Multi-threading capabilities in Geant4 Version 10.0

A. Dotti for Geant4 Collaboration

Technical Forum 10 Dec 2013

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Introduction

SLAC

- **Event level parallelism** via multi-threading (POSIX based)
- Built on top of experience of G4MT prototypes
- Main design driving goal: **minimize user-code changes**
- Integrated into Version 10.0 codebase

- Proof of principle
- Identify objects to be shared
- First testing

- MT code integrated into G4

- API re-design
- Example migration
- Further testing
- First optimizations

- Public release
- All functionalities ported to MT

- Further Refinements
- Focus on further performance improvements

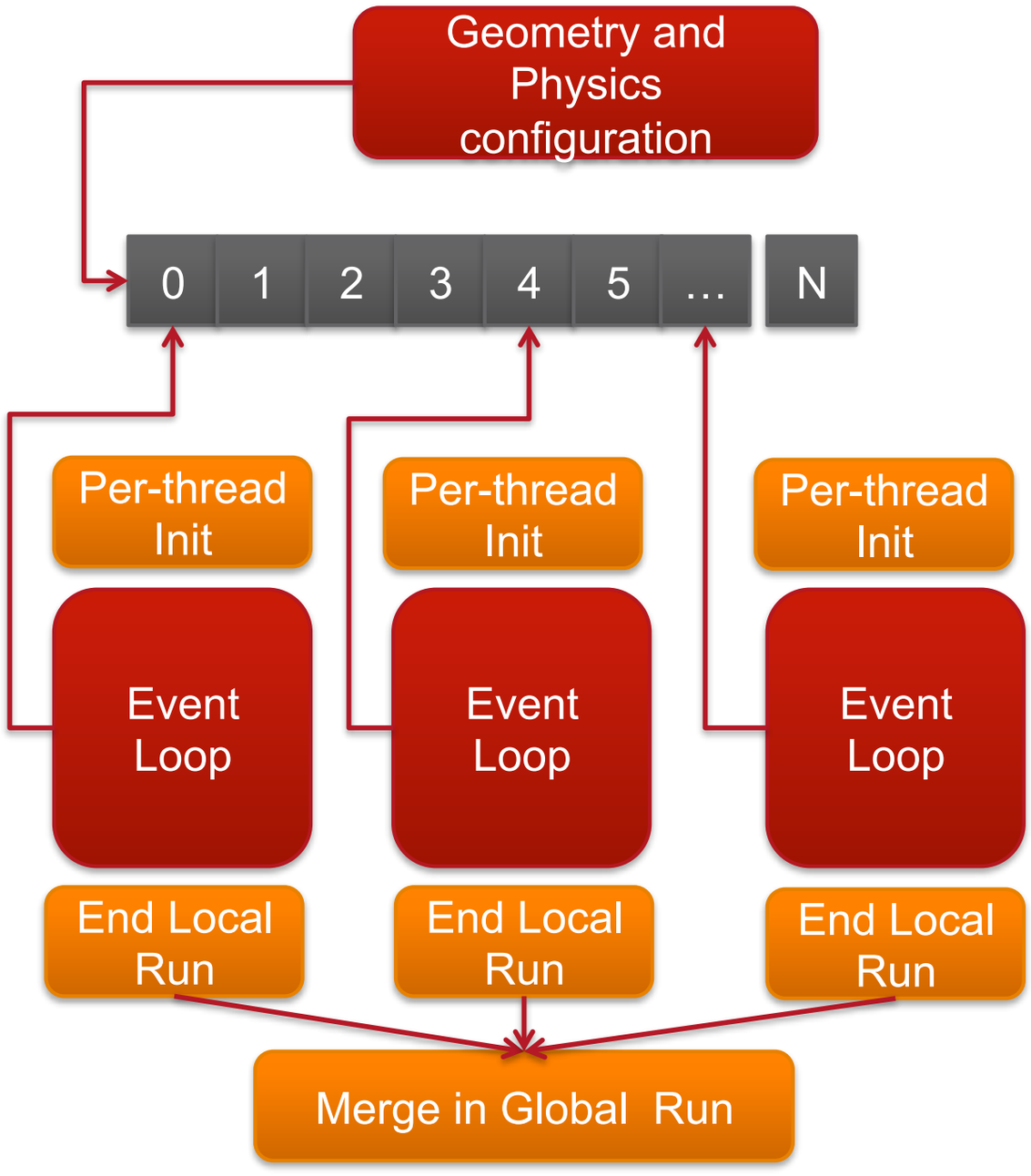
Geant4 Version 10.0: how to use

SLAC

