Subject: CoWork::FinLock
Posted by mirek on Sat, 09 Jan 2016 19:24:59 GMT
View Forum Message <> Reply to Message

Do not ask me why, but recently I was having fun with parallel programming (I looks like it will be unintended addition for the next release... :).

In the process, while optimizing everything around CoWork, I got an interesting idea, CoWork::FinLock. It is intended to lock storing partial results into shared target at the end of scheduled routine. The idea is that CoWork has to lock some mutex anyway after the routine ends, so it is possible to 'prelock' this internal mutex and join both critical sections.

Usage looks something like this:

```
template <typename I, typename Lambda>
void CoLoop(I begin, I end, const Lambda& lambda)
{
 size_t chunk = max(size_t((end - begin) / CPU_Cores()), (size_t)1);
 CoWork co;
 while(begin < end) {
  co & [=] {
   lambda(begin, begin + min(chunk, size_t(end - begin)));
  };
  begin += chunk;
 }
}

template <class I, class Better>
I CoBest(I begin, I end, const Better& better)
{
 I best = begin++;
 CoLoop(begin, end,
  [=, &best](I i, I e) {
   I b = i++;
   while(i < e) {
    if(better(*i, *b))
     b = i;
    i++;
   }
   CoWork::FinLock(); // better is shared across threads, needs a lock!
   if(better(*b, *best))
    best = b;
  }
 );
 return best;
}
```

(CoBest is intended to lookup e.g. minimum or maximum, 'best' value).

Subject: Re: CoWork::FinLock
Posted by koldo on Sun, 10 Jan 2016 17:10:32 GMT
View Forum Message <> Reply to Message

Hello Mirek

To help with parallel programming in U++ I have uploaded some benchmarks in new package Bazaar/OpenMP_demp. I hope it will happen as positive NTL vs. STL comparison :)

About storing partial results, it could be interesting to see demo "Pi" at it uses reduction() clause to handle temporary results between cores.

```
static double pi_device() {
 double x, y;
 long count = 0, i;

 // Parallel loop with reduction for calculating PI
 #pragma omp parallel for private(i, x, y) shared (samples) reduction(+:count)
 for (i = 0; i < samples; ++i) {
  x = Random(1000000)/1000000.;
  y = Random(1000000)/1000000.;
  if (sqrt(x*x + y*y) < 1)
   count++;
 }
 return 4.0 * count / samples;
}
```

Subject: Re: CoWork::FinLock
Posted by mirek on Sun, 10 Jan 2016 21:42:32 GMT
View Forum Message <> Reply to Message

koldo wrote on Sun, 10 January 2016 18:10Hello Mirek

To help with parallel programming in U++ I have uploaded some benchmarks in new package Bazaar/OpenMP_demp. I hope it will happen as positive NTL vs. STL comparison :)

On this level (I mean reduction demo), it is unlikely - at best the results will be the same.

Mirek

Subject: Re: CoWork::FinLock
Posted by mirek on Sun, 10 Jan 2016 21:51:45 GMT
View Forum Message <> Reply to Message

For the record, here is the implementation with CoLoop listed here:

```
int samples = 100000000;
int count = 0;
CoLoop(0, samples,
 [=, &count](int a, int b) {
  int lcount = 0;
  while(a < b) {
   double x = Randomf();
   double y = Randomf();
   if(sqrt(x*x + y*y) < 1)
    lcount++;
   a++;
  }
  CoWork::FinLock();
  count += lcount;
 }
);
```