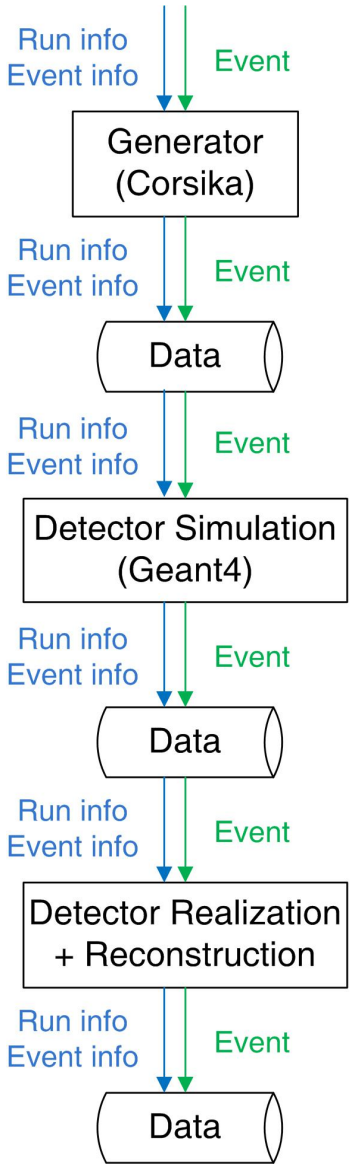
一种基于Hit流的模拟方法

姚志国
中科院高能所
2016/08/15

内容提要

- 主要目标
- 解决方法
- 实现细节
- 相关其它工具：
  - OptParser
- 总结

实现方法：原有方式



- Very popular:
  - The way as what the WCDA did;
  - Event stream: as *Event Data Model (EDM)* in collider experiments such as CMS.
- Old-fashion:
  - Memory consuming for large events;
  - Probably not very suitable for treatment of events piling-up.

目标之一：解决内存耗尽问题

- Generator:
  - Plenty of number of particles from a high energy air shower event.
- Hits collection:
  - huge number of hits for a high energy air shower event;
  - Even, hits of a high energy secondary particle (e.g., a hadron) occasionally falling into the detector can use up the memory.
- Many copies of the detector cells / units:
  - Detailed implementation of every cells may bring additional memory cost.

目标之二：优化中间结果的存储

- Generator:
  - Air shower data such as Corsika output are not necessary to be fully kept as many detectors are sparsely placed;
  - Positions of particles relative to a detector cell recorded may shrink the data size with the same precision especially when a compression algorithm is applied.
- More information on hits:
  - More information rather than only the hits themselves may do help for some particular analyses and the detector optimization;
  - The reading and storing of the information on the run and the event are not necessarily be taken care by the program developer in every step.

