Experience with Multi-Core Optimization for Geant4-MT

Gene Cooperman
College of Computer and Information Science
Northeastern University, Boston, USA
gene@ccs.neu.edu

Achievements of Geant4-MT (MultiThreaded)

1. Plans to merge Geant4-MT into main Geant4 distribution (collaboration with Makoto Asai, Gabriele Cosmo, Xin Dong, Daniel Brandt)
2. Geant4-MT shown to scale linearly through the 80-core level on an Intel 80-core machine (reported by Sverre Jarp, OpenLab, CERN)
3. Geant4-MT is one of the first complex benchmarks on which the Intel MIC architecture (Knights Tour/Knights Ferry) was tested.
   From: `http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html`

   "Consider the example of CERN OpenLab, the European Organization for Nuclear Research. This group took advantage of the Knights Ferry kit . . . to migrate a complex benchmark written in C++ code to the new architecture in just a few days. According to Sverre Jarp, CTO of the CERN open lab, 'The familiar hardware programming model allowed us to get the software running much faster than expected.' "

Acknowledgment: Thanks for frequent help and encouragement of John Apostolakis for both Geant4-MT and the older ParGeant4.

History of Geant4-MT and Earlier ParGeant4

- ParGeant4 (distributed nodes on a cluster, see examples/advanced in Geant4 distro): National Science Foundation grant ACR-9872114, "Parallel Infrastructure for Recognition of Non-Local Patterns", 1999–2002 (with co-PIs: Stephen Reucroft and John Swain)
- "Using TOP-C and AMPIC to Port Large Parallel Applications to the Computational Grid", G. Cooperman, H. Casanova, J. Hayes and T. Witzel, Proc. of 2nd IEEC/ACM International Symposium on Cluster Computing and the Grid (CCGrid), IEEE Press, 2002, pp. 120-127 (updated version in Future Generation Computer Systems **19**(4))
- 2007: Start of multi-threaded Geant4; result of Cooperman's sabbatical at CERN
- April, 2008: "First Results in a Thread-Parallel Geant4", Gene Cooperman (joint with Xin Dong), *Workshop on Virtualization and Multi-Core Technologies for LHC*, CERN; (later reports at Geant4 Collaboration Meetings: 2008, 2009, 2010, 2011)
- "Multithreaded Geant4: Semi-automatic Transformation into Scalable Thread-Parallel Software", X. Dong, G. Cooperman and J. Apostolakis, *Proc. of Euro-Par 2010*, Lecture Notes in Computer Science **6272**, Springer, 2010, pp. 287–303
- 2012–13: Plans to incorporate Geant4-MT in main branch of Geant4 (in collaboration with Makoto Asai, Gabriele Cosmo, Xin Dong)

History of Geant4-MT and Earlier ParGeant4 (Part 2)

- 1999–: ParGeant4 (distributed Geant4): Non-disruptive: Geant4 C++ design consists of many virtual methods; Replace one library (G4RunManager) by parallel library (ParG4RunManager).
- 2007–: Geant4-MT (MultiThreaded Geant4): “tearing Geant4 apart”

1. Semi-automatic thread-parallelization (see thesis of Xin Dong)
2. Source-code transformation of Geant4 based on modifying GNU g++ parser
3. Replace one library (G4RunManager) by thread-parallel library (ParG4RunManager).
4. Plan for Prototype: If a new Geant4 release very 6 months, then transform Geant4 code base every 6 months
5. Current plans to modify Geant4 code base to include Geant4-MT (no more source code transformations; Geant4-MT is maintained as part of Geant4 itself) — led by Makoto Asai and Gabriele Cosmo

Strategy for Thread Parallelization

Note: Geant4 has 3/4 million lines of C++ code, with new release every six months. See

talk of Xin Dong for details of thread parallelism:
`https://indico.fnal.gov/getFile.py/access?contribId=116&sessionId=12&resId=0&materialId=slides&confId=4535`

1. Move read-write fields of objects to TLS (thread-local storage) This data cannot be shared by all threads. (And use `__thread` keyword for rest)
2. Reduce memory footprint: Identify large-memory objects that can be shared among all threads.
3. Verify correctness: thread-shared data should now be read-only; remove write permission to verify (Note: see issue of reproducibility on next slide.)

Lessons from Geant4-MT (MultiThreaded)

1. Global variables with frequent writes can cause mysterious loss of linear scaling above 12 cores (e.g. our experience with **G4cout.precision**)
2. *Consequence:* Implementations of multi-threaded malloc are not linearly scalable with many threads!
   The malloc standard requires that any thread can free a malloc buffer allocated by *any other* thread. This requires a central point of control.
   - *Solution:* Build a custom malloc library for the common case in Geant4-MT: The thread that allocates a buffer is the one that frees it.
3. Geant4 does lazy initialization
   - *Issue:* Geant4 is close to trivially parallel, but only after the initializations are completed.

Issues for All Parallelizations of Geant4

1. *Multi-core implementations:* Decisions about how to share fields of objects (Many Geant4 classes include writable fields to cache intermediate computations.)
2. *Multi-core implementations:* Global variables with frequent writes (e.g. **G4cout.precision**, even though value *not used* when `G4verbose` not defined)
3. **Reproducibility:**
   Random number generators: should the random seed be copied to all threads or set independently
   - For Parallel Deterministic Reproducibility: Set random seed excplicitly at beginning of each task (each event, in case of Geant4-MT)
   - To Reproduce Sequential Geant4: Set number of worker threads to one: each task will have same thread as sequential Geant4

Speculation for the Next Big Issue

1. Future architectures all appear to stress multiple levels of a memory hierarchy:
   (a) Intel MIC (Many Integrated Core)
   (b) NVIDIA "Project Denver" (NVIDIA GPU + on-board ARM CPU)
   (c) AMD Fusion
2. All of these architectures stress *many small and light cores*.
3. Small and light cores will have small L1 and L2 caches.
4. Yet some sharable Geant4 objects require many megabytes of memory.

A possible strategy is *stratified processing of events* according to particle type: Events dealing with the same type of particle can share memory (e.g. shared L2 cache) more easily.