LA-UR-17-27811
Approved for public release; distribution is unlimited.

Title: An Unstructured Mesh Performance Assessment in the MCNP® Code Version 6.2

Author(s): Roger L. Martz

Intended For: General Reference / MCNP® Website

Issued: August 2017

This page intentionally left blank.

An Unstructured Mesh Performance Assessment in the MCNP® Code Version 6.2

Roger L. Martz
Monte Carlo Methods, Codes, and Applications
X-Computational Physics Division
Los Alamos National Laboratory

August 2017

1 Introduction

The unstructured mesh (UM) capability in the Los Alamos National Laboratory (LANL) Monte Carlo N-Particle® (MCNP®) transport code has been under development since the mid-2000's [1]. This capability has been in every release of MCNP6 to the Radiation Safety Information Computational Center (RSICC) at Oak Ridge National Laboratory (ORNL) since the first beta release. Each MCNP release has provided improved UM features through scheduled code development, bug fixes, and integration with other code capabilities.

For the 6.2.0 version of MCNP6, the top-level UM tracking routine (`regl_track_um`) that is used for all particle types has been refactored primarily so that the code is easier to maintain in the future and with a secondary goal of trying to improve performance. The old routine that it replaced relied on too many special cases and was difficult to understand; this was in the 6.1.1 version of the code. The refactored routine may be faster for dense meshes where the particle mean free path (mfp) is very large.

The purpose of this report is to document the UM calculational performance like what was done in Reference 2 for the series of Godiva meshes.

2 Background

Some of the Reference 2 work compared the calculation performance of the UM to that of equivalent problems using MCNP's legacy constructive solid geometry (CSG) capability. This previous work was done with the 6.1.0 version of MCNP6. Calculations for the Godiva series of meshes from Reference 2 were repeated on some newer computing hardware using the 6.1.0, 6.1.1, and 6.2.0 threaded versions of MCNP6 as released to RSICC. Details follow.

2.1 Modeling Details

With the MCNP CSG capability, it is possible to model polyhedrons bounded by planes with arbitrary orientations. Because first-order tetrahedral finite elements are guaranteed to have planar faces, one can exactly reproduce these finite elements in MCNP with cells defined by arbitrary planes [3]. That approach was chosen for the comparisons in this work. From the same Abaqus input file used in the MCNP6 calculations with the UM, an equivalent representation of the first-order tetrahedra using arbitrary polyhedral cells (APCs) with the CSG capability was created. To make it convenient to describe what is known as "the outside world" in MCNP (i.e., the phase-space outside of the geometry of interest) each benchmark geometry was placed in a box of air so that one macrobody surface, defining the box's surface, represented the boundary between the geometry of interest and the outside world. The space between the benchmark geometry and the box's surface was "meshed" with additional finite elements / cells; these elements contained air. All of the geometry inside this macrobody was described with either a mesh of finite elements or with APCs. These boxes were loosely fitted around the Godiva spheres because space was needed to generate well-formed elements adjacent to the outer spherical surfaces. Therefore, the Abaqus model consisted of one part with two sections — one with enriched uranium and one with air.

In the course of this work, the number of finite elements and APCs was varied to look at code performance as a function of elements and cells. Central to meshing with the Abaqus tool is the seed size that controls the number of elements in a mesh. The Abaqus meshing algorithm attempts to make each edge for each finite element no bigger than this value. Hence, the seed size controls the number of elements that are in the mesh. Units for the seed are the same as the units used in the Abaqus solid model on which the mesh is generated. Values used in this work are provided in Table 1.

Table 1: Mesh Parameters

Seed Size (cm)	Number of Elements
4.0	1375
3.0	3142
2.0	9722
1.7	13305
1.5	16349
1.1	39250
1.0	51336
0.95	59160
0.80	93593
0.65	161971
0.50	320069
0.45	401723

Using APC's to recreate a "tetrahedral mesh representation" is an approach that users may rarely implement for real world calculations. However, it was necessary for this work in order to have an apples to apples comparison of code performance for these two geometry types.

2.2 Problem Description

The Godiva benchmark [4] is a simple highly enriched uranium sphere of radius 8.7407 cm and density 18.74 g/cm^3. Traditionally, neutrons are tracked in batches using a power iteration method to calculate k-effective for this fissile system (called a kcode calculation in MCNP6).

As with the Reference 2 work, meshes generated for the Godiva geometry were run with three different physical arrangements:

1. A neutron-only, kcode calculation with 5,000 histories per batch and 2000 cycles (to start approximately 10 million particles).
2. A voided geometry arrangement with an isotropic, fixed-point source of gammas near the origin (to avoid starting on a finite element node at the origin).
3. A gamma problem with the same source (monoenergetic gamma of 6.1 MeV) as with the void problem and the same material as the kcode problem.

