Geant4 MT: an update

J. Apostolakis for Geant4-MT developers

Xin Dong, Gene Cooperman (Northeastern Univ.)
Makoto Asai, Daniel Brandt (SLAC)
J. Apostolakis, G. Cosmo (CERN)

Outline

- Extending model of parallelism (TBB, dispatch) - CMS, ATLAS/ISF
  - Need to adapt to HEP experiment frameworks
- Folding of Geant4-MT into Geant4 release-10 (end 2013)
  - Streamlining for maintainability,
  - New major release: some interface changes are allowed.
- Challenge: assess and ensure the compatibility of these directions

Geant4MT - Background

- What is Geant4 MT ?
  - Goals, design, .. see background slides in backup (Purple header)
- It is the PhD-thesis work of Xin Dong (*Northeastern Univ.*)
  - under the supervision of Prof. Gene Cooperman, in collaboration with me (J.Ap.) - see paper in Europar and Xin's Thesis
    - Updated to G4 9.4p1 by Xin, Daniel, Makoto and Gabriele.
    - Updated to 9.5p1 by Daniel, Makoto and Gabriele.
- Performance: Good scaling, but overhead 1-worker vs. sequential
  - Excellent speedup from 1-worker to 40+ workers - see CHEP 2012 poster
    - But: Overhead vs Sequential found (first reported by Philippe Canal, 2011)

G4 MT Prototype - brief update

- MT updated to Geant4 9.5 patch01 - 15 Aug (Daniel Brandt, Makoto, Gabriele)
  - Improved integration of parallel main();
  - Corrected inclusion of tpmalloc.
- 'One-worker' overhead is now 18% - was reduced by 12% (Xin)
  - Change is using different gcc option to improve the 'interaction' of Thread Local Storage (TLS) and dynamic libraries
    - See A. Oliva and G. Araujo, "Speeding Up Thread-Local Storage Access in Dynamic Libraries", in GCC Developers' Summit 2006, 2006, pp. 159-178.

Adapting Geant4-MT for LHC Experiments

Adapting Geant4-MT for Experiments

- Request for support of '*on-demand*' parallelism
  - The CMS requirement
  - New trial usage in ATLAS ISF
  - Adapting to this requirement: Analysis and plans.
- Adapting process of migrating applications
  - review current recipe for migrating applications to MT
  - simplify for all applications
  - adapt to presence of HEP framework.

CMS & on-demand event simulation

- CMS model of concurrency: CMSsw creates tasks for evgen/sim/reco/digi, and its dispatcher (in TBB) manages the tasks
  - see presentation of Chris Jones on TBB (at last meeting)
- Request integration of G4-MT with 'on-demand' work model
  - workload is handled by outside framework (CMSsw, TBB= Thread Building Blocks)
  - unit of work: a full event.
- Q: How many changes are needed to adapt Geant4-MT to 'on-demand' / dispatch parallelism ?

