MT Issues Affecting Hadronics after 10.0

Michael H. Kelsey
SLAC GEANT4 Group

GEANT4 Collaboration Meeting
26 Sep 2013

Introduction

Hadronics framework, including all processes and models, is functioning properly in multithreaded (MT) environment for 10.0 release

Three major areas required for “re-” optimization

- Reducing per-thread memory footprint (size and churn)
- Isolating large computed tables from static shared tables
- Handling per-thread initialization from kernel

Code Modifications

Substantial automated and manual modifications to code required

- Conversion of `static` local variables to "**TLS**" (thread-local storage) pointers
- File-scoping (anonymous namespace) of hardcoded numeric arrays
- Separation of static/shared data from per-thread active classes
- Beginning-of-job initialization of some shared objects

Some changes may increase memory use or churn (new/delete cycles)

Possible marginal increase in CPU vs. 9.6 base release

Want to re-optimize hadronic models following modifications required for MT compatibility

Memory Footprint

Threads have limited block of local memory available

Hardcoded arrays should be moved to (“unlimited”) shared memory, replacing within-function declarations of “const” to file-scoped (anonymous namespace) “const” (equivalent to static)

Variable names need not be changed, and do not need class or namespace prefixes

Function-local buffers, especially vectors, should be moved to *per-thread* class data members, with `clear()` and `resize()` calls for initialization

Eliminating "TLS" Code

Automated code processing for MT prototype converted function-local static variables to pointers with "`_G4MT_TLS_`" suffix

```
static G4ThreadLocal G4ThreeVector *pvec_G4MT_TLS_ = 0;
if (!pvec_G4MT_TLS_) pvec_G4MT_TLS_ = new G4ThreeVector;
G4ThreeVector &pvec = *pvec_G4MT_TLS_;
```

1. If TLS object is constant, use static-const and make it shared
2. Move to class data member, if class has thread-lifetime
3. Leave as is
4. Convert to local (non-static) variable (causes memory churn!)

Memory Trade-offs

Balance between footprint (“memory used by thread”) and churn (“memory needed during run”)

Memory churn happens as small objects are created, used, then destroyed asyncronously

Small blocks of free memory are left unusable by later, larger allocation needs

- Eliminate temporary buffers (function-local objects/arrays/vectors)
- If class has thread-lifetime, use data member buffers
- Pass output objects into functions as non-const references

Cross Section Tables

Cross-section tables most prominent part of “shared memory” in multithreaded environment

Many tables are truly static/constant numeric values

- Not pre-initialized or hard-coded arrays
- Loaded from data files at initialization time, or on-demand

Others computed on-demand using material and track properties

Keeping both in same data structure requires frequent writing from threads to shared memory: locks (mutexes) impair performance

Thread Initialization

For EM processes, run-by-run initialization must be done in both master thread (for shared, static tables) and in worker threads (computed tables)

At start of each run, kernel calls **PreparePhysicsTable**, **BuildPhysicsTable** for each process

“Master” and “Worker” versions to separate shared from thread-local initialization

No interface for model-specific initialization, needed for hadronics

Processes and Models

Hadronic processes are primarily "shells" or "wrappers"

- Translate between kernel objects (G4Tracks) and hadronic objects (G4HadProjectile, G4HadSecondary, G4Fragment, etc.)
- Detailed physics encapsulated in models

Processes generally do not own their models (RDM, Stopping are exceptions)

- Model passed to `G4HadronicProcess::RegisterMe` in physics list
- Stored in *registry* with info about particles, targets, energy range, etc.
- At interaction, process asks registry for applicable model

Hadronic process cannot use **BuildPhysicsTable** interface to initialize "its" model(s)

Initializing Models

Currently, BERT is initialized *by hand* with call to `static void G4CascadeInterface::Initialize()`

Hard-coded into kernel beginning-of-run code (**bad!** deep dependency)

Proposal

Add non-static `void G4HadronicInteraction::Initialize() {;}` and `InitializeForWorker() { Initialize(); }`

Models implement as needed to pre-create shared or large objects

Extend `RegisterMe` to call *model*->`Initialize()` or `InitializeForWorker()` as appropriate

Summary

Significant progress: all hadronics now run under MT jobs

Reduce thread-local (TLS) variables by making class data members (use `mutable` where necessary)

Move in-function hardcoded arrays to file scope (`namespace {...}`)

Create needed shared-memory objects in *model*`::Initialize()`

See **MT Developers' TWiki** for more information, details