Multithreading III

Jonathan R. Madsen

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

madsen_jr@tamu.edu

Outline

1. MT Review, Verification, and Additional Tools
2. UI commands related to MT
3. Thread-safe Scoring

Multithreading Review

- One process (*i.e.* one PID) with worker threads to divide up the work
  - Easiest method of determining whether a process is using MPI or MT is inspecting the task manager on the UNIX OS and looking for the running process
    - One PID $>$ 100% CPU utilization $\Rightarrow$ MT
    - Multiple PID $\leq$ 100% CPU utilization $\Rightarrow$ MPI
    - MPI processes can utilize MT and poor MT can overuse mutual exclusion so this is just a general rule
- Data races
  - Occur when one thread attempts to update a shared value while another is reading the shared variable
  - Primary concern in MT
  - Do not often cause fatal errors during execution
  - Locking (via mutual exclusion) has severe performance penalties when used at high levels in the code and in highly-contested areas

Multithreading Verification

- Run the simulation with one thread and in serial mode and verify the solutions match
  - Compile the Geant4 toolkit with GEANT4_BUILD_MULTITHREADED=ON
  - Run with environment variable G4FORCENUMBEROFTHREADS=1, compare output with G4FORCENUMBEROFTHREADS=MAX
  - Allocate G4RunManager in the main() instead of G4MTRunManager

```
//#ifdef G4MULTITHREADED
//  G4MTRunManager* runmanager = new G4MTRunManager();
//  runmanager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
//#else
  G4RunManager* runmanager = new G4RunManager();
//#endif
```

(which forces serial execution) and compare the output from G4MTRunManager and G4FORCENUMBEROFTHREADS=MAX

- NOTE: Very minute differences may occur due to round-off

Multithreading Tools

- Lecture MT II covered G4Cache, G4ThreadLocalSingleton, G4AutoDelete, G4Mutex, and G4AutoLock
- Some additional MT tools are included in Geant4
  - G4Parameter
    - Provided by Ivana in source/analysis/parameter
    - Template class to handle data scoring and accumulation
    - Keeps thread-local copies of data for each thread (high performance)
    - Handles merging of thread local copies
    - G4MergeMode can be overloaded to operation of user's choice
  - G4atomic (atomic, in this sense, is a computer science term)
    - Provided by myself in examples/extended/parallel/ThreadsafeScorers
    - G4atomic uses lock-free mechanisms (if provided by compiler) to handle updates of shared plain-old data (POD)
    - Can be used for shared counting variables or thread-safe scoring where thread-local copies are not desired due to memory restrictions

User-defined UI commands

- Master thread "accumulates" the commands and passes the commands to all the threads at the beginning of the run
- Threads execute the same commands sequence as master thread
- However, some commands make sense only in master threads (*e.g.* modifying geometry)
  - UI commands can be marked as "not to be broadcasted":
    - G4UICommand::SetToBeBroadcased(false);
    - Do not forget this step if you implement user-defined UI commands

Thread-safe scoring

- Scoring with Multifunctional Detectors (MFD) and Primitive Scorers
  - By far, the easiest way to handle scoring in a thread-safe way
  - Other lectures cover this so it will not be covered here
- Scoring in Stepping Action
  - In some cases, users desire to personally handle the scoring themselves in the G4UserSteppingAction object for various reasons
  - In serial Geant4, this was fairly straightforward
    - Determine the value(s) to be scored in G4UserSteppingAction object and send the value(s) to value(s)/container(s) located in G4UserRunAction object
  - In multithreaded Geant4, this can still be done but will require a lock around the update in the thread-global G4UserRunAction object to ensure the absence of a data race
    - Will suffer from severe performance issues due the massive number of steps that will have to wait to acquire the lock

Scoring in Stepping Action

- Similar to example ≪ basic/B1 ≫ but without route through G4UserEventAction and variance calculation
- Components
  - Run Action for the master thread
  - Run Action for the worker threads (passed to worker G4UserSteppingAction)
  - Utilize G4Parameter to handle automatic merging of thread-local copies
- Workflow
  1. Setup Run Action to use G4Parameter
  2. Construct thread-global Run Action in Action Initialization
  3. Construct thread-local Run Action and provide pointer to Stepping Action
  4. Use thread-local Run Action pointer in Stepping Action to accumulate scoring value(s), which are located in Run Action
  5. At conclusion of thread-local run, in RunAction::EndOfRunAction(), use G4ParameterManager to merge thread-local results

Setup Run Action to use G4Parameter

```
class RunAction : public G4UserRunAction // in RunAction.hh
{
public:
  ...
  void AddEdep (G4double edep);

private:
  G4Parameter<G4double> fEdep;
};

void B1RunAction::AddEdep(G4double edep) // in RunAction.cc
{
  fEdep  += edep;
}

void RunAction::RunAction() // in RunAction.cc
: fEdep("Edep", 0.0)
{
  // Register parameter to the parameter manager
  G4ParameterManager* parameterManager = G4ParameterManager::Instance();
  parameterManager->RegisterParameter(fEdep);
}
```

Construct thread-global Run Action in Action Initialization

```
void ActionInitialization::BuildForMaster() const
{
  RunAction* globrunact = new RunAction();
  SetUserAction(globrunact);
}
```

Construct thread-local Run Action and provide pointer to Stepping Action

```
void ActionInitialization::Build() const
{
  RunAction* tlrunact = new RunAction();
  SetUserAction(tlrunact);
  ...
  SetUserAction(new SteppingAction(tlrunact));
}
```

Provide pointer to thread-local Run Action to Stepping Action

```
void SteppingAction::SteppingAction(RunAction* ra)
: runact(ra)
{ }
```

Use thread-local Run Action pointer in Stepping Action

```
void SteppingAction::UserSteppingAction(const G4Step* s)
{
  // runact is thread-local Run Action
  runact->AddEdep(s->GetTotalEnergyDeposit());
}
```

Use G4ParameterManager to Merge thread-local results

```
void RunAction::BeginOfRunAction(const G4Run* aRun)
{
  // reset parameters to their initial values
  G4ParameterManager* parameterManager = G4ParameterManager::Instance();
  parameterManager->Reset();
}

void RunAction::EndOfRunAction(const G4Run* aRun)
{
  // Merge parameters
  G4ParameterManager* parameterManager = G4ParameterManager::Instance();
  parameterManager->Merge();

  // Get total energy deposit in a run
  G4double edep  = fEdep.GetValue();

  // #include "G4UnitsTable.hh" for G4BestUnit
  G4cout << "Total energy deposit : " << G4BestUnit(edep, "Energy") << G4endl;
}
```