LA-UR-21-26363
Approved for public release; distribution is unlimited.

Title: Improved FMESH Capabilities in the MCNP 6.3 Code

Author(s): Josey, Colin James
Kulesza, Joel A.

Intended for: 2021 MCNP User Symposium, 2021-07-12/2021-07-16 (Los Alamos, New Mexico, United States)

Issued: 2021-07-06

Los Alamos
NATIONAL LABORATORY

Improved FMESH Capabilities in the MCNP® 6.3 Code

Colin Josey, Joel Kulesza

XCP-3 (Monte Carlo Codes)

2021 MCNP® User Symposium
July 12, 2021

Introduction

Since 6.2, `FMESH` has undergone a substantial revision in capabilities.

- The default `FMESH` configuration was heavily optimized.
- 3 new tally backends were added for various needs (mainly, larger tallies).
- MESHTAL is deprecated and replaced with an HDF5 + XDMF output format.
  (This format allows for much faster and easier postprocessing and analysis)

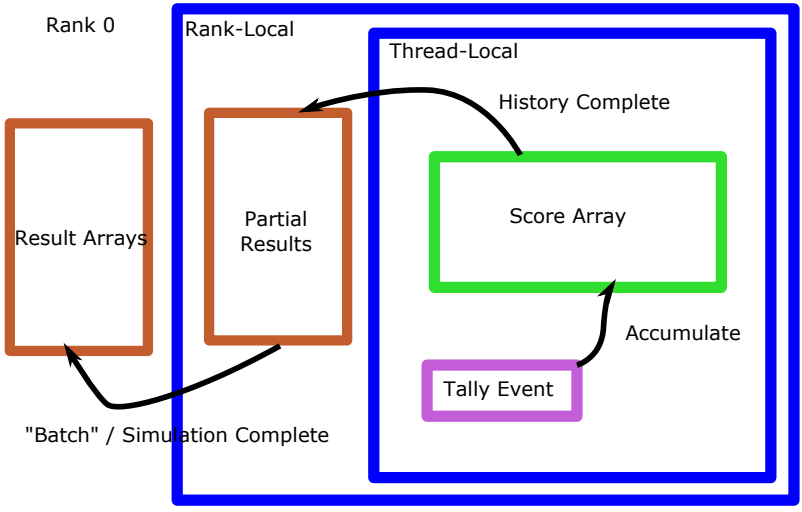
Tally Algorithms

It started as a side project - can we use MPI remote memory access to scale further than ever before?

I needed a point of comparison, so I implemented 4 algorithms:

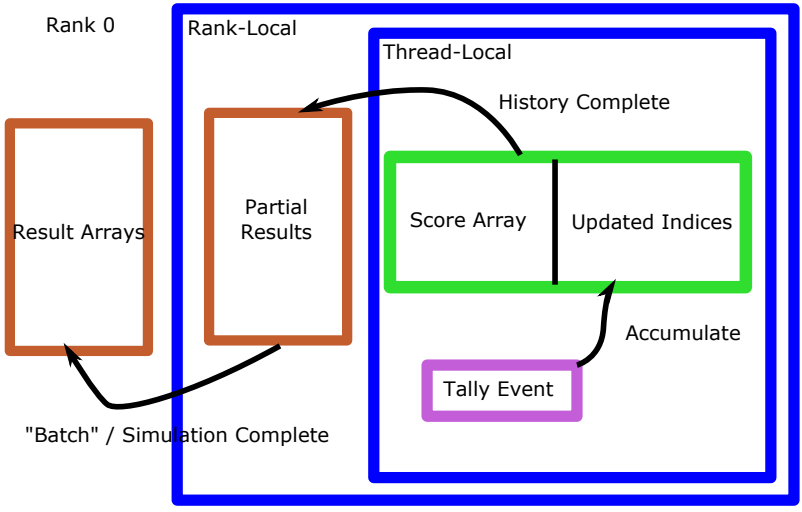
- History - basic history statistics without optimization.
- Fast History - a tuned version of 6.2's `FMESH` algorithm, tracks changed indices to reduce memory bandwidth usage.
- Batch - Threads share a tally array, so memory usage is reduced.
- Batch RMA - The Batch algorithm, but using MPI-3 RMA to distribute tallies over all MPI ranks.

