G4HepEm: a Geant4 EM physics R&D

Jonas Hahnfeld, Benjamin Morgan, Mihály Novák

Outline

Contents

G4HepEm: **motivations** in a nutshell

- initiated by the **Geant4 EM physics working group** as part of looking for solutions **to reduce the computing performance bottleneck** experienced by the **HEP detector simulation** applications
- **targeting** the most performance critical part of the HEP detector simulation applications, i.e. the **EM shower generation** covering(initially) e^-/e^+ and γ particle transport
- the main goal is to investigate the **possible computing performance benefits of**
  - **providing alternative, highly specialised** (for particle types, e^-/e^+, γ and HEP applications) **optional stepping loops** beyond the current general one
    $\Longrightarrow$ **giving up the** *"unutilised"* **flexibility with the hope of some performance gain**
  - having a very **compact and efficient implementation of** all the related **run time functionalities** required for an EM shower simulation
    $\Longrightarrow$ **compact run time library** and **data layout with the hope of some performance gain**
- the main design principles
  - separation of initialisation- and run-time functionalities $\Longrightarrow$ in order to have a compact run-time library
  - separation of data and functionality $\Longleftarrow$ since data are filled at initialisation- while used at run-time
- resulted in a run-time **EM shower simulation library with** many attractive characteristics such as the device(**GPU**) side **support** of all related computations (utilised in **AdePT**) or its **stateless** property that, together with its simplicity, provides **an excellent domain to check many further interesting ideas**
- see the **initial presentation** or the one at the last **Geant4 technical forum** on G4HepEm for more details

Contents

G4HepEm: library **structure** is determined by the main goals and design

- **clear separation of run-time and initialisation-time functionalities**:
  - many information are needed at initialisation time but only a small fraction of that is used at run time

`G4HepEm`: library **structure** is determined by the main goals and design

- **clear separation of run-time and initialisation-time functionalities**:
  - many information are needed at initialisation time but only a small fraction of that is used at run time

Example: material description

- `Geant4` provides a **very rich and sophisticated material description library** with all the extended properties and built in material data bases needed for a wide range of simulations
- but **most of these** functionalities and data **are** actually **used for the** (user) **material definition and at initialisation time computations** (e.g. computation of density correction in the stopping power)
- **only a couple of these** material properties **are used at run-time** during the EM shower generation
- therefore, **a very simple data structure** (with couple of double/integer fields) **is perfectly sufficient** to keep all material related information needed at run-time
- as a consequence, there is no any run-time dependence on the `Geant4` material description library
- the same is true for the element, material-cuts couple data and many more complex data

G4HepEm: library **structure** is determined by the main goals and design

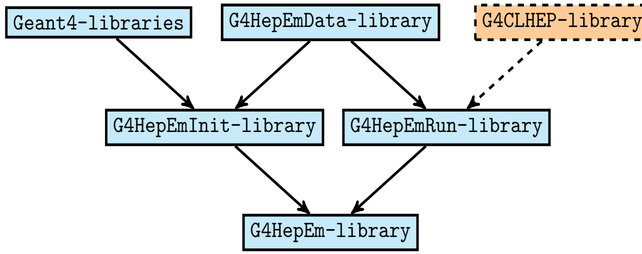
- **clear separation of run-time and initialisation-time functionalities**:
  - many information are needed at initialisation time but only a small fraction of that is used at run time
  - in order to obtain **as compact run-time library as possible**

G4HepEm: library **structure** is determined by the main goals and design

- **clear separation of run-time and initialisation-time functionalities**:
  - many information are needed at initialisation time but only a small fraction of that is used at run time
  - in order to obtain **as compact run-time library as possible**
- results in **separation of** the **data definitions and functionalities** (i.e. very often more C-style than C++): isolated, *"single function"* implementation of the **run-time functionalities**, acting on and according to their input arguments (mostly primitive types with some data structures)
- all these above have lots of benefits (see some soon)

G4HepEm: library **structure** is determined by the main goals and design

- **clear separation of run-time and initialisation-time functionalities**:
  - many information are needed at initialisatio
  - in order to obtain **as compact run-time libr**
- results in **separation of** the **data definitions** C++): isolated, *"single function"* implementa according to their input arguments (mostly pr
- all these above have lots of benefits (see som
- G4HepEm is structured along this separation:
  - G4HepEmData: definition of all data structures **filled at initialisation** and **used at run-time**
  - G4HepEmInit: all **initialisation time functionalities**, e.g. for **constructing** and **populating** the above **data structures** (based on the given application setup) **relying heavily on core** Geant4 **functionalities**
  - G4HepEmRun: all **run-time functionalities**, e.g. for **reading**/(interpolating) **the data structures** constructed and populated at the initialisation time, **compute the step lengths** and **perform the physics interactions**
  - G4HepEm: a tiny library for connecting all the above (a G4VProcess interface implementation as well)



