Geant4: improving computing performance by removing redundant G4Log calls

Mihály Novák
CERN (EP-SFT)

Contents

Outline

`Geant4`: improving computing performance by removing redundant `G4Log` calls

Everything started with the `G4EmElementSelector`[1]:

- for EM models, the `G4VEmModel` base class provides the possibility to (automatically)build a collection of `G4EmElementSelector`-s for each material cuts couple
- this collection can be used at run-time, to sample the target atom for the given interaction (in case of multi-element materials)
- each individual (i.e. for a given *model* given *material* cuts couple) `G4EmElementSelector` of the collection stores a table of discrete probabilities of having the given interaction on a given element of the material (i.e. $P(Z_i) = \Sigma_{Z_i}/\Sigma$) over a discrete energy grid: equally spaced in log energy scale
- the implementation of the table is a vector of `G4PhysicsLogVector` (as many as elements in the given material)
- at run-time, the target atom is sampled according to this discrete probability distribution: the probabilities are interpolated for the given primary energy
- however, **the energy bin index was re-computed for each possible target element during the interpolation: because selectors are individual log-vectors** (was fixed in `10.05`)

[1] see my presentation at the last collaboration meeting in Lund (Section 3. from slide #37 here)

The issue was however more general:

- the above `G4EmElementSelector` problem, i.e. extensive use of the `G4Log`, was clearly shown by the profiling done as part of the new `G4SeltzerBergerModel` development[1]
- however, the origin of the issue was in the underlying `G4PhysicsLogVector`
- `G4PhysicsLogVector` is heavily used in `Geant4` to store kinetic energy dependent data ($\mathrm{d}E/\mathrm{d}x$, R, Σ_t, Σ_{Z_i}, σ, etc.) in a discretised form and to obtain interpolated values at run-time
- at the interpolation, the logarithm of the actual kinetic energy value i.e. `G4Log(E_k)` needs to be known (for the computation of the abscissa bin index `i` such that $E_i \leq E_k < E_{i+1}$)
- the bin index is very often computed, i.e. `G4Log(E_k)` is evaluated, in spite of the fact that the last index used `ilast` is (often) cached and checked before the bin index computation
- on the same time, these `G4PhysicsLogVector`s are accessed at each simulation step to provide values needed to determine the actual physics step length: **individual table accesses with the same value of `E_k` results in redundant `G4Log(E_k)` computations**
- this is the main source of the significant run-time cost of the `G4PhysicsVector::Value( G4double E_k, size_t& ilast)` method that could have been observed for a long time

[1] see my presentation at the last collaboration meeting in Lund (Section 2. from slide #30 here)

Outline

`Geant4`: improving computing performance by removing redundant `G4Log` calls

2. **Solution**

To avoid redundant computations of `G4Log(E_k)`, the proposed [1] solution was applied:

- to (1) store the logarithm of the kinetic energy whenever is already computed:

```
inline void G4DynamicParticle::SetKineticEnergy(G4double aEnergy)
{
  isLogEkinUpToDate = ( isLogEkinUpToDate && (theKineticEnergy == aEnergy) );
  theKineticEnergy = aEnergy;
}
```

```
inline G4double G4DynamicParticle::GetLogKineticEnergy() const
{
  if (!isLogEkinUpToDate) {
    theLogKineticEnergy = (theKineticEnergy > 0.) ?
                          G4Log(theKineticEnergy) : LOG_EKIN_MIN;
    isLogEkinUpToDate   = true;
  }
  return theLogKineticEnergy;
}
```

- and (2) propagate to the interpolation method i.e. introduce a new `G4PhysicsVector::Value( G4double E_k, G4double LogE_k, size_t& ilast)` method

```
G4double G4PhysicsVector::Value(G4double theEnergy, G4double theLogEnergy,
                                size_t& lastIdx) const
```

[1] see my presentation at the last collaboration meeting in Lund (Section 3. slide #40 here)

Additional information:

- about 40 files have been modified over 5 categories to introduce and to make use of the new functionality
- the simulation results are not affected by the changes: results stay **numerically identical** before and after merging the corresponding changes
- the profiling results shown in the following slides were obtained by:
  - using `valgrind` with the `--tool=callgrind`
  - instrumentation only in the event-loop: `CALLGRIND_START_INSTRUMENTATION` and `CALLGRIND_STOP_INSTRUMENTATION` at the end/beginning of the `Begin-/EndOf RunAction` with the `--instr-atstart=no` (i.e. pure run-time contributions are measured)
  - `Geant4-10.05-ref02` refers to the `-ref02` master **before** while `Geant4-10.05-ref03` to the master **after** merging the corresponding changes
  - both versions of `Geant4` were build and used in `DEBUG` mode during the profiling to see all `inlined` methods (i.e. look only the call counters and do not give significance neither to the inclusive nor to the exclusive performance contributions)

