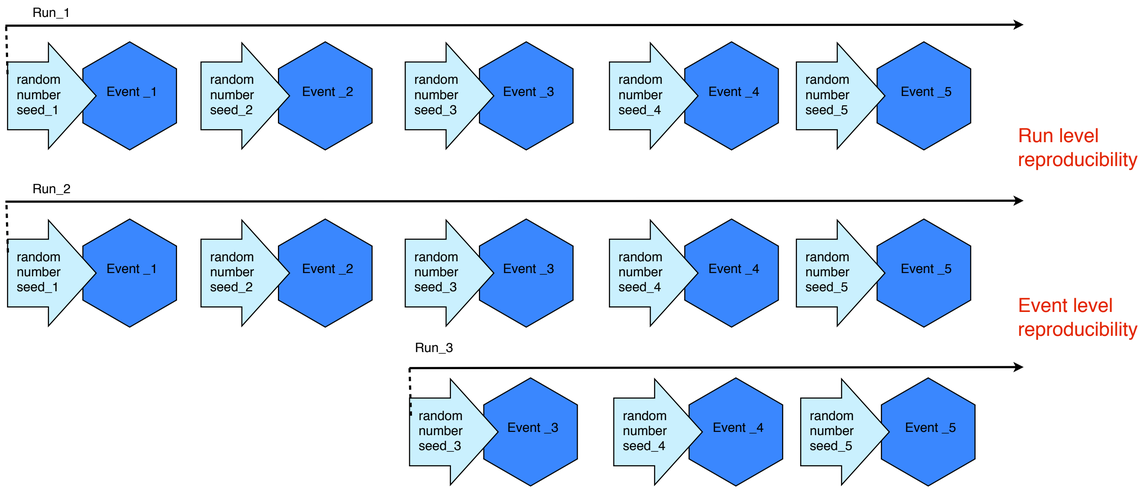
Recent Developments in the Geant4 Hadronic Framework

Witold Pokorski (CERN), Alberto Ribon (CERN)

Geant4 is the main simulation toolkit used by the LHC experiments and therefore a lot of effort is put into improving the physics models in order for them to have more predictive power. As a consequence, the code complexity increases, which requires constant improvements and optimizations on the programming side. In this poster, we discuss the recent developments and improvements in the hadronic framework of the Geant4 simulation toolkit.

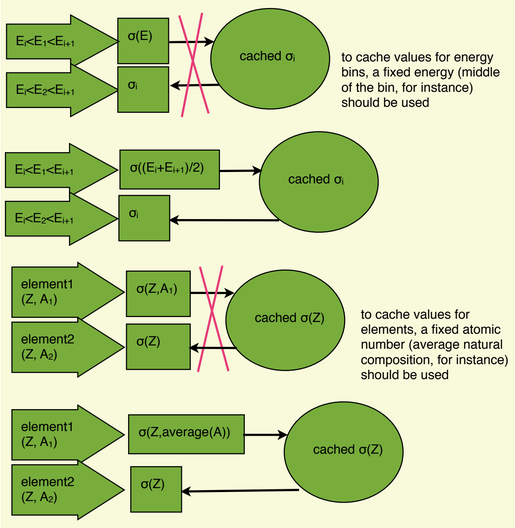
Reproducibility of simulated events

Using pseudo-random number generator implies that events should be reproducible. Non-reproducibility of events makes it difficult to debug the code. Simulation results should be reproducible not only at the level of the run (starting from the same random-number generator seed), but also at the level of each event (starting from any event within a run).



Sources of irreproducibility:

- bugs (uninitialized variables)
- history-dependent approximations
- incorrect caching



Non-reproducibility fixes for G4 9.6

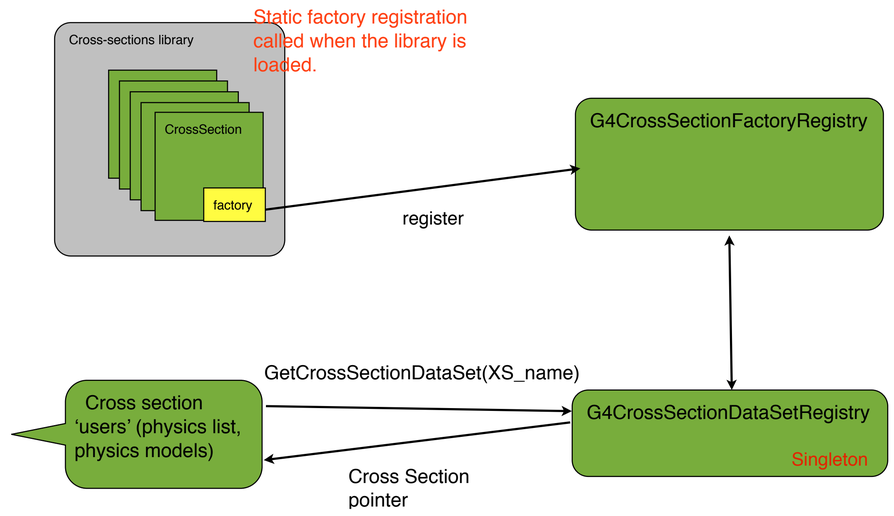
Up to now eleven non-reproducibility fixes have been made.

