Kernel III

Makoto Asai (SLAC)
Geant4 Tutorial Course
Contents

• Fast simulation (Shower parameterization)
• Multi-threading
• Computing performance
Fast simulation
(shower parameterization)
Fast Simulation - Generalities

• Fast Simulation, also called as shower parameterization, is a shortcut to the "ordinary" tracking.

• Fast Simulation allows you to take over the tracking and implement your own "fast" physics and detector response.

• The classical use case of fast simulation is the shower parameterization where the typical several thousand steps per GeV computed by the tracking are replaced by a few ten of energy deposits per GeV.

• Parameterizations are generally experiment dependent. Geant4 provides a convenient framework.
Parameterization features

- Parameterizations take place in an **envelope**. An envelope is a region, that is typically a mother volume of a sub-system or of a major module of such a sub-system.
- Parameterizations are often dependent to particle types and/or may be applied only to some kinds of particles.
- They are often not applied in complicated regions.
Models and envelope

- Concrete models are bound to the envelope through a G4FastSimulationManager object.
- This allows several models to be bound to one envelope.
- The envelope is simply a G4Region which has G4FastSimulationManager.
- All [grand[…]daughter volumes will be sensitive to the parameterizations.
- A model may return to the "ordinary" tracking the new state of G4Track after parameterization (alive/killed, new position, new momentum, etc.) and eventually adds secondaries (e.g. punch-through) created by the parameterization.
Fast Simulation

• The Fast Simulation components are indicated in white.

• When the G4Track comes in an envelope, the G4FastSimulationManagerProcess looks for a G4FastSimulationManager.
  • If one exists, at the beginning of each step in the envelope, each model is asked for a trigger.
  • In case a trigger is issued, the model is applied at the point the G4track is.
  • Otherwise, the tracking proceeds with a normal tracking.
Multi-threading
Different levels of parallelism

- **Job execution** = $O(1\sim 10)$ runs
  - X-section data files, external geometry description

- **Run** = $O(10^2\sim 10^9)$ events
  - X-sections in memory, optimized geometry in memory, histograms
  - Event loop

- **Event** = $O(10\sim 10^9)$ tracks
  - Primary tracks, secondary tracks
  - Hits, score

- **Track** = $O(1\sim 10^3)$ steps
  - Travelling in geometry
  - Generating secondary tracks

- **Step**
  - Geometrical navigation
  - Physics processes
  - Hits, score

$\iff$ Multi-job
$\iff$ Run parallelism
$\iff$ Event parallelism
$\iff$ Track parallelism
DIANE (Distributed ANalysis Environment)

- DIANE is a tool which helps application communities and smaller Virtual Organizations using the distributed computing infrastructures more efficiently. The automatic control and scheduling of computations on a set of distributed worker nodes leads to an improvement of the quality of service of the EGEE/LCG Grid.
- This is a “multi-job” approach based on GRID environment,
- Geant4 offers one example to illustrate the use of DIANE.
- Similar approaches with commercial Cloud computing facilities are seen in company users.
MPI (Message Passing Interface)

- MPI is a language-independent communications protocol used to program parallel computers. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today. MPI is not sanctioned by any major standards body; nevertheless, it has become a de facto standard for communication among processes that model a parallel program running on a distributed memory system.
- Geant4 offers a built-in MPI layer. Currently, LAM, MPICH2 and Open MPI are supported. Geant4 also offers a couple of examples which illustrate the use of MPI.
- “Run parallelism” approach.
- For example of exMPI01 in geant4.9.3/examples/extended/parallel/MPI, core i7 took 122 seconds (single thread), 62 seconds (2 threads) and 34 seconds (4 threads) wall clock time.
Parallel computing

- Using OpenMPI (Message Passing Interface)
- Referring to the example/extended/parallel/MPI.
- JAEA PC-cluster A (Altix350/32 node, Intel Itanium2 : 64CPU)
- 46 time faster by 64CPU
  - For $10^9$ events: ~81 hours by PC (PentiumM 1.7GHz)
    -> 4.6 hours by PC-cluster A (24CPU)
- $10^{10}$-$10^{13}$ events for irradiation time of 1 sec.
TOP-C (Task Oriented Parallel C/C++)

- TOP-C is a tool of “task-oriented” master-slave architecture to make an application parallelized with a distributed memory model based on MPI.
  - Shared memory model (thread-based for a multi-processor node) is under development. See later slides.
- TOP-C is developed and maintained by G. Cooperman and his team at Northeastern University.
  - [http://www.ccs.neu.edu/home/gene/topc.html](http://www.ccs.neu.edu/home/gene/topc.html)
- Geant4 offers a couple of examples to illustrate the use of TOP-C.
- “Event parallelism” approach.
## Parallelism

- GRID (and Cloud) is surely a valid option. But it is outside of parallelization of Geant4 itself.
  - Use it if you have such an environment.
- For parallelism inside Geant4, we can qualitatively say:

