Multithreading I

Jonathan R. Madsen

Department of Nuclear Engineering
Texas A&M University
College Station, TX, USA 77843

madsen_jr@tamu.edu

NUCLEAR ENGINEERING
TEXAS A&M UNIVERSITY

Outline

1. What is multithreading (MT)?
2. Geant4 MT Design
3. How to activate MT
4. How to migrate code
5. MT-ready Components
6. Summary

Overview

- Modern CPU architectures introduce **parallelism**
  - CPU frequency advances have stagnated ⇒ negligible advances in processing speed
  - However, the cost per microprocessor has dropped significantly ⇒ multiple cores instead of faster cores
- Parallelism:
  - Running tasks that are not dependent on the result of each other simultaneously on different compute nodes (cores)
  - Two core types of parallelism:
    1. Shared-memory parallelism (multithreading)
    2. Distributed memory parallelism (*e.g.* MPI)
    - Hybridization (*i.e.* distributed shared-memory parallelism) is possible

Threads and Processes

- Both processes and threads are independent sequences of execution
- Process
  - Has own virtual address space, executable code, open handles to system objects, security context, unique process identifier (PID), and at least one thread
- Thread
  - Entity within a process
  - All threads within a process share its virtual address space and system resources
  - Has own exception handlers, scheduling priority, thread-local storage, and unique thread identifier
- Summary: Each process has it's own memory space while threads share the memory space of their parent process

Distributed Memory Parallelism — MPI

- Multiple instances of main program (multiple PIDs)
- Each instance splits the work either by handling a specific section of the geometry (domain-decomposition) or handling a fraction of the particles (particle-decomposition) in Monte Carlo calculations
  - Particle decomposition is generally considered "embarrassingly parallel" since each process only requires a different starting random number seed
- MPI will suffer from excessive memory requirements in large problems using particle-decomposition due to the overhead of replicating the entire problem
- MPI will suffer from poor performance in many domain-decomposition cases due to communication overhead when particles leave the domain of one process and enter into the domain of another process
- Geant4 has an implementation in the extended/parallel/MPI example(s)

Shared Memory Parallelism — MT

- One instance of the main program (one unique PID)
- There is a "master" thread associated with the process, all subsequent threads are known as "worker" threads
- Subsets of the program are split into tasks, which run on threads
- Memory is shared among all the threads, although threads can allocate thread-local memory
- Geant4 has chosen MT as it's core method of parallelism
- At the core of any multithreading on UNIX systems, POSIX threads (pthreads) are used
    - Geant4 uses the pthreads library
    - Many higher level interfaces
        - OpenMP
        - Thread Building Blocks (TBB) [1]
        - Boost threads
        - C++11 threads

[1] Example in examples/extended/parallel/TBB

MT vs. MPI

- Since each MPI process has it's own memory space, MPI applications generally require syncing data at some point
  - *E.g.* In a Monte Carlo application, start each MPI process with own random number seed and, at conclusion, distribute individual results to all other MPI processes — each process then has its own copy of the sum of all the results
- Since a MT process shares it's memory space, excluding thread-local memory, access to the shared memory must be governed to prevent data races (covered in MT II)
  - Thread-local memory is also generally synced, as with MPI, however, the distribution of the individual results is only sent to the "master" thread, instead of the other worker threads
  - Governing the access to the shared memory can add significant complexity to non-trivial applications

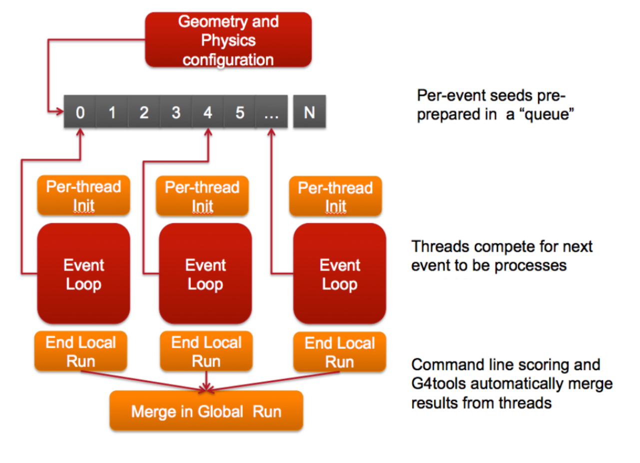
Geant4 MT

- Multithreading is introduced in Geant4 at the G4Event[2] level
  - In general, for performance, the higher up in the execution sequence, the better
  - Events are independent of each other — particles between events neither interact nor depend on each other
  - Each event is given its own unique random number seed
    - By reproducing the unique seeds for each event, the simulation can reproducible in multithreaded or serial mode
- Geant4 needs back-compatibility with user code and simple approach (physicists != computer scientists)
  - We have made every effort to provide an efficient and friendly multithreaded version of Geant4 that requires a minimum amount of effort to ensure thread-safety, however, thread-safety must be kept in mind because there is only so much that can be done at the toolkit level

