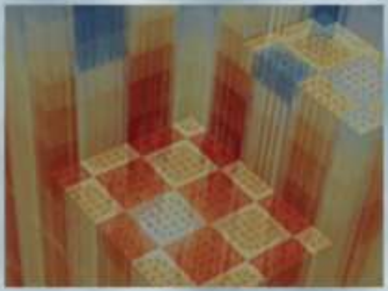
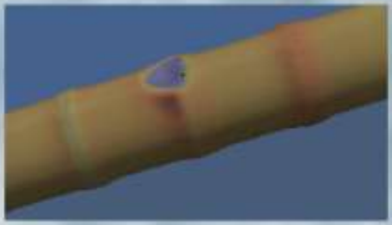
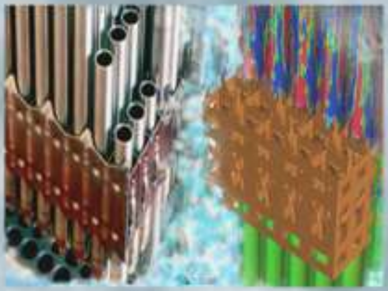
CASL-U-2016-1171-000

Summary of VERA Core Simulator Performance Improvements

Benjamin Collins (ORNL)
Steven Hamilton (ORNL)
Mike Jarrett (UM)
Kang Seog Kim (ORNL)
Brendan Kochunas (UM)
Yuxuan Liu (UM)
Scott Palmtag (Core Physics)
Robert Salko (ORNL)
Shane Stimpson (ORNL)
Alex Toth (SNL)
Ben Yee (UM)

8/31/2016

REVISION LOG

Revision	Date	Affected Pages	Revision Description
0	8/31/2016	All	Released Version

Document pages that are:

Export Controlled ________________none__________________________________

IP/Proprietary/NDA Controlled______none__________________________________

Sensitive Controlled_______________none__________________________________

Requested Distribution:

To:

Copy:

EXECUTIVE SUMMARY

This report describes the performance improvements made to the VERA Core Simulator (VERA-CS) during FY2016.

At the beginning of the year, a timing study was performed to identify areas of the code where the run-times could be improved. This study identified six broad areas for improvement:

- Cross section lookup and CMFD stabilization,
- CTF parallel partitioning,
- MOC methodology,
- Subgroup methodology,
- MOC parallel partitioning, and
- CMFD Krylov solution techniques.

The goal of this work was to reduce the VERA-CS run-times so a complete cycle depletion could be run with 1,000 cores overnight (i.e., the “1,000 core depletion” criterion). This goal was set by our industry partners so they could achieve reasonable turnaround on medium-sized computer clusters.

The results presented in this report show a code speed-up by a factor of 5, which successfully satisfies the goal set by industry.

CONTENTS

FIGURES

TABLES

ACRONYMS

CASL	Consortium for Advanced Simulation of Light Water Reactors
CMFD	Coarse Mesh Finite Difference Method
CP	challenge problem
FP	fission product
FSR	flat source region
HZP	Hot Zero Power
HFP	Hot Full Power
LWR	Light Water Reactor
MOC	Method of Characteristics
ORNL	Oak Ridge National Laboratory
PHI	Physics Integration
PWR	Pressurized Water Reactor
RI	resonance integral
SNL	Sandia National Laboratory
SOR	successive over-relaxation
TCP_0	transport-corrected P_0 scattering
T/H	thermal-hydraulics
UM	University of Michigan
VERA	Virtual Environment for Reactor Applications
VERA-CS	VERA Core Simulator

1. INTRODUCTION

Since the early history of the Consortium for Advanced Simulation of Light Water Reactors (CASL), the development of the VERA Core Simulator (VERA-CS) has focused on the capability needed to deplete physical reactors and help solve CASL challenge problems. This capability required the accurate simulation of many operating cycles of a nuclear power plant. This capability was demonstrated in late 2015 with the simulation of 12 cycles of Watts Bar Unit 1 [1].

Now that the basic capability has been developed, this year's emphasis has been on improving VERA-CS run-times, targeting the ability to run one full cycle depletion case in less than 24 hours on 1,000 computer cores. This target was set so that industry partners with limited computer resources can run the codes on industry-sized clusters with reasonable turnaround times.

This report describes many of the changes made to VERA-CS over the past year to successfully achieve the "1,000 core" goal.

The first section of this report introduces two test problems used to assess the run-time performance of VERA-CS using a source dated February 2016. The next section provides a brief overview of the major modifications made to decrease the computational cost. Each of these modifications is documented in more detail in the individual milestone reports and conference papers cited as references. Following the descriptions of the major improvements, the run-time for each improvement is shown. Conclusions on the work are presented, and further follow-on performance improvements are suggested.

2. OVERVIEW OF VERA-CS PERFORMANCE

To understand where the computer resources were being spent in VERA-CS, an initial performance assessment of the code was made. The run-times referred to in this section were obtained using an executable built from the source on February 7, 2016.

Two VERA progression problems [2] were evaluated. The first case is Progression Problem 7, which is a quarter-core hot full power (HFP) case with geometry based on Watts Bar Unit 1, as shown in Figure 1.

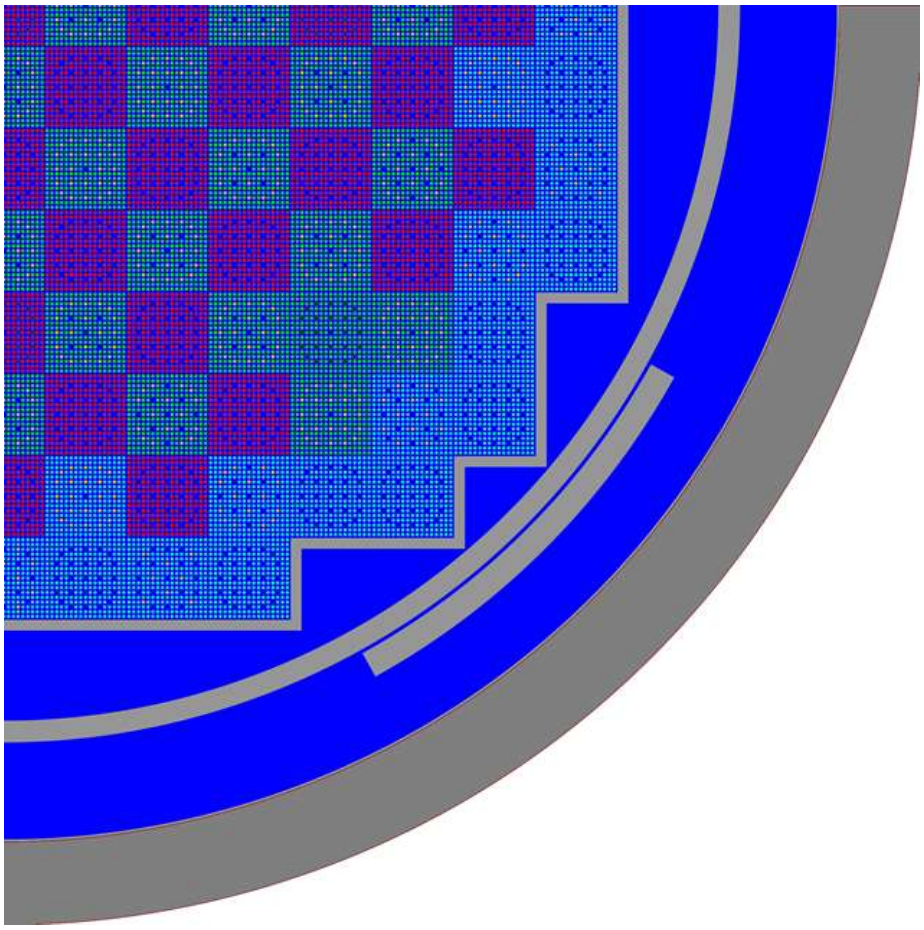


Figure 1. Watts Bar Unit 1 Cycle 1 core layout

The HFP case was run on 870 processors using the Nuclear Science and Engineering Directorate (NSED) cluster and took 2 hours and 21 minutes to run. (Note that this case is a single statepoint, not a depletion.)

A timing analysis was performed on this calculation, and the run-time was split into several different calculational components. The run-time spent in each component is shown in Figure 2. Each of these components was examined to identify ways to optimize them and decrease run-times.

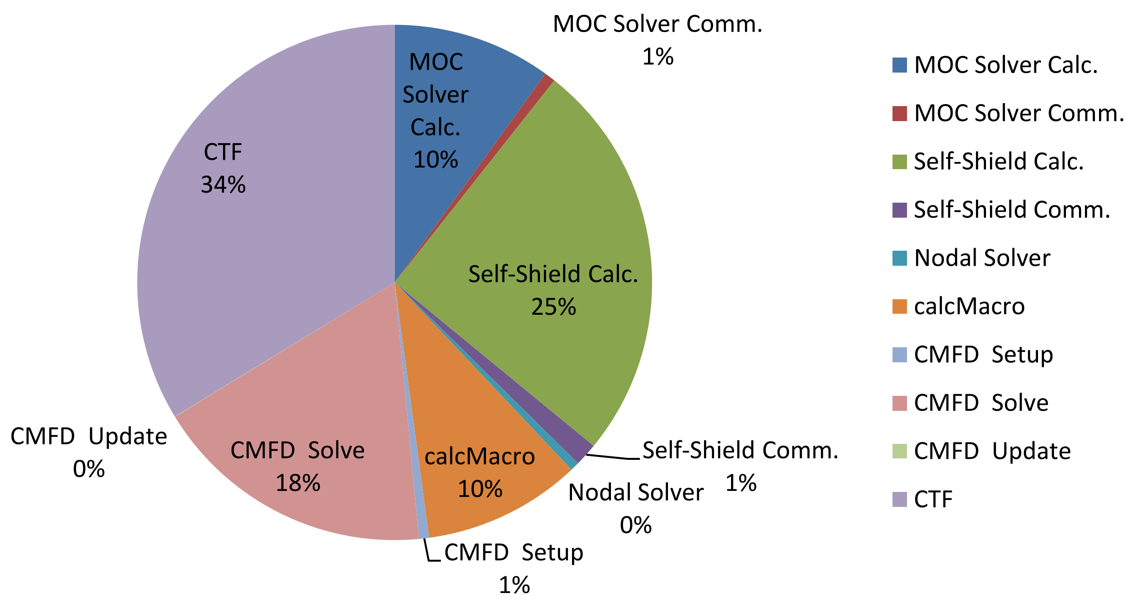


Figure 2. Initial fraction of run-time for HFP case.

