LA-UR-25-23947

Approved for public release; distribution is unlimited.

Title:	MCNP6® New Features, Algorithms, and Workflows
Author(s):	Rising, Michael Evan Sood, Avneet Vaquer, Pablo Andres Weaver, Colin Andrew
Intended for:	Mathematics & Computation (M&C) 2025, 2025-04-27/2025-04-30 (Denver, Colorado, UNITED STATES)
Issued:	2025-04-29 (rev.1)

Los Alamos NATIONAL LABORATORY

MCNP® MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-01: MCNP6.3 and MCNP6.3.1 code, algorithm, and tool improvements

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

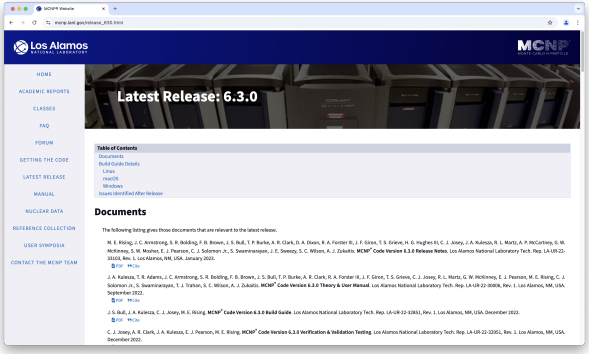
MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

MCNP6.3 Release

- See the MCNP6.3 release webpage on the MCNP website
- The primary documents supporting the release are:
  - Release Notes [1],
  - Theory & User Manual [2],
  - Build Guide [3], and
  - Verification and Validation Testing [4]



MCNP6.3 New Features

MCNP
MONTE CARLO N-PARTICLE

- BURN: added option to reduce memory via DISABLE card
- DBRC: added Doppler broadening resonance correction & utility code
- FMESH: added HDF5+XDMF output capability
- FMESH: added `colsci` and `cfsci` output formats
- FMESH: added new tallying backends, particularly for extreme scaling
- KCODE: added convergence acceleration and convergence detection
- PTRAC: added HDF5-formatted output (with parallel-execute) capability
- Unstructured mesh: added HDF5+XDMF output capability
- Unstructured mesh: added element quality metric reporting
- Added Qt-based plotter as a “technology preview” [5]
- Added mixed-material treatments for structured-mesh tracking
- Added stochastic temperature mixing of $S(\alpha, \beta)$ data
- Converted unformatted Fortran binary runtape to HDF5 format
- Moved from a bespoke build system to a CMake-based one

MCNP6.3 New Tally Options

MCNP
MONTE CARLO N-PARTICLE

- Added special tally treatments for reactor analysis (FT card)
  - FNS: induced fission neutron spectra
  - LCS: Legendre coefficients for scatter reactions
  - MGC: flux-weighted multigroup cross sections
  - SPM: collision exit energy-angle scatter probability matrix
  - More information in [6, 7]

MCNP6.3 Capability Enhancements

MCNP
MONTE CARLO N-PARTICLE

- Added preliminary support for upcoming ENDF/B-VIII.1 data changes
  - New $S(\alpha, \beta)$ format options
  - Photonuclear physics
  - But, more work underway for MCNP6.3.1...
- Converted MCNP to meet Fortran 2008 and C++17 standard requirements
  - Removed `common`, `equivalence`, `forall`, and `numbered do` statements
- Decreased delayed-gamma-line memory usage
- Deprecated various features with plans for future removal
- Implemented several modernization efforts: naming, reducing global state, etc.
- Improved informational and reference items
  - Manual, automated V&V report, website, forum, user symposia
- Open-source release of MCNPTools v5.3.1 via GitHub [8]
- Updated ISC v2.1.0 to include ENDF/B-VIII.0 data [9, 10]
- Updated to use open-source version of CGMF v1.1.1 [11]

MCNP6.3 Performance Enhancements

MCNP
MONTE CARLO N-PARTICLE

- Decreased memory and improved performance of ACT `DG = line` option
- Improved cross section cache array (re)initialization behavior
- Improved unstructured mesh input file processing performance: $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n)$
- Reduced UM memory requirements, but more work to be done

MCNP6.3.1 Fixes and New Features

- Primarily a bug-fix release
  - Lots of changes for MCNP6.3—what bumps need to be smoothed?
  - Fix `cindergl.dat` gamma-line formatting (incorrect ACT card results)
    - New file is `cindergl_v3.dat`
  - Improved next-event estimator neutron inelastic scattering contributions [12]
- New features
  - New, optional, SFC64-based (pseudo)random number generator (RNG) [13]
    - Available as `gen=8` on the RAND card
    - Expect this to become default RNG in future version of MCNP6
  - Cross-section table identifiers rather than ZAIDs
    - "Plain English" ZAs with extended library IDs
    - `U-238.Lib80x-293.6K.c` versus `92238.00c`
  - New (α, n) utility, `mesa`, built from updated ISC-3.0.0 utility [14]

MCNP6.3.1 Improvements, V&V, and Data Updates

MCNP MONTE CARLO N-PARTICLE

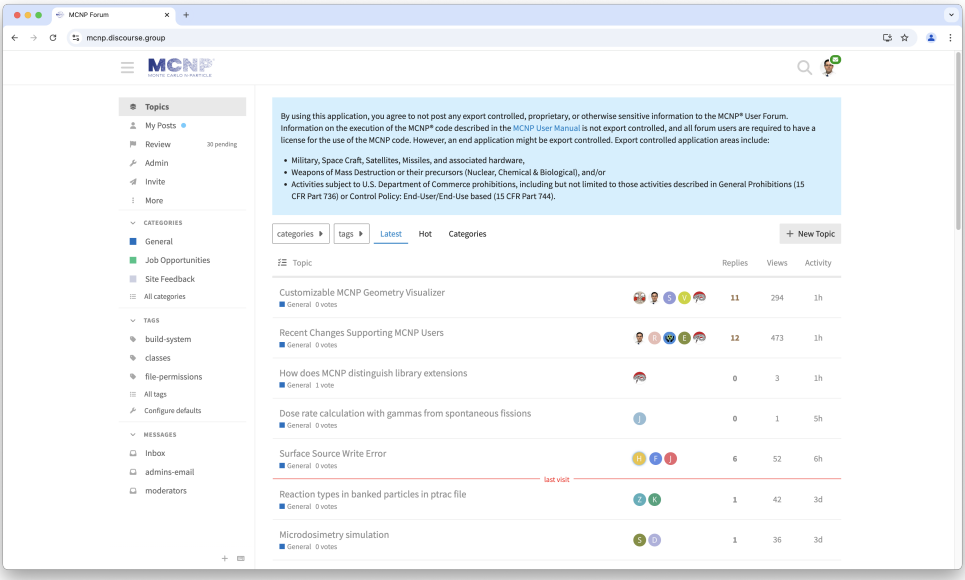
- General improvements
  - Updates to Qt-based Plotter Technology Preview
    - Supports Qt6, improved fonts, GUI improvements, etc.
    - The Qt plotter will replace legacy X11 plotter in MCNP6.4
  - New electron stopping power validation suite
    - Added to `vnvstats` utility
    - Added section in V&V report with MCNP6.3.1 results
- Data updates
  - `nd_manager` updates and minor fixes for improved downloading of nuclear data
  - New ENDF/B-VIII.1-based DBRC data files
  - Updated ISC-3.0.0 data files for new (α, n) capabilities

MCNP6.3.1 Documentation and Resource Updates

MCNP MONTE CARLO N-PARTICLE

- Also, and always: documentation updates
  - Theory & user manual fixes [15]
  - Release notes [16]
  - Verification & validation report [17]
    - MCNP6.3.1 is a full production version
    - Same level of rigorous V&V as MCNP6.3
  - Build guide [18]
- Online resources
  - New Discourse-based MCNP Forum
  - MCNP6.3.1 release page on website (coming soon...)

New MCNP Forum (June 2024)



What's (Maybe?) Coming with MCNP6.4

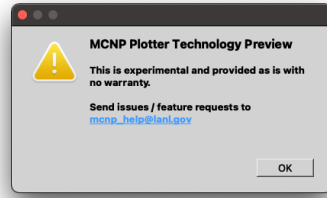
MCNP
MONTE CARLO N-PARTICLE

- New, default, SFC64-based (pseudo)random number generator [13]
- Dynamically-linked sources and tallies (cf. `source.F90` and `tallyx.F90`)
- Neutron and photon Delta tracking on CSG
- TMESH deprecation (subsumed by FMESH)
- Cinder 2025
- UM element-wise density and temperature
- HDF5-formatted k-eigenvalue source tape (SRCTP)
- Consistent and centrally specified physical constants
- New (and configurable) continuum color palette
- Ray-traced geometry visualization
- New fixed-source tally sensitivity capability
- New installable Python-based tools package

Workshop Exercises

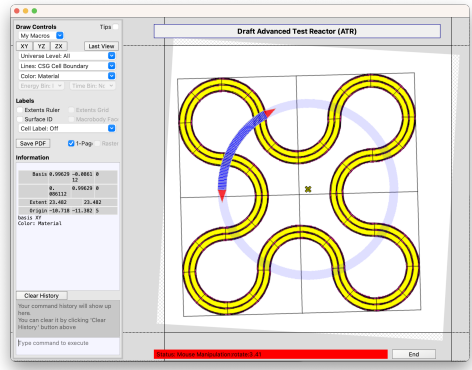
Qt Plotter Overview and Rollout

- Qt-based render window
- Modern buttons (geometry plotter)
- Command box supporting > 29 characters
- Mouse control to pan, zoom, and rotate slices
- New plotter view file formats (png, pdf)
- Macro support



Qt Plotter Release Timeline

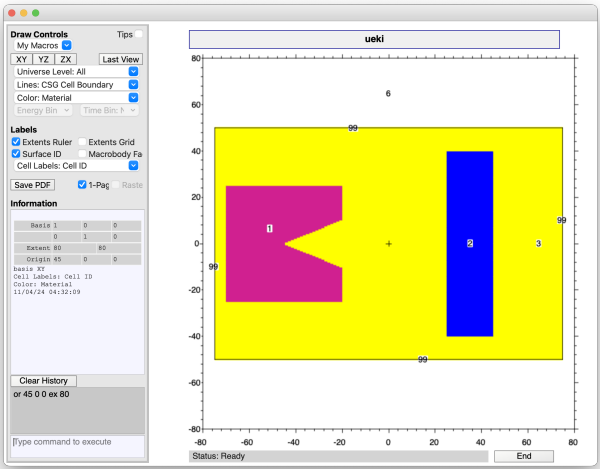
- MCNP6.3 and MCNP6.3.1: Provided as separate technology preview executable (X11 plotter in production executable)
- MCNP6.4: Replaces X11 plotter in production executable!



See https://mcnp.lanl.gov/reference_collection.html#visualization for more Qt Plotter Information and Recent Updates

Ueki Detailed UM Model

- Full detailed model with 3 parts totaling ~ 30k hexahedral elements
  - Production calculation uses 10^8 histories with weight windows
  - Exercise calculation uses 10^6 histories without weight windows



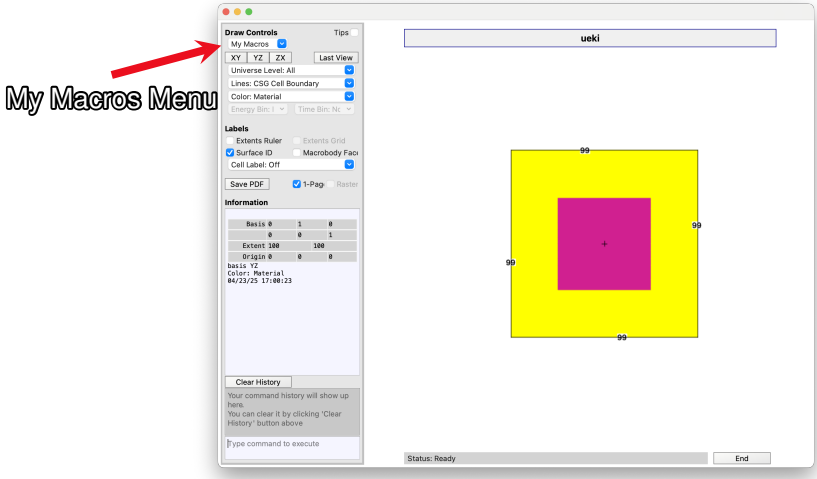
MCNP Input File Embedded as `ueki_hex_20cm_1e6.mcnp.inp.txt`
Abaqus Input File Embedded as `ueki_hex_20cm_1e6.abaq.inp.txt`
DE/DF Input File Embedded as `neutron_ansians-611-1977_dedf.txt`

Qt Plotter Macros for Ueki UM Model

Step 1: Open Ueki Model with Qt Plotter

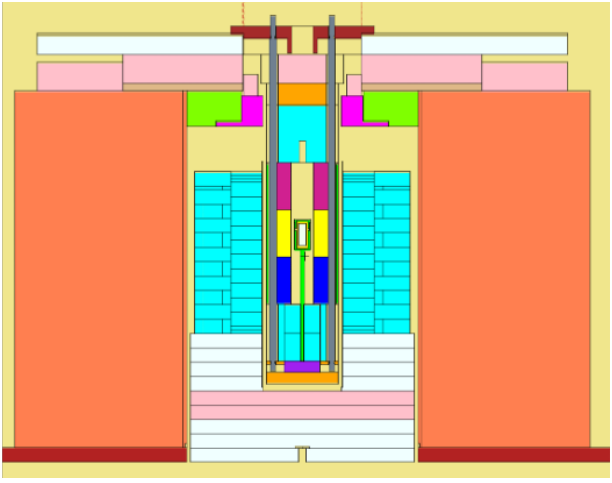
```
> mcnp6.qt i=ueki_hex_20cm_1e6.mcnp.inp ip
```

Step 2: Load `ueki_views.um` Macros



Krusty Detailed CSG Model

- Full detailed core
  - Production calculation uses 10^5 histories per cycle with 1250 cycles
  - Exercise calculation uses 10^4 histories per cycle with 150 cycles



Qt Plotter Macros for Krusty CSG Model

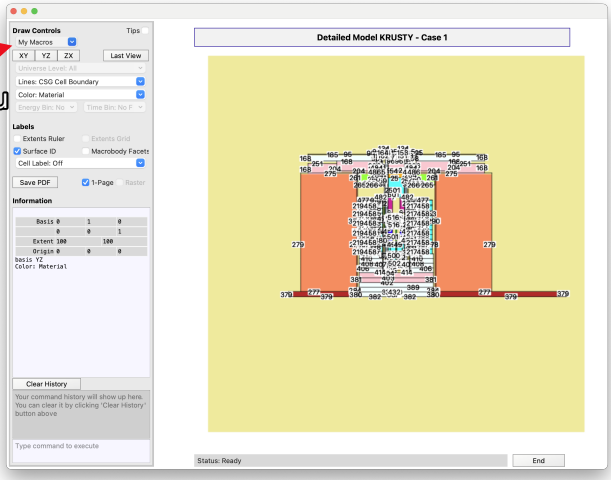
MCNP MONTE CARLO N-PARTICLE

Step 1: Open Krusty Model with Qt Plotter

```
> mcnp6.qt i=krusty_csg_detailed.mcnp.inp ip
```

Step 2: Load `krusty_views.um` Macros

My Macros Menu



Questions?

MCNP6.3 and other Resources

New Features

Enhancements
- Capability and Performance

MCNP6.3.1 Updates

What's (Maybe?) Coming with MCNP6.4

Workshop Exercises

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] M. E. Rising, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, D. A. Dixon, R. A. Forster, III, J. F. Giron, T. S. Grieve, H. G. Hughes, III, C. J. Josey, J. A. Kulesza, R. L. Martz, A. P. McCartney, G. W. McKinney, S. W. Mosher, E. J. Pearson, C. J. Solomon, Jr., S. Swaminarayan, J. E. Sweezy, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.0 Release Notes," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-33103, Rev. 1, Jan. 2023. DOI: 10.2172/1909545

[2] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.0 Theory & User Manual," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

References

[3] J. S. Bull, J. A. Kulesza, C. J. Josey, and M. E. Rising, "MCNP® Code Version 6.3.0 Build Guide," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-32851, Rev. 1, Dec. 2022. DOI: 10.2172/1906011

[4] C. J. Josey, A. R. Clark, J. A. Kulesza, E. J. Pearson, and M. E. Rising, "MCNP® Code Version 6.3.0 Verification & Validation Testing," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-32951, Rev. 1, Dec. 2022. DOI: 10.2172/1907750

[5] C. S. Frederick and S. Swaminarayan, "New Plotter in MCNP," Los Alamos National Laboratory, Los Alamos, NM, USA, Presentation LA-UR-24-28892, Aug. 2024.

[6] M. E. Rising, "Multigroup Cross-section Generation in MCNP6.3," Los Alamos National Laboratory, Los Alamos, NM, USA, Presentation LA-UR-22-30839, Rev. 1, Oct. 2022.

[7] R. B. Wilkerson, G. W. McKinney, M. E. Rising, and J. A. Kulesza, "MCNP Reactor Multigroup Tally Options Verification," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-20-27819, Oct. 2020.

References

[8] C. R. Bates, S. R. Bolding, C. J. Josey, J. A. Kulesza, C. J. Solomon, Jr., and A. J. Zukaitis, "The MCNPTools Package: Installation and Use," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-28935, Aug. 2022, MCNPTools Version 5.3. DOI: 10.2172/1884737

[9] C. J. Solomon, Jr., C. R. Bates, J. A. Kulesza, and M. J. Marcath, "The Intrinsic Source Constructor Package: Installation and Use," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-31848, Nov. 2022, ISC Version 2.1. DOI: 10.2172/1897420

[10] C. J. Solomon, Jr., C. R. Bates, and M. J. Marcath, "MCNP Intrinsic Source Constructor (MISC): A User's Guide," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-32893, Dec. 2022. DOI: 10.2172/1903539

References

[11] M. E. Rising, P. Talou, I. Stetcu, P. Jaffke, A. E. Lovell, and T. Kawano, "Open-source Release of CGMF and Integration into the MCNP6.3 Code," Los Alamos National Laboratory, Los Alamos, NM, USA, Presentation LA-UR-21-26275, Rev. 1, Aug. 2021. URL: https://www.osti.gov/biblio/1813823-open-source-release-cgmf-integration-mcnp6-code

[12] J. E. Sweezy, "Improvements to Contributions from Neutron Inelastic Scattering for Next-Event Estimators in MCNP® Software," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-23-31561, Rev. 1, Nov. 2023. DOI: 10.2172/2205018

[13] C. J. Josey, "Reassessing the MCNP Random Number Generator," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-23-25111, Rev. 1, Sep. 2023. DOI: 10.2172/1998091

[14] C. R. Bates, "MCNPs Easy Sources for (α,n) (MESA) 1.0: A User's Guide," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-24-32000, Nov. 2024. DOI: 10.2172/2476821

References

[15] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, A. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, C. A. Weaver, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.1 Theory & User Manual," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-24-24602, Rev. 1, May 2024. DOI: 10.2172/2372634

[16] M. E. Rising, J. C. Armstrong, S. R. Bolding, J. S. Bull, A. R. Clark, C. S. Frederick, J. F. Giron, T. S. Grieve, W. Haeck, F. B. Jones, C. J. Josey, J. A. Kulesza, M. A. Lively, S. Swaminarayan, J. E. Sweezy, C. A. Weaver, and A. J. Zukaitis, "MCNP® Code Version 6.3.1 Release Notes," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-25-23548, Rev. 1, Apr. 2025.

[17] C. J. Josey, A. R. Clark, J. A. Kulesza, M. A. Lively, E. J. Pearson, and M. E. Rising, "MCNP® Code Version 6.3.1 Verification & Validation Testing," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-25-22398, Rev. 1, Mar. 2025.

References

[18] J. S. Bull, C. J. Josey, J. A. Kulesza, M. E. Rising, and S. Swaminarayan, "MCNP® Code Version 6.3.1 Build Guide," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-25-22869, Rev. 1, Apr. 2025.

Los Alamos NATIONAL LABORATORY

MCNP® MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-02: Study of a New Random Number Generator in MCNP6.3.1

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

Introduction

MCNP
MONTE CARLO N-PARTICLE

The random number generators used within the MCNP code are key to the quality of our results. The current generators are well-characterized and fast.

However, they:

- fail some random number tests,
- do not have all that many states, and
- require user configuration in many circumstances to ensure quality results.

This work was an investigation on if new generators could outperform our current ones in the above aspects.

Current Generators

Linear Congruential Generators

MCNP
MONTE CARLO N-PARTICLE

All current MCNP generators take the form of a linear congruential generator (LCG):

$$X_{n+1} = (a \times X_n + c) \mod m$$

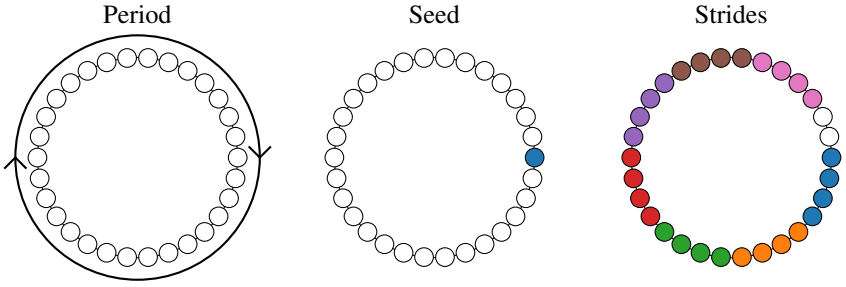
- Maximum period of m, if a and c chosen with care
- One sequence for a given a, c, m
- $\mathcal{O}(\log n)$ skip-ahead in sequence
- Quick implementation for $m = 2^p$
- Least significant bits have short period

LCG Usage in MCNP (1/2)

MCNP has 7 generators, with periods 2^{46} (gen 1), 2^{61} (gen 5–7) and 2^{63} (gen 2–4).

History initialization is performed by:

1. Setting the value X_0 to the seed
2. Skipping ahead $(\text{stride} \times i_h)$ where i_h is the history index



LCG Usage in MCNP (2/2)

MCNP
MONTE CARLO N-PARTICLE

Benefits:

- One seed changes all values
- Each history has $n =$ stride random numbers
- No history depends on another for parallelism

Drawbacks:

- Stride must be selected to be sufficient for a problem
- Period overflow at $\left\lceil \frac{\text{period}}{\text{stride}} \right\rceil$ histories
- Overrunning period reduces the “effective” stride
  For generator 1, after 631 billion histories, each new history is within 1 of another.
- Changing seed just changes position in same sequence

Random Number Reuse

MCNP MONTE CARLO N-PARTICLE

Multiple papers indicating that short strides (< 100) affect results:

- J. S. Hendricks, “Random Number Stride in Monte Carlo Calculations,” Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-90-1845, Nov. 1990.
- T. E. Booth, “Bad Estimates as a Function of Exceeding the MCNP Random Number Stride,” Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-14-23159, May 2014.
- A. R. Hakim and D. A. Fynan, “Challenges of Near Critical-Fixed Source Monte Carlo Simulations of CANDU-6 Reactor: Bundle Power Tally Bias and Error Autocorrelation from Exceeding Random–Number Stride,” in M&C 2023, Niagara Falls, ON, CA; Aug 13–17, 2023.