- Multi-threading **activated at configuration** step via cmake option
  - -DGEANT4_BUILD_MULTITHREADED=ON (default OFF)
- Code changes:

  1. Instantiate G4MTRunManager (or user-derived) instead of G4RunManager
  2. Split detector construction in two:
     - Construct() method : geometry, shared among threads
     - ConstructSDandField() method (new) : SensitiveDetector and B-Field, thread-local
  3. Even if user-actions are thread-local, user should verify thread safety
  4. Complex experimental framework may need additional code changes
- Sequential application can run **without changes** with MT-enabled Geant4 build

Multi-threading master/worker model



Per-event seeds pre-prepared in a “queue”

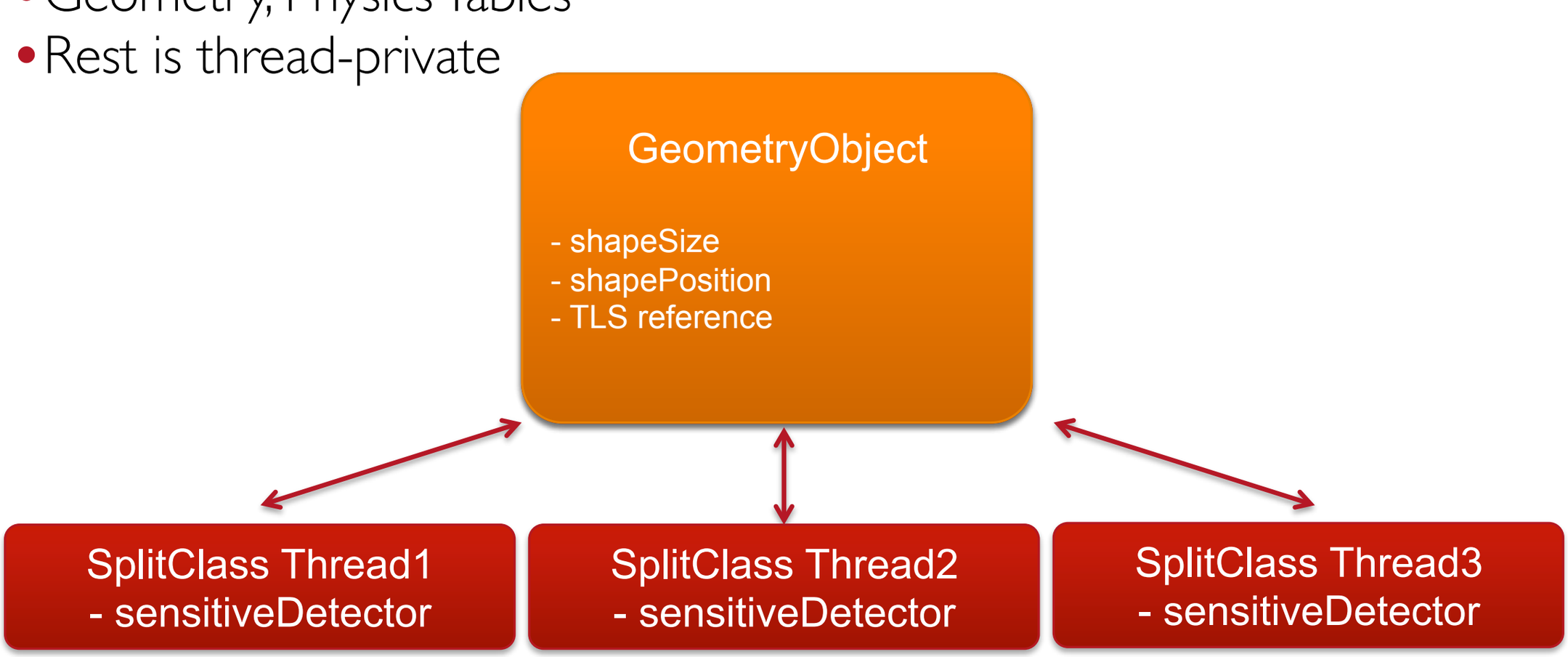
Threads compete for next event to be processes (new in ref-08)