While the Problem 7 run-times provide valuable information, they do not include any run-time contributions resulting from depletion. Therefore, a second case was evaluated that included depletion. To save computing time, only the first five states of the Watts Bar cycle 1 depletion are used in this evaluation. (A real cycle depletion will have approximately 20–25 depletion steps.) The depletion problem was also run on 870 processors on the NSED cluster and took 13 hours and 39 minutes to complete. The distribution of each run-time component is shown in Figure 3.

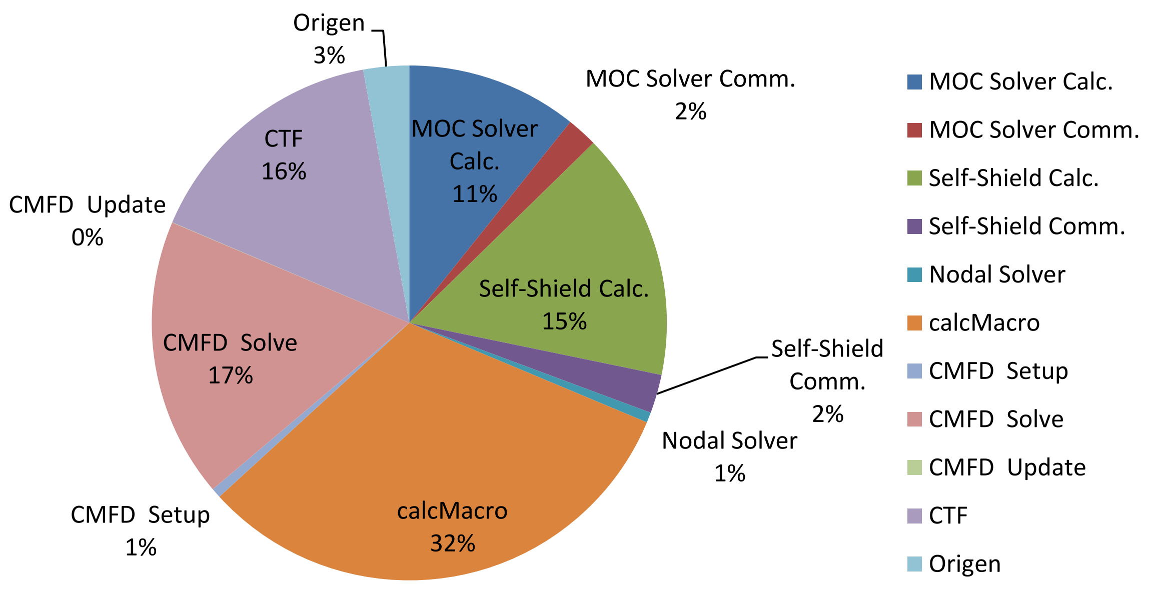


Figure 3. Initial fraction of run-time for core depletion.

Unlike the HFP case, the limiting component for the depletion is the cross section generation. This is due to the significant increase in isotopics and microscopic cross section evaluations that occur once the core starts to deplete. In addition to depletion, CTF, coarse mesh finite difference (CMFD), and the shielding calculation all contribute significantly to the run-time.

With this initial assessment complete, the identified major components of VERA-CS were examined to optimize performance and improve run-times.

3. OVERVIEW OF INDIVIDUAL IMPROVEMENTS

This section describes the key individual improvements made to VERA-CS to improve the overall run-time. More details are provided in individual milestone reports, and this section only includes a high-level overview of the work. The improvements are described in the order of implementation into VERA-CS to give the reader a chronological view of the development of VERA-CS over the course of the year.

3.1 Cross-section Improvements

The first improvement was in the calculation of the macroscopic cross section and fission spectrum [3]. The related routines in MPACT include *calcMacroXS*, *calcMacroChi* and *Segev*.

The initial MPACT profiling results show that for a 2-D quarter-core depletion case, about one third of the computing time is spent on computing the fission spectrum. In MPACT, the effective fission spectrum is computed for each fuel region, as follows.

For the multigroup neutron transport equation, the fission source in group *g* is written as

$$F_g(r) = \sum_{iso} \chi_{g,iso}(r) \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \Phi_{g'}(r) . \qquad (3.1.1)$$

Although the fission spectrum is dependent on the fissionable isotope, an effective fission spectrum $\chi_g^{eff}(r)$ can be defined with only spatial dependence:

$$F_g(r) = \chi_g^{eff}(r) \sum_{iso} \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \Phi_{g'}(r) . \qquad (3.1.2)$$

The effective fission spectrum is defined to preserve the total fission source of a material region by weighting the fission spectrum with contributions from the individual isotopes:

$$\chi_g^{eff}(r) = \frac{\sum_{iso} \chi_{g.iso}(r) \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \Phi_{g'}(r)}{\sum_{iso} \sum_{g'=1}^{G} N_{iso} \nu\sigma_{f,iso,g'}(r) \Phi_{g'}(r)} . \qquad (3.1.3)$$

In MPACT, the routine *calcMacroChi* is called when the effective fission spectrum of a cross section mesh (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. Previously, in every outer iteration, when the fission source is updated (and thus fission spectrum needs to be updated), repeated calculations were performed to compute the isotopic fission cross sections in *calcMacroChi*, even though the values do not change. This was considered overly time consuming, especially in the resonance energy range due to the self-shielding calculation.

To avoid the repeated calculations, the fission cross section is stored in the cross section mesh when first computed in *calcMacroXS*. The fission cross sections are then used directly in *calcMacroChi* to save computing time.

The initial MPACT profiling results also showed that for a 2-D quarter-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 [4] interpolates the resonance integral (RI) at a specific background cross section from a RI table.

For the subgroup method, a fresh pin cell test showed that the *Segev* routine was called 15,232 times. More than 90% of the calls were used in the non-uniform fuel temperature treatment, and the remainder of the calls were used to shield the resonance scattering cross section. The number of resonance scattering calls is reasonable (proportional to the number of resonance groups, isotopes, and regions). The massive number of calls for non-uniform temperature treatment was due to inefficient loops of subgroup category and level, where *Segev* was called unnecessarily.

Segev interpolation was used to compute an approximate, effective cross section to determine the temperature adjustment ratio for the subgroup method. The background cross section is approximated as $\lambda\sigma_p$ because an accurate background cross section cannot be obtained before performing the subgroup calculation. There is no reason to use a sophisticated/expensive interpolation scheme given the argument is far away from the true value. Therefore, in addition to modifying inefficient loops encompassing *Segev*, the call was replaced by using the base effective absorption provided by the multigroup library. Physically, these cross sections should be much more accurate than the current *Segev* interpolated values, especially for LWR applications.

In addition to the major improvements to *calcMacroXS* and *Segev*, a few minor improvements were implemented in *calcMacroXS*, as follows:

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

Another speedup implemented is to decrease the number of subgroup calculations. Standard subgroup calculations require the fixed source problem (FSP) to be solved for every 2-D plane in each resonance category, energy group, and subgroup level. For most MPACT calculations using the ORNL 47-group library [5], there are 17 resonance groups and 4 subgroup levels per group. The number of categories can be changed by choosing a different subgroup set based on user input. A typical set of resonance categories used for most MPACT applications is shown in
Table 1.

Table 1. 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

To improve the performance of the subgroup calculation, efforts can be made to reduce the number of FSP. Figure 4 shows the continuous-energy cross section of Zr-91 and Zr-96, two resonance isotopes defined in the third category. According to the group structure of the 47-group library, most resonances of the two isotopes fall into the first two resonance groups (130.1-9118.8eV). Therefore, it is unnecessary to perform the FSP calculations for all resonance groups for the clad isotopes. To generalize the approach, a one-group subgroup scheme for non-important categories was developed.

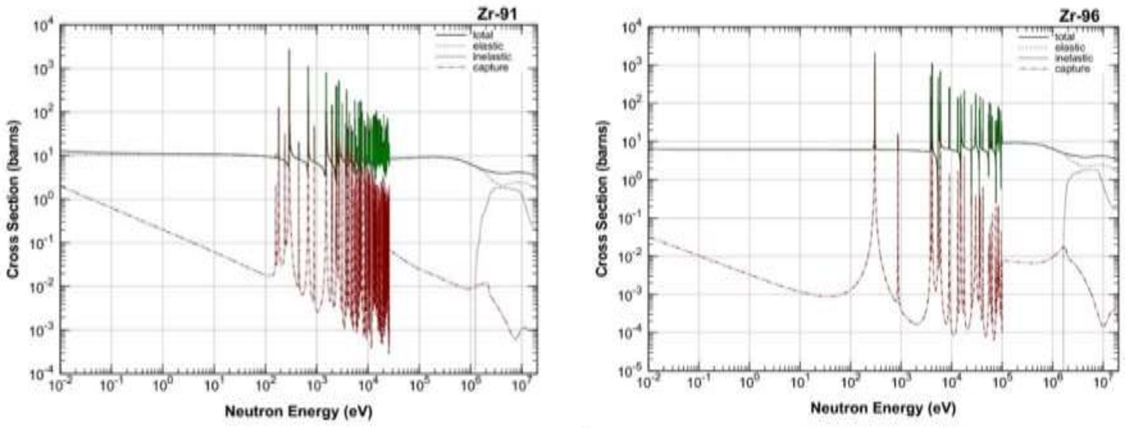


Figure 4. Continuous-energy cross section of Zr-91 and Zr-95 [6].