Outline

`Geant4`: improving computing performance by removing redundant `G4Log` calls

3. **Results**

`Geant4`: improving computing performance by removing redundant `G4Log` calls

3 Results

Electromagnetic component

Electromagnetic component

10 [GeV] e^- in Simplified Sampling Calorimeter: 50 layers of 2.5 [mm] Pb and 5.7 [mm] liquid-Ar

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.48	0.67	40 680 456	G4PhysicsVector::Value(double, unsigned long&) const

[1] see my presentation at the last collaboration meeting in Lund (Section 3. slide #40 here)

Electromagnetic component

10 [GeV] e^- in Simplified Sampling Calorimeter: 50 layers of 2.5 [mm] Pb and 5.7 [mm] liquid-Ar

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.48	0.67	40 680 456	G4PhysicsVector::Value(double, unsigned long&) const

`Geant4-10.05-ref03`: with the new `Value(E_k, LogE_k, ilast)` interface method

Incl.	Self	Called	Function
5.01	0.52	30 353 168	G4PhysicsVector::Value(double, double, unsigned long&) ...
2.45	0.18	10 327 288	G4PhysicsVector::Value(double, unsigned long&) const

[1] see my presentation at the last collaboration meeting in Lund (Section 3. slide #40 here)

Electromagnetic component

10 [GeV] e^- in Simplified Sampling Calorimeter: 50 layers of 2.5 [mm] Pb and 5.7 [mm] liquid-Ar

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.48	0.67	40 680 456	G4PhysicsVector::Value(double, unsigned long&) const

~25 % goes through the old while **~75 % goes through the new interface**

`Geant4-10.05-ref03`: with the new `Value(E_k, LogE_k, ilast)` interface method

Incl.	Self	Called	Function
5.01	0.52	30 353 168	G4PhysicsVector::Value(double, double, unsigned long&) ...
2.45	0.18	10 327 288	G4PhysicsVector::Value(double, unsigned long&) const

[1] see my presentation at the last collaboration meeting in Lund (Section 3. slide #40 here)

10 [GeV] e^- in Simplified Sampling Calorimeter: 50 layers of 2.5 [mm] Pb and 5.7 [mm] liquid-Ar

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.48	0.67	40 680 456	G4PhysicsVector::Value(double, unsigned long&) const

Incl.	Self	Called	Function
6.57	1.95	57 714 780	G4Log(double)

`Geant4-10.05-ref03`: **~24 % reduction in run-time `G4Log`-calls for EM shower**[1]

Incl.	Self	Called	Function
5.01	0.52	30 353 168	G4PhysicsVector::Value(double, double, unsigned long&) ...
2.45	0.18	10 327 288	G4PhysicsVector::Value(double, unsigned long&) const

Incl.	Self	Called	Function
4.96	1.47	44 014 751	G4Log(double)

[1] see my presentation at the last collaboration meeting in Lund (Section 3. slide #40 here)

`Geant4`: improving computing performance by removing redundant `G4Log` calls

3 Results

Including hadronic component

Including hadronic component

CMS simulation: 100 random (particle, direction, energy $\in$ [1GeV $-$ 100GeV]) events without field

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.97	0.68	320 702 629	G4PhysicsVector::Value(double, unsigned long&) const
7.35	0.25	222 889 366	G4PhysicsVector::Value(double) const

An additional `Value(G4double E_k)` interface method:

- that eventually calls the earlier `Value(G4double E_k, size_t& ilast)` interface method
- used mainly (∼97%) by neutron XS (elastic, inelastic, xapture)
- do not benefit from cached last bin index used → `G4Log` is always recomputed !

```
inline
G4double G4PhysicsVector::Value(G4double theEnergy) const
{
  size_t idx=0;
  return Value(theEnergy, idx);
}
```

Including hadronic component

CMS simulation: 100 random (particle, direction, energy $\in$ [1GeV $-$ 100GeV]) events without field