[2] This does not mean there are not thread-local G4Run instances

Geant4 MT (cont.)

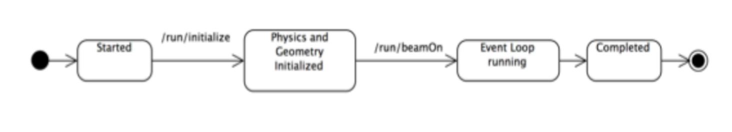
General Design



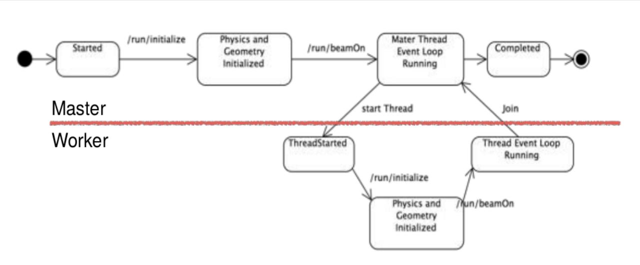
Credit: Andrea Dotti, SLAC

Geant4 MT (cont.)

Simplified serial Geant4

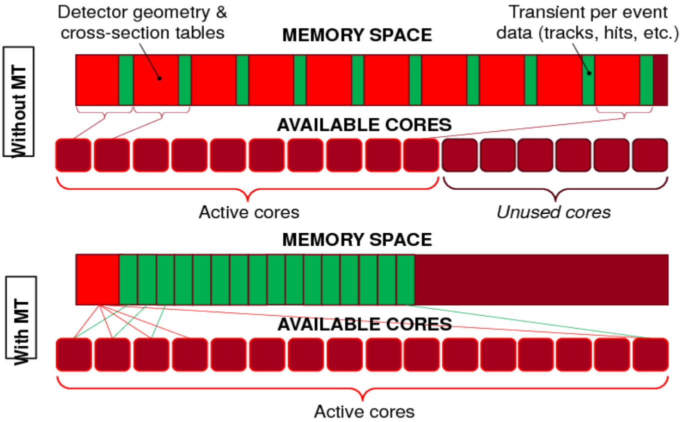


Simplified MT Geant4



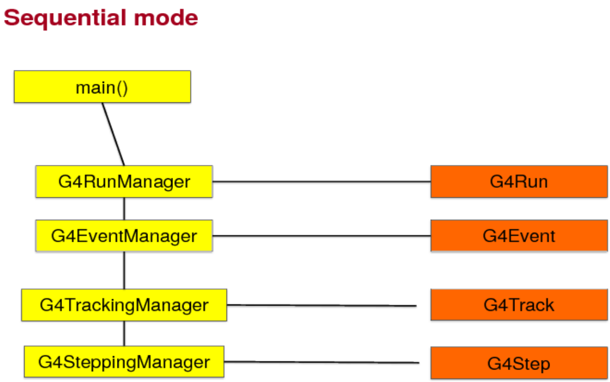
Credit: Andrea Dotti, SLAC

Geant4 MT (cont.)



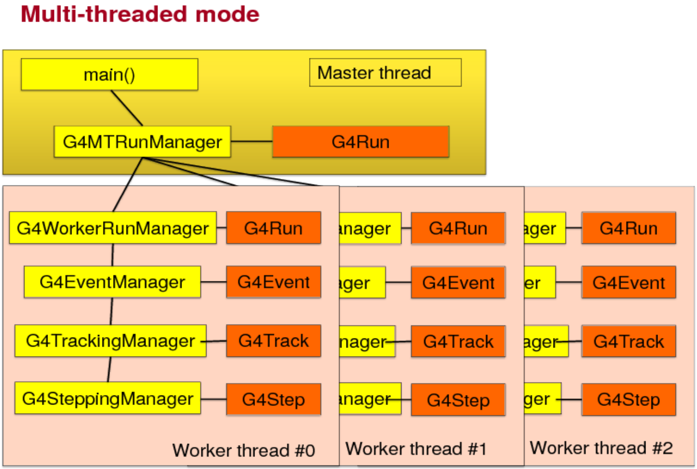
Credit: Andrea Dotti, SLAC

Geant4 MT (cont.)



