Toward Geant4 version 10

Makoto Asai (SLAC PPA/SCA)
For the Geant4 Collaboration
9th Geant4 Space Users Workshop @ Barcelona
March 6th, 2013

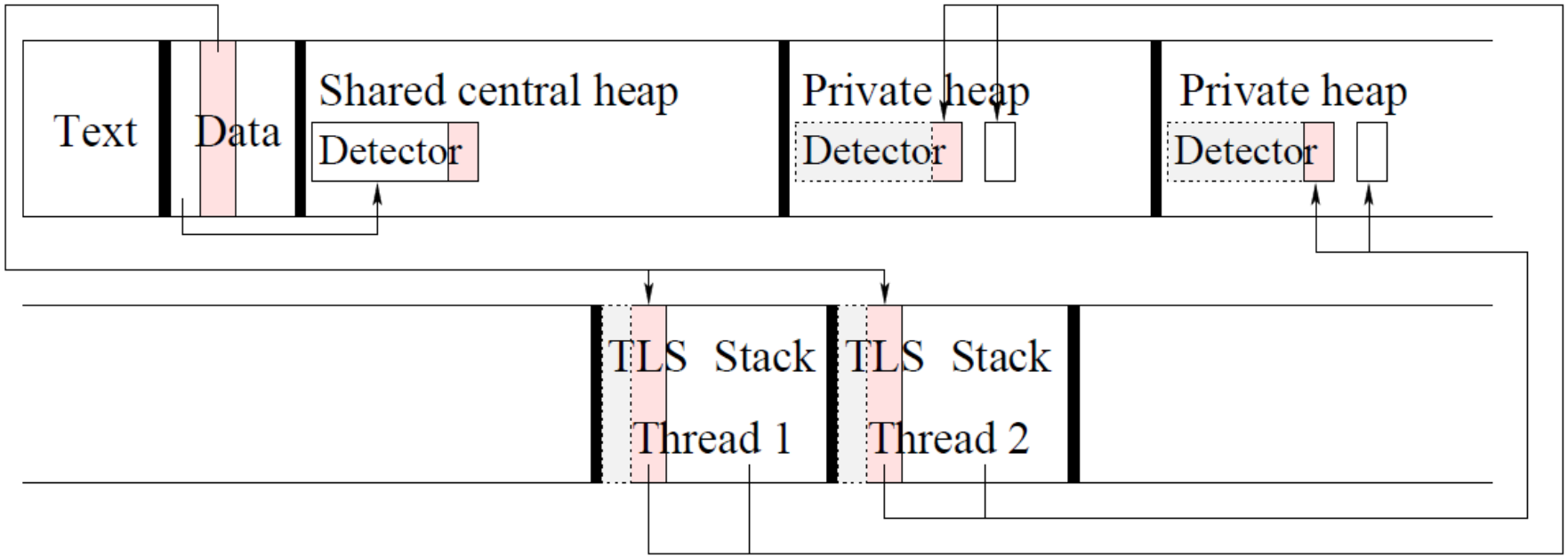
SLAC NATIONAL ACCELERATOR LABORATORY
U.S. DEPARTMENT OF ENERGY
Office of Science

