SNA + MC 2013, 06016 (2014)
DOI: 10.1051/snamc/201406016

OpenMC: A State-of-the-Art Monte Carlo Code for Research and Development

Paul K. Romano[1], Nicholas E. Horelik[1], Bryan R. Herman[1], Adam G. Nelson[2], Benoit Forget[1], and Kord Smith[1]

[1]*Massachusetts Institute of Technology, Department of Nuclear Science and Engineering, 77 Massachusetts Avenue, Cambridge, MA 02139*
[2]*University of Michigan, Department of Nuclear Engineering and Radiological Sciences, 2355 Bonisteel Boulevard, Ann Arbor, MI 48104*

This paper gives an overview of OpenMC, an open source Monte Carlo particle transport code recently developed at the Massachusetts Institute of Technology. OpenMC uses continuous-energy cross sections and a constructive solid geometry representation, enabling high-fidelity modeling of nuclear reactors and other systems. Modern, portable input/output file formats are used in OpenMC: XML for input, and HDF5 for output. High performance parallel algorithms in OpenMC have demonstrated near-linear scaling to over 100,000 processors on modern supercomputers. Other topics discussed in this paper include plotting, CMFD acceleration, variance reduction, eigenvalue calculations, and software development processes.

KEYWORDS: Monte Carlo, neutron transport, OpenMC, parallel, XML, HDF5

I. Introduction

OpenMC is a relatively young Monte Carlo particle transport code, having been developed starting in 2011 and first released to the public in December 2012. While the code does not benefit from decades of experience and feedback from users as do other popular Monte Carlo codes such as MCNP[(1)] and TRIPOLI,[(2)] it nevertheless possesses a number of features that may be very attractive to both users and developers.

Development of OpenMC was spearheaded by the Computational Reactor Physics Group (CRPG) at Massachusetts Institute of Technology (MIT) as part of a project to develop scalable parallel algorithms for future exascale supercomputers. While this was the original focus of the code development efforts, there are now a wide variety of research and development efforts underway using OpenMC.[(3–6)] In the last year, the development team has also grown to span multiple organizations.

Various aspects of the OpenMC code have been described previously.[(3,7)] However, due to the developmental nature of the code, many changes have been, and continue to be, made. The objective of the present work is to give a fairly complete and up-to-date overview of the present capabilities and features of OpenMC.

II. Methods

1. Physics

At the present time, OpenMC is capable of simulating only neutrons either in fixed source[1] or k-eigenvalue problems. The data governing the interaction of neutrons with various nuclei are represented using the ACE format[(8)] which is also used by MCNP and Serpent.[(9)] ACE-format data can be generated with the NJOY nuclear data processing system[(10)] which converts raw ENDF/B data into a representation that is suitable for use in a Monte Carlo code. The use of a standard cross section format allows for a direct comparison of OpenMC with other codes since the same cross section libraries can be used. However, the downside is that the implementation of physical methods is necessarily limited by the data that is available in the ACE format.

An indexing technique[(11)] based on pointers is used to speed up energy grid searches when calculating cross sections. For problems with tens of nuclides or less, the indexing technique provides a considerable performance benefit with modest additional memory requirements. However, with hundreds of nuclides in a problem, the memory requirements may become too prohibitive. As a result, alternative energy grid treatments are now being explored.

OpenMC is capable of faithfully simulating all nuclear reactions producing secondary neutrons, including $(n,2n)$, $(n,3n)$, fission, and level inelastic scattering, according to the various secondary energy and angle distribution laws in the ACE format data. Photon transport capability has not yet been implemented, and thus OpenMC does not explicitly track photon production resulting from (n,γ) or fission reactions.

To properly treat scattering kinematics when the target nucleus is not at rest, OpenMC uses a free gas approximation[(12)] wherein the velocities of the target nuclei are sampled from a Maxwellian distribution. For thermal neutrons scattering from bound molecules such as hydrogen or deuterium in water, graphite, beryllium, etc., the free gas approximation will not accurately capture the scattering kinematics and $S(\alpha,\beta,T)$ scattering law data must be used. The $S(\alpha,\beta,T)$ data are given on ACE files separate from the normal nuclide data. To account for self-shielding in the unresolved resonance range, OpenMC uses the probability table method.[(13)] Probability tables are included in the ACE data for many nuclides.

The method of successive generations[(14)] is used to solve

[1]Subcritical multiplication problems are not yet supported.

k-eigenvalue problems. The user also has the option to group multiple generations into a "batch" to reduce correlation between realizations of the tally random variables.[15] Like MCNP, OpenMC keeps track of the collision, absorption, and track-length estimators of k_{eff} and then calculates a minimum-variance combined estimator based on the covariance matrix of the three single estimators. To assess convergence of the source distribution, the user can also define a mesh over which the Shannon entropy should be calculated.

