Hadronics Framework and MT

A. Dotti, M. Kelsey

Parallel session 7B - Hadronics issues related to MT

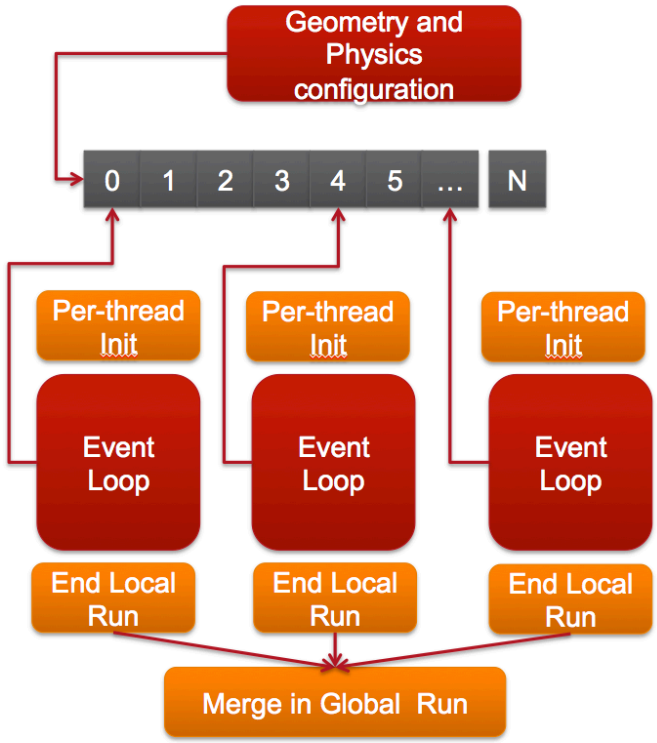
U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Review of MT workflow

SLAC

- Basic design: only most memory consumption objects are shared
  - Geometry, EM tables
- There is a special "thread" (not a real thread, the main function): the master. It owns fully initialized G4 (physics, geometry), done in sequential mode, but does not process events during the event loop



Our goal

SLAC

- Up to now **Hadronics is thread-private**:
  - Each worker owns instances of hadronics model/physics
  - Processes do not share anything
- To further reduce memory usage we can share parts of hadronics
- Use master thread to get data to-be-shared
  - Similarly to what is done in EM

1. The master thread is configured before workers
2. Workers EM processes get the pointer of the "pre-initialized" data to be shared

Kernel

SLAC

- Kernel cannot have knowledge of hadronics framework
- Kernel has **single shared** instance of G4VUserPhysicsList, during run initialization:

1. It loops on all particles and calls:
   `G4VserPhsyicsList::PreaprePhysicsTable( G4ParticleDefinition*) ::BuildBhysicsTable(G4ParticleDefinition*)`
2. These will loop on all processes attached to the particles and call:
   - `G4VProcess::{Prepare,Build}PhysicsTable( const G4ParticleDefinition&)` for sequential and master thread
   - `G4VProcess::{Prepare,Build} WorkerPhysicsTable( const G4ParticleDefinition&)` for workers

Extended G4VProcess interface

SLAC

- `virtual void BuildWorkerPhysicsTable(const G4ParticleDefinition& part) { BuilPhysicsTable(part); }`
- `virtual void PrepareWorkerPhysicsTable(const G4ParticleDefinition&) { PreparePhysicsTable(part); }`
- The two methods provide default behavior (**fully backward compatible**)
- Additional method:
  - `const G4VProcess* GetMasterProcess() const;`
  - Can be used to get to-be-shared parts of process

Hadronic framework

SLAC

- Two **separate** entities that can have a MT awareness:
  - Cross-sections
  - Hadronic Models

- Since the two are separate need to address both independently
- **G4HadronicProcess is generic container, should be modified minimally**