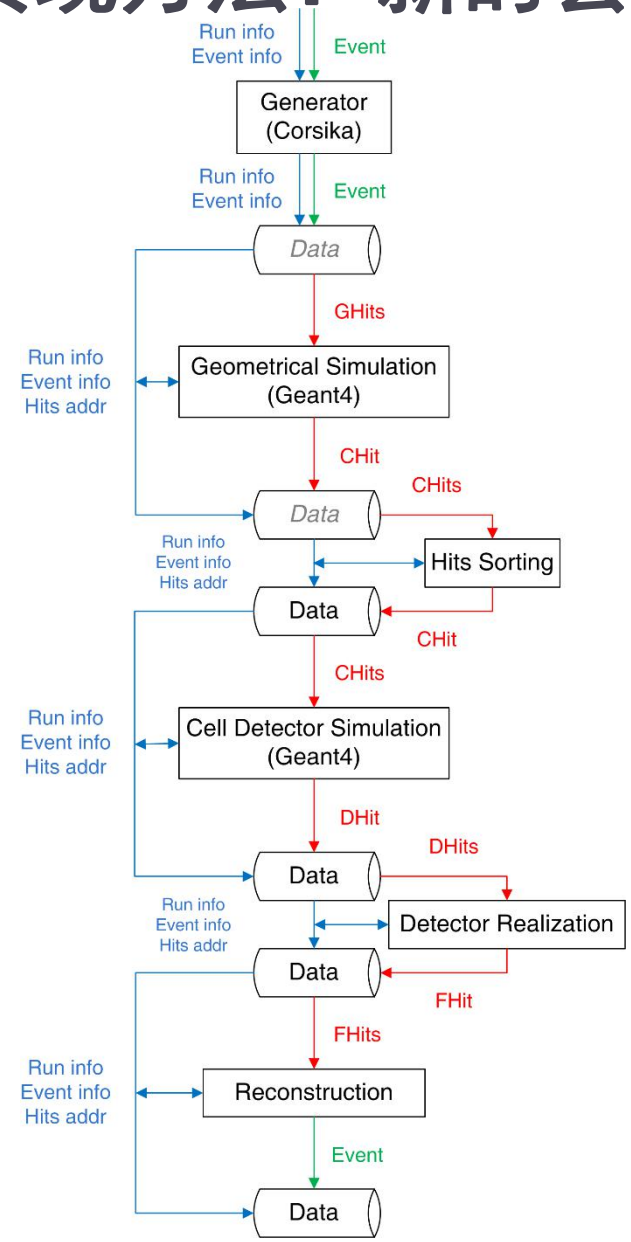
目标之三：易于探测器的真实化

- Maintenance of the code:
  - Split the simulation into several steps, leave the some mutable processes relying more on the realization in a later-on dedicated program;
- CPU time:
  - Put the solid implementation depending little on detector realization but time-consuming in the first step, so that saving the CPU time.

目标之四：简化各类探测器的统一模拟

- Skeleton simulation:
  - The detector configuration, such as the geometrical position and the skeleton structure can be simulated in the first step;
- Detailed simulation:
  - The detailed detector simulations can be carried out in the second step with only a few detector cells (e.g., a few detectors per detector kind) residing in a simple box.

实现方法：新的尝试



- Hit stream:
  - In: a batch of hits;
  - Out: a hit.
- Storage & buffering:
  - ROOT tree.
- 4 kinds of hits:
  - Ghit: generator hit;
  - Chit: cell hit;
  - Dhit: detector hit;
  - Fhit: final hit.

关于Hits

- Hits:
  1) Generator hits - Ghits (particles of a shower, e.g., from Corsika);
  2) Cell hits - Chits (particles hitting a detector unit / cell, obtained by a simple tracking or geant4 simulation without real sensitive detectors);
  3) Detector hits - Dhits (particles / lights / energy collected by the detector);
  4) Final hits - Fhits (signals converted from detector hits, added-on with noises).
- Hits stream:
  - These hits form a stream, flowing from the top to the bottom. Any two kind of consequential hits are connected by a “repeater”, comprising of storage, processors and switches. In the GEANT4 nomenclature, actually 1) and 2) are two steps of the particle generation; 3) and 4) are two steps of particle simulation.

关于Events

- Event fragmentizing:
  - There is no run and event concept, run / event start, run / event end are just special hits, merged into the hits stream, to mark the separation of these hits;
  - Hits are delivered in batches through the stream. The size of the batch can be controlled, but internally forced to be aligned in the event gaps, and detector gaps (once the detector makes sense).

关于实现方式

- Implementation:
  - Carrier:
    - Hits are represented by int / float variables in different c++ classes, whose carriers are by the means of std::vector<class name>;
  - Storage:
    - The storage in the repeater are root trees, which are well-designed with buffering and accessing model;
  - Processor:
    - The processors are Corsika / Geant4 / User codes;
  - Switch:
    - The switches are user codes, which can decide where to go by judging particle position and / or detector ID;
  - Interface:
    - Interface code: a class HitsReader. In addition, a program converting corsika output to root and a program sorting hits according to detector ID is implemented;
  - Indexing:
    - The entry positions of hits of all events are indexed and stored in a tree, enabling randomly accessing hits of any event.