Does overrunning a large (152917) stride affect results?
Unknown, low probability. However, it would be nice to **eliminate the question altogether** so users do not need to be aware of the generator.

Eliminating reuse requires generator state space to be larger than the product of the maximum numbers used per history, the maximum histories, and the maximum number of independent simulations.

Looking at Other Generators

What We Need

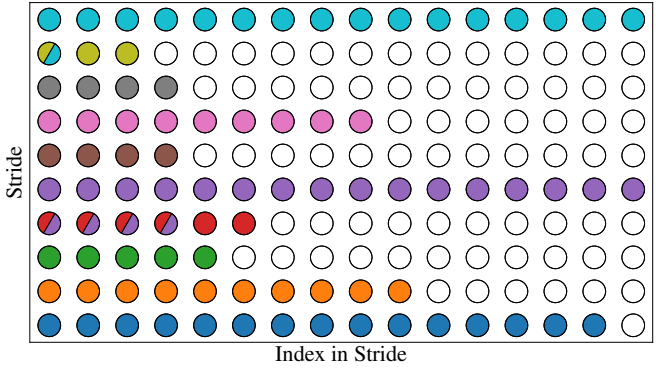
MCNP
MONTE CARLO N-PARTICLE

Target is a generator that:

- can handle parallel streams easily
- can generate longer parallel streams (stride $\gg 152917$)
- can generate more parallel streams (histories $\gg 2^{45}$)
- generates better quality bits
- performs similarly to, or faster than an LCG

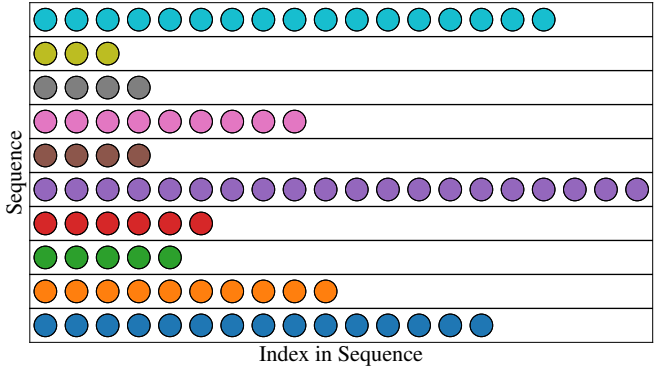
Parallelism - Skip Ahead

Start histories uniformly in the sequence:



Parallelism - Counter

Generator state contains a counter, guaranteeing n independent random numbers per initial state:



For some algorithms, sequences never intersect.

Bit Quality

MCNP MONTE CARLO N-PARTICLE

Cannot prove a generator good, just prove it bad (null hypothesis is generator is random). So we use 2 test suites:

- TestU01 version 1.2.3, BigCrush
  - 106 tests
  - Performed on bits both forward and backward
  - For 64-bit generators, performed on high and low bits
  - For < 64-bit generators, used only high bits
- PractRand version 0.95-pre
  - Extremely powerful series of statistical tests
  - Tests powers of two bytes until correlation tests fail
  - A “pass” is typically 32 TiB (1-4 days running)

Old and New Generator Comparison

Current 63-bit LCG

MCNP
MONTE CARLO N-PARTICLE

Generator Size	1 sequence of length 2^{63}
Bits Output	63
Parallelism	$\mathcal{O}(\log n)$ skip-ahead
Storage	8 bytes
TestU01	Failures on 10 tests
PractRand	Failed at 32 MiB
Time per Val	2.5 ns
Time Init.	115 ns

- Period insufficient
- Poor bit quality

SFC64

Generator Size	2^{192} sequences of length 2^{64} min., 2^{255} expected
Bits Output	64
Parallelism	Incrementing counter no random access
Storage	32 bytes
TestU01	Passed
PractRand	Passed to at least 32 TiB
Time per Val	2.6 ns
Time Init.	23 ns

- Part of state is a counter
- Each seed yields a new sequence
- Very high performance
- Selected for further use

SFC64 in the Code

SFC64

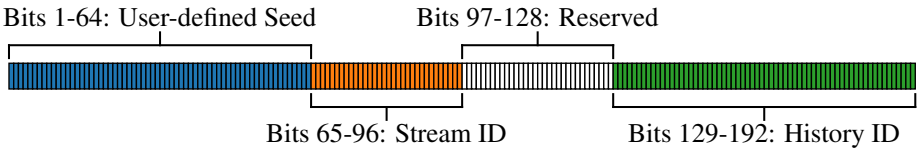
$$
\begin{aligned}
t_n &= a_n + b_n + n \\
a_{n+1} &= b_n \oplus \texttt{shift\_right}(b_n, 11) \\
b_{n+1} &= c_n + \texttt{shift\_left}(c_n, 3) \\
c_{n+1} &= \texttt{barrel\_shift}(c_n, 24) + t_n
\end{aligned}
$$

t_n output for generator.

- Added as Generator 8 for 6.3.1
- Plan to make default in 6.4

SFC64 Seeding

Seed is 192-bits:



- Seed set, generator iterated 18 times to mix bits
- Every history is a unique sequence
- Every user-seed changes all history sequences to new ones
- Multiple generator streams can be used from the same user-seed

Summary

- New random generator 8, SFC64, for 6.3.1
- As fast or faster than 63-bit LCG to generate numbers
- Faster to initialize state
- Significantly higher output bit quality
- Significantly more states (2^{192} streams of 2^{64})
- Eliminates the concept of strides and random number reuse
- No bad seed values

Case Study on RNG Behavior in MCNP6

An Analytic Benchmark

Hendricks' Spherical Benchmark

In 1991, John Hendricks studied the effects of changing the random number stride in Monte Carlo calculations using a simple test problem [1].

- Static, one-speed transport
- 15-mean-free-path radius sphere
- Isotropic point source at center
- 10% absorbing, 90% scattering (mock cross sections made with `simple_ace.pl` utility)
- Random numbers (RNs) are used for only a few basic functions
  - direction-of-flight: 2 RNs
  - distance-to-collision: 1 RN
  - reaction sampling or rouletting: 1 RN
  - total: 4 RNs per track

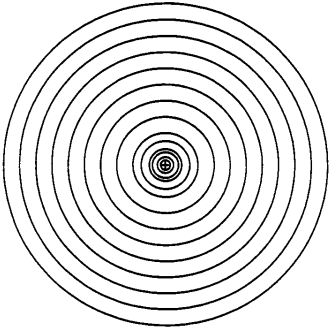


Fig. 4. A 15-mfp spherical geometry test problem. Fluxes are tallied at each of the concentric spheres. A mono-energetic, isotropic point source is at the center, and the scattering medium is 10% absorbing.

Spherical Integral Transport Equation

MCNP MONTE CARLO N-PARTICLE

Hendricks' spherical benchmark is expressed analytically by the static, one-speed, integral neutron transport equation for a homogeneous material in one-dimensional spherical coordinates [2], i.e.,

$$r\phi(r) = \int_0^a r' q(r') \left[E_1\left(\Sigma_T |r - r'|\right) - E_1\left(\Sigma_T |r + r'|\right) \right] dr', \tag{1}$$

for a finite or infinite radial boundary a, where

$$q(r') = \frac{\Sigma_S}{2}\phi(r') + \frac{Q(r')}{2} \tag{2}$$

and $E_n(x)$ is defined by the integral

$$E_n(x) = \int_0^1 e^{-x/\mu} \mu^{n-2} d\mu. \tag{3}$$

Numerical Approximation of the Integral

The integral in the transport equation is approximated by an N-point quadrature rule,

$$r\phi(r) = \sum_{j=1}^{N} w_j r_j q(r_j)[E_1(\Sigma_T|r - r_j|) - E_1(\Sigma_T|r + r_j|)], \tag{4}$$

where r_j and w_j are the quadrature points and weights, respectively. The scalar flux is evaluated at the quadrature points to form a closed system of equations,

$$r_i\phi_i = \sum_{j=1}^{N} w_j r_j q(r_j)[E_1(\Sigma_T|r_i - r_j|) - E_1(\Sigma_T|r_i + r_j|)], \tag{5}$$

where $\phi_i \equiv \phi(r_i)$.

Removal of the Logarithmic Singularity

The exponential integral has a logarithmic singularity at $r_i - r_j = 0$. It is removed by adding and subtracting $rq(r)$ under the integral,

$$r\phi(r) = \int_0^a \left[r'q(r') - rq(r) + rq(r)\right]\left[E_1\left(\Sigma_T|r - r'|\right) - E_1\left(\Sigma_T|r + r'|\right)\right]dr'. \quad (6)$$

The integral is separated into two parts and the second integral is solved exactly,

$$r\phi(r) = \int_0^a \left[r'q(r') - rq(r)\right]\left[E_1\left(\Sigma_T|r - r'|\right) - E_1\left(\Sigma_T|r + r'|\right)\right]dr' + rq(r)p(r), \quad (7)$$

where

$$p(r) = \frac{1}{\Sigma_T}\{2[1 - E_2(\Sigma_T r)] - E_2[\Sigma_T(a - r)] + E_2[\Sigma_T(a + r)]\}. \quad (8)$$

Formulation of the System of Equations (1/2)

The integral in the transport equation with the logarithmic singularity removed is approximated by an N-point quadrature rule,

$$r_i\phi_i = \sum_{\substack{j\neq i \\ j=1}}^{N} w_j[r_j q(r_j) - r_i q(r_i)][E_1(\Sigma_T|r_i - r_j|) - E_1(\Sigma_T|r_i + r_j|)] + r_i q(r_i)p(r_i). \tag{9}$$

This is recast as a $N \times N$ system of equations,

$$\overline{\overline{A}}\,\overline{\phi} = \overline{Q}, \tag{10}$$

Formulation of the System of Equations (2/2)

MCNP MONTE CARLO N-PARTICLE

where

$$A_{ii} = r_i \left[1 - \frac{\Sigma_S}{2} p(r_i) \right] + \sum_{\substack{j \neq i \\ j=1}}^{N} w_j r_i \frac{\Sigma_S}{2} [E_1(\Sigma_T |r_i - r_j|) - E_1(\Sigma_T |r_i + r_j|)] \quad (11)$$

for $i = j$,

$$A_{ij} = -w_j r_j \frac{\Sigma_S}{2} [E_1(\Sigma_T |r_i - r_j|) - E_1(\Sigma_T |r_i + r_j|)] \quad (12)$$

for $i \neq j$, and

$$Q_i = r_i \frac{Q(r_i)}{2} p(r_i) + \sum_{\substack{j \neq i \\ j=1}}^{N} w_j \left[r_j \frac{Q(r_j)}{2} - r_i \frac{Q(r_i)}{2} \right] [E_1(\Sigma_T |r_i - r_j|) - E_1(\Sigma_T |r_i + r_j|)]. \quad (13)$$

Isotropic Point Source

The uncollided scalar neutron flux is Q_i/r_i. For a spherical system with an isotropic point source at the center, i.e.,

$$Q(r) = \frac{s_0}{4\pi}\frac{\delta(r)}{r^2}, \tag{14}$$

the uncollided flux is

$$\phi_i = s_0\frac{\exp\left(-\Sigma_T r_i\right)}{4\pi r_i^2}, \tag{15}$$

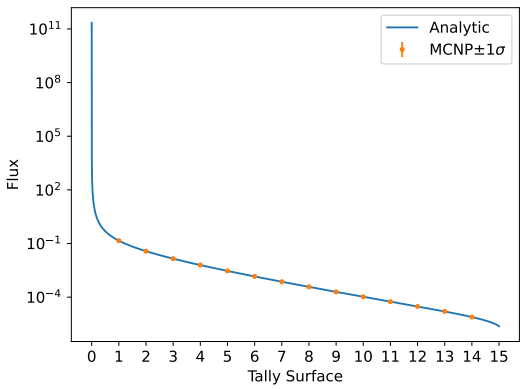
which means that

$$Q_i = s_0\frac{\exp\left(-\Sigma_T r_i\right)}{4\pi r_i}. \tag{16}$$

Results using Gauss-Legendre Quadrature

10,000 Gauss-Legendre quadrature points were used to converge the results to 6 significant figures.

Tally Surface	Flux
1	1.43284E-01
2	3.71264E-02
3	1.41400E-02
4	6.19403E-03
5	2.91509E-03
6	1.43307E-03
7	7.25476E-04
8	3.75078E-04
9	1.96984E-04
10	1.04658E-04
11	5.60141E-05
12	2.99863E-05
13	1.57773E-05
14	7.70698E-06



The MCNP6 [3] default random number generator (GEN=1) is used with STRIDE=1.

Behavior of Tallies when the Stride is Exceeded

Implications of Exceeding RNG Stride

```
warning.  random number stride 1 exceeded 10000000 times.
```

The sample (population) variance is used to calculate the reported standard deviation of the mean, assuming that all histories are independent and uncorrelated. That is,

$$S^2 = \frac{N}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)^2\right]. \tag{17}$$

If the random number stride is exceeded, then the correlations between histories must be taken into account to obtain the proper sample variance and subsequent standard deviation of the mean.

Generalized Population Variance

The sample (population) covariance between neighboring (offset) histories can be defined as,

$$C_{i,i+j} = \frac{N-j}{N-j-1}\left[\frac{1}{N-j}\sum_{i=1}^{N-j} x_i x_{i+j} - \left(\frac{1}{N-j}\sum_{i=1}^{N-j} x_i\right)\left(\frac{1}{N-j}\sum_{i=1+j}^{N} x_i\right)\right], \tag{18}$$

where $1 \leq j < N$ represents the history offset. For example, if $j = 1$ the sample covariance $C_{i,i+1}$ is the covariance between subsequent histories. If $j = 0$, we recover the sample variance. The covariance-corrected sample variance can then be defined as,

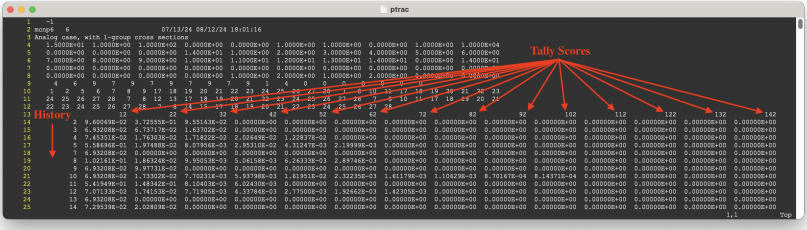
$$\tilde{S}^2 = S^2 + 2\sum_{j=1}^{N-1} C_{i,i+j}, \tag{19}$$

Computing Sample Covariance

PTRAC has a legacy feature where individual history scores can be printed into an easy-to-use column-formatted file:

```
ptrac file=asc write=all max=9e9 coinc=col
      tally=12,22,32,42,52,62,72,82,92,102,112,122,132,142
      value=0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
```

This produces a PTRAC file in the following form:



Correlations Between Histories

The normalized correlation coefficient can be used to visualize the magnitude of any issues when exceeding the stride:

$$\rho_{i,i+j} = \frac{C_{i,i+j}}{\sqrt{C_{i,i}C_{i+j,i+j}}} \tag{20}$$

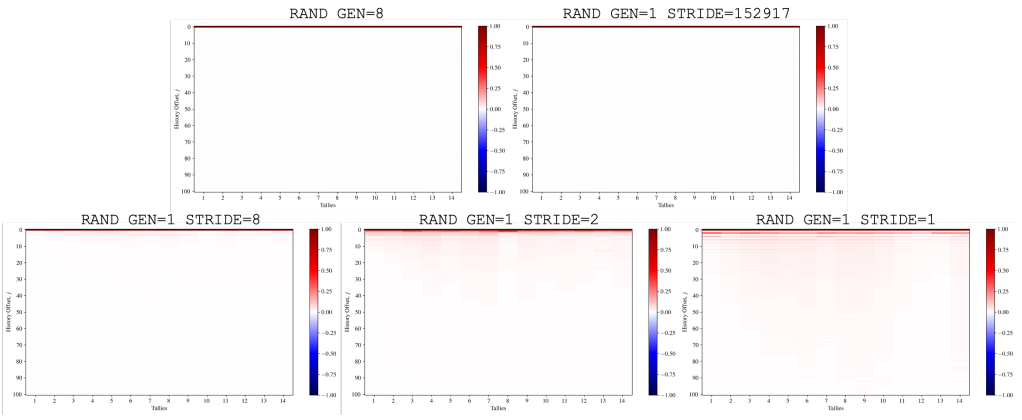
The history-offset correlations are computed using the MCNP6 default and the new SFC64 random number generator [4] in MCNP6.3.1 and later.

- SFC64 (`RAND GEN=8`) has no stride. Each history uses an independent stream of random numbers.
- 48-bit LCG (`RAND GEN=1`) with strides of 1, 2, 8, 16, 100, 1,000, and 152,917 (default stride).

Numerical Results: Correlations

MCNP
MONTE CARLO N-PARTICLE

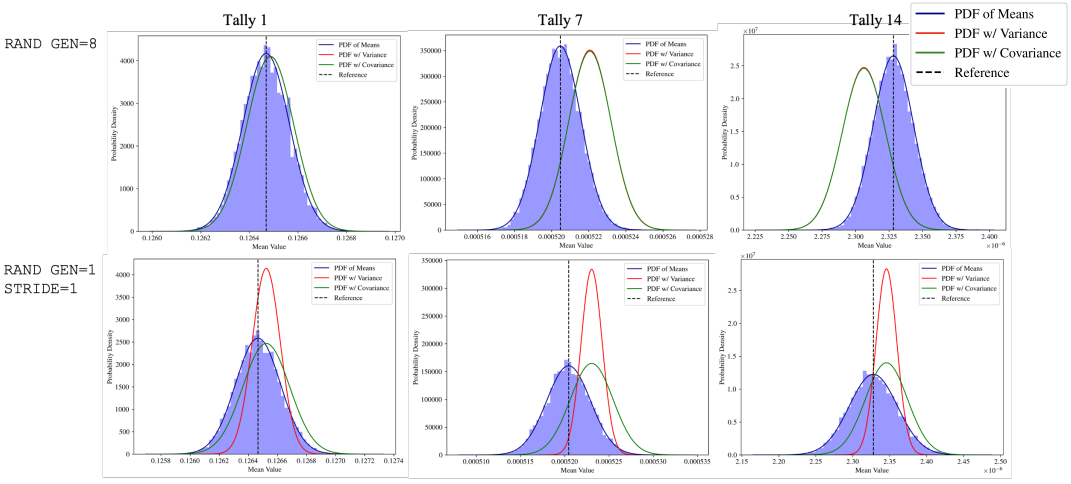
Positive correlation values between neighboring histories can be seen in red when the random number generator stride is exceeded.



Numerical Results: Probability Density Functions

MCNP MONTE CARLO N-PARTICLE

Each histogram and curve in blue represent 10,000 independent (varying random number seed) mean values.

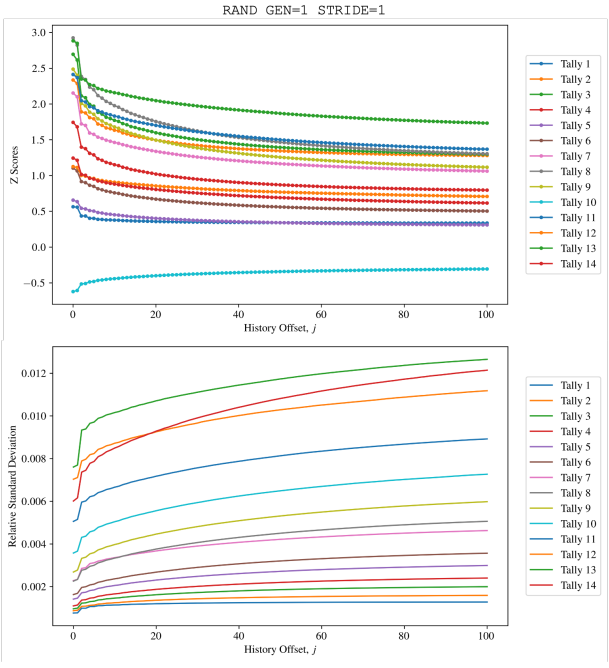


Numerical Results: Z Scores and Rel. Std. Dev.

A metric to determine how close the distribution is to the reference solution is the Z score, defined as

$$Z = \frac{\mu - x}{\sigma}, \qquad (21)$$

where x is the semi-analytic reference flux.



Summary

- A semi-analytical solution to a simple benchmark is provided along with consistent MCNP6 results.
- The standard deviation of the mean of the tallies can be underestimated if the random number generator stride is exceeded; the mean of the tallies are unaffected.
- The covariance-corrected standard deviation can be computed to obtain more appropriate confidence intervals.
- **The best way to avoid any stride exceedance issues is to use the new SFC64 (`RAND GEN=8`) random number generator in MCNP6.3.1 and later.**

Questions?

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] J. S. Hendricks, “Effects of Changing the Random Number Stride in Monte Carlo Calculations,” Nuclear Science and Engineering, vol. 109, no. 1, pp. 86–91, Sep. 1991. DOI: 10.13182/NSE91-A23846

[2] J. J. Duderstadt and W. R. Martin, Transport Theory. New York, NY, USA: John Wiley & Sons, 1979.

[3] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, “MCNP® Code Version 6.3.0 Theory & User Manual,” Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

[4] C. J. Josey, “Reassessing the MCNP Random Number Generator,” Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-23-25111, Rev. 1, Sep. 2023. DOI: 10.2172/1998091

Los Alamos NATIONAL LABORATORY

MCNP® MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-03: MCNP6 Parallel Capabilities and Algorithms

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

MCNP
MONTE CARLO N-PARTICLE

Monte Carlo Parallelism

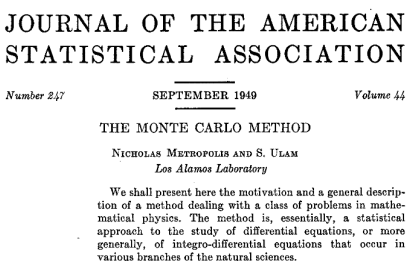
MCNP6 Parallelism Options

MCNP6.3 Parallelism Updates

Monte Carlo Method and Parallelism

MCNP
MONTE CARLO N-PARTICLE

Before massively parallel computing systems were invented, the original Monte Carlo Method theorists knew of its potential. In 1949, Metropolis and Ulam published "The Monte Carlo Method" [1] and made mention of the potential to perform parallel calculations.



JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION

Number 247 SEPTEMBER 1949 *Volume 44*

THE MONTE CARLO METHOD

NICHOLAS METROPOLIS AND S. ULAM
Los Alamos Laboratory

We shall present here the motivation and a general description of a method dealing with a class of problems in mathematical physics. The method is, essentially, a statistical approach to the study of differential equations, or more generally, of integro-differential equations that occur in various branches of the natural sciences.