The differences between the multigroup subgroup and the one-group subgroup methods are described below [3]. The non-uniform temperature treatment is also included in these equations, making it consistent with the current MPACT code.

For standard **multigroup subgroup calculations**, the FSP is given as

$$\Omega\cdot\nabla\psi_{g,c,n}(r,\Omega)+\left[\Sigma_{a,g,c,n}(r)+\lambda_g\Sigma_p(r)\right]\psi_{g,c,n}(r,\Omega)=\frac{1}{4\pi}\lambda_g\Sigma_p(r)\Delta u_g\,. \qquad (3.1.4)$$

This equation should be solved for every energy group *g*, resonance category *c* and subgroup level *n*, where

$$\Sigma_{a,g,c,n}=\frac{\sum_{i\in c}N^i I^i_{a,g,\infty}(T_{loc})}{I^r_{a,g,\infty}(T_{loc})}\sigma^r_{a,g,n}(T_{loc}) \quad \text{and} \quad \lambda_g\Sigma_p=\sum_i\lambda^i_g\Sigma^i_p\,. \qquad (3.1.5)$$

$I^i_{a,g,\infty}(T_{loc})$ is the infinite absorption RI per lethargy for isotope *i* at local temperature T_{loc}. To adjust the subgroup level for temperature effect,

$$\sigma^r_{a,g,n}(T_{loc}) = \frac{\sigma^r_{a,g}(T_{loc})}{\sigma^r_{a,g}(T_{ave})}\bar{\sigma}^r_{a,g,n} = f(T_{loc})\bar{\sigma}^r_{a,g,n}, \quad (3.1.6)$$

where $\bar{\sigma}^r_{a,g,n}$ is the subgroup level of the representative isotope in category *c* in the library. The temperature adjustment ratio can be determined in various ways. In MPACT, $\sigma_{a,g}(T_{loc})$ and $\sigma_{a,g}(T_{ave})$ are obtained from the base cross section data. Once the solution of Equation (3.1.4) is available for all subgroup levels, a table of $\Sigma_{e,g,c,n}(\sigma^r_{a,g,n})$ can be obtained by the level-dependent flux:

$$\Sigma_{e,g,c,n} = \frac{\Sigma_{a,g,c,n}\phi_{g,c,n}}{\Delta u_g - \phi_{g,c,n}}. \quad (3.1.7)$$

The temperature effect is already embedded in $\sigma^r_{a,g,n}$, so the table entries are no longer the original $\bar{\sigma}^r_{a,g,n}$. Usually there are 4 subgroups for solving a fixed source problem and 7 subgroups for collapsing the effective cross section. The table $\Sigma_{e,g,c,n}(\sigma^r_{a,g,n})$ (4 levels) must be interpolated/extrapolated into $\Sigma_{e,g,c,m}(\sigma^r_{a,g,m})$ (7 levels). This table is not only for the representative isotope, but also for the other resonance isotopes in the same category *c*, so a conversion is needed from a resonance isotope to the representative isotope at about the subgroup level

$$\sigma^{i,\arg}_{a,g,m} = \frac{I^r_{a,g,\infty}(T_{loc})}{I^i_{a,g,\infty}(T_{loc})} f(T_{loc})\bar{\sigma}^i_{a,g,m}, \quad (3.1.8)$$

where $f(T_{loc})$ is defined the same as in Equation (3.1.6), but it is about isotope *i*. Using $\sigma^{i,\arg}_{a,g,m}$ as the argument to obtain $\Sigma_{e,g,c,m}(\sigma^{i,\arg}_{a,g,m})$, the background cross section can be computed:

$$\Sigma^i_{b,g,m} = \Sigma_{e,g,c,m}(\sigma^{i,\arg}_{a,g,m}) + \lambda_g \Sigma_p. \quad (3.1.9)$$

With all the subgroup parameters, the effective cross section is computed as

$$\sigma^i_{a,g} = \frac{\sum_m \bar{\sigma}^i_{a,g,m} \dfrac{\Sigma^i_{b,g,m}}{\Sigma^i_{a,g,m}(T_{loc}) + \Sigma_{x,g} + \Sigma^i_{b,g,m}} w^i_{w,g,m}(T_{loc})}{\sum_m \dfrac{\Sigma^i_{b,g,m}}{\Sigma^i_{a,g,m}(T_{loc}) + \Sigma_{x,g} + \Sigma^i_{b,g,m}} w^i_{w,g,m}(T_{loc})} \quad (3.1.10)$$

where $\Sigma^i_{a,g,m}(T_{loc}) = N^i f(T_{loc})\bar{\sigma}^i_{a,g,n}$ and $\Sigma_{x,g} = \sum_{j\neq i}\Sigma^j_{a,g}$. This equation means the temperature adjustment is only performed within the flux term by $\Sigma^i_{b,g,m}$ and $\Sigma^i_{a,g,m}(T_{loc})$. Also, $\Sigma_{x,g}$ term requires iterations to determine $\sigma^i_{a,g}$ (Bondarenko iteration).

In the **one-group subgroup** scheme, a procedure similar to that shown in Eqs. (3.1.4) to (3.1.10) is performed by integrating over the resonance groups. Two approximations have been made: (1) the IR source is approximated by averaging over the entire resonance energy range; (2) subgroup levels for FSP are no longer group dependent.

The fixed source problem for one-group subgroup calculation is given as

$$\Omega\cdot\nabla\psi_{c,n}(r,\Omega)+[\Sigma_{a,c,n}(r)+\lambda\Sigma_p(r)]\psi_{c,n}(r,\Omega)=\frac{1}{4\pi}\lambda\Sigma_p(r)\Delta u\,. \qquad (3.1.11)$$

This equation should be solved for every resonance category *c* and subgroup level *n*, where

$$\Sigma_{a,c,n}=\frac{\sum_{i\in c}\sum_g N^i I^i_{a,g,\infty}(T_{loc})\Delta u_g}{\sum_g I^r_{a,g,\infty}(T_{loc})\Delta u_g}\sigma^r_{a,n}(T_{loc}) \quad \text{and} \quad \lambda\Sigma_p=\frac{\sum_g\sum_i \lambda^i_g\Sigma^i_p\Delta u_g}{\sum_g \Delta u_g}\,. \qquad (3.1.12)$$

To adjust the subgroup level for temperature effect,

$$\sigma^r_{a,n}(T_{loc})=\frac{\sigma^r_a(T_{loc})}{\sigma^r_a(T_{ave})}\bar{\sigma}^r_{a,n}=f(T_{loc})\bar{\sigma}^r_{a,n}\,, \qquad (3.1.13)$$

where $\bar{\sigma}^r_{a,n}$ can be the subgroup level for any energy group (MPACT uses the first resonance group). $\sigma_a(T_{loc})$ and $\sigma_a(T_{ave})$ are obtained by averaging the base cross section over all resonance groups with flat flux. Once the solution of Equation (3.1.11) is available for all subgroup levels, we can obtain a table $\Sigma_{e,c,n}(\sigma^r_{a,n})$ by the level-dependent flux

$$\Sigma_{e,c,n}=\frac{\Sigma_{a,c,n}\phi_{c,n}}{\Delta u-\phi_{c,n}}\,. \qquad (3.1.14)$$

Note the temperature effect is already embedded in $\sigma^r_{a,n}$, so the table entries are no longer the original $\bar{\sigma}^r_{a,n}$. Usually there are 4 subgroups for solving fixed source problem but 7 subgroups for collapsing the effective cross section. The table $\Sigma_{e,c,n}(\sigma^r_{a,n})$ (n=1…4) must be interpolated/extrapolated into $\Sigma_{e,c,m}(\sigma^r_{a,m})$ (m=1…7). In fact, this table is not only for the representative isotope, but also for the other resonance isotopes in the same category *c*, so a conversion is needed from a resonance isotope to the representative isotope about the subgroup level

$$\sigma^{i,\arg}_{a,g,m}=\frac{I^r_{a,g,\infty}(T_{loc})}{I^i_{a,g,\infty}(T_{loc})}f(T_{loc})\bar{\sigma}^i_{a,g,n}\,, \qquad (3.1.15)$$

where $f(T_{loc})$ is defined the same as in Eq. (3.1.13) but is about isotope *i* at a specific group. Note that Eq. (3.1.15) retrieves the group index by assuming that the table $\Sigma_{e,c,m}(\sigma^r_{a,m})$ is applicable to all resonance groups. The differences of source term ($\lambda\Sigma_p$) among groups are neglected when estimating the dependence of $\Sigma_{e,c,m}$ on $\sigma^{r}_{a,m}$. Using $\sigma^{i,\arg}_{a,g,m}$ as the argument to obtain $\Sigma_{e,g,c,m}(\sigma^{i,\arg}_{a,g,m})$, the background cross section can be computed:

$$\Sigma^{i}_{b,g,m} = \Sigma_{e,g,c,m}(\sigma^{i,\arg}_{a,g,m}) + \lambda_g \Sigma_p \,. \tag{3.1.16}$$