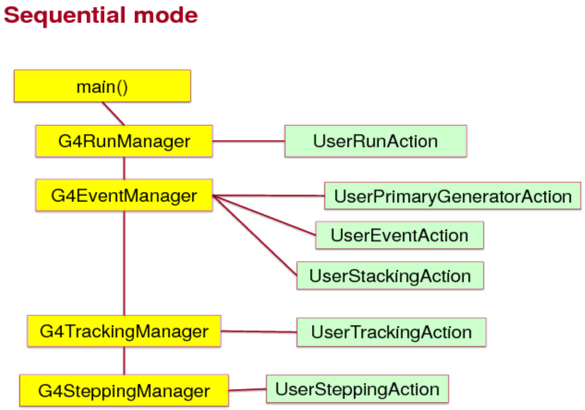
Credit: Andrea Dotti, SLAC

Geant4 MT (cont.)



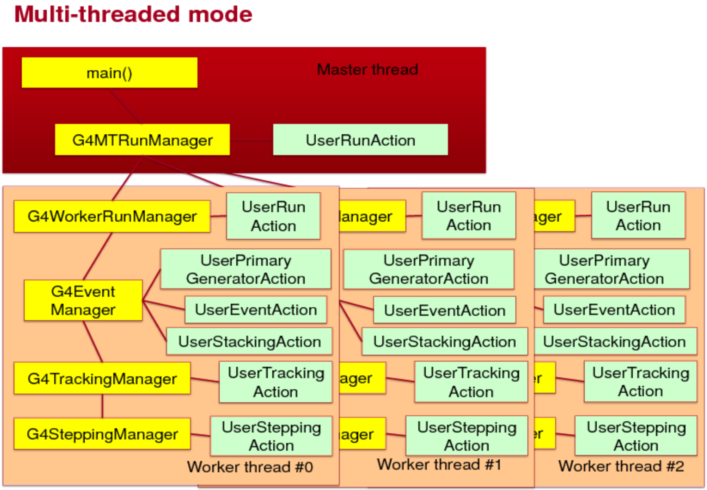
Credit: Andrea Dotti, SLAC

Geant4 MT (cont.)



Credit: Andrea Dotti, SLAC

Geant4 MT (cont.)



Credit: Andrea Dotti, SLAC

Shared vs. Thread-local

- To reduce memory footprint, threads must share a portion of the memory
- In general, any portion of the simulation that is static/constant[3] throughout the simulation is shared
  - Geometry definitions
  - Physics data tables
  - ... etc.
- In general, for reasons explained later, any portion of the toolkit that is dynamic/changing, during the simulation is allocated to thread-local (*i.e.* "private") memory
  - Scoring tallies
  - G4Step
  - G4Track
  - ... etc.

[3] "read-only"

Compiling Geant4 with multithreading enabled (UNIX)

- POSIX threads (pthreads) is required to be installed
- cmake -DGEANT4_BUILD_MULTITHREADED=ON [...]
- Requires recent compiler that supports thread-local storage
  - Check cmake output:
    - –– Performing Test HAVE_TLS
    - –– Performing Test HAVE_TLS – Success
- Geant4 applications will inherit the MT mode of the Geant4 toolkit it is built against and the preprocessor flag "G4MULTITHREADED" can be used in your application for specific MT-only code
  - In general, this preprocessor flag is only needed once in the main() of your application, which will be detailed later
  - All other Geant4 "threading-specific" tools detailed later will be ignored by the compiler when MT is turned off by the utilization of "G4MULTITHREADED" in the toolkit itself

NUCLEAR ENGINEERING
TEXAS A&M UNIVERSITY

Compiling Geant4 with multithreading enabled (Windows)