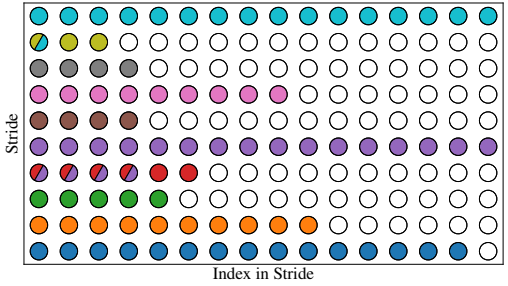
Regarding the sequence of operations on a machine, more can be and has been done. The choice of the *kind* of step to be performed by the machine can be made to depend on the values of certain parameters just obtained. In this fashion even dependent probabilistic processes can be performed. Quite apart from mechanized computations, let us point out one feature of the method which makes it advantageous with, say, stepwise integration of differential equations. In order to find a particular solution, the usual method consists in iterating an algebraical step, which involves in the nth stage values obtained from the $(n-1)$th step. The procedure is thus serial, and in general one does not shorten the time required for a solution of the problem by the use of more than one computer. On the other hand, the statistical methods can be applied by many computers working in parallel and independently. Several such calculations have already been performed for problems of types discussed above.[2]

Due to the need to follow the independently and identically distributed (IID) process, Monte Carlo is sometimes considered an embarrassingly parallel method. Regardless, there are many challenges as computing systems continue to evolve.

Parallelism – (pseudo) Random Numbers

MCNP
MONTE CARLO N-PARTICLE

Typically, Monte Carlo codes use a pseudo-Random Number Generator (RNG) to achieve the independence of individual histories. That is, histories may start uniformly in the RNG sequence where the stride is the offset between starting points in the sequence.



- Within a single calculation, each history must start at a unique place in the RNG sequence; the notion of skipping ahead in the RNG sequence is important
- Using a multiple calculation approach, typically the RNG sequence can be initialized with a unique seed to achieve independence

MCNP6 Built-in Parallelism Options

MCNP
MONTE CARLO N-PARTICLE

- Two modes of parallelism within the code; can be used separately or together
- Thread- or task-based parallelism through OpenMP
  - Single program, independent sections or subprograms
  - Common address space, must distinguish private and shared data
  - Critical sections must be “locked”
  - Can run only on shared memory systems, not distributed memory
  - Accessed through `tasks` option on the command line

```
> mcnp6 i=myinput tasks 8
```

- Message-passing parallelism through MPI:
  - Independent programs
  - Separate memory address space for each program
  - All data must be passed between programs by explicit messages
  - Can run on distributed or shared memory systems
  - Efficient only when compute time, $T_{\text{Monte Carlo}} >> T_{\text{message passing}}$

```
> srun -n 8 mcnp6.mpi i=myinput
```

- OpenMP and MPI parallelism can be mixed together

MCNP6 Parallelism Options

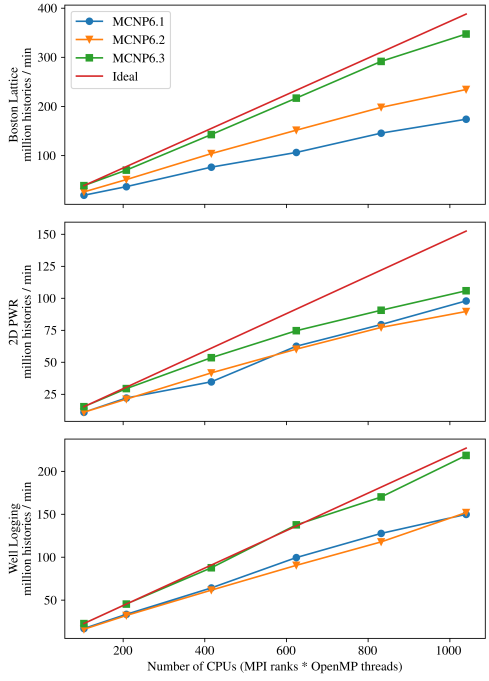
MCNP6 Parallel Scaling – Applications

Computational efficiency varies by **application**:

Boston Lattice A fixed-source photon transport problem in a ~ 100 million element lattice model of the city of Boston.

2D PWR A k-eigenvalue neutron transport problem of a 2D pressurized water reactor, infinite in the axial direction.

Well Logging A fixed-source neutron transport problem with several reaction rate and energy spectrum tallies throughout the model.

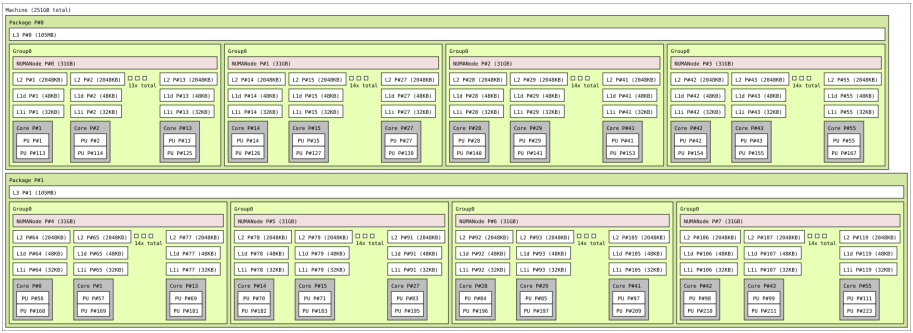


MCNP6 Parallel Scaling – Hardware

MCNP
MONTE CARLO N-PARTICLE

Computational efficiency varies with **hardware**:

- For example, Rocinante HPC system at LANL includes:
  - Intel Xeon Platinum 8479 processors
  - Node contains: 256 GB memory, 8 logical domains w/ 14 CPU cores
- Reasonable node-to-node scaling (MPI)
- On-node scaling (OpenMP and MPI) challenging
  - Non-uniform memory access (NUMA) kills OpenMP performance



MCNP6 “Poor Man’s” Parallelism

MCNP
MONTE CARLO N-PARTICLE

- Run same model, with identical geometry, sources, tallies, etc., multiple times using different random number sequences
  - Not the most elegant approach, but can be practical
  - May be needed when code feature is not compatible with OpenMP or MPI
- The `RAND` card can be used to achieve this
  - Use `seed=<#>` to change pseudo RNG sequence
    - Trivial to setup and run

```
Calc. A:  nps 2000 ; rand seed=123456789
Calc. B:  nps 2000 ; rand seed=987654321
```

    - Generally recommended to use batch statistics (with same batch size, `nps`) to combine results
  - The `mcnp_pstudy` tool [§E.8 of 2] can be used to make quick work of this
    - Add input file `mcnp_pstudy` directives

```
c @@@ RNGSEED = ( rn_seed() )
c @@@ REPEATN = repeat 10000
rand seed=RNGSEED
```

    - Setup, run, and collect results

```
> mcnp_pstudy.pl -i myinput -setup -run -collect
```

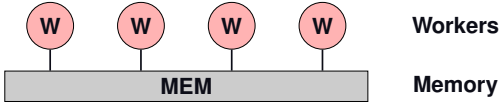
Most Efficient Parallel MCNP6 Calculations

MCNP MONTE CARLO N-PARTICLE

- Must match the parallelism strategy to the hardware being used
- CPU considerations
  - Consider how many cores are available; no reason to oversubscribe
  - Consider how many logical (or NUMA) domains exist (should use MPI)
  - Consider how many cores in each logical domain (can use OpenMP or MPI)
- Memory considerations
  - Consider how much memory the model requires
  - When running with shared memory (OpenMP), there are memory savings
  - When running with separate memory (MPI), the model will be replicated in memory; might need to undersubscribe to fit a model on a piece of hardware
- Tips
  - Resource limitations, such as wallclock time, queque priorities, etc., may change strategy
    - Do lots of smaller, independent jobs make sense?
  - Run shorter test calculations to understand total job size needed
    - Find report `# M-histories/hr` and scale accordingly
  - For k-eigenvalue calculations, more histories per cycle will generally improve MPI-based efficiency

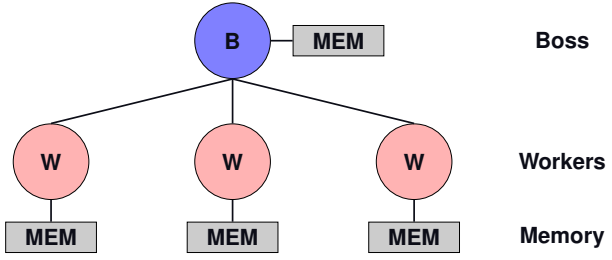
MCNP6 Shared Memory (`tasks`) Parallelism

- Usage of OpenMP threading:
  - On shared memory nodes, hardware (e.g., laptop computers)
  - Typically works well for neutron/photon and k-eigenvalue problems
  - If hyperthreads (logical cores) available, typically provide little to no benefit
  - The number of `tasks` created is determined at the command line by the user
    - If more threads are requested than available, MCNP6 will create the user-selected number of threads regardless of the hardware specifications
    - Efficiency will likely suffer due to significant context switching by threads
- When not to use:
  - On distributed memory systems (can only access local memory)
  - Across NUMA domains (poor performance)
  - When features do not allow it (e.g., high-energy physics models, event log)



MCNP6 Distributed Memory (MPI) Parallelism

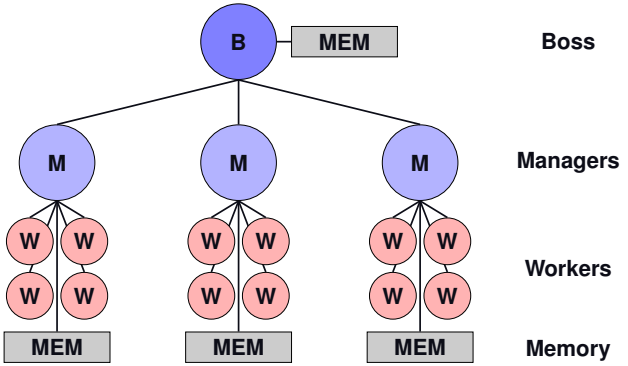
- Usage of MPI:
  - On either shared or distributed memory nodes, hardware (e.g., server, cluster)
  - Typically works well for most applications
  - Boss rank does not simulate histories – Workers pass information to Boss
    - Message passing occurs at each rendezvous (e.g., writing to the `RUNTPE`, every k-eigenvalue cycle)
    - If possible try to limit rendezvous frequency (see `PRDMP` card)
- When not to (or limit) use:
  - When model memory usage is too large and replication is not possible
  - When features do not allow it (e.g., event log)



MCNP6 Mixed-mode (OpenMP+MPI) Parallelism

MCNP
MONTE CARLO N-PARTICLE

- Usage of mixed OpenMP+MPI:
  - On either shared or distributed memory nodes, hardware (e.g., server, cluster)
  - Typically works well for most applications
  - Limiting worker threads to logical (or NUMA) domains and using MPI across domains is effective
  - Some memory savings compared to MPI-only calculations
- When not to (or limit) use:
  - When features do not allow it (e.g., event log)



Los Alamos
NATIONAL LABORATORY

MCNP6.3 Parallelism Updates

MCNP6 PRDMP for Parallel Calculations

MCNP
MONTE CARLO N-PARTICLE

Print and dump frequency (PRDMP) card

```
PRDMP ndp ndm mct ndmp dmmp
```

- `ndp` increment for printing tallies
- `ndm` increment for dump to RUNTPE
- `mct` flag to control printing to MCTAL file
- `ndmp` max number of dumps in RUNTPE
- `dmmp` increment for TFC and rendezvous

Some notes to help in parallel calculations:

- Increments are in units of
  - particles for fixed-source problems
  - cycles for k-eigenvalue problems
- Increasing `ndp`, `ndm`, and `dmmp` can limit rendezvous frequency
- Writing MCTAL file using `mct` is helpful when combining tallies from independent calculations

MCNP6.3 Parallelism Updates

MCNP
MONTE CARLO N-PARTICLE

- PTRAC with added HDF5-formatted output is capable of parallel (OpenMP and MPI) execution
- FMESH added new remote memory access (RMA) batch statistics capability to exceedingly large tallies
- New PIO card to handle certain parallel input/output operations
- Some NUMA issues have been improved such that threads may have faster memory accesses
- Best effort MPI builds of the binaries now distributed
  - Still need to install compatible MPI library to use
  - May still need source code to build MPI version (e.g., using RMA option)

MCNP6.3 HDF5 PTRAC Parallelism

MCNP
MONTE CARLO N-PARTICLE

- HDF5 PTRAC files can be produced with MPI, OpenMP threads, or both
  - MPI-parallel PTRAC requires an MPI enabled installation of HDF5
  - OpenMP-parallel PTRAC (`tasks` option) works with any HDF5 build
- Each process buffers data into memory over multiple histories
  - Periodically all processes write data to the file
  - Provides greater speed than writing semicontinuously
- The buffers' memory usage can be very large
  - A `std::bad_alloc` error will appear if memory is exhausted (memory swapping may occur instead)
  - Prevented with the FLUSHNPS option
- Several important MPI-parallel HDF5 considerations
  - Intended for use on parallel file systems, e.g., Lustre
  - Will be slow or hang on most Network File Systems (NFS)
    - The MCNP code cannot detect or predict this misbehavior
    - Always try a quick initial run to verify that a file is written correctly
  - OpenMP-parallel works on any file system

MCNP6.3 HDF5 `turbotally` Parallelism

A new C++ library in MCNP6.3 provides a variety of tally algorithms:

- `hist` This algorithm has no optimizations and is therefore computationally slower than the default algorithms used previously.
- `fast_hist` Statistically equivalent to the history-based algorithm implemented in MCNP6.2 and earlier, with optimizations in place for increased computational speed. This is the default algorithm in MCNP6.3.
- `batch` A new batch statistics algorithm is available which computes mean values over a fixed batch size and combines the batch mean values.
- `rma_batch` The batch algorithm enabled using the remote memory access (RMA) capability of MPI. With this algorithm exceedingly large tallies can be modeled because the tally data structures are effectively decomposed across all memory available across all nodes.

Approximate Peak `turbotally` Memory Usage

Algorithm	Node Memory Usage (Bytes)
`hist`	$16T_{\text{size}}n_{\text{ranks}} + 8T_{\text{size}}n_{\text{ranks}}n_{\text{threads}}$
`fast_hist`	$16T_{\text{size}}n_{\text{ranks}} + 8.8T_{\text{size}}n_{\text{ranks}}n_{\text{threads}}$
`batch`	$16T_{\text{size}} + 8T_{\text{size}}n_{\text{ranks}}$
`rma_batch`	$24T_{\text{size}}/n_{\text{nodes}}$

MCNP6 `PIO` for Parallel Calculations

MCNP
MONTE CARLO N-PARTICLE

Enable Parallel IO (PIO) card

```
PIO value
```

value=blank or ON The code is built with parallel HDF5 support, and is run with MPI, features that have parallel IO support will use it.

value=OFF Parallel IO is disabled (DEFAULT)

Some notes to help in parallel calculations:

- This option is not saved in the runtape. It must be specifically enabled for all runs, continue or otherwise.
- Certain components of the code (such as FMESH with the XDMF output format) can write results using parallel HDF5. Enabling this feature will perform input and output in parallel in these circumstances.
- This feature is not enabled by default as not all file systems will benefit from parallel IO.
  - Some file systems will even cause the MCNP code to lock up if parallel IO is used. In testing, NFS partitions would often cause this.
  - As a result, one should test file systems with short simulations to see if this will work and provide a benefit to a simulation before running a large problem.

Questions?

MCNP
MONTE CARLO N-PARTICLE

Monte Carlo Parallelism

MCNP6 Parallelism Options

MCNP6.3 Parallelism Updates

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] N. Metropolis and S. Ulam, "The Monte Carlo Method," Journal of the American Statistical Association, vol. 44, no. 247, pp. 335–341, Sep. 1949. DOI: 10.1080/01621459.1949.10483310

[2] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.0 Theory & User Manual," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

Los Alamos NATIONAL LABORATORY

MCNP® MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-04: HDF-Based Input/Output of Unstructured Mesh Models

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

Introduction to HDF5 [1]

- HDF: Hierarchical Data Format
- Developed by The HDF Group
  - Non-profit organization
  - Spun off from Nat'l Center for Supercomp. Appl. at Univ. of Illinois
  - Central authority to ensure quality and prevent fragmentation
- BSD-like license, freely available, portable, numerous APIs
  - Official APIs: C, C++, Fortran, Java
  - Unofficial APIs: Julia, Matlab, Mathematica, Perl, Python, R
- Developed with speed and scalability in mind
- Three major objects: groups, datasets, and attributes
  - Groups are containers for datasets or other groups
  - Datasets are homogeneous n-dimensional arrays
    - Can contain complex objects, e.g., images
  - Attributes can be added to either groups or datasets

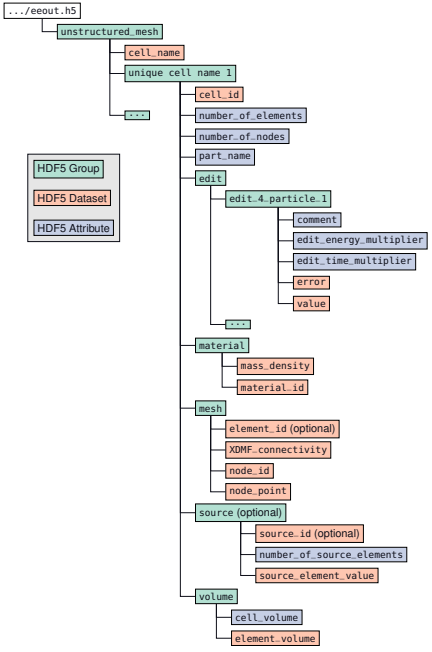
History of HDF5

MCNP MONTE CARLO N-PARTICLE

- 1987: work to develop all-encompassing hierarchical object-oriented file
- 1990 and 1992: NSF grants provided crucial funding
  - NSF wanted to harmonize netCDF and HDF formats
  - Drove improved V&V basis
  - NASA selected HDF as its standard data and information system
- 1996: major redesign: went to current group & dataset approach
- More information in videos at: https://www.hdfgroup.org/about-us

- HDF4 is older but actively supported
- HDF5 is current (and actively supported)
  - Attempts to address some HDF4 limitations

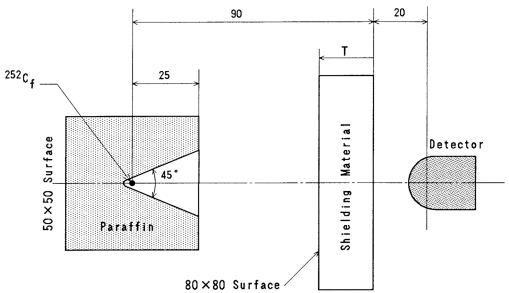
Example HDF5 File Hierarchy

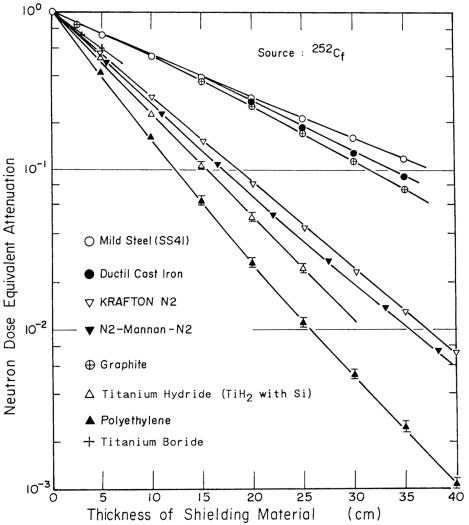


Ueki Example

Ueki with 20 cm of Graphite

- Shielding benchmark from early 1990s [2, 3]
  - This work focuses on a 20 cm graphite shield as described in [2]
  - Detector approximated as $5 \times 5 \times 5$ cm cube



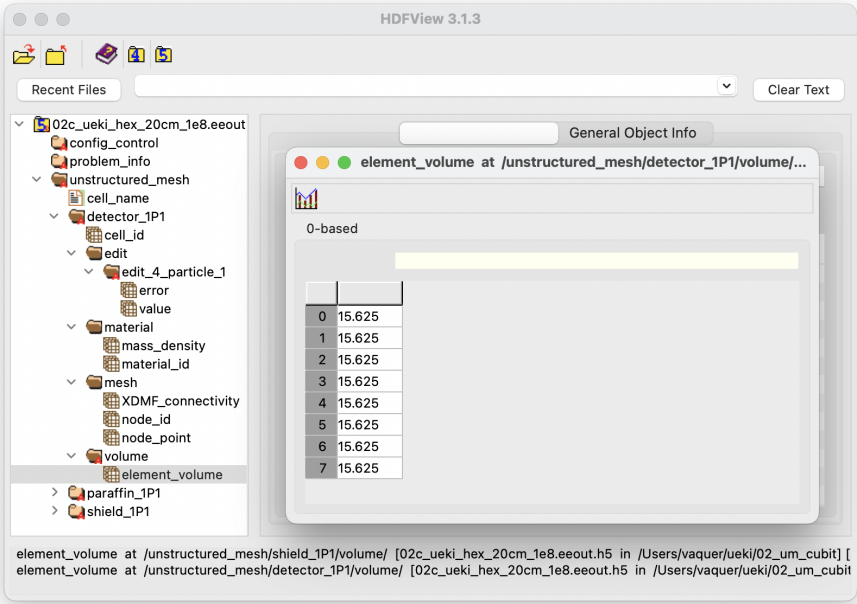


Ueki – UM HDF5 File Interrogation

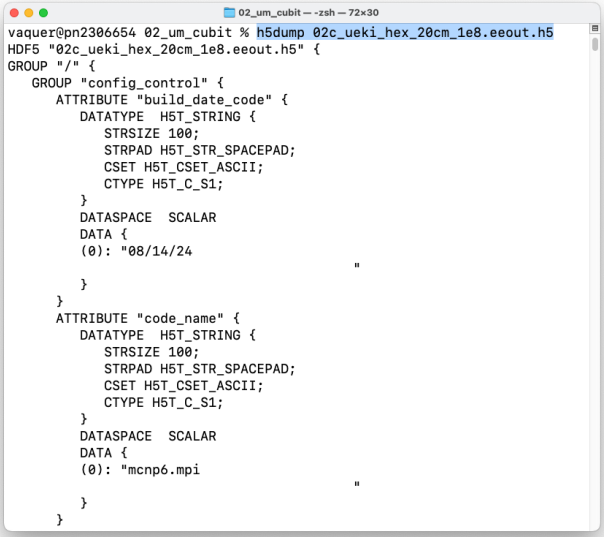
MCNP MONTE CARLO N-PARTICLE

- The line `hdf5file=02c_ueki_hex_20cm_1e8.eeout.h5` inside of `02c_ueki_hex_20cm_1e8.mcnp.inp` instructs MCNP to generate an HDF5 file named `02c_ueki_hex_20cm_1e8.eeout.h5`
  - `hdf5file` is an MCNP6.3 feature
- Running MCNP with the "i" command line option will create a UM HDF5 file without simulation results:
  - `mcnp6 i i=02c_ueki_hex_20cm_1e8.mcnp.inp`
- Running MCNP with the "ixr" command line options will include simulation results in the UM HDF5 file:
  - `mcnp6 ixr i=02c_ueki_hex_20cm_1e8.mcnp.inp`
    - Note that `ixr` is the default set of execution options
- The generated UM HDF5 can then be interrogated using:
  - HDFView: a graphical user interface for viewing HDF files
  - `h5dump`: a command line interface for viewing HDF5 files
  - `h5py`: a Python package for reading and writing HDF5 files