With all the subgroup parameters, the effective cross section is computed as

$$\sigma^{i}_{a,g} = \frac{\sum_m \bar{\sigma}^{i}_{a,g,m} \dfrac{\Sigma^{i}_{b,g,m}}{\Sigma^{i}_{a,g,m}(T_{loc}) + \Sigma_{x,g} + \Sigma^{i}_{b,g,m}} w^{i}_{w,g,m}(T_{loc})}{\sum_m \dfrac{\Sigma^{i}_{b,g,m}}{\Sigma^{i}_{a,g,m}(T_{loc}) + \Sigma_{x,g} + \Sigma^{i}_{b,g,m}} w^{i}_{w,g,m}(T_{loc})} \,, \tag{3.1.17}$$

where $\Sigma^{i}_{a,g,m}(T_{loc}) = N^{i} f(T_{loc}) \bar{\sigma}^{i}_{a,g,n}$ and $\Sigma_{x,g} = \sum_{j \neq i} \Sigma^{j}_{a,g}$. This equation indicates that the temperature adjustment is only performed within the flux term by $\Sigma^{i}_{b,g,m}$ and $\Sigma^{i}_{a,g,m}(T_{loc})$. Also, $\Sigma_{x,g}$ term requires iterations to determine $\sigma^{i}_{a,g}$.

3.2 CTF Improvements

Several approaches were considered for improving CTF parallel performance [7]. These included using a hybrid shared/distributed memory parallelization using OpenMP and MPI, using MPI to parallelize large loops in the solution, and simply breaking the solution space into smaller solution domains. It was ultimately determined that the third option would be the fastest solution to implement and would have the highest probability of success in achieving good parallel efficiency. This was the case partly because almost no source code changes would be required in CTF itself; rather, only the domain decomposition algorithm in the CTF Preprocessor utility—which is a separate utility used to convert the more user-friendly VERA Common Input file to a CTF input file—would need to be changed. While using a hybrid approach is a potentially attractive solution, it would not be compatible with other coupled codes when CTF is used in VERA-CS due to different threading requirements by the individual code packages.

Rather than allowing the user to provide any arbitrary number of processors, it was decided that the number of processors should be an integer multiple of the number of assemblies in the model. Although this approach still lacks flexibility, it enforces a high parallel efficiency by ensuring that each assembly gets the same number of processors for its work. Furthermore, CASL applications for CTF target high-processor-count machines, and they use thousands of processors per simulation. Therefore, increasing the core count by a factor of two (i.e., 56 processors to 112) or four (224 processors) is not an issue.

For this initial work, a 4× refinement in the number of subdomains was added, which results in the preprocessor dividing each full assembly into four quarter assemblies. In a model that uses symmetry resulting in half- and quarter-size assemblies, a half-assembly is split into two domains, and a quarter assembly is modeled using one domain. This is an improvement over the old domain decomposition because each domain will be the same size in symmetry models. Before, the half- and quarter-assemblies sat idle and waited for the full assemblies to finish their calculations before data could be shared between assemblies.

An example of this domain-refinement approach is shown in Figure 5 for a model of five 17 × 17 assemblies in a cross configuration. The different colors in the figure denote different solution domains in the model. There are four domains per assembly, leading to 20 solution domains for the whole model.

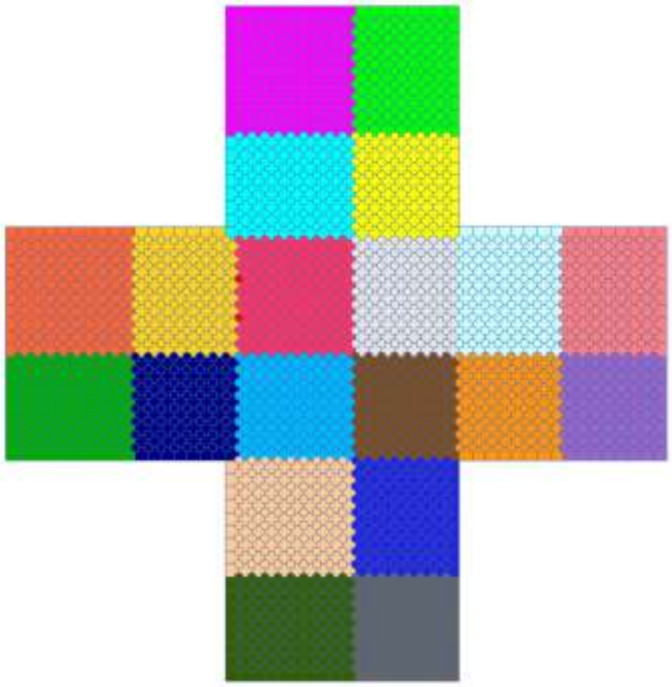


Figure 5. Solution domains (denoted by color) resulting from a 4× refinement in a model of five 17 × 17 assemblies.

A concern with refining the domain size was that the ratio of ghost cells to owned cells would grow very quickly as we make domain sizes smaller. This is because the original parallelization [8] required two layers of ghost channels around the outside of a domain due to a need for gap data that comes into the ghost channels from other ghost channels. Figure 6 shows how the original preprocessor domain decomposition worked for a model of two 3 × 3 assemblies. It is known that each ghost channel adds MPI overhead to the solution because it requires data to be updated periodically in the solution algorithm. The higher the ratio of ghost cells to owned cells, the higher the overhead and the poorer the scaling will be.

To address this concern, an investigation to find a way to eliminate the second level of ghost channels was undertaken. It was discovered that a few extra terms needed to be shared between domains to keep results consistent between serial and parallel runs. However, the biggest problem was that a defect was leading to incorrect lateral transfer of momentum for cases with void gradients. The second layer of ghost channels was needed to retain this dysfunctional behavior and thus keep serial and parallel results consistent. This defect was fixed as a part of this work, making it possible to eliminate the second layer of ghost channels. Additionally, it was also possible to start cycling out of several expensive loops in the code, further improving code performance. Overall, a roughly 6% speedup was realized for a full-core 193-assembly model (Problem 7). This was accomplished by eliminating the extra layer of ghost channels using the original domain decomposition scheme. It is anticipated that the speedup would be more significant for models with refined domains.

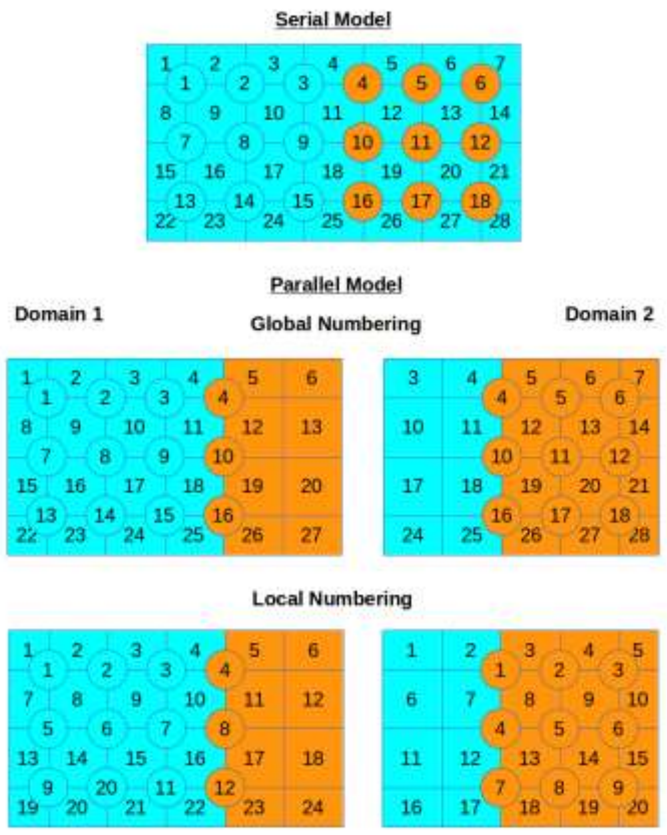


Figure 6. Example of channel ghosting scheme for original domain decomposition approach.

Finally, during performance testing, it was discovered that the pressure matrix solve, which involves solving a matrix whose size is equal to the number of fluid control volumes in the model (e.g., ~700,000 volumes for Problem 7), was the most computationally expensive portion of the CTF solution. It was also the most poorly scaling section of the code. To address this, different preconditioner options were tested, and it was discovered that using a successive over-relaxation (SOR) preconditioner, instead of the Jacobi preconditioner that was originally used, leads to the number of total PETSc iterations being reduced by half.

3.3 Method of Characteristics (MOC) Improvements

The MOC solver in MPACT is used for both the subgroup and the radial solve of the 2D/1D eigenvalue solution. While MOC was not a significant component of the baseline run-times, as the other components are accelerated, MOC becomes limiting. In addition, the performance improvements in MOC significantly contribute to the speedup improvements in the subgroup calculation.

