Subject: New parallelization pattern with CoWork
Posted by mirek on Tue, 26 Dec 2017 10:44:31 GMT
View Forum Message <> Reply to Message

While trying to parallelize Painter, I have found that there is a set of algorithms that are 'too short' to parallelize with CoWork operator& and 'too unstable' to parallelize with CoPartition. 'too short', because the cost of creating Event (which is about 80ns) is too large compared to the work done by job in parallel. 'too unstable' means that the time taken by single job highly unpredictable, so assigning regularly sized chunks of work to threads tends to some threads ending early and one thread going for long time.

Thinking about the issue, I have found a new parallelization pattern (and implemented it with CoWork):

```
CONSOLE_APP_MAIN
{
 SeedRandom(0);
 Vector<String> data;
 for(int i = 0; i < 1000; i++) {
  int n = Random(7);
  data.Add();
  for(int j = 0; j < n; j++)
   data.Top() << Random() << ' ';
 }

 double sum = 0;
 CoWork co;
 co * [&] {
  double m = 0;
  int i;
  while((i = co.Next()) < data.GetCount()) {
   CParser p(data[i]);
   while(!p.IsEof())
    m += p.ReadDouble();
  }
  CoWork::FinLock();
  sum += m;
 };

 RDUMP(sum);
}
```

operator* schedules all CoWork threads to run a single routine. 'co.Next' then atomically returns increasing numbers (from 0), which are used to distribute the work.

So in this 'outside-in' approach, just a single lambda is created and only a handful of jobs have to

be started for things to work and management overhead is thus greatly reduced...