2. Geometry

In order to model arbitrarily complex geometric objects, OpenMC uses a constructive solid geometry representation. In the current implementation, closed volumes, or *cells*, can be represented as the intersection of multiple *half-spaces*. Each half-space is in turn defined as the positive or negative side of a plane or quadratic surface. This allows curved surfaces such as spheres and cylinders to be modeled exactly with no error due to mesh discretization. Most geometries of interest in particle transport can be modeled with first and second-order surfaces with the exception of some exotic geometries, e.g., a torus in a fusion system where a fourth-order equation is required. The restriction of closed volumes to only those that are formed by the intersection of surfaces (so-called simple cells) results in greater simplicity of the geometry implementation. However, the burden is shifted to the user who may then need to break up certain closed volumes into multiple cells.

In addition to the simple geometry constructs described above, OpenMC provides constructs that allow the user to model a two or three-dimensional structured mesh consisting of quadrilaterals. These constructs are useful for modeling the core and assembly layout in a variety of commercial and research reactor designs. As in MCNP and Serpent, these repeated structures are handled through the use of *universes*, a collection of cells occupying all of space that may be substituted within another closed volume. A universe may also be translated and/or rotated. Transmitting, vacuum, or reflective boundary conditions can be applied to any surface, thus giving the user full flexibility in the treatment of boundaries.

2.1. Geometry Plotting

Currently two different plotting capabilities are available in OpenMC. The first is a 2D raster plotting capability that allows the user to visualize the geometry along a cut plane. Plots can be colored by unique material or cell, and users have the option to selectively include/exclude certain regions. The 2D plot is written to a portable pixmap (.ppm) file which is natively viewable on many Linux distributions. Since the PPM file is uncompressed, an image converting utility can significantly reduce the size of the plot by converting it to a compressed format such as portable network graphics (.png). Figure 1 shows a 2D raster plot of a model of the Advanced Test Reactor (ATR).

In addition to 2D raster plotting, a new plotting capability has been added to OpenMC that allows for the visualization of geometry in 3D using standard viewers such as ParaView[16] and VisIt.[17] By specifying a uniform rectilinear grid of voxels (analogous to image pixels), users can produce a binary file containing the cell or material ids for each cell in the grid. These files are created with the same raster method used by the 2D plotter in OpenMC, where a 'find_cell' routine is called to determine the attributes of the geometry at each voxel center. Since this 'find_cell' routine is the same one used during simulation, the voxel information produced by this method is representative of the actual simulation geometry, limited only by the user-specified granularity of the voxels.

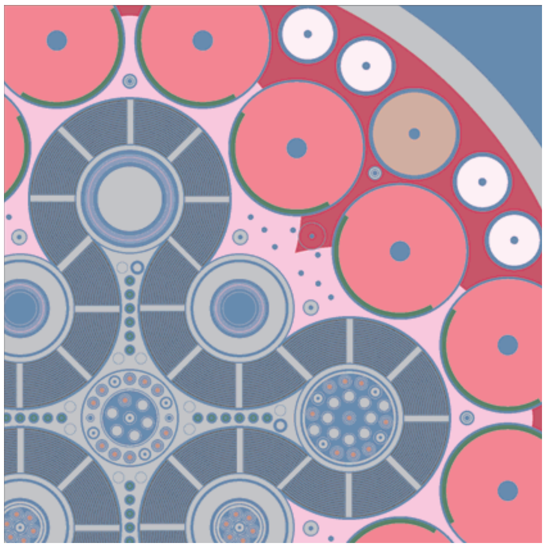


Figure 1: Cross-sectional view of a section of an OpenMC model of the Advanced Test Reactor (ATR).

Once the voxel binary file is produced, users can process the data into a standard 3D datafile and visualize however desired. For convenience, a Python utility is provided to convert voxel files into either SILO[18] or VTK[19] files that can be viewed in many well-established 3D data visualization tools. The example provided by this utility easily enables users to write custom scripts that section geometry features or mask specific cells or materials. For example, Figure 2 and Figure 3 show plots of a PWR grid spacer and burnable absorber pin, respectively, with various materials and features clipped for easy inspection of the geometry.

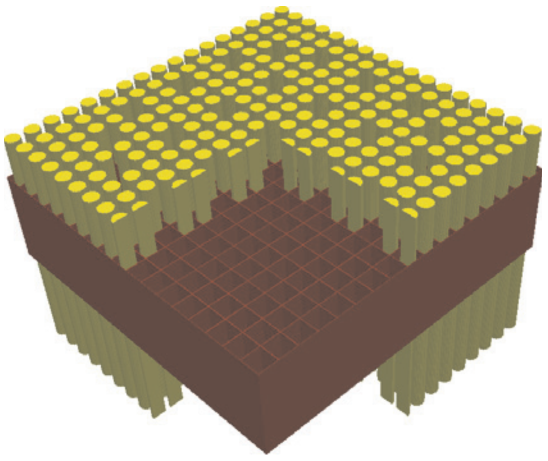


Figure 2: 3D plot of a grid spacer in the BEAVRS benchmark[20] as shown in ParaView.

