Multi-core and multi-threading: Tips on how to write “thread-safe” code in Geant4

Gene Cooperman and Xin Dong
High Performance Computing Lab
College of Computer and Information Science
Northeastern University
Boston, Massachusetts 02115
USA
{gene,xindong}@ccs.neu.edu

Early 2010: First Version of Geant4MT

Geant4MT: *Geant4-MultiThreaded*

This takes advantage of multi-core architectures now commonly available. A multithreaded version of Geant4 (using multiple threads, and also copy-on-write) is more computationally efficient than running:

(i) multiple independent Geant4 processes; or

(ii) multiple forked child Geant4 processes (using UNIX copy-on-write semantics)

A first version for use by developers is planned for early 2010

Goals of initial Geant4MT release

- *Goal:* allow developers to become aware of the new technology
- *Delivered:* Geant4MT source code — a modified version of the Geant4 source code
- *Delivered:* Manual
- *Tools:* Tool for transforming Geant4 code into Geant4MT code. (Most Geant4 kernel developers will not need to use this tool.)

How is Geant4MT Created from Geant4?

1. Most of the Geant4MT modifications are created automatically by the source code transformation tool. About 10,000 lines of code are automatically modified.
2. The “default” is for Geant4 classes to be thread-private. Each thread has its own private copy of a Geant4 object.
3. Certain Geant4 classes are modified in Geant4MT to create shared objects (objects shared by the threads). Two primary ways to modify Geant4 source code:
   (a) C++ stylized comments (hints for Geant4MT)
   (b) C preprocessor conditionalizations: `#define`
4. *Geant4MT source code transformation tool:* based on modified g++ parser: generates database of symbols in Geant4 source code requiring modifications in their use; then applies modifications.

How does this Affect the Geant4 Kernel Developer?

1. If the Geant4MT source code transformation tool recognizes your coding style, and if your Geant4 classes should be thread-private (each thread has a private copy of its objects), then you don't need to do anything.
2. The manual documents a coding style that is recognized by the source code transformation tool. More than 99% of the current Geant4 source code already conforms to this coding style. We call this code *Geant4MT-ready*.
3. In some cases, kernel developer code may not be Geant4MT-ready. In such cases, there are two options:
   (a) Rewrite the code to conform to conform to a style recognized by the tool.
   (b) Use C conditionalizations (`#define`) and/or C++ stylized comments/hints for the source code transformation tool.
4. In most cases, the Geant4MT source code transformation tool will detect and report code that is not Geant4MT-ready. The Geant4MT team will be running the tool *for this first version to promote developer awareness*. Kernel developers do not need to run the tool themselves.

What categories of code need to be verified as Geant4MT-ready?

1. Static initialization of global variables (static data members of classes that are initialized).
2. Classes that are intended to be shared, which contain data members with read-write access. (Read-only data members are fine.)

The Geant4MT Experience

1. Manual with detailed examples for correct versus incorrect Geant4MT-ready kernel code.
2. Manual will explain issues of thread-private versus thread-shared objects.
3. If the code appears Geant4MT-ready, don't change it. You're done.
4. If the code appears not Geant4MT-ready, change the coding style to conform if possible.
5. Otherwise, use C preprocessor conditionalizations and stylized comments/hints. Cross your fingers and hope it works.
6. Then wait for the next run of the Geant4MT source code transformation tool, and see if it worked.
7. In unusual cases, you may have to work with the Geant4MT developers. The source code transformation tool can be extended to recognize additional coding styles.

Later Versions of Geant4MT

1. Geant4MT (version 2): Incorporates extended coding styles learned from first version.
2. Geant4MT (version 3): User manual to allow for creation of Geant4MT-ready Geant4 user applications.

Geant4 → Geant4MT Transformation *(automatic)*

- Follow the “change list”
- Transform the original Geant4 to be thread-safe
- Example for a static variable that is not a class member

BEFORE:

```
static G4FieldTrack endTrack( ’0’);
```

AFTER:

```
static __thread G4FieldTrack *endTrack_NEW_PTR_ = 0 ;
if ( ! endTrack_NEW_PTR_ )
  endTrack_NEW_PTR_ = new G4FieldTrack (’0’) ;
G4FieldTrack &endTrack = *endTrack_NEW_PTR_;
```

Automatically transform Geant4 (cont.)

- Example for a static class member

  BEFORE:

  static G4String dirName;

  AFTER:

  static __thread G4String dirName_NEW_PTR_;

  BEFORE:

  G4String G4NuclearLevelStore::dirName("");

  AFTER:

  __thread G4String G4NuclearLevelStore::dirName_NEW_PTR_ = 0;

  G4NuclearLevelStore* G4NuclearLevelStore::GetInstance()

  {if (! dirName_NEW_PTR_)

  dirName_NEW_PTR_ = new G4String("") ;

  G4String &dirName = * dirName_NEW_PTR_;

  . . . }

Sharable Class Transformation *(by user)*

Redefine the references for read-write data members

```
class G4PVReplica : public G4VPhysicalVolume
{
   int g4PVReplicaObjectOrder;
   static G4PVReplicaPrivateObjectManager g4PVReplicaPrivateObjectManager;
   ...
   // G4int fcopyNoG4PVReplica;
   ...
}
```

```
#define    fcopyNoG4PVReplica
((g4PReplicaPrivateObjectManager.offset[g4PVReplicaObjectOrder]).fcopyNo)
```

Sharable Class Transformation (continued)

Implement the array for all thread-private data members

```
class ReplicaPrivateObject
{
public:
   G4int fcopyNo;
};

class G4PVReplicaPrivateObjectManager
{
public:
   ReplicaPrivateObject* privateDataArray;
   int MasterAddNew()...
   void WorkerInitialization()...
   void WorkerFree()...
}
```

Questions?