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.97	0.68	320 702 629	G4PhysicsVector::Value(double, unsigned long&) const
7.35	0.25	222 889 366	G4PhysicsVector::Value(double) const

`Geant4-10.05-ref03`: with the new `Value(E_k, LogE_k, ilast)` interface method

Incl.	Self	Called	Function
5.42	0.69	304 046 692	G4PhysicsVector::Value(double, double, unsigned long...
0.69	0.04	16 655 937	G4PhysicsVector::Value(double, unsigned long&) const
0.29	0.00	4 275 138	G4PhysicsVector::Value(double) const

Including hadronic component

CMS simulation: 100 random (particle, direction, energy $\in$ [1GeV $-$ 100GeV]) events without field

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.97	0.68	320 702 629	G4PhysicsVector::Value(double, unsigned long&) const
7.35	0.25	222 889 366	G4PhysicsVector::Value(double) const

$\sim$98 % reduction in `Value(G4double E_k)` calls

$\sim$5 % goes through the old while **$\sim$95 % goes through the new interface**

`Geant4-10.05-ref03`: with the new `Value(E_k, LogE_k, ilast)` interface method

Incl.	Self	Called	Function
5.42	0.69	304 046 692	G4PhysicsVector::Value(double, double, unsigned long...
0.69	0.04	16 655 937	G4PhysicsVector::Value(double, unsigned long&) const
0.29	0.00	4 275 138	G4PhysicsVector::Value(double) const

Including hadronic component

CMS simulation: 100 random (particle, direction, energy $\in$ [1GeV $-$ 100GeV]) events without field

`Geant4-10.05-ref02`:

Incl.	Self	Called	Function
9.97	0.68	320 702 629	G4PhysicsVector::Value(double, unsigned long&) const
7.35	0.25	222 889 366	G4PhysicsVector::Value(double) const

Incl.	Self	Called	Function
4.93	1.46	322 222 206	G4Log(double)

`Geant4-10.05-ref03`: **$\sim$62 % reduction in run-time `G4Log`-calls !!!**

Incl.	Self	Called	Function
5.42	0.69	304 046 692	G4PhysicsVector::Value(double, double, unsigned long...
0.69	0.04	16 655 937	G4PhysicsVector::Value(double, unsigned long&) const
0.29	0.00	4 275 138	G4PhysicsVector::Value(double) const

Incl.	Self	Called	Function
1.94	0.58	124 351 365	G4Log(double)

Including hadronic component

CMS simulation: 100 random (particle, direction, energy $\in$ [1GeV $-$ 100GeV]) events without field

These give a run-time speed-up of:

- $\sim$**7.6-8** % on my **MacBook Pro** (MacOS 10.13.2), 2.8 GHz Intel Core i7 processor with 16 GB 1600 MHz DDR3 memory using Apple LLVM version 10.0.0 (clang-1000.10.44.4) (**all local**)
- $\sim$**3.8-4** % on my **AMD Desktop** (SLC 6.10 Carbon), 3.5 GHz AMD PRO A10-9700 R7 processor with 8 GB DDR4-2400 SDRAM memory using gcc 8.2.0 sourced from /cvmfs/sft.cern.ch/lcg/... (i.e. **not all local**)
- the run time of the same simulation is $\sim 2\times$ more on the AMD desktop compared to the (all local) laptop: more time spent on communication than floating point operations $\rightarrow$ less visible any floating point operation improvements
- additional performance measurements were done by Gabriele (uniform field turned ON) giving $\sim$**11** % on his laptop **MacBook Pro** (MacOS 10.14.3), 2.8 GHz Intel Core i7 processor with 16 GB 1600 MHz DDR3 memory using Apple LLVM version 10.0.0 (clang-1000.11.45.5) (**all local**)

Outline

`Geant4`: improving computing performance by removing redundant `G4Log` calls

4. **Conclusion**

A significant number of redundant `G4Log` calls have been eliminated in `Geant4-10.05-ref03` :

- based on the results of profiling and detailed understanding of the toolkit
- by introducing functionality to store and re-use the already computed `G4Log(E_k)` value
- $\sim$**24-62** % of the run time `G4Log` calls is eliminated by making use of the new functionality in the EM and Hadronic (neutornXS) physics
- it gives a measurable run-time improvement of $\sim$**2-8** % depending on the configuration
- the measured run-time effect strongly depends on the architecture
- the current way of performance monitoring might be reconsidered and improved in order to provide more stable and more sensitive results