How to make an example multi-threaded

Makoto Asai

2012 Geant4 Collaboration Meeting @ Chartres

Preamble

- Recipe provided here is for the current G4MT prototype.
  - Interfaces in particular to frameworks of large experiments are still under discussion. At least method names of user detector construction will be changed.

- All information in this talk are found in Geant4MT_UsersGuide.pdf file included in the Documents directory of Geant4MT-9.5.p01 distribution.
  - Installation guide is also found in the same directory.

- Basically you need to modify four kinds of classes.
  - User detector construction
  - Hit class
    - If you use it
  - User action class
    - If you accumulate values
  - main()

- Note: Geant4MT.9.5-p01 works only with CMake.
  - GNUmake set-up does not exist. It is removed.

Detector construction – 1

- Sensitive detectors are thread-local, while geometry itself is common.

1. Add three public methods
   - G4VPhysicalVolume* ConstructSlave()
   - void SlaveDetectorConstruction()
   - void DestroySlave()
     - Note: these method names will be changed at next release
2. Make G4LogicalVolume pointers where you define sensitive detector to be class scope data members. Also G4VPhysicalVolume pointer for the world volume.
   - Modify implementation in you Construct() method accordingly.
3. If you have a field, make the pointer to your field class object thread local.
   - In header file

     ```
     static __thread A01MagneticField* fpMagField;
     ```
   - In source file

     ```
     __thread A01MagneticField* A01DetectorConstruction::fpMagField = 0;
     ```

Detector construction – 2

4. Implement SlaveDetectorConstruction() and DestroySlave() methods for instantiation and deletion of your field class

5. Move instantiation of all sensitive detectors and assignment to logical volumes into ConstructSlave() method.

Hit class

- G4Allocator for hit class must be thread local.

1. Header file

```
typedef G4THitsCollection<A01Hit> A01HitsCollection;
extern __thread G4Allocator<A01Hit> *A01HitAllocator;

inline void* A01Hit::operator new(size_t) {
  if(!A01HitAllocator)
        A01HitAllocator = new G4Allocator<A01Hit>;
  void* aHit = (void*)(*A01HitAllocator).MallocSingle();
  return aHit; }

inline void A01Hit::operator delete(void* aHit) {
  if(!A01HitAllocator)
        A01HitAllocator = new G4Allocator<A01Hit>;
  (*A01HitAllocator).FreeSingle((A01Hit*) aHit); }
```

2. Source file

```
__thread G4Allocator<A01Hit> *A01HitAllocator = 0;
```

User action

- In case you accumulate values in your user action classes, you need to make them thread local.

1. Header file

   static __thread G4bouble eDep;

2. Source file

   __thread G4double MyStepStion::eDep = 0.;

main() – 1

Note: Modification required for *main()* will change significantly at the next release.

1. The *main()* needs to be modified in order to make use of Geant4MT multi-threading features. The following *include* statement provides access to the G4MTTopC parallelRunManger class. The second statement initializes the *DetectorConstruction* pointer.

```
#include "G4MTParTopC.icc"
A01DetectorConstruction* detector = 0;
```

main() – 2

2. The *main()* then initializes the run manager. If the constructor of the run manager is invoked with an integer argument then it runs as a slave thread and the integer argument becomes the thread rank. Depending on whether the main method is called from the master or from a slave thread it calls the corresponding *A01DetectorConstruction* constructor or *SlaveA01Construct()* method.

```
// Multi-threaded RunManager construction
G4RunManager* runManager;
if (threadRank == 0) runManager = new G4RunManager;
else runManager = new G4RunManager(1);

// Multi thread detector construction
if (threadRank == 0) detector = new A01DetectorConstruction;
else detector->SlaveDetectorConstruction();
```

main() – 3

3. Finally, after all the threads have returned, the allocated memory must be freed to prevent memory leaks.

```
// Make sure that slave threads free their geometry pointers
if (threadRank != 0) detector->SlaveDestroy();
//make sure the runManager is deleted
if (threadRank == 0) delete runManager;
```

Note:

1. Each thread generates two output files for *stdout* and *stderr.*
2. To start an execution, put an additional parameter to indicate how many threads you use.

```
$ A01app run.mac 8
```