LA-UR-17-28985
Approved for public release; distribution is unlimited.

Title: Status Report on the MCNP 2020 Initiative

Author(s): Brown, Forrest B.
Rising, Michael Evan

Intended for: MCNP information

Issued: 2017-10-02

Status Report on the MCNP 2020 Initiative

Forrest B. Brown & Michael E. Rising

Monte Carlo Methods, Codes, & Applications Group (XCP-3)

X Computational Physics Division, LANL

2017-09-25

I. Introduction

The discussion below provides a status report on the MCNP 2020 initiative. It includes discussion of the history of MCNP 2020, accomplishments during 2013-17, priorities for near-term development, other related efforts, a brief summary, and a list of references for the plans and work accomplished.

II. History

The year 2017 is the 70th anniversary of the first Monte Carlo (MC) computer code for radiation transport written by John von Neumann for LANL and the ENIAC computer, and the 40th anniversary of the first release of the MCNP MC code. MCNP was a merger of several smaller MC codes from LANL, derived from von Neumann's seminal work and made possible by the greatly increased speed and memory size of the 1970s computers. Why does this matter? Computer architecture, performance, and programming have changed drastically over the past 70 years; the overall structure, algorithms, and programming of MCNP have not.

Over the past 40 years, 100s of new "features" were added to MCNP, largely by kludging the features on top of the existing code base. During 2001-2002, the code was overhauled to conform to the Fortran-90 coding standards of the time. No changes were made to the fundamental code data structures and algorithms, except that full support for multi-level parallelism (multicore threading plus MPI) was provided. A detailed review of the MCNP coding today reveals a Rube Goldberg approach - an incredibly powerful, accurate, multi-featured, mature MC code - that is extremely fragile, hard to maintain, and difficult to adapt to the rapid changes occurring in computer architecture.

In 2013, the MCNP 2020 initiative [1-5] was proposed to address the many years of neglecting improvements to code infrastructure and algorithms. The proposal was founded on experience based on a 100% success rate for 5 similar previous endeavors. The key elements of the MCNP-2020 proposal included:

1. **Improve code performance**

   With a faster MC code, analysts can run more calculations in a given amount of time, leading directly to improved quality and realism.

   Goal: Provide a 2x speedup within 2 years.

2. **Upgrade core MCNP6 software**

   Review and upgrade the most-used (core) coding in MCNP6. This includes efforts to: restructure the core coding; clean up the algorithms and coding syntax; comply with Fortran and C++ coding standards; reorganize data structures; and more.

   This is to be achieved by evolution, not revolution. That is, a major rewrite-from-scratch effort is not feasible for a large complex code such as MCNP6; instead many incremental changes and structural changes are needed to make progress while retaining all code capabilities.

   Goal: Produce a sustainable code, with reduced costs for future development and maintenance.

3. **Prepare for the future**

   The next generations of advanced computers will have massive parallelism, with millions of *cpu*-cores. It is expected, however, that the available memory per *cpu*-core will not increase and may even decrease. Accordingly, significant effort must be applied to improving the parallel computing algorithms used in MCNP, including both MPI message-passing and OpenMP threading. While MCNP6 currently uses both MPI and threading, the algorithms were targeted to 100s or 1000s of *cpu*-cores, not millions.

   In addition, future systems will be heterogeneous, with a mix of *cpu*s, *gpu*s, *mic*s (many integrated cores), reconfigurable *fpga*s (field programmable gate arrays), and graphics processors.

   Goal: Produce more flexible and adaptable code, with capabilities for massive parallelism and heterogeneous processing.

Full support for the MCNP 2020 effort would require a full-time dedicated team of 3-4 highly computer-literate nuclear engineers or physicists devoted not to "feature" development, but to overhauling the code algorithms, data structures, parallelism, and more for the base code structure. Code modernization would provide a robust, maintainable, and extensible code base that could be readily adapted to the coming exascale systems and permit faster integration of new "features."