Command line scoring and G4tools automatically merge results from threads

Thread-safety in Version 10.0

SLAC

- Design: lock-free code during event-loop
- Thread-safety implemented via **Thread Local Storage**
- "Split-class" mechanism: reduce memory consumption
  - Read-only part of most memory consuming objects shared between thread
  - Geometry, Physics Tables
  - Rest is thread-private

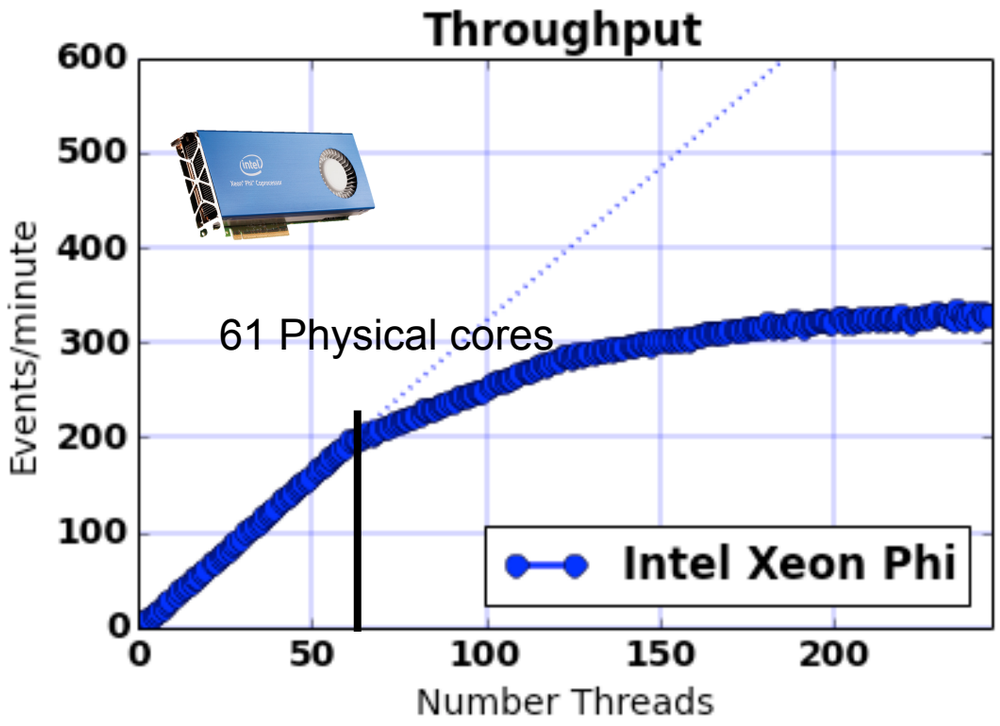


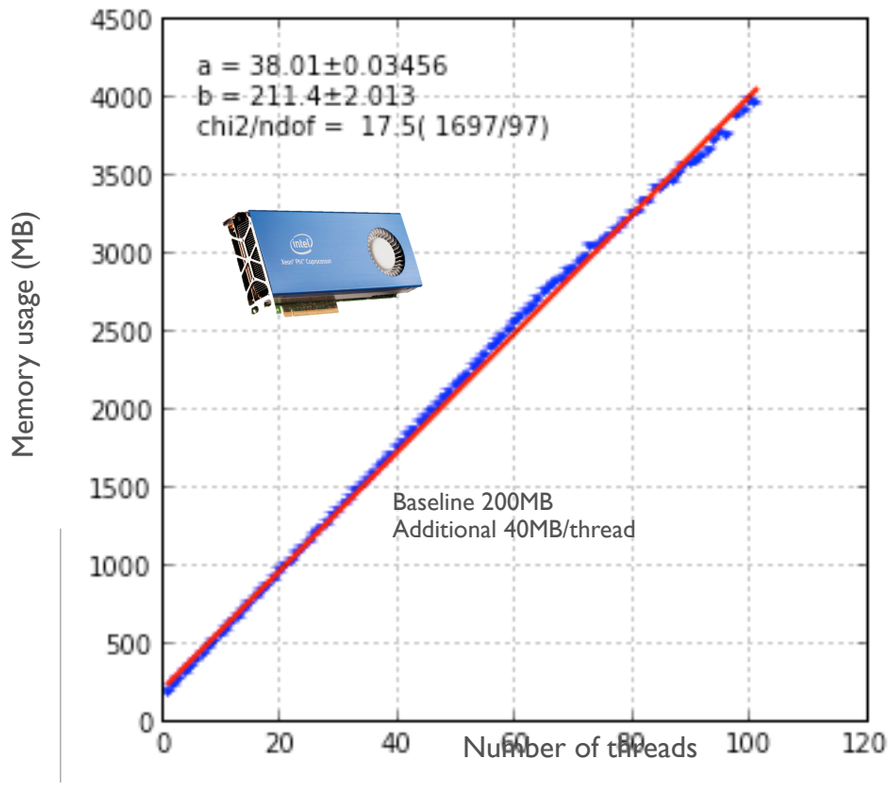
Results

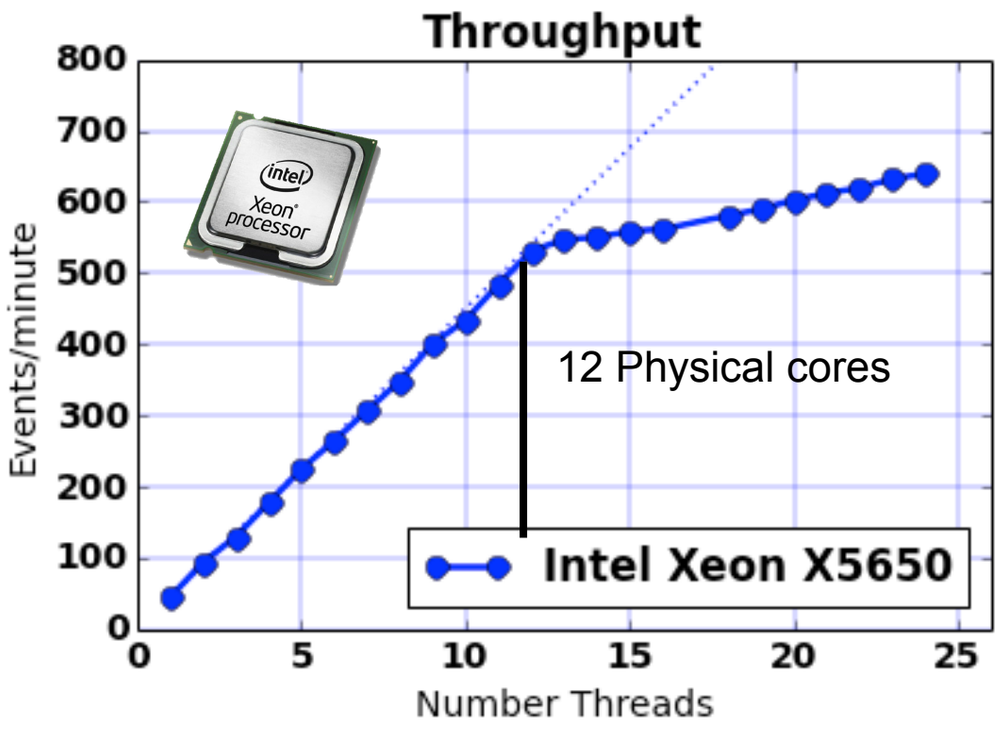
- **Fully reproducible:** given an event and its initial seed the RNG history is independent of the number of threads and order in which these are simulated
  - Corollary: given the seeds, sequential and MT builds are equivalent

