Subject: CoWork::Finish() can wait in a worker thread while there are jobs to do
Posted by busiek on Thu, 15 Dec 2016 20:30:55 GMT
View Forum Message <> Reply to Message

Hi Mirek,

It is said that CoWork instances can be nested. Looking into the code I see that it is also possible to use CoWork in a job (i.e. in a worker thread). Therefore CoWork::Finish() can be called inside a worker thread. The current implementation of Finish() waits if there is no more jobs to take from the local queue of CoWork while not all local jobs are finished. This is a waste of resources because a worker should never wait while there are any global jobs to do from the pool. Why not implement CoWork::Finish() like that:

```
void CoWork::Finish() {
 Pool& p = GetPool();
 p.lock.Enter();
 while(todo) {
  if(!jobs.IsEmpty(1)) {
   LLOG("Finish: todo: " << todo << " (CoWork " << FormatIntHex(this) << ")");
   p.DoJob(*jobs.GetNext(1));
  } else if(is_worker && !p.jobs.IsEmpty()) {
   LLOG("Do global job while WaitForFinish (CoWork " << FormatIntHex(this) << ")");
   p.DoJob(*p.jobs.GetNext());
  } else if(is_worker)
   p.waiting_threads++;
   LLOG("Waiting for job in WaitForFinish");
   p.waitforjob.Wait(p.lock);
   LLOG("Waiting ended in WaitForFinish");
   p.waiting_threads--;
  } else {
   LLOG("WaitForFinish (CoWork " << FormatIntHex(this) << ")");
   waitforfinish.Wait(p.lock);
  }
 }
 p.lock.Leave();
 LLOG("CoWork " << FormatIntHex(this) << " finished");
}?
```

The only problem is that a worker can wait on different conditional variables. If the worker waits on the global conditional variable (p.waitforjob) and some other worker finishes remaining jobs in local CoWork queue reducing todo to zero, the worker will not be waked up since the other worker signals waitforfinish. Also removing the entire "else if(is_worker)" block and waiting only on waitforfinish is a bad solution either since if a new job is scheduled by PushJob() the worker will not be waked up. Therefore more complicated code is needed. I attach a patch with a proposition. Simply one has to track all waiting workers in Finish() which I call "waiting masters" and singal one of them in PushJob().

Jakub

File Attachments

1) 0001-CoWork-Finish-don-t-wait-in-a-worker-thread.patch, downloaded 198 times

---

Subject: Re: CoWork::Finish() can wait in a worker thread while there are jobs to do
Posted by mirek on Fri, 16 Dec 2016 00:33:03 GMT
View Forum Message <> Reply to Message

busiek wrote on Thu, 15 December 2016 21:30Hi Mirek,

It is said that CoWork instances can be nested. Looking into the code I see that it is also possible to use CoWork in a job (i.e. in a worker thread). Therefore CoWork::Finish() can be called inside a worker thread. The current implementation of Finish() waits if there is no more jobs to take from the local queue of CoWork while not all local jobs are finished. This is a waste of resources because a worker should never wait while there are any global jobs to do from the pool. Why not implement CoWork::Finish() like that:

```
void CoWork::Finish() {
 Pool& p = GetPool();
 p.lock.Enter();
 while(todo) {
  if(!jobs.IsEmpty(1)) {
   LLOG("Finish: todo: " << todo << " (CoWork " << FormatIntHex(this) << ")");
   p.DoJob(*jobs.GetNext(1));
  } else if(is_worker && !p.jobs.IsEmpty()) {
   LLOG("Do global job while WaitForFinish (CoWork " << FormatIntHex(this) << ")");
   p.DoJob(*p.jobs.GetNext());
  } else if(is_worker)
   p.waiting_threads++;
   LLOG("Waiting for job in WaitForFinish");
   p.waitforjob.Wait(p.lock);
   LLOG("Waiting ended in WaitForFinish");
   p.waiting_threads--;
  } else {
   LLOG("WaitForFinish (CoWork " << FormatIntHex(this) << ")");
   waitforfinish.Wait(p.lock);
  }
 }
 p.lock.Leave();
 LLOG("CoWork " << FormatIntHex(this) << " finished");
}?
```

The only problem is that a worker can wait on different conditional variables. If the worker waits on the global conditional variable (p.waitforjob) and some other worker finishes remaining jobs in local CoWork queue reducing todo to zero, the worker will not be waked up since the other worker signals waitforfinish. Also removing the entire "else if(is_worker)" block and waiting only on waitforfinish is a bad solution either since if a new job is scheduled by PushJob() the worker will not be waked up. Therefore more complicated code is needed. I attach a patch with a proposition.