Contents

Contents

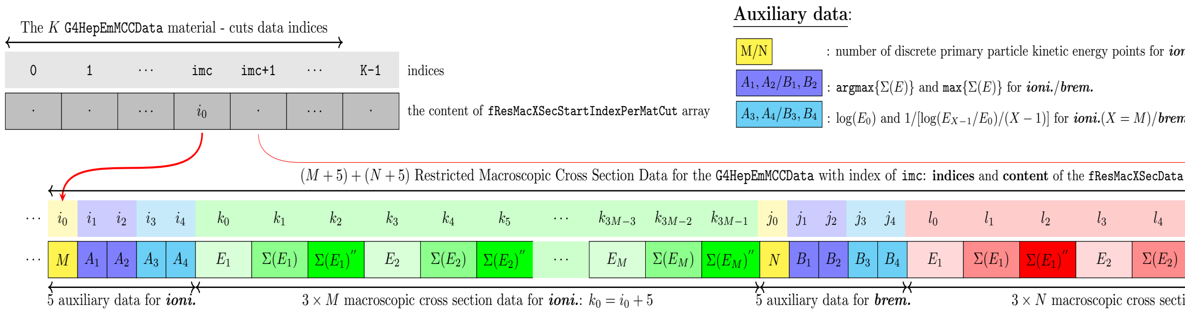
Compact data structures:

- **data structures** (defined in `G4HepEmData`, filled in `G4HepEmInit` and utilised in `G4HepEmRun`), were **designed driven by their run-time usage**
- it means that **their memory layouts** are **determined by their access patterns**

Com

Example: restricted macroscopic cross sections for e^-/e^+ ionisation and bremsstrahlung.

- **depends on the** material and secondary production threshold, i.e. on the **material-cuts** couple
- **computed**/stored **at initialisation over a discrete energy grid** (unique: for the couple and interaction)
- **used at run-time**: the inverse MFP between hard(/discrete) ionisation/bremsstrahlung events
- the **run-time** (spline) **interpolation** for a given E_{kin} is based on 6 discrete values: $E_i \leq E_{\text{kin}} < E_{i+1}$, $\Sigma(E_i), \Sigma(E_{i+1}), \Sigma(E_i)'', \Sigma(E_{i+1})''$
- evaluated for both interactions during the simulation step in the given material-cuts couple
- **all** the required **values are as close as possible in the memory**


The K G4HepEmMCCData material - cuts data indices
0
1
· · ·
imc
imc+1
· · ·
K-1
indices
·
·
· · ·
i_0
·
· · ·
·
the content of fResMacXSecStartIndexPerMatCut array
Auxiliary data:
M/N : number of discrete primary particle kinetic energy points for ion
$A_1, A_2/B_1, B_2$: argmax$\{\Sigma(E)\}$ and max$\{\Sigma(E)\}$ for ioni./brem.
$A_3, A_4/B_3, B_4$: $\log(E_0)$ and $1/[\log(E_{X-1}/E_0)/(X-1)]$ for ioni.$(X = M)$/brem
$(M+5)+(N+5)$ Restricted Macroscopic Cross Section Data for the G4HepEmMCCData with index of imc: indices and content of the fResMacXSecData
· · · i_0 i_1 i_2 i_3 i_4 k_0 k_1 k_2 k_3 k_4 k_5 · · · k_{3M-3} k_{3M-2} k_{3M-1} j_0 j_1 j_2 j_3 j_4 l_0 l_1 l_2 l_3 l_4
· · · M A_1 A_2 A_3 A_4 E_1 $\Sigma(E_1)$ $\Sigma(E_1)''$ E_2 $\Sigma(E_2)$ $\Sigma(E_2)''$ · · · E_M $\Sigma(E_M)$ $\Sigma(E_M)''$ N B_1 B_2 B_3 B_4 E_1 $\Sigma(E_1)$ $\Sigma(E_1)''$ E_2 $\Sigma(E_2)$
5 auxiliary data for ioni.
$3 \times M$ macroscopic cross section data for ioni.: $k_0 = i_0 + 5$
5 auxiliary data for brem.
$3 \times N$ macroscopic cross secti

