Zoheir El Houari

Matriculation: a12044027

Assignment 2: OpenMp Tasking

Introduction:

When we observe the construction of our score matrix M using dynamic programming approach, we can observe any entry ***(i, j)*** have data dependency with the entries ***(i – 1, j), (i, j – 1)*** and ***(i – 1, j – 1).*** This means that the current entry ***(i, j)*** depends on data that's in the same row and column. There for we cannot calculate the cells in the same row and column in parallel.

The solution was done by the wavefront (anti-diagonal) iteration of our score matrix M, with the data dependencies mentioned above, the parallelization approach is to calculate the entries which are in the same diagonal in parallel since there is no data dependency between them.

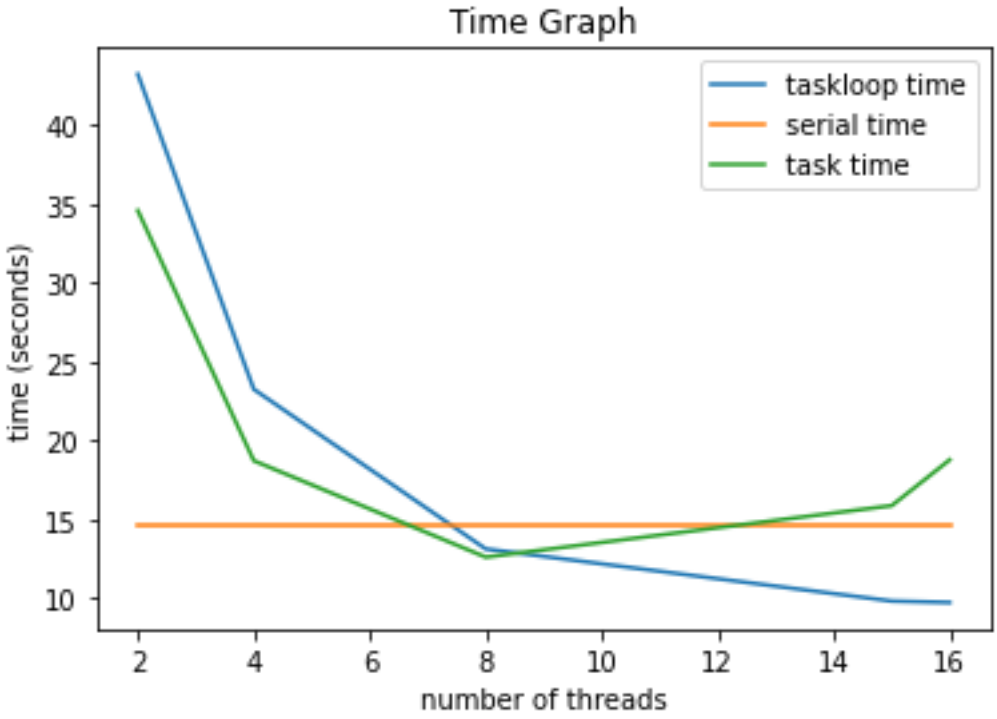
OpenMP Task version:

The ***llcs_parallel_tasks*** version is implemented by having the master thread invoke N tasks for each diagonal. Each thread iterates these tasks in block wise manner, having the block size equal to the number of available threads. The indices ***i*** and ***j*** are separately calculated for each thread outside the task region, these indices have the data scope of ***firstprivate*** to each thread since they their value is calculated outside of the task region.

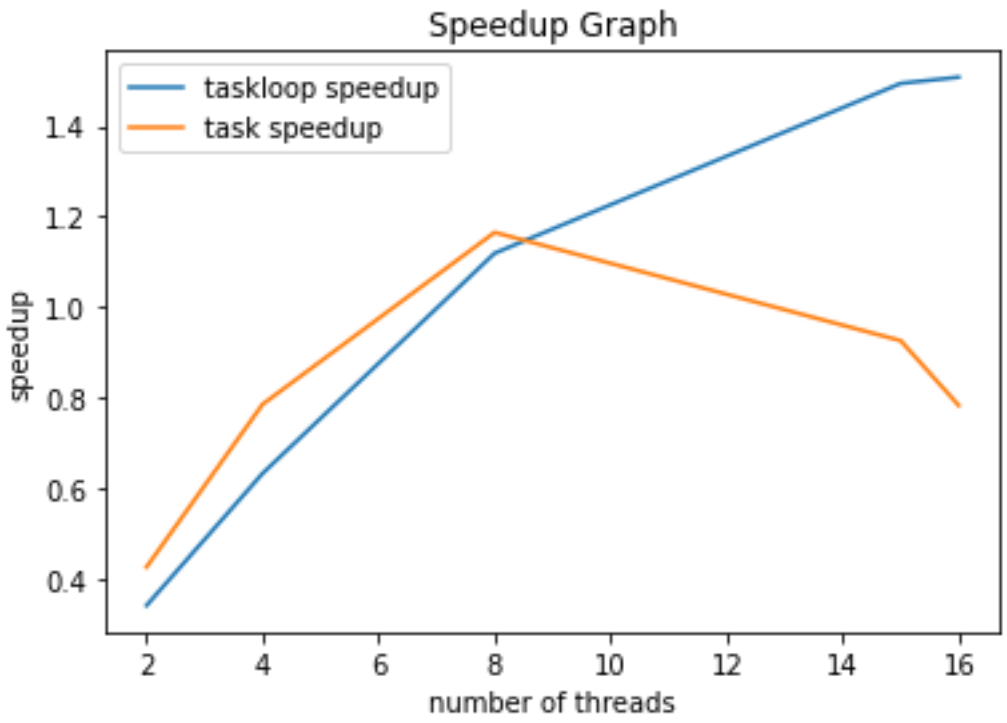
I used ***omp taskgroup*** in order to group the tasks generated for each diagonal computation, also for making sure that before iterating the next diagonal all previous tasks must be finished which preserve our data dependency. For the counter variable ***entries_visited***, I create another variable ***local_entries*** which would serve as a local counter for each block, then right after the task is finished, using ***omp atomic*** i then add the local counter to the ***entries_visited*** which makes sure that the race condition is avoided.

OpenMP Taskloop version:

The ***llcs_parallel_taskloop*** version was implemented in the same logic described in the ***omp tasks*** version, the approach has a slight difference, using the ***taskloop*** construct, for each diagonal iteration, I create the number of tasks equal to the length of the current diagonal. For the counter variable ***entries_visited***, I sum it with the value length of diagonal outside the for loop in order to avoid data race. Also Due to the high traffic on the Alma server, I was only able to test my solution with max 16 threads.



We can see clearly that the both version perform better as we increase the number of threads, however the explicit task version's performance decrease when we increase the number of threads past 8. I couldn't explain this behavior since I couldn't really debug, and I would like to maybe get an explanation when we go through the assignment in the lecture.



I didn't achieve the required speedup of 9x, which I think is due to my poor implementation of the wavefront iteration method since I didn't take into consideration the runtime of my wavefront algorithm without the OpenMP directives. The runtime was 74.286647 seconds without the directives.

My best speedup was 1.5x that achieved by the taskloop version using 16 threads.