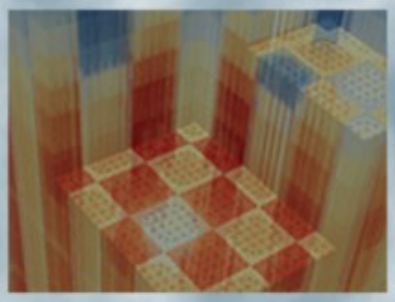
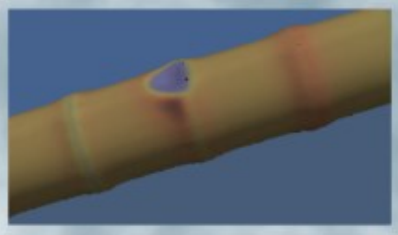
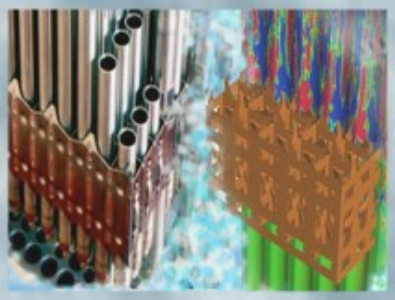
CASL-U-2016-1056-002







MOC Efficiency Improvements Using a Jacobi Inscatter Approximation

Shane Stimpson, Oak Ridge National Laboratory
Benjamin Collins, Oak Ridge National Laboratory
Brendan Kochunas, University of Michigan

August 31, 2016

REVISION LOG

Revision	Date	Affected Pages	Revision Description
0	03/23/2016	All	Original Report
1	03/30/2016	5	Updated Memory Profiling Table
2	08/31/2016	All	Cleared PTS

Document pages that are:

Export Controlled NO

IP/Proprietary/NDA Controlled NO

Sensitive Controlled NO

Approved for Public Release YES

ACRONYMS

GS	Gauss-Seidel
J	Jacobi
MOC	Method of Characteristics
ORNL	Oak Ridge National Laboratory
TCP_0	Transport-Corrected P_0
VERA	Virtual Environment for Reactor Applications

DESCRIPTION

In recent weeks, attention has been given to resolving the convergence issues encountered with TCP_0 by trying a Jacobi (J) inscatter approach when group sweeping, where the inscatter source is constructed using the previous iteration flux. This is in contrast to a Gauss-Seidel (GS) approach, which has been the default to-date, where the scattering source uses the most up-to-date flux values. The former is consistent with CASMO, which has no issues with TCP_0 convergence [1]. Testing this out on a variety of problems has demonstrated that the Jacobi approach does indeed provide substantially more stability, though can take more outer iterations to converge. While this is not surprising, there are improvements that can be made to the MOC sweeper to capitalize on the Jacobi approximation and provide substantial speedup. For example, the loop over groups, which has traditionally been the outermost loop in MPACT, can be moved to the interior, avoiding duplicate modular ray trace and coarse ray trace setup (mapping coarse mesh surface indexes), which needs to be performed repeatedly when group is outermost. Figure 1 shows the pseudocode for Gauss-Seidel sweeping with group being the outermost loop (as implemented in MPACT):

```
1: for each group (g from 1 to N_groups)
2:     Setup source for group g (using updated flux solution from previous groups)
3:     for each inner iteration (i from 1 to N_inners)
4:         for each azimuthal angle (a from 1 to N_angles)
5:             for each longray in angle a (l from 1 to N_longrays(a))
6:                 Connect modular rays and determine coarse mesh surfaces
7:                 Calculate exponential values for each segment in longray l and group g
8:                 for each polar angle (p from 1 to N_polar)
9:                     Grab incoming angular flux (φ_in,l,p,g) at both ends of longray l
10:                    for each segment (s from 1 to N_segments(l))
11:                        Evaluate forward direction:
12:                            Calculate outgoing angular flux (φ_out,l,p,g)
13:                            Tally contribution to fine mesh scalar flux in region r1 (φ_g,r1)
14:                        Evaluate backward direction:
15:                            Calculate outgoing angular flux (φ_out,l,p,g)
16:                            Tally contribution to fine mesh scalar flux in region r2 (φ_g,r2)
17:                    end for
18:                    Store outgoing angular flux (φ_out,l,p,g)
19:                    for each coarse mesh surface intersection (c from 1 to N_cm)
20:                        Tally surface flux and currents for coarse mesh surface c, group g
21:                    end for
22:                end for
23:            end for
24:        end for
25:    end for
26: end for
```