Now Geant4 events are reproducible (with the exception of neutron HP).

- Chips quasi-elastic
- Starkov elastic final state model for $\pi^{\pm} > 1$ GeV
- Ion ionization corrections
- Fission in Bertini intra-nuclear cascade
- Bertini intra-nuclear cascade, when hyperons are involved (it turned out a problem in G4PhaseSpaceDecayChannel)
- (Decoupled) Chips hadron-nucleon inelastic cross sections, used by FTFP (2 different problems)
- Multiple scattering (3 different problems)
- Binary intra-nuclear cascade

Factory mechanism for instantiation of cross sections

To share cross-section objects between different ‘users’ (physics processes, models, physics lists, etc) we have introduced the factory pattern for the instantiation of the objects and we have extended the functionality of G4CrossSectionDataSetRegistry to store and to provide the pointer to those objects.



➡ ‘Cross-section user’ asks G4CrossSectionsDataSetRegistry for a given G4CrossSectionDataSet by specifying its name (string).

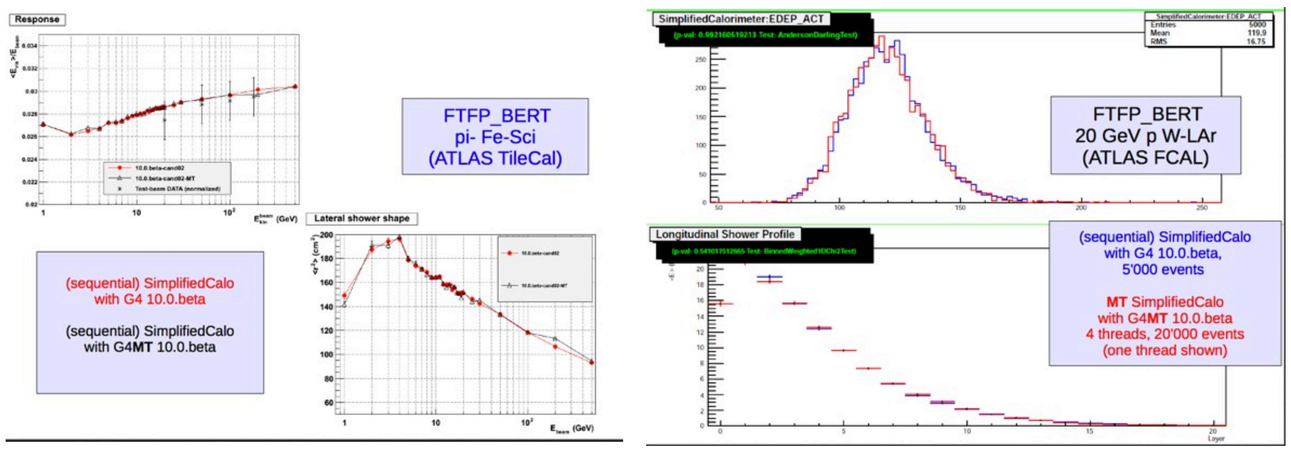
➡ The registry checks if this cross-section has been already instantiated.

➡ If yes, it returns the pointer to it (shared between all ‘cross-section users’).

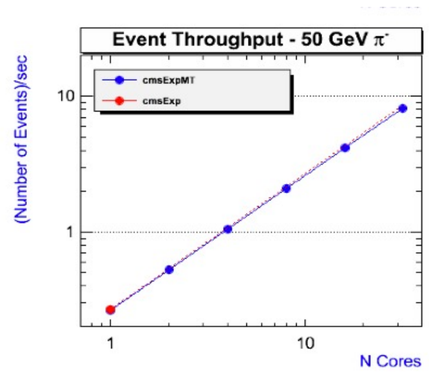
➡ If not, the registry uses the factory to instantiate the given cross-section. If the factory does not exist, it return an error ‘cross-section not found’.

Multi-Threading in Geant4 hadronic physics

The Geant4 code is undergoing a major development in order to run in the multi-threading mode. A number of technical issues needs to be addressed to eliminate any interference between several threads accessing the hadronic physics classes. Objects that can be easily shared are those that are read only.
Caching becomes tricky because of possible simultaneous write access to cache.
In order to validate the multi-threaded code we require that the calorimeter (and other) observables remain statically the same between sequential and multi-threaded modes.