<table>
<thead>
<tr>
<th></th>
<th>Run parallelism</th>
<th>Event parallelism</th>
<th>Track parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network traffic</td>
<td>+ (less traffic)</td>
<td>- (more traffic)</td>
<td>--- (much more traffic)</td>
</tr>
<tr>
<td>Load distribution</td>
<td>-</td>
<td>+</td>
<td>???</td>
</tr>
<tr>
<td>Advantage for</td>
<td>Simpler geometry</td>
<td>Complicated geometry</td>
<td>Ultra-high energy</td>
</tr>
<tr>
<td></td>
<td>Lower energy</td>
<td>Higher energy</td>
<td>???</td>
</tr>
</tbody>
</table>
An issue for multi-threading

- One of advantages of multi-thread / multi-core is efficient memory consumption.
  - MPI approach basically requires full copy of memory space for each slave.
  - For example, large x-section tables and complicated geometry could be shared by threads.
- X-section table in Geant4 has a caching mechanism.
  - Once a track accesses to the x-section for a certain particle in a certain material at a certain energy, the next access is likely for the same particle in the same material and at the nearby energy.
  - This caching mechanism never works if a table is shared by threads.
- Geant4 geometry is “dynamic”.
  - To reduce the memory size required for complicated geometry, Geant4 has a concept of “parameterized volume”. A volume returns its position, rotation, material, shape, size, etc. as a function of the “copy number”. And the copy number and some of these attributes are cached.
  - This caching mechanism never works if geometry is shared by threads.
Geant4 approach

• Making cashes thread-local using TOP-C shared memory model.
• TLS (Thread-local-storage)
  – static/global variables to thread-local with “__thread” (gcc)
  – Automatic TLS conversions with patched C++ parser.
• For non-thread-safe variables
  – Lock with mutex (mutual exclusion) : potential performance bottle neck
• Development of semi-automatic conversion tool.
• The first Geant4MT prototype release (based on Geant4 version 9.4-p01) is ready for your trial.
  – Note: This is still a prototype and we may change interfaces in the future releases.
Geant4 Modification to Share Data

Worker threads share read-only fields for some object instances.
Medium/longer term developments

- **Year 2011**
  - More automated conversion of Geant4 → Geant4MT
  - More thread-safety check of STL and CLHEP
  - Expecting (caching mechanism of) x-section tables and geometry to be fully multi-threaded
    - By splitting class data members so that R/W data members to be thread local
- **Year 2012-2013 (-2014?)**
  - Major architectural revision
    - Moving “dynamic” components of x-section tables and geometry to track object
  - Planning all the x-section tables and geometry to be fully shared by threads
General purpose GPU?

- Though the new Fermi Architecture supports C++, it supports only for the data processing. It does not yet support object instantiation/deletion in GPU.
  - It may/will support at PTX 2.0.
  - But what do we do for secondary tracks?
- Size of L1 cach (16/48 KB) and L2 cache (768 KB) are too small. Accessing to the main memory is too costly (>>100 Cycles).
  - Calculating x-section for every step is faster than accessing to x-section tables. How do we do for date-driven tables?
  - Sharing complicated geometry in main memory does not offer any benefit. Could our user live with just replicated boxes?
- GPU seems not to be feasible, at least for the near future, to a particle-transport type Monte Carlo simulation like Geant4.
  - “Density- or probability-transport” calculation with simplest geometry may fit to GPGPU.
  - We will watch Kepler and Maxwell Architectures
Tips for computing performance
Some tips to consider - 1

- We are making our best effort to improve the speed of Geant4 toolkit. But, since it is a toolkit, a user may also make the simulation unnecessarily slow.

- For general applications
  - Check methods which are invoked frequently, e.g. UserSteppingAction(), ProcessHits(), ComputeTransformation(), GetField() etc.
  - In such methods, avoid string manipulation, file access or cout, unnecessary object instantiation or deletion, or unnecessary massive polynomial calculation such as sin(), cos(), log(), exp().

- For relatively complex geometry or high energy applications
  - Kill unnecessary secondary particles as soon as possible.
  - Use stacking action wisely. Abort unnecessary events at the earliest stage.
  - Utilize G4Region for regional cut-offs, user limits.
  - For geometry, consider replica rather than parameterized volume as much as possible. Also consider nested parameterization.
  - Do not keep too many trajectories.

- For relatively simple geometry or low energy applications
  - Do not store the random number engine status for each event.
Some tips to consider - 2

- Chop out unnecessary objects in memory. This is not only the issue of memory size of your CPU, but also the matter of cache-hit rate.
  - By default cross-section tables of EM processes are built for the energy range of 0.1 keV to 10 TeV. If your simulation does not require higher energies, cut higher part out.
    - Do not change the granularity of sampling bins (7 bins per decade).
  - Delete unnecessary materials.
  - Limit size (number of bins) of scoring meshes.

- If you believe your simulation is unnecessarily slow, your application may have:
  - Memory leak
  - Geometry overlap