Ueki – HDFView



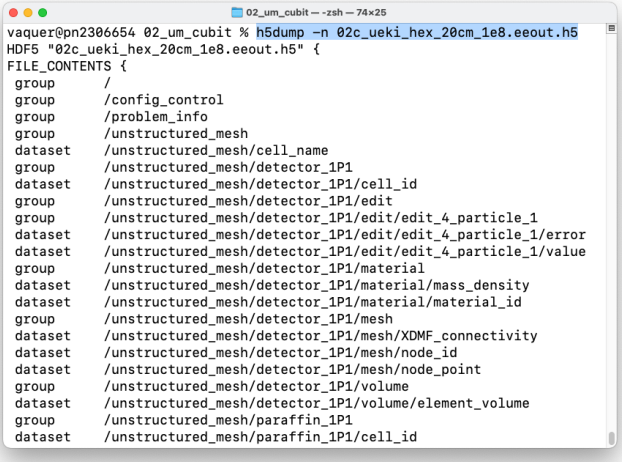
Ueki – h5dump


02_um_cubit — -zsh — 72×30
vaquer@pn2306654 02_um_cubit % h5dump 02c_ueki_hex_20cm_1e8.eeout.h5
HDF5 "02c_ueki_hex_20cm_1e8.eeout.h5" {
GROUP "/" {
GROUP "config_control" {
ATTRIBUTE "build_date_code" {
DATATYPE H5T_STRING {
STRSIZE 100;
STRPAD H5T_STR_SPACEPAD;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE SCALAR
DATA {
(0): "08/14/24
"
}
}
ATTRIBUTE "code_name" {
DATATYPE H5T_STRING {
STRSIZE 100;
STRPAD H5T_STR_SPACEPAD;
CSET H5T_CSET_ASCII;
CTYPE H5T_C_S1;
}
DATASPACE SCALAR
DATA {
(0): "mcnp6.mpi
"
}
}

Ueki – h5dump “-n” option

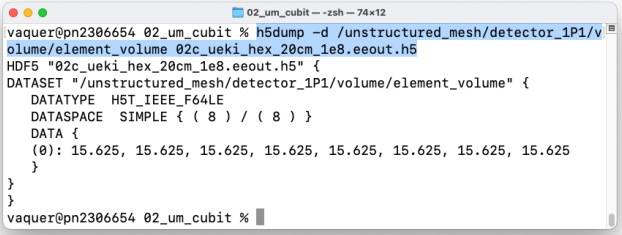
MCNP
MONTE CARLO N-PARTICLE

- Use the “-n” command line option to display the structure of an HDF5 file, and identify the exact path to a group or dataset



Ueki – h5dump “-d” option

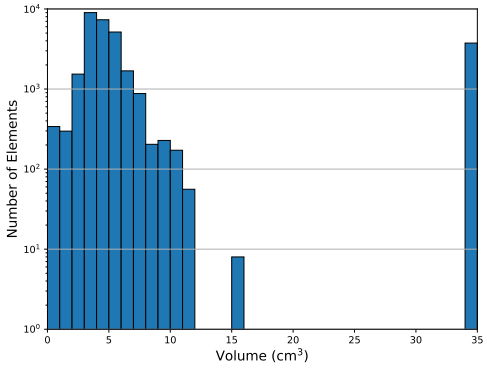
- Use the “-d” command line option, followed by the path to a dataset, to view the values for that dataset



Ueki – Plotting a Histogram of Volumes

MCNP MONTE CARLO N-PARTICLE

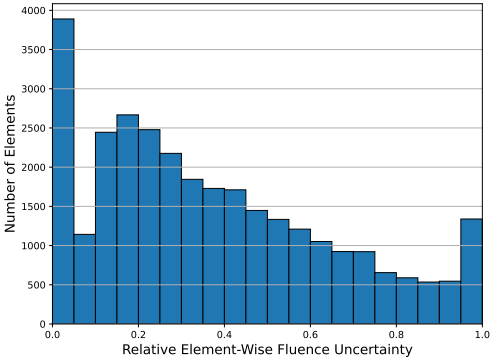
```
import h5py
import matplotlib.pyplot as plt
import numpy as np

# Open the HDF5 file
f = h5py.File("02c_ueki_hex_20cm_1e8.eeout.h5", "r")

# Initialize a list to store all volumes
volumes = []

# Iterate over the groups to extract volumes
for g in f["unstructured_mesh"].items():
    if isinstance(g[1], h5py.Group):
        volumes.extend(g[1]["volume/element_volume"])

# Plot histogram
plt.figure(figsize=(8, 6))
plt.hist(volumes, bins=np.linspace(0, 35, 36), edgecolor='black')
plt.xlabel("Volume (cm$^3$)", fontsize=14)
plt.ylabel("Number of Elements", fontsize=14)
plt.yscale("log")
plt.xlim([0, 35]); plt.ylim([1, 1e4])
plt.grid(axis='y')
plt.savefig('ueki_volume_histogram.pdf')
```



Ueki – Plotting a Histogram of Uncertainties

MCNP MONTE CARLO N-PARTICLE

```
import h5py
import matplotlib.pyplot as plt
import numpy as np

# Open the HDF5 file
f = h5py.File("02c_ueki_hex_20cm_1e8.eeout.h5", "r")

# Initialize a list to store all fluence errors
fluence_errors = []

# Iterate over the groups to extract errors
for g in f["unstructured_mesh"].items():
    if isinstance(g[1], h5py.Group):
        fluence_errors.extend(g[1]["edit/edit_4_particle_1/error"])

# Plot histogram
plt.figure(figsize=(8, 6))
plt.hist(fluence_errors, bins=np.linspace(0, 1, 21), edgecolor='black')
plt.xlabel("Relative Element-Wise Fluence Uncertainty", fontsize=14)
plt.ylabel("Number of Elements", fontsize=14)
plt.xlim([0, 1])
plt.grid(which='both', axis='y')
plt.savefig('ueki_uncertainty_histogram.pdf')
```



Questions?

MCNP
MONTE CARLO N-PARTICLE

Introduction to HDF5

Ueki Example

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] The HDF Group, “Hierarchical Data Format, Version 5,” Website, Apr. 2020. URL: http://www.hdfgroup.org/HDF5/

[2] K. Ueki, A. Ohashi, and Y. Anayama, “Neutron Shielding Ability Of Krafton N2 - Mannan - Krafton N2 Sandwich-Type Material And Others,” in New Horizons in Radiation Protection and Shielding, Pasco, WA, USA, 1992, pp. 130–137.

[3] K. Ueki, A. Ohashi, N. Nariyama, S. Nagayama, T. Fujita, K. Hattori, and Y. Anayama, “Systematic Evaluation of Neutron Shielding Effects for Materials,” Nuclear Science and Engineering, vol. 124, no. 3, pp. 455–464, Nov. 1996. DOI: 10.13182/NSE124-455

[4] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, “MCNP® Code Version 6.3.0 Theory & User Manual,” Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

Los Alamos NATIONAL LABORATORY

MCNP® MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-05: Geometry and Tally Visualization via XDMF-Based Output

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

Introduction to XDMF [3, 4]

MCNP MONTE CARLO N-PARTICLE

- ASCII, XML-formatted, file
  - Standard, albeit not actively developed, file format
  - Supports various structured and unstructured geometries
  - Can contain data and/or **contain pointers into HDF5 files**

- Cross-platform & open-source support (direct visualization of output)
  - ParaView [1]
  - VisIt [2]

- No additional MCNP library dependencies
  - MCNP outputs are XDMF version 2
  - MCNP HDF5 and XDMF files are assumed to be in the same directory

ParaView Introduction

MCNP
MONTE CARLO N-PARTICLE

- Open-source & cross-platform software
- Maintained by Kitware Inc.
  - Kitware also maintains CMake, VTK, etc.
- Supported by, among others:
  - Advanced Simulation and Computing Program
  - Army Research Laboratory
  - Los Alamos National Laboratory
  - Sandia National Laboratories

- Getting it: `https://www.paraview.org/download`
- Getting help with it: `https://www.paraview.org/community-support`
- Issues: `https://gitlab.kitware.com/paraview/paraview/-/issues`

Motivating Calculation Geometry & Results

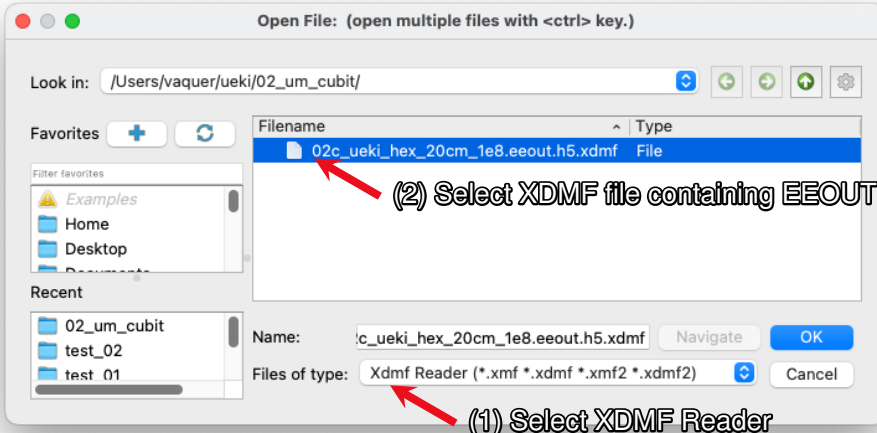
MCNP MONTE CARLO N-PARTICLE

- Focus: UM geometry and results
  - EEOUT file converted to an ASCII Unstructured VTK File (.vtu) [5]
  - HDF5+XDMF output files (not available in MCNP6.2 and earlier)

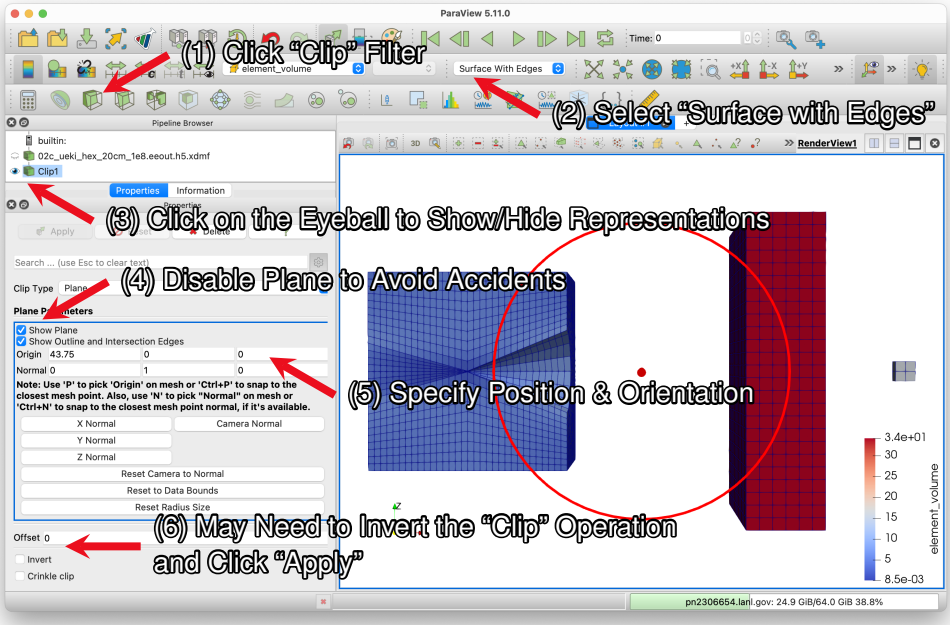
- Other possibilities (not covered here)
  - Also: MCNP mesh tally files
    - `meshtal` converted to an ASCII Structured VTK File (.vts)
    - HDF5+XDMF output files (not available in MCNP6.2 and earlier)
  - Particle tracks
  - Point data (collisions, fissions, etc.)
  - Voxelized CSG representations [6]
  - Weight-window mesh files

Ueki Example

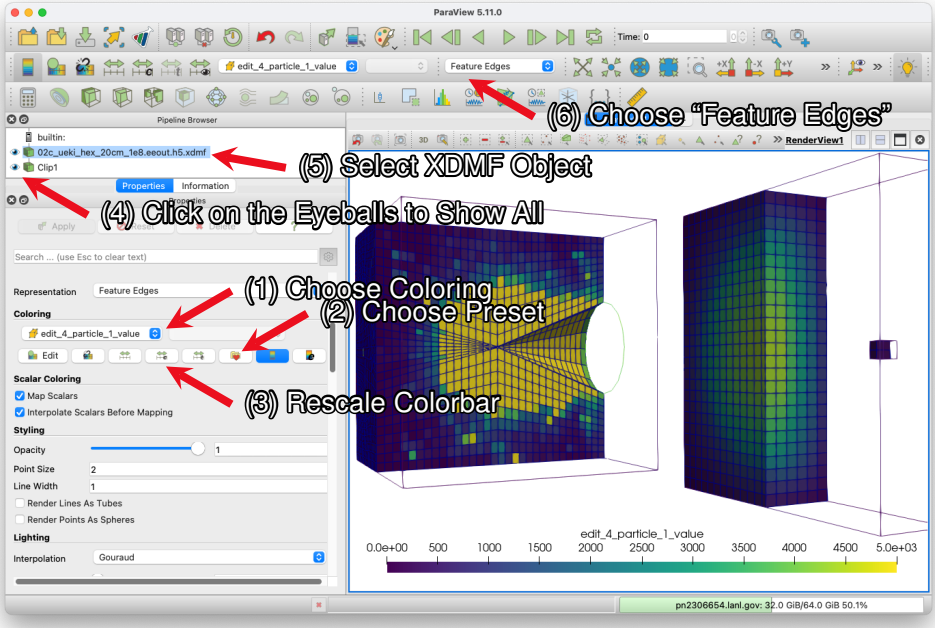
Open the Ueki Model in ParaView



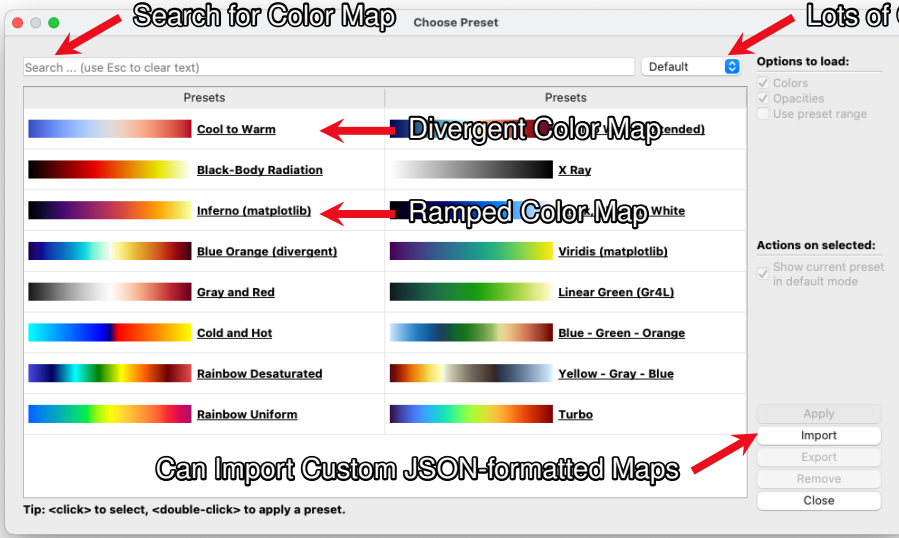
Clip (Hide Half Space)



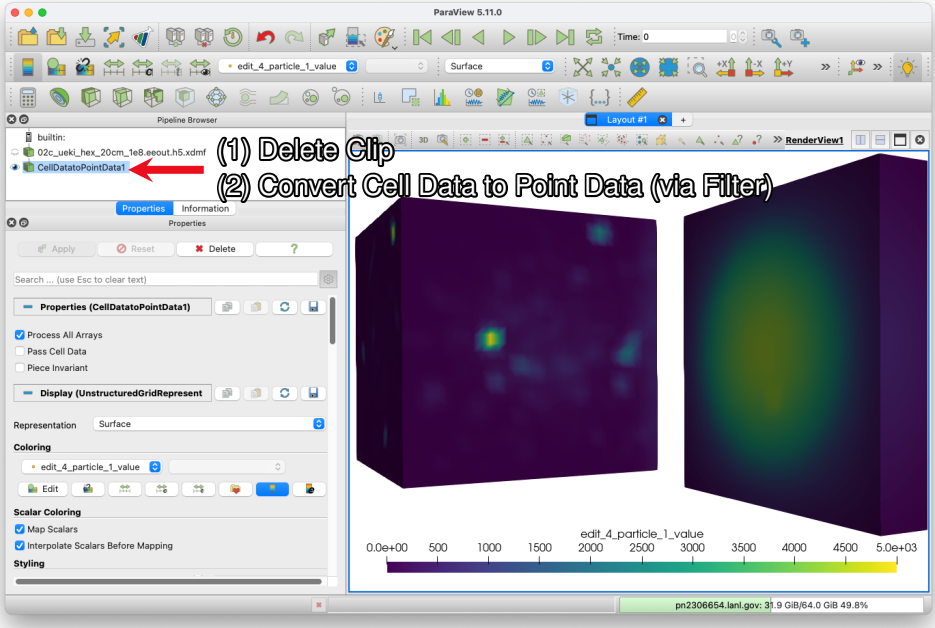
Modify Coloring and Add “Feature Edges”



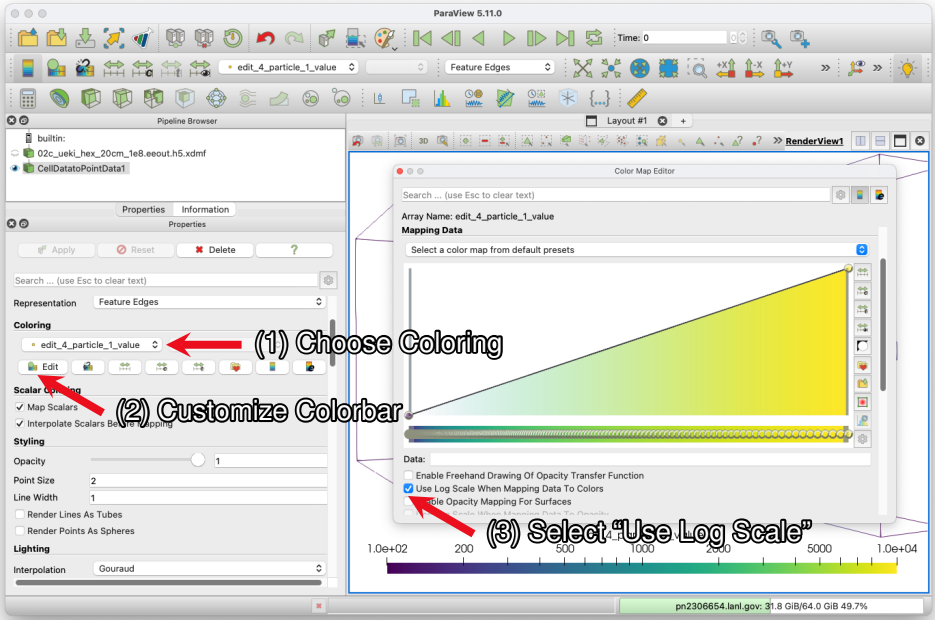
Choose Color Map



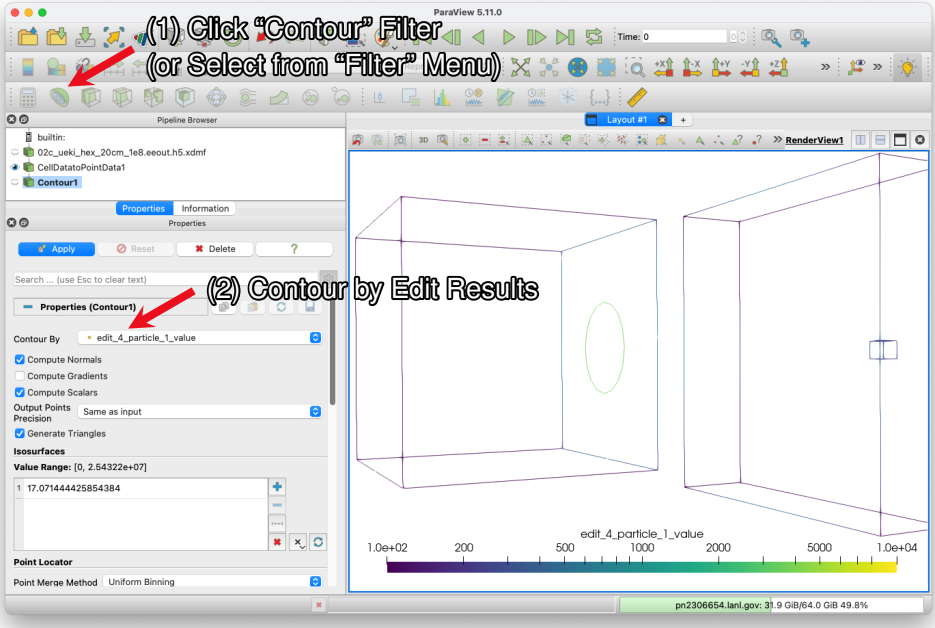
Convert Cell Data to Point Data



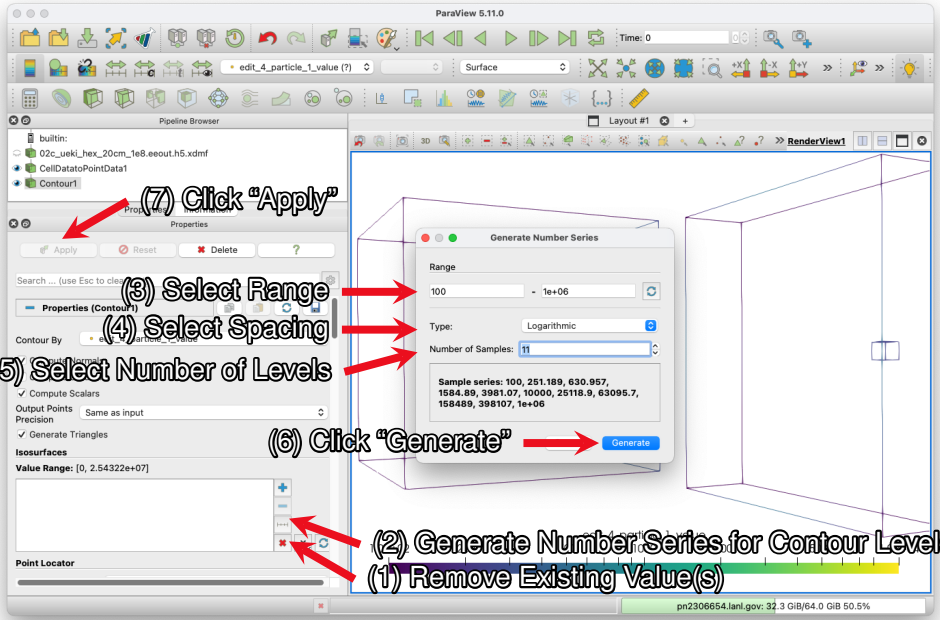
Use Logarithmic Colorbar Scaling



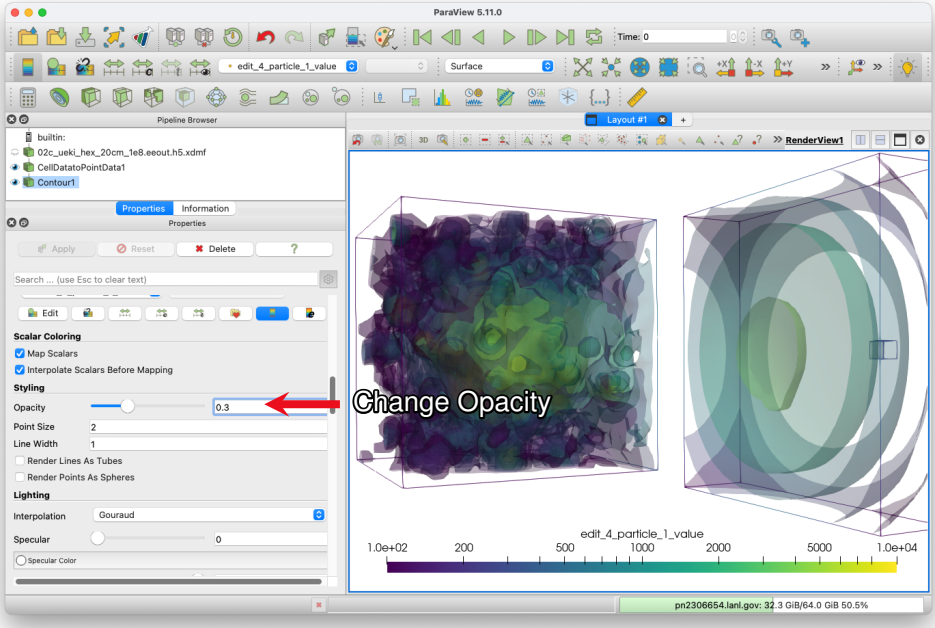
Apply Contour Filter and "Contour By" Value