These arrangements enabled the various tracking routines (UM and CSG) to be exercised under a variety of physics options.

Fifteen independent calculations (using 1 thread per run) were completed for each arrangement (of the mesh parameters in Table 1) and the results averaged. Each fixed-source calculation was run for exactly 10 million histories. Each kcode calculation was run for 2000 cycles with 5000 histories per batch. Calculations were performed on single nodes of two different high performance compute clusters; the clusters are described below.

2.3 Test Systems

The original test system for the Reference 2 work was not available for the current work. Two different Linux high performance compute clusters were used to run test problems for this report. They are described next.

2.3.1 System 1 — Moonlight

This cluster has 308 nodes where each node has two 8-core Intel Xeon Sandy Bridge [ES-2670] sockets for a total of 16 CPUs per node. The CPUs run at 2.6 GHz. The 20 MB of SmartCache is shared among the 8 cores of a socket. Each node has 32 GB of RAM. This system uses the Clustered High Availability Operating System (CHAOS), a modified version of RedHat Linux.

2.3.2 System 2 — Snow

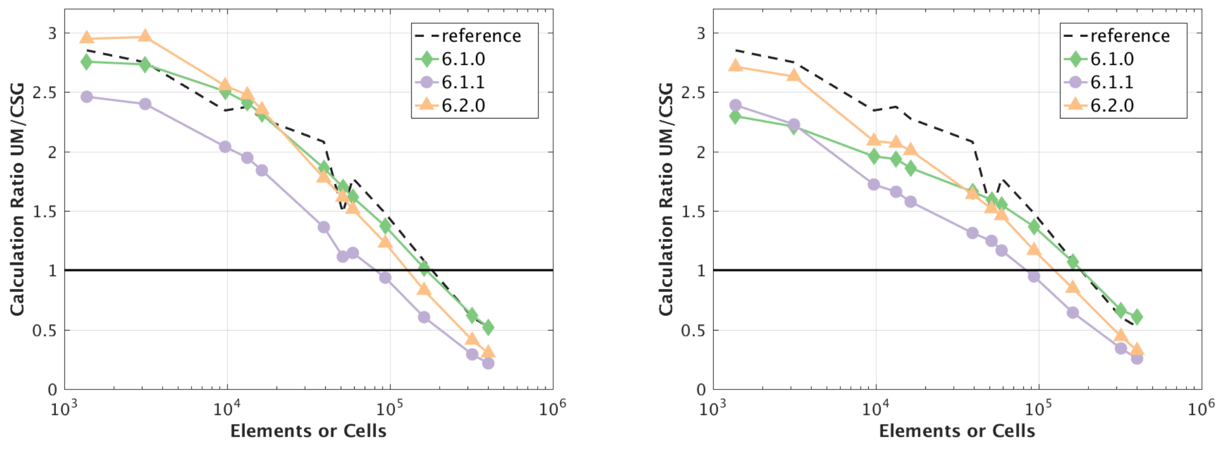
This cluster has over 368 nodes where each node has two 18-core Intel Xeon Broadwell [ES-2695v4] sockets for a total of 36 CPUs per node. The CPUs run at 2.1 GHz. The 45 MB of SmartCache is shared among the 18 cores of a socket. Each node has 128 GB of RAM. This system uses the Clustered High Availability Operating System (CHAOS), a modified version of RedHat Linux

2.4 Comparison Objective

One of the ways to show the performance of the UM models relative to the CSG models, from the Reference 2 work, is to divide the UM calculation time by the corresponding CSG calculation time, holding the number of histories constant, to generate calculational ratios as a function of element or cell count. The calculation times does not include input processing time. If the ratio is greater than 1, the CSG calculation is more efficient, and the reverse is true if the ratio is less than 1. A ratio greater than 1 is convenient in expressing how much faster CSG is compared to UM. The inverse ratio, CSG/UM, is likewise convenient in expressing how much faster UM is compared to CSG when its value is greater than 1. For reader convenience, both ratios are provided in the discussion below.

2.4.1 Kcode Results

Figure 1 shows kcode timing results on the two HPC systems mentioned in Section 2.3 using various versions of MCNP6. Included in this figure are results (noted as reference) from the previous comparison work [2]. The UM calculations become more efficient compared to the equivalent CSG starting at 85,000 elements for the 6.1.1 implementation. The 6.1.0 version doesn't reach this point until approximately 185,000 elements. The 6.2.0 version reaches this point at approximately 130,000 elements. As expected, the worst UM performance compared to CSG occurs for the mesh with the least number of elements. Using the 6.2.0 version of the code, the UM is approximately a factor of 2.9 slower (than CSG) on Snow and approximately a factor of 2.7 slower on Moonlight for the worst case.