III. Accomplishments, 2013-17

Since 2013, the DOE-NNSA Nuclear Criticality Safety Program (NCSP) has enthusiastically supported the MCNP 2020 initiative; other MCNP sponsors have provided only occasional minor support. Because of this, only a few of the proposed tasks have been accomplished, with 100% success, and those tasks are closely tied to NCSP needs [6-9]:

1. MCNP6 performance improvements

In 2013-14, an intense effort was made to speed up the code [10-12]. For single-thread performance, many classic code optimization techniques were applied. In addition, a dramatic speed increase came from the development of a novel hash-based energy lookup scheme coupled with inline binary searches (both had previously consumed 35-50% of overall *cpu* time). Parallel threading was also somewhat improved. The result of these optimizations was roughly a 2x speedup for nuclear criticality safety problems. The resultant code was released to users as MCNP6.1.1 in 2014. The goal for this initial portion of MCNP 2020 was met.

2. Fortran-2003 standards compliance

Beginning in the summer of 2014, discussions with the MCNP development team lead to a team-wide effort to upgrade the MCNP6 Fortran coding to fully comply with the Fortran-2003 International Standard. At that time, there were over 3,000 instances of nonstandard coding involving 100s of source files. All team members reviewed sections of the coding that they were most familiar with and made any coding changes necessary to comply with the Fortran-2003 standard. Note that these changes were in coding syntax only, not algorithms. This was considered a low-priority background task, and was completed in 2015. Over 500k lines of Fortran coding were reviewed and brought into compliance. Since 2015, MCNP6 has been 100% compliant with the Fortran-2003 standard. In addition, the current procedures for building the code enforce this requirement, checking for standards-compliance every time anyone compiles any version of MCNP6. This task was a small part of the "upgrade core MCNP6 software" portion of MCNP 2020, and was entirely successful.

3. Parallel algorithm improvements

A number of minor improvements in parallel algorithms were made to MCNP6 over the past few years. One algorithm improvement led to the elimination of a parallel bottleneck for large criticality calculations – reordering the fission bank between cycles. Other minor improvements to OpenMP threading were also made for the range of 100-1000 *cpu*-cores. MCNP6 was also tested on the Intel Xeon Phi *mic* processor, where 240 threads/processor are needed for effective use.

While these minor efforts were successful, the impact was low for current computing systems. The payoff from these improvements will be apparent over the next generations of computers. The effort in this area will of course continue.

4. Modernization

With the upgrade of MCNP6 to Fortran-2003 standards compliance, the decision was made to only allow Fortran compilers that fully support that standard (e.g., Intel, gfortran). Older Fortran compliers that don't support Fortran-2003 cannot be used for compiling MCNP6. With that requirement in place, a number of Fortran features that were avoided in the past can now be safely used. The most significant of these is polymorphism. Previously, when a routine could be invoked with different types of arguments, separate copies of the routine were needed – one for each type of argument. This led to the replication of large blocks of coding, creating an error-prone situation where any changes needed to be replicated in many different sections of coding. Using Fortran-2003 polymorphism, all of the instances can be combined, reducing duplication and tedious corrections. This capability is being used in a few portions of MCNP6 today, with more planned as needed.

IV. Priorities for Near-Term Development

Most of the current development activities by the MCNP developers are driven by programmatic needs at LANL, by programmatic needs of MCNP sponsoring agencies, and by the needs of the MCNP user community. The discussion that follows is separate from those application needs and only addresses activities and priorities for tasks related directly to the MCNP 2020 initiative (i.e., performance, code and data structures, modernization, and preparing for future computers).

A. List-Tallies and Tally-Servers

From a high-level viewpoint, the most important, critical need in MC code advancement is to decouple the compact *cpu*-intensive random walk simulation from the huge memory address space required for today's detailed tallies (i.e., results, especially large mesh tallies). The MCNP 2020 proposal included tasks for "list-tallies" and "tally-servers" to address this need. The coming exascale systems will not work effectively for MC unless these tasks are completed.

Brown proposed the idea of list-tallies and tally-servers as a PhD thesis topic for an MIT graduate student in 2009 [13]. The method was found to work effectively, enabling very large mesh tallies. A number of other MC researchers around the world have also begun to adopt the list-tally + tally-server approach to handle large-scale MC problems.