- Geant4 only supports multithreading on UNIX machines (Linux, Mac OS X) but can be utilized on Windows 10 with Windows Anniversary Update (Redstone) using the Windows Linux Subsystem (WLS)
  - Essentially, WLS $\Leftrightarrow$ Ubuntu 14.04 (Trusty Tahr)
- Instructions for enabling Windows Linux Subsystem can be found online[4], The installation procedure for Geant4 within the WLS will follow the instruction procedure for Geant4 on a Linux system
- Visualization can be enabled by installing Xming on Windows and setting the environment display variable in the WLS to the one specified when setting up Xming (*e.g.* DISPLAY=":0.0")
  - You will need to install the X11 libraries on the WLS + others
  - Ask me for a script I wrote to install all the necessary libraries to run Geant4 with MT, OpenGL, and Qt

[4] provided by Microsoft

Migration to Geant4 v10

- API has changed to smoothly allow multithreading (this is why it was a major release)
- A minimum of three modifications must be made:
  - G4VUserDetectorConstruction
  - G4VUserActionInitialization
  - G4MTRunManager
- G4VUserDetectorConstruction
  - G4VPhysicalVolume* G4VUserDetectorConstruction::Construct();
    - Build geometry here except Sensitive Detectors and magnetic field
    - Called by master thread once
  - void G4VUserDetectorConstruction::ConstructSDandField(); [mandatory]
    - Build Sensitive Detectors and Magnetic fields here
    - Called by each thread

Migration to Geant4 v10 (cont.)

- Create a new class that inherits from G4VUserActionInitialization and implements:
  - void G4VUserActionInitialization::Build()
    - Instantiate user-actions for worker threads
    - Called by each thread
  - void G4VUserActionInitialization::BuildForMaster()
    - Instantiate user-actions for master (optional)
    - Called by master thread

Migration to Geant4 v10 (cont.)

- Add G4MTRunManager to $int\ main(int\ argc,\ char**argv)\{...\}$

```
int main(int argc, char** argv)
{
  ...
#ifdef G4MULTITHREADED
  // If Geant4 compiled with GEANT4_BUILD_MULTITHREADED=ON
  G4MTRunManager* runmanager = new G4MTRunManager();
  runmanager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
#else
  // If Geant4 compiled with GEANT4_BUILD_MULTITHREADED=OFF
  G4RunManager* runmanager = new G4RunManager();
#endif

  // Detector initialization
  runmanager->SetUserInitialization(new DetectorConstruction);
  // Physics list
  runmanager->SetUserInitialization(new FTFP_BERT);
  // User-action initializations
  runmanager->SetUserInitialization(new ActionInitialization);
  ...
}
```

Scoring

- Geant4 sensitive detector, hits collections are MT ready
  - Hits objects, as well as sensitive detectors, are instantiated on worker threads
  - A keyword G4ThreadLocal (covered in MT II) will inform the user that a hit class is MT ready

```
// Below are sequential only (not MT compatible)
extern G4Allocator<MyHit>* MyHitAllocator; // in MyHit.hh
G4Allocator<MyHit>* MyHitAllocator = 0; // in MyHit.cc
```

```
// Below are MT-ready
extern G4ThreadLocal G4Allocator<MyHit>* MyHitAllocator; // in MyHit.hh
G4ThreadLocal G4Allocator<MyHit>* MyHitAllocator = 0; // in MyHit.cc
```

Analysis

- Geant4 analysis tools are MT-ready
- Histograms and profiles
  - Each thread owns its own copy of given histograms and profiles
  - At the end of the run, worker objects are "merged" into a single object on the master thread
  - A single file with merged histograms and profiles will be produced
- When using G4AnalysisManager with histograms, the UserRunAction class must be instantiated on **both the master and worker threads**

Analysis (cont.) — ntuples

- Each thread owns a copy of the ntuple
- ntuples are **not merged** at the end of the run — concatenate in analysis!
- Output files
  - Each thread will write own separate file, where the files names are generated automatically from the thread-id:

    [fileName]_[ntupleName]_[thread-id].[extension]

    e.g. run_eDep_1.txt
- When using ROOT output, the ntuple files per thread can be analyzed with use of the TChain class

Visualization

- Geant4 visualization is MT-ready
- Rendering is done by the master thread, based on event keeping settings
- Events are drawn directly from worker threads as soon as they are ready

Summary

- Geant4 supports parallelism via multithreading since Geant4 v10.0
- Multithreading is at the event-level
- MT is only available on Linux/OS X, but can be run on Windows within the Windows Linux Subsystem
- Migration to Geant4 v10+ requires changes to existing code written for Geant4 v9.6 and older