The original MOC computation begins by looping over all of the energy groups. For each group, the source is set up and then loops over azimuthal angles and rays. For each ray, the ray segment data are constructed, and the ray is traced in both directions for every polar angle. Figure 7 shows the MOC algorithm before the updates [9].

```
for each group ($g$ from 1 to $N_{groups}$)
    Setup source for group $g$ (using updated flux solution from previous groups)
    for each inner iteration ($i$ from 1 to $N_{inners}$)
        for each azimuthal angle ($a$ from 1 to $N_{angles}$)
            for each longray in angle $a$ ($l$ from 1 to $N_{longrays}$(a))
                Connect modular rays and determine coarse mesh surfaces
                Calculate exponential values for each segment in longray $l$ and group $g$
                for each polar angle ($p$ from 1 to $N_{polar}$)
                    Grab incoming angular flux ($\varphi_{in,l,p,g}$) at both ends of longray $l$
                    for each segment ($s$ from 1 to $N_{segments}(l)$)
                        Evaluate forward direction:
                            Calculate outgoing angular flux ($\varphi_{out,l,p,g}$)
                            Tally contribution to fine mesh scalar flux in region $r1$ ($\phi_{g,r1}$)
                        Evaluate backward direction:
                            Calculate outgoing angular flux ($\varphi_{out,l,p,g}$)
                            Tally contribution to fine mesh scalar flux in region $r2$ ($\phi_{g,r2}$)
                    end for
                    Store outgoing angular flux ($\varphi_{out,l,p,g}$)
                    for each coarse mesh surface intersection ($c$ from 1 to $N_{cm}$)
                        Tally surface flux and currents for coarse mesh surface $c$, group $g$
                    end for
                end for
            end for
        end for
    end for
end for
```

Figure 7. Pseudocode for Gauss-Seidel sweeping with group on outermost loop.

The new algorithm shifts the loop over groups inside the loop over rays and is shown in Figure 8. The advantage of this algorithm is that the segment data for the ray only need to be assembled once per iteration instead of N_{groups} times. This new iteration scheme does come with a memory cost, and it may potentiall require more outer iterations. Because the energy group loop is in the inner most loop, the Gauss-Siedel iteration scheme for energy scattering cannot be used. Instead, all of the scattering source must be set up for all groups, and a Jacobi iteration must be used for the scattering iteration. This increases the memory to store the source of all groups, but it also has the potential to slow the convergence. It is observed that the use of CMFD to accelerate the multigroup MOC solution captures the group coupling effect, and the number of outer iterations is not significantly impacted.

```
Setup source for all groups
for each inner iteration ($i$ from 1 to $N_{inners}$)
    for each azimuthal angle ($a$ from 1 to $N_{angles}$)
        for each longray in angle $a$ ($l$ from 1 to $N_{longrays}$(a))
            Connect modular rays and determine coarse mesh surfaces
            Calculate exponential values for each segment in longray $l$ and all groups
            for each polar angle ($p$ from 1 to $N_{polar}$)
                Grab incoming angular flux ($\varphi_{in,l,p,g}$) at both ends of longray $l$
                for each segment ($s$ from 1 to $N_{segments}(l)$)
                    for each group (1 to $N_{groups}$)
                        Evaluate forward direction:
                            Calculate outgoing angular flux ($\varphi_{out,l,p,g}$)
                            Tally contribution to fine mesh scalar flux in region $r1$ ($\phi_{g,r1}$)
                        Evaluate backward direction:
                            Calculate outgoing angular flux ($\varphi_{out,l,p,g}$)
                            Tally contribution to fine mesh scalar flux in region $r2$ ($\phi_{g,r2}$)
                    end for
                end for
                Store outgoing angular flux ($\varphi_{out,l,p,g}$)
                for each coarse mesh surface intersection ($c$ from 1 to $N_{cm}$)
                    Tally surface flux and currents for coarse mesh surface $c$, group $g$
                end for
            end for
        end for
    end for
end for
```

Figure 8. Pseudocode for Jacobi sweeping with group on innermost loop.

3.4 Subgroup Sweep Improvements

The pre-existing subgroup calculation scheme in MPACT is shown in Figure 9, where there are three loops: (1) over the resonant groups, (2) over the subgroup categories for that group, and (3) over the subgroup levels. Inside these loops, there is an iteration loop where a transport sweep for each resonant group, category, and level are performed (line 7 of Figure 9) [10].

```
 1: for each resonant group ($g$ from $g_{res,beg}$ to $g_{res,end}$)
 2:     for each subgroup category ($c$ from 1 to $N_{cat}$(g))
 3:         for each subgroup level ($l$ from 1 to $N_{levels}$)
 4:             Setup $\Sigma_{t,g,c,l}$ based on $\Sigma_{a,g,c,l}$ and $\lambda_g \Sigma_p$
 5:             Setup source for this group/category/level based on $\lambda_g \Sigma_p$
 6:             for each iteration ($i$ from 1 to $N_{iters}$)
 7:                 Perform transport sweep for this group/category/level
 8:                 Compare residual based on scalar flux (terminate if below criteria)
 9:             end for
10:             Calculate equivalence cross section ($\Sigma_{eq,g,c,l}$)
11:         end for
12:     end for
13: end for
```

Figure 9. Pseudocode for pre-existing subgroup scheme with group on outermost loop.

Most MPACT calculations have 4 categories per group and 4 levels. However, the number of categories can be changed by choosing a different subgroup set based on the user's input. The number of levels depends on the library being used. All results covered in this work make use of the 47-group library developed by ORNL [5], which contains 17 resonant groups (from group 10 to group 26).

To take advantage of the new multigroup kernels described in the previous subsection, the scheme is restructured slightly. The first step is to define each unique combination of resonant group, category, and level as a single pseudogroup. The number of pseudogroups for the entire subgroup calculation will be the product of the number of resonant groups times the average number of subgroup categories per group times the number of subgroup levels. In theory, the number of categories can vary from group to group, though this does not seem to be the case for the 47-group library, which yields 272 pseudogroups. Based on this concept, a transport kernel could be constructed to sweep over all pseudogroups concurrently, but the source, cross section, scalar flux, and angular flux need to be stored for each up-front, whereas in the previous scheme, only one group of storage was necessary at a time. Figure 10 shows the pseudocode for the refactored scheme, taking advantage of the multigroup kernel concept:

```
1: for each pseudogroup (pg from 1 to $N_{pseudogroups}$)
2:     Setup and store $\Sigma_{t,pg}$ for this pseudogroup based on $\Sigma_{a,pg}$ and $\lambda_{pg}\Sigma_p$
3:     Setup and store source for this pseudogroup based on $\lambda_{pg}\Sigma_p$
4: end for
5: for each iteration ($i$ from 1 to $N_{iters}$)
6:     Perform transport sweep for all pseudogroups
7:     Compare residual based on scalar flux (terminate if below criteria)
8: end for
9: for each pseudogroup (pg from 1 to $N_{pseudogroups}$)
10:    Calculate equivalence cross section $\Sigma_{eq,pg}$
11: end for
```

Figure 10. Pseudocode for subgroup scheme using the multigroup transport kernel.

As one might expect, the amount of memory required to store source and flux data for 272 pseudogroups can be a concern. One way to keep the memory low while still allowing the scheme to make use of the multigroup kernels is to divide the pseudogroups into batches. Figure 11 shows the pseudocode for the batched approach, where each batch contains a starting and stopping pseudogroup index:

```
1: for each batch ($b$ from 1 to $N_{batch}$)
2:     for each pseudogroup (pg from $pg_{beg}(b)$ to $pg_{end}(b)$)
3:         Setup and store $\Sigma_{t,pg}$ for this pseudogroup based on $\Sigma_{a,pg}$ and $\lambda_{pg}\Sigma_p$
4:         Setup and store source for this pseudogroup based on $\lambda_{pg}\Sigma_p$
5:     end for
6:     for each iteration ($i$ from 1 to $N_{iters}$)
7:         Perform transport sweep for all pseudogroups
8:         Compare residual based on scalar flux (terminate if below criteria)
9:     end for
10:    for each pseudogroup (pg from $pg_{beg}(b)$ to $pg_{end}(b)$)
11:        Calculate equivalence cross section $\Sigma_{eq,pg}$
12:    end for
13: end for
```

Figure 11. Pseudocode for subgroup scheme using the multigroup transport kernel and batching.

Because the subgroup calculation solves a purely absorbing problem with a fixed source, the transport sweeps can be vastly simplified by condensing the angular flux transmission into one operation instead of sweeping across all segments along a ray. Because the MOC kernels in MPACT sweep over two angles travelling in opposite directions (forward/backward) at the same time, effectively two equations are needed (Eqs 3.4.1a and 3.4.1b):

$$\varphi_{pg}^{out,for} = \varphi_{pg}^{in,for} A_{pg} + B_{pg} \tag{3.4.1a}$$

$$\varphi_{pg}^{out,back} = \varphi_{pg}^{in,back} A_{pg} + C_{pg} \tag{3.4.1b}$$

To visualize this, consider a ray in a simple pin cell problem (Figure 12). On the left is the discretization showing 5 segments along the ray (blue) with the incoming and outgoing angular fluxes at the ends of the ray. On the right is the same problem but with all 5 segments condensed into one. This is only valid and effective because the source is not changing between iterations as is the case during the eigenvalue calculation sweeps. Thus, the $A/B/C$ lumped parameters can be used in a fast, intermediate kernel that only updates the outgoing angular flux.