(a) Snow (b) Moonlight

Figure 1: Performance Results: kcode

Figure 2 shows the kcode performance on Moonlight and Snow for the 6.2.0 version. Performance is comparable (agreement is good) for large element counts. At approximately 130,000 elements, the 6.2.0 version becomes more efficient for UM compared to CSG. Data used to generate this figure are provided in Tables 2 and 3. Figure 3 plots the calculation time vs. element count for the data in these tables. The scales have been adjusted to better show the transition point where UM is faster than CSG.

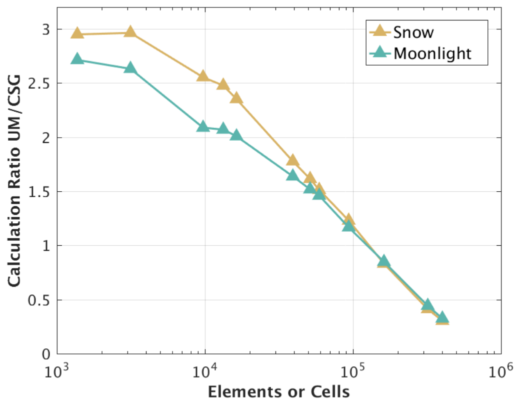


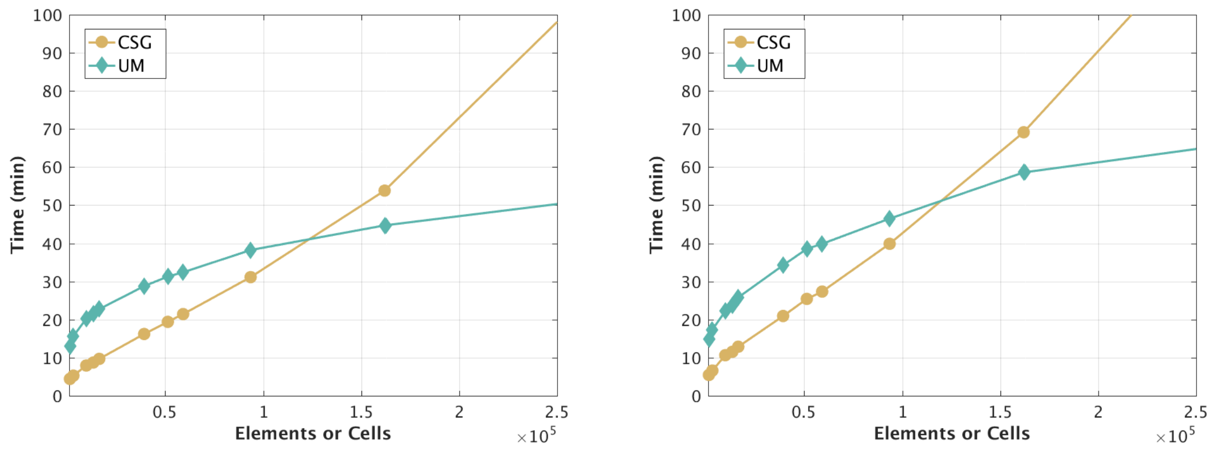
Figure 2: Snow vs. Moonlight Performance for KCODE Using Version 6.2.0

Table 2: Calculation times (minutes) and performance ratios for kcode runs on Moonlight with version 6.2.0

Case	UM Time	CSG Time	UM/CSG	CSG/UM
4.0	14.78	5.45	2.710	0.369
3.0	17.36	6.60	2.629	0.380
2.0	22.18	10.63	2.087	0.479
1.7	23.83	11.52	2.068	0.484
1.5	25.81	12.86	2.008	0.498
1.1	34.24	20.92	1.637	0.611
1.0	38.55	25.41	1.517	0.659
0.95	39.87	27.33	1.459	0.685
0.80	46.49	39.88	1.166	0.858
0.65	58.61	69.16	0.847	1.180
0.50	69.66	157.67	0.442	2.263
0.45	71.59	220.74	0.324	3.083

Table 3: Calculation times (minutes) and performance ratios for kcode runs on Snow with version 6.2.0

