MOCFE-Bone: The 3D MOC Mini-Application for Exascale Research

Nuclear Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

Availability of This Report

This report is available, at no cost, at <http://www.osti.gov/bridge>. It is also available on paper to the U.S. Department of Energy and its contractors, for a processing fee, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone (865) 576-8401
fax (865) 576-5728
reports@adonis.osti.gov

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

MOCFE-Bone: The 3D MOC Mini-Application for Exascale Research

prepared by
E. R. Wolters and M. A. Smith
Nuclear Engineering Division, Argonne National Laboratory

December 19, 2012

1. Introduction

MOCFE-Bone is a mini-application that simulates the main procedures in a 3D method of characteristics (MOC) code for numerical solution of the steady state neutron transport equation. 3D-MOC was chosen as a specific methodology of interest in exascale computing for nuclear reactor applications because of its heterogeneous geometry capability, high degree of accuracy, and potential for scalability.

In developing the mini-app, we focused on emulating the behavior of a production 3D MOC tool without requiring the input overhead associated with such a code. The MOCFE code [1] at Argonne National Laboratory was used to investigate and characterize the behavior of the mini-app; the results of are presented here. Note that the existing MOCFE code resulted from research forays into the potential of 3D-MOC for large scale parallelism and is *not* a production level tool nor is it optimally implemented. Parts of the mini-app therefore reflect lessons learned and represent how a production-level tool would be written from this point forward rather than simply duplicating MOCFE, which has known scalability issues (i.e. parallelization in energy).

2. Purpose

MOCFE-Bone is *not* meant to provide actual solutions to a given problem, as it does not properly connect the systems of equations. MOCFE-Bone simulates the communication and computational requirements of a 3D-MOC production code in correct proportions for use on different architectures. It requires only a few command line arguments to run. This document describes how to access, compile, and use the mini-app.

3. MOCFE-Bone Description

We briefly describe the MOC methodology, discuss the features of MOCFE-Bone, and draw comparisons between its behavior and a production 3D MOC code.

3.1 Methodology

The (steady-state) neutron transport equation describes the neutron flux as a continuous function of energy, angle, and space. We choose to use a deterministic formulation which requires the creation of effective multi-group cross sections, the details of which are beyond the scope of this report other than to state that they exist in a form where one can get representative solutions to the multigroup equation

$$\frac{1}{v_g} \frac{\partial}{\partial t} \psi_g(r, \hat{\Omega}, t) + \nabla \cdot \hat{\Omega} \psi_g(r, \hat{\Omega}, t) + \Sigma_{t,g}(r) \psi_g(r, \hat{\Omega}, t) = S_g(r, \hat{\Omega}, t). \quad (1)$$

Numerous methods, including MOC, have been successfully used to discretize and solve this equation in these three spaces. In all of the codes a mesh of some form is used to identify different materials in the domain. MOCFE and MOCFE-Bone both rely on a finite element mesh.

The main difference between MOC and other approaches is that MOC solves Eq. (1) along straight, parallel lines that traverse the problem domain (as opposed to using the finite element method as an interpolative basis). First, the angular variable in Eq. (1) is discretized into set of directions derived

from an angular cubature, an example of which is shown in Figure 1. In the figure, the black dots show the points where a given direction intersects the unit sphere and the coloring shows the weights assigned to each direction (needed to integrate the scattering kernel).

Figure 1. Angular cubature on the unit sphere.

For each direction, a series of straight lines, termed characteristic lines or trajectories, are laid out on the geometric domain as shown in Figure 2. As one can see in Figure 2, the trajectories are not equally spaced through the domain.

Figure 2. Example of characteristic lines traversing a finite element domain.

The introduction of characteristic lines turns Eq. (1) into T coupled differential equations for each direction where T is the total number of characteristic lines for a given direction. (Technically the number of trajectories T can be different for each direction, but we assume an average value here.) If N is the number of directions in the cubature, the $N \cdot T$ equations are coupled together on an element-wise basis by the scattering (and fission) source kernel.