Simply one has to track all waiting workers in Finish() which I call "waiting masters" and singal one of them in PushJob().

Jakub

Hi,

it is deliberate. The issue is that

a) 'global' job can be part of some really unrelated work and can take order of magnitude more time, thus blocking this CoWork to finish.

b) stack issue. Diving into global job can end in another Finish, that would dive into yet another globa... and we unexpectedly run out of stack.

Really, Finish doing 'any global job' was previous implementation, the limit to 'my jobs' is new.

Mirek

Subject: Re: CoWork::Finish() can wait in a worker thread while there are jobs to do
Posted by busiek on Fri, 16 Dec 2016 00:43:59 GMT
View Forum Message <> Reply to Message

I see. However my implementation schedules "master worker" in PushJob() only if there is no other workers waiting. Nevertheless, it is a problem dependent.
In general one could even set priorities to jobs and the policy can be then to jump to global job only if it has higher priority. But this can get too complicated. For my purposes it suffices that Finish() has 2nd bool argument telling whether it is allowed to jump to global job or not. Modification to the patch is simple then.

Subject: Re: CoWork::Finish() can wait in a worker thread while there are jobs to do
Posted by mirek on Fri, 16 Dec 2016 08:22:04 GMT
View Forum Message <> Reply to Message

I do not believe that 'jumping to higher priority' really helps (but depends on how you define priority...).

Realistically, I am going to keep it on the safe side here. I think that eventual benefit from 'global jobs' is really small and dangers of really unexpected behavior are high. Really, think when the Finish would really go to perform global job: It is when all of its CoWork jobs are spend, means it is waiting for workers to finish the last pieces of work. That should be quite short periods of time.

BTW, here is some reading:

https://software.intel.com/en-us/blogs/2010/12/09/tbb-schedu ler-clandestine-evolution

(however, that thing seems a bit overengineered to me, but I believe it illustrates some points)

All that said, are you solving the real problem (in your code), or just poking around CoWork sources?

Subject: Re: CoWork::Finish() can wait in a worker thread while there are jobs to do
Posted by busiek on Fri, 16 Dec 2016 11:54:32 GMT
View Forum Message <> Reply to Message

I am working for a real problem. It is very difficult algorithmic problem. After reducing complexity and optimizing all possible bottlenecks the last step is parallelization. There are many dependent chunks. It looks more like DAG of jobs with low branching. Each chunk can be time consuming and because it often happens that I can distinguish just few (often two) parallel local jobs and than move forward to next part, I would loose too much time waiting in Finish(). I had to edit it. I am using Pipe also, it is really helpful. Generally U++ for algorithmic problems is just great.

Subject: Re: CoWork::Finish() can wait in a worker thread while there are jobs to do
Posted by mirek on Fri, 16 Dec 2016 14:27:29 GMT
View Forum Message <> Reply to Message

busiek wrote on Fri, 16 December 2016 12:54I am working for a real problem. It is very difficult algorithmic problem. After reducing complexity and optimizing all possible bottlenecks the last step is parallelization. There are many dependent chunks. It looks more like DAG of jobs with low branching. Each chunk can be time consuming and because it often happens that I can distinguish just few (often two) parallel local jobs and than move forward to next part, I would loose too much time waiting in Finish(). I had to edit it. I am using Pipe also, it is really helpful. Generally U++ for algorithmic problems is just great.

OK. You might consider:

- maybe you can have single CoWork for the whole problem (but probably not)

- you can also increase the number of threads - then OS thread would be spent waiting in Finish, but not CPU core

Anyway, I am really interested how this goes. How is Pipe working for you? I have created that as sort of experiment, so it is nice to see it used.

Please let me know, after you are done: I expect you to test with 'global' stealing and without; the I shall decide based on results whether it is worth changing things.

Maybe we could allow stealing 'parent' jobs? That would solve the problem, right?

Mirek