ATLAS input

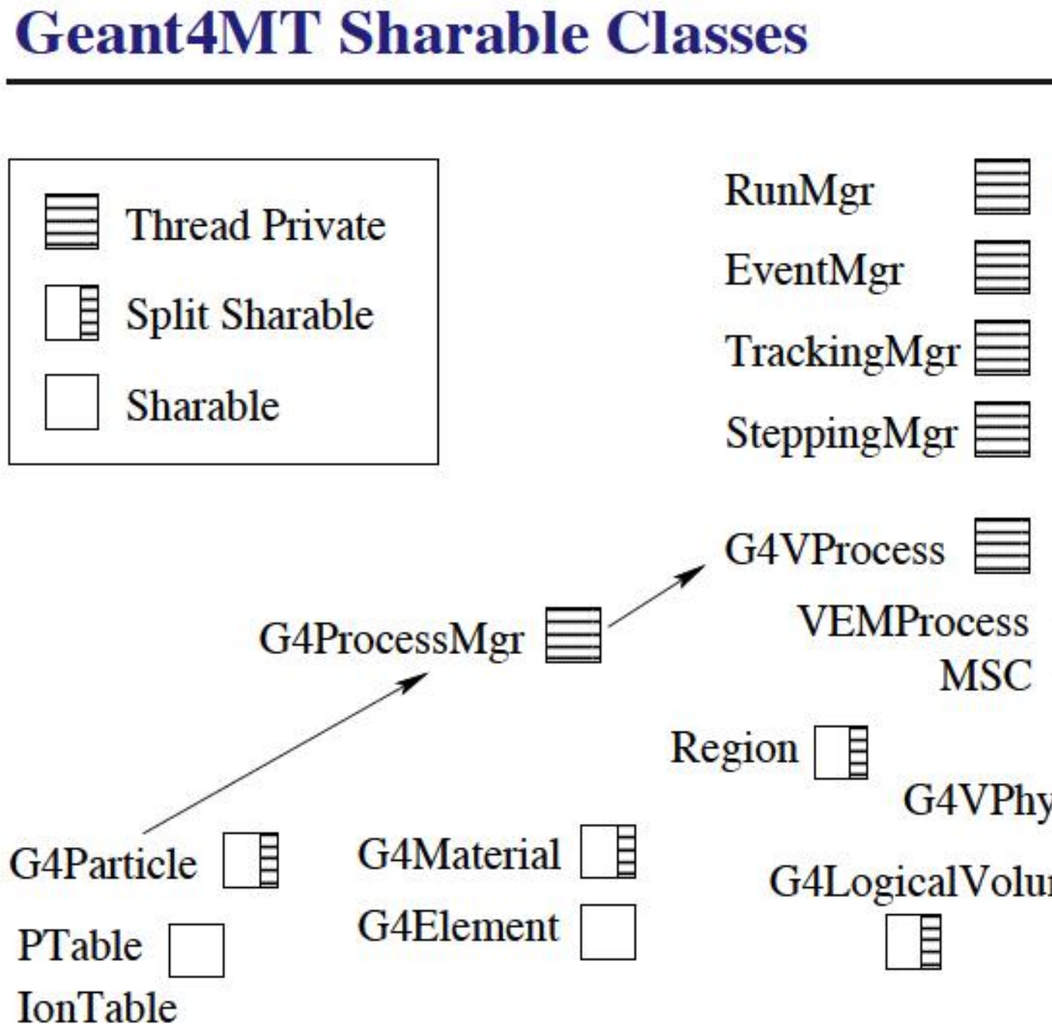
- The Integrated Simulation Framework (ISF) treats G4 uniquely:
  - it passes one track at a time to G4, packaged as a G4 ‘event’ - for each primary or one entering a sub-detector
- Developing trial use of Geant4-MT: pass each track to a separate worker
  - Sub-event level parallelization - using ‘event-level’ parallel Geant4-MT
- This is the first use of this capability / potential of Geant4-MT
  - It opens some new issues, in particular for output: hits, ..

Analysis: changes foreseen

- Needs are similar. Expect to know maximum number of workers.
- Must move from use of ‘thread-id’ to **worker-id**
  - any dependence in the code on thread-id must be replaced
- Each worker will require a **workspace**
  - this must be initialized - exactly as the thread’s workspace in G4MT today
- When **work** is ‘dispatched’ a workspace must be found
  - it could be assigned with the work (CMS model: pass worker id in request)
  - or identified by our system (likely at a small cost for locking.)

Draft Plans

- Create prototype ‘on-demand’ G4-MT
  - Adapt initialization of workspaces
  - Use & propagate worker-id in key G4 classes - instead of thread-id
- Issues to check
  - Ensure that Thread Local Storage (__thread) is compatible with TBB
- Schedule
  - Prototype ‘on-demand’ by end-November.



Migrating applications to G4MT

Pere Mato

- Review current recipe for migrating applications to MT
  - Simplify for all applications and
  - Adapt to presence of HEP experiment frameworks.
- Typical issue:
  - A logical volume (LV) must have many Sensitive Detectors (SD) - one per worker
  - How to create each additional SD per worker, and attach it to the LV ?
    - and with small or no changes to the experiment code?