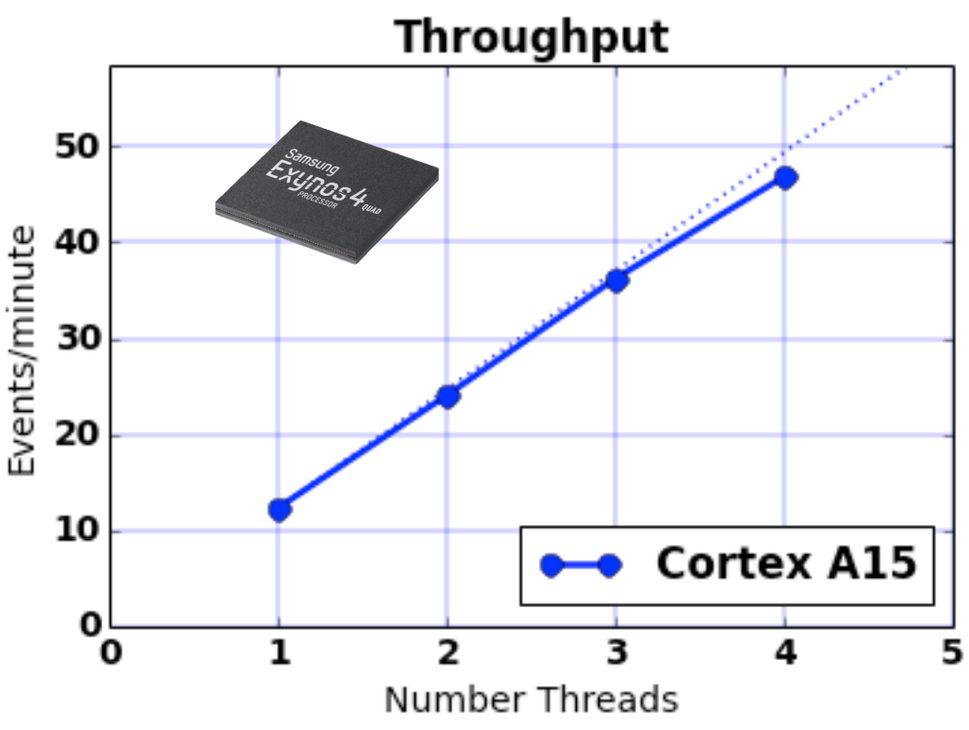
- MT functionality introduce **minimal overhead** in single thread (~1%)
- Very good **linear scalability** up to very large number of threads O(100)
- Good **memory reduction**: only 30-50MB/thread (depends on application, these are for CMS geometry, FTFP_BERT physics, but no SD)
- Hyper-threading adding additional +20% throughput
- Working out-of-the-box with success on different architectures x86, ARM, MIC, Atom

