Subject: Re: Job package: A lightweight multithreading tool, using promise/future mechanism
Posted by Oblivion on Sun, 10 Sep 2017 17:08:28 GMT
View Forum Message <> Reply to Message

And below is where CoWork starts to outperform Job (As you pointed out, because of the additional copying involved, I guess.):

I changed the code to see other options:

```
{
  CoWork jobs;
  Vector<String> results;
  TIMING("CoWork");
  jobs & [=, &results] {
   Vector<String> s;
   for(int i = 0; i < 50000; i++)
     s.Add() = GetDivisors();
    CoWork::FinLock();
    results = pick(s);
  };
  jobs.Finish();
 }

 {
  Job<Vector<String>> job;
  TIMING("Job");
  job.Start([=]{
   Vector<String> s;
   for(int i = 0; i < 50000; i++)
    s.Add() = GetDivisors();
   return pick(s);
  });
  job.Finish();
  auto s = job.GetResult();
 }
```

```
TIMING Job           :  1.42 s  -  1.42 s  ( 1.42 s  / 1 ), min:  1.42 s , max:  1.42 s , nesting: 1 - 1
TIMING CoWork        :  1.39 s  -  1.39 s  ( 1.39 s  / 1 ), min:  1.39 s , max:  1.39 s , nesting: 1 - 1
```

Job is not an alternative to CoWork, but it's not a bad tool either. It does simplify writing high performance MT code in a convenient way, thanks to U++.

It is suitable for such asynchronous operations mainly where a high latency is expected (IO/sockets, etc.) and where the code needs to be easily managable (errors, and results should be easily and immediately dealt with.)

Best regards,
Oblivion