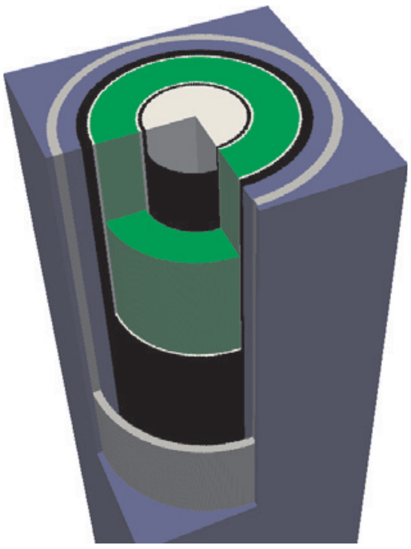


Figure 3: 3D plot of a burnable absorber pin in the BEAVRS benchmark as shown in ParaView.

3. Tallies

OpenMC has a flexible, low-overhead tally system which enables users to obtain physical results of interest. Tallies are defined by combinations of *filters* and *scores*, borrowing terminology from a paper by Sutton et al.[21] Each filter limits what events can score to the tally based on the attributes of the particle. For example, a filter could limit scoring events to particles traveling within a specified cell or a specified range of pre-collision energies. A full and up-to-date list of filters which may be applied to tallies can be found in the OpenMC User's Guide.[22] Each score identifies an actual physical quantity to be scored upon an event which matches the specified filters. Some examples of valid scores include the flux, fission reaction rate, neutron production rate, or local energy deposition from fission. In addition to a set of defined scores, it is also possible to obtain reaction rates for an arbitrary MT value. Users also have the option of scoring tallies for specific nuclides within a material. If no nuclides are specified, macroscopic cross sections for the material are used in determining scores.

With filters for pre- and post-collision energy and scoring functions for scattering and fission production, it is possible to use OpenMC to generate cross sections with user-defined group structures. The coarse mesh finite difference (CMFD) solver within OpenMC, discussed at length later, uses the tally system directly to obtain multi-group cross sections.

All tallies are scored using a track-length estimator by default. However, for tallies requiring post-collision attributes of the particle, e.g., scattering moments, a collision estimator is used instead. Users can also explicitly specify that the track-length or collision estimator should be used for a given tally.

Historically, some Monte Carlo codes have suffered severe performance penalties when tallying a large number of quantities.[23] Care must be taken to ensure that a tally system scales well with the total number of tally bins. In OpenMC, a mapping technique is used that allows for a fast determination of what tally/bin combinations need to be scored to a given particle's phase space coordinates. For each discrete filter variable, a list is stored that contains the tally/bin combinations that could be scored to for each value of the filter variable. If a particle is in cell n, the mapping would identify what tally/bin combinations specify cell n for the cell filter variable. In this manner, it is not necessary to check the phase space variables against each tally. Note that this technique only applies to discrete filter variables and cannot be applied to energy bins. For energy filters, it is necessary to perform a binary search on the specified grid.

4. Parallelism

OpenMC is capable of running in parallel using the message passing interface (MPI). Particles within a batch are divided equally among processes such that each processor has approximately the same amount of work to perform. Results of a parallel calculation are reproducible, i.e. the same result is given regardless of the number of processors that is used. At the time of writing, an implementation of shared-memory parallelism via the OpenMP API is still under development.

A substantial amount of research and development related to OpenMC has focused on scalable parallel algorithms. This R&D has culminated in the development of two algorithms which have enabled nearly linear scaling to over 100,000 processes. The first algorithm is related to the collection and sampling of fission sites, which are stored in an array known as the *fission bank*. The algorithm forgoes a typical master-slave communication pattern in favor of nearest-neighbor exchanges of fission/source sites. A derivation and analysis of the algorithm complexity is given in Romano and Forget;[24] the salient point is that nearest-neighbor exchanges, in the context of this algorithm, have an expected communication cost $O(\sqrt{N/p})$ whereas the computational work scales as $O(N/p)$. Thus, arbitrarily good scaling can be achieved. Previous scaling results were reported in [7] for the Jaguar supercomputer at Oak Ridge National Laboratory. Those results alongside with new scaling results on the Intrepid Blue Gene/P supercomputer at Argonne National Laboratory are shown in Figure 4.

