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...

---

Subject: Re: New parallelization pattern with CoWork
Posted by Oblivion on Tue, 26 Dec 2017 23:36:36 GMT
View Forum Message <> Reply to Message

Hello Mirek,

This is good news. Thanks for your efforts!
U++ MT is really improved over time.

My initial impression: It looks and works good.

Here's another example (a visually more pleasing, and more concrete one I hope) :)
This is to give some basic idea about it's possible usage to our fellow U++ users and newcomers.

```
void Mandelbrot::DrawFractal(const Rectf& r, int ix, int wcount)
{
 // This method draws the famous Mandelbrot fractal, using the escape time algorithm (slow).
 // Important note: Imagebuffer is accessed without using any locking mechansim.
 // While it is OK here, do not do this in real life unless you know what you're doing!

 auto sz = GetSize();
 Vector<Rect> regions;
 Sizef scale = iscale(sz, Sizef(1.0, 1.0), r.GetSize());

 for(int i = 0, cx = sz.cx / wcount, mod = sz.cx % wcount; i < wcount; i++) {
  auto x = i * cx;
  regions.Add(Rect(x, 0, x + cx + (i == wcount - 1 ? mod : 0), sz.cy));
 }
 ImageBuffer canvas(sz);

 if(CoWork::GetPoolSize() < wcount)
  CoWork::SetPoolSize(wcount);
 CoWork co;
 {
  RTIMING("Mandelbrot calculation with CoWork");
  co * [&] {
   int j = 0;
   while((j = co.Next()) < regions.GetCount()) {
    const auto& rr = regions[j];
    for(auto y = rr.top; y < rr.bottom; y++) {
     RGBA *pixel = canvas[y];
     for(auto x = rr.left; x < rr.right; x++) {
      Complex c(r.left + x / scale.cx, r.top + y / scale.cy);
      auto z = c;
```

```
      auto i = 0;
      while(abs(z) < 2 && i < ix) {
       z = z * z + c;
       i++;
      }
      if(i < ix) {
        double jj = i + 1 - log(log2(abs(z)));
        *(pixel + x) = HsvColorf(jj / double(ix),  1.0, (double(ix) / 256.0) * 1.888);
      }
      else *(pixel + x) = Black();
     }
    }
   }
  };
 }
 img = canvas;
 Refresh();
}
```

Best regards,
Oblivion

## File Attachments

1) Mandelbrot.png, downloaded 698 times

---

Subject: Re: New parallelization pattern with CoWork
Posted by koldo on Wed, 27 Dec 2017 10:15:52 GMT
View Forum Message <> Reply to Message

Hello Mirek

How would it be better to include parallel routines into U++ classes ?
- Just to include them directly
- Adding an option to choose the parallel version or not

---

Subject: Re: New parallelization pattern with CoWork
Posted by mirek on Wed, 27 Dec 2017 15:19:18 GMT
View Forum Message <> Reply to Message

koldo wrote on Wed, 27 December 2017 11:15Hello Mirek

How would it be better to include parallel routines into U++ classes ?
- Just to include them directly
- Adding an option to choose the parallel version or not

Not sure what you mean.

We have parallel algos

here

and I am adding support where possible, like with Painter...

Anyway, generic tools are definitely needed, one way or another.

---

Subject: Re: New parallelization pattern with CoWork
Posted by koldo on Wed, 27 Dec 2017 20:58:56 GMT
View Forum Message <> Reply to Message

Hi Mirek

For example ScatterCtrl uses either Draw or Painter.
Painter is much better, but Draw is faster.
In case of Painter, what could be the best way to do it?:
- If MT is chosen, to use CoWork directly
- Use CoWork just if it is explicitly indicated

---

Subject: Re: New parallelization pattern with CoWork
Posted by mirek on Thu, 28 Dec 2017 07:40:41 GMT
View Forum Message <> Reply to Message

koldo wrote on Wed, 27 December 2017 21:58Hi Mirek

For example ScatterCtrl uses either Draw or Painter.
Painter is much better, but Draw is faster.
In case of Painter, what could be the best way to do it?:
- If MT is chosen, to use CoWork directly
- Use CoWork just if it is explicitly indicated

Add 'Co' method to ScatterCtrl that activates 'Co' method of Painter.

BTW, the problem with Painter::Co is that sometimes it slows it down.

---

Subject: Re: New parallelization pattern with CoWork
Posted by koldo on Thu, 28 Dec 2017 08:57:54 GMT
View Forum Message <> Reply to Message

OK Mirek

I will see how and when the performance is improved.

Subject: Re: New parallelization pattern with CoWork
Posted by Didier on Sat, 06 Jan 2018 11:17:24 GMT
View Forum Message <> Reply to Message

Hi,

looks very interesting and most importantly very easy and handy to use

Good job