Parallelizing ROOT geometry

Andrei Gheata

Forum on Concurrent Programming Models and Frameworks

June 6, 2012

Why parallelizing geometry ?

- Geometry is a key component in many HEP applications
  - Simulation MC, reconstruction, event displays, geometry DB's, …
- Some use geometry as wrapper to extract 3D or material information
  - like positions, sizes, matrices, alignment, densities, …
  - Optimizing the usage of this info is application responsibility
    - Like OpenGL internally decomposes the objects in lower level representations that can be handled in parallel
- Some of the HEP applications use directly geometry functionality, namely navigation
  - Namely transport MC and tracking code
  - As these applications can and will be parallelized, geometry has to follow…

What kind of parallelism ?

- Geometry is a utility – it has to be thread safe
  - Covered in this presentation
- Navigation is iterative – next step cannot start unless last one finished
  - Query -> propagate -> query -> propagate
  - Has to support at top level a task-based parallelism (e.g. different tracks to different threads)
- Navigation algorithms are hard to factorize
  - Tree-oriented queries (ups and downs in a hierarchy of volumes/nodes)
  - Answers are results of a minimization procedure, it is hard to work ahead
  - The state changes and has to be propagated all along the query
- Most natural low level factorization – solids
  - Main loops organized at volume/voxels level
  - 3D shapes are the local “computation objects” and contain most CPU-expensive algorithms
  - Good candidates for GPU kernels, but communication of the state can be a limiting factor due to memory bus latency
- Vectorization – ideal for low level computation
  - Propagating several state vectors (position, direction) to the same solid type
    - If solids are vector-aware, can we assemble decent vectors to feed the same solid?
- Long term development – to be addressed by the unified solids project

Geometry data structures

- ROOT geometry was NOT thread safe by design
  - In the attempt to maximize re-usage of cached geometry states or pre-computed values, state-related info was carried by many geometry data types
    - Voxel optimisation structures, divisions, assembly shapes, composite shapes, geometry manager
- Many methods, including simple getters, were not thread safe
- The stateful part of the geometry was not clearly separated from the ***const*** one

```
class TGeoPatternFinder : public TObject
{
…
   Double_t         fStep;          // division step length
   Double_t         fStart;         // starting point on divided axis
   Double_t         fEnd;           // ending point
   Int_t            fCurrent;       // current division element
   Int_t            fNdivisions;    // number of divisions
   Int_t            fDivIndex;      // index of first div. node
   TGeoMatrix      *fMatrix;        // generic matrix
   TGeoVolume      *fVolume;        // volume to which applies
   Int_t            fNextIndex;     //! index of next node
```