Case	UM Time	CSG Time	UM/CSG	CSG/UM
4.0	13.00	4.41	2.950	0.339
3.0	15.61	5.27	2.963	0.338
2.0	20.24	7.93	2.551	0.392
1.7	21.56	8.71	2.476	0.404
1.5	22.73	9.67	2.351	0.425
1.1	28.77	16.20	1.776	0.563
1.0	31.28	19.39	1.614	0.620
0.95	32.39	21.42	1.512	0.661
0.80	38.24	31.13	1.229	0.814
0.65	44.71	53.84	0.831	1.204
0.50	54.77	133.36	0.411	2.435
0.45	57.04	188.67	0.302	3.308

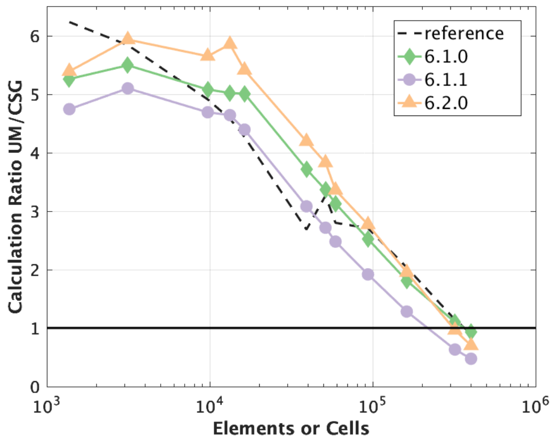


(a) Snow (b) Moonlight

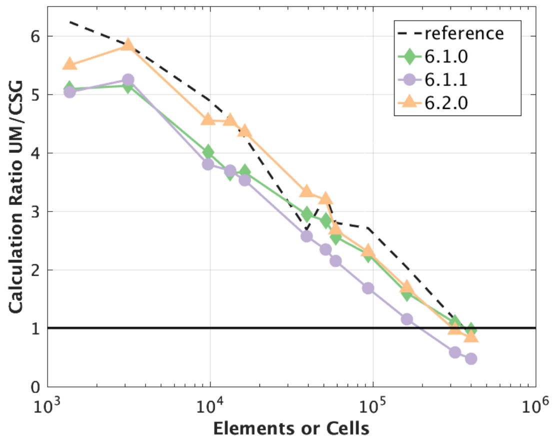
Figure 3: Alternate time vs. element comparison: kcode results

2.4.2 Void Results

As the Reference 2 work showed, the void arrangements yielded the worst performance of UM vs. CSG. Figure 4 shows void timing results on the two HPC systems mentioned in Section 2.3 using various versions of MCNP6. Included in this figure are results (noted as reference) from the previous comparison work [2]. The UM calculations don't become more efficient until about 200,000 elements with the 6.1.1 version being the most efficient for the UM. The worst UM performance is about a factor of 6 slower (than CSG) for the models with the least number of mesh elements.



(a) Snow



(b) Moonlight

Figure 4: Performance Results: void

Figure 5 shows the void arrangement performance on Moonlight and Snow for the 6.2.0 version. Performance is comparable (agreement is good) for large element counts. At approximately 300,0000 to 330,000 elements, the 6.2.0 version becomes more efficient for UM compared to CSG. Data used to generate this figure are provided in Tables 4 and 5. Figure 6 plots the calculation time vs. element count for the data in these tables. Since this void case exercises little more than the tracking routines in the code, it is probably the best, straight-up comparison of the two different tracking methodologies and appears to bound the performance ratios.

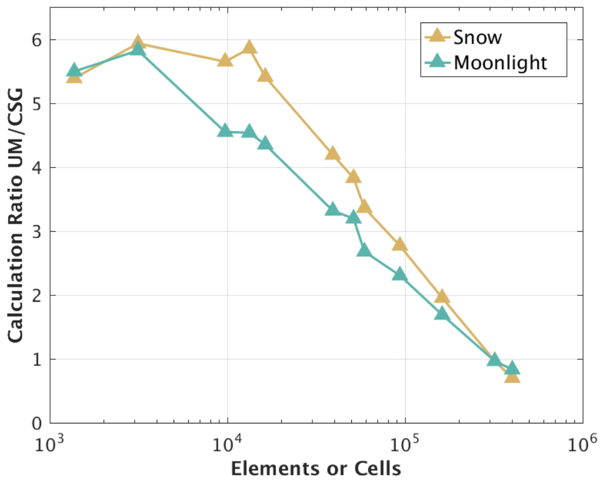


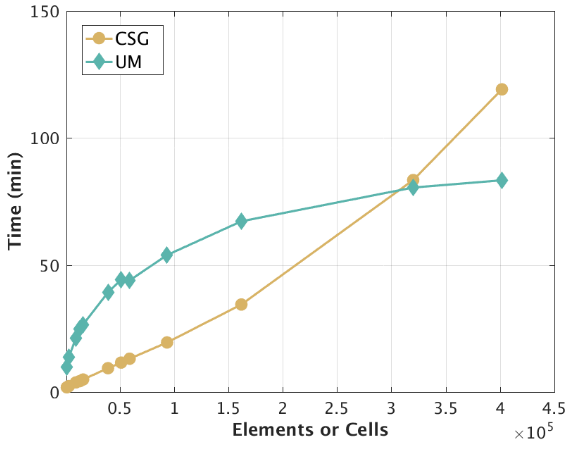
Figure 5: Snow vs. Moonlight Performance for Void Using Version 6.2.0

