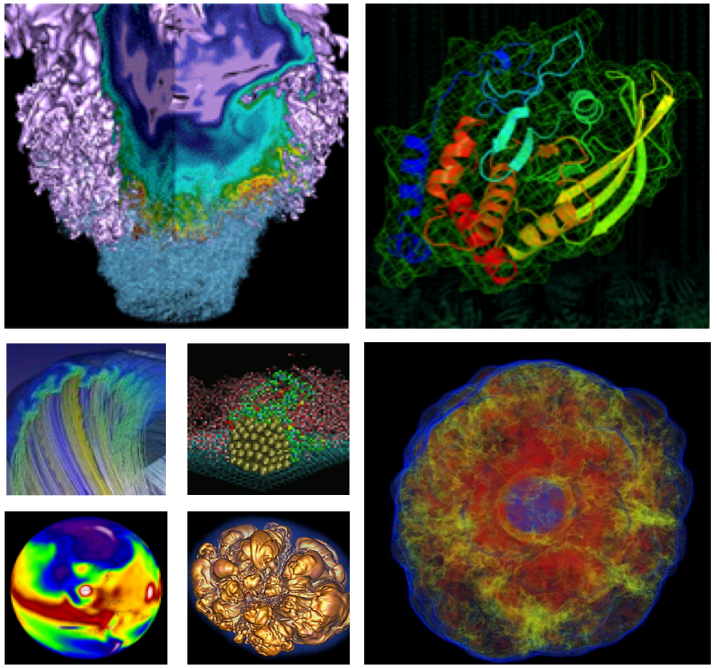
Next Steps for Geant4 Tasking Framework



Jonathan R. Madsen

jrmadsen@lbl.gov

Geant4 R&D Task Force

April 29, 2019

NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Why Tasking Library?

- Call my library PTL (Parallel Tasking Library)
  - https://github.com/jrmadsen/PTL/tree/implicit-manager-interface
- TBB is fast and excellent library
  - Faster than my library for very deep recursion → uses fibers
- Using TBB would mean a fundamental MT dependency
  - Complicate build/testing → version checking, API changes, etc.
  - More difficult to track down MT issues
- Using PTL would give full control i.e. internal CLHEP
  - Simplified testing, ability to customize to our needs
  - We don't have complicated tasking (e.g. flow-graph, deep recursion)
  - Compatible with TBB, if desired
  - Extra goodies → better signal handler, GetEnv<T> interface

Short- and Intermediate-term Goals

- Make the library available (could be done ~immediately)
  - Build option (-DGEANT4_BUILD_THREADPOOL=ON)
  - Keep existing MT but allow thread-pool to be created by master G4RunManager → *G4RunManager::GetThreadPool()*
    - Visualization would use this initially → simplified parallel viz
    - Users could supplement their pre-/post-processing work-flow
- Sub-event parallelism
  - R&E WG goal
- Eliminate G4MTRunManager
  - Thread-pool with zero threads would simply execute functions instead of bundling function into task and placing in queue → significant code simplification!

Down the road…

- It is a general consensus that GPUs are here to stay
  - Certain forms of transport have demonstrated to work well on GPU, e.g. medical physics
    - The problem was those implementations were not compatible with Geant4 → static polymorphism
  - Philippe, Soon, and I (+others) are working on solution as part of U.S. Department of Energy Exascale Computing Project (ECP)
  - It is generally known/believed that we will need to:
    - Significantly hide latency via async (overlap memory copies, kernel launches, etc.)
    - Balance work between CPU (hadronic) and GPU (EM, optical)
- Tasking provides load-balancing

Implementation Requirements

- Biggest hurdle w.r.t. to Geant4 for sub-event parallelism is RNG
  - Thread-local generators need to separated from threads
- Three possible solutions:
  1. High-level reference counting + protected inheritance
     - Action class instances, etc. would have unique seed
     - G4UniformRand(), etc. would be protected member function instead of global function
     - User code would be mostly unaffected
  2. Reseeding global generators when thread picks up sub-event
     - May require RNGs to allow “skip-ahead”
  3. More generators
     - Instead of run-seeds + event-seeds, need to add another layer
- Other recommendations welcome – we need to minimize user code changes

First Steps

- Before undergoing a migration to tasking, need to make sure performance is fully quantified → performance testing
  - Docker workflow
  - TiMemory package (introduced last G4 release) has undergone a re-write and is significantly improved
    - Completely modular: 11 timers, 14 resource usage metrics, PAPI counters
    - Header-only (for C++)
- **I would appreciate volunteers that would be willing to run benchmark problems** and submit them to performance dashboard at NERSC – please email me: jrmadsen@lbl.gov
  - Benchmarks could be advanced/extended examples
  - Build the container for your code using base image (provided) and run
    - *docker run –it Example-XYZ:baseline example-xyz bench.mac*
    - Collection of performance metrics would be automated along with submission to dashboard

Thank You