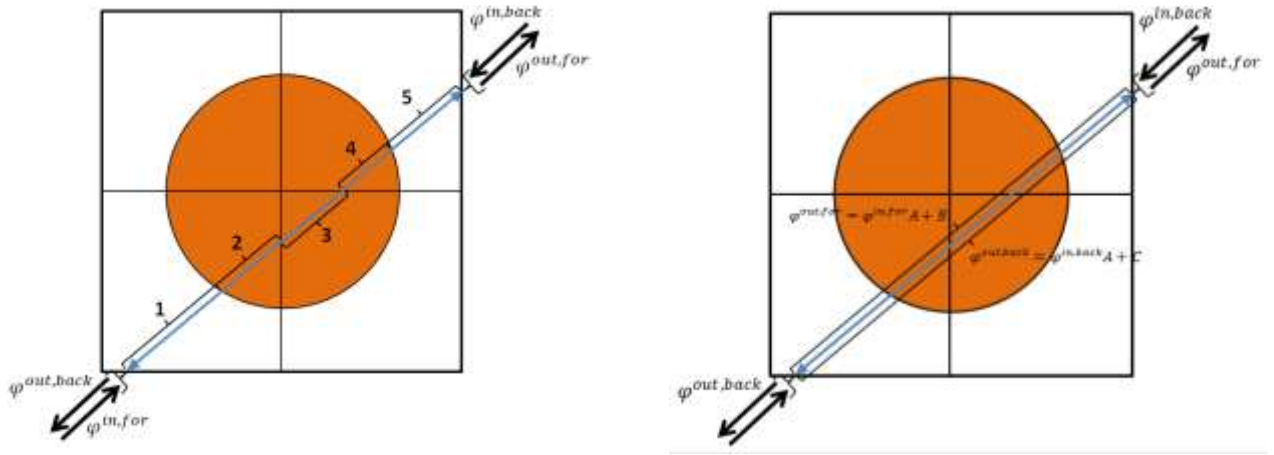


Figure 12. Visualization of MOC ray tracing (left) and lumped parameter (right) on a pin cell.

There are a couple of ways of deriving the equations for the lumped parameters. However, it is likely that A will be a product of the exponential terms for each segment (Eq. 3.4.2a). With A in hand, B and C can be easily calculated using the incoming and outgoing angular flux values (Eqs. 3.4.2b and 3.4.2c), assuming that a typical sweep is performed in calculating the factors.

$$A_{pg} = \exp\left(\sum_{i=1}^{N_{seg}} -\Sigma_{t,i,pg} l_i\right) \tag{3.4.2a}$$

$$B_{pg} = \varphi_{pg}^{out,for} - \varphi_{pg}^{in,for} A_{pg} \tag{3.4.2b}$$

$$C_{pg} = \varphi_{pg}^{out,back} - \varphi_{pg}^{in,back} A_{pg} \tag{3.4.2c}$$

The notation in (Eqs. 3.4.1 and 3.4.2) does not show all of the indexes to avoid clutter, but the lumped parameters must be calculated and saved for each angle and ray. However, given that these will only be three values over $O(100)$ segments, the storage for this is not concerning.

Figure 13 shows the pseudocode for lumped parameters, building off of the multigroup kernel with batching. The important changes here to note are (1) there is an initial sweep to calculate the lumped parameters, and (2) there are several fast sweeps that simply apply the factors to update the angular flux (per Eq. 3.4.1, and 3.4.3). A final standard sweep is completed to compute the scalar flux, which is required for the equivalence cross section calculation.

```
for each batch ($b$ from 1 to $N_{batch}$)
    for each pseudogroup (pg from $pg_{beg}(b)$ to $pg_{end}(b)$)
        Setup and store $\Sigma_{t,pg}$ for this pseudogroup based on $\Sigma_{a,pg}$ and $\lambda_{pg}\Sigma_p$
        Setup and store source for this pseudogroup based on $\lambda_{pg}\Sigma_p$
    end for
    Perform an initial sweep accumulating the $A_{pg}/B_{pg}/C_{pg}$ lumped parameters
    for each iteration ($i$ from 2 to $N_{iters}$)
        Perform a transport sweep applying $A_{pg}/B_{pg}/C_{pg}$ parameters for this batch
        Compare residual based on boundary angular fluxes (terminate if below criteria)
    end for
    Perform a final, normal sweep accumulating scalar flux ($\phi_{pg}$)
    for each pseudogroup ($pg$ from $pg_{beg}(b)$ to $pg_{end}(b)$)
        Calculate equivalence cross section $\Sigma_{eq,pg}$
    end for
end for
```

Figure 13. Pseudocode for subgroup scheme using the multigroup transport kernel, batching, and lumped parameter approach.

Since only the last iteration yields a scalar flux distribution, the convergence residual for this scheme is based on the angular flux updates instead of the scalar flux, which is used in the current scheme. Choosing the correct convergence criteria is important to ensure consistency between these two schemes. The current scheme imposes a maximum change of 10^{-6} for the scalar flux in any region for each pseudogroup. Since the new scheme will perform an additional sweep once the angular flux is considered to be converged, a similar maximum change is imposed on the angular flux, but with a criteria of 10^{-5}. In practice, this has been observed to be conservative, in most cases requiring one additional iteration. However, this is tolerable since it is only one additional "fast" iteration.

This approach would not be beneficial in problems with full vacuum radial boundary conditions in serial. In this scenario, only one iteration would be necessary since the boundary conditions do not need to be converged, as the zero incoming angular flux is correct. This, however, is not a likely scenario given that most problems are executed with quarter symmetry and in parallel. Additionally, it should be noted that the larger a spatial domain size, the greater benefit this new approach will yield. Small problems such as pin cell cases will have considerably less benefit than larger lattice cases.

Another improvement is that fewer azimuthal angles are necessary in the quadrature when performing the subgroup calculation. The self-shielding parameters in the 47-group library were generated using 8 azimuthal angles per octant. In theory, the answers may improve when using only 8 angles, compared to the 16 angles that are default, but it is also suspected that the subgroup results may be less sensitive than for the eigenvalue sweeps.

3.5 Parallel Partition Improvements

Another run-time improvement was to modify the way that MPACT performs its parallel decomposition. The 2D/1D method makes it natural to decompose the problem into axial planes, but further decomposition requires breaking each radial plane into partitions. The original partitioning scheme was fairly rigid in the constraints it imposed on the user to define a partition by requiring a partition line to extend across the full domain of the problem. This type of partitioning scheme made it difficult to obtain well-balanced partitions with a small number of cores per radial plane.

The extension of the radial partitioning scheme allows MPACT to decompose the planer problem along the boundaries of each quarter assembly. This creates a flexible partition which removes many of the constraints of the previous partition method. This allows the user to define a close to optimally balanced partition which provides good performance for all parts of the MPACT calculation.

The efficiency of the parallel partition is defined as the *parallel imbalance*, which is given by

$$\text{Imbalance} = \frac{N_{max}}{N_{min}} - 1,$$

where N_{max} is the maximum amount of nodes on a core, and N_{min} is the minimum amount of nodes on a core. If the all the cores have an equal number of nodes, the imbalance is 0.

Figure 14 and Figure 15 show two different 15-core radial partitions. Figure 14 shows the original partitioning scheme, which was restrictive in the partitions that were allowed. The original partition results in an imbalance of 52%. Figure 15 shows the new partition which removes many of the constraints of the original scheme and improves the imbalance to 5.5%.

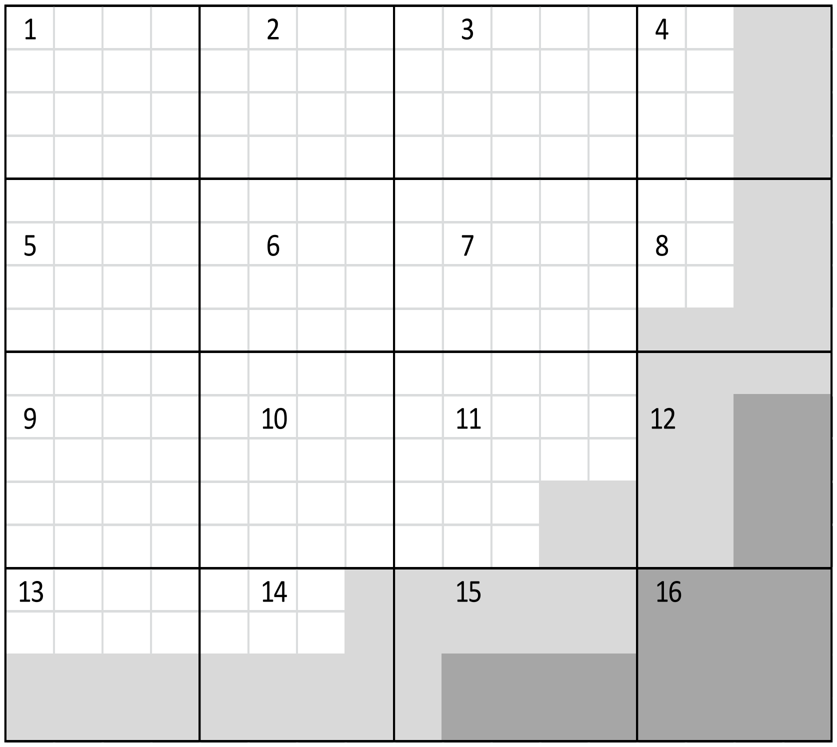


Figure 14. Original 15 core partition.

1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	4
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	4
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	4
1	1	1	1	2	2	2	2	3	3	3	3	4	4	5	5	5
7	7	7	1	2	6	6	6	6	6	6	3	5	5	5	5	5
7	7	7	7	7	7	7	6	6	6	6	6	5	5	5	5	5
7	7	7	7	7	7	7	6	6	6	6	6	6	5	5	5	5
8	8	8	8	9	9	9	9	10	10	10	10	11	11	11	11	11
8	8	8	8	9	9	9	9	10	10	10	10	11	11	11	11	11
8	8	8	8	9	9	9	9	10	10	10	10	11	11	11		
8	8	8	8	9	9	9	9	10	10	10	10	11	11	11		
14	14	14	8	9	13	13	13	10	12	12	12	12	12	11		
14	14	14	14	13	13	13	13	13	12	12	12	12	12	12		
14	14	14	14	14	13	13	13	13	12	12	12	12				
14	14	14	14	14	13	13	13	13	13	12	12	12				
15	15	15	15	15	15	15	15	15								
15	15	15	15	15	15	15	15	15								

Figure 15. Balanced partition with 15 cores.

