Lock Yourself Out

Ruud van der Pas

Distinguished Engineer

SPARC Microelectronics

Santa Clara, CA, USA

SC'17 Talk at OpenMP Booth

Tuesday, November 14, 2017

Heads Up! Mini Tutorial!

OpenMP

If you have a question

Our friendly booth staff is always happy to help

In case your question is about OpenMP, there is on-site support too

OpenMP™

OpenMP Booth – Lock Y

Ruud van der Pas

SC17 Denver, CO hpc connects.

What Is A Lock ?

OpenMP

A lock is "something" acquired by a thread

While it has the lock, nobody else gets it

This allows the thread to do its work in private, not bothered by other threads

Once finished, it has to release the lock

And allow another thread to do its private business

A Real-World Example

OpenMP

All want to steal a cookie from the jar

While someone selects a cookie, nobody else is allowed to steal a cookie as well

The Code

```
#include <omp.h>

omp_lock_t myLock;

int main(….)
{
   (void) omp_init_lock(&myLock);

   #pragma omp parallel
   {
       (void) TheCookieJar();

   } // End of parallel region

   (void) omp_destroy_lock(&myLock);
}
```

Inside The Cookie Jar

```
void TheCookieJar()
{
   (void) omp_set_lock(&myLock);     // acquire lock

    (void) grabMyFavouriteCookie(); // get cookie

   (void) omp_unset_lock(&myLock);   // release lock

}
```

Why Bother About Locks ?

In OpenMP, a lock is a special variable

And lives in memory

To check availability, a thread has to read it

That means data has to travel to the thread

This takes some time, or a lot of time

So What ?

OpenMP

A lock serializes execution

The bigger the system, the worse it gets

But don't worry, small systems suffer too

Sooner than later, a lock destroys scalability

The Evil Family Member

```
#include <omp.h>

int64_t myCounter = 0;

int main(….)
{

   #pragma omp parallel
   {
      #pragma omp atomic update
         myCounter++;

   } // End of parallel region

}
```

Deep Inside The Dark Dungeon

OpenMP

(void) my_atomic_add (&a,1);

This is the code in "fake assembly"*

```
   ld    [%o0],%r1       ! Load: Copy value from address in o0 into r1
again:
   add   %r1,1,%r2       ! Compute new value: %r2 = %r1 + 1
   cas   [%o0],%r1,%r2   ! CAS: if [%o0] == %r1 then swap [%o0] and %r2
   cmp   %r1,%r2         ! Compare: if %r2 == %r1, CAS succeeded, values
                         ! are swapped and [%o0] has the new value
   bne   %icc, again     ! Branch: CAS failed, try again

   <register management and return>
```

**) And a very naive implementation*

Why Is It Evil ?

OpenMP

Atomic operations are potentially evil too

Despite instruction level support (e.g. “cas”)

A big part of the cost is in the data transfer

Sooner than later, an atomic operation destroys scalability

A Case Study

The Code

```
#pragma omp parallel shared(job)
{
     int64_t i;
     while(1) {
       #pragma omp atomic capture
       {
           i = job->counter;
           job->counter++;
       }
       if ( i >= job->end ) break;

       do_the_work(i);

     } // End of while-loop
} // End of parallel region
```

Good for load balancing

Bad for scalability

Main Idea

This Can Be Transformed Into A Loop

The Modified Code – Set Up


```
int64_t Nthreads                = … // Number of threads
int64_t total_chunks_of_work    = … // Total size of work
int64_t chunk_size_per_thread = … // Work per thread

for(int k=0; k < Nthreads; k++)
{
    thread_work_chunks_start[k] = …
    thread_work_chunks_end  [k] = …
}
```

The Modified Code – Core Part/1

```
#pragma omp parallel
{

  omp_set_lock(&my_lock);
    if ( index_work >= Nthreads ) {
       done = true;
    } else {
      start_loop = thread_work_chunks_start[index_work];
      end_loop   = thread_work_chunks_end  [index_work];
      index_work++;
    }
  omp_unset_lock(&my_lock);

    <the computational part of the code is on the next slide>

} // End of parallel region
```

The Modified Code – Core Part/2

```
if (!done) {

  for (chunk=start_loop; chunk <= end_loop; chunk++)
  {

      do_the_work(chunk);

  } // End of loop over chunks

}
```