应用程序片段

- 初始化：

```
hr = new HitsReader(inputfilename,outputfilename);
hr->SetBatchSize(kxIn,BatchSize);
hr->SetObservationLevel(ObservationLevel);
hr->SetHitTypes(kxHits,kxChit);
hr->SetIoStyles(11,0);
```

应用程序片段

- A batch of hits:

```
while true {
  int istat = hr->GetBatchOfHits(hlist);
  if (hlist.size()<=0) {
    G4RunManager::GetRunManager()->AbortRun();
    return;
  }

  if (istat/100%10%2==1) {
    ++irun;
    RunHeader = hr->GetRunh(kxOut);
  }

  if (istat/10%10%2==1) {
    ++ievent;
    EventHeader = hr->GetEvth(kxOut);
    PrepareCoreDistributions();
  }

  break;
}
```

Run & event header can be accessed, modified, updated, or added with some extra information such as the core position.

应用程序片段

- Primary generating：

```
for (unsigned int i=0; i<hlist.size(); ++i) {
  G4int idpart = hlist[i].idpart/1000;
  if ((idpart>=66&&idpart<=69)||
      (idpart>=133&&idpart<=134)) continue;

  G4String pname = hr->GetParticleNameG4(idpart);
  particledef = particleTable->FindParticle(pname);

  Position = G4Point3D(hlist[i].v[3]*cm, hlist[i].v[4]*cm,
        InjectionAltitude);
  Momentum = G4ThreeVector(hlist[i].v[0]*GeV,hlist[i].v[1]*GeV, -
hlist[i].v[2]*GeV);
  … //Some shifting and rotation codes here
  particleGun->SetParticleDefinition(particledef);
  particleGun->SetParticlePosition(Position);
  particleGun->SetParticleMomentumDirection(Momentum.unit());
  particleGun->SetParticleEnergy(particle_energy);
  particleGun->SetParticleTime(hlist[i].v[5]*ns);
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

应用程序片段

- Save a hit (CHit)：

```
chit.iddet = pmtnumber;
chit.idpart = hr->GetParticleId(particleg4name.c_str(),Z);
chit.v[0] = px/GeV;
chit.v[1] = py/GeV;
chit.v[2] = pz/GeV;
chit.v[3] = x/cm;
chit.v[4] = y/cm;
chit.v[5] = t/ns;
chit.v[6] = z/cm;
chit.v[7] = igen;
chit.v[8] = ltrack/cm;
hr->FillHit(chit);
```

数据文件

- ROOT tree：

```
KEY: TTree    t_runh;1        Run Header
KEY: TTree    t_evth;1        Event Header
KEY: TTree    t_hits;1        Hits
```

- Hits：

```
******************************************************************************
*Tree    :t_hits    : Hits
******************************************************************************
*Br    0 :iddet     : iddet/I
*Br    1 :v         : v[4]/F
*Br    2 :weight    : weight/F
******************************************************************************
```

```
******************************************************************************
*Tree    :t_runh    : Run Header
******************************************************************************
*Br    0 :n         : n/I
*Br    1 :v         : v[n]/F
*Br    2 :irun      : irun/I
*Br    3 :nevent    : nevent/I
*Br    4 :neventall : neventall/I
*Br    5 :ndet      : ndet/L
*Br    6 :ndetall   : ndetall/L
*Br    7 :nhit      : nhit/L
*Br    8 :nhitall   : nhitall/L
*Br    9 :ipevth    : ipevth/L
*Br   10 :iphits    : iphits/L
******************************************************************************
```

```
******************************************************************************
*Tree    :t_evth    : Event Header
******************************************************************************
*Br    0 :n         : n/I
*Br    1 :v         : v[n]/F
*Br    2 :irun      : irun/I
*Br    3 :ievent    : ievent/I
*Br    4 :ndet      : ndet/I
*Br    5 :nhit      : nhit/L
*Br    6 :iphitsrun : iphitsrun/L
*Br    7 :iphits    : iphits/L
******************************************************************************
```

OptParser – A Class to Parse Command Line Options

- Setting strings:
  - c-string format, like

