Asynchronous Execution of Single Work Items

D. Piparo (CERN)

PPP

Passing Work to the Runtime

- In ROOT, users submit work to the runtime with *TThreadExecutor*
- Wait for all the tasks to be finished before continuing: **blocking**
- Can we submit work in a non-blocking fashion, e.g. à la std::async ?

```
auto myFuture = std::async(doWork, a, b); // creates thread, run doWork(a,b)
auto myResult = myFuture.get(); // waits for the result, blocking
```

Why? Why Now?

- **Context: read of ROOT columnar datasets**
- **Use case: parallel decompression of baskets** (Zhe Zhang)
- Principle ("circular" explanation):
  1. Wait for previous bunch of data to be decompressed (contains many events)
  2. Fetch next bunch of data (from disk or remotely)
  3. Kick-off its decompression asynchronously
  4. Start processing of data
  5. goto 1.

Presently implemented with direct calls to TBB, it's desirable to avoid that

Enters Asynchronous Execution

- Need to submit work to the runtime and continue program flow
- **Check if work is done even after seconds**
- Cannot do this with // for: it blocks!

Two possible solutions:

1. ROOT::Experimental::Async
2. ROOT::Experimental::TTaskGroup

ROOT::Experimental::Async

- Returns a future, can take functions and arguments in input

```
template< class Function, class... Args>
std::future<std::invoke_result_t<std::decay_t<Function>,std::decay_t<Args>...>>
ROOT::Experimental::Async( Function&& f, Args&&... args );
```

Usage:

```
auto myFuture = ROOT::Experimental::Async(doWork, a , b);
doOtherWork();
auto result = myFuture.get();
```

Difficulties:

- Must avoid leakage of TBB headers and we have templates
- Type erasure?

ROOT::Experimental::TTaskGroup

- A class.
- Accepts work items with void(void) signature.
- Has a *void Run(const std::function<void, void>& f)* and *void Wait()* methods
- Inspired from tbb::task_group

Usage:

```
ROOT::Experimental::TTaskGroup tg;
tg.AsyncRun(doWork); // doWork could be a closure...
tg.AsyncRun(doWork2); // doWork could be a closure...
doWork3();
tg.wait();
```

It seems much easier to implement

Proposal

First round:

- Implement TTaskGroup
- Put it in libIMT
- Protect it with ifdefs as TThreadExecutor
- Add test and tutorial

→ Zhe should be unblocked and can finalise PR

Second round:

- Provide an implementation for ROOT::Async