Multi-threading increases significantly the event throughput, however the challenge is the reproducibility of events. After a number of fixes and improvements, full reproducibility has been achieved.



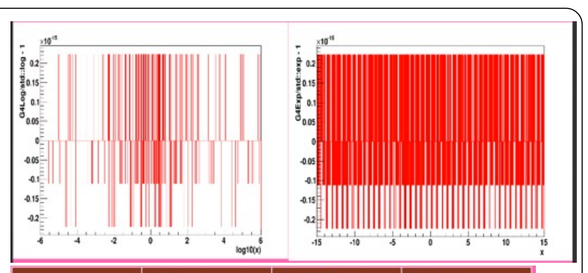
Results for FTFP_BERT physics list:

- ~9% reproducibility violations in G4 10.0.beta
  - Found a problem (cached value) in G4MuPairProductionModel
- ~0.1% reproducibility violations in G4 9.6.ref07
  - Found thread-collision problem (cache shared among threads) in Bertini
- ~7% reproducibility violations in G4 9.6.ref08
  - Found problems in G4MuPairProductionModel
- 0% reproducibility violations in G4 9.6.ref08 + fixes in muon physics

=> Full reproducibility (sequential & MT) achieved

Use of fast mathematical functions

Precision of hadronic cross sections is at the level of 5-10% and therefore there is no need to do high-precision calculations involving those cross-sections. Using fast log and exp functions can increase significantly the CPU performance without any significant lost in the precision of the simulation results. We have replaced the std::log and std::exp by faster implementation extracted from VDT library (see Danilo Piparo talk at CHEP 2013). The effect is much below the precision of the cross-sections, while the calculation of the cross-sections values was faster by ~5%.



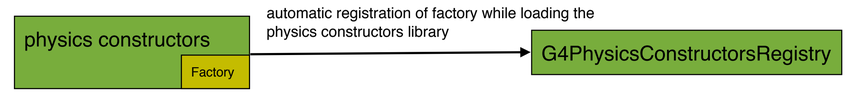
	Std	G4 VDT	G4Pow
Log	8.97	4.91	5.19
Exp	13.93	1.95	1.34
$A^{1/3}$	20.46	7.03	0.77
$Z^{1/3}$	-	-	0.01

Generic Physics List

The Generic Physics list class allows to remove the compile- and link-time dependency between the users code and the specific physics models. In order to achieve this we have:

➡ introduced registry of physics “constructors”

➡ instrumented physics “constructors” to provide factories that get registered in the registry

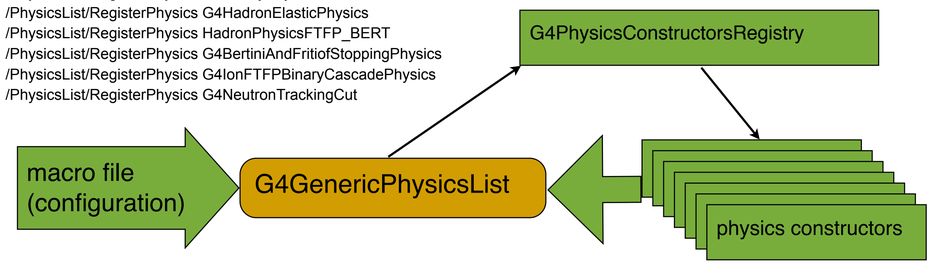


Physics Lists can now be constructed in two new ways:

➡ through G4GenericPhysicsList class using a macro file:

```
FTFP_BERT.mac:
/PhysicsList/defaultCutValue  0.7
/PhysicsList/SetVerboseLevel 1

/PhysicsList/RegisterPhysics G4EmStandardPhysics
/PhysicsList/RegisterPhysics G4EmExtraBertiniPhysics
/PhysicsList/RegisterPhysics G4DecayPhysics
/PhysicsList/RegisterPhysics G4HadronElasticPhysics
/PhysicsList/RegisterPhysics HadronPhysicsFTFP_BERT
/PhysicsList/RegisterPhysics G4BertiniAndFritiofStoppingPhysics
/PhysicsList/RegisterPhysics G4IonFTFPBinaryCascadePhysics
/PhysicsList/RegisterPhysics G4NeutronTrackingCut
```



and the main() containing:

```
phys = new GenericPhysicsList();
G4UImanager* UImanager = G4UImanager::GetUIpointer();

UImanager->ApplyCommand("/control/execute
FTFP_BERT.mac");
```

➡ by passing a vector of physics ‘constructors’ names at the instantiation time

- create

  vector<string> MyPhysicsConstructors

  containing names of the physics constructor
- instantiate generic physics list

  new GenericPhysicsList(MyPhysicsConstructors)

  where this constructor iterates over the physics constructors names and registers them