Compact data structures:

- **data structures** (defined in `G4HepEmData`, filled in `G4HepEmInit` and utilised in `G4HepEmRun`), were **designed driven by their run-time usage**
- it means that **their memory layouts** are **determined by their access patterns**
- the same is true for all energy loss related data (i.e. restricted stopping power, range, inverse range) but also for the target element selectors, etc.
- the goal is to **enhance data locality** as much as possible, that might bring performance improvements:
  - especially when a complete simulation step within the compact `G4HepEmRun` library
  - even more if sub-sequent steps can be done
  - even more when all these can be done with more than one particles simultaneously (opportunistically see next slide)

Contents

Benefits of (opportunistic) multi particle computations:

- **separating run-time and initialisation time functionalities** results in a **very compact `G4HepEmRun` library** $\rightarrow$ might give performance improvements, especially when a complete simulation step can be performed within this library
- **separation of data definition** and **functionalities** $\rightarrow$ **self contained, *"single-function"* implementation of most of the `G4HepEmRun` functionalities** (e.g. all interactions, step-limit, etc.)
- it means that these functions do **not contain or interact with further objects** and **act on and according to their input arguments** (mostly primitive types with some data structures)
  $\Longrightarrow$ `G4HepEmRun` library **do not have any states**:
  - this gives the possibility in the future of e.g. popping-up more than one e^-/e^+ or γ tracks (from the internal secondary stacks) and tracking them together (opportunistically, e.g. when subsequent steps can be done within the *"current safety sphere"*)

Contents

Support of device side EM shower simulation:

- **separating run-time and initialisation time functionalities** → the `G4HepEmRun` **library** contains exactly what is needed at run-time for the EM shower simulation (since we wanted a compact run-time libarry)
- **separation of data definition** and **functionalities** → **self contained, *"single-function"* implementation of most of the `G4HepEmRun` functionalities** (e.g. all interactions, step-limit, etc.)
- it means that these functions do **not contain or interact with further objects** and **act on and according to their input arguments** (mostly primitive types with some data structures)

  $\Longrightarrow$ **implicit device side support**:
  - exactly *the same* run time *functions*, used on the host (i.e. CPU) side EM shower simulation, *can be invoked on the GPU* as device side functions

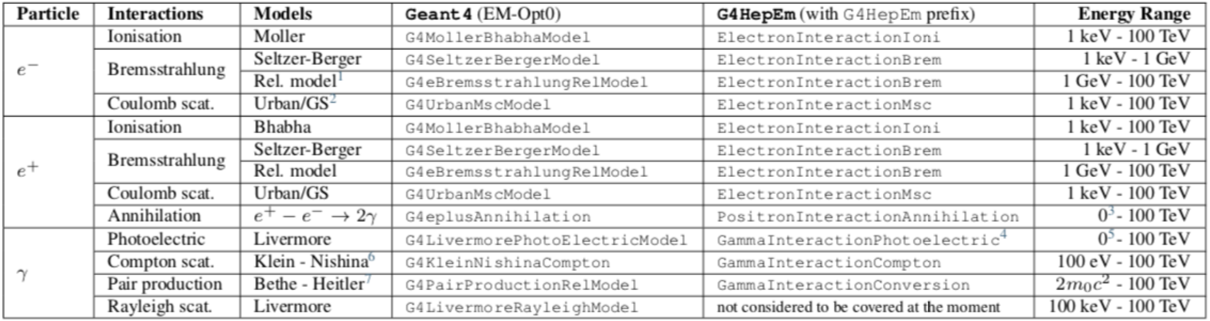
  $\Longrightarrow$ **explicit device side support**:
  - *all* the *data*, required at run-time by these functions, can be *transferred to the device* (by a single call to a function after the initialisation)
- most of the of the run-time `G4HepEm` EM shower simulation functionalities can be reused, as it is, on the device side:
  - accelerates significantly the related R&D activities targeting the same EM shower simulation on GPU (utilised in **AdePT**, see **Andrei's presentation last Thursday**)
  - no code duplications which is great for the maintenance and validation

Contents

Physics coverage: for the very first evaluation

- the core part of the physics, required for an EM shower simulation in HEP detectors, has just been implemented (see below with all details in the **documentation**)
- this is the essential set required for the very first performance evaluations
- the remaining parts (such as energy loss fluctuation or gamma- and lepto-nuclear interactions) will be handled in case this core part provides promising results

Table 1.1: Summary of the physics interactions and models used in "Geant4" and "G4HepEm" (current state).