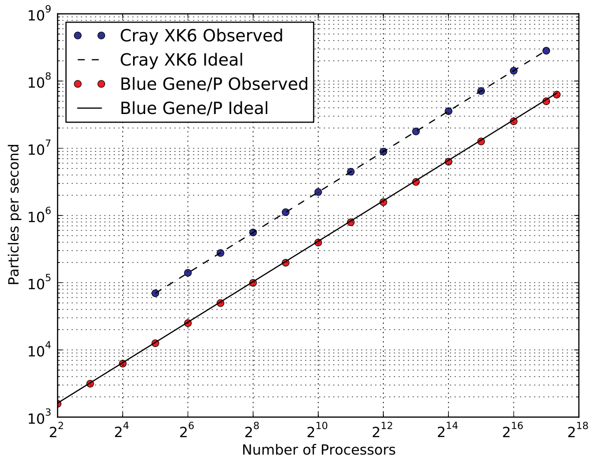


Figure 4: Parallel scaling for the Monte Carlo Performance Benchmark on the Cray XK6 (Jaguar) and Blue Gene/P (Intrepid) supercomputers.

While the novel fission bank algorithm substantially reduces communication between fission generations, it is still necessary to combine tally results from multiple processors which might entail significant network communication. The communication associated with synchronizing tallies across processors is proportional to both the number of processors and the number of tally bins. Thus, for problems with very large tally requirements, and hence large computational requirements, this source of communication can erode parallel efficiency. It was shown in[25] that by grouping realizations of tally random variables over successive generations rather than over multiple processors, the communication associated with tallies can be reduced dramatically. While the reported sample means will not change when this technique is employed, the variance of the sample mean will in general be different. However, the expected value of the variance remains the same. A user input option is available in OpenMC that modifies the grouping of tally results to reduce overall communication.

5. Variance Reduction

Extensive variance reduction techniques are not yet available in OpenMC. However, a survival biasing method has been implemented that can, under certain circumstances, increase the figure-of-merit in a simulation. When survival biasing is used, absorption never occurs explicitly and instead, a particle's weight is reduced by the probability that it would have been absorbed at each collision. Survival biasing is not turned on by default in OpenMC, but can be enabled by the user. The weight cutoff and survival weight are also adjustable parameters in the user input.

For *k*-eigenvalue problems, users can optionally use the uniform fission site method[15] on a Cartesian mesh to help flatten the distribution of variance in problems with non-uniform particle densities. The effectiveness of this method as implemented in OpenMC is discussed at greater length in.[3] One limitation currently is that the implementation assumes that the volume of fuel in each mesh cell is equal.

6. CMFD Acceleration

The coarse mesh finite difference (CMFD) method has been widely applied in deterministic nodal diffusion calculations to reduce the number of fission source iterations. Recently, CMFD has been applied to reactor calculations using multi-group Monte Carlo[26] to accelerate convergence of the fission source. The CMFD acceleration method has been integrated into OpenMC. CMFD acceleration works by solving the multi-group neutron diffusion equation after each Monte Carlo batch to obtain a better estimate of the global fission source. This process can be characterized by three steps:

1. Compute diffusion parameters by preserving OpenMC neutron balance on a coarse mesh.

2. Solve the multi-group neutron diffusion equation to obtain fission source distribution.

3. Force Monte Carlo to use fission source distribution from Step 2 by modifying source weights.

In order to use CMFD acceleration, the user needs to specify the mesh over which to calculate multi-group cross sections. Solution of the linear system of diffusion equations relies on PETSc,[27] and so OpenMC must be compiled against the PETSc libraries.

III. Design and Development

OpenMC is written in standard Fortran 2008. While C and C++ were considered as other possible languages for development, ultimately Fortran 2008 was chosen due to MIT's research focus on parallel algorithms coupled with the availability of co-array features in the Fortran 2008 standard. For input processing, OpenMC relies on a modified version of the xml-fortran[28] parser. Almost all important data are encapsulated in derived types. While object-oriented features are available in Fortran 2008, their use in OpenMC is largely precluded by limited compiler support.

1. Compilation and Installation

OpenMC has been successfully compiled with the gfortran, Intel, PGI, Cray, and IBM compilers on various platforms/architectures including several Linux distributions, Mac OS X, and Windows 7. On Windows, OpenMC can be compiled using gfortran within cygwin or the MinGW port of gcc. Recently, a binary package has been available for Debian derivatives via an Ubuntu Personal Package Archive (PPA). By installing from the binary package, the package manager ensures that all dependencies, e.g., MPI, are satisfied, and the user is spared the trouble of building the code.

2. Input/Output

2.1. Input

OpenMC uses Extensible Markup Language (XML) for all user input files. The use of XML enables developers to make changes to the user input format easily, adding or modifying options, and the xml-fortran parser within OpenMC gracefully handles the changes with little effort from the developer. Users are free to form their input files as they wish, as long as the overall structure of the files is well-formed and the content conforms to the specification of the file.