Table 4: Calculation times (minutes) and performance ratios for void runs on Moonlight with version 6.2.0

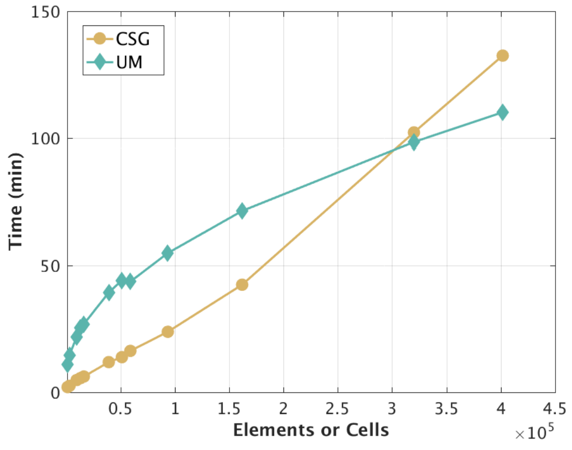
Case	UM Time	CSG Time	UM/CSG	CSG/UM
4.0	11.04	2.01	5.500	0.182
3.0	14.61	2.51	5.816	0.172
2.0	21.82	4.80	4.546	0.220
1.7	25.30	5.58	4.531	0.221
1.5	26.83	6.17	4.345	0.230
1.1	39.25	11.84	3.316	0.302
1.0	44.05	13.80	3.192	0.313
0.95	43.55	16.28	2.675	0.374
0.80	54.78	23.79	2.303	0.434
0.65	71.38	42.32	1.687	0.593
0.50	98.52	102.30	0.963	1.038
0.45	110.22	132.53	0.832	1.202

Table 5: Calculation times (minutes) and performance ratios for void runs on Snow with version 6.2.0

Case	UM Time	CSG Time	UM/CSG	CSG/UM
4.0	9.86	1.83	5.393	0.185
3.0	13.58	2.29	5.931	0.169
2.0	21.23	3.76	5.654	0.177
1.7	24.92	4.26	5.848	0.171
1.5	26.56	4.91	5.413	0.185
1.1	39.21	9.36	4.188	0.239
1.0	44.26	11.57	3.826	0.261
0.95	43.92	13.08	3.358	0.298
0.80	53.91	19.48	2.768	0.361
0.65	67.20	34.43	1.952	0.512
0.50	80.49	83.44	0.965	1.037
0.45	83.35	119.16	0.699	1.430



(a) Snow



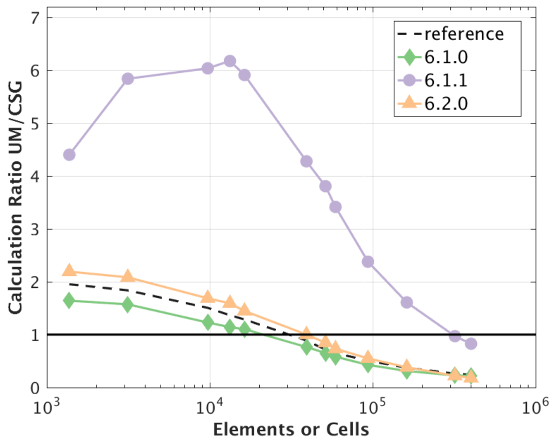
(b) Moonlight

Figure 6: Alternate time vs. element comparison: void results

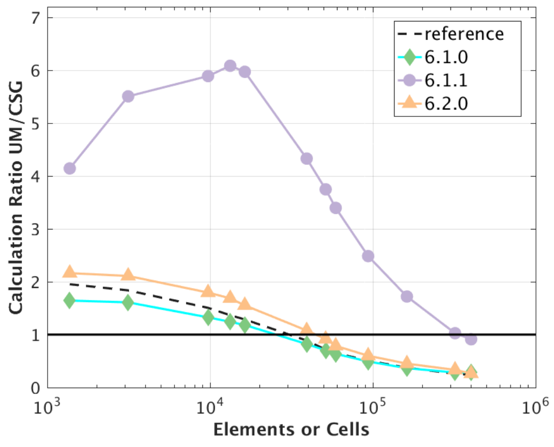
2.4.3 Gamma Results

Figure 7 shows performance results for the gamma arrangements. The first thing that is noticeable in these figures it the poor performance of the 6.1.1 version. This is attributed to a bug that was in the particle banking routine when a particle had to be banked from the mesh. This bug was introduced in version 6.1.1 and has been subsequently fixed. This bug only affected performance and did not produce any wrong answers.