Performance and Portability

Performance and portability

- Performance
  - Good scaling from 1-worker to 40 cores (+25% gain with hyperthreading.)
  - The '*one-worker*' slowdown
- Portability
  - Use of __thread gcc extension (thread_local in C++ 11)
  - Today's prototype is restricted to Linux
    - Know how to extend to Windows; not clear how to port to Mac OS X.
  - Potential to use C++ 11 Threads in future.

The ‘one-worker’ slowdown

- Philip Canal reported ~30% cost (Sept 2011) one-worker MT vs sequential G4
- Xin Dong identified *the key reasons:*
  - the interaction of Thread Local Storage (TLS) and dynamic libraries
  - calls to *get_thread_id()* - singleton TLS & our “TLS for objects”
- Using improved gcc option, Xin *reduced* overhead to *18%*

The ‘one-worker’ slowdown

- *Need more benchmarks and profiling.* Current known causes:
  - interaction of Thread Local Storage (TLS) and dynamic libraries?
  - calls to *get_thread_id()* - singleton TLS & our “TLS for objects”
- *Can we avoid slowdown* from interaction of TLS & dynamic libraries?
  - *Proposal :* try putting all of G4 into one shared library
  - *First trial*: use static libraries in benchmarks.

  - *Alternative:* put the core of Geant4 into one library, excluding only auxiliaries (that can have external dependencies): persistency, visualization.

C++ 11 Threads – Marc Paterno

- std::thread has great potential for portability
- New capabilities – move from C to C++
  - Full checking of arguments
  - C++ type mutex locks: safe for exceptions
  - Sentry object to guard resource
- Status: gcc 4.7.1 with flag –std=c++11
  - Has std::thread
  - Does **not** have '*thread_local*' TLS. Does '*__thread*' co-work w std::thread?

Geant4 MT - next steps

- SFT prototype of ‘on-demand’ parallelism: November 2012
- Geant4 9.6-MT: February 2013 (tbc)
  - reduce number and types of changes in MT - to ease merge
  - simplify migration of application code.
- Geant4 10-beta release (June 2013)
  - Multi-threading included in ‘base’ code (choice at installation)
  - Interface changes: plans and path (see appended slides, adapted)
- Geant4 10 production release (Dec 2013)

Summary

- Geant4 MT was updated to 9.5-patch 01
- Adapting G4-MT for ‘on-demand’ work
  - Analysis is done
  - Challenge is to see how many adaptations (thread to worker)
  - Plans to create prototype by end-November.
- Performance: Scaling is excellent
  - Seeking new solutions for ‘single-worker’ slowdown
- Geant4 MT will be integrated into Geant4 release 10 (beta: June)

Backup slides

References

- [Europar] "Multithreaded Geant4: Semi-automatic Transformation into Scalable Thread-Parallel Software", Xin Dong, Gene Cooperman and John Apostolakis, *Proc. of Euro-Par 2010* -- Parallel Processing, Lecture Notes in Computer Science 6272, Springer, 2010, pp. 287-303.

Intro to Geant4-MT

J. Apostolakis

Outline of the Geant4-MT design

- There is one *master thread* that
  - initializes the geometry & physics - data is write-once, then read-only
  - then spawns workers, and awaits their termination.
- The worker threads
  - create their work area and initialize their instances and
  - execute all the ‘work’ of the simulation.
- The unit of work for a worker is a Geant4 event
  - limited sub-event parallelism was foreseen by splitting a physical event (collision or trigger) into several Geant4 events.
- Choice: limit changes to a few classes
  - other classes have a separate object for each worker

Goals of Geant4-MT

Key goals of G4-MT

- allow full use of multi-core hardware (including hyper-threading)
- reduce the memory footprint by sharing the large data structures
- enable use of additional threads within limited memory
- reduce cost of memory accesses.

