Report from Pthreads lab

Mikael Åsberg & Valentine Svensson

1.

We added argument `(void *)t` to `pthread_create`, then in `*HelloWorld` we made a `long t` that we assigned to `(long) arg`, which we then printed in a a formatted string.

To make the `hello2` print different values for different threads we modified line 57 so that `thread_data_array[t].message = messages[t]`.

2.

Since we are setting the `GlobalData[tid] = -tid` in each thread, we will have a race condition causing the sum in that particular thread to have a value depending on which threads have been run before. And we can not know the order of the threads.

3.

If we set the attribute to `PTHREAD_CREATE_DETACHED` and remove the `pthread_join` function the loop over the threads will print a message indicating the threads are finished even though they are not. However, they will finish when we get to `pthread_exit(NULL)`, and join.

If we also remove `pthread_exit(NULL)` the threads will just abort when the program finishes.

Also, `pthread_join` will not work unless `PTHREAD_CREATE_JOINABLE` is set.

4.

If we don't lock the `mutexsum` (but still use `-O3`) each thread will locally sum the 100 000, but the threads starting after some thread have managed to finish will read e.g. 100 000, and sum from that as a starting value of `sum`.

When we also remove `-O3` from the compiler the value of `sum` is not cached, and will be read and written in each (parallel) iteration. So that some results from the summation will be overwritten by a thread slighly out of sync. Giving us results like 115 403, where the ~15 403 corresponds to how much later the non-first started.

5.

If we remove the barrier then the printouts of `"Hello World!"` and `"Bye Bye World!"` will be interleaved.

The compiler optimizes in such a way that it assumes that variable `state` is not changed outside of a threads context (even though other threads can change this value). Hence, the first 7 threads will spinlock forever in the while- loop. Setting the variable `state` as `volatile` will force the compiler to put code that will do an actual read of the memory location of variable `state`.

6.

- Our pthreaded application `mypi.c` has the following speed-up (average values):

```
Original pi.c: 883551 micro-secs

mypi.c, 5 threads: 179428 micro-secs
mypi.c, 10 threads: 157552 micro-secs
mypi.c, 32 threads: 156199 micro-secs
mypi.c, 64 threads: 168429 micro-secs
mypi.c, 100 threads: 171807 micro-secs
```

Hence, there seems to be an optimal speed-up around 10–32 threads.

- Our parallel code for matrix multiplication, `matmul2.c` gives the following speed-up for 1000x1000 (average values):

```
Original matmul.c: 14.27 s

matmul2.c, 5 threads, 5.40 s
matmul2.c, 10 threads, 4.45 s
matmul2.c, 20 threads, 5.53 s
```

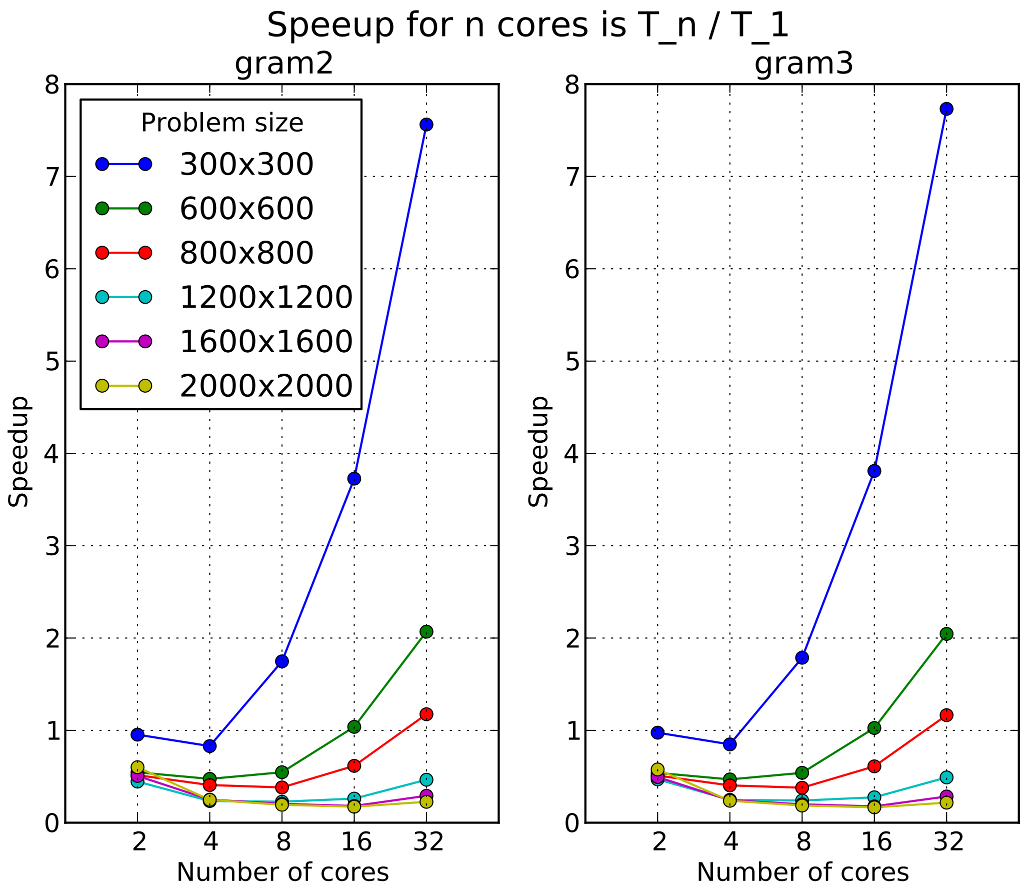
The maximum speed-up factor seems to be around 3 (for 10 threads).

For 100x100 matrices we never get faster performance with 5, 10 or 20 threads. Hence, perhaps 500x500 matrices and more gives a speed-up.

- When we increase the problem size, non-synchronize check pointing is only better than global barrier when the number of threads matches the problem size. In the below measurements, when the problem size is larger than ~1001x1001, more number of threads is required in order for non-synchronize check pointing to be better than global barrier, i.e., 16 threads are not enough wrt the problem size.

```
32 threads, Problem size: 256 x 256

Global barrier: 0.536643 sek
Non-synchronize check pointing: 0.247219 sek

16 threads, Problem size: 256 x 256

Global barrier: 0.759400 sek
Non-synchronize check pointing: 0.413608 sek

---------------------------------------------

32 threads, Problem size: 501 x 501

Global barrier: 2.900717 sek
Non-synchronize check pointing: 1.581953 sek

16 threads, Problem size: 501 x 501

Global barrier: 3.535042 sek
Non-synchronize check pointing (501 501): 2.164934 sek

---------------------------------------------

32 threads, Problem size: 1001 x 1001

Global barrier: 18.386948 sek
Non-synchronize check pointing: 12.324078 sek

16 threads, Problem size: 1001 x 1001

Global barrier: 21.372936 sek
Non-synchronize check pointing: 20.464841 sek

---------------------------------------------

32 threads, Problem size: 1501 x 1501

Global barrier: 42.394640 sek
Non-synchronize check pointing: 41.047683 sek

16 threads, Problem size: 1501 x 1501

Global barrier: 59.834858 sek
Non-synchronize check pointing: 75.415063 sek
```

- Plot for speedups of various problem sizes.



Speeup for n cores is T_n / T_1
gram2
gram3
Problem size
300x300
600x600
800x800
1200x1200
1600x1600
2000x2000
Speedup
Number of cores
0
1
2
3
4
5
6
7
8
2
4
8
16
32