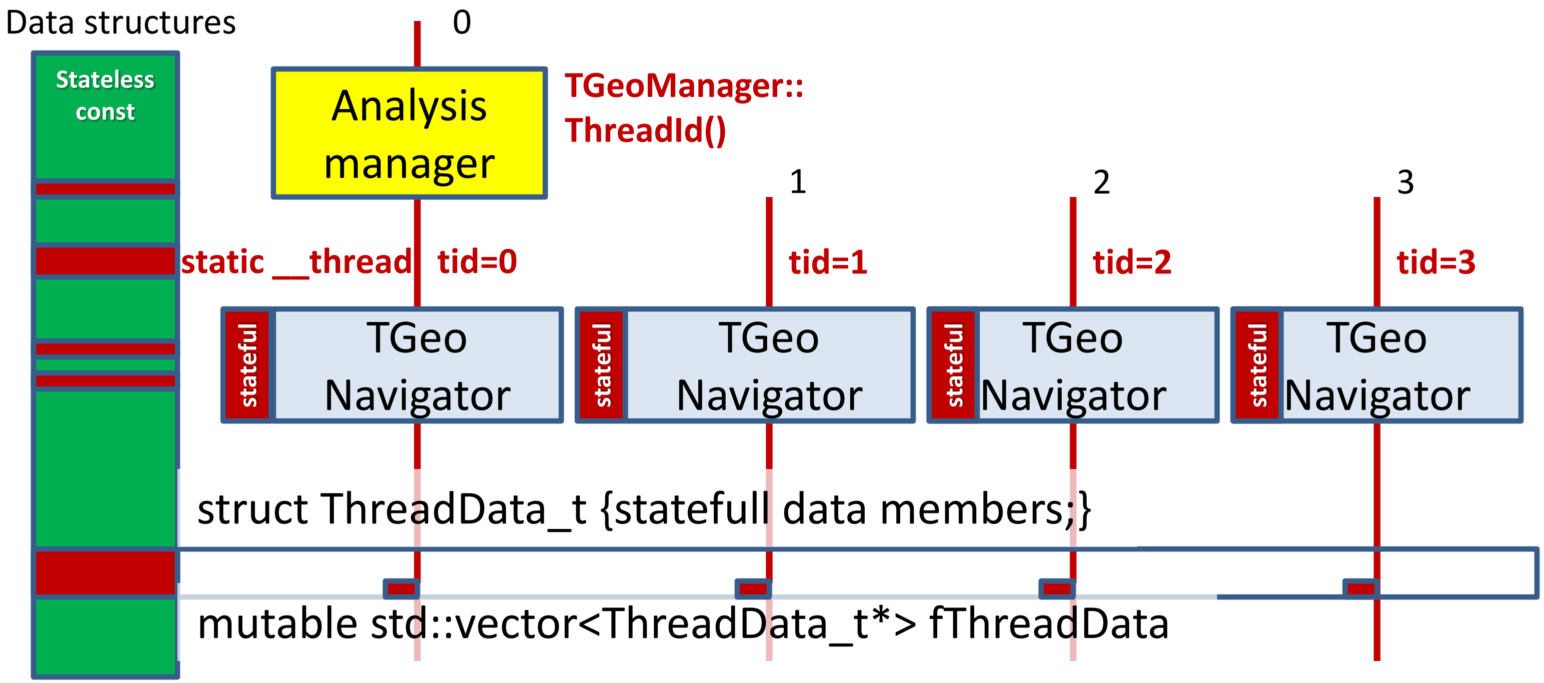
Re-design strategy

- The goal was to make geometry thread safe without sacrificing existing optimizations
- Step 1: Split out the navigation part from the geometry manager
  - Most data structures here depend on the state
  - Different calling threads will work with different navigators
- Step 2: Spot all thread unsafe data members and methods within the structural geometry objects and protect them
  - Shapes and optimization structures
  - Convert object->data into object->data[thread_id]
- Step 3: Rip out all stateful data from structural objects to keep a compact ***const*** access geometry core
  - Whenever possible, percolate the state in the calling sequence

Problems along the way

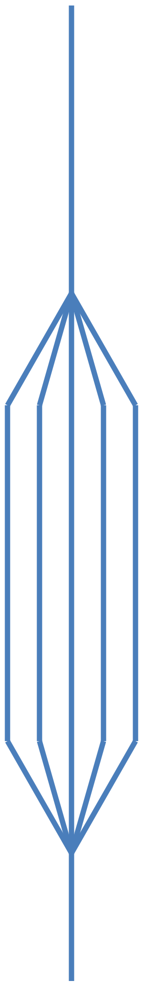
- Separating navigation out of the manager was a tedious process
  - Keeping a large existing API functional
- Spotting the thread-unsafe objects was not obvious
  - Practically all work done by Matevz Tadel (thanks!)
- Changing calling patterns was sometimes impossible, resources needed to be locked
  - First approach suffered a lot from Amdahl law
- Many calls to get the thread Id needed, while there was no implementation of TLS in ROOT
  - *__thread* not supported everywhere

Implementation



- Thread data pre-alocated via *TGeoManager::SetMaxThreads()*
- User threads have to ask for a navigator via *TGeoManager::CreateNavigator()*
- Getting access to a stateful data member goes via:
  - statefulObject->GetThreadData(tid)->fData
  - For voxel structures they are ripped out into stateful data in the navigator, passed as arguments to methods