The basic idea is that near-term advanced computers may have millions of *cpu*-cores, but limited memory per core. Simulating the random walks in a MC calculation requires only modest storage for geometry and nuclear data, so that the basic random walk simulation could readily be replicated millions of times. However, current MC codes couple the simulation with a potentially very large storage space for the tally data. In current usage of MC codes, mesh tallies can require 100s or 1000s of GBs of storage, which cannot be replicated across millions of *cpu*-cores. While tally storage can of course be shared, the overhead from lock-contention (to preserve correctness of the tallies) would almost certainly be excessive and prohibitive when millions of *cpu*s are targeting the same tally address space. In addition, communication delays for non-local memory access could be significant. The natural solution to this problem is to return to the roots of MC – perform the simulation on millions of *cpu*s, each producing list-tally information that would be collected and processed elsewhere. Only 2 items are required per tally – the index of a tally bin and the value to be added, so that the list-tally information is very compact. Tally-servers could collect the lists and then process tallies efficiently (perhaps in segments, tiles, or blocks) using vector operation on *cpu*s or *gpu*s. This approach separates the compute-intensive simulation with modest address space from the simpler processing of tallies with huge address space. It also enables an asynchronous heterogeneous processing approach with *cpu*s or *mic*s for the simulation, and *cpu*s or *mic*s or *gpu*s for the tally processing.

Since the proposal for the MCNP 2020 initiative, additional consideration and planning has suggested that the following sequence be followed in implementing the list-tally + tally-server approach:

1. Apply the methodology to a single (but important) feature in MCNP, the PTRAC feature (discussed next). It is very straightforward to apply the list-tally approach to PTRAC, buffering lists of event information and then flushing to an output file later. Experience gained in this first straightforward application would ease the adoption for more complicated tally features. This would also facilitate PTRAC for parallel calculations (currently prohibited).
2. Apply the methodology to mesh tallies. This would have an immediate impact on some of the very large memory problems that analysts struggle with today. The approach would be staged: list-tallies without a separate tally-server, list tallies with a single tally-server, and finally list-tallies with multiple tally-servers and routing.
3. Apply the methodology to all other standard MCNP tallies. This could be challenging, due to the very general tally capabilities with many special cases.

B. Parallel PTRAC

The PTRAC feature in MCNP provides the ability to produce a log of the detailed physics and geometry information for each event that occurs in the simulation of an individual particle. Essentially, all of the detailed steps in the simulation produce output that can be post-processed by some other program or script. Normally this

kind of detailed event-by-event output for every particle is suppressed to avoid the huge amount of data produced. Instead, MCNP internally uses this information to produce the normal tally output. When users need to manipulate this data, typically to perform certain tallies that cannot be handled by standard MCNP features, PTRAC is invoked to record the detailed simulation data for post-processing. (As a historical note, nearly all of the original MC codes in the 1950-60s worked this way. The MC codes did only the simulation, dumping all event information to a "runtape." Other codes read the "runtape" and then performed the tally operations for results. This approach was needed due to the small computer memories of the time.)

Since the release of MCNP5 in the early 2000s, the PTRAC feature could not be used in parallel calculations; long-running sequential execution was required. This situation has continued through the current MCNP6.2. In the past few years, the use of PTRAC has increased due to homeland-security-related needs for detector analysis and the production of "list-mode" data for simulations of critical experiments. (The list-mode data from a simulation is produced in the same format as the data obtained from detectors in the experiments, and is processed identical to the experiment data.)

While there are a number of straightforward ways to modify the PTRAC scheme to permit parallel calculations, parallel PTRAC provides a very timely application of the list-tally approach mentioned in MCNP 2020. Lessons-learned from this effort should benefit the subsequent application of list-tallies + tally-servers to other type of MCNP tallies.

C. Memory Management and Generalized Stacks

A very large amount of MCNP coding is devoted to processing problem input, determining the memory required for each data array in the problem (depending on input options, some arrays are not needed, others are), and then allocating the memory. There are many 1000s of lines of extremely ugly and confusing coding devoted to these tasks, and the coding is very prone to inadvertent coding errors. In addition, the coding is nearly impossible to review and check, and is a significant bottleneck for new developers or even experienced developers in adding any new features. Nearly all of the current difficulties could be resolved by implementing a memory manager. The memory manager could keep track of which arrays were allocated along with the size and dimensions, could perform the needed error-checking and provide consistent error messages, and could be used to facilitate a new generalized direct-access restart file (i.e., dumpfile, runtpe). An initial version of an MCNP memory manager was prototyped over the last year or so, using Fortran-2003 polymorphism. It was found to greatly simplify, streamline, and clarify the memory allocation in MCNP6 where it was applied. This prototype could be readily implemented everywhere with modest effort.