Another notable difference between OpenMC and many other transport codes is that the input is divided into multiple files rather than one file. The following files are required for every simulation:

- **settings.xml** – describes all simulation parameters, e.g., how many particles to run and the starting source, and other options that can be turned on or off.

- **materials.xml** – describes the composition of all materials in the model by their constituent elements/nuclides and densities. Natural elements are automatically expanded into individual nuclides by their natural abundance.

- **geometry.xml** – describes the model geometry using constructive solid geometry primitives (second-order surfaces, cells, universes, lattices) and assigns materials to cells.

In addition to these three basic inputs, there are a number of optional XML files:

- **tallies.xml** – specifies what physical quantities the user wants tallied during the simulation.
- **plots.xml** – describes parameters for 2D or 3D plots that are created when OpenMC is run in plotting mode.
- **cmfd.xml** – describes geometry and execution parameters for coarse mesh finite difference acceleration.

As one example of the XML format, Figure 5 shows the materials.xml describing two materials used in the U233-MET-FAST-002 benchmark from the International Handbook of Evaluated Criticality Safety Benchmark Experiments.[29]

```
<?xml version="1.0"?>
<materials>

  <default_xs>70c</default_xs>

  <material id="1">
    <density value="18.644" units="g/cm3" />
    <nuclide name="U-233" ao="4.7312e-2" />
    <nuclide name="U-234" ao="5.2770e-4" />
    <nuclide name="U-238" ao="3.3015e-4" />
  </material>

  <material id="2">
    <density value="18.80" units="g/cm3" />
    <nuclide name="U-235" ao="4.4892e-2" />
    <nuclide name="U-238" ao="3.2340e-3" />
  </material>

</materials>
```

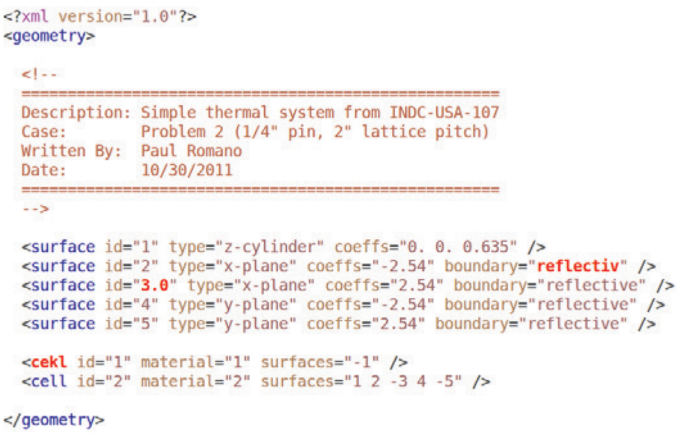
Figure 5: Material XML file for benchmark model U233-MET-FAST-002.

A set of schemata based on the RELAX NG schema language[30] makes it possible to verify not only that an input file is well-formed but also that it has the correct tags, attributes, and datatypes. There are two ways that input files can be checked for conformance against the schemata. The first method is post-validation where an input file is checked against a corresponding schema using a tool such as `jing`.[31] A more elegant method to check conformance is real-time validation with an editor such as GNU Emacs. When using GNU Emacs to write an input file, the input is continually checked against a corresponding schema (based on the root element in the document). If any errors are found, they are highlighted in red giving the user immediate visual feedback. Figure 6 shows an example of an input file being validated against a schema in GNU Emacs.

2.2. Output

When a simulation in OpenMC completes, a number of output files are written to disk. The number and format of these files depends on how the code was compiled and what options are given in the user input. The most common output files include:

1. **tallies.out** – plain text ASCII file listing the mean and standard deviation (or confidence interval half-width) for each tally bin. For simple problems with only a few tallies, this file is likely adequate for analyzing results.



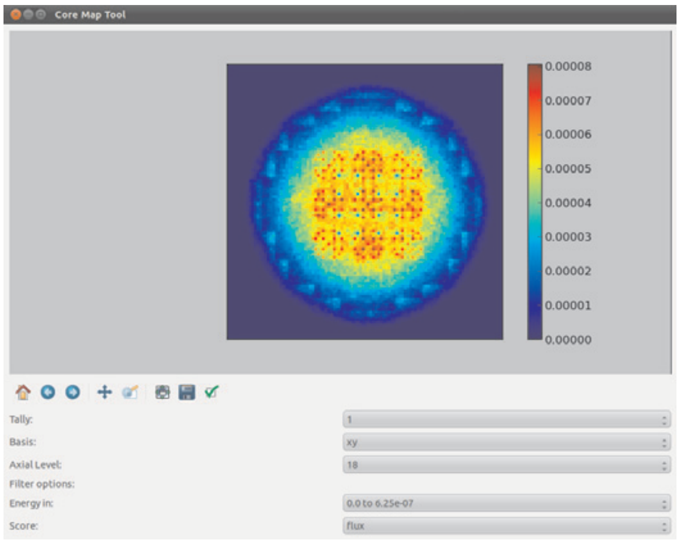
```
<?xml version="1.0"?>
<geometry>

  <!--
  ==========================================================
  Description: Simple thermal system from INDC-USA-107
  Case:        Problem 2 (1/4" pin, 2" lattice pitch)
  Written By:  Paul Romano
  Date:        10/30/2011
  ==========================================================
  -->

  <surface id="1" type="z-cylinder" coeffs="0. 0. 0.635" />
  <surface id="2" type="x-plane" coeffs="-2.54" boundary="reflectiv" />
  <surface id="3.0" type="x-plane" coeffs="2.54" boundary="reflective" />
  <surface id="4" type="y-plane" coeffs="-2.54" boundary="reflective" />
  <surface id="5" type="y-plane" coeffs="2.54" boundary="reflective" />

  <cekl id="1" material="1" surfaces="-1" />
  <cell id="2" material="2" surfaces="1 2 -3 4 -5" />

</geometry>
```