Preamble

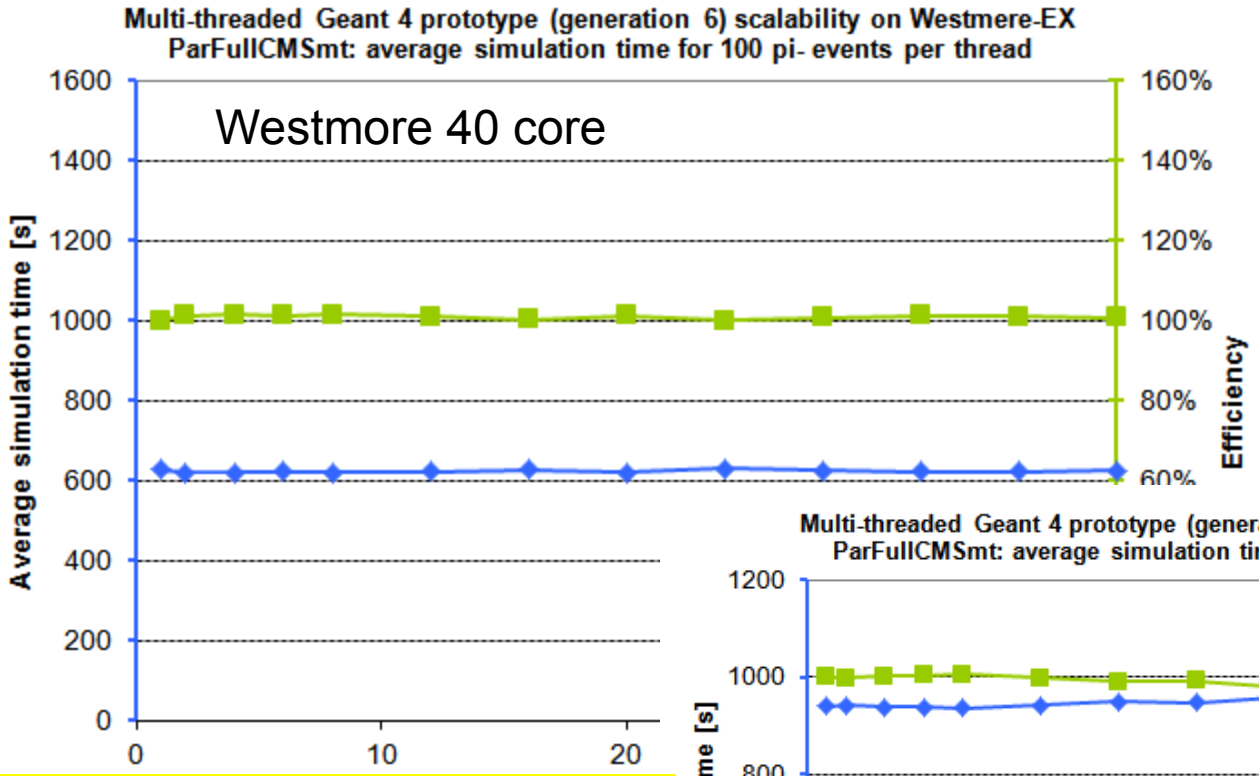
- The release in 2013 will be a major release.
  - Geant4 version 10
- The highlight is its multi-threading capability.
  - Some interfaces need to be changed due to multi-threading
- It offers two build options.
  - Multi-threaded mode (including single thread)
  - Sequential mode
    - In case a user depends on thread-unsafe external libraries, he may install Geant4 in sequential mode.
- This is the first major release since 2007.
  - This is a rare opportunity for us to clean up obsolete code and make interface improvements.
  - GNUmake will be dropped.

Multi-threading of Geant4 version 10

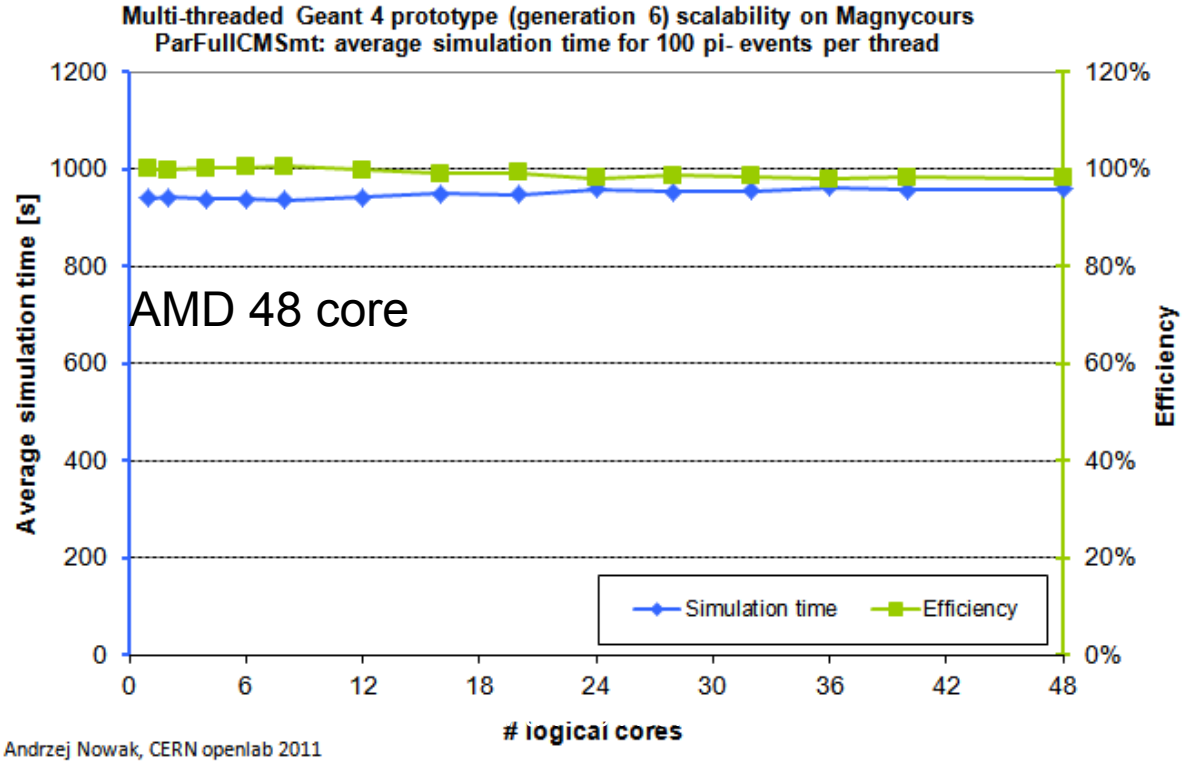
- Geant4 version 10 offers so-called event-level parallelism.
  - Each thread is tasked for an event or a bunch of events.
- Every data that is updated at event-level frequency or shorter has to be thread local.
  - To avoid the race problem.
- Status of current prototype (G4MT-9.5.p01)
  - Being tested on a Intel® Xeon Phi coprocessor (MIC)
  - Initial tests show good scalability up to hundred of concurrent threads