Results



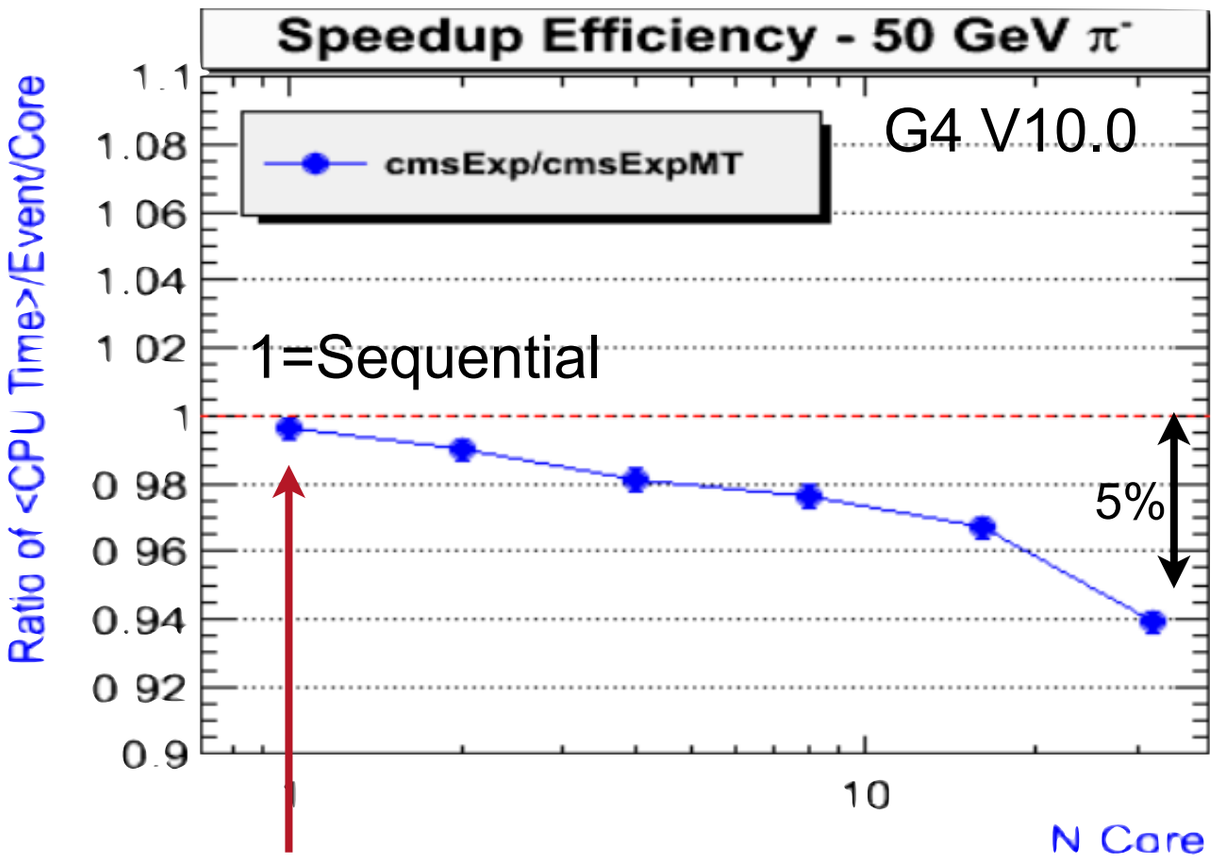




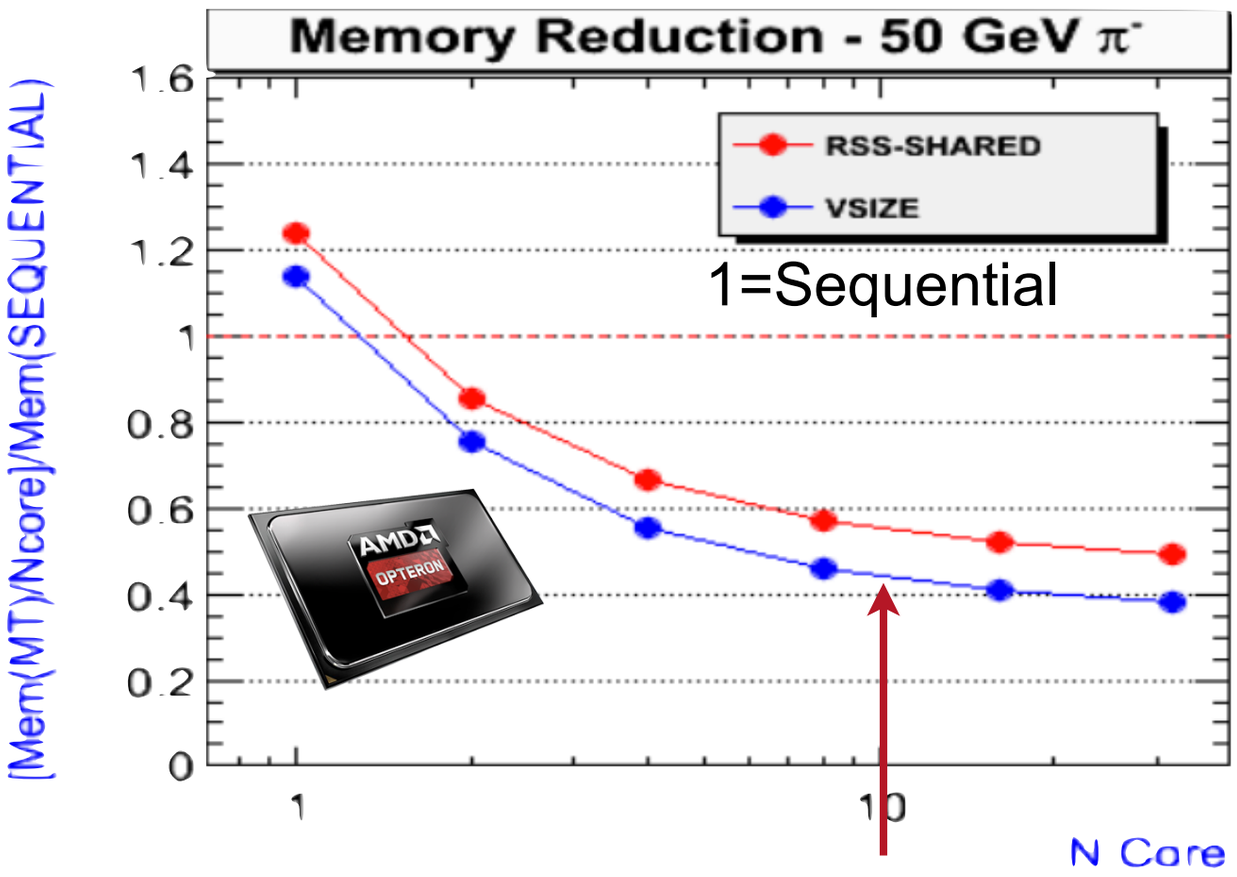


NB: not final release

Comparing to sequential



1 thread =>
Overhead for MT
Very small CPU penalty
~1%



10 threads =>
50% memory w.r.t.
10 sequential instances

Courtesy of S.Y. Jun (FNAL)

Conclusions

- Multi-threading for event-level parallelism in Geant4:
  - Feasibility studies and prototyping has been successfully concluded in 2012
  - In 2013 Geant4 code has been:
    - Made thread-safe
    - Extended to allow event-level parallelism
    - API improvements to simplify as much as possible usage from users
  - For 2014 and beyond:
    - CPU/Memory improvements
    - Improve support for external parallelism frameworks (first prototypes available for TBB, MPI)
- **Very good results obtained:**
  - **Same physics results as for sequential**
  - **Good linearity and memory reduction**
  - **Working on different architectures**
- Documentation has been updated:
  - Application Developer's guide : How to migrate to MT
  - Toolkit Developer's guide : Internals of MT for G4 developers (but can be useful for users too)
  - Twiki: https://twiki.cern.ch/twiki/bin/view/Geant4/MultiThreadingTaskForce