Figure 1. Pseudocode for Gauss-Seidel Sweeping with Group on Outermost Loop

As can be seen, the work to setup the modular rays and determine the coarse mesh surfaces (all of which are group-independent) is duplicated for each group (line 3). This layout is necessary for the Gauss-Seidel approach since each group must be completely solved so the inscatter source for the next group can be updated. A Jacobi inscatter approach can also use group as the outermost loop, but in this memo, Gauss-Seidel will be the only one with group as outermost.

Figure 2 similarly shows the Jacobi approach with the group on the innermost:

```
1: Setup source for all groups
2: for each inner iteration (i from 1 to N_inners)
3:     for each azimuthal angle (a from 1 to N_angles)
4:         for each longray in angle a (l from 1 to N_longrays(a))
5:             Connect modular rays and determine coarse mesh surfaces
6:             Calculate exponential values for each segment in longray l and all groups
7:             for each polar angle (p from 1 to N_polar)
8:                 Grab incoming angular flux (φ_in,l,p,g) at both ends of longray l
9:                 for each segment (s from 1 to N_segments(l))
10:                    for each group (1 to N_groups)
11:                        Evaluate forward direction:
12:                            Calculate outgoing angular flux (φ_out,l,p,g)
13:                            Tally contribution to fine mesh scalar flux in region r1 (φ_g,r1)
14:                        Evaluate backward direction:
15:                            Calculate outgoing angular flux (φ_out,l,p,g)
16:                            Tally contribution to fine mesh scalar flux in region r2 (φ_g,r2)
17:                    end for
18:                end for
19:                Store outgoing angular flux (φ_out,l,p,g)
20:                for each coarse mesh surface intersection (c from 1 to N_cm)
21:                    Tally surface flux and currents for coarse mesh surface c, group g
22:                end for
23:            end for
24:        end for
25:    end for
26: end for
```

Figure 2. Pseudocode for Jacobi Sweeping with Group on Innermost Loop

Now, the modular ray connections and coarse mesh surfaces are only calculated once per kernel call and avoids the duplication over groups. However, this restricts the ability to update inscatter sources since all groups are being solved at once, and when the sources are setup, they are all using the multigroup flux solution from the previous iteration.

It is also important to note that the variables for storing the angular fluxes were refactored to provide better access efficiency by moving the group index to the innermost index. This subsequently required an extension of the boundary parallel communication routines to pass multiple groups of data at once, instead of only one group at a time, leading to larger buffers and likely less communication overhead.

The performance of these two approaches has been tested on three cases, all from the VERA progression problem suite [2]: 1) a single quarter assembly lattice [VERA Problem 2a, Figure 3] and 2) a 2D slice of the 3x3 assembly cluster [VERA Problem 4a-2D, Figure 4], and 3) a 2D quarter core model [VERA Problem 5a-2D core layout in Figure 5]. It is worth noting that the 5a-2D case also includes baffle and reflector regions that are not included in the figure. All cases used a 0.05 cm ray spacing, 16 azimuthal angles per octant, and 2 polar angles in a Tabuchi-Yamamoto [3] 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 [4].