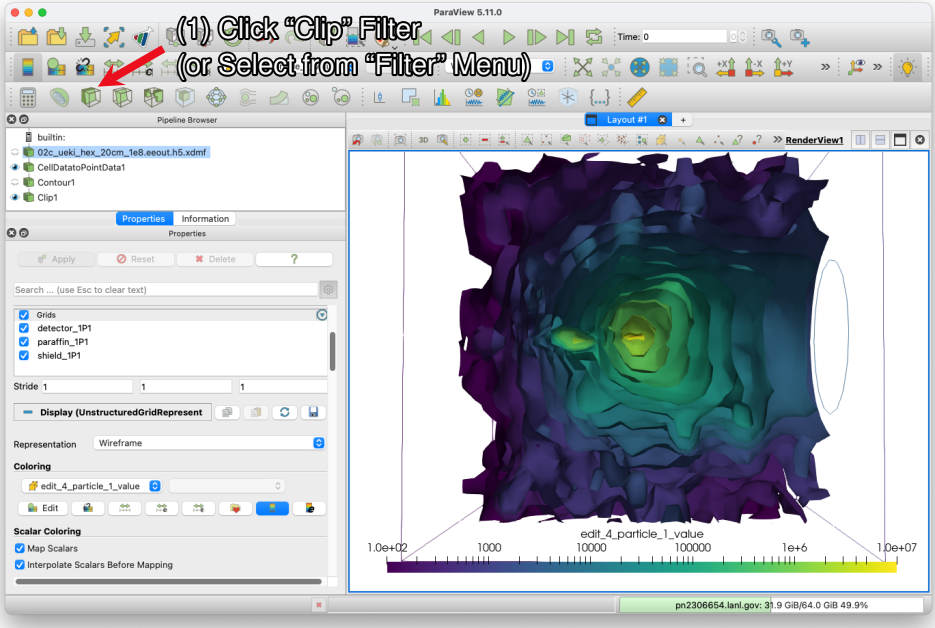
Choose Isocontour Values



Modify Opacity

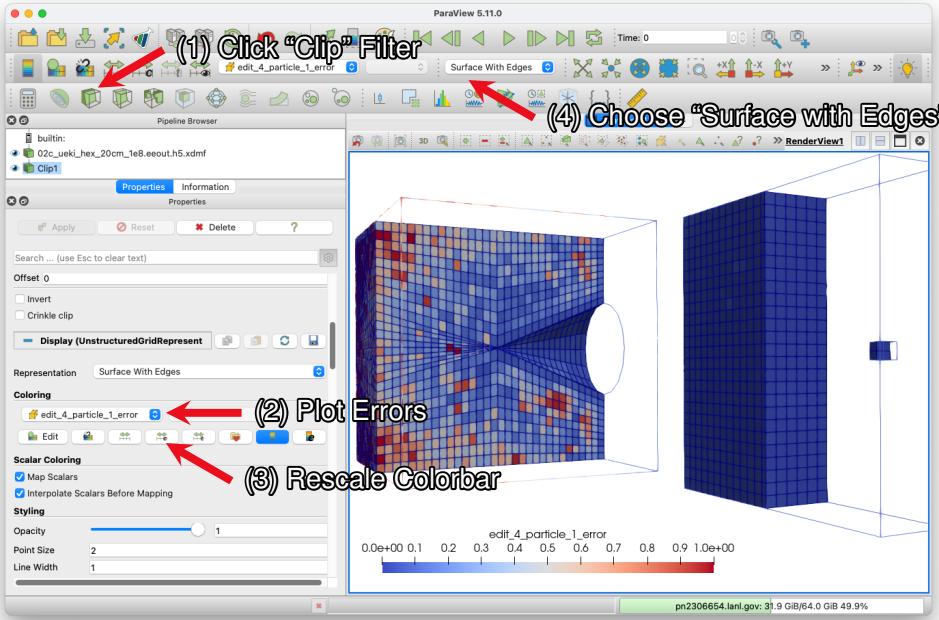


Clip a Contour



Plotting Statistical Uncertanties

View Fluence Relative Uncertainties



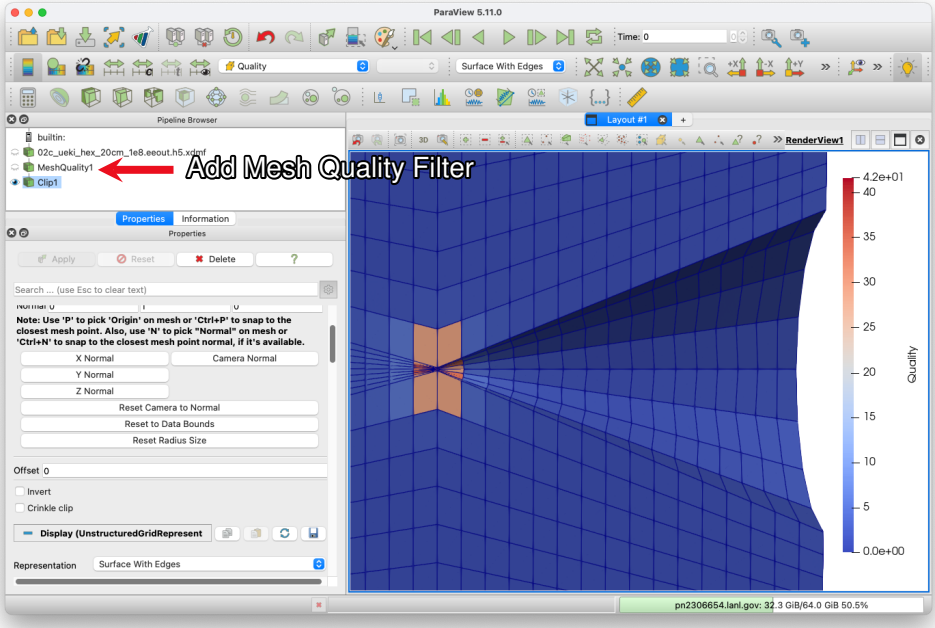
Inspecting Mesh Quality

What is “Mesh Quality”

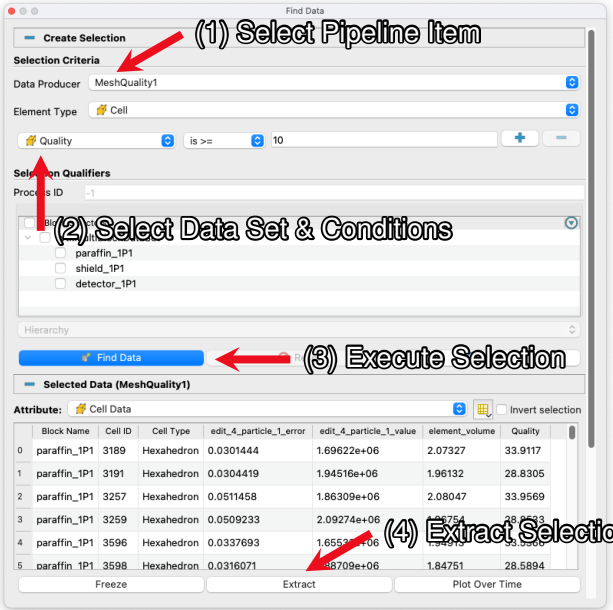
MCNP MONTE CARLO N-PARTICLE

- Mesh element “quality” can be assessed by a variety of metrics
  - Deterministically calculated
  - Useful for mesh-generation algorithms
  - Useful for deterministic engineering calculations
- Body of knowledge mainly for linear tetrahedra and hexahedra
  - Verdict library [7] summarizes these metrics
  - ParaView incorporates Verdict
- Useful for Monte Carlo transport to identify undesirable elements
  - Negative-Jacobian elements have negative volume
    - Problematic for edits
  - “Paper-thin” elements can challenge particle tracking

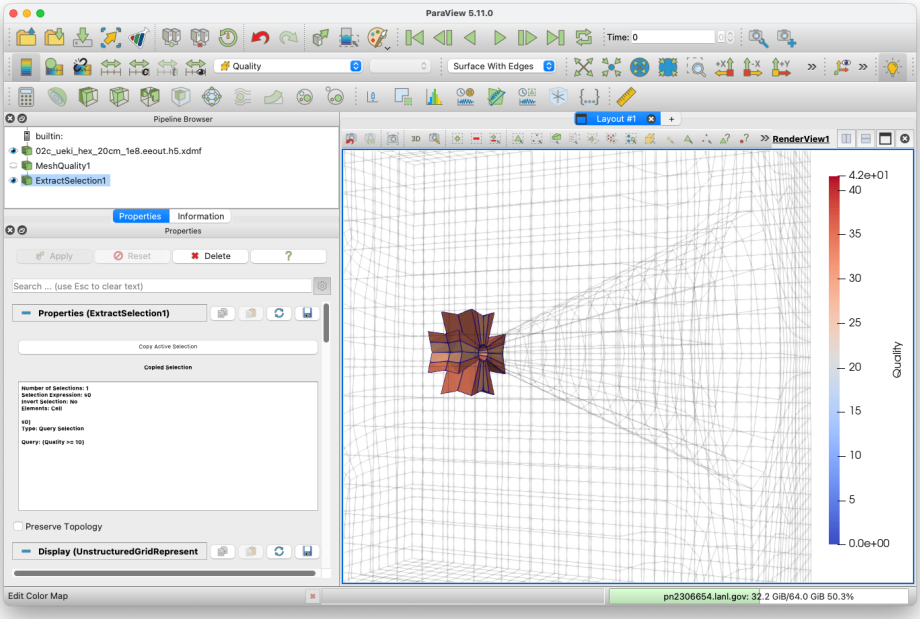
Mesh Quality Filter & Finding Elements



Find Data Dialog Use



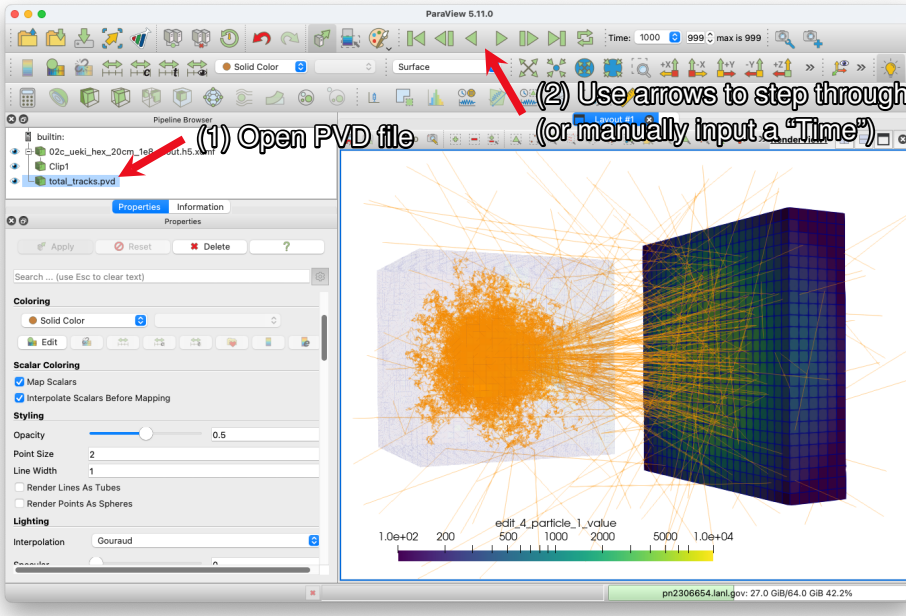
Viewing & Interrogating Specific Elements



Information on the extracted data is in the Properties pane. One can also hover on points to get location information.

ParaView Data (PVD) Files

ParaView Data (PVD) Files



Questions?

MCNP
MONTE CARLO N-PARTICLE

Introduction to XDMF

ParaView Overview

Ueki Example

Plotting Statistical Uncertanties

Inspecting Mesh Quality

ParaView Data (PVD) Files

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] U. Ayachit, The ParaView Guide, community ed., L. Avila, K. Osterdahl, S. McKenzie, and S. Jordan, Eds. Kitware Inc., Jun. 2018.

[2] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil, "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data," in High Performance Visualization–Enabling Extreme-Scale Scientific Insight, Oct. 2012, pp. 357–372.

[3] J. A. Clarke and E. R. Mark, "Enhancements to the eXtensible Data Model and Format (XDMF)," in HPCMP User's Group Conference 2007. High Performance Computing Modernizatio Pittsburgh, PA, USA; June 18–21, 2007, pp. 322–327.

[4] "XDMF Model and Format," Website, Apr. 2020. URL: http://xdmf.org/index.php/XDMF_Model_and_Format

References

[5] J. A. Kulesza and T. C. McClanahan, “A Python Script to Convert MCNP Unstructured Mesh Elemental Edit Output Files to XML-based VTK Files,” Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-19-20291, rev. 2, Nov. 2019. DOI: 10.2172/1574726

[6] J. A. Kulesza, J. L. Alwin, J. D. Hutchinson, E. F. Shores, and R. C. Little, “l3d2vtk: An MCNPTools Utility to Enable LNK3DNT File Visualization & Post-processing,” in Transactions of the Winter 2019 American Nuclear Society Meeting, vol. 121, no. 1, Nov. 2019, pp. 1233–1236, Los Alamos National Laboratory Tech. Rep. LA-UR-19-24947. DOI: 10.13182/T30909

[7] C. J. Stimpson, C. D. Ernst, P. K. and. P. P. Pébay, and D. Thompson, “The Verdict Geometric Quality Library,” Sandia National Laboratories, Albuquerque, NM, USA, Tech. Rep. SAND2007-1751, Mar. 2007. DOI: 10.2172/901967

References

[8] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.0 Theory & User Manual," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

Los Alamos NATIONAL LABORATORY

MCNP MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-06: Visualizing Particle Events and Tracks via PTRAC Output

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

What is PTRAC?

MCNP MONTE CARLO N-PARTICLE

- PTRAC == Particle Track Output
- An optional output that lists discrete information about individual events that particles undergo within individual histories of a MCNP simulation
- The PTRAC input card controls the creation and contents of the output
  - Output can become exceedingly large
  - Both event- and history-based filters can be applied to limit data written to disk
- The PTRAC file contains the particle state and event data
  - Event types include source, collision, surface crossing, bank, and termination events
  - Each event includes unique information tailored to the type
  - Particle state includes information such as position, direction, energy, time, weight, etc.
- Given the PTRAC data represents nearly everything that a particle undergoes during its lifetime, a wealth of opportunities exist in making use of this data

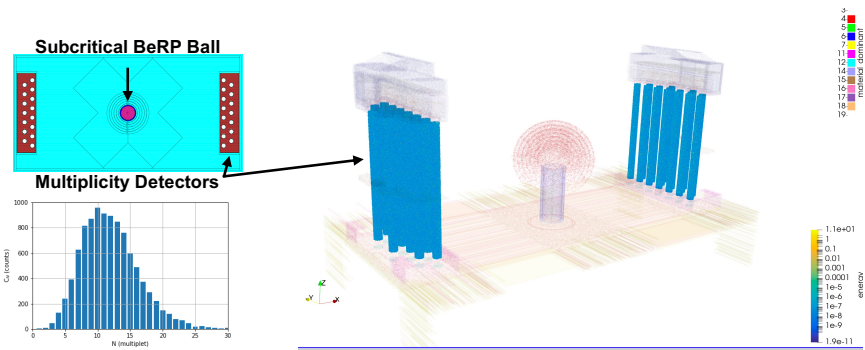
PTRAC Event Types

- Source events
  - In fixed-source calculations, the particle information of user-defined source events (i.e., sampling from SDEF)
  - In k-eigenvalue calculations, the particle information of fission bank events at the beginning of each cycle (or initial source guess in first cycle)
- Collision events
  - Record of particle state after a physical collision
  - Only record collision types being simulated (i.e., no capture events occur with implicit capture turned on)
- Surface crossing events
  - Record of particle state after crossing a surface
- Bank events
  - Secondary particle bank events; not from k-eigenvalue calculation fission bank
  - Record when particle is pulled from bank; order is last in first out
  - Can be from physical collisions or variance reduction events
- Termination events
  - When particle is killed (e.g., collision, zero importance region, roulette)

PTRAC Applications

- PTRAC is often used for advanced detector responses, where more detailed, correlated, or time-dependent analysis is needed
- The PTRAC file is used as input for custom post-processing software
  - Examples include advanced detector response simulation framework DRiFT [1]
  - Subcritical neutron multiplicity and neutron noise calculations
  - ...and more

Example of Subcritical Multiplication Analysis and Visualization using PTRAC



HDF5 PTRAC in MCNP6.3 and later

MCNP
MONTE CARLO N-PARTICLE

- Prior to MCNP6.3, PTRAC only available in ASCII and Fortran binary formats
  - Limited MCNP6 to sequential calculations only
  - Custom tools needed to read particle track data
- HDF5 PTRAC simulations can be executed in parallel
  - Removes a significant computational bottleneck
  - Even in serial, HDF5 PTRAC is faster for large problems
- Organized output structure makes post-processing more accessible
  - Reduces processing errors of legacy formats
  - More flexible, so it can be extended in the future
- The MCNP6.3 release notes provide more detail on the feature
  - Several PTRAC bug fixes
  - Legacy formats and two infrequently used features are DEPRECATED
  - Improved interface for event-wise cell and surface features

HDF5 PTRAC Easily Interrogated

MCNP MONTE CARLO N-PARTICLE

- New HDF5 PTRAC file format described in MCNP6.3 user manual

```
/ ..... (root)
├── config_control ..... (group)
├── problem_info ..... (group)
└── ptrack ..... (group)
    ├── RecordLog ..... (dataset)
    ├── Bank ..... (dataset)
    ├── Collision ..... (dataset)
    ├── Source ..... (dataset)
    ├── SurfaceCrossing ..... (dataset)
    └── Termination ..... (dataset)
```

- Each group (e.g., `/ptrack/`) is like a filesystem directory
- Each dataset (e.g., `Bank`) is just an array of data that can be processed
- Interrogate the HDF5 file from the command line:
  - **`h5ls`** and **`h5dump`** for terminal usage

```
h5ls -r sf_ptrac_mcnp63.p.h5
/                        Group
/config_control          Group
/problem_info            Group
/ptrack                  Group
/ptrack/Bank             Dataset {0/Inf}
/ptrack/Collision        Dataset {67170/Inf}
/ptrack/RecordLog        Dataset {67170/Inf}
/ptrack/Source           Dataset {0/Inf}
/ptrack/SurfaceCrossing  Dataset {0/Inf}
/ptrack/Termination      Dataset {0/Inf}
```

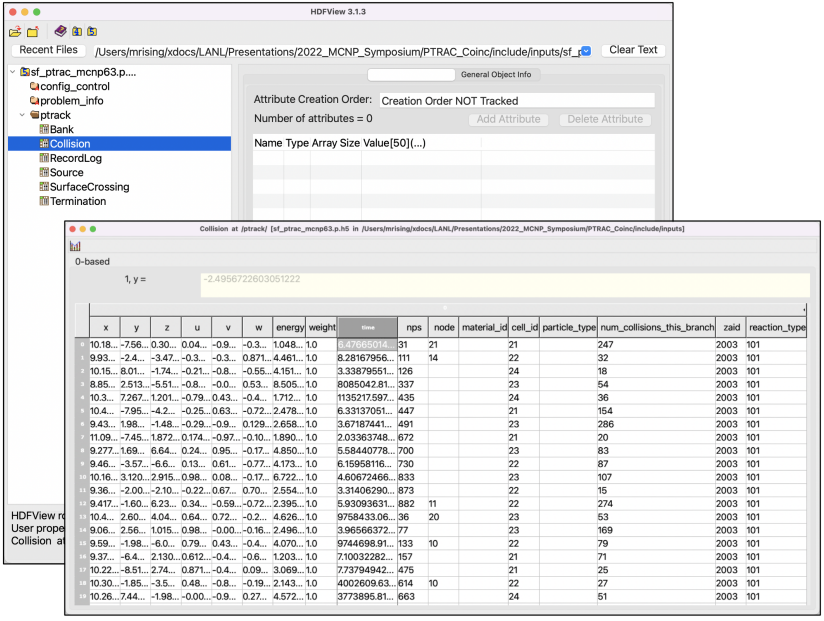
HDF5 PTRAC Compound-data Structure: Layout

MCNP MONTE CARLO N-PARTICLE

Collision event compound data type fields:

Data Field	Description
x	x coordinate of the particle position
y	y coordinate of the particle position
z	z coordinate of the particle position
u	Particle direction cosine relative to $+x$ axis
v	Particle direction cosine relative to $+y$ axis
w	Particle direction cosine relative to $+z$ axis
energy	Particle energy
weight	Particle weight
time	Time at particle position
nps	History identifier
node	Number of nodes in track from source to here
material_id	Program material ID of the cell containing event
cell_id	Problem number of the cell containing event
num_collisions_this_branch	Count of collisions per track
reaction_type	Number identifying the reaction type
zaid	**ZZZAAA** for reaction isotope
particle_type	particle type enumeration

HDF5 PTRAC Compound-data Structure: HDFView



Access to HDF5 PTRAC Data

MCNP MONTE CARLO N-PARTICLE

MCNPTools

- Version 3.8 released with MCNP6.2
- **Latest version, supports both legacy and HDF5 formats, now available as open source at https://github.com/lanl/mcnptools**
- Example below:

Direct access through HDF5 API's

- Official APIs: C, C++, Fortran, Java
- Unofficial APIs: Julia, Matlab, Mathematica, Perl, Python, R
- Python `h5py` example to follow

MCNPTools PTRAC Processing Example

Python

```python
# Print out event types, as they occurred in the code
histories = ptrac_data.ReadHistories(1000)   # load first 1000 hists
for hist in histories:                       # process each history object
    for e in range(hist.GetNumEvents()): # event loop, per history
        event = h.GetEvent(e)            # load event data
        print(event.Type())              # Prints enumeration
```

Converting PTRAC to CSV

Converting PTRAC to CSV Format

MCNP MONTE CARLO N-PARTICLE

Conversion of HDF5 PTRAC event locations made easy through the Python h5py library. Here, the resulting format is a standard Comma Separated Values (CSV) formatted file.