3.6 Coarse Mesh Finite Difference (CMFD)Improvements

This work focuses is on the coarse mesh finite difference (CMFD) [11] solution in MPACT. CMFD serves multiple purposes in the 2D/1D solution methodology. First, it is a natural mechanism to tie together the radial MOC transport and the axial SP$_3$ solution. Because the CMFD system solves the multigroup 3D core in one system, it pulls together the global response of the system. In addition, the CMFD solution provides a framework to accelerate the convergence of the eigenvalue problem.

The CMFD methodology is based on the 0th moment of the neutron transport equation

$$\nabla\cdot J_g + \Sigma_{t,g}\phi_g = \sum_{g'}\Sigma_{s,g'\to g}\phi_{g'} + \frac{\chi_g}{k_{eff}}\sum_{g'}\nu\Sigma_{f,g'}\phi_{g'} \ . \qquad (3.6.1)$$

This equation is discretized onto a 3D Cartesian grid:

$$\sum_{\substack{f\in n,s,e,\\ w,t,b}} J_{g,f}\cdot A_f + \bar{\Sigma}_{t,g,i}\bar{\phi}_{g,i}V_i = \sum_{g'}\bar{\Sigma}_{s,g'\to g,i}\bar{\phi}_{g',i}V_i + \frac{\bar{\chi}_{g,i}}{k_{eff}}\sum_{g'}\nu\bar{\Sigma}_{f,g',i}\bar{\phi}_{g',i}V_i \ . \qquad (3.6.2)$$

In this equation, there have been no approximations made to the multigroup transport equation, but there are two unknowns: the current and the scalar flux. The CMFD methodology creates a relationship between the current and the flux as follows:

$$J_s = -\frac{2D^+D^-}{\Delta\left(D^+ + D^-\right)}\left(\bar{\phi}^+ - \bar{\phi}^-\right) + \hat{D}_s\left(\bar{\phi}^+ + \bar{\phi}^-\right) \ , \qquad (3.6.3)$$

where + and - denote the cells on each side of any given surface *s*. This equation defines the relationship between the current and the cell average flux using a coarse mesh diffusion approximation, but an additional term ($\hat{D}_s$) is added to correct the error in this approximation. This equation is still exact, but two unknowns still exist: the cell average flux and a nonlinear correction coefficient, $\hat{D}_s$. The solution marching scheme used will approximate this correction coefficient using the high order MOC solution in the radial direction and the SP$_3$ solution in the axial direction for the current iterate;

$$\hat{D}_s = \frac{J_s^{MOC} + \frac{2D^+D^-}{\Delta\left(D^+ + D^-\right)}\left(\bar{\phi}^+ - \bar{\phi}^-\right)}{\left(\bar{\phi}^+ + \bar{\phi}^-\right)} \quad . \tag{3.6.4}$$

The iteration scheme in MPACT first assumes $\hat{D}_s$ is zero and solves the full core CMFD equations. Once the coarse mesh fluxes are obtained, they are projected onto the fine MOC mesh, and a single MOC sweep is performed. The MOC equations solves for the current on the boundaries of the coarse mesh, which are then used to estimate a new value for $\hat{D}_s$. This process is repeated until both the coarse and fine mesh solutions are converged.

Another key purpose of CMFD is to accelerate the solution to the eigenvalue problem. To consider this, it is easier to adopt a matrix notation of the CMFD system. There are four major components to the matrix notation: the diffusion operator $\mathbf{D}$, the collision operator $\mathbf{T}$, the scattering operator $\mathbf{S}$, and the fission operator $\mathbf{F}$. These four terms are combined into the generalized CMFD eigenvalue problem:

$$\left(\mathbf{D} + \mathbf{T} - \mathbf{S}\right)\phi = \frac{1}{k_{eff}}\mathbf{F}\phi \tag{3.6.5}$$

and is simplified to

$$\mathbf{M}\phi = \frac{1}{k_{eff}}\mathbf{F}\phi \quad . \tag{3.6.6}$$

The main focus of this work will be accelerating the convergence of this eigenvalue problem by looking at advanced solution schemes to solve for the dominant eigenvalue of this linear system.

The current implementations of eigenvalue solvers in MPACT have been fixed-point methods (i.e., the next estimate of the solution depends only on the estimate immediately preceding it). An alternative to fixed-point iterations is subspace eigenvalue solvers in which information from several previous vectors is used to generate the next approximate solution. To extract the solution, an estimate of the eigenvalue is obtained as a linear combination of the subspace basis vectors through a Rayleigh-Ritz procedure which solves the projected eigenvalue problem:

$$\mathbf{V}^T\mathbf{M}\mathbf{V}y = \lambda\mathbf{V}^T\mathbf{F}\mathbf{V}y \tag{3.6.7}$$

where $\mathbf{V}$ contains a set of (typically orthogonal) basis vectors for the current subspace. For an appropriate selection of the subspace, the eigenvalues of the projection problem will closely approximate the original system, and the vectors of $\mathbf{V}y$ will approximate the eigenvector. The approximate eigenvalues and eigenvectors obtained from the Rayleigh-Ritz procedure are generally referred to as Ritz values and Ritz vectors respectively.

The determination of the new vectors to place into the subspace is generally what distinguishes the majority of subspace eigenvalue solvers. In the Arnoldi method [12], the subspace is taken to the Krylov subspace corresponding to the operator $\mathbf{M}^{-1}\mathbf{F}$, which is similar to power iteration. Unlike power iteration, the convergence of Arnoldi's method is not dictated by the dominance ratio, so the number of iterations required to converge can be significantly smaller.

Another subspace eigenvalue solver that is the main focus for implementation of this work is the Davidson method [13]. The idea behind the Davidson method is that at iteration n, one should seek a correction $t^{(k)}$, such that the eigenvalue equation given by

$$\mathbf{M}\left(\phi^{(n)}+t^{(n)}\right)=\lambda^{(n)}\mathbf{F}\left(\phi^{(n)}+t^{(n)}\right) . \tag{3.6.8}$$

This equation can be rearranged to

$$\left(\mathbf{M}-\lambda^{(n)}\mathbf{F}\right)t^{(n)}=\left(\mathbf{M}-\lambda^{(n)}\mathbf{F}\right)\phi^{(n)}\equiv -r^{(n)}, \tag{3.6.9}$$

where $r^{(n)}$ is the residual of the eigenvalue problem. The evaluation of $t^{(n)}$ would require a linear system solution of a nearly singular system. To avoid this computational expense, $\left(\mathbf{M}-\lambda^{(n)}\mathbf{F}\right)$ is approximated using a preconditioner $\mathbf{P}$, which leads to the Davidson correction equation

$$\mathbf{P}t^{(n)}=-r^{(n)}. \tag{3.6.10}$$

The Davidson method is extremely attractive because unlike all of the previous methods discussed, a linear system solve is not required. Instead, only a preconditioner application to an approximate matrix is needed. The choice of preconditioner is important for Davidson to quickly converge. To avoid singularities in the preconditioner system, the preconditioner is chosen based on the migration operator $\mathbf{M}$ rather than the shifted operator $\left(\mathbf{M}-\lambda^{(n)}\mathbf{F}\right)$. Several different preconditioners based on $\mathbf{M}$ are explored in this work, but it was determined that a multigrid preconditioner performed the best across a wide range of problems.

4. PERFORMANCE IMPROVEMENTS

4.1 Final Performance Results

The same two problems used to profile VERA-CS in Section 2 are used to determine the impact of the performance improvements discussed in Section 3. First the HFP quarter core is performed for all six of the performance improvements discussed. Figure 16 shows the total run time (blue), as well as each component for the baseline case and each improvement.

The x-axis shows the six categories of improvements made to VERA-CS in chronological order. For each improvement, the run-time is plotted by component. The run-time changes for each improvement can be seen from left to right.

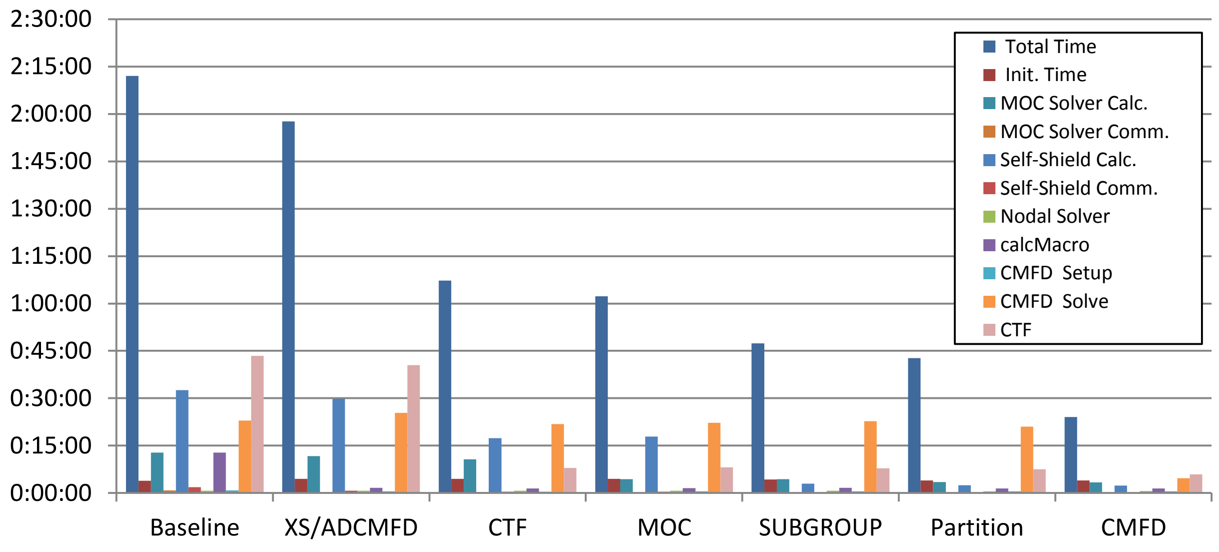


Figure 16. Progression of speedup for HFP case.