History Algorithm



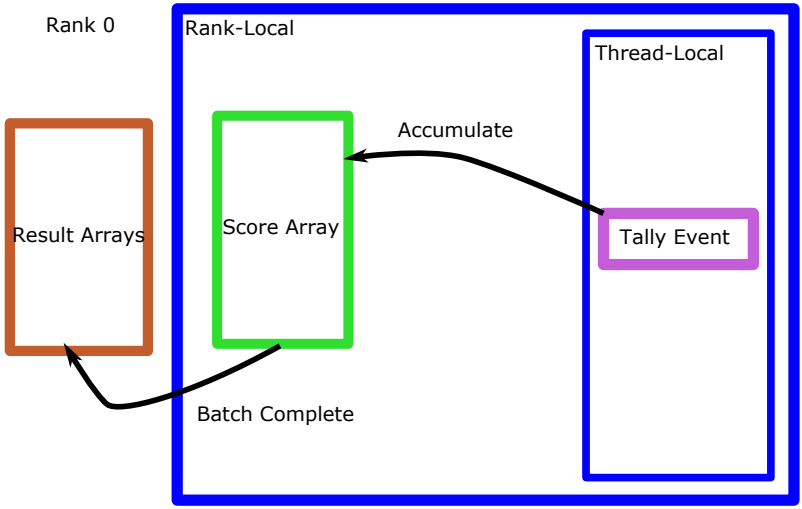
Every thread has a full score array.

Fast History Algorithm



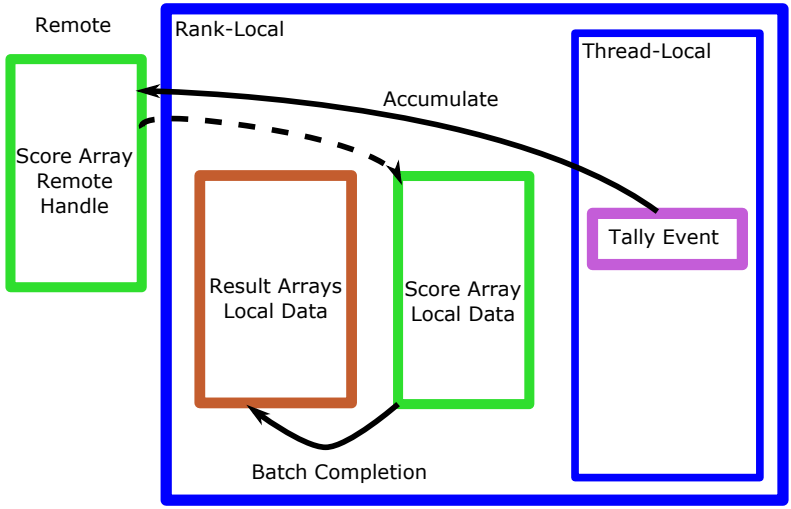
Every thread has a full score array and an indices array.

Batch Algorithm



Each MPI rank has a score array, rank 0 has the results.

Batch RMA Algorithm



Tallies are uniformly distributed without duplication.

Infrastructure Changes

The MCNP code didn't support batch statistics in any way. A number of changes had to be made:

- `NPS` now has a batch size option.
- When any batch tallies are enabled, `KCODE` will resample the fission bank to a fixed size.

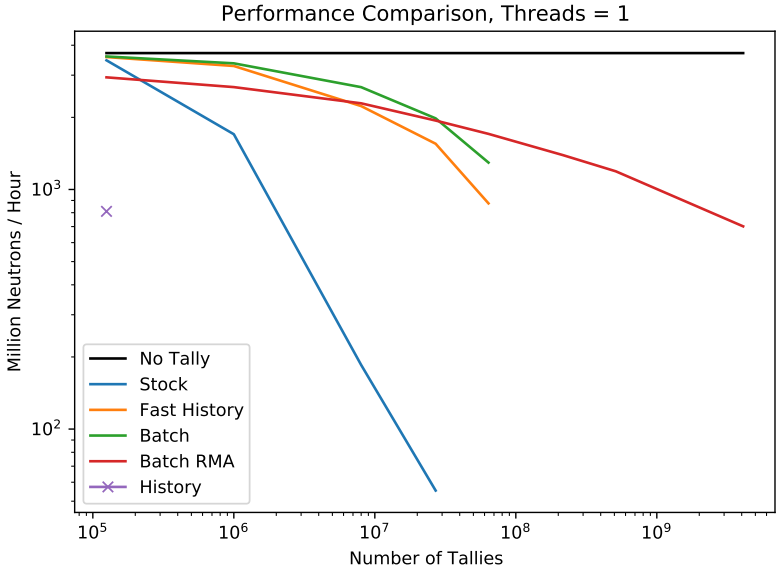
This means the RNG sequence will change if batch tallies are added!