The advantage of MOC is that each characteristic line is an independent differential equation that has a simple analytical solution. Each trajectory is assigned an area such that the set of trajectories crossing the domain defines the total volume of the domain and, by extension, the total volume of each

element in the domain. The global number of trajectories for MOCFE-Bone is influenced by the user input “TrajectoryArea.” Specifying a smaller trajectory area (and hence requiring more trajectories) adds computational expense but provides a more accurate solution.

The workflow in any MOC code includes (1) problem definition, (2) ray tracing (considerable component of the setup cost), and (3) numerical solution of the global problem via an iterative method.

3.2 Numerical Solver

A discretized neutron transport problem can ultimately be written as a linear matrix-vector product, $Ax=b$, where x is the vector of unknowns (various moments of the neutron flux). MOCFE-Bone uses a multigroup Krylov formulation (restarted GMRES) to solve this system of equations. In contrast, MOCFE uses a Krylov formulation (restarted GMRES) for the within-group system of equations, and Gauss-Seidel for the multigroup solution to connect different energy groups. For problems with one energy group, these two methods are equivalent. When the number of energy groups is greater than one, the two formulations differ. However, experience indicates that the methodology in MOCFE-Bone (Multigroup Krylov) is the preferred way to implement any future MOC code because of the improved scalability in energy. We note that the matrix A is never explicitly formed. The action of A is implemented through on-the-fly actions that require much less memory than does storing A explicitly.

Because the MOCFE-Bone mini-app is intended only to emulate certain performance characteristics and not actually solve problems, certain features necessary in a production level MOC solver were not implemented. One example is the simplified implementation of spatial decomposition.

3.3 Spatial Domain Decomposition (Strong Spatial Scaling) in MOCFE and MOCFE-Bone

In MOC, spatial decomposition yields a smaller per-process geometric domain, but “broken trajectory flux” moments must be added to the vector space of unknowns. These moments arise from the fact that the actual solution vector in MOC is the solution at every (element) intersection point along every trajectory in the domain. Storing such a large global vector space is impractical, and thus in serial we can instead store only the average element flux (a vector space about ~10,000 times smaller). In parallel, however, these moments are required to properly connect the spatial sub-domains and must be included in the global vector space. The communication between processors is thus the transmission of the trajectory flux moments between processors with adjacent sub-domains.

Previous studies showed that spatial decomposition of a given problem increases the GMRES (inner) iteration count in MOCFE [1]. This means that more work must be performed as the problem is spatially decomposed, which reduces the strong spatial scaling efficiency. Table 1 shows the iteration growth observed for single VHTR fuel block with reflected boundaries taken from Reference 1. In Table 1, a considerable increase in the number of inner (GMRES) iterations is seen when going from serial to parallel followed by a nearly constant regime and then another increase in the iteration count as the element count per process decreases.

Table 1. GMRES iteration growth observed in single VHTR fuel block problem under strong spatial scaling in MOCFE.

Elements per Process	# Spatial Processors	# GMRES Iterations
20,044	1	681
10,022	2	1158
5,011	4	1162
2,505	8	1239
1,252	16	1214
626	32	1192
313	64	1313

Here we discuss the origin and significance of the iteration growth behavior in MOCFE noting that the following factors can affect the number of iterations: communication latency, increased vector space due to storing broken trajectory fluxes in parallel, and scattering properties.

Two one-group test problems were used to display the iteration growth in MOCFE resulting from spatial decomposition: (1) a transmission (no scattering) problem and (2) a strongly scattering problem. Note that the average trajectory area (TA) was varied from 0.02 to 0.005 for these two problems in order to determine the effect of trajectory density, if any, on iteration count. The physical element size (0.3125 cm x 0.3125 cm x 0.35 cm) and mesh (32x32x32 elements) are fixed for both problems.