The performance plots of Figure 7 show that the 6.2.0 version performance is slightly worse compared to the 6.1.0 version and the reference case. Version 6.2.0 becomes more efficient relative to the CSG equivalent at about 40,000 to 45,000 elements. Worst case performance, as shown with the models of only a few thousand elements, is slightly more than a factor of 2.



(a) Snow



(b) Moonlight

Figure 7: Performance Results: gamma

Figure 8 shows the gamma arrangement performance on Moonlight and Snow for the 6.2.0 version. Agreement of performance results is much better over the entire range of elements compared to the other arrangements, but not quite as good for the larger element counts. At approximately 40,000 - 45,000 elements, the 6.2.0 version becomes more efficient for UM compared to CSG. Data used to generate this figure are provided in Tables 6 and 7. Figure 9 plots the calculation time vs. element count for the data in these tables. The scales have been adjusted to better show the transition point where UM is faster than CSG.

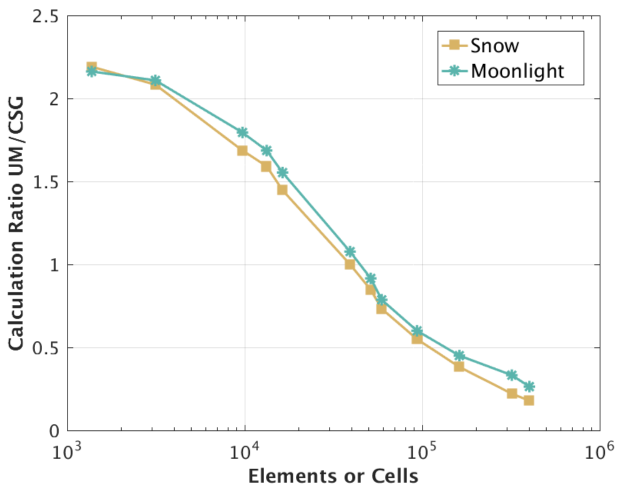


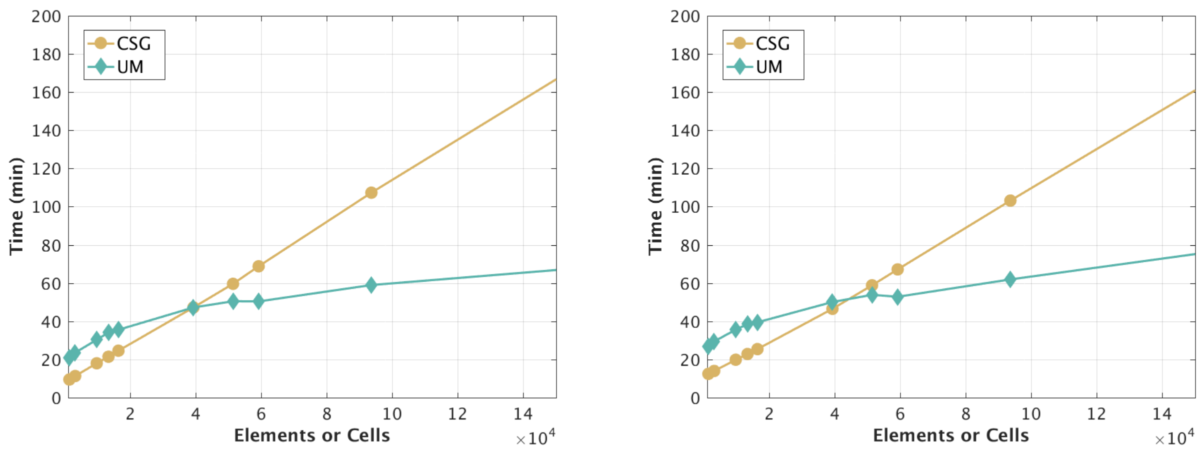
Figure 8: Snow vs. Moonlight Performance for Gamma Using Version 6.2.0

Table 6: Calculation times (minutes) and performance ratios for gamma runs on Moonlight with version 6.2.0