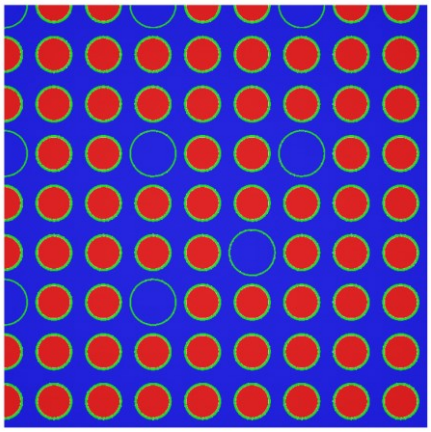


Figure 3. Geometry Visualization of VERA Problem 2a [2]

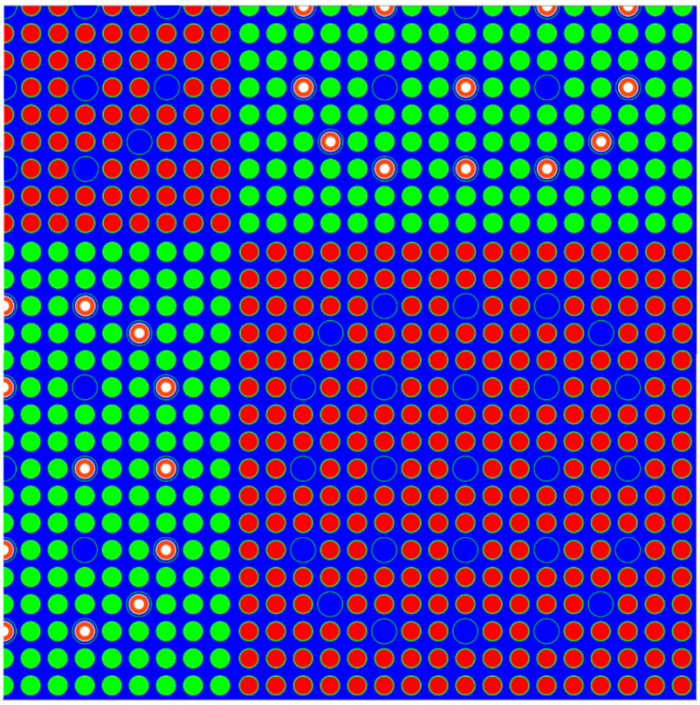


Figure 4. Geometry Visualization of VERA Problem 4a-2D [2]

In Figures 3 and 4, the red pins denote 2.1% enriched fuel pins and green denotes 2.6%. It can also be seen that there are several pyrex rods (red annular regions with white center) in the 2.6% enriched assemblies.