Worse for load balancing

Very good for scalability

The Experiments

OpenMP

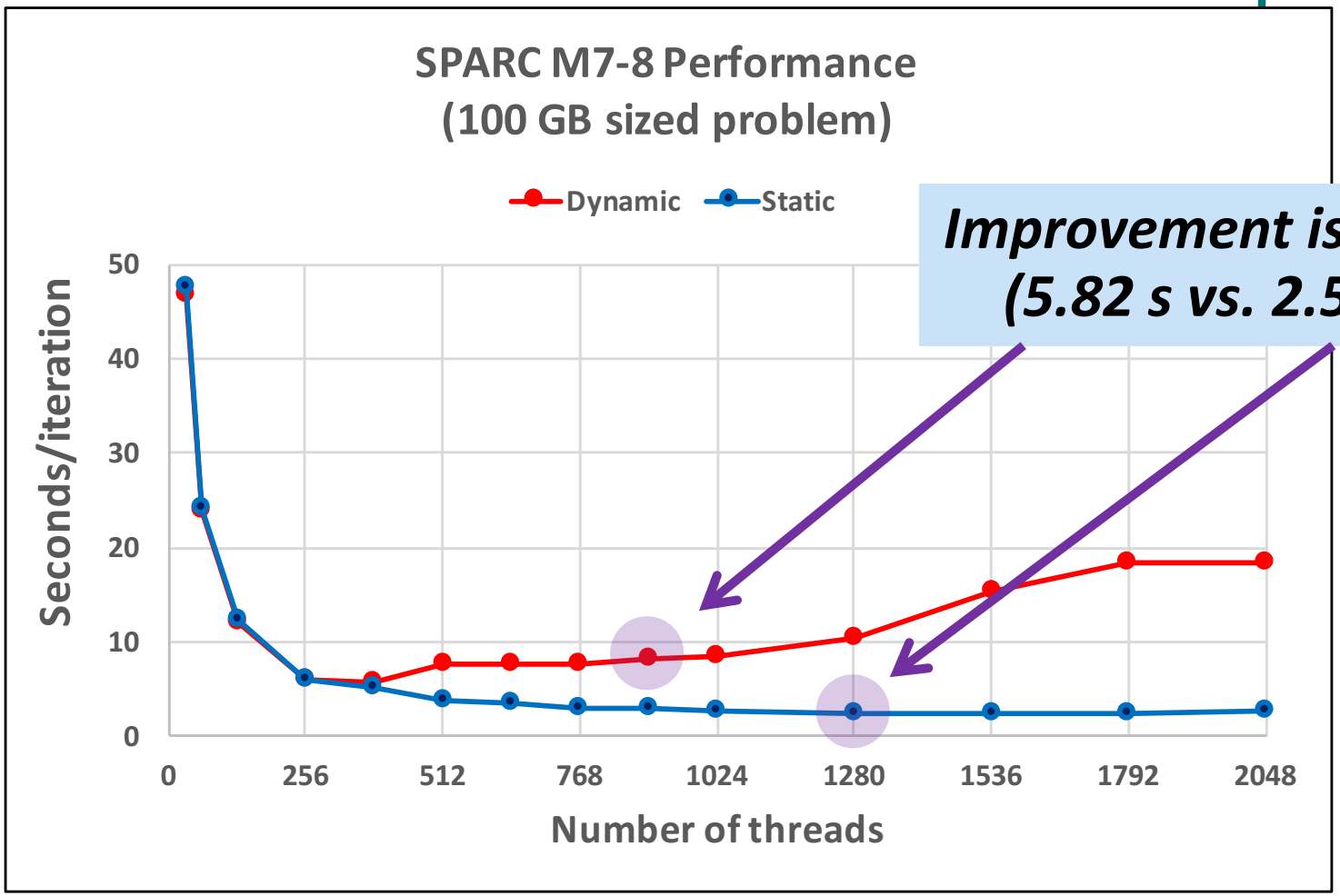
System Used

SPARC M7-8 Server, 4.1 GHz
Total of 256 cores, 2048 threads
8 sockets, 32 cores/socket, 8 threads/core
4 TB of memory