CrossSection : general considerations

SLAC

- Base class of hadronics framework, inherits from G4VProcess
- `G4HadronicProcess::PreparePhysicsTable( part )` registers process for particle in TLS G4HadronicProcessStore, nothing to do with XS
- `G4HadronicProcess::BuildPhysicsTable( part )`
  - Forward calls to G4CrossSectionDataStore::BuildPhysicsTable
  - That loops on all XS to call equivalent method

- Do we need to implement a `BuildWorkerPhysicsTable` in cross-section classes?
  - **No** if we use factory when we want to share

Factory mechanism

SLAC

- Two assumptions:

1. Cross-section is implemented with factory mechanism
2. **The entire cross-section object can be shared among threads**

- if (1&&2) use factory macros:
  - **G4_DECLARE_XS_FACTORY** Factory creates cross-section for each thread
  - **G4_DECLARE_SHAREDXS_FACTORY** Factory creates a singleton (shared) cross-section
- To be tested, will need some further tuning
- If this does not cover all cases, we need to implement new `WorkerBuildPhysicsTable` mechanism

Models

SLAC

- `G4HadronicProcess` inherits from `G4VProcess`
- **Models are not owned directly by processes**, but registered in the `G4EnergyRangeManager` (one for each `G4HadronicProcess`)
- Models can be shared among processes

- For models `G4HadronicInteraction` there is **no state aware** methods: needs an "initialize" and "initialize for thread"

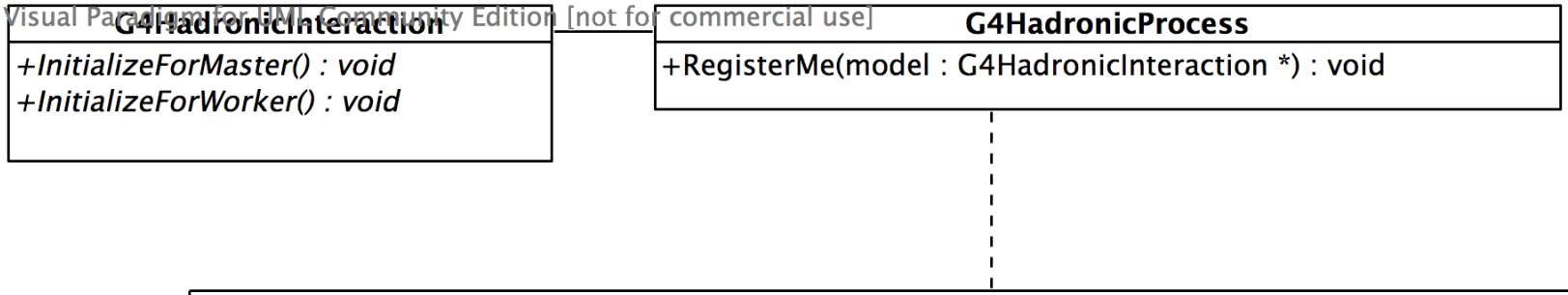
Proposal

SLAC

- `G4HadronicInteraction::InitializeForMaster() , ::InitializeForWorker()`
  - Virtual methods
  - With default empty implementation
  - Full backward compatibility
- Modify `G4HadronicProcess::RegisterMe( G4HadronicInteraction *a )` to call the correct initialize
  - Process knows if it is master or worker

- **Limitation**: to implement for worker models "GetMasterModel" is more complex (but can be done with some caveats), do we need this?

Possible implementation



RegisterMe modifications:

```
void RegisterMe( G4HadronicInteraction* aModel)
// [...]

  G4bool isMaster = ( this == GetMasterProcess() );
  if ( isMaster ) {
    aModel->InitializeForMaster();
  } else {
      aModel->InitializeForWorkers();
  }

return;
}
```

Conclusions

SLAC

- A possible inclusion of MT capabilities in HAD framework is possible in an **evolutionary approach**
  - Without changing public interfaces (e.g. only adds methods)
  - Fully backward compatible
  - One limitation: models do not have access to "master" model (can be changed)
  - Cross-sections are shared entirely (e.g. full object) in a very simple way (single XS can still implement ad-hoc sharing of parts of data strictures)