```
optkey = "value|unit=unit1,unit2,…|key=key1,key2,…|text=explanation to this option";
optkey += "value|unit=unit1,unit2,…|key=key1,key2,…|text=explanation to this option";
```

  - Where optkey is the option; value is its default value; unit1, unit2, … are units of the value recursively; key1, key2, … are alternative keys; text is some extra text to describe the meaning of the option.
- There are definition for parameters (positioning arguments which are not covered by options) too, with that you can set the default value for unspecified parameters:

```
_p0 = "value|unit=unit1,unit2,…|key=key1,key2, …|text=this is the first parameter";
_m1 = "value|unit=unit1,unit2,…|key=key1,key2, …|text=this is the last parameter";
_functions = "explanation to this program";
_nset = "value|number of parameters";
_nmin = "value|minimum number of parameters specified";
```

Command Line to Overwrite the Default Values

- Command line:

```
command [ cpp options ] [ parser options ] [ user options ] [ parameters ]
```

  - Where cpp options are cpp specific options; Parser options include such as “-settingfile filename”, “-savesetting filename”, “-rerun”, “-showoptions”, “-showsettings”, “-showunits”, "-help", …; User options are user-specific options; Parameters are those positioning arguments not covered the cpp / parser / user options.

- The option line can be in one of the forms like: "-optkey value", "--optkey=value", "-optkey+ value", "--optkey+=value", "-optkey", "+optkey", "--optkey=true", "--optkey=false", where the latter few are for switches (options has bool values).

Access Values

- Access to option / parameter values:
  - For options, using functions like

```
optchar("optkey"), optint("optkey"), optfloat("optkey"), optlong("optkey"),
optlonglong("optkey"), optbool("optkey"), optstring("optkey"),
optvdouble("optkey",&n), ...
```

  - For parameters, using functions like

```
optchar(ipos), optint(ipos), optfloat(ipos), optlong(ipos),
optlonglong(ipos), optbool(ipos), optstring(ipos), optvdouble(ipos,&n), ...
```

where ipos is the parameter position.

More On the OptParser

- Some special features:
  - Run control:
    - Could be used to define and access a key-wised configuration database. The speed is quite fast, and the capacity is large:
      - Dependent on number of keys, at 1000 keys level, very access takes <8e-7 seconds;
      - 1000 keys with full explanations uses only 2.3 MB RSS memory.
  - cpp:
    - Support cpp in the setting file:
      - where you can include any other files in cpp style and directory control algorithm with "#include", and control part of the setting lines appearing / disappearing with "#ifdef", …
  - Unit:
    - Support physical constants & units that defined by GEANT4/CLHEP;
  - Save settings:
    - Support saving all settings to a file:
      - rerun the program in the completely same conditions later on with the saved setting file (as all arguments are saved in the file too).
    - Book-keeping the full settings of a production job.

Examples

- See the package itself;
- See the new version of G4WCDA.

Might be A Superior Solution

- XML:
  - Need xerces;
  - Need to define DTD;
  - Need external setting files (not a solution for a small program);
  - …
- Python:
  - Need to learn the language;
  - Might be slow;
  - Need to external setting files;
  - …
- OptParser: a standard c++ class:
  - Convenient, easy to compile;
  - The syntax is quite simple and explicit;
  - With / without setting files;
  - Powerful.

总结

- 提出了一种全新的基于hit流的探测器模拟方法：
  - 可解决WCDA模拟计算经常遇到的内存饱和的问题；
  - 大幅降低探测器真实化所需的计算资源；
  - 可以把所有LHAASO探测器整合，实现联合模拟。
- 实现了此种方法的框架程序：
  - Corsika数据的随机读取程序（CorsikaReader）；
  - Hit流处理程序包（HitsReader）；
  - 命令行解析、控制参数（包括参数单位）和数据库调用程序包（OptParser）。
- 基于此概念和框架程序，正在实现WCDA模拟程序的完全更新。