Particle	**Interactions**	**Models**	**Geant4** (EM-Opt0)	**G4HepEm** (with G4HepEm prefix)	**Energy Range**
e^-	Ionisation	Moller	`G4MollerBhabhaModel`	`ElectronInteractionIoni`	1 keV - 100 TeV
	Bremsstrahlung	Seltzer-Berger	`G4SeltzerBergerModel`	`ElectronInteractionBrem`	1 keV - 1 GeV
		Rel. model[1]	`G4eBremsstrahlungRelModel`	`ElectronInteractionBrem`	1 GeV - 100 TeV
	Coulomb scat.	Urban/GS[2]	`G4UrbanMscModel`	`ElectronInteractionMsc`	1 keV - 100 TeV
e^+	Ionisation	Bhabha	`G4MollerBhabhaModel`	`ElectronInteractionIoni`	1 keV - 100 TeV
	Bremsstrahlung	Seltzer-Berger	`G4SeltzerBergerModel`	`ElectronInteractionBrem`	1 keV - 1 GeV
		Rel. model	`G4eBremsstrahlungRelModel`	`ElectronInteractionBrem`	1 GeV - 100 TeV
	Coulomb scat.	Urban/GS	`G4UrbanMscModel`	`ElectronInteractionMsc`	1 keV - 100 TeV
	Annihilation	$e^+ - e^- \to 2\gamma$	`G4eplusAnnihilation`	`PositronInteractionAnnihilation`	0[3] - 100 TeV
γ	Photoelectric	Livermore	`G4LivermorePhotoElectricModel`	`GammaInteractionPhotoelectric`[4]	0[5] - 100 TeV
	Compton scat.	Klein - Nishina[6]	`G4KleinNishinaCompton`	`GammaInteractionCompton`	100 eV - 100 TeV
	Pair production	Bethe - Heitler[7]	`G4PairProductionRelModel`	`GammaInteractionConversion`	$2m_0c^2$ - 100 TeV
	Rayleigh scat.	Livermore	`G4LivermoreRayleighModel`	not considered to be covered at the moment	100 keV - 100 TeV

Simplified sampling calorimeter: 50 layers of [2.3 mm $PbWO_4$ + 5.7 mm lAr]

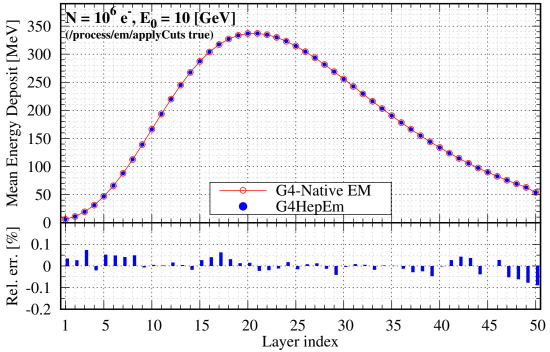


Table 1: Mean values per event of some selected quantities when modelling 10^6, $E_0 = 10$ [GeV] e^- in a simplified sampling calorimeter (50 layers of [2.3 mm $PbWO_4$ + 5.7 mm lAr]). Using `/process/em/applyCuts true`

		Geant4	G4HepEm	Rel. err. [%]
E_{dep} [MeV]	$PbWO_4$	6753.94	6746.7	-0.1066
	lAr	2545.71	2553.1	0.2907
#secondary	γ	4360.45	4359.59	-0.0197
	e^-	1744.83	1743.24	-0.0911
	e^+	429.39	429.30	-0.0209
#steps	charged	28151.3	28143.3	-0.0284
	neutral	40077.9	39935.0	-0.3565

`/process/em/applyCuts true`: mean number of secondary e^- is reduced by ~ 80 % (charged steps by ~ 25 %)!