The final run-time for the HFP case is 24 minutes and 5 seconds on 870 cores. This is a speedup of 5.5× between the baseline case and the case with all of the runtime improvements.

Similar results are shown for the cycle depletion case with 5 depletion points. Figure 17 shows the total run-time and the run-time by component for each improvement category.

As seen with the previous case, a significant reduction in run-time is obtained with all of the improvements culminating in a 5× speedup overall. The final distribution of the major components of VERA-CS is shown in Figure 18.

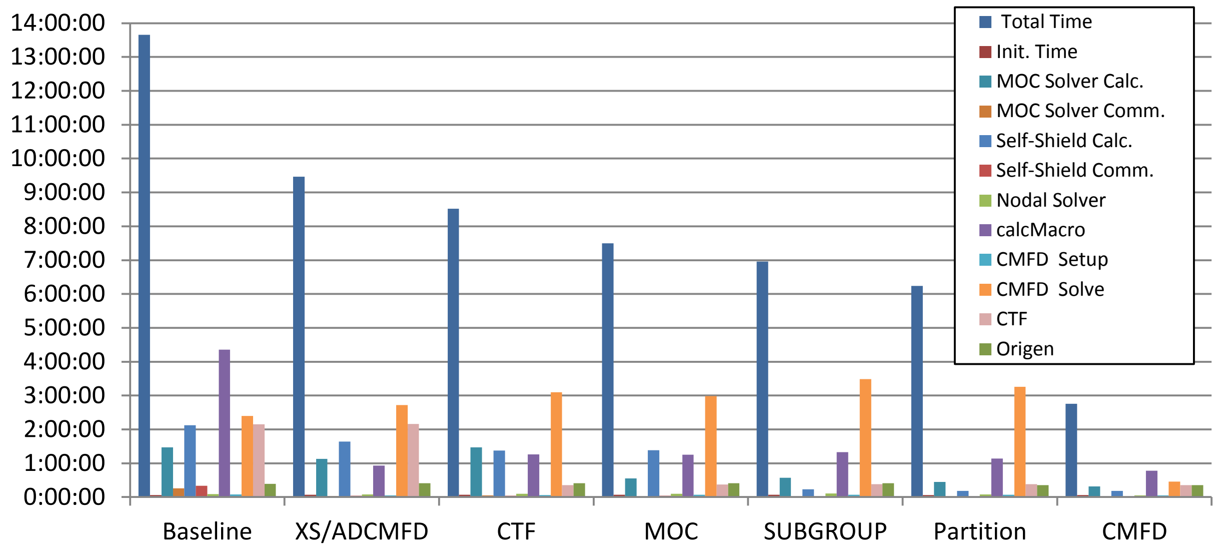


Figure 17. Progression of speedup for cycle depletion case.

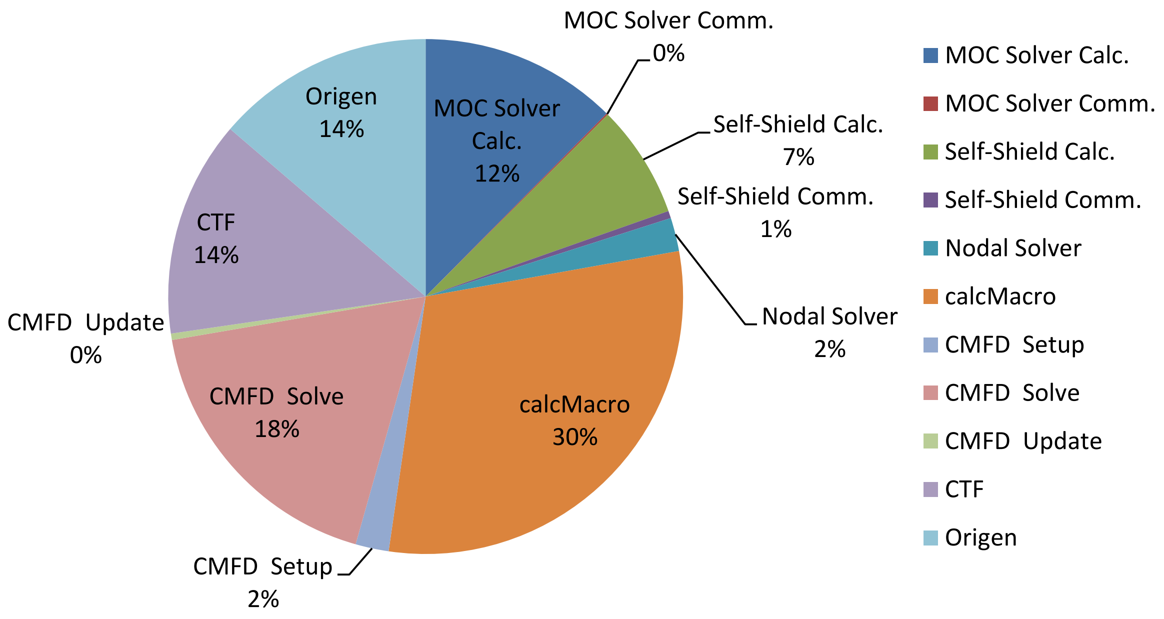


Figure 18. Final fraction of run-time for cycle depletion.

The final run-time obtained for 5 depletion steps on 870 cores is 2 hours and 45 minutes. We can estimate the run-time for an actual cycle depletion on 1000 cores by assuming that a cycle depletion will have 25 depletion steps instead of 5, and multiplying by 870/1000 to account for the different core counts. If we do this, the run-time for a cycle depletion on 1000 cores will be approximately 12 hours. This satisfies our goal of running a cycle depletion with 1000 cores “overnight”.

4.2 Future Work

A significant amount of time is still spent in *calcMacro*, even though the total run-time of this routine was decreased by over 3.5 hours. CMFD, CTF, ORIGEN, and the MOC time now share roughly similar shares of the time.

It is important to note that CTF is only using about one quarter of the total available processors in this simulation (193 out of 870). Moving to higher processor counts in CTF has not yet been attempted due to the high computational cost and poor scaling observed for the pressure-matrix solve portion of the solution. Future work will include investigating better preconditioning strategies for the pressure matrix in CTF so that further domain refinement can be implemented and CTF can use closer to the full number of processors allotted to the VERA-CS simulation.

5. CONCLUSIONS

This report documents a significant amount of work performed by the VERA-CS development team to accelerate individual components of VERA-CS for the target applications for CASL. Several improvements were made, including optimizing the cross section processing, increasing the parallelism of CTF, improving the efficiency of the MOC sweeper, developing a new algorithm for subgroup iterations, and implementing a new CMFD solution methodology. Overall, the improvements in run-time observed are 5× for a cycle depletion. This improvement exceeds the goal for VERA-CS to be able to perform a cycle depletion overnight on 1,000 cores.

Although significant advances have been made to VERA-CS, there are still improvements that can be made. Additional investigation is being made to improve the cross section calculation, which still takes a quarter of the total run-time.

6. REFERENCES

[1] A. Godfrey, et al., *VERA Benchmarking Results for Watts Bar Nuclear Plant*, CASL Technical Report: CASL-U-2015-0206-000, June (2016). http://www.casl.gov/docs/CASL-U-2015-0206-000.pdf

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

[3] Y. Liu, *Runtime Improvements to the Cross Section Calculation in MPACT*, CASL Technical Report CASL-X-2016-1105-000, May (2016).

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

[5] 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).

[6] K. Shibata et al., “JENDL-4.0: A new library for nuclear science and engineering,” *J. Nucl. Sci. Technol.*, 48, 1 (2011).

[7] R. Salko, S. Palmtag, B. Collins, “CTF Parallel Performance Improvements,” Submitted to *Transactions of the American Nuclear Society,* November (2016).

[8] R. Salko, R. Schmidt, and M. Avramova, “Optimization and Parallelization of the Thermal-Hydraulics Sub-channel Code CTF for High-Fidelity Multi-Physics Applications,” *Annals of Nuclear Energy*, **84**, 122–130, Oct (2014).

[9] S. Stimpson, B. Collins, and B. Kochunas, *MOC Efficiency Improvements Using a Jacobi Inscatter Approximation*, CASL Technical Report: CASL-U-2016-1056-001, March (2016).

[10] S. Stimpson, Y. Liu, B. Collins, K. Clarno, *MPACT Subgroup Self Shielding Efficiency Improvements*, CASL Technical Report: CASL-U-2016-1063-000, April (2016).

[11] K. Smith. “Nodal Method Storage Reduction by Nonlinear Iteration,” *Transactions of the American Nuclear Society*, **44**, 265 (1983).

[12] W. Arnoldi, “The Principle of Minimized Iterations in the Solution of Matrix Eigenvalue Problems,” *Quarterly of Applied Mathematics*, **9**, 17–29 (1951).

[13] R. Morgan, “Davidson’s Method and Preconditioning for Generalized Eigenvalue Problems,” *Journal of Computational Physics*, **89**, 241–245 (1990).