The transmission problem is not representative of reactor physics problems, as almost all medium have considerable amounts of (within-group) scattering. For the transmission problem, the cross sections were selected such that the global domain is well connected (each element is about 3% of a mean free path, with the entire domain being about 1 mfp) and the scattering source is effectively zero. Table 2 tabulates the average number of iterations per fission source (outer) iteration required for convergence in MOCFE, which is plotted in Figure 3. Note that we used not only the MOCFE GMRES algorithm, but also the common source iteration algorithm (simplest scheme with low memory overhead using Richardson iteration), which is identically a block Jacobi scheme when used in parallel.

Figure 3. Transmission problem: inner iteration growth under strong spatial scaling.

Table 2. Transmission problem: average # inners per outer iteration under strong spatial scaling.

Spatial Processors	Transmission TA=0.02 (GMRES)	Transmission TA=0.015 (GMRES)	Transmission TA=0.01 (GMRES)	Transmission TA=0.005 (GMRES)	Transmission TA=0.02 (Source Iteration)
1	2.0	2.0	2.0	2.0	4.0
2	4.0	4.0	4.0	4.0	5.0
4	7.4	7.1	6.6	5.7	7.0
8	9.7	9.6	9.4	8.2	11.0
16	13.6	13.9	13.8	13.2	16.0
32	20.5	20.5	19.7	19.6	25.0

Note that the trajectory density (TA) has no real impact on the iteration count for this problem. From Figure 3, one can see that the parallel methodology of MOC effectively degrades into a block Jacobi scheme, although even a rudimentary preconditioner would considerably change that behavior. The primary problem with using a block Jacobi methodology is exactly what is observed above: the introduction of sub-domains translates directly into latency in the algorithm for which unpreconditioned GMRES is no better than source iteration. In this example, the iteration count grows almost linearly with processor count, which is closely tied to the domain decomposition algorithm.

Comparison with the MOCFE VHTR problem in Table 1 shows that the iteration growth in Table 2 is much more severe and that no stable plateau in iterations is present. We can conclude that the VHTR problem is not a transmission-only case, although transmission can factor into the performance. While we can construct algorithms that are modestly scalable for transmission only, considerable work is required

to deploy those schemes on parallel machines and maintain scalability. In the literature one finds few examples of such schemes being scalable on more than a few hundred to a few thousand processors for non-idealized problems. More problematic for such sweeping schemes are the facts that (1) all reactor physics problems do have scattering and (2) if one is presented with a pure-absorber transport problem, one should use the point kernel technique because it will give the exact answer with far less coding effort than MOCFE requires. In conclusion, the pure transmission case above shows that the unpreconditioned GMRES algorithm breaks down into a block Jacobi scheme when no scattering is present.

Given the preceding problem, one should not be surprised that the second problem studied has considerable scattering and therefore better represents the reactor physics problems of interest. The cross section data for the second problem was selected such that each element is ~ 1 mean free path across with a global domain size of ~ 32 mfp. The scattering ratio is close to unity, which is unrealistically high but displays the other physics limit of the MOCFE code (on the other end of the spectrum from a pure transmission problem). Table 3 tabulates the MOCFE iteration count for the scattering test problem, and Figure 4 plots the data.

Figure 4. Inner iteration growth under strong spatial scaling for scattering problem.

Table 3. Scattering problem: average # inner iterations per outer iteration under strong spatial scaling.

Spatial Processors	Scattering TA=0.02 (GMRES)	Scattering TA=0.015 (GMRES)	Scattering TA=0.01 (GMRES)	Scattering TA=0.005 (GMRES)
1	13.0	13.1	12.9	15.6
2	28.8	27.5	27.2	24.6
4	32.4	32.3	31.7	32.8
8	38.3	38.3	38.7	39.4
16	37.6	37.2	36.8	40.1
32	39.2	39.6	42.2	41.7

As seen in Table 3, the scattering problem requires a considerably larger number of iterations in serial relative to the transmission-only case. We did not attempt to use source iteration on this problem because doing so would require $\gg 100$ iterations unless one implements diffusion synthetic acceleration (i.e. a preconditioner). Instead of growing linearly (as seen in the transmission only problem), the iteration count stabilizes after an initial jump from serial to parallel, a result that is consistent with the VHTR results from MOCFE in Table 1. In effect, the VHTR problem lies somewhere between the two bounding cases of Table 2 and Table 3.