Figure 6: Example of an XML input file in GNU Emacs being validated against a corresponding RELAX NG schema in real-time. Errors in the input are automatically highlighted in red.

2. **State point files** – binary files containing all the information needed either to determine confidence intervals for tallies or to restart the run completely. By default, one state point file is written at the end of the simulation, but the user can specify particular batches at which state points should be written.

In addition to these files, numerous other output files may be generated at the request of the user or upon hitting an error in the code.

State point files are the primary means of obtaining, interpreting, and post-processing tally results. By itself, a state point file is not very useful since it is in an arbitrary binary format. However, a Python module, statepoint.py, is available that makes it easy and intuitive to extract and visualize tally results. In addition, a graphical user interface built on top of the statepoint.py module and PyQt provides mesh tally plotting. Figure 7 shows a screenshot of the PyQt mesh tally plotting application. Alternatively, a separate Python utility is provided to process mesh tallies from statepoints into VTK or SILO files. For example, Figure 8 shows a 3D visualization of a mesh tally in ParaView.



Tally: 1
Basis: xy
Axial Level: 18
Filter options:
Energy in: 0.0 to 6.25e-07
Score: flux

Figure 7: Example of PyQt mesh tally plotting application displaying a radial flux distribution.

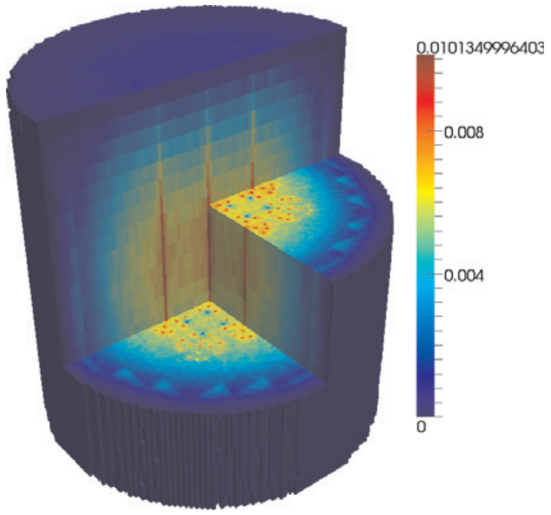


Figure 8: Example of a 3D visualization of a mesh tally using ParaView.

State point files (and other binary files) can be written either in a raw binary format or in HDF5 format.[32] While the HDF5 format should be preferred and ensures portability across different architectures, the former is made available to ensure that users can take advantage of state point capabilities even on systems where HDF5 is unavailable.

One recent capability added to OpenMC is the ability to create **particle restart files**. These files contain a particle's attributes at birth, as well as the random number seed used to start its history, and are created whenever OpenMC hits a "fatal error" related to geometry tracking causing it to abort. This allows the particle history to be simulated in order to better determine what caused OpenMC to abort unexpectedly.

3. Documentation

Extensive documentation for the OpenMC code is available online at `http://mit-crpg.github.io/openmc`. The documentation consists of the following major pieces:

- **Release notes** describing what changes were made between subsequent releases, what new features were introduced, what bugs were fixed, and what platforms the code is known to run on.
- A **theory and methodology** section describing in detail the algorithms used for geometry, cross sections, random number generation, physics, tallies, k-eigenvalue calculations, and parallelization in OpenMC including derivations of key equations. This section contains a wealth of information and should be the first stop for anyone wondering how the code works.
- A **user's guide** that describes how to build and install OpenMC, how to write XML input files and the various elements available, how to post-process results and visualize data, and a troubleshooting guide.
- A **developer's guide** that contains a description of important data structures and variables, a style guide for developers active on the master branch OpenMC, and binary file format specifications.

In addition to the online documentation, users and developers can discuss various issues on separate Google Groups mailing lists.