Performance

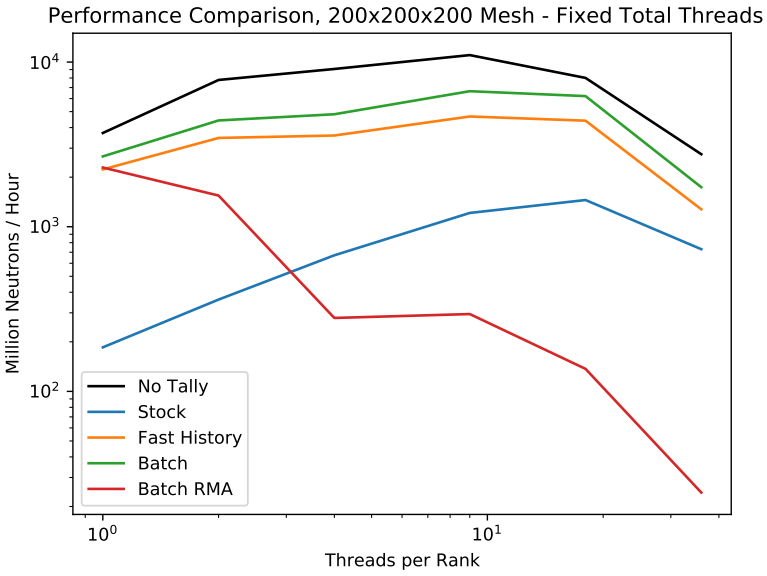
- Tested on a k-eigenvalue problem with a 10-cm, 10-g/cc ball of ^{235}U.
- Maximizes the effect of tally performance on the problem.
- Mesh was scaled from $50 \times 50 \times 50$ to $1600 \times 1600 \times 1600$
- Neutrons/hr and memory usage tallied
- Tested on 6 nodes of a cluster with 2 sockets, 18 cores each, 128 GB memory.
- Ran combinations of MPI, OpenMP.

Note: OpenMPI 3.1.6 + Omni-Path does not support `MPI_THREAD_MULTIPLE`, so threading performance is poor for Batch RMA.

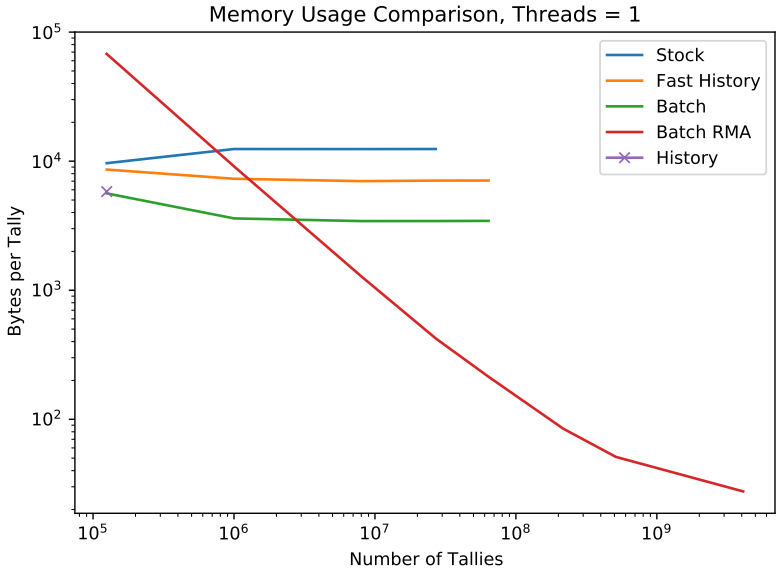
Performance



Performance

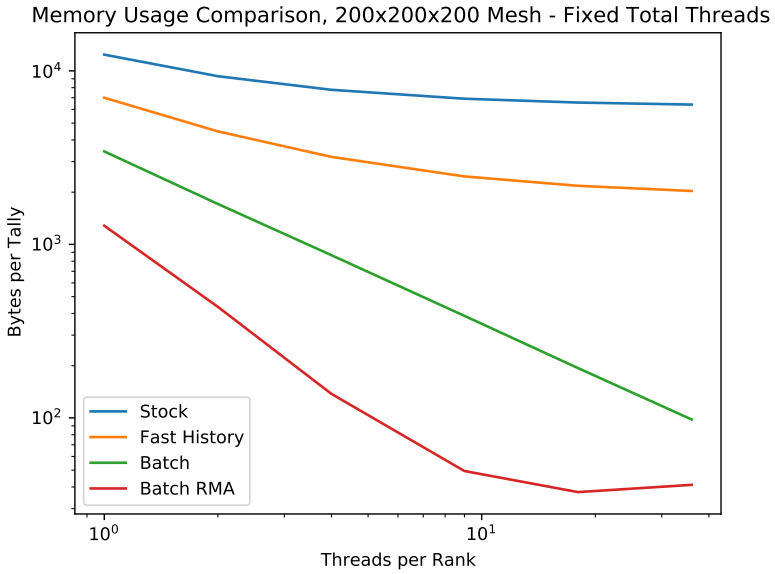


Memory



Batch RMA has high overhead that dissipates for large problems.