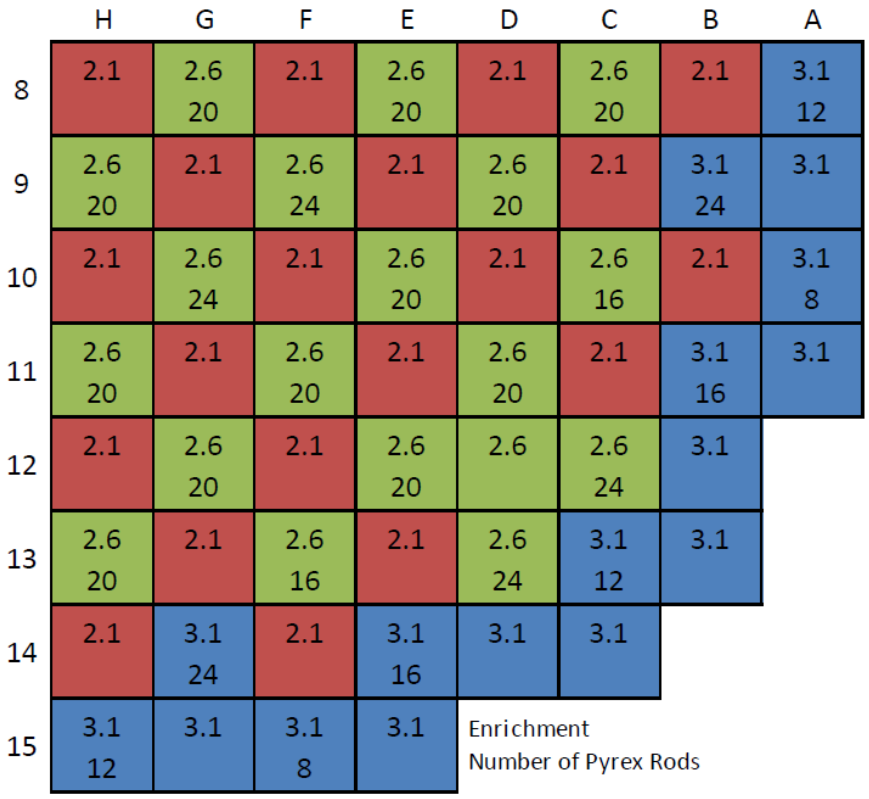


Figure 5. Geometry Visualization of VERA Problem 5a-2D [2]

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® processors (Xeon® CPU E5-1650v3, 3.2 GHz). Below are the results for Problem 2a (Table 1), showing the timing (in nanoseconds 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 longrays), 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 1. Timing Results for Problem 2a

	AMD (2.3 GHz)			Intel (3.2 GHz)		
	ns/integration			ns/integration		
	GS	J	Ratio	GS	J	Ratio
Kernel w/o Currents:	7.39	5.38	1.37	3.52	2.92	1.21
Kernel w/ Currents:	15.65	5.78	2.71	6.58	3.25	2.03

Similar results are observed on Problem 4a-2D, which is not very surprising (Table 2). 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 2. Timing Results for Problem 4a-2D

	AMD (2.3 GHz)			Intel (3.2 GHz)		
	ns/integration			ns/integration		
	GS	J	Ratio	GS	J	Ratio
Kernel w/o Currents:	7.60	5.33	1.42	3.95	3.36	1.18
Kernel w/ Currents:	14.80	5.90	2.51	7.02	3.71	1.89

5a-2D was run on the Titan supercomputer [5], on which each 16-core node contains 2.2 GHz AMD Opteron™ 6274 (Interlagos) processors, using 73 and 257 spatial parallelization processors. Table 3 shows the sweeping results (excluding communication) in terms of core-min/iteration. Since the goal with the Jacobi sweeper is to use only 1 inner iteration, the results here are basically a comparison of the sweep times for the kernel with currents. Additionally, because the GS sweeper 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 J iteration strategy generally requires a few more iterations to achieve the same level of convergence as the GS solver. Using Problems 2a and 4a as a guide, we can roughly estimate that 5a-2D has roughly 187 million segments (35 billion integrations).

Table 3. Timing Results for Problem 5a-2D

	73 Spatial Domains			257 Spatial Domains		
	GS	J	Ratio	GS	J	Ratio
MOC Solver Time (core-min/iteration):	14.61	5.94	2.45	11.87	5.02	2.36

However, these notable speedups do incur some cost in terms of the memory burden. Table 4 shows the total memory required for each of the problems presented (in gigabytes, GB). With the GS sweeper, 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 J sweeper, 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 J sweeper requires between 15-20% more overall storage than the GS sweeper. 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 transport solvers used to performance the self-shielding and eigenvalue calculation are completely separate, so two sweepers are initialized.

Table 4. Total Memory Requirements (GB) for Test Problems

	GS	J	Ratio
Problem 2a	0.20	0.23	1.15
Problem 4a-2D	0.88	1.02	1.15
Problem 5a-2D (73 Spatial)	23.74	28.40	1.20
Problem 5a-2D (257 Spatial)	37.42	44.67	1.19

Additional work is investigating how to fully incorporate these new kernels into the self-shielding calculation, which presently is still sweeping over one group at a time.

REFERENCES

[1] Kord Smith, personal communication (2015).

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

[3] 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*, Vol. 4, No. 2, p. 129-136 (2007).

[4] 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).

[5] Oak Ridge Leadership Computing Facility. "Introducing Titan - The World's #1 Open Science Supercomputer" (2014), http://www.olcf.ornl.gov/titan/.