All documentation for OpenMC is written in a markup language called reStructuredText. This markup format can be parsed and translated by Sphinx[33] to produce documentation as a website (.html), a PDF, or various other formats.

4. Version Control and Workflow

All version control of OpenMC and its documentation is handled through the git distributed revision control system. In addition to git, the web-based hosting service GitHub is used to provide a central host, issue/milestone tracking, workflow control via pull requests, a wiki, and documentation hosting. The combination of git and GitHub enables developers to maintain high productivity in collaborating with one another, testing out new ideas, and documenting their work.

Active development on OpenMC is conducted using an integration-manager workflow as described by Chacon.[34] This workflow works particularly well with the GitHub hosting service that is used by OpenMC. On GitHub, each developer can easily fork a project creating their own public copy of the OpenMC repository. They are then free to make whatever changes and modifications they wish and are not required to have any special access on the original repository. If a developer wants their changes to be merged into the official project, they can issue a *pull request* — at this point, the person designated as the integration manager then reviews the request and, if the changes are acceptable, merges it in. Figure 9 illustrates the general integration-manager workflow.

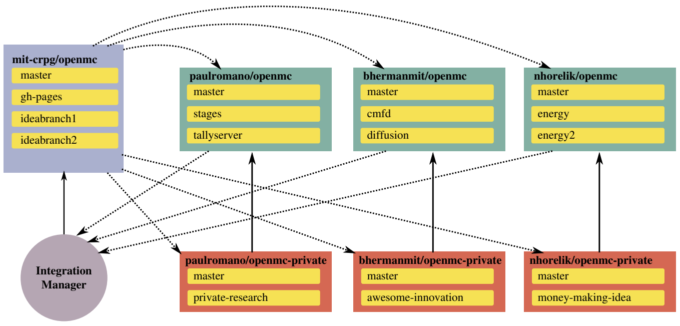


Figure 9: Integration manager workflow pattern used in OpenMC development.

IV. Licensing

OpenMC is licensed under the MIT/X open source license. This permissive license allows any user to copy, modify, redistribute, and even sell the software if they so wish. Unlike copyleft licenses such as the GNU General Public License, it does not require that modifications to the code be released under the same license, and thus commercial entities are free to use any part of the code within their own proprietary software without having to release it for free.

V. Conclusions

OpenMC has been developed from scratch with a focus on high-performance algorithms and modern software development practices. While the code is relatively young, it is already being used in a number of advanced R&D projects including the Consortium for Advanced Simulation of LWRs and the ANL Center for Exascale Simulation of Advanced Reactors. OpenMC is available as free software under an open source license, enabling wider collaboration within the nuclear science and engineering community.

Acknowledgments

This research was performed under appointment of the first and third authors to the Rickover Fellowship Program in Nuclear Engineering sponsored by Naval Reactors Division of the U.S. Department of Energy. This work was also supported in part by the Consortium for Advanced Simulation of Light Water Reactors, an Energy Innovation Hub for Modeling and Simulation of Nuclear Reactors under U.S. Department of Energy Contract No. DE-AC05-00OR22725, and by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