Usage



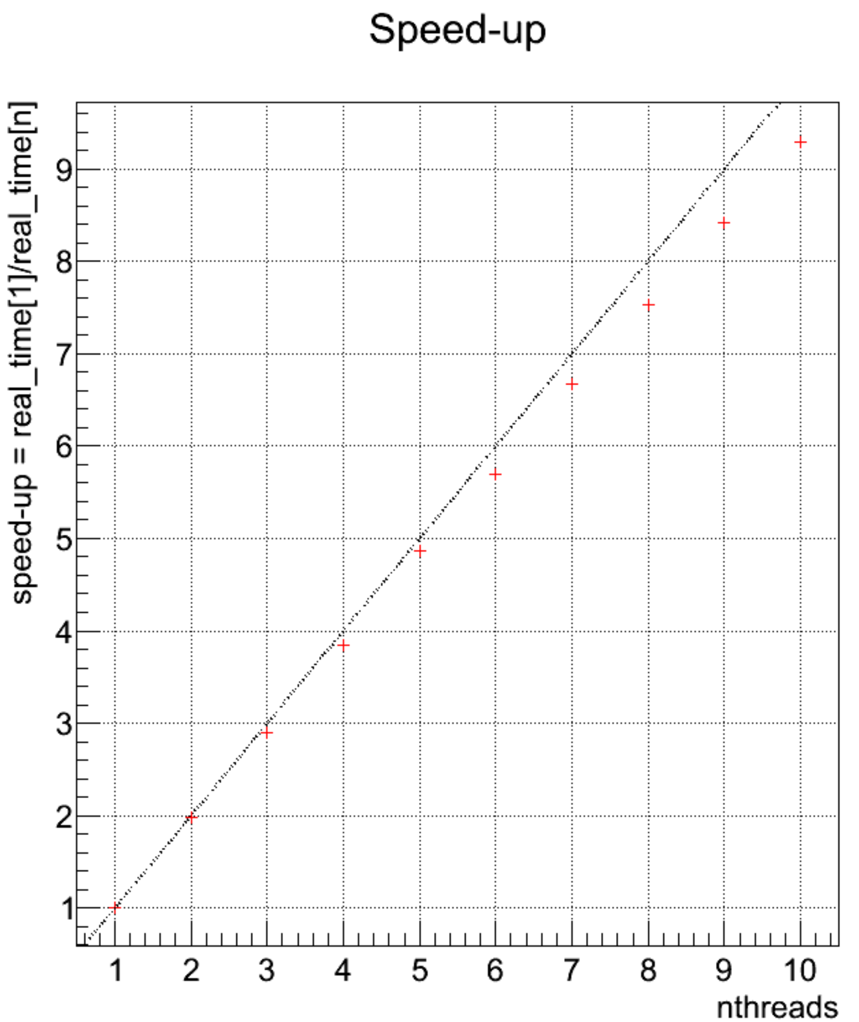
```
//_______________________________________________________________
MyTransport::PropagateTracks(tracks)
{
// User transport  method called by the main thread
   gGeoManager->SetMaxThreads(N);  // mandatory
   SpawnNavigationThreads(N, my_navigation_thread, tracks)

      void *my_navigation_thread(void  *arg)
      {
      // Navigation method to be spawned as thread
         TGeoNavigator *nav = gGeoManager->GetCurrentNavigator();
          if (!nav) nav = gGeoManager->AddNavigator();
          int  tid = nav->GetThreadId();       // or   TGeoManager::ThreadId()
          PropagateTracks(subset(tid,tracks));
          return 0;
      }

JoinNavigationThreads();
}
```

Speed-up

- Good scalability with rather small Amdahl effects (~0.7 % sequential
  - No lock on memory resources however !
  - Work balancing is not perfect (worsen by CPU throttling)
- Small overheads due to several hidden effects
  - Context switches, false cache sharing (?), pthread calls
  - May need to re-organize stateful data per thread rather than



Overview

- Thread safety for geometry achieved, introducing 1-2% overhead compared to the initial version
  - Additional ThreadId() and GetThreadData() calls
  - Fast and portable thread ID retrieval implemented via ***ThreadLocalStorage.h***
    - ***__thread*** for Linux/AIX/MACOSX_clang
    - ***__declspec(thread)*** on WIN
    - ***pthread_(set/get)specific*** for SOLARIS, MACOSX
- Parallel navigation to be enabled via: gGeoManager->SetMaxThreads(N)
  - Each thread works with its own navigator: gGeoManager->AddNavigator()
- No locks, very good scalability
  - Stateful data structures migrated to thread data arrays, allocated at initialization
    - fStateful -> struct ThreadData_t {<type> fStateful;} -> std::vector<ThreadData_t*>
  - Small overheads (~0.7%) to be investigated

Future plans

- Geometry navigation is not a simple parallel problem – re-shuffling of navigation algorithms will be needed
  - Propagating vectors from top to bottom
    - When available from the tracking code…
    - Re-think algorithms for solids from this perspective
  - Factorizing loops within a volume
    - Low level optimization at the level of voxels
    - Data flow model and locality to be thought over
      - Minimizing latencies and cache misses when using low-level computational units (GPU)
- The time scale is few years, but the work has to start now
  - Many of these issues to be addressed in the Unified Solids framework