HPC Homework 3

Austin McDowell

OpenMP warm-up

a)

For the first loop thread 0 takes steps $\{i = 1, 2, ..., (n-1)/2\}$ while thread 1 takes the remaining steps $\{i = (n+1)/2, (n+3)/2, ..., n-1\}$. The total time for thread 0 to execute its chunk is $t_0 = \sum_{i=1}^{(n-1)/2} i$ miliseconds and the time for thread 1 to execute its chunk is $t_1 = \sum_{i=(n+1)/2}^{n-1} i$ miliseconds. Thread 0 will be waiting on thread 1 for $t_1 - t_0$ miliseconds.

In the second loop thread 0 will take $T_0 = \sum_{i=1}^{(n-1)/2}(n-i)$ miliseconds to finish and thread 1 will take $T_1 = \sum_{(n+1)/2}^{n-1}(n-i)$ miliseconds. This time thread 1 will be waiting for thread 0 for $T_0 - T_1$ miliseconds.

b)

If we use `schedule(static,1)` then the steps are distributed to the threads in a chunk size of 1. In this case, thread 0 will execute all odd steps and thread 1 will do all even steps. The time for thread 0 would then be the sum of all odd numbers from 1 to n and time for thread 1 would be the sum of all even numbers from 2 to n. Thread 0 would wait for thread 1 for $n/2$ miliseconds in the first loop. In the second loop, thread 1 will wait for thread 0 for $n/2$ miliseconds.

c)

I would expect dynamic scheduling to improve the timing because the work will be given to threads as the threads become available. Dynamic scheduling would balance the computation among the threads and reduce the wait time when compared to static scheduling.

d)

We could use the clause `nowait` to eliminate the waiting time. In this case, the total time would be $T = 2\sum_{i=1}^{n-1} f(i)$ miliseconds for the threads to complete both loops.

Finding OpenMP bugs

Please see code for comments :)

Parallel Scan in OpenMP

Below is a plot of my timings for the serial and parallel scan problem for different thread numbers. I could only get my scan working for integer multiples of the thread number and my parallel scan takes longer than the serial scan which makes me think my code could have a bug even though my error is 0. My processor is `Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz` which has 4 cores and 8 threads per core. I timed the individual pieces of my code and it seems like the largest overhead is in the loop that adds the sum correction to the final result. I have tried parallelizing this loop too but it doesn't offer much speed up.

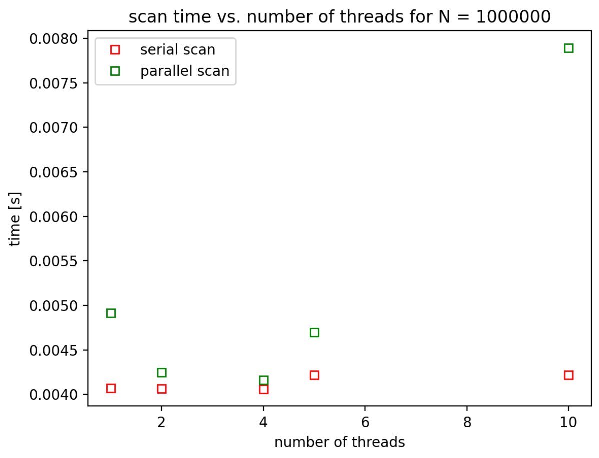

Figure 1: parallel and serial scan for N = 1000000

OpenMP & 2D Jacobi Modeling

Below are my timings for the 2D Jacobi solver with OpenMP for different N and number of threads. We see the parallel version is slower for lower N when the overhead for parallelization increases the compute time. For higher N we see an improvement as we increase the thread number.

N	threads	parallel Jacobi [s]	serial Jacobi [s]
10	4	0.067970	0.001872
50	4	0.069461	0.007386
100	4	0.072300	0.022999
1000	4	1.227166	3.162666
10	8	0.114649	0.001872
50	8	0.127204	0.007386
100	8	0.122812	0.022999
1000	8	1.262442	3.162666
10	16	0.203007	0.001872
50	16	0.20903	0.007386
100	16	0.217951	0.022999
1000	16	1.315189	3.162666