Cross-section improvements for MT

Witek Pokorski

26.09.2013

Content

- Cross-section framework improvements and adaptation
- Cross-section classes improvements and adaptation

Cross-section framework improvements and adaptation

Factory and MT (1/2)

- functionality of G4CrossSectionDataSetRegistry has been extended
  - responsible for instantiating cross sections
    - user should never 'new' cross section object
    - registry (singleton) provides the method GetCrossSectionDataSet(const G4String& name)
- unique cross-sections objects (for a give cross section) shared across the application

Factory and MT (2/2)

- thanks to Andrea's work this is now MT compatible
- G4CrossSectionFactoryRegistry split from DataSetRegistry
- G4CrossSectionDataSetRegistry is G4ThreadLocal
- G4CrossSectionFactoryRegistry is shared among threads
  - cross sections shared

Cross-section classes improvements and adaptation for MT

My personal comment

- This part of the talk should not be needed ;-)
  - Cross sections should have nothing (or very little) to do with MT
- If cross sections needs adaption for MT it means that there is a problem with the design of cross-section classes
  - there should be nothing to improve in cross-sections for MT
- cross sections (after some initialization) should be 'read-only' functions of particle, energy, material, returning a number

CHIPS-derived cross-sections

- bad design
- enormous use of 'statics'
  - completely not needed
  - creating problems for MT
- obscure code to implement cache
  - inefficient, error-prone, MT-unsafe

Statics...

- enormous abuse of statics in CHIPS-derived code
- none of those statics are needed!

```
private:
  static G4ThreadLocal G4int     lastN;    // The last N of calculated nucleus
  static G4ThreadLocal G4int     lastZ;    // The last Z of calculated nucleus
  static G4ThreadLocal G4int     lastF;    // Last used in the cross section TheFirstBin
  static G4ThreadLocal G4double* lastJ1;   // Pointer to the last array of the J1 function
  static G4ThreadLocal G4double* lastJ2;   // Pointer to the last array of the J2 function
  static G4ThreadLocal G4double* lastJ3;   // Pointer to the last array of the J3 function
  static G4ThreadLocal G4int     lastL;    // Last used in the cross section TheLastBin
  static G4ThreadLocal G4double  lastE;    // Last used in the cross section Energy
  static G4ThreadLocal G4double  lastTH;   // Last value of the Energy Threshold
  static G4ThreadLocal G4double  lastSig;  // Last value of the Cross Section
  static G4ThreadLocal G4double  lastG;    // Last value of gamma=lnE-ln(me)
  static G4ThreadLocal G4double  lastH;    // Last value of the High energy A-dependence

  // Vector of pointers to the J1 tabulated functions
  static G4ThreadLocal std::vector <G4double*> *J1_G4MT_TLS_;

  // Vector of pointers to the J2 tabulated functions
  static G4ThreadLocal std::vector <G4double*> *J2_G4MT_TLS_;

  // Vector of pointers to the J3 tabulated functions
  static G4ThreadLocal std::vector <G4double*> *J3_G4MT_TLS_;
};
```

```
// Vector of pointers to the J1 tabulated functions
G4ThreadLocal std::vector<G4double*> *G4ElectroNuclearCrossSection::J1_G4MT_TLS_ = 0;

// Vector of pointers to the J2 tabulated functions
G4ThreadLocal std::vector<G4double*> *G4ElectroNuclearCrossSection::J2_G4MT_TLS_ = 0;

// Vector of pointers to the J3 tabulated functions
G4ThreadLocal std::vector<G4double*> *G4ElectroNuclearCrossSection::J3_G4MT_TLS_ = 0;
```

```
  // *** Begin of the Associative memory for acceleration of the cross section calculations
  static G4ThreadLocal std::vector <G4int> *colN_G4MT_TLS_ = 0 ; if (!colN_G4MT_TLS_) colN_G4MT_TLS_ = new  std::ve
// Vector of N for calculated nucleus (isotop)
  static G4ThreadLocal std::vector <G4int> *colZ_G4MT_TLS_ = 0 ; if (!colZ_G4MT_TLS_) colZ_G4MT_TLS_ = new  std::ve
// Vector of Z for calculated nucleus (isotop)
  static G4ThreadLocal std::vector <G4int> *colF_G4MT_TLS_ = 0 ; if (!colF_G4MT_TLS_) colF_G4MT_TLS_ = new  std::ve
// Vector of Last StartPosition in the Ji-function tables
  static G4ThreadLocal std::vector <G4double> *colTH_G4MT_TLS_ = 0 ; if (!colTH_G4MT_TLS_) colTH_G4MT_TLS_ = new  s
*colTH_G4MT_TLS_;   // Vector of the energy thresholds for the eA->eX reactions
  static G4ThreadLocal std::vector <G4double> *colH_G4MT_TLS_ = 0 ; if (!colH_G4MT_TLS_) colH_G4MT_TLS_ = new  std:
*colH_G4MT_TLS_;    // Vector of HighEnergyCoefficients (functional calculations)
  // *** End of Static Definitions (Associative Memory) ***
```