Listing 1: PTRAC to CSV Function

```
import h5py
import numpy as np


def to_csv(ptrack_file_name, event_type, particle_info=["x", "y", "z", "energy"], max_events=None):
    """Converts HDF5 PTRAC into a CSV file for a specific event type"""

    h5file = h5py.File(ptrack_file_name, "r")

    group = f"ptrack/{event_type}"

    if max_events is None:
        data = [h5file[group][info] for info in particle_info]
    else:
        data = [h5file[group][info][:max_events] for info in particle_info]

    data = np.vstack(data)

    np.savetxt(f"{event_type}s.csv", data.T, delimiter=",", header=",".join(particle_info))
```

Introduction to PyTrac

What is PyTrac?

- PyTrac (Python PTRAC) is a Python utility that converts the HDF5 PTRAC file into a list of trees using the `anytree` library
- The individual “particle trees” reproduce the MCNP histories as they were simulated (`KCODE` included) and can be post-processed to calculate quantities that are currently unavailable in the code

PyTrac Capabilities

- `pytrac.py`, contains the `PyTrac` class that converts the HDF5 PTRAC file to a list of “particle trees”
- `processors/`, a directory that contains post-processing utilities
  - Visualization
    - `print_trees`
    - `vtk_utils`

The `print_trees` Processor

Using `print_trees`

MCNP MONTE CARLO N-PARTICLE

Listing 2: Printing ASCII Trees of PTRAC Histories

```python
from pytrac import PyTrac
from pytrac.processors.print_trees import print_trees

# Load the ptrac.h5 file
pytrac = PyTrac('ptrac.h5')

# Print trees
print_trees(
    pytrac.tree_root_nodes,
    print_particle_fields=['event_type','reaction_type'],
    float_precision=3,
    colorama=True,
    particle_filter=lambda p: p.reaction_type == 16,
    pause=True
)
```

Example of `print_trees` Output

```
Particle Tree for nps 1:
SRC: event_type=1000, reaction_type=None
└── COL: event_type=4000, reaction_type=16 [FILTER]
    ├── COL: event_type=4000, reaction_type=2
    │   └── COL: event_type=4000, reaction_type=16 [FILTER]
    │       ├── COL: event_type=4000, reaction_type=2
    │       │   └── COL: event_type=4000, reaction_type=2
    │       │       └── COL: event_type=4000, reaction_type=2
    │       │           └── COL: event_type=4000, reaction_type=2
    │       │               └── COL: event_type=4000, reaction_type=2
    │       │                   └── COL: event_type=4000, reaction_type=2
    │       │                       └── COL: event_type=4000, reaction_type=2
    │       │                           └── SUR: event_type=3000, reaction_type=None
    │       │                               └── TER: event_type=5000, reaction_type=None
    │       └── BNK: event_type=2000, reaction_type=2
    │           └── COL: event_type=4000, reaction_type=2
    │               └── COL: event_type=4000, reaction_type=2
    │                   └── COL: event_type=4000, reaction_type=2
    │                       └── COL: event_type=4000, reaction_type=2
    │                           └── COL: event_type=4000, reaction_type=2
    │                               └── COL: event_type=4000, reaction_type=2
    │                                   └── SUR: event_type=3000, reaction_type=None
    │                                       └── TER: event_type=5000, reaction_type=None
    └── BNK: event_type=2000, reaction_type=2
        └── COL: event_type=4000, reaction_type=2
            └── COL: event_type=4000, reaction_type=2
                └── SUR: event_type=3000, reaction_type=None
                    └── TER: event_type=5000, reaction_type=None
```

Using `print_trees` with `kcode=True`

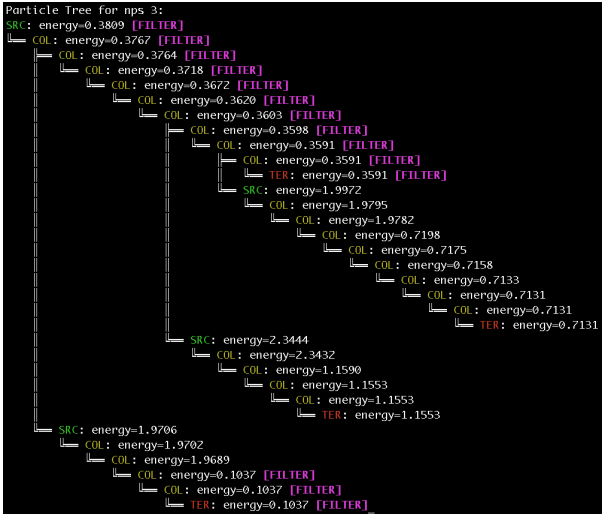
MCNP MONTE CARLO N-PARTICLE

Listing 3: Printing ASCII Trees of PTRAC k-eigenvalue Histories

```
from pytrac import PyTrac
from pytrac.processors.print_trees import print_trees

pytrac = PyTrac('ptrac.h5', kcode=True)

print_trees(
    pytrac.tree_root_nodes,
    print_particle_fields=['energy'],
    float_precision=4,
    colorama=True,
    particle_filter=lambda p: p.energy < 0.5,
    pause=True
)
```

Example Output Using `kcode=True`

MCNP
MONTE CARLO N-PARTICLE



The `vtk_utils` Processor

Introduction to `vtk_utils`

MCNP MONTE CARLO N-PARTICLE

- The `vtk_utils` module provides functions for generating:
  - `VTP`: A VTK PolyData file
    - Contains tracks for single particle history
  - `PVTP`: A Parallel VTK PolyData file
    - Contains tracks for particle histories 1 through N
    - Simply references a set of VTP files
  - `PVD`: A ParaView Data collection file
    - Used for stepping through a series of datasets/snapshots
    - Partically useful for time-dependent simulations
    - Simply references a set of files (VTP, PVTP, etc.)

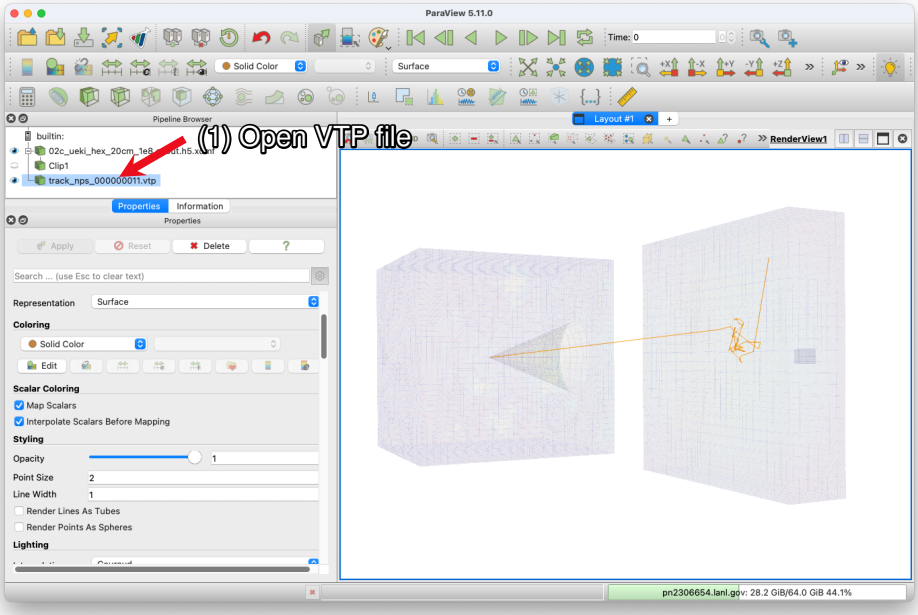
Example using `vtk_utils`

MCNP MONTE CARLO N-PARTICLE

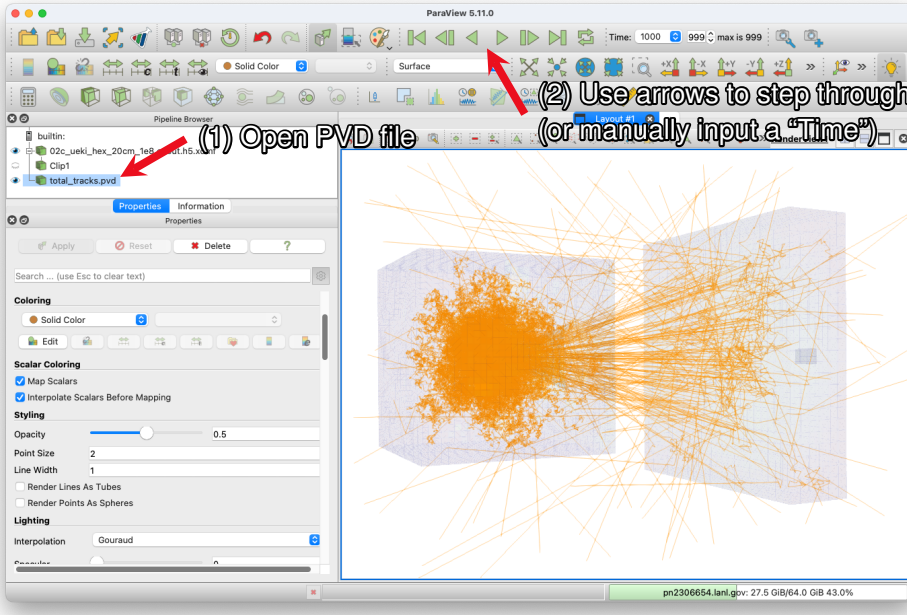
Listing 4: Creating VTP, PVTP, and PVD files of PTRAC Histories

```
from mcnp.ptrack.pytrac import PyTrac
from mcnp.ptrack.pytrac.processors import make_vtp_and_pvtp_files, make_pvd_file

# Load the ptrac.h5 file
pytrac = PyTrac('ptrac.h5', max_nps=1000)

# Create VTP and PVTP files
vtp_files, pvtp_files = make_vtp_and_pvtp_files(pytrac, "vtk_tracks")

# Create a PVD file consisting of VTP files
make_pvd_file(vtp_files, "tracks.pvd")

# Create a PVD file consisting of PVTP files
make_pvd_file(pvtp_files, "total_tracks.pvd")
```

Example of a VTP File of a Single Particle Track



Example of a PVD File of All Particle Tracks



Creating a Movie of Using `vtk_utils`

MCNP MONTE CARLO N-PARTICLE

Listing 5: Creating a GIF movie of PTRAC Histories

```
from mcnp.ptrack.pytrac import PyTrac
from mcnp.ptrack.pytrac.processors import make_vtp_and_pvtp_files, make_pvd_file
import pyvista as pv
import imageio

# Load the ptrac.h5 file and create VTP files
pytrac = PyTrac('ptrac.h5', max_nps=200)
vtp_files, pvtp_files = make_vtp_and_pvtp_files(pytrac, "vtk_tracks")

# Create a plotter and specify as camera view
pv.start_xvfb()
plotter = pv.Plotter(off_screen=True)
plotter.camera_position = [
    (0, -250, 0),  # camera location
    (0,   0,  0),  # focal point
    (0,   0,  1)   # "upwards" direction
]
images = []

# Loop through each VTP file and save a screenshot
for vtp_file in vtp_files:
    mesh = pv.read(vtp_file)
    plotter.add_mesh(mesh)
    img = plotter.screenshot()
    images.append(img)

# Write the images to a GIF file
imageio.mimsave("animated.gif", images, duration=0.05)
```

Krusty PTRAC and Python-based MCNP Project

Disclaimer for `mcnp` Python Package

- This version of the `mcnp` Python package is in early development
- The software provided here is intended to work with the given examples
  - May or may not work for all MCNP applications
  - Use at your own risk!
- A production version of the `mcnp` Python package will be released with the MCNP6.4 version (or earlier)...stay tuned!

Credit: disclaimer image generated by ChatGPT

Install `mcnp` Python Package

MCNP
MONTE CARLO N-PARTICLE

Step 1: Install Python

Step 2: [Optional] Create `venv` virtual environment

```
> python -m venv mcnp-venv
> source mcnp-venv/bin/activate
```

Step 3: Install `mcnp` Python package with `pip`

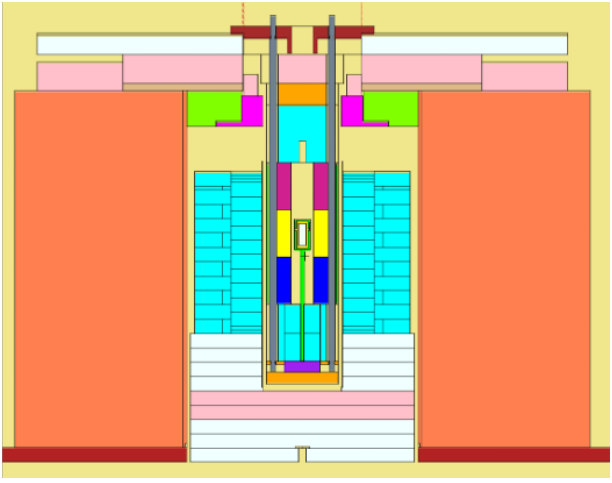
```
> pip install mcnp
```

Step 4: Test import of installed `mcnp` package

```
> python
>>> import mcnp
```

Krusty Detailed CSG Model

- Full detailed core
  - Production calculation uses 10^5 histories per cycle with 1250 cycles
  - Exercise calculation uses 10^4 histories per cycle with 150 cycles



Krusty PTRAC Points in ParaView

Run Krusty and Create PTRAC to CSV Files

MCNP MONTE CARLO N-PARTICLE

Step 1: Run Krusty model with MCNP6

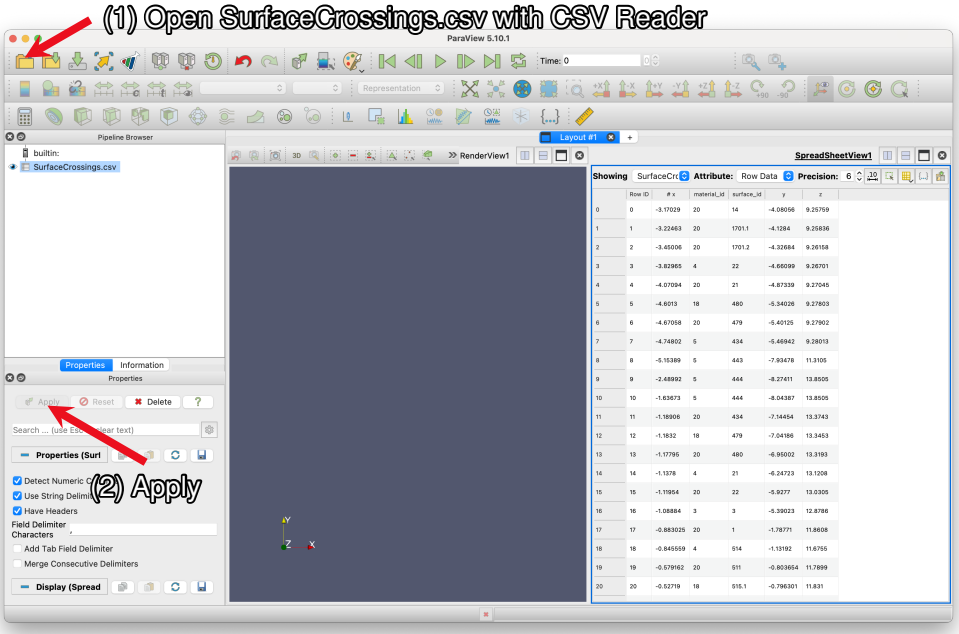
```
> mcnp6 i=krusty_csg_detailed.mcnp.inp tasks 8
```

Step 2: Run `to_csv.py.txt` script

Listing 6: PTRAC to CSV Script

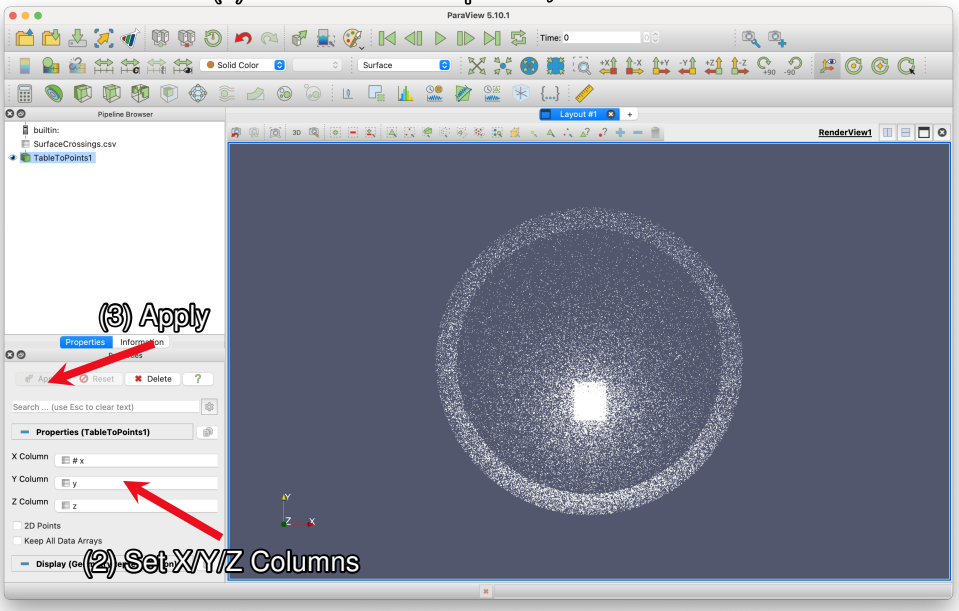
```
#!/usr/bin/env python3
import mcnp.ptrack as ptrack

# Converts ptrac.h5 surface crossing data to SurfaceCrossings.csv file
ptrack.to_csv(
    "ptrac.h5",
    "SurfaceCrossing",
    particle_info=["x", "y", "z", "surface_id", "material_id"],
    max_events=10000000,
)

# Converts ptrac.h5 source data to Sources.csv file
ptrack.to_csv("ptrac.h5", "Source", particle_info=["x", "y", "z", "energy"])
```