The transmission component is obviously driving the increased costs. In effect this shows that the unpreconditioned GMRES algorithm used in MOCFE will always have a scalability issue when compared with the serial run case. Of course we cannot run many interesting reactor physics problems on a serial workstation, and thus we can get scalability numbers only for overly simplistic problems like the single VHTR assembly or the two test problems cited above. What makes the parallel methodology of MOCFE different from existing approaches is the acceptance that, given we are going to use a parallel machine, what we *really* desire is a consistent scalability without a punishingly difficult to use (or tune) algorithm. In this context, the nearly flat iteration count observed for *parallel* calculations in Table 3, and that reported in Reference 1 for the full core VHTR, makes the MOCFE methodology viable for providing the desired solution capability. The facts that one can create a preconditioner that accounts for some part of the transmission and that larger reactor physics problems are more dominated by scattering (i.e. not homogeneous) rather than transmission-only provides more support for this choice. To summarize, we have displayed the following behavior for MOCFE:

1. The serial case always requires the fewest number of iterations.
2. There is always a penalty in the form of iteration growth when going from serial to parallel spatial decomposition. This is due to the additional degrees of freedom (broken trajectory fluxes along interior spatial domain boundaries) which are added to the system of equations and also need to be communicated between neighboring processors, as well as the latency in transmitting information across several processor domains.
3. For realistic (scattering) problems, the iteration count stays roughly constant (and thus scalable) for parallel calculations given that the number of elements per process is at least 1,500. (i.e., do not decompose to 1 element per process – use at least 1,500 elements per process).

With the behavior of MOCFE displayed, we return the focus to MOCFE-Bone.

In MOCFE-Bone, the spatially-decomposed domains are not connected: the right amount of data is communicated to the correct processors, but it is not used (it is zeroed out). The reason behind this is that MOCFE-Bone poses the ideal case of neighbor to neighbor exchange (six neighbors for an interior domain) and that a true unstructured mesh will have more than six neighbors. The intent is to make MOCFE-Bone emulate MOCFE by using additional input to allow additional artificial connections to appear for more neighbors where the total message size is kept constant (i.e., correctly proportioned to the trajectories traversing the sub-domain). Additional input is also going to be added to vary the trajectory density (roughly constant in MOCFE-Bone), which appears as a load imbalance in MOCFE and can be emulated easily in MOCFE-Bone. Both of these factors affect scalability and one can thus use the

MOCFE-Bone to construct the bounds on scalability for a given machine: the idealized case (current MOCFE-Bone) and the largest tractable communication pattern and acceptable load imbalance.

To use MOCFE-Bone the user should select an intelligent number of iterations (measurements) relative to the GMRES backvectors based on the behavior of MOCFE for large scale problems which we loosely state as **60 iterations for 30 GMRES backvectors (per outer iteration)**. Obviously, one need not run 5,000 iterations as seen for the full core VHTR problem in Reference 1, but one should at a minimum use 60 iterations with 30 GMRES backvectors (to get the proper ratio of GMRES work to MOC-based work). If that is an insufficient number of measurements, then we suggest increasing the iteration count as needed, such as using 240 iterations with 30 backvectors. To simulate X outer iterations, one can multiply the number of GMRES iterations by X and keep the backvectors constant.

3.4 Discretization in Space, Energy, and Angle

Here we briefly describe the space, energy, and angle discretizations used in MOCFE and MOCFE-Bone.

Space Discretization

Defining trajectories for an unstructured mesh is complicated and requires substantial amounts of coding. In order to reduce that overhead, MOCFE-Bone was built around 3D structured Cartesian element geometry. (MOCFE uses unstructured grids with prisms, tetrahedral, and/or hexahedral elements where defining trajectories requires considerable computational effort). The choice of structured geometry in the mini-app, as opposed to unstructured geometry, has an impact on both the computational and communication effort.