G4ElectroNuclearCross Section

```
G4bool
G4ElectroNuclearCrossSection::IsIsoApplicable(
          const G4DynamicParticle* aParticle, G4int /*Z*/,
          G4int /*A*/, const G4Element*, const G4Material*)
{
  G4bool result = false;
  if (aParticle->GetDefinition() == G4Electron::ElectronDefinition())
    result = true;
  if (aParticle->GetDefinition() == G4Positron::PositronDefinition())
    result = true;
  return result;
}
```

```
G4double
G4ElectroNuclearCrossSection::GetIsoCrossSection(
          const G4DynamicParticle* aPart,
          G4int ZZ, G4int AA,
          const G4Isotope*, const G4Element*, const G4Material*)
{
  static const G4int nE=336; // !!  If you change this, change it in GetFunctions() (*.hh) !!
  static const G4int mL=nE-1;
  static const G4double EMi=2.0612; // Minimum
  static const G4double EMa=50000.; // Maximum
  static const G4double lEMi=std::log(EMi);  //
  static const G4double lEMa=std::log(EMa);  //
  static const G4double dlnE=(lEMa-lEMi)/mL; //
  static const G4double alop=1./137.036/3.14159265; //coef. for the calculated functions (Ee>50000.)
  static const G4double mel=0.5109989;       // Mass of the electron in MeV
  static const G4double lmel=std::log(mel);       // Log of the electron mass
  // *** Begin of the Associative memory for acceleration of the cross section calculations
  static std::vector <G4int> colN;       // Vector of N for calculated nucleus (isotop)
  static std::vector <G4int> colZ;       // Vector of Z for calculated nucleus (isotop)
  static std::vector <G4int> colF;       // Vector of Last StartPosition in the Ji-function tables
  static std::vector <G4double> colTH;   // Vector of the energy thresholds for the eA->eX reactions
  static std::vector <G4double> colH;    // Vector of HighEnergyCoefficients (functional calculations)
  // *** End of Static Definitions (Associative Memory) ***

  const G4double Energy = aPart->GetKineticEnergy()/MeV; // Energy of the electron
  const G4int targetAtomicNumber = AA;
  const G4int targZ = ZZ;
  const G4int targN = targetAtomicNumber-targZ; // @@ Get isotops (can change initial A)
  if (Energy<=EMi) return 0.;                   // Energy is below the minimum energy in the table

  G4int PDG=aPart->GetDefinition()->GetPDGEncoding();
  if (PDG == 11 || PDG == -11)                  // @@ Now only for elec
  {
```

Many of those repeated in several methods of the class

completely useless check

Modifications

- moved all the static consts to the beginning of the .cc file (outside any method)
  - they are static (local) in the compilation unit (.o file)
    - unitialized/calculated when loading the library - no problem anymore for MT
- static variables moved to become data members

Cache in CHIPS XS

- caching has been greatly simplified by moving to per-element cross sections
- we need to validate it in MT environment
  - probably need a lock for writing in the cache
    - writing in cache happens only at the beginning (when going through new materials), so lock should not be a problem

Conclusion

- cross section registry and factories ported to MT by Andrea
- most of the MT-related 'tricks' became not needed in CHIPS cross-sections
  - still need to look at the cache
- my hope is that we can make all cross-sections classes completely MT-neutral (and shared between all the threads)