Krusty Surface Crossing CSV in ParaView

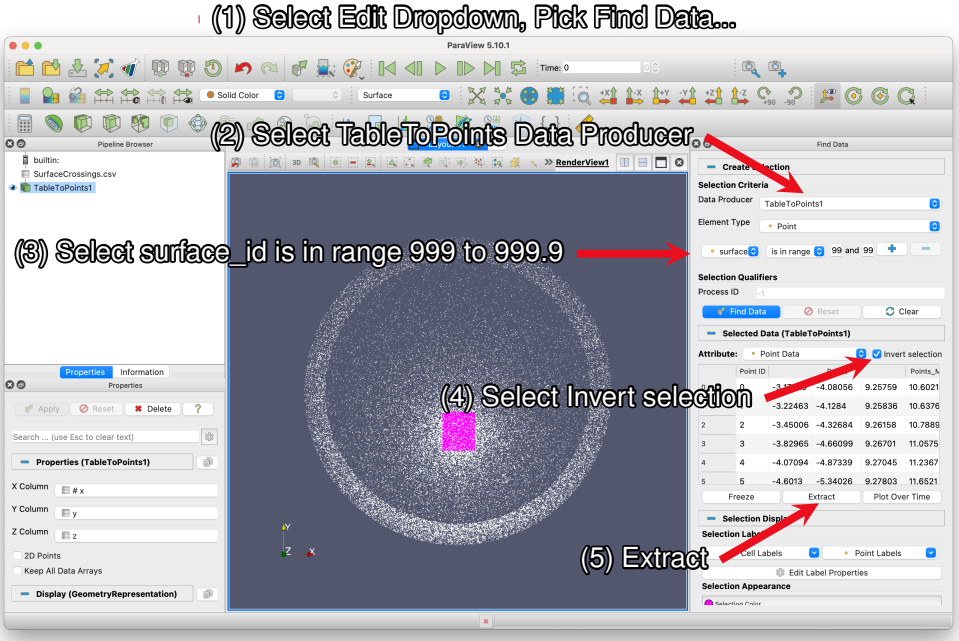


Krusty Create Points in ParaView

(1) Select Filters Dropdown, Pick Table To Points

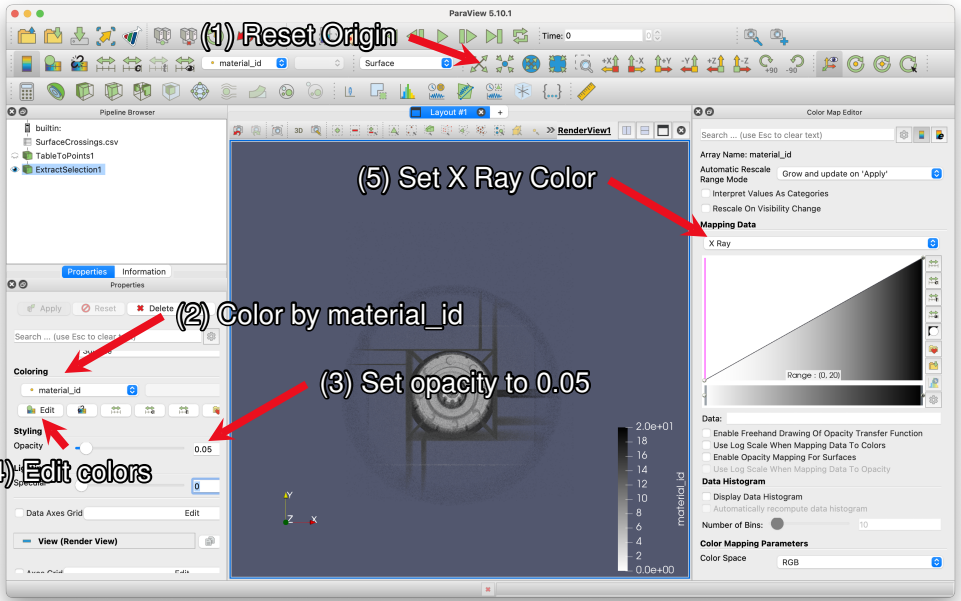


Krusty Remove Boundary Points in ParaView



Krusty Color Materials on Surfaces in ParaView

MCNP
MONTE CARLO N-PARTICLE

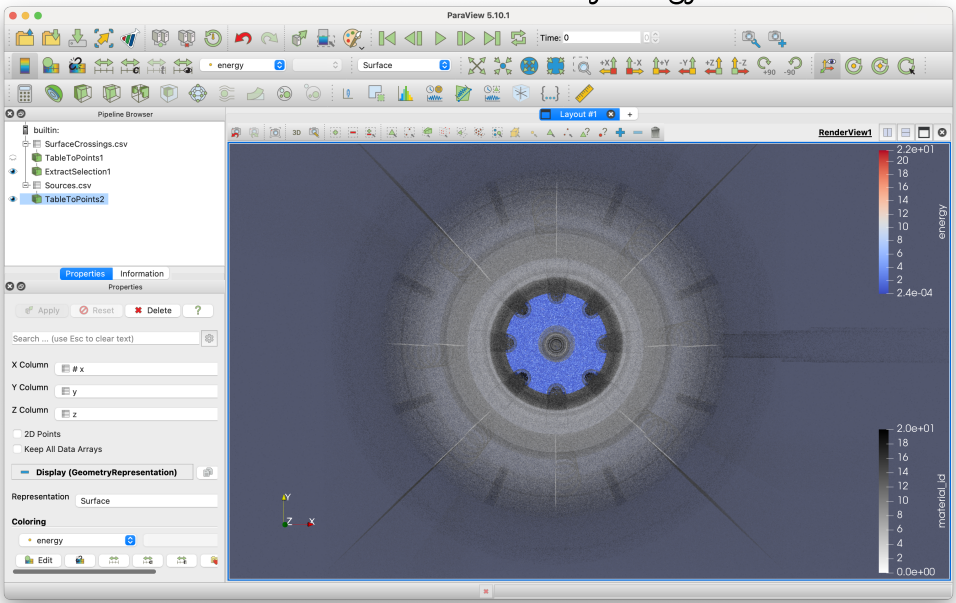


Los Alamos
NATIONAL LABORATORY

Krusty Add Source Points by Energy in ParaView

MCNP
MONTE CARLO N-PARTICLE

Add Source Points and Color by Energy



Krusty PTRAC Tracks in ParaView

Update Krusty and Create PYTRAC Files

MCNP MONTE CARLO N-PARTICLE

Step 1: Update Krusty model; remove SDEF and update KCODE card to:

```
kcode  100  1.0   0   50
```

Step 2: Rename `ptrac.h5` file and run Krusty model using existing SRCTP as initial guess

```
> mv ptrac.h5 initial_ptrac.h5
> mcnp6 i=krusty_csg_detailed.mcnp.inp src=srctp tasks 8
```

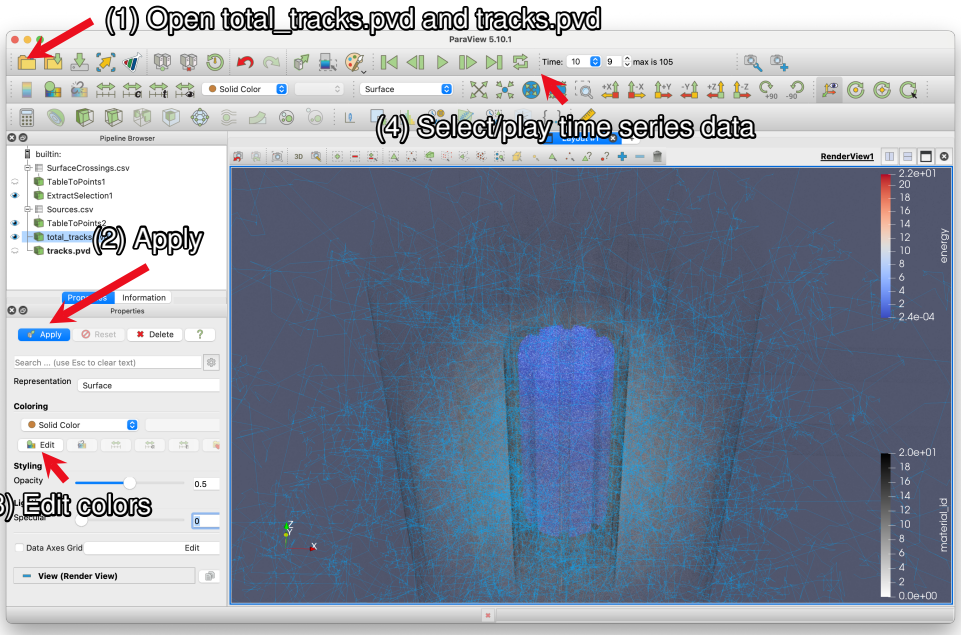
Step 3: Run `pytracs.py.txt` script

Listing 7: `pytracs.py` Script

```
#!/usr/bin/env python3
from mcnp.ptrack.pytrac import PyTrac
from mcnp.ptrack.pytrac.processors import make_vtp_and_pvtp_files, make_pvd_file

# Load the ptrac.h5 file
pytrac = PyTrac('ptrac.h5', kcode=True)

# Create VTP and PVTP files
vtp_files, pvtp_files = make_vtp_and_pvtp_files(pytrac, "vtk_tracks")

# Create a PVD file consisting of VTP files
make_pvd_file(vtp_files, "tracks.pvd")

# Create a PVD file consisting of PVTP files
make_pvd_file(pvtp_files, "total_tracks.pvd")
```

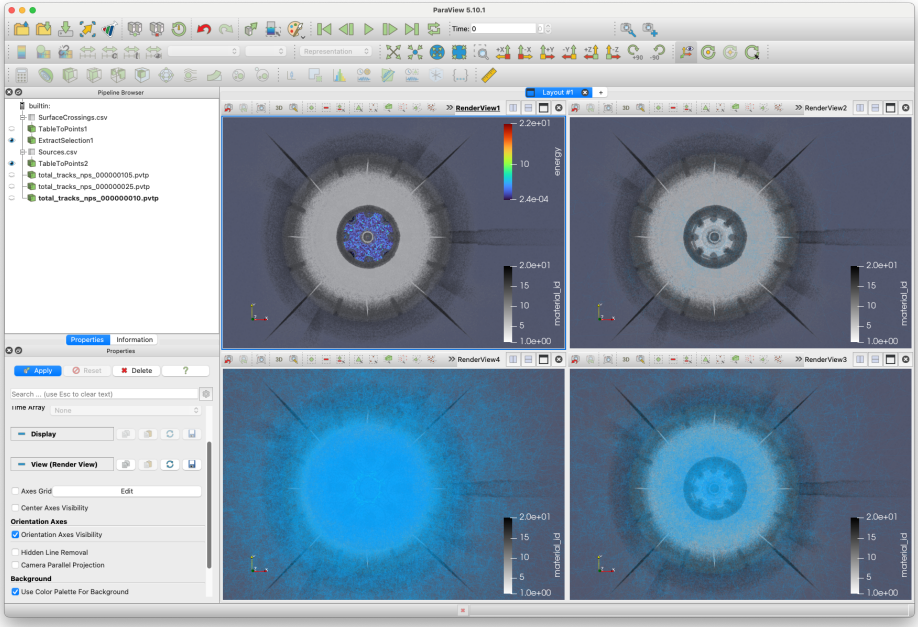
Krusty Particle Tracks from PVD files in ParaView

MCNP
MONTE CARLO N-PARTICLE



Multiple Views of CSV and PVD data in ParaView

MCNP
MONTE CARLO N-PARTICLE



Los Alamos
NATIONAL LABORATORY

Questions?

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] M. T. Andrews, C. R. Bates, E. A. McKigney, C. J. Solomon, and A. Sood, "Organic Scintillator Detector Response Simulations with DRiFT," Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 830, no. 11, pp. 466–472, Sep. 2016. DOI: 10.1016/j.nima.2016.06.011

[2] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.0 Theory & User Manual," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

Los Alamos NATIONAL LABORATORY

MCNP® MONTE CARLO N-PARTICLE

MCNP® Workshop

MCNP6 New Features, Algorithms, and Workflows

MC2025-07: Ongoing Developments for MCNP6.4

NNSA Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

MCNP® Trademark

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate.

- Please note that trademarks are adjectives and should not be pluralized or used as a noun or a verb in any context for any reason.
- Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Outline

MCNP
MONTE CARLO N-PARTICLE

Unstructured Mesh

Reactor Applications

Adjoint and Sensitivity Capabilities

Other Capabilities

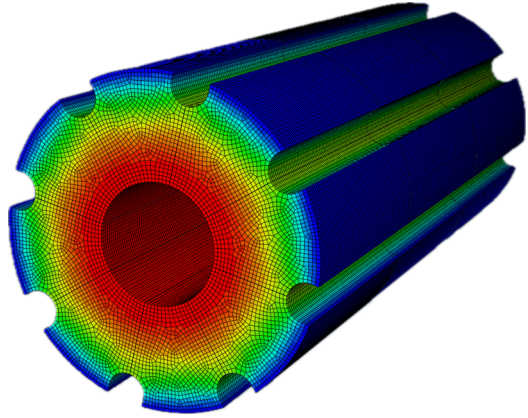
Code Modernization/Improvements for MCNP6.4

MCNP
MONTE CARLO N-PARTICLE

- Rewrite MCNP source code without changing functionality to reduce the memory usage and speed up the calculations
- Replace `um_pre_op` and `um_pos_op` with Python programs
- Verify and improve VOLUMER option
- Restart calculations without processing input models
- Remove ASCII EEOUT and GMV file writer
- Extend the UM feature for single-event electron transport
- Verification and validation for more test problems

New UM Features for MCNP6.4

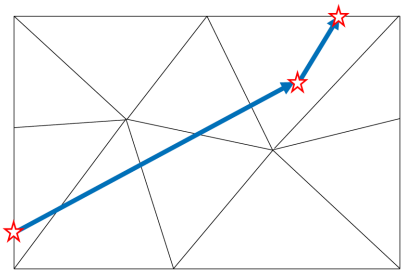
- Particle tracking for element-wise density and temperature models [1, 2]
  - Element-wise properties can only be provided via UM HDF5 file
- Cell-based elemental edit outputs
- Write/read MCNP global tracking model data to/from HDF5 files
  - Replacement of MCNPUM option in EMBED card



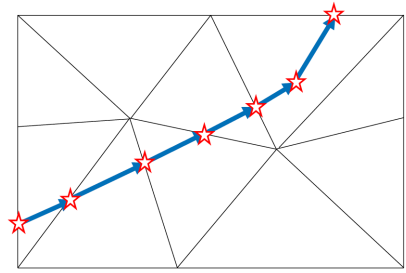
Example of an element-wise temperature distribution that can be imported in MCNP6.4 via a UM HDF5 file

Tracking in MCNP6.4 is Problem Dependent

If there **are no** element-wise differences



If there **are** element-wise differences



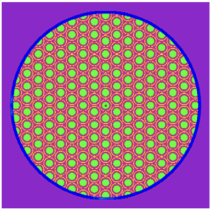
→ represents particle tracks

☆ represents material property look-ups & resampling of distance-to-collision

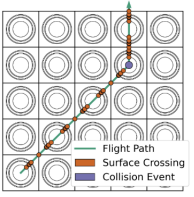
Delta Tracking in MCNP6.4

MCNP
MONTE CARLO N-PARTICLE

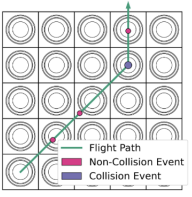
- Delta tracking (aka Woodcock tracking) is an alternative algorithm to the standard Monte Carlo surface-to-surface tracking algorithm
- Can be beneficial when collisions occur rarely with respect to the frequency that surfaces are crossed
- Implemented, tested, documented, and reviewed the Delta tracking capability – will be released in MCNP6.4
- Figures on right from Kristin Stolte ANS Summer 2023 presentation where discrete modeling of TRISO particles compared both tracking algorithms



Surface Tracking Algorithm

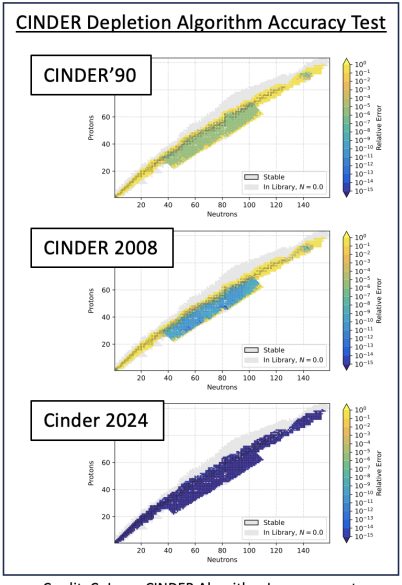


Delta Tracking Algorithm



Cinder Burn-up/Depletion in MCNP6.4

- The CINDER'90 code, used for material depletion and transmutation physics, was introduced into the MCNPX code in the early 2000's
  - Used inline for nuclear reactor depletion and activation/delayed particle emission physics
- Although it was planned, the CINDER 2008 code was never merged into MCNP6
- We are actively modernizing the code, with plans to release Cinder 2025 including improved numerical algorithms and better nuclear data



Credit: C. Josey CINDER Algorithm Improvements

Fixed-source Adjoint-weighted Sensitivity Tallies

New FSEN card under development for MCNP6.4

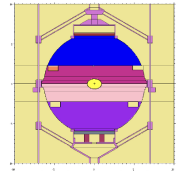
- Advanced tools for nuclear experiment design and nuclear data improvements
- Improved understanding of nuclear data library differences
- Rapid uncertainty quantification
- Nuclear data adjustment/optimization
- Application similarity assessment
- Experiment design optimization

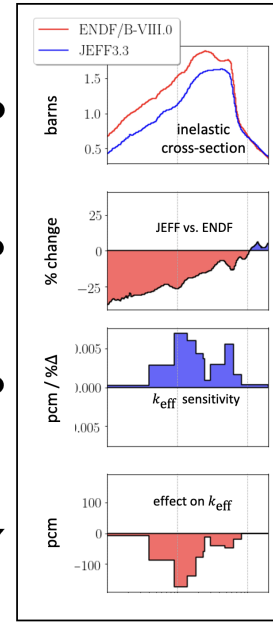
ENDF B-VIII.0

Joint Evaluated Fission and Fusion (JEFF) Nuclear Data Library Version 3.3

MCNP MONTE CARLO N-PARTICLE

Jezebel Critical Experiment





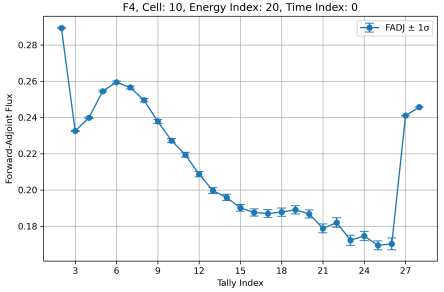
Generalized Continuous-energy Adjoint Tallies

- FADJ works with the F-series of tallies

```
F4:n 10
FM4 -1.0 1000 1
E4 0 25i 20
```

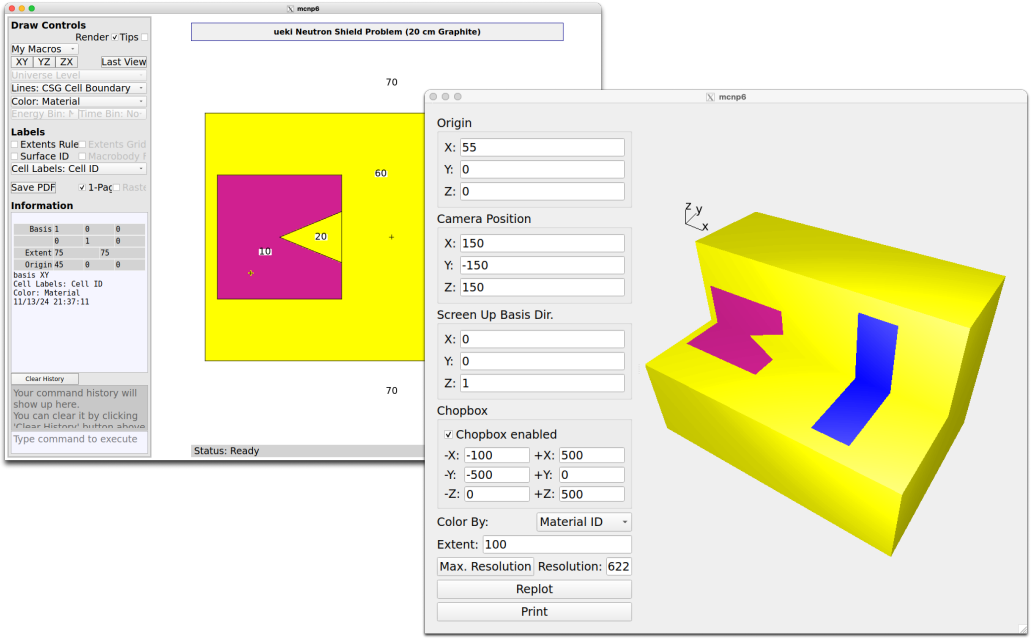
- The FADJ card will have the same functionality as WWG, WWGE, and WWGT combined into one card

```
FADJ4 CELL=10
      ENERGY=0 1 2 ...  20
      TIME=
```



Interpret as adjoint flux in CELL 10 and ENERGY 19-20 MeV from F4 tally with FM4 multiplier for each E4 energy bin.

Qt Plotter Preview: Raytracing (in 6.4?)



Other Changes to Expect in MCNP6.4 (or Earlier)

MCNP® MONTE CARLO N-PARTICLE

- Generalized ZAID-specifier input processing (in MCNP6.3.1)
- New random number generator (see `RAND GEN=8` in MCNP6.3.1)
- Extended general sensitivity/uncertainty tally capabilities (`FSEN`)
- Completion of `FMESH` improvements to supplant `TMESH` capabilities
- Dynamically-linked source and tally capabilities
- More output data converted into HDF5-accessible formats
- And many more code enhancements, bug fixes, etc.
- For annual updates on MCNP6 code developments, consider attending our MCNP User Symposium (https://mcnp.lanl.gov/symposia.html)

Questions?

MCNP
MONTE CARLO N-PARTICLE

Unstructured Mesh

Reactor Applications

Adjoint and Sensitivity Capabilities

Other Capabilities

Backup Slides

Outline

MCNP
MONTE CARLO N-PARTICLE

References

References

[1] P. A. Vaquer, M. E. Rising, J. A. Kulesza, and C. A. Weaver, “Implementation and Verification of Element-Wise Density and Temperature Specifications in MCNP6 Unstructured Mesh Simulations,” in Transactions of the American Nuclear Society, vol. 130, no. 1. Las Vegas, NV, USA; June 16–19: American Nuclear Society, 2024, pp. 479–482, Los Alamos National Laboratory Tech. Rep. LA-UR-24-21108. URL: https://www.ans.org/pubs/transactions/article-56046

[2] P. A. Vaquer, M. E. Rising, J. A. Kulesza, and C. A. Weaver, “Implementation and Verification of Element-Wise Density and Temperature Specifications in MCNP6 Unstructured Mesh Simulations,” Los Alamos National Laboratory, Los Alamos, NM, USA, Presentation LA-UR-24-25512, Rev. 1, Jun. 2024. DOI: 10.2172/2426605

References

[3] J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster, III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon, Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, and A. J. Zukaitis, "MCNP® Code Version 6.3.0 Theory & User Manual," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep. LA-UR-22-30006, Rev. 1, Sep. 2022. DOI: 10.2172/1889957

1 mcnp

This is a Python package hosting a collection of tools to support the MCNP6 radiation transport code developed by the development team at LANL.

1.1 disclaimer

This is an alpha version of the tools with no warranty or support provided.

