

# Reducing Memory footprint Strategies for MT

A. Dotti, M. Kelsey

Parallel session 7B – Hadronics issues related to MT

- Each threads own instances of Hadronic models/processes/cross-section
- A part of the per-thread memory overhead is due to hadronics
- How much is it?
- Memory Profile FullCMS application: 1 thread, a single 50 GeV pi- event with FTFP\_BERT
  - Check memory allocations (e.g. churn)
  - Concentrate on initialization routines
  - Show here only methods that allocate more than 1 MB of memory (e.g. concentrate on the “hot-spots”)

# Overview

- Measurements done on Mac OS X, numbers not so different from Linux box
- Master thread allocates 113.4 MB
- Worker thread: 24 MB
  - Includes everything
  - Concentrate on Hadronics calls in next slides

## Results (all tags updated to Monday 16<sup>th</sup> Sept)

SLAC

3.51 MB	▼ G4WorkerRunManager::DoEventLoop(int, char const*, int) libG4run.dylib	➡
3.51 MB	▼ G4WorkerRunManager::ProcessOneEvent(int) libG4run.dylib	
3.51 MB	▼ G4EventManager::DoProcessing(G4Event*) libG4event.dylib	
3.50 MB	▼ G4TrackingManager::ProcessOneTrack(G4Track*) libG4tracking.dylib	
3.48 MB	▼ G4SteppingManager::Stepping() libG4tracking.dylib	
2.27 MB	▼ G4SteppingManager::InvokePostStepDoItProcs() libG4tracking.dylib	
2.27 MB	▼ G4SteppingManager::InvokePSDIP(unsigned long) libG4tracking.dylib	
1.38 MB	G4HadronicProcess::PostStepDoIt(G4Track const&, G4Step const&) libG4pr	
1.02 MB	▼ G4SteppingManager::DefinePhysicalStepLength() libG4tracking.dylib	
1.00 MB	▼ G4VDiscreteProcess::PostStepGetPhysicalInteractionLength(G4Track const&, d	
1.00 MB	▼ G4HadronicProcess::GetMeanFreePath(G4Track const&, double, G4ForceCon	
1.00 MB	▼ G4CrossSectionDataStore::GetCrossSection(G4DynamicParticle const*, G4M	
1.00 MB	G4CrossSectionDataStore::GetCrossSection(G4DynamicParticle const*, G4	

Processes/models during event loop: lost of relatively small (<1MB) objects, to be studied more in detail

Hadronic cross-sections account for about 2MB in total, see previous presentation, possibilities to reduce

## Results (all tags updated to Monday 16<sup>th</sup> Sept)

SLAC

6.82 MB	▼G4RunManagerKernel::RunInitialization()	libG4run.dylib	➡
6.82 MB	▼G4RunManagerKernel::BuildPhysicsTables()	libG4run.dylib	
6.82 MB	▼G4VUserPhysicsList::BuildPhysicsTable()	libG4run.dylib	
5.64 MB	▼G4VUserPhysicsList::PreparePhysicsTable(G4ParticleDefinition*)		
3.27 MB	▶G4VEnergyLossProcess::PreparePhysicsTable(G4ParticleDefinition)		
2.34 MB	▶G4VEmProcess::PreparePhysicsTable(G4ParticleDefinition const&)		
1.18 MB	▼G4VUserPhysicsList::BuildPhysicsTable(G4ParticleDefinition*)	lib	
1.18 MB	▼G4HadronicProcess::BuildPhysicsTable(G4ParticleDefinition const&)		
1.18 MB	G4CrossSectionDataStore::BuildPhysicsTable(G4ParticleDefinit		

Hadronic cross-sections account for about 2MB in total, see previous presentation, possibilities to reduce