Case	UM Time	CSG Time	UM/CSG	CSG/UM
4.0	26.88	12.43	2.162	0.462
3.0	29.42	13.95	2.109	0.474
2.0	35.59	19.83	1.795	0.557
1.7	38.46	22.81	1.686	0.593
1.5	39.44	25.39	1.553	0.644
1.1	50.07	46.49	1.077	0.928
1.0	53.84	58.81	0.915	1.092
0.95	52.81	67.14	0.787	1.271
0.80	61.91	103.21	0.600	1.667
0.65	77.98	173.23	0.450	2.221
0.50	100.81	303.35	0.332	3.009
0.45	112.09	425.92	0.263	3.800

Table 7: Calculation times (minutes) and performance ratios for gamma runs on Snow with version 6.2.0

Case	UM Time	CSG Time	UM/CSG	CSG/UM
4.0	20.75	9.47	2.190	0.457
3.0	23.54	11.30	2.084	0.480
2.0	30.30	17.97	1.686	0.593
1.7	34.08	21.41	1.592	0.628
1.5	35.53	24.56	1.447	0.691
1.1	47.21	47.22	1.000	1.000
1.0	50.47	59.60	0.847	1.181
0.95	50.38	68.79	0.732	1.365
0.80	58.96	107.35	0.549	1.821
0.65	68.49	179.94	0.381	2.627
0.50	79.49	360.48	0.221	4.535
0.45	82.38	461.12	0.179	5.597



(a) Snow (b) Moonlight

Figure 9: Alternate time vs. element comparison: gamma results

2.4.4 Projecting For Larger Models

Creating a more-detailed Godiva model with a larger number of mesh elements than what is described in Table 1 can certainly be done. However, it would take considerably longer for the CSG input to be processed before particle tracking could take place. In lieu of doing this, the last four data points (cases 0.80, 0.65, 0.50, 0.45 in Table 1) from the Snow computer calculations (presented above) were used to extrapolate out to larger element counts using a LaGrange method; CSG/UM ratio results are provided in Figures 10 to 12.

The actual algorithm used is Neville's algorithm [5] for Lagrange interpolation. This produces a polynomial fit through the data points and also produces an error estimate for any requested value. In Figures 10 to 12 the error bands are plotted with the extrapolated results and increase as the element count increases. This increase is expected since the large element counts in the extrapolation region are farther away from the actual data points where the LaGrange method (polynomial fit) is more accurate for interpolation.

Considering an element count of 1 million, the CSG calculations are projected to take approximately 5, 11, and 15 times longer than the corresponding UM runs for the void, kcode, and gamma cases, respectively.