Next target: Make Geant4 thread-safe (Geant4 10 beta - June 2013)

- for use in multi-threaded applications.

Longer term goal - a personal view:

- increase the throughput of simulation by enabling the use of additional resources: additional hardware threads, latency hiding, co-processors, ...

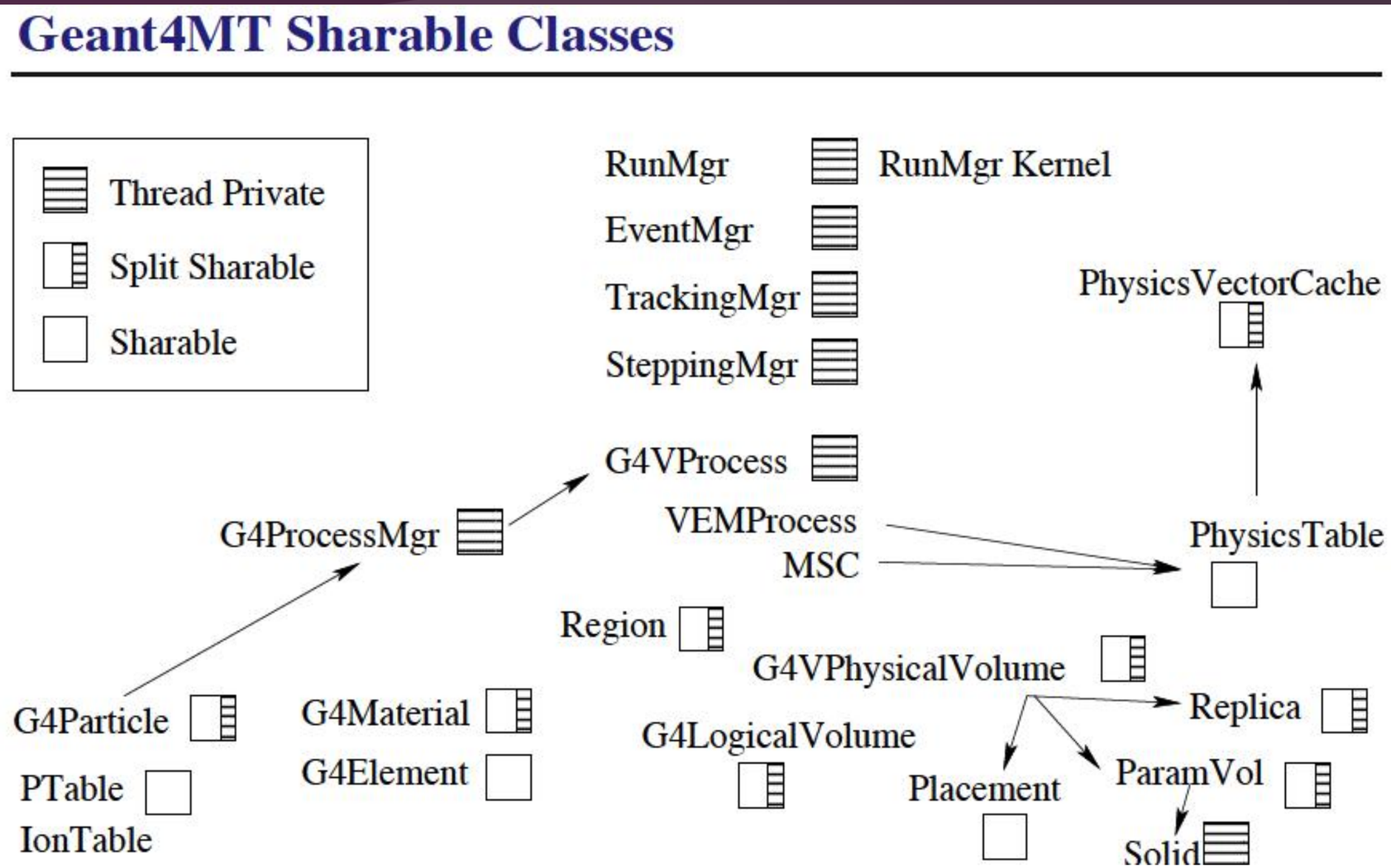
Implementation

- Uses the POSIX threads library (pthreads)
  - currently works only on Linux.
- Global data is separated by thread
  - using the gcc construct __thread - this includes singletons.
- The master thread
  - reads all parameters
  - initializes all the geometry
  - calculates any tables of data: cross sections, dE/dx, range, ..
  - starts the worker threads.
- Instances of separate objects are cloned by each worker
  - copying the contents of all the corresponding objects in the master thread (deep copy where needed.)

Shared data - and 'split' classes

- Most **shared** ‘data’ is *invariant* in the event loo
- *Challenge*: ion table
  - either must be initialised with all 3000 nuclei, or
  - locking must be used to update.
    - Choice: initialise.

- Some classes are **split**:
  - part of their data is *shared*, and
  - part is *thread local*.
- Example: G4Electron has
  - One set of properties
  - Separate ProcessMgr per thread



Limit extent of changes

- The choice was to concentrate revisions to a few classes
  - to reduce the effort required to create, test and maintain it

- The few *classes* that are *changed* are ones that
  - manage the event loop
  - touch geometry objects with multiple physical instances (replicas etc.)
  - must share cross-sections for EM processes,
  - which create or configure the above classes.

- All other classes are unchanged
  - a separate object is created by each worker.

'Split' classes

- Implementation of split classes is via a customized method:
  - each instance of a split object has an integer ***id***
  - each thread instantiates an *array of stub object* for each split class;
  - an object uses the entry in the array - index= int ***id***
  - the (sub-)object data is initialized by the worker thread that uses it.

Merging Geant4-MT to the main development trunk and deadlines

Adapted from slides of Gabriele Cosmo, CERN PH/SFT

Geant4 version 10

- The release in 2013 will be a major release
  - New functionalities and interface changes are allowed
  - Classes and codes declared obsolete can be removed
- Will incorporate multi-threading capability