## Results (all tags updated to Monday 16<sup>th</sup> Sept)

SLAC

9.00 MB	▼G4RunManager::Initialize() libG4run.dylib
9.00 MB	▼G4RunManager::InitializePhysics() libG4run.dylib
9.00 MB	▼G4RunManagerKernel::InitializePhysics() libG4run.dylib
8.79 MB	▼G4VModularPhysicsList::ConstructProcess() libG4run.dylib
5.70 MB	▼G4IonPhysics::ConstructProcess() libG4physicslists.dylib
5.53 MB	▼G4BinaryLightIonReaction::G4BinaryLightIonReaction(G4VPreComp
5.53 MB	▼G4BinaryCascade::G4BinaryCascade(G4VPreCompoundModel*) l
5.53 MB	▼G4Scatterer::G4Scatterer() libG4processes.dylib
5.53 MB	►void G4ForEach<G4Pair<G4CollisionNN, G4Pair<G4CollisionN



Memory churn from BIC model: note this is done even if the model is not used

Some work needed (not a trivial fix)

## Measurements Conclusions

- The hadronics most memory hungry (5MB) hot-spot is **BIC** model (even when not used). Some rework needed
- The second Hadronics components using more memory are **cross-sections** (2.2MB) stored in G4CrossSectionDataStore
- Models/processes account for about 1MB of memory
- It is realistic to reduce memory footprint for Hadronics of a factor 2
- Note: other models have a completely different profile
  - HP models: currently each thread load all HP tables, test I I for HP uses several GB of memory. No work on this done yet
  - Requires strategy for sharing database files

# Reducing memory footprint

---

- In the following a procedure to reduce memory footprint is shown
  - The aim is to propose a step-by-step guide that can help also non-MT experts
- Some special cases may require thinking or redesigning few spots here and there



# Reducing memory footprint

- Good candidates for sharing are “static” objects/tables
  - Search in your code large arrays of numbers (cross-sections) these are very good candidate for sharing
  - Also look at large objects created at run time (e.g. a table being calculated)
- In G4 there is a very good chance these objects are already marked as “static”
- To make these thread-safe these have been transformed to TLS:
  - `Static G4ThreadLocal double largeData[100] = { .... };`
  - `Static G4ThreadLocal double largeData[100] = calculateXS();`

# Const objects

- The easiest thing to do is to try to use the “const” keyword. If you can add “const” than you can probably transform:
  - `Static G4ThreadLocal double largeData[100] = {...}`
  - `Static const double largeData[100] = {...}`
- Nothing else to do
- Good practice: use const as much as you can, including in method signatures:
  - `Const G4Something* GetSomething( const G4Data& ) const;`

# Const objects

- However, consider the following example (they do exist in G4):

- G4Class.cc

```
G4Class::G4Class() { ... }
```

```
Void G4Class :: Method() {  
    static G4ThreadLocal double largeData[100] = someFunc();  
}
```

- This should not be transformed simply removing G4ThreadLocal (long discussion, I can provide pointers)

- Try the following:

```
Namespace {  
    static double largeData[100];  
    G4Mutex aMutex = G4MUTEX_INITIALIZER;  
}  
G4Class::G4Class() {  
    G4AutoLock l(&aMutex);  
    largeData = someFunc(); //initialize static data  
}  
Void G4Class::Method() {  
}
```

## Lazy initialized objects

- None of the above work if: initialization is lazy and depends on quantities calculated during event loop (e.g. a model cross-section for a specific ion created only if ion is found in interaction)
- In such case, **you probably cannot** share the object (unless you use costly locks that should be always avoided!)
  - Safer solution: leave as it is, however try to remove both static and G4ThreadLocal (it has a small but non zero cost every time you use the variable): move to class data member
  - If these data are top list of memory consuming, we can work together on that and find a different solution

## Tradeoffs: memory vs speed

- Remember Mike's receipt, in order of preference when you see a "static G4ThreadLocal":
  1. Try to **remove** G4ThreadLocal applying one of the suggested receipt
  2. If not possible and memory consumption is not large: **move** to class data member (no memory reduction, but at least no penalty for G4ThreadLocal) – do a profile yourself! (ask Performance Task Force how to)
  3. If neither possible/desirable: **leave** as it is now, probably the best solution
  4. In very special cases (though I cannot think of good example): **convert** to local variables
  5. In any case **avoid** locks and mutex, if you think that you absolutely need them, let's discuss ...