Memory


Memory Usage Comparison, 200x200x200 Mesh - Fixed Total Threads
Bytes per Tally
10^4
10^3
10^2
10^0
10^1
Threads per Rank
Stock
Fast History
Batch
Batch RMA

File Formats

Previous versions of the MCNP code used the MESHTAL format:

- ASCII output results in large file sizes.
- The binary to ASCII conversion was generally slow.
- It is tricky to bring into other tools (needs a processing script).

Version 6.3 uses HDF5 + XDMF:

- Binary file format for smaller sizes and faster IO.
- Trivial to load into ParaView, VisIt[1], Python, etc.
- (Optional) parallel HDF5 for even faster performance.

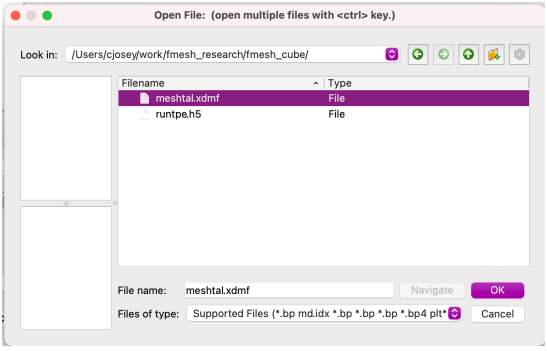
[1] Note that VisIt uses HDF5 1.8 at the time of this writing. MCNP outputs files that use the 1.10 format. Future versions of VisIt will use 1.10+. For now, `h5repack` can be used to convert to a 1.8 file.

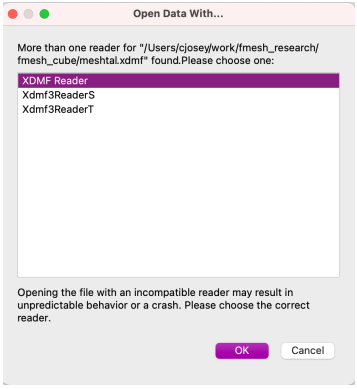
File IO Performance

216 million cell mesh, Lustre filesystem, 8 stripes, 1M stripe size, 8 MPI Ranks:

Method	Time (s)	File Size
MESHTAL	617.5	12 GB
HDF5 + XDMF	18.1	3.2 GB (in runtape)
Parallel HDF5 + XDMF	7.5	3.2 GB (in runtape)

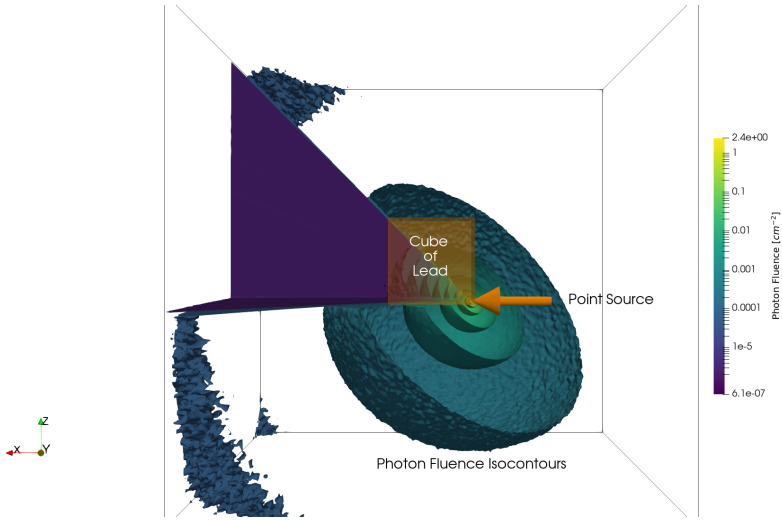
ParaView Example





Make sure to open with “XDMF Reader”, which is the reader for XDMF version 2 files.

ParaView Example - Point Source on Cube Corner



Python Example

Listing 1: Python 3.6+ Example for Reading FMESH

```python
import h5py
import numpy

def read_fmesh(filename, tally_id):
    with h5py.File(filename, 'r') as handle:
        group = handle[f"/results/mesh_tally/mesh_tally_{tally_id}"]

        data = {}
        # Transpose converts indices to x, y, z, e, t
        data["mean"] = numpy.transpose(group["mean"][()])
        data["relative_standard_error"] = \
            numpy.transpose(group["relative_standard_error"][()])
        data["grid_x"] = group["grid_x"][()]
        data["grid_y"] = group["grid_y"][()]
        data["grid_z"] = group["grid_z"][()]
        data["grid_energy"] = group["grid_energy"][()]
        data["grid_time"] = group["grid_time"][()]

        return data

data = read_fmesh("runtpe.h5", 4)
```

By slicing `group[''mean'']` instead of using `[()]`, one can load a portion into memory without loading all of it.

Summary

- New FMESH outperforms 6.2's in most workloads.
- New modes allow for much lower memory usage for large problems.
- File formats are fast and easy to work with.

In the future, we expect to extend this capability to more parts of the MCNP code.