G4MT prototype has shown excellent scalability



Courtesy of Andrzej Nowak (OpenLab)



Note: scaling was still perfect with using 80 threads on Westmore (2 threads per core). Latest news – G4MT showed perfect scalability for Intel Xeon Phi as well.

Andrzej Nowak, CERN openlab 2011

Preliminary studies on TBB

- Intel Threading Building Block is a library for task-based multi-threading code. Some LHC experiments show their interest in the use of TBB in their frameworks.
- We have verified that the current G4MT prototype can be used in a TBB-based application where TBB-tasks are responsible for simulating events.
  - You don't need to modify any concrete G4MT class to adapt to TBB.
- A simple test code has been prepared that uses TBB and G4MT.
- We will provide an example or two at the beta release of version 10 to demonstrate the way of integrating TBB and G4MT.
  - We will keep communicating with our users to polish our top-level interfaces.

Timeline toward version 10

- Geant4 9.6 released on Nov.30
  - Final release of version 9 series
- Dec 2012 / Jan 2013
  - Conversion of v9.6 to G4MT
  - Move G4MT v9.6 to main development trunk
    - All development toward version 10 should be made to this development trunk.
    - We maintain native v9.6 in SVN brunch for potential patch release.
- Feb-May 2013
  - Migration of examples and tests
  - Massive tests for both computing performance and physics performance
- June 2013
  - Beta release : all the major changes related to multi-threading should be included
  - If necessary, more than one beta-releases may be made.
- Dec 2013
  - Public major release of Geant4 version 10.0

Interface changes in version 10 – before the beta release

- Obsolete classes / methods
  - All classes / methods to be removed have warning message in v9.6.

    "This class becomes obsolete and will be removed at the next release."
- Changes caused by / related with multi-threading
  - Finalize major changes by February 2013 before migration of examples / tests begins.
  - We're doing our best to minimize the migration cost of user's code.
  - Reference tags should be made available to testers.
- Changes independent to multi-threading
  - Given it's a major release, we may have some other interface changes. Some come with the beta release, some come after. We make sure they run in multi-threading mode.

Note: In addition to the massive tests in multi-threaded mode, Beta release should also have reasonable number of already-migrated examples to demonstrate the ideas of multi-threading.

Interface changes in version 10 – after the beta release

- After the beta release (or even after the first reference tags), we invite feedbacks from our customers.
  - Interfaces visible to users would be iterated.
  - Hoping that iteration is just adding interfaces rather than changing them.
  - If necessary, more than one beta-releases may be made.
- Most, if not all, of examples we release with version 10 will migrate to all interface changes including multi-threading.
- Documents also will be updated accordingly.