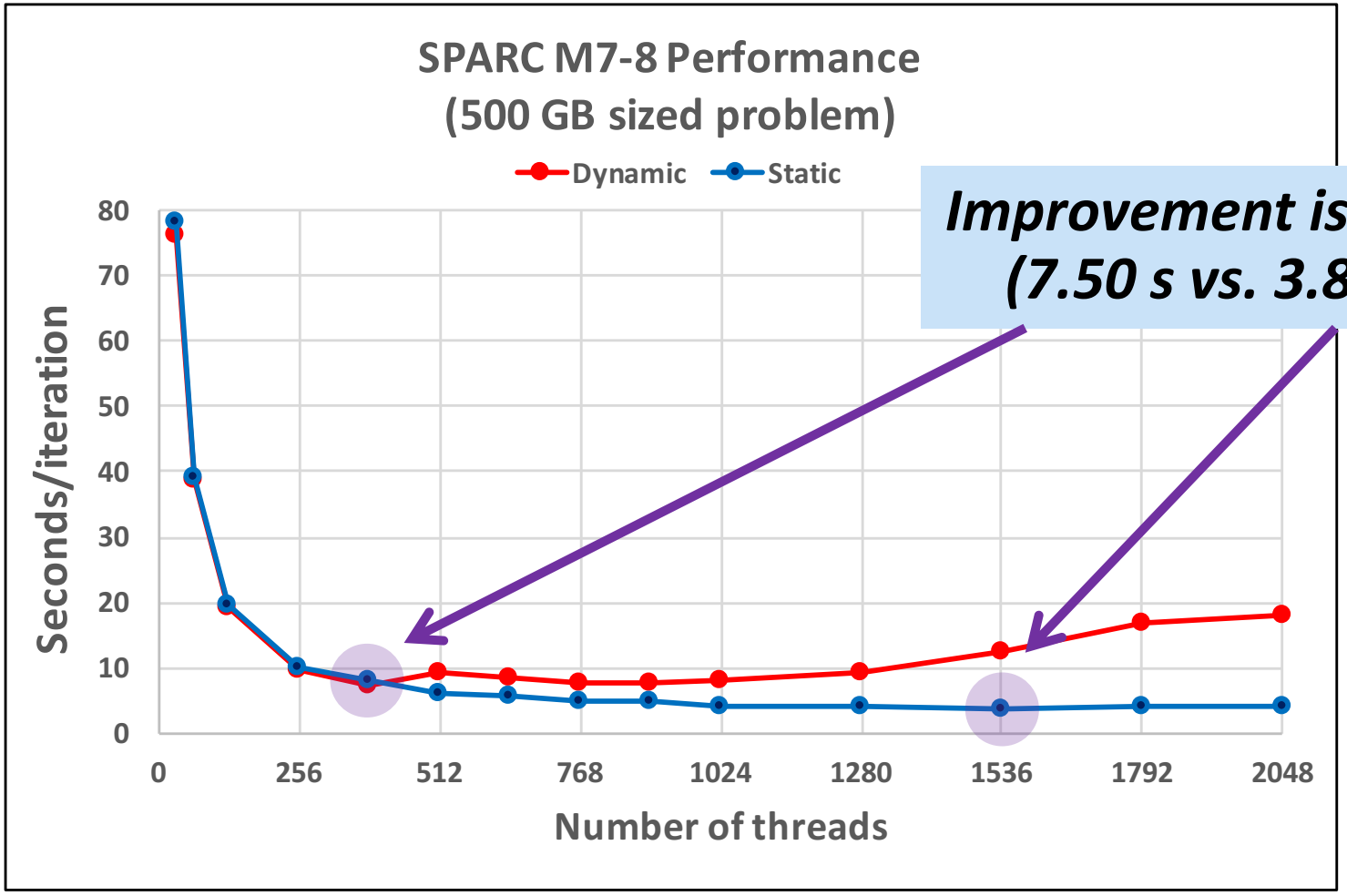
”Dynamic” – Uses the while-loop with atomic add

”Static” – Uses the for-loop

Performance Results/1



Performance Results/2



Cliffhanger

This is a great candidate for the taskloop construct

To Be Continued!

Summary

Locks And Atomic Operations Are Like Gold Coins

You Like To Have Them, But Not Use Them

Thank You And ..... Stay Tuned !

ruud.vanderpas@oracle.com