Currently some data arrays are allocated with an arbitrary size, elements are added, and then if the array fills up it is reallocated and copied from the old to new space. This approach works, but is error-prone, inefficient, and difficult for developers to

work with. A much better approach is to implement a general stack-based scheme with linked-list storage. This would make use of the Fortran-2003 polymorphism to handle any type of array or complex data objects. Such a unified approach to expandable memory storage would simplify dozens of different areas in the MCNP coding, greatly ease future development, and conserve memory size for a problem. The mechanism could also be used in implementing the list-tally approach, since the size of the list-tally arrays is not known in advance.

D. Explicit Threading Loop and Dispatch Algorithm

The current OpenMP threading approach in MCNP6 essentially starts the requested number of threads in concurrent infinite loops. Each thread locks certain variables, checks to see if another particle should be run, grabs the particle number, increments some global control variables, then unlocks memory and runs that particle history. The algorithm is essentially concurrent infinite loops with exit points. A better approach is a single dispatch loop over defined particle numbers, with each iteration of the loop handled by a different thread. This would introduce some predictability for enabling fault-tolerance, permit more control over binding particles to threads, permit options on thread sequencing, and eliminate some of the current lock/unlock overhead. This may be necessary for scaling up from 1000s of threads to millions of threads in the future.

E. Storage Changes for Improved Cache Usage

For historical reasons, most of the data storage in MCNP6 is based on a structure-of-arrays (SOA) approach. That is, for cell-related data, there is an array of the material in each cell, an array of the temperature of each cell, an array of indexes for the isotope-lists for each cell, an array of flags for treating fission in each cell, etc., etc. This approach was required until the 2000s, due to the lack of features in old-style Fortran for defining objects (i.e., defined-types in Fortran, structures or classes in C++). When a particle is in a cell, information must be fetched separately from arbitrarily-located data arrays (e.g., material, temperature, index for isotope-list, etc.). Due to the essentially random memory layout, each of the memory accesses leads to a cache-miss and very much longer access times. About 70% of MCNP operations during simulation are in this state, with only about 30% or less involved in arithmetic.

Today Fortran-2003 permits the definition of objects (or structures or defined types), where all of the properties may be group together. This is termed an array-of-structures (AOS) approach. For the cell example, there could be a “cell object” which would contain the material, temperature, etc., for a cell, and there would be an array of these objects. When a particle enters a cell, the cell object would be referenced, fetching all of the relevant cell properties together. This could greatly reduce cache-misses and cache-contention, especially for threaded calculations.

Conversion of MCNP6 from an SOA approach to an AOS approach should be done gradually and methodically, with performance monitored at each incremental step. Relevant data include cells, surfaces, materials, and other quantities.

F. Direct-Access Run-tape File

To provide some fault-tolerance for computer system crashes during long running MC calculations, MCNP has always relied on the simple, time-tested approach of periodically writing all current data to a "runtpe" file (also commonly called a run-tape file, dump file, or restart file). Unfortunately, there has never been a standard format for the MCNP run-tape file. All data and arrays are written sequentially to the file with explicit sequential-access output statements, and the file is read for subsequent restarts sequentially, array by array with explicit input statements. Adding even 1 extra variable to the data saved on the run-tape makes the file unreadable by previous versions of MCNP. In addition, to retrieve even 1 number from a run-tape requires reading the entire file in exactly the same order as it was written. All of these inconveniences become serious problems for many current calculations where run-tapes can be many GBs or 100s of GBs in size.

It is straightforward to define a standard format for the run-tape file, using named sections of data, a direct-access format, and a table of contents for directly locating data by name. Single data items or arrays could then be quickly and easily retrieved (without sequentially reading the entire file). Run-tapes could be interchanged among different versions of MCNP (as long as the versions supported the features actually used in a calculation). Many other MC codes have well-defined run-tape formats, named data sections, and direct-access capabilities.