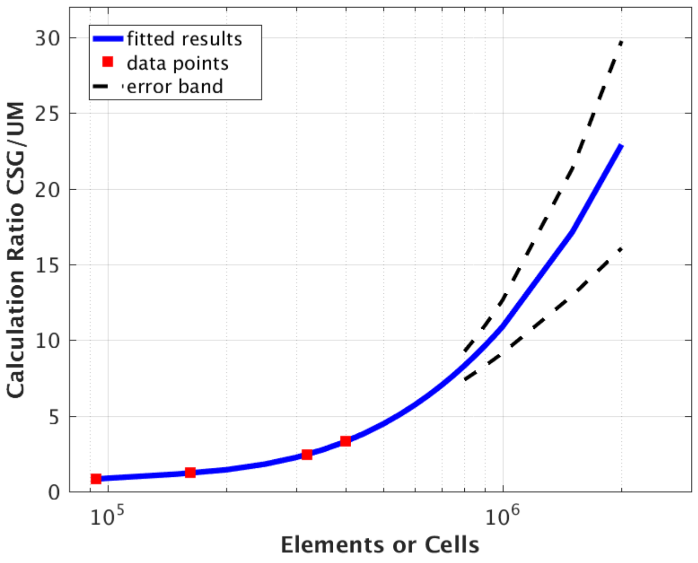


Figure 10: Extrapolation of kcode performance results

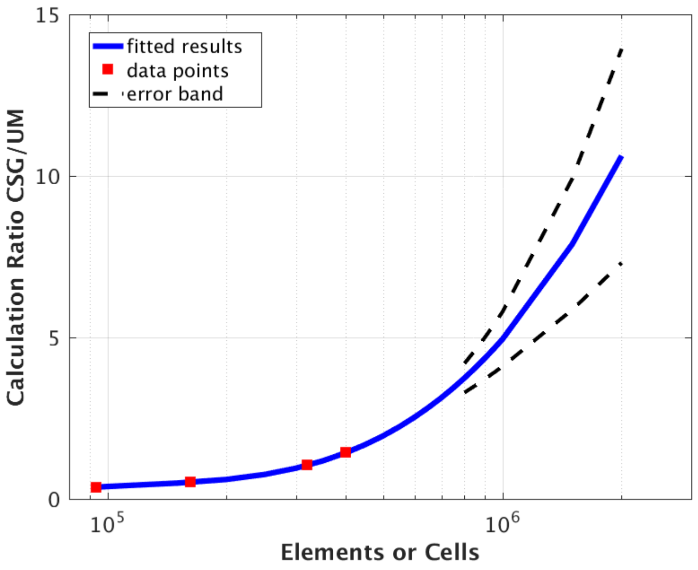


Figure 11: Extrapolation of void performance results

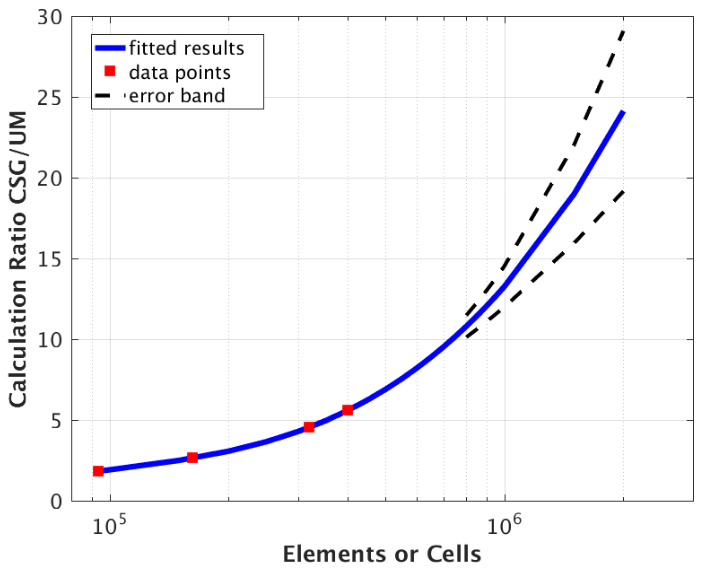


Figure 12: Extrapolation of gamma performance results

3 Summary

For MCNP Version 6.2, the UM top-level tracking routine (`regl_track_um`) has been refactored to make the code less convoluted and to make it faster, at least in some cases. This report compares performance results for equivalent CSG and UM calculations that use a "mesh" for some "Godiva-like" models. The refactored UM tracking loop was designed for those UM's where the number of elements per mfp is large along the particle's trajectory. Certainly as the mesh density increases, the UM performance improves, but it's performance relative to the older methodology shows no significant increase. Other problems with a higher mesh density could show better performance relative to the older tracking methodologies, but has yet to be demonstrated.

For models with large elements counts (where UM would most likely be considered), the 6.2.0 version of the code has comparable performance relative to previous code versions. As expected, there are situations where one version of the code performs somewhat better than the other version. Generally, the 6.1.1 version tends to produce faster runs. However, the UM tracking routine in 6.1.1 is unmaintainable going forward. There is a slight variation in performance depending upon computer hardware.

A particle banking bug that manifested itself with the UM in photon problems was inadvertently introduced in the 6.1.1 version of MCNP6. This was detrimental to code performance and has been corrected for the 6.2.0 version. Code performance for these types of problems has improved. This bug did not produce wrong results.

Extrapolating results to larger element counts indicates that the UM should dramatically outperform the equivalent CSG models in this comparison.

Acknowledgements

Thanks to Karen Kelley and Joel Kulesza for their technical reviews and suggestions that made this report better. Thanks to Jeffery Bull for suggesting the addition of the time vs element count figures that add a degree of clarity with respect to the intersecting performance curves.

References

[1] R. L. Martz, "Preliminary Investigative Results of Mixed Geometry Models in MCNP with Some Benchmark Problems," April 2008.

[2] R. L. Martz and K. M. Marshall, "A Notable Comparison of Computational Geometries in MCNP6 Calculations," *Nuclear Technology*, vol. 184, pp. 239–248, November 2013.

[3] D. B. Pelowitz, A. J. Fallgren, and G. E. McMath, "MCNP6 User's Manual: Code Version 6.1.1 beta," Tech. Rep. LA-CP-14-00745, Los Alamos National Laboratory, Los Alamos, NM, USA, June 2014.

[4] "International Handbook of Evaluated Criticality Safety Benchmark Experiments," Tech. Rep. NEA/NSC/DOC(95)03/I-IX, Organization for Economic Co-operation and Development — Nuclear Energy Agency (OECD-NEA), Paris, France, September 2013.

[5] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes – The Art of Scientific Computing.* Cambridge University Press, 1986.