Both MOCFE and MOCFE-Bone use a flat source approximation (the source within an element is restricted to a flat function in space) where we know that a linear one is needed. This limits the variation of the flux over an element, and requires substantially more elements to get an accurate solution. Because of the large number of global elements required, parallelism in space is needed to solve any practical 3D problem. In the previous section, the impact of using disconnected spatially-decomposed domains in MOCFE-Bone was discussed and circumvented by specifying the appropriate number of iterations to perform. Additionally, the communication patterns are different in MOCFE-Bone, as discussed here.

The structured geometry in MOCFE-Bone naturally results in different communication patterns from MOCFE because of the pre-determined location of nearest neighbors in a structured grid. For example, a process owning a box-shaped local domain in MOCFE-bone has at most six nearest neighbors in 3D (north, east, west, south, top, and bottom). The number of nearest neighbors could be much higher in MOCFE due to unstructured grid partitioning. The MOCFE-Bone mini-app currently communicates to its six nearest neighbors and is ideal with regard to communication. One can make the MOCFE-bone code emulate the unstructured mesh behavior by reducing the amount of information to communicate and connecting to more “neighbors”; this project has been identified as future work.

Energy Discretization

MOCFE-Bone uses the multigroup approximation common to nearly all deterministic transport methods. The same approximation is used in MOCFE; however, the method used to solve the group

fluxes differs in MOCFE and MOCFE-Bone. MOCFE uses a Krylov within-group formulation, and Gauss-Seidel to solve the multigroup system, which does not lend itself to efficient scaling in energy. Using the lessons learned from MOCFE, we built MOCFE-Bone with a Krylov multigroup formulation, which is scalable in energy with respect to the time-dependent problems of interest.

Angular Discretization

Like MOCFE, MOCFE-Bone uses discrete angles (ordinates) on the unit sphere. Each discrete angle has a corresponding set of trajectories lying parallel to it that traverse the domain. Parallelism in angle for MOCFE and MOCFE-Bone indicates that each processor evaluates trajectories and degrees of freedom corresponding to a subset of angles. MOCFE has demonstrated 90% strong scaling in angle and 93% weak scaling in angle [1], and the mini-app should also exhibit these good angle-scaling properties. Since the angular cubatures used in MOCFE and MOCFE-Bone are slightly different, we impose using a multiple of 8 angles in MOCFE-Bone to emulate a realistic cubature.

4. Accessing the Source Code

We assume that interested readers of this manual are aware of and have access to the exascale SVN repository. The most recent source code for MOCFE-Bone can be downloaded from the repository:

https://lewis.ci.uchicago.edu/repos/ne-codesign/mini_apps/mocfe_bone

The MOCFE-Bone source code is under development, and the most recent version may be obtained by using *svn update* commands.

5. Linking and Compiling

MOCFE-Bone is written in Fortran and uses MPI for parallelism. The makefiles “Makefile.txt” and “Makefile_BGP.txt” are provided. The user should specify the installed MPI directory and preferred compilers if they differ from those specified in the makefile.

The user should then copy the appropriate text file into a new file called “Makefile” and type “make”. An executable called “emulate_MOC.x” will be created.

6. Usage

The mini-app does not require any input files and instead uses a limited number of command line options. Here we describe what the mini-app is simulating and what the user can change with the input options.

Physical Size

The physical dimensions of an element (0.125 cm x 0.125 cm x 0.14 cm) are hard-wired inside the code. This element size is consistent with the typical element size in MOC transport methods using flat sources (roughly 512 elements per cubic cm). Increasing the problem size in MOC (weak scaling) typically corresponds to increasing the physical dimensions of the problem, without refinements of the mesh. For this reason, the physical dimension of each cell has been hardwired. We emphasize that when one specifies more elements, one is running a physically larger problem. Additionally, when you specify

asymmetric spatial decomposition (e.g. 2 procs in X, 2 procs in Y, 1 proc in Z) the system physically extends in dimensions X and Y because the definition of MeshScale and the hard-wired geometry (see below).