Contents

Configurations and notations:

- G4HepEm build: dynamic and it is still not optimised, e.g. the standard mathematical functions are used instead of the optimised versions of GEANT4
- Geant4 build: master with static libraries; `-DGEANT4_BUILD_STORE_TRAJECTORY=OFF` and `-DGEANT4_BUILD_VERBOSE_CODE=OFF`; on `AMD Ryzen 9 3900`
- hardware: `12 core AMD Ryzen 9 3900`

- application `TestEm3`: 10 [GeV] e^- in a simplified sampling calorimeter with 50 layers of 2.3 [mm] $PbWO_4$ and 5.7 [mm] `liquid-Ar` using the default EM settings (e.g. `/process/em/applyCuts false`)
  $\Longrightarrow$ pure EM shower simulation
- application `cms2018`: CMS geometry with `gg2ttbar` events with physics settings similar to `CMSSW`, including hadronic physics, gamma-lepto-nuclear processes, a propagation in constant field and optimisations such as `/process/em/applyCuts true`
  $\Longrightarrow$ close to production settings with its realistic EM fraction

- **`results Physics List`**: using the usual physics list interface, i.e. the general stepping loop, either with **`G4NativeEm`** processes or **`G4HepEm`**
- **`results Specialised Tracking`**: using a specialised stepping/tracking loop implementation, i.e. NOT the general stepping loop, either with **`G4NativeEm`** processes or **`G4HepEm`**

Setup: TestEm3, 100k e$^-$, 10 GeV, 24 threads on `AMD Ryzen 9 3900` (**default EM settings**)

	`Physics List`	`Specialised Tracking`	difference
`G4NativeEm`	500 s	426 s	-14.8 %
`G4HepEm`	459 s	373 s	-18.7 %
difference	-8.2 %	-12.4 %	-25.4 %

Setup: `cms2018`, 1000x the same `gg2ttbar` event, 24 threads on `AMD Ryzen 9 3900` (**optimised EM**)

	`Physics List`	`Specialised Tracking`	difference
`G4NativeEm`	2889 s	2747 s	-4.9 %
`G4HepEm`	2847 s	2660 s	-6.6 %
difference	-1.5 %	-3.2 %	-7.9 %

Note: significant performance gain due to the specialised tracking of e^-/e^+ and γ even already using GEANT4 native processes that is boosted further with `G4HepEm` (even in its current, preliminary phase)

Using native GEANT4 processes with a single thread. The minimum time of three runs are reported.

setup	Physics List	Specialised Tracking	difference
TestEm3, 1000 e^-, 10 GeV	65.99 s	57.40 s	-13.02 %
plus magnetic field, $B_z = 1\,T$	78.45 s	71.59 s	-8.74 %
cms2018, one gg2ttbar event*	42.35 s	40.29 s	-4.85 %

Note: significant performance gain due to the specialised tracking of e^-/e^+ and γ even already using GEANT4 native processes while obtaining numerically identical results!

Using native Geant4 processes with a single thread. The minimum time of three runs are reported.

setup	`Physics List`	`Specialised Tracking`	difference
TestEm3, 1000 e^-, 10 GeV	65.99 s	57.40 s	-13.02 %
plus magnetic field, $B_z = 1\,T$	78.45 s	71.59 s	-8.74 %
`cms2018`, one `gg2ttbar` event*	42.35 s	40.29 s	-4.85 %

Note: significant performance gain due to the specialised tracking of e^-/e^+ and γ even already using Geant4 native processes while obtaining numerically identical results!

Both the specialised tracking/stepping and `G4HepEm` (already in its current state) provides significant performance improvements! How do we provide the possibility of implementing these specialised/external tracking? See the **discussion on the `G4VTrackingManager`** last Tuesday.

Contents

What's next?

- while these results are definitely promising, keep in mind that these are just the very first results right after the verification
  - some obvious but important optimisations still need to be done together with a clean-up
  - some of the current functionalities should reach their final form before moving further
- hopefully **the discussion**, started last Tuesday on how to provide the possibility to plug-in the specialised stepping loop and `G4HepEm` into a `Geant4` application, **will converge soon** and **the possibility will be provided by version 11.0.**
- these are the main items till the next point where:
  - we might show even more attractive performance gain
  - detailed performance analysis regarding their origin
  - some important decisions can be made regarding the future
  - a detailed list of the **further interesting ideas** (multi-particle tracking, general process like optimisations, etc.) and the **functionalities missing for production** (fluctuation, gamma-nuclear interactions, etc.)
- experiments already show some interest: we already had a meeting with our ATLAS colleagues who would be happy to provide us feedbacks and we will keep working with Vladimir to make sure that all the functionalities that are required by CMS and ATALS in production are provided
- still a long way to go, but rather encouraging results form the first evaluations