1) F. B. Brown, B. Kiedrowski, and J. Bull, "MCNP5-1.60 Release Notes," LA-UR-10-06235, Los Alamos National Laboratory (2010).
2) C. M. Diop et al., "TRIPOLI-4: A 3D Continuous-Energy Monte Carlo Transport Code," *Proc. PHYTRA1: First International Conference on Physics and Technology of Reactors and Applications*, Marrakech, Morocco, Mar. 14–16, 2007.
3) P. K. Romano et al., "Progress and Status of the OpenMC Monte Carlo Code," *Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, Idaho, May 5–9, 2013.
4) A. G. Nelson and W. R. Martin, "Improved Convergence of Monte Carlo Generated Multi-Group Scattering Moments," *Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, Idaho, May 5–9, 2013.
5) A. R. Siegel, K. Smith, P. K. Romano, B. Forget, and K. Felker, "The effect of load imbalances on the performance of Monte Carlo codes in LWR analysis," *J. Comput. Phys.*, **235**, 901–911 (2013), doi:10.1016/j.jcp.2012.06.012.
6) A. R. Siegel, K. Smith, P. K. Romano, B. Forget, and K. Felker, "Multi-core performance studies of a Monte Carlo neutron transport code," *Int. J. High Perform. Comput. Appl.* (2013), Accepted.
7) P. K. Romano and B. Forget, "The OpenMC Monte Carlo Particle Transport Code," *Ann. Nucl. Energy*, **51**, 274–281 (2013), doi:10.1016/j.anucene.2012.06.040.
8) X-5 Monte Carlo Team, "MCNP - A General Monte Carlo N-Particle Transport Code, Version 5, Volume III: Developer's Guide," LA-CP-03-0284, Los Alamos National Laboratory (2008).
9) J. Leppänen, "Serpent — a Continuous-energy Monte Carlo Reactor Physics Burnup Calculation Code, User's Manual," VTT Technical Research Centre of Finland (2012).
10) R. E. MacFarlane, D. W. Muir, R. M. Boicourt, and A. C. Kahler, "The NJOY Nuclear Data Processing System, Version 2012," LA-UR-12-27079, Los Alamos National Laboratory (2012).
11) J. Leppänen, "Two practical methods for unionized energy grid construction in continuous-energy Monte Carlo neutron transport calculation," *Ann. Nucl. Energy*, **36**, 878–885 (2009).
12) T. M. Sutton, T. H. Trumbull, and C. R. Lubitz, "Comparison of Some Monte Carlo Models for Bound Hydrogen Scattering," *Proc. Int. Conf. Mathematics, Computational Methods, and Reactor Physics*, Saratoga Springs, New York, May 3–7, 2009.
13) L. B. Levitt, "The Probability Table Method for Treating Unresolved Neutron Resonances in Monte Carlo Calculations," *Nucl. Sci. Eng.*, **49**, 450–457 (1972).
14) J. Lieberoth, "A Monte Carlo Technique to Solve the Static Eigenvalue Problem of the Boltzmann Transport Equation," *Nukleonik*, **11**, *5*, 213–219 (1968).
15) D. J. Kelly, T. M. Sutton, and S. C. Wilson, "MC21 Analysis of the Nuclear Energy Agency Monte Carlo Performance Benchmark Problem," *Proc. PHYSOR – Advances in Reactor Physics – Linking Research, Industry, and Education*, Knoxville, Tennessee, Apr. 15–20, 2012.
16) A. H. Squillicote, *The ParaView Guide: A Parallel Visualization Application*, Kitware, Inc. (2007).
17) VisIt Development Team, "VisIt User's Manual, Version 1.5," UCRL-SM-200449, Lawrence Livermore National Laboratory (2005).
18) Silo Development Team, "Silo User's Guide for Version 4.8," LLNL-SM-453191, Lawrence Livermore National Laboratory (2010).
19) Kitware, Inc., *VTK User's Guide*, 11th edition, (2010).
20) N. Horelik, B. Herman, B. Forget, and K. Smith, "Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS)," *Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, Idaho, May 5–9, 2013.
21) T. M. Sutton et al., "The MC21 Monte Carlo Transport Code," *Proc. Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications*, Monterey, California, Apr. 15–19, 2007.
22) P. K. Romano, "The OpenMC Monte Carlo Code — OpenMC Documentation," `http://mit-crpg.github.io/openmc`, accessed April 27, 2013.
23) D. V. Veen and J. E. Hoogenboom, "Efficiency Improvement of Local Power Estimation in the General Purpose Monte Carlo Code MCNP," *Progress in Nuclear Science and Technology*, **2**, 866-871 (2011).
24) P. K. Romano and B. Forget, "Parallel Fission Bank Algorithms in Monte Carlo Criticality Calculations," *Nucl. Sci. Eng.*, **170**, *2*, 125–135 (2012).
25) P. K. Romano and B. Forget, "Reducing Parallel Communication in Monte Carlo Simulations via Batch Statistics," *Trans. Am. Nucl. Soc.*, **107**, 519–522 (2012).
26) M. J. Lee, H. G. Joo, D. Lee, and K. Smith, "Monte Carlo Reactor Calculation with Substantially Reduced Number of Cycles," *Proc. PHYSOR – Advances in Reactor Physics – Linking Research, Industry, and Education*, Knoxville, Tennessee, Apr. 15–20, 2012.
27) S. Balay et al., "PETSc Web page," `http://www.mcs.anl.gov/petsc`, 2013, (accessed April 29, 2013).
28) A. Markus, "XML-Fortran Page," `http://xml-fortran.sourceforge.net`, accessed April 23, 2013.
29) NEA Nuclear Science Committee, "International Handbook of Evaluated Criticality Safety Benchmark Experiments," NEA/NSC/DOC(95)03, OECD Nuclear Energy Agency (2012).
30) ISO/IEC JTC1/SC34, "Information technology – Document

Schema Definition Language (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG," ISO/IEC 19757-2:2008, International Organization for Standardization (2008).

31) Thai Open Source Software Center Ltd., "Jing - A RELAX NG validator in Java," `http://www.thaiopensource.com/relaxng/jing.html`, 2012, (accessed April 29, 2013).
32) S. Koranne, "Hierarchical Data Format 5: HDF5," in *Handbook of Open Source Tools*, p. 191–200, Springer US, 2011.
33) G. Brandl, "Sphinx Python Documentation Generator," `http://sphinx-doc.org`, 2013, (accessed April 29, 2013).
34) S. Chacon, *Pro Git*, Apress, Berkeley, California (2009).