Composition/Geometry

Material cross sections are hard-wired inside the code such that a well-posed system of equations will be produced for any number of energy groups. The physical pattern of cross sections is a “red/black” checkerboard, where two compositions alternate in a red/black pattern in 3D. The black composition has a high total cross section and has no fission. The red composition has a low total cross section and has fission. This pattern creates a meaningfully “difficult” problem for the GMRES solver, although the solver will execute the specified number of iterations regardless of how quickly it converges (due to the disconnection of spatially-decomposed domains).

User-Specified Input

Command-line input is used to define the number of energy groups, number of angles, number of mesh cells per spatial process, and the parallelization in energy, angle, and space. Additionally, the trajectory spacing, number of Krylov iterations and number of Krylov backvectors are provided on the command line. The 11 input arguments are described in Table 4 and must be provided to MOCFE-Bone in the same order:

```
emulate_MOC.x <1> <2> <3> <4> <5> <6> <7> <8> <9> <10> <11>
```

Table 4. Input arguments for MOCFE-Bone.

Arg.	Name	Description	Type	Possible Values
<1>	Group_Visible	Number of energy groups to simulate.	Int	1,2, ...
<2>	GroupsPerProcess	Number of energy groups to assign per process (parallelism in energy)	Int	Min value 1 (maximum parallelism) Max value Group_Visible (no parallelism)
<3>	Angle_Visible	Number of angular directions to simulate	Int	Multiple of 8: 8,16, 24, 32...
<4>	AnglesPerProcess	Number of angular directions to assign per process (parallelism in angle)	Int	Min value 1 (maximum parallelism) Max value Angle_Visible (no parallelism)
<5>	MeshScale	Number of cells in the X,Y, and Z directions per process. MeshScale ³ gives the total number of elements per process.	Int	1,2, ... (max value limited by machine memory) Recommended to use at least 8, which gives 512 cells per process.

Arg.	Name	Description	Type	Possible Values
<6>	ParallelInX	Number of X-axis processors to partition into. The total number of elements along the X-axis is MeshScale*ParallelInX.	Int	Min value 1 (no parallelism in X) Max value limited by # processors available
<7>	ParallelInY	Number of Y-axis processors to partition into. The total number of elements along the Y-axis is MeshScale*ParallelInY.	Int	Min value 1 (no parallelism in X) Max value limited by # processors available
<8>	ParallelInZ	Number of Z-axis processors to partition into. The total number of elements along the Z-axis is MeshScale*ParallelInZ.	Int	Min value 1 (no parallelism in X) Max value limited by # processors available
<9>	TrajectorySpacing	Maximum trajectory area (cm ²) during the trajectory creation process	Real	<1.0 Recommended to use at most 0.1. Decrease this number if the output reports “missed” elements.
<10>	KrylovIteration	Total number of Krylov (GMRES) iterations to run	Int	≥ 60 Increase this number to simulate: - a more difficult problem - known iteration growth in parallel spatial decomposition One may also want to increase this to get more timing measurements.
<11>	KrylovBackVectors	Number of backvectors to use in restarted GMRES method	Int	≥ 30 Restarted GMRES is not guaranteed to converge unless the number of backvectors is sufficient. We recommend at least 30 to simulate a practical calculation.

7. Example Inputs

In this section we provide a few examples of MOCFE-Bone input and explain the use of each input parameter.

Ex. 1: Example with no parallelization

```
emulate_MOC.x  2 2  8 8  32 1 1 1  0.01 60 30
```

This input specifies 2 energy groups and indicates that both groups are to be assigned to every process (i.e., no parallelization in energy). It also says to use 8 angles and to assign all of them to every process, (i.e., no parallelization in angle). The MeshScale variable is 32, which indicates that a $32 \times 32 \times 32$ mesh will be placed on each spatial processor. There is 1 processor along each of the X, Y, and Z-directions, for a total of $1 \times 1 \times 1 = 1$ spatial processors (i.e., serial). The physical dimensions of this problem are therefore $(32 \times 0.125) \times (32 \times 0.125) \times (32 \times 0.14) = 4 \text{ cm} \times 4 \text{ cm} \times 4.48 \text{ cm}$.

