Merging NeutronHP and ParticleHP

KOI, Tatsumi

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Timeline

2014-June

Most codes of ParticleHP are duplicated from NeutronHP

2014-Summer

MT migration of NeutronHP

Implementation of ParticleHP for handling inelastic reactions of p,d,t,He3 and a

2014-Dec

Geant4 10.01 was released

NeutronHP: sharing data in multithreading library

ParticleHP : handling p, d, t, He3, a and n as projectile particle

2015-Spring

MT migration of ParticleHP

2015-June

Geant4 10.02 beta was released

ParticleHP : sharing data in multithreading library

2015-Summmer

Checking physics performance of ParticleHP for neutron

Delete NeutronHP from release code

2015-Dec

Geant4 10.02 will be released

Only ParticleHP will exist in release

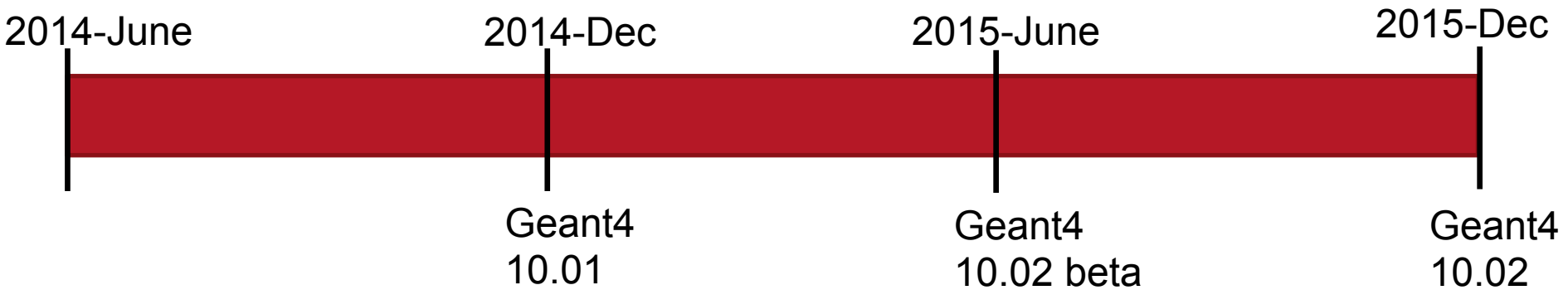
Timeline

NeutronHP

Implementation for sharing data in MT mode

Comparing Result

Delete most of classes from release



Codes are duplicated from NeutronHP

Implementation for handling inelastic reactions of d,t,He3 and a

Implementation for sharing data in MT mode

Comparing Result

ParticleHP

Requirement

Validate equivalent physics performance

Reproducibility in random number level will not provide

See Pedro's plenary presentation

Functionalities that currently NeutronHP offers should be remained

Several functionalities had lost during the first implementation of ParticleHP, but now they are revived.

Change should be transparent from user (and other codes)

NeutronHP classes disappeared from release code

G4NeutronHPElasticData will be replaced by G4ParticleHPElasticData(G4Neutron::Neutron())

Many reference physics list use NeutronHP

```
#include "G4NeutronHPElasticData.hh"
#include "G4NeutronHPElastic.hh”

void G4NeutronHPBuilder::
Build(G4HadronElasticProcess * aP)
{
  if(theHPElastic==0) theHPElastic = new
G4NeutronHPElastic;
  theHPElastic->SetMinEnergy(theMin);
  theHPElastic->SetMaxEnergy(theMax);
  if(theHPElasticData == 0) theHPElasticData
= new G4NeutronHPElasticData;
  aP->AddDataSet(theHPElasticData);
  aP->RegisterMe(theHPElastic);
}
```

We hope to remain this codes

by introducing

G4NeutronHPElasticData.hh

```
#ifndef G4NeutronHPElasticData_h
#define G4NeutronHPElasticData_h 1

#include "G4ParticleHPElasticData.hh"
using G4NeutronHPElasticData =
G4ParticleHPElasticData;

#endif
```

G4NeutronHPElastic.hh

```
#ifndef G4NeutronHPElastic_h
#define G4NeutronHPElastic_h 1

#include "G4ParticleHPElastic.hh"
using G4NeutronHPElastic = G4ParticleHPElastic;

#endif
```

Several other package also uses NeutronHP

SLAC

I have modified them to use ParticleHP with consent of responsible

Confirmed they work as good as before

Sharing data in Multithreading library

NeutronHP package had spent huge amount of o memory (~GB).

Most of them are static in calculation. They had to be shared among worker threads

This was happen in v10.01 on NeutronHP and in v10.02.beta in ParticleHP

Sharing data in Multithreading library

Remain codes unchanged as much as possible

- do not re-design entire package

Rely on support of G4 utilities for multithreading

- G4Threading::IsWorkerThread()
- G4Cache
- No explicit Mutex Lock in HP

I should admit that this work made the package more fragile than before

- Need careful treatment in future development

Sharing data in MT library IsWorkerThread()

SLAC

```
void G4NeutronHPElasticData::BuildPhysicsTable(const G4ParticleDefinition& aP)
{
,,,,,,,,,,,,,,
   if ( G4Threading::IsWorkerThread() ) {
      theCrossSections = G4NeutronHPManager::GetInstance()->GetElasticCrossSections();
      return;
   }

  size_t numberOfElements = G4Element::GetNumberOfElements();

  if ( theCrossSections == NULL )
      theCrossSections = new G4PhysicsTable( numberOfElements );
   else
      theCrossSections->clearAndDestroy();

  // make a PhysicsVector for each element
  static G4ThreadLocal G4ElementTable *theElementTable  = 0 ; if (!theElementTable) theElementTable= G4Element::GetElementTable();
  for( size_t i=0; i<numberOfElements; ++i )
  {
    G4PhysicsVector* physVec = G4NeutronHPData::
      Instance()->MakePhysicsVector((*theElementTable)[i], this);
    theCrossSections->push_back(physVec);
  }

   G4NeutronHPManager::GetInstance()->RegisterElasticCrossSections(theCrossSections);
}
```

Sharing data in MT library G4Cache

SLAC

G4NeutronHPProduct.hh

```
#include "G4Cache.hh”
class G4NeutronHPProduct
{
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
   struct toBeCached {
      G4ReactionProduct* theNeutron;
      G4ReactionProduct* theTarget;
      G4int theCurrentMultiplicity;
      toBeCached() :
theNeutron(NULL),theTarget(NULL),theCurre
ntMultiplicity(-1) {};
   };
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
 private:
     G4Cache<toBeCached> fCache;
}
```

G4NeutronHPProduct.cc

```
G4ReactionProductVector *
G4NeutronHPProduct::Sample(G4double
anEnergy)
{
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
 G4double mean = theYield.GetY(anEnergy);
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
theDist->SetTarget(fCache.Get().theTarget);
  theDist-
>SetNeutron(fCache.Get().theNeutron);
  G4int i;
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
fCache.Get().theCurrentMultiplicity =
static_cast<G4int>(mean);
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
}
```

Reproducibility problem in multithreading library

Random number level reproducibility is not grantee in HP package under current design of hadronic process and mechanism of event distribution in multithreading library.

Physics level reproducibility is fine

If we turn off caches in CrossSectionDataStore, then the random number level reproducibility will be gotten.

Now there is no way to turn off them

This is not related to sharing data in multithreading library

Current status

SLAC

Most merging works are done

We are waiting green signal from validation efforts for deleting NeutronHP classes

I think now we have based on Pedro's Tuesday presentation

But I am completely neutral to postpone deleting them to 10.03