  - Will require user-code migration to exploit this feature
  - Current method is described in Makoto's talk - *to investigate simplifying and adapting to external frameworks*
- Will offer two build options
  - Multi-threaded mode (including single thread) - default
  - Sequential mode
    - To allow user with code depending on thread-unsafe external libraries to still use the new version
    - To facilitate migration and porting of features

Merging Geant4-MT features

- Import and integrate Geant4-MT utilities code
  - Translate relevant code to C++
  - Encapsulate global data and functions to utility classes
  - Inspect code and module dependencies
  - Generalize use/activation of thread-local-storage (TLS) for porting on supported platforms
- Implement thread-safety for sharable classes, thread-private data and volatile class members
  - Where possible, investigate possibility to modify existing code for thread safety (improper use of static/const or mutable variables)
  - Proper implementation of TLS for global and static variables
- Provide utilities and minimize exposure of thread-specific code in user classes

Timeline

- Geant4 9.6 release on November 30
  - Final release of version 9 series
  - Must already include “obsoleting” messages for functionalities planned for removal in the major release of 2013
  - Can already include first changes for thread-safety
- December 2012 / January 2013
  - Conversion of release 9.6 to new Geant4-MT prototype
  - Adiabatically integrate Geant4-MT 9.6 features to main development trunk
- February - May 2013
  - Adaptation of CMake build system and integration testing; migration of tests and key examples
  - Stress tests for CPU and physics performance
- June 2013
  - Beta release: all changes for multi-threading and interfaces should be included
  - **Feedback** from our customers and users will be essential
- December 2013
  - Final release

After Beta release …

- Feedback from our customers and users will be essential
  - To review/refine and eventually improve the API
  - To spot problems or use-cases not considered
- All examples being released will have to migrate to the new API and guarantee to work also in sequential mode
- Documentation 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 functionality, which we currently have but cannot catch up necessary interface changes or assuring thread safety, may be staged as long as we release base interfaces with version X