Trajectories will be assigned to the domain surfaces with a minimum density of 1 trajectory/ 0.01 cm^2 , or 100 trajectories/ cm^2 . The Krylov method (restarted GMRES) will be executed for 60 iterations, using 30 backvectors.

Because there is no parallelization in energy, angle, or space, this problem should be invoked with 1 processor “`mpiexec -n 1 emulate_moc.x 2 2 8 8 32 1 1 1 0.01 60 30`” (or an equivalent script notation).

Ex. 2: Example with parallelization in space

```
mpiexec -n 32 emulate_MOC.x 2 2 8 8 32 4 4 2 0.01 120 30
```

This problem is the same in energy and angle as example 1 and uses a MeshScale of 32. There are 4 processors in the X direction, 4 processors in the Y direction, and 2 processors in the Z direction for a total of 32 spatial processors. Therefore, the total number of elements is $(32 \times 4) \times (32 \times 4) \times (32 \times 2)$, that is, 1,048,576 elements divided evenly over 32 processors. The physical problem size is, correspondingly, $(32 \times 4 \times 0.125) \times (32 \times 4 \times 0.125) \times (32 \times 2 \times 0.14)$, or $16 \text{ cm} \times 16 \text{ cm} \times 8.96 \text{ cm}$. Trajectories will be assigned to the domain surfaces with a minimum density of $1/0.01 \text{ cm}^2$, or $100/\text{cm}^2$. The Krylov method (restarted GMRES) will be executed for 120 iterations, using 30 backvectors. The number of GMRES iterations was chosen to be roughly double the number specified in serial to simulate the iteration growth under spatial decomposition.

Ex. 3: Example input using parallelization in angle

```
mpiexec -n 2 emulate_MOC.x 2 2 8 4 32 1 1 1 0.01 60 30
```

This problem is identical to example 1 except that only 4 angles are assigned to each process instead of the full 8. Therefore, $8/4=2$ processors are needed to parallelize in angle. The problem is serial in space and energy, so only $1 \times 1 \times 2 = 2$ processors are required in total.

8. Example output

The output is printed to the screen and contains an echo of the input options used; counts of the global elements, trajectories, and intersections; and a report of any missed elements. If missed elements are reported (should be rare), the user should request a smaller trajectory area as missing elements will destroy the load balance.

The number of trajectories broken in (X,Y,Z) indicates the additional degrees of freedom added to the problem as a result of spatial decomposition. In the following example, parallelization was specified only along the X direction, so broken trajectories appear only along X. The reported time at the end of the simulation is the time spent in GMRES only.


```

$ mpiexec -n 2 emulate_MOC.x 1 1 8 8 32 2 1 1 0.1 100 30

[MOC emulator] Welcome to the emulated world of MOC
[MOC emulator] Processed input variables
[MOC emulator] ReactorType          0 (0,1,2)=(Fast,thermal,a mess).....
[MOC emulator] SolutionMethod       0 Specifies the solution method used.....
[MOC emulator] Group_Visible        1 The number of energy groups to simulate.....
[MOC emulator] GroupsPerProcess     1 The number of energy groups to assign per process.....
[MOC emulator] Angle_Visible        8 The number of angular directions to simulate.....
[MOC emulator] AnglesPerProcess     8 The number of angular directions to assign per process....
[MOC emulator] MeshScale            32 Specifies the number of X,Y,Z cells per process.....
[MOC emulator] ParallelInX          2 Specifies the number of X-axis processors to partition into
[MOC emulator] ParallelInY          1 Specifies the number of Y-axis processors to partition into
[MOC emulator] ParallelInZ          1 Specifies the number of Z-axis processors to partition into
[MOC emulator] TrajectorySpacing 0.100000000 Specifies the maximum trajectory area to
assign.....
[MOC emulator] KrylovIterations      100 Specifies the number of Krylov iterations to run.....
[MOC emulator] KrylovBackVectors     30 Specifies the number of back vectors to use.....
[MOC emulator] Global number of elements..... 65536
[MOC emulator] Global number of trajectories..... 40960
[MOC emulator] Global number of intersections..... 1.589384000E+06
[MOC emulator] Max by angle number of trajectories... 3142
[MOC emulator] Min by angle number of trajectories... 3072
[MOC emulator] Total elements missed in all angles... 0
[MOC emulator] Number of trajectories broken in X... 8752
[MOC emulator] Number of trajectories broken in Y... 0
[MOC emulator] Number of trajectories broken in Z... 0
[MOC emulator] Global space-angle dofs..... 574000
[MOC emulator] Global space-angle dofs..... 574000.
[MOC emulator] Global space-angle-energy dofs..... 574000.
[MOC emulator] Global trajectory memory storage (MB). 20.
[MOC emulator] Global Krylov vector storage.....(MB). 276.
[MOC emulator]...Solving Ax=b with FGMRES
[MOC emulator]...Convergence checks have been disabled on purpose
[MOC emulator]...Exactly performing the number of user-specified iterations!
[MOC emulator]...Iteration          1
[MOC emulator]...Iteration          2
...
[MOC emulator]...Iteration          100
[MOC emulator] Krylov solver appears to have exited normally
[MOC emulator] Reason for convergence 1 and norm 3.251141104E-03 FGMRES took 1.459878E+01 seconds
and 100 iterations

