**Part1.1 Output**

Printing 'Hello world!' using omp parallel and omp sections

ellH Woorl!d

Printing 'Hello world!' using omp parallel for

llHoW oreld!

**Part 1.2**

It can be seen for all three functions that the OpenMP functions are slower than the regular baseline functions. This is because when OpenMP distributes the work for each function to its threads, there is a lot of synchronization taking place so the values in the matrix are not corrupted.

**Part 1.3**

**Part 1.3b**

It can be seen that the versions where the private variables were moved to the shared variables the number of cycles are greater. This is because there is greater work being done so synchronization is maintained between the different threads and there is no values getting corrupted.

It can be seen that the parallel for at the outermost loop has the least cycles, after which the cycles start to increase as the parallel for is moved in front of the middle and the innermost for loop. Since more synchronization is required for the outer loops as each thread is not distributed an iteration, having the pragma for, for the inner loops creates more cycles.

**Task 2**

Explanation- For the SOR function, I used:

pragma omp parallel shared(change,data,length) private(i,j)

pragma omp for

I decided to use change, data and length as shared variables since all the threads will be contributing to these and hence synchronization is necessary, however I kept i and j private since the loop iterations don’t need to be synchronized between the threads.

Explanation- For the mmm\_block function, I used:

#pragma omp parallel shared(sum,length, a0, b0, c0, block\_size) private(i,j)

#pragma omp for

Again sum, length, a0, b0, c0 and block\_size are being used by all the threads and are edited by all the threads so synchronization is necessary, whereas for variables i & j synchronization is not necessary as they are used for loop iterations.