- Staging???
  - As usual, new features / classes may be added at any minor release as long as they won't cause user's migration. Thus any functionalities, which we currently have but we cannot catch up necessary interface changes or assuring thread safety, may be staged as long as we release base interfaces with version 10.
  - Some GUI/Vis features may be supported only for sequential mode at version 10.

Migration required for user's multi-threaded code

Preliminary !

- Please note that this slide shows preliminary current design
- *main()*
  - *G4MTRunManager* instead of *G4RunManager*
  - New mandatory user initialization class *G4VUserWorkerInitialization*, which instantiates all the user action class objects for each thread
  - Define number of threads you want to use
- *G4VUserDetectorConstruction*
  - Split *Construct()* method to
    - *Construct()* : materials and geometry (common for all threads)
    - *ConstructSDAndField()* : sensitive detectors and field (thread-local)
- If you opt to stick on sequential mode, you do not need to change anything in you application code for multi-threading.
  - Some migration may still be necessary for obsolete classes.

Basic user's code

```
main()
{ G4MTRunManager* rm  = new G4MTRunManager();
  rm->SetUserInitialization(new UserDetectorConstruction);
  rm->SetUserInitialization(new PhysicsList);
  rm->SetUserInitialization(new UserWorkerInitialization);
  rm->SetNumberOfThreads(/*number of threads*/);
  rm->BeamOn(/*total number of events*/);
  …
}
void UserWorkerInitialization::WokerStart()
{  SetUserAction(new UserPrimaryGeneratorAction);
  SetUserAction(new UserSteppingAction);
  …
}
void UserDetectorConstruction::ConstructSDAndField()
{  SetSensitiveDetector(/*name of logical volume*/,
            new MySensitiveDetector(/*detector name*/) );
  …
}
```

Preliminary !

File I/O in user's code is a challenge

- Every file I/O for local thread is a challenge
  - Input : primary events
  - Output : event-by-event hits, trajectories
- G4MTRunManager collects run objects from worker threads and "reduces".
  - Scores
    - Footnote to educate ourselves ☺

      "A reduction combines all the elements in a collection into one using an associative two-input, one-output operator."

      http://www.drdobbs.com/architecture-and-design/parallelpattern-7-reduce/222000718
- Histograms
  - ROOT is thread-unsafe. Geant4 analysis tool (ROOT-bound) is thread-safe.
- Tracking action, stepping action
  - If you are accumulating quantities in your tracking action or stepping action in your current application, you should note that these action classes will be thread-local.

Collaboration-wide developments in 2012-2013

- Event/track level full reproducibility
  - Recently ATLAS identified an incorrect result in Geant4 physics interaction, which occurs only 20 times in 1 billion events (each event has millions of interactions).
    - But these 20 events were so significant that they all passed all the ATLAS event filters.
  - To pinpoint the problem, we do need event/track level full reproducibility.
    - As long as it starts with the same random number engine status, it should regenerate exactly the same result regardless of other conditions.
      - In particular, as we shift to multi-threading, we cannot reproduce the problem without track level reproducibility.
    - Geant4 caches many values for performance reasons. We need to identify each of them and add Clear() if necessary.
      - For example, if kinetic energy is very close to previous value, the previous cross section value is used rather than recalculation.
    - We are confident now that event level reproducibility is OK with v9.6 for most of the physics options, except CHIPS stopping and neutron HP.

Ongoing Collaboration-wide developments

- Performance improvements
  - Design iterations for some kernel classes
    - Cache-hit-rate improvement, reduction of virtual abstract layers, avoiding too deep recursive calls, etc.
  - Review implementations of physics and transportation
    - Many of these code were implemented by physicists who has poor knowledge of programming.
      - We must not loose physics performance, though. Massive verifications are required.
  - Changes must be transparent to user's code (at least for average users).
- Longer term
  - New trends
    - Hardware : GPGPU, Intel new generation chips, etc.
    - Programming language : CUDA, OpenCL, OpenACC, DSL, etc.
  - The Geant4 Collaboration acknowledges several pilot / prototyping projects worldwide which pursue major architectural revisions of Geant4.
  - We eager to make Geant4 faster.
    - Without sacrificing functionality, physics performance, flexibility.
    - We also want to be free from specific hardware / programming paradigm.