```

In addition, a file called “ParallelSetupDetails.out” is created that indicates how the degrees of freedom are divided up by processor (rank). This example has two ranks, and as expected, each rank owns half of the total 574,000 space-angle-energy dofs. Additionally, each rank owns half of the element-wise SN fluxes (elements x angles = 65536 x 8 = 524,288). The “BC DOMAIN OUT” is the number of surfaces that define the boundary of the domain multiplied by the number of exiting directions where “BC DOMAIN IN” is the incident directions. The “TRAJ OUT” is the number of broken trajectory moments leaving process 1 in all directions while “TRAJ IN” is the number of broken trajectory moments entering the domain. These numbers can be highly variable as processors with sub-domains that lie on the domain boundary can have zero exiting or entering broken trajectories. These numbers give a sense of the GMRES load imbalance associated with a given spatial decomposition.

```
$ more ParallelSetupDetails.out
```

TOTAL	GROUP	ANGLE	SPACE	SPACE-ANGLE	SPACE-ANGLE	ELEMENT	BC DOMAIN	BC DOMAIN	TRAJ	TRAJ	TOTAL	TOTAL
RANK	RANK	RANK	RANK	OWNED	GHOSTED	OWNED	OUT	IN	OUT	IN	TRAJ	INTERS
0	0	0	0	287000	24856	262144	20480	20480	4376	4376	24856	794692
1	0	0	1	287000	24856	262144	20480	20480	4376	4376	24856	794692

9. Summary

The MOCFE-Bone mini-application is a standalone program for exascale research. The purpose of MOCFE-Bone is to model the defining behaviors of a parallel 3D method of characteristics code for neutron transport applications, without the overhead of multiple input files. The MOCFE research code at Argonne National Laboratory was used as a model to determine what behaviors the mini-application should exhibit. Similarities and differences between MOCFE-Bone and MOCFE are discussed in this report. Additionally, a brief discussion was included of the known iteration growth behavior in MOCFE under strong spatial scaling. Instructions on how to access, compile, and use MOCFE-Bone were given and examples were provided to show proper usage of the code.

10. References

- [1] M.A. Smith, A. Marin-Lafleche, W.S. Yang, D. Kaushik, and A. Siegel, “Method of Characteristics Development Targeting the High Performance Blue Gene/P Computer at Argonne National Laboratory”, International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Rio de Janeiro, Brazil, May 8-12, 2011.

Nuclear Engineering Division

Argonne National Laboratory

9700 South Cass Avenue, Bldg. 208

Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC