GEant4 Parallelisation

J. Apostolakis

Session Overview

- Part 1: Geant4 Multi-threading
  - C++ 11 threads: opportunity for portability ?
  - Open, revised and new requirements (from HEP experiments)
- Part 2: Beyond MT
  - Geant4 on GPUs: prototypes
  - The ‘Geant’ prototype - moving towards Vector

Part 1: Geant4MT & new requests

Geant4 MT - major topics

- New Requirements (2012)
  - Extending model of parallelism (TBB, dispatch) - CMS
  - Adapting to HEP experiment frameworks
- Folding of Geant4-MT into Geant4 release-X (end 2013)
  - Streamline for maintainability, ...
- Need to assess and ensure the compatibility of these directions

Geant4MT - Background

- What is Geant4 MT ?
  - Goals, design, .. see background slides in Addendum (Purple header)
- Implementation is the PhD-thesis work of Xin Dong (NorthEastern Univ.) under the supervision of Prof. Gene Cooperman, in collaboration with me (J.Ap.)
  - Updated to G4 9.4p1 (X+D+M+G), & 9.5p1 by Daniel, Makoto and Gabriele.
- Excellent speedup from 1-worker to 40+ workers - see [CHEP 2012 poster]
  - But: Overhead vs Sequential found (first reported by Philippe

Geant4 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.
- Improvements to ‘one-worker’ overhead - now decreased from 30% to 18% (Xin)
  - Due to the interaction of Thread Local Storage (TLS) and dynamic libraries

Goals of Part 1

- Geant4 MT and its future
  - Evaluate whether C++ 11 threads can replace pthreads (soon)
  - Identify issues, roadblocks for ‘on-demand’ version of G4MT
  - Note issues which arise from other new requirements.

Topics of Part 1 - Geant4MT

- C++ 11 Threads and Portability
  - Talk by Marc Paterno
- Request for support of ‘on demand’ parallelism
  - Talk in plenary by Chris J., Liz S.-K. (CMS)
- New trial usage in ATLAS ISF
- Discussion on these & related topics

C++ 11 threads:

- Do C++11 ‘standard’ threads enable better portability (than pthreads) ?
- What other benefits can C++11 threads offer ?
- Are they available today - or soon ?

CMS & on-demand event simulation

- Plenary presentation (Chris Jones, Eliz. Sexton-Kennedy)
- Request integration into on-demand event simulation
  - workload is handled by outside framework (CMSsw, TBB= Thread Building Blocks)
  - unit of work: a full event.
- What is required to adapt Geant4-MT to ‘on-demand’ / dispatch parallelism ?
  - Key topic of Discussion session

ATLAS input

- Developing trial use - in new Integrated Simulation Framework (ISF)
- Passes one track at a time, packaged as a G4 ‘event’ - for each primary or one entering a sub-detector
  - Sub-event level parallelization - using ‘event-level’ parallel Geant4-MT
- This is the first use of this capability / potential

The ‘one-worker’ slowdown

- Need more benchmarks and profiling. Current known causes:
  - interaction of Thread Local Storage (TLS) and dynamic libraries?
  - extra calls to get_thread_id() - in singleton TLS and our “TLS for objects”
- Can we avoid the slowdown due to interaction of (TLS) and dynamic libraries?
  - Proposal : try putting all of G4 into one shared library
  - Or put the core - ‘nearly all’ - into one library, excluding only auxiliaries: persistency, visualization.

Other Topics for Discussion

- Your issues here

Part 2: Beyond threads/tasks

Overview

- Need for more events by LHC/HEP experiments, medical users, ..
- Challenge in CPUs: instruction fetch is bottleneck due to ‘granular’ OO methods, large number of branches, code size large compared to caches.
  - Each instruction, method does too little work
- How to get more out of each instruction - and utilize the emerging architectures: GPUs, MIC, CPU with wider SIMD execution units?
  - Explore GPUs and Vectors

Opportunities

- CPU evolution - wider Vector Units + instructions:
  - Widespread: CPUs with128-bit units = 2 doubles or 4 floats
  - Emerging: 256-bit (AVX) = 4 doubles or 8 floats
- GPUs: SIMD hardware, specialised languages (CUDA, OpenCL)
  - Hundreds of ‘threads’, tens of - one MultiProcessor
- MIC: New public information - wide Vectors, 4 threads per core, ~60 cores

G4 - GPU efforts: external

- GPU efforts external to G4 Collaboration
  - hGATE project, full gamma processes (2011), e- in progress: CUDA
  - G4MCD - team in Germany, both gamma and e-
- These efforts focus on use in medical physics
  - Simple geometry (regular voxel volumes - trivial geometry, no Navigator.)
- In touch with hGATE, part of OpenGATE: D. Visvikis (Brest)

Intro to Geant4-MT

J. Apostolakis

Outline of the Geant4-MT design

- There is one master thread that initialises and spawns workers; and several worker threads that 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.

- *Looking forward - a personal view:*
- Medium term goals: make Geant4 thread-safe (Geant4 X - Dec 2013)
  - for use in multi-threaded applications.
- Longer term goal
  - increase the throughput of simulation by enabling the use of additional resources: co-processors and/or additional hardware threads.

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.

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 initializes all data
  - reads all parameters and starts the other threads;
- Instances of separate objects are cloned by each worker
  - copying the contents of all these objects in the master thread (shallow copy or deep copy ?)

'Split' classes

- Some classes are split:
  - part of their data is shared, and
  - part is thread local.
- Shared data
  - is typically invariant in the event loop
  - but also 'joint' and updated: ion table, particle table.
- Implementation - customized methodology
  - each instance of split object has an integer ***id***
  - instantiates an *array of stub object* for each thread
  - an object uses the entry in the array - index= int ***id***
  - the (sub-)object data is initialised by the worker thread that uses it.