1.2 contents

The contents of the source are as follows:

```
|-- src
|   |-- mcnp
|       |-- __init__.py
|       |-- ptrack
|       |   |-- __init__.py
|       |   |-- ptrack.py
|       |   |-- pytrac
|       |       |-- __init__.py
|       |       |-- processors
|       |       |   |-- __init__.py
|       |       |   |-- print_trees.py
|       |       |   |-- vtk_utils.py
|       |       |-- pytrac.py
```

```
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
def lazy_import(module_name, submodules, submod_attrs):
    import importlib
    import os

    name_to_submod = {func: mod for mod, funcs in submod_attrs.items() for func in funcs}

    def __getattr__(name):
        if name in submodules:
            attr = importlib.import_module(
                "{module_name}.{name}".format(module_name=module_name, name=name)
            )
        elif name in name_to_submod:
            submodname = name_to_submod[name]
            module = importlib.import_module(
                "{module_name}.{submodname}".format(module_name=module_name, submodname=submodname)
            )
            attr = getattr(module, name)
        else:
            raise AttributeError(
                "No {module_name} attribute {name}".format(module_name=module_name, name=name)
            )
        globals()[name] = attr
        return attr

    if os.environ.get("EAGER_IMPORT", ""):
        for name in submodules:
            __getattr__(name)

        for attrs in submod_attrs.values():
            for attr in attrs:
                __getattr__(attr)
    return __getattr__


__getattr__ = lazy_import(
    __name__,
    submodules={
        "ptrack",
        "pytrac",
    },
    submod_attrs={
        "ptrack": [
            "to_csv",
        ],
        "pytrac": [
            "LargeRecursionLimit",
            "Particle",
            "PyTrac",
            "make_pvd_file",
            "make_vtp_and_pvtp_files",
            "make_vtp_files",
            "print_trees",
            "processors",
```

```
            "pytrac",
            "vtk_utils",
            "vtp_files_to_pvtp",
        ],
    },
)


def __dir__():
    return __all__


__all__ = [
    "LargeRecursionLimit",
    "Particle",
    "PyTrac",
    "make_pvd_file",
    "make_vtp_and_pvtp_files",
    "make_vtp_files",
    "print_trees",
    "processors",
    "ptrack",
    "pytrac",
    "to_csv",
    "vtk_utils",
    "vtp_files_to_pvtp",
]
```

```
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
"""mcnp package with utilities to support using the MCNP6 code."""


def lazy_import(module_name, submodules, submod_attrs):
    import importlib
    import os

    name_to_submod = {func: mod for mod, funcs in submod_attrs.items() for func in funcs}

    def __getattr__(name):
        if name in submodules:
            attr = importlib.import_module(
                "{module_name}.{name}".format(module_name=module_name, name=name)
            )
        elif name in name_to_submod:
            submodname = name_to_submod[name]
            module = importlib.import_module(
                "{module_name}.{submodname}".format(module_name=module_name, submodname=submodname)
            )
            attr = getattr(module, name)
        else:
            raise AttributeError(
                "No {module_name} attribute {name}".format(module_name=module_name, name=name)
            )
        globals()[name] = attr
        return attr

    if os.environ.get("EAGER_IMPORT", ""):
        for name in submodules:
            __getattr__(name)

        for attrs in submod_attrs.values():
            for attr in attrs:
                __getattr__(attr)
    return __getattr__


__getattr__ = lazy_import(
    __name__,
    submodules={
        "ptrack",
        "tests",
    },
    submod_attrs={
        "ptrack": [
            "LargeRecursionLimit",
            "Particle",
            "PyTrac",
            "make_pvd_file",
            "make_vtp_and_pvtp_files",
            "make_vtp_files",
            "print_trees",
            "processors",
```

```
            "ptrack",
            "pytrac",
            "to_csv",
            "vtk_utils",
            "vtp_files_to_pvtp",
        ],
        "tests": [
            "TestTest",
            "test_tests",
        ],
    },
)


def __dir__():
    return __all__


__all__ = [
    "LargeRecursionLimit",
    "Particle",
    "PyTrac",
    "TestTest",
    "make_pvd_file",
    "make_vtp_and_pvtp_files",
    "make_vtp_files",
    "print_trees",
    "processors",
    "ptrack",
    "pytrac",
    "test_tests",
    "tests",
    "to_csv",
    "vtk_utils",
    "vtp_files_to_pvtp",
]
```

```python
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
import h5py
import numpy as np


def to_csv(ptrack_file_name, event_type, particle_info=["x", "y", "z", "energy"], max_events=None):
    """Converts HDF5 PTRAC into a CSV file for a specific event type"""

    h5file = h5py.File(ptrack_file_name, "r")

    group = f"ptrack/{event_type}"

    if max_events is None:
        data = [h5file[group][info] for info in particle_info]
    else:
        data = [h5file[group][info][:max_events] for info in particle_info]

    data = np.vstack(data)

    np.savetxt(f"{event_type}s.csv", data.T, delimiter=",", header=",".join(particle_info))
```

```
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
def lazy_import(module_name, submodules, submod_attrs):
    import importlib
    import os

    name_to_submod = {func: mod for mod, funcs in submod_attrs.items() for func in funcs}

    def __getattr__(name):
        if name in submodules:
            attr = importlib.import_module(
                "{module_name}.{name}".format(module_name=module_name, name=name)
            )
        elif name in name_to_submod:
            submodname = name_to_submod[name]
            module = importlib.import_module(
                "{module_name}.{submodname}".format(module_name=module_name, submodname=submodname)
            )
            attr = getattr(module, name)
        else:
            raise AttributeError(
                "No {module_name} attribute {name}".format(module_name=module_name, name=name)
            )
        globals()[name] = attr
        return attr

    if os.environ.get("EAGER_IMPORT", ""):
        for name in submodules:
            __getattr__(name)

        for attrs in submod_attrs.values():
            for attr in attrs:
                __getattr__(attr)
    return __getattr__


__getattr__ = lazy_import(
    __name__,
    submodules={
        "processors",
        "pytrac",
    },
    submod_attrs={
        "processors": [
            "make_pvd_file",
            "make_vtp_and_pvtp_files",
            "make_vtp_files",
            "print_trees",
            "vtk_utils",
            "vtp_files_to_pvtp",
        ],
        "pytrac": [
            "LargeRecursionLimit",
            "Particle",
            "PyTrac",
```

```
        ],
    },
)


def __dir__():
    return __all__


__all__ = [
    "LargeRecursionLimit",
    "Particle",
    "PyTrac",
    "make_pvd_file",
    "make_vtp_and_pvtp_files",
    "make_vtp_files",
    "print_trees",
    "processors",
    "pytrac",
    "vtk_utils",
    "vtp_files_to_pvtp",
]
```

```python
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
"""Defines Particle dataclass and PyTrac class."""

# std libs
from dataclasses import dataclass
import gzip
import pickle
import sys
from typing import Optional

# 3rd-party libs
from anytree import Node
import h5py
from tqdm import tqdm


@dataclass
class Particle:
    """The Particle dataclass has the same fields as the PTRAC events."""

    # Required - standard
    x: float
    y: float
    z: float
    u: float
    v: float
    w: float
    energy: float
    weight: float
    time: float
    nps: int
    node: int
    material_id: int
    cell_id: int
    particle_type: int
    num_collisions_this_branch: int
    # Required - additional
    event_type: int
    particle_id: int
    # Optional - depends on event_type
    zaid: Optional[int] = None  # Bank, Collision
    reaction_type: Optional[int] = None  # Bank, Collision
    bank_type: Optional[int] = None  # Bank
    source_type: Optional[int] = None  # Source
    surface_id: Optional[float] = None  # SurfaceCrossing
    surface_normal_cosine: Optional[float] = None  # SurfaceCrossing
    termination_type: Optional[int] = None  # Termination
    # Optional - additional
    progenitor_id: Optional[int] = None

    @staticmethod
    def make_particle(event_type, particle_id, event):
        """Set the required particle fields from an event."""
        particle = {
```

```
            "event_type": event_type,
            "particle_id": particle_id,
            "x": event["x"],
            "y": event["y"],
            "z": event["z"],
            "u": event["u"],
            "v": event["v"],
            "w": event["w"],
            "energy": event["energy"],
            "weight": event["weight"],
            "time": event["time"],
            "nps": event["nps"],
            "node": event["node"],
            "material_id": event["material_id"],
            "cell_id": event["cell_id"],
            "particle_type": event["particle_type"],
            "num_collisions_this_branch": event["num_collisions_this_branch"],
        }
        return particle


class LargeRecursionLimit:
    """Context manager to temporarily increase recursion limit."""

    def __enter__(self):
        """Enter LargeRecursionLimit."""
        self.old_limit = sys.getrecursionlimit()
        sys.setrecursionlimit(10**6)

    def __exit__(self, exc_type, exc_value, traceback):
        """Exit LargeRecursionLimit."""
        sys.setrecursionlimit(self.old_limit)


class PyTrac:
    """The PyTrac class loads histories from a pickle or ptrac.h5 file and builds trees."""

    def __init__(self, ptrac_file, kcode=False, max_nps=1e15):
        """Initialize PyTrac instance."""
        self.ptrac_file = ptrac_file
        self.tree_root_nodes = []
        if ptrac_file.endswith(".h5"):
            self.load_histories(kcode, max_nps)
        elif ptrac_file.endswith(".pkl.gz"):
            self.load_pickle(ptrac_file)

    def __str__(self):
        """PyTrac string."""
        return f"PyTrac object for {self.ptrac_file}"

    def pickle(self):
        """Pickle particle trees."""
        print("Saving pickle to pytrac.pkl.gz... ", end="", flush=True)
        with LargeRecursionLimit():
```

```
        with gzip.open("pytrac.pkl.gz", "wb") as f:
            pickle.dump(self.tree_root_nodes, f, protocol=pickle.HIGHEST_PROTOCOL)
    print("Pickle saved!")

def load_pickle(self, ptrac_file):
    """Load pickle."""
    print(f"Loading pickle from {ptrac_file}... ", end="", flush=True)
    with LargeRecursionLimit():
        with gzip.open(ptrac_file, "rb") as f:
            self.tree_root_nodes = pickle.load(f)
    print("Pickle loaded!")

def load_histories(self, kcode=False, max_nps=1e15):
    """Load histories from ptrac_file."""
    if kcode:
        kcode_hash_table = {}

    def build_tree(history):
        """Build tree from history."""
        nodes = {}
        for particle in history:
            node = nodes.setdefault(
                particle.particle_id, Node(particle.particle_id, particle=particle)
            )
            if kcode and particle.event_type == 4000:
                kcode_hash_table[(particle.x, particle.y, particle.z)] = node
            if particle.progenitor_id is not None:
                progenitor_node = nodes.setdefault(
                    particle.progenitor_id, Node(particle.progenitor_id)
                )
                node.parent = progenitor_node
            else:
                root_node = node
        return root_node

    with h5py.File(self.ptrac_file, "r") as ptrac:
        # ptrack datasets
        record_log = ptrac["ptrack/RecordLog"]
        bank = ptrac["ptrack/Bank"]
        collision = ptrac["ptrack/Collision"]
        source = ptrac["ptrack/Source"]
        surface_crossing = ptrac["ptrack/SurfaceCrossing"]
        termination = ptrac["ptrack/Termination"]

        # load histories using RecordLog
        history = []
        hash_table = {}
        current_nps = record_log["nps"][0]
        particle_id = -1
        progenitor_id = None
        for current_event in tqdm(
            record_log, desc="Loading", unit="events", total=len(record_log)
        ):
            # event data
```

```
event_nps = current_event["nps"]
event_type = current_event["type"]
event_array_index = current_event["event_array_index"]

if event_nps > max_nps:
    break
else:
    # update particle_id
    particle_id = particle_id + 1

    # build particle tree
    if event_nps != current_nps:  # new history detected
        self.tree_root_nodes.append(build_tree(history))
        history.clear()  # clear history
        hash_table.clear()  # clear hash table
        current_nps = event_nps  # reassign currrent_nps
        if not kcode:
            particle_id = 0  # reset particle_id

    # make particle dictionary for event_type
    if event_type == 1000:  # source
        event = source[event_array_index]
        particle = Particle.make_particle(event_type, particle_id, event)
        particle["source_type"] = event["source_type"]
    elif event_type == 2000:  # bank
        event = bank[event_array_index]
        particle = Particle.make_particle(event_type, particle_id, event)
        particle["zaid"] = event["zaid"]
        particle["reaction_type"] = event["reaction_type"]
        particle["bank_type"] = event["bank_type"]
        particle["progenitor_id"] = hash_table[
            (event["x"], event["y"], event["z"], event["time"])
        ]
    elif event_type == 3000:  # surface_crossing
        event = surface_crossing[event_array_index]
        particle = Particle.make_particle(event_type, particle_id, event)
        particle["surface_id"] = event["surface_id"]
        particle["surface_normal_cosine"] = event["surface_normal_cosine"]
        particle["progenitor_id"] = progenitor_id
    elif event_type == 4000:  # collision
        event = collision[event_array_index]
        particle = Particle.make_particle(event_type, particle_id, event)
        particle["zaid"] = event["zaid"]
        particle["reaction_type"] = event["reaction_type"]
        particle["progenitor_id"] = progenitor_id
    elif event_type == 5000:  # termination
        event = termination[event_array_index]
        particle = Particle.make_particle(event_type, particle_id, event)
        particle["termination_type"] = event["termination_type"]
        particle["progenitor_id"] = progenitor_id

    # associate space-time coordinates with particle_id
    hash_table[(particle["x"], particle["y"], particle["z"], particle["time"])] = (
        particle["particle_id"]
```

```
            )

            # add particle instance to history
            history.append(Particle(**particle))

            # update progenitor_id
            progenitor_id = particle_id

    # build last particle tree
    self.tree_root_nodes.append(build_tree(history))

if kcode:
    # loop over all root Nodes
    for root in tqdm(
        self.tree_root_nodes, desc="Linking", unit="trees", total=len(self.tree_root_nodes)
    ):
        # get (x, y, z) coordinates of root
        root_x = root.particle.x
        root_y = root.particle.y
        root_z = root.particle.z
        # set progenitor_id and parent of root from the KCODE hash table
        if (root_x, root_y, root_z) in kcode_hash_table:
            progenitor = kcode_hash_table[(root_x, root_y, root_z)]
            root.particle.progenitor_id = progenitor.particle.particle_id
            root.parent = progenitor
    # remove false tree root nodes
    self.tree_root_nodes = [node for node in self.tree_root_nodes if node.is_root]
```

```python
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
def lazy_import(module_name, submodules, submod_attrs):
    import importlib
    import os

    name_to_submod = {func: mod for mod, funcs in submod_attrs.items() for func in funcs}

    def __getattr__(name):
        if name in submodules:
            attr = importlib.import_module(
                "{module_name}.{name}".format(module_name=module_name, name=name)
            )
        elif name in name_to_submod:
            submodname = name_to_submod[name]
            module = importlib.import_module(
                "{module_name}.{submodname}".format(module_name=module_name, submodname=submodname)
            )
            attr = getattr(module, name)
        else:
            raise AttributeError(
                "No {module_name} attribute {name}".format(module_name=module_name, name=name)
            )
        globals()[name] = attr
        return attr

    if os.environ.get("EAGER_IMPORT", ""):
        for name in submodules:
            __getattr__(name)

        for attrs in submod_attrs.values():
            for attr in attrs:
                __getattr__(attr)
    return __getattr__


__getattr__ = lazy_import(
    __name__,
    submodules={
        "print_trees",
        "vtk_utils",
    },
    submod_attrs={
        "print_trees": [
            "print_trees",
        ],
        "vtk_utils": [
            "make_pvd_file",
            "make_vtp_and_pvtp_files",
            "make_vtp_files",
            "vtp_files_to_pvtp",
        ],
    },
)
```

```
def __dir__():
    return __all__


__all__ = [
    "make_pvd_file",
    "make_vtp_and_pvtp_files",
    "make_vtp_files",
    "print_trees",
    "vtk_utils",
    "vtp_files_to_pvtp",
]
```

```python
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
# std libs
import sys

# 3rd-party libs
from anytree import RenderTree
from anytree.render import DoubleStyle
from colorama import Fore, Style


def print_trees(
    tree_root_nodes,
    print_particle_fields=["particle_id", "progenitor_id"],
    float_precision=5,
    colorama=False,
    particle_filter=None,
    pause=False,
):
    """
    Print Particle trees.

    Arguments:
    - print_particle_fields: The fields from the Particle dataclass to print.
    - float_precision: The float fields precision.
    - colorama: Set to True to highlight the Particle tree.
    - particle_filter: A lambda function that filters Particle fields.
    - pause: Set to True to pause printing after every tree.
    """
    for idx, root in enumerate(tree_root_nodes):
        print(f"\nParticle Tree for nps {root.particle.nps}:")
        for pre, _, node in RenderTree(root, style=DoubleStyle):
            particle = node.particle
            particle_fields = ", ".join(
                (
                    f"{field}={getattr(particle, field, 'N/A'):.{float_precision}f}"
                    if isinstance(getattr(particle, field, "N/A"), float)
                    else f"{field}={getattr(particle, field, 'N/A')}"
                )
                for field in print_particle_fields
            )
            if particle.event_type == 1000:
                event_type = f"{Fore.GREEN}SRC{Style.RESET_ALL}" if colorama else "SRC"
            elif particle.event_type == 2000:
                event_type = f"{Fore.BLUE}BNK{Style.RESET_ALL}" if colorama else "BNK"
            elif particle.event_type == 3000:
                event_type = f"{Fore.CYAN}SUR{Style.RESET_ALL}" if colorama else "SUR"
            elif particle.event_type == 4000:
                event_type = f"{Fore.YELLOW}COL{Style.RESET_ALL}" if colorama else "COL"
            elif particle.event_type == 5000:
                event_type = f"{Fore.RED}TER{Style.RESET_ALL}" if colorama else "TER"
            if particle_filter is None or not particle_filter(particle):
                print(f"{pre}{event_type}: {particle_fields}")
            else:
                print(
```

```
            f"{pre}{event_type}: {particle_fields} {Fore.MAGENTA}{Style.BRIGHT}[FILTER]{Style.RESET_ALL}"
            if colorama
            else f"{pre}{event_type}: {particle_fields} [FILTER]"
        )
if pause and idx != len(tree_root_nodes) - 1:
    response = input("Press Enter to continue printing, or type 'q' to quit: ")
    sys.stdout.write("\033[F")
    sys.stdout.write("\033[K")
    if response.lower() == "q":
        break
```

```python
# Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file
# std libs
import os
import sys

# 3rd-party libs
import numpy as np
import vtk
from vtk.util.numpy_support import numpy_to_vtk


def make_vtp_files(pytrac_tree, output_dir, track_filter=None):
    """
    Converts a PyTrac tree into multiple `.vtp` files stored in a single directory.
    Each `.vtp` file contains a particle track.

    Parameters:
        pytrac_tree: A PyTrac tree object.
        output_dir (str): Directory where `.vtp` files will be saved.
        track_filter (function): Arbitrary function for filtering out particle tracks.

    Returns:
        vtp_files (list): List of `.vtp` files that were generated
    """

    sys.setrecursionlimit(10**6)

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Create list to keep track of VTP files that were generated
    vtp_files = []

    def _recursively_process_node(node, x, y, z, track_start_id, track_end_id, chain_max_id):
        """
        Recursively process the node and its children.
        """

        # Check that node_id, progenitor_id, x, y, z are already initialized to lists
        if not all(
            isinstance(variable, list) for variable in [x, y, z, track_start_id, track_end_id]
        ):
            raise TypeError("x, y, z, track_start_id, and track_end_id must all be lists.")

        # Recursively process all children of the current node
        for child in node.children:

            # Increment id value
            chain_max_id += 1
            child.particle.chain_id = chain_max_id

            # Append id for beginning and end of particle track
            if track_filter is None or track_filter(node.particle):
                track_start_id.append(node.particle.chain_id)
```

```
            track_end_id.append(child.particle.chain_id)

        # Append coordinates for the end of track
        x.append(child.particle.x)
        y.append(child.particle.y)
        z.append(child.particle.z)

        chain_max_id = _recursively_process_node(
            child, x, y, z, track_start_id, track_end_id, chain_max_id
        )

    return chain_max_id

# Iterate through root nodes and save each track as a '.vtp' file
for i, root in enumerate(pytrac_tree.tree_root_nodes):

    # Initialize current chain id
    root.particle.chain_id = 0
    chain_max_id = 0

    # Initialize lists for storing coordinate data for this track.
    track_start_id = []
    track_end_id = []
    x = [root.particle.x]
    y = [root.particle.y]
    z = [root.particle.z]

    _recursively_process_node(root, x, y, z, track_start_id, track_end_id, chain_max_id)

    # Convert lists to arrays
    track_start_id = np.asarray(track_start_id)
    track_end_id = np.asarray(track_end_id)
    x = np.asarray(x)
    y = np.asarray(y)
    z = np.asarray(z)

    # Store total number of tracks
    num_tracks = len(track_start_id)

    # Create a vtkPoints object to store particle positions
    points = vtk.vtkPoints()

    # Convert the NumPy arrays to VTK data arrays
    vtk_points = numpy_to_vtk(np.column_stack((x, y, z)), deep=True)

    # Set up the vtkPoints object using the converted data
    points.SetData(vtk_points)

    # Create a vtkCellArray to store lines
    lines = vtk.vtkCellArray()

    for i in range(num_tracks):
        line = vtk.vtkLine()
        line.GetPointIds().SetId(0, track_start_id[i])
```

```
            line.GetPointIds().SetId(1, track_end_id[i])
            lines.InsertNextCell(line)

        # Create a vtkPolyData object to represent the particle data
        poly_data = vtk.vtkPolyData()
        poly_data.SetPoints(points)
        poly_data.SetLines(lines)

        # Name the vtk file using the root particle's nps value
        vtp_filename = os.path.join(output_dir, f"track_nps_{str(root.particle.nps).zfill(9)}.vtp")

        # Use vtkXMLPolyDataWriter to write the file
        writer = vtk.vtkXMLPolyDataWriter()
        writer.SetFileName(vtp_filename)
        writer.SetInputData(poly_data)
        writer.SetDataModeToAscii()

        # Write the vtkPolyData object to file
        writer.Write()

        print(f"Saved: {vtp_filename}")

        vtp_files.append(vtp_filename)

    return vtp_files


def vtp_files_to_pvtp(vtp_files, output_filename):
    """
    Generates a `.pvtp` file that consolidates a list of `.vtp` files.

    Parameters:
      vtp_files (list of str): List of paths to the `.vtp` files.
      output_filename (str): The filename for the generated `.pvtp` file.
    """

    # Begin the XML for a parallel polydata file.
    xml_str = '<?xml version="1.0"?>\n'
    xml_str += '<VTKFile type="PPolyData" version="1.0" byte_order="LittleEndian">\n'
    xml_str += '  <PPolyData GhostLevel="0">\n'

    # Define the points array.
    xml_str += "    <PPoints>\n"
    xml_str += '      <PDataArray type="Float64" NumberOfComponents="3" format="ascii"/>\n'
    xml_str += "    </PPoints>\n"

    # Add each piece reference.
    for vtp_file in vtp_files:
        xml_str += f'    <Piece Source="{vtp_file}"/>\n'

    xml_str += "  </PPolyData>\n"
    xml_str += "</VTKFile>\n"

    # Write the XML string to the output file.
```

```
    with open(output_filename, "w") as f:
        f.write(xml_str)

    print(f"Saved: {output_filename}")


def make_vtp_and_pvtp_files(pytrac_tree, output_dir, track_filter=None):
    """
    Generates `.pvtp` files and corresponding `.vtp` files.

    Parameters:
        pytrac_tree: A PyTrac tree object.
        output_dir (str): Directory where `.vtp` and `.pvtp` files will be saved.
        track_filter (function): Arbitrary function for filtering out particle tracks.
    """

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Create list to keep track of PVTP files that were generated
    pvtp_files = []

    # Create VTP files
    vtp_files = make_vtp_files(pytrac_tree, output_dir, track_filter=track_filter)

    # Obtain base names for VTP files
    vtp_file_basenames = []
    [vtp_file_basenames.append(os.path.basename(vtp_file)) for vtp_file in vtp_files]

    for nps in range(1, len(vtp_files) + 1):

        # Name the PVTP file
        pvtp_filename = os.path.join(output_dir, f"total_tracks_nps_{str(nps).zfill(9)}.pvtp")

        # Create the PVTP file
        vtp_files_to_pvtp(vtp_file_basenames[:nps], pvtp_filename)

        pvtp_files.append(pvtp_filename)

    return vtp_files, pvtp_files


def make_pvd_file(vtk_files, output_filename):
    """
    Generates a `.pvd` file from a list of `.vtp` files.

    Parameters:
      vtp_files (list of str): List of paths to the `.vtp` or `.pvtp` files.
      output_filename (str): The filename for the generated `.pvd` file.
    """

    with open(output_filename, "w") as f:
        f.write('<?xml version="1.0"?>\n')
        f.write('<VTKFile type="Collection" version="0.1" byte_order="LittleEndian">\n')
```

```
    f.write("  <Collection>\n")
    for i, vtk_filename in enumerate(vtk_files):
        f.write(f'    <DataSet timestep="{i+1}" group="" part="0" file="{vtk_filename}"/>\n')
    f.write("  </Collection>\n")
    f.write("</VTKFile>\n")

print(f"Saved: {output_filename}")
```