V. Other Related Work

The XCP-3 Group management announced in September 2017 that a new initiative will begin to modernize the XCP-3 MC codes. This will be supported by LANL institutional funding (through PADWP and PADGS) and by some funding from the DOE-NNSA-ASC Program, totaling about 4 FTE per year for the next 3-4 years. The principle goal for this modernization is to share common MC components among MCNP6, the MCATK toolkit, and other MC codes supported by the group, in order to reduce development effort, share best coding and algorithms, reduce maintenance costs, and fully comply with DOE-NNSA software quality assurance (SQA) requirements (DOE Order 414.1D, LANL Policy P1040_Rev9). To avoid duplication of effort, a common library or framework of routines will be assembled and shared among the various MC codes. These common routines are also required to include documentation, unit test routines, and verification-validation results where appropriate. This modernization effort is in the planning stages now, with work expected to begin in 2018.

This effort is focused on all of the MC codes supported by XCP-3. This important work is neither a replacement for nor an alternative to the ideas included in the

MCNP 2020 initiative. Both efforts are needed – one provides a common support platform and compliance with SQA requirements for the suite of XCP-3 MC codes; the other provides guidance and improvements to the detailed MC coding, data structures, and algorithms. Coordination is necessary of course.

VI. Summary

The discussion above has provided background, recent accomplishments, and high-level plans for near-term development related to the MCNP 2020 initiative. The tasks accomplished to date have been successful, but of course much more work is needed. Many of the tasks carried out in 2016-17 were prototype efforts, to be fully included into MCNP6 after the final release of MCNP6.2 (expected October 2017). It is expected that this work will continue indefinitely at a low-to-moderate level of effort.

References

1. F.B. Brown, B.C. Kiedrowski, J.S. Bull, L.J. Cox, “MCNP 2020 – Preparing LANL Monte Carlo for Exascale Computer Systems,”, white paper from 2013, LA-UR-15-22524 (2015)
2. F.B. Brown, “MCNP 2020 – An initiative to preserve 70 years of LANL investment in Monte Carlo radiation transport & prepare for future computer systems,” presentation, LA-UR-15-22523 (2015).
3. F.B. Brown, “Recent Advances and Future Prospects for Monte Carlo”, *Progress in Nuclear Science & Technology* , Vol. 2 [also LA-UR-10-05634] (2011).
4. F.B. Brown, “Theory & Practice of Criticality Calculations with MCNP5,” class notes, LA-UR-08-0849 (2008).
5. Forrest Brown, “Fundamentals of Monte Carlo Particle Transport”, lecture notes for LANL class, LA-UR-05-4983, (2005).
6. F.B. Brown, B.C. Kiedrowski, J.S. Bull, "MCNP Progress & Performance Improvements", talk at DOE NCSP Technical Program Review, LANL, March 2015, LA-UR-14-21786 (2014)
7. F.B. Brown, J.S. Bull, M.E. Rising, "MCNP Progress & Performance Improvements", talk at DOE NCSP Technical Program Review, LLNL, March 2015, LA-UR-15-21869 (2015)
8. F.B. Brown, M.E. Rising, J.L. Alwin, "MCNP Progress for NCSP", presentation at DOE NCSP Technical Program Review, Sandia National Laboratory, March 15-16, 2016, LA-UR-16-21302 (2016)
9. F.B. Brown, M.E. Rising, J.L. Alwin, "MCNP Progress for NCSP", presentation at DOE NCSP Technical Program Review, Washington DC, March 14, 2017, LA-UR-17-21840 (2017)
10. F.B. Brown, "MCNP6 Monte Carlo Code Optimization", Proc. ANS M&C + SNA + MC 2015, Nashville, TN, April 2015, LA-UR-15-20019 (2016)
11. F.B. Brown, "MCNP6 Optimization and Testing for Criticality Safety Calculations", Trans. ANS 112, San Antonio, TX, June 2015, LA-UR-15-20422 (2015)
12. F.B. Brown, “New Hash-based Energy Lookup Algorithm for Monte Carlo Codes”, ANS 2014 Winter Meeting, Anaheim CA, LA-UR-14-24530 (2014)
13. P.K. Romano, “Parallel Algorithms for Monte Carlo Particle Transport Simulation on Exascale Computing Architectures,” PhD thesis, Dept. Nuclear Science & Engineering, MIT (2013).