Rethinking thread pool management in ROOT

Xavier Valls

ROOT
Data Analysis Framework
https://root.cern

Current functionality of TPoolManager

- A ROOT::Internal singleton that, once it’s been initialized, pins the number of threads to use for any (TBB-based) multithreaded application in ROOT.

- Also acts as a lifetime manager for the TBB scheduler, destroying the instance when not in use (when the shared_ptr to the TBB task scheduler becomes unowned). Once destroyed, can be initialized with a different number of threads.

- Solves undefined behaviours caused by the interaction of the implicit and explicit multithreading modes in ROOT. (PPP meeting 9-Feb-2017)

Current functionality

```cpp
//We initialize the scheduler with 4 threads
ROOT::EnableIMT(4);
{
    //The scheduler is active, so the value passed to TThreadExecutor
    //is overriden with the number of threads the scheduler has been
    //initialized with (4)
    TThreadExecutor executor(9);
}
ROOT::DisableIMT();

//The scheduler is not alive at this point,
//so we initialize it with 2 threads
ROOT::EnableIMT(2);
ROOT::TThreadExecutor executor(8);
ROOT::DisableIMT();

executor.MapReduce(...); //Runs on two threads!

ROOT::EnableIMT(3);
//Still two threads! TThreadExecutor instance was keeping the scheduler alive
```

Current functionality

```
ROOT::TThreadExecutor executor(4);
ROOT::TThreadExecutor executor2(8); //will run limited to two threads!!
```

Defects

- Doesn’t solve dependencies between the implicit and explicit multithreading modes.
- Unexpected behaviour in the eyes of the user. Why does IMT affect EMT? Why can’t I have several instances of EMT classes with differing number of threads? “It’s in the documentation” not enough.
- Uninformed usage of pool size getters. “But it works!” doesn’t make it correct.

Objectives

- Decoupling the implicit and explicit multithreading executions modes in ROOT
- In EMT, allow the instantiation of MT classes managing a different number of threads.
- Keep current functionality of IMT

How

Working with a tbb::task_arena we create on top of the implicit one returned by tbb::task_scheduler_init

Proposed changes

- TPoolManager is still the life manager of the TBB task scheduler, but always initializes to the max number of threads in the system.
- IMT, TThreadExecutor save the number of threads they have been initialized with.
- TThreadExecutor handles tbb::task_arenas instead of directly the task_scheduler.
- If IMT enabled, the default constructor of TThreadExecutor is built with the number of threads defined by IMT.

New behaviour

```
//Each of the executor manages its own tbb::task_arena,
//which allows the co-existance of TThreadExecutors
//handling different number of threads.
TThreadExecutor executor1(8); //will run on 8 threads
TThreadExecutor executor2(4); //will run on 4 threads

//IMT keeps a different task Arena too!
ROOT::EnableIMT(4); //4 threads will be used in IMT operations
//executor3 will be initialized with 4 threads for backward
//compatibility. Should we not allow this interaction?
//Should it be initialized with the default number of threads?
TThreadExecutor executor3; //Implicit constructor. Initialized with 4 threads.
ROOT::DisableIMT();

ROOT::EnableIMT(2); //2 threads will be used in IMT operations
ROOT::TThreadExecutor executor(8); //Explicit number of threads.
                                   // Will execute on 8 threads.
ROOT::EnableIMT(4); //2 threads will be used in IMT operations
                    //Doesn't change until disabled!
                    //Should we allow it instead?
ROOT::DisableIMT();
```

New Behaviour

```cpp
{
//TThreadExecutor holds a shared_ptr to the tbb::task_scheduler
TThreadExecutor executor1(8); //will run on 8 threads
}

//executor1 went out of scope and was destroyed together with the scheduler.
//No scheduler active at this point.

ROOT::EnableIMT(4);

//"IMT" holds holds a shared_ptr to the tbb::task_scheduler. Scheduler alive here.
// DisableIMT() will destroy the IMT reference to the scheduler. The reference count of
// the scheduler reaches zero and it gets destroyed.
ROOT::DisableIMT();


ROOT::EnableIMT(4);
TThreadExecutor executor3; //Implicit constructor. Initialized with 4 threads.
ROOT::DisableIMT();

//The scheduler is still alive here because of executor3
```

PR #2389

https://github.com/root-project/root/pull/2389

Decisions to take

- Decide on explicit-implicit MT execution modes' interactions. Should TThreadExecutor totally independent of IMT?
- Allow EnableImplicitMT to change number of threads without disabling?
- Rename TPoolManager (not exposed to the user anymore)