**Assignment – 2**

**Name: Tejal Vijay Nijai**

**Student No.:19210412**

**Email Address: tejal.nijai2@mail.dcu.ie**

**Module:CA670**

**Programme: MCM1**

An assignment submitted to Dublin City University, School of Computing for module CA670 Concurrent Programming, 2019/2020. I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work. By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online I confirm that I have read and understood the DCU Academic Integrity and Plagiarism Policy (available at: https://www.dcu.ie/policies/policies.shtml)

Name: Tejal Vijay Nijai

Date:19/04/2020

**Problem Statement:** To develop an efficient large matrix multiplication algorithm in OpenMP.

**1. Design of the Program:**

My solution has implemented a block method for large multiplication of matrices which indicates the multiplication of different dimensions of the matrix ranging from 200 to 2000 with a dimension increasing at 200 steps. It compares the multiplication with OpenMP and without it and explains how OpenMP makes the program more effective.

Program has the method randomSquareMatrix that is being used to construct matrix with random values in it. The rand() function has been used which will repeatedly develop the same sequence every time the program runs. The statement: rand() % MAX VAL + MIN VAL produces values ranging from 1 to 10 as MAX VAL = 10 and MIN VAL=1 defined in the program. Both functions take dimension as an input and build matrix with the number of rows and columns specified in the parameters. The pragma attribute(**#pragma omp parallel for**) has been used to speed up the process of forming matrix elements which parallelize the process across threads. To estimate the computing time elapsed, the gettimeofday() function was quantified, which measures the time difference between before and after multiplication code executes. Elapsed time(in seconds) is the measure has been used to compare the both program performance to determine which one is efficient. Program has blockMatrixMultiply and parallelBlockMatrixMultiply functions which depict block matrix multiplication of matrices in a normal way and using OpenMP respectively. The matrixMultiplicationTest function is written in the program to get the matrix multiplication time with and without using OpenMP, and the percentage quality increase using OpenMP.

I have referred [1][2] for the design of my program.

**2.Efficieny of the program:**

**Block Matrix Multiplication:**

Block matrix multiplication is a technique in which a large matrix is fragmented into blocks or submatrices. On these submatrices, matrix multiplication is performed to calculate the result matrix elements which imply the summation of all of these submatrices which is basically the outcome of matrix multiplication of two large matrices.

In this strategy, blocks of the said block\_size size will be loaded into the fastest cache i.e. L1 cache to minimize memory traffic during calculation. For the optimum result, the block\_size can be taken as the square root of L1 cache size. Based on L1 cache size (256 KB) of the device, the optimized result was given by block size = 16.So, block\_size has been set to 16 in the program.

Various pragma attribute clauses such as shared, private, and schedule clause have been used to optimize multiplication using OpenMP. The shared clause declares the variables to be shared by all available threads in the list. The private clause denotes the variables to be private for each thread within the list. The static clause specifies how loop iterations are broken down among the group threads. It divides the loop iterations into chunk size bits and then assigns them statically to threads. If chunk size is not mentioned, the iterations are divided contiguously between the threads evenly (where possible).

**#pragma omp parallel shared(matrixA, matrixB, matrixC, size, chunk) private(i, j, k, jj, kk, tid, tmp):** Having all matrices, matrix size and chunk size shared, and all loop counters private, makes the computation move quicker as all threads operate at the same time parallelly. Matrix A and B are the matrices to multiply and result is saved in matrix C. Loop variables have been kept private so that each thread will work on their individual copy of variables independently.

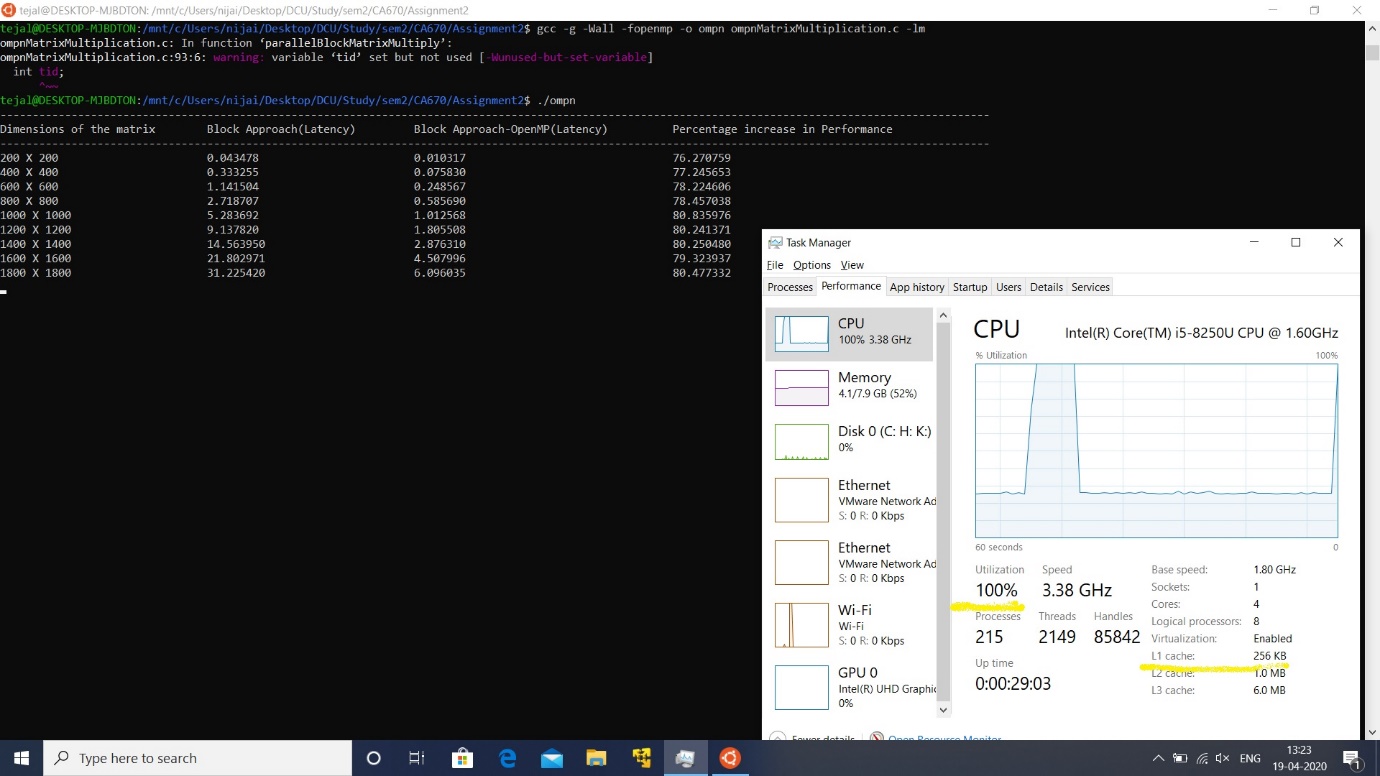
**#pragma omp for schedule (static, chunk):** It defines static scheduling type with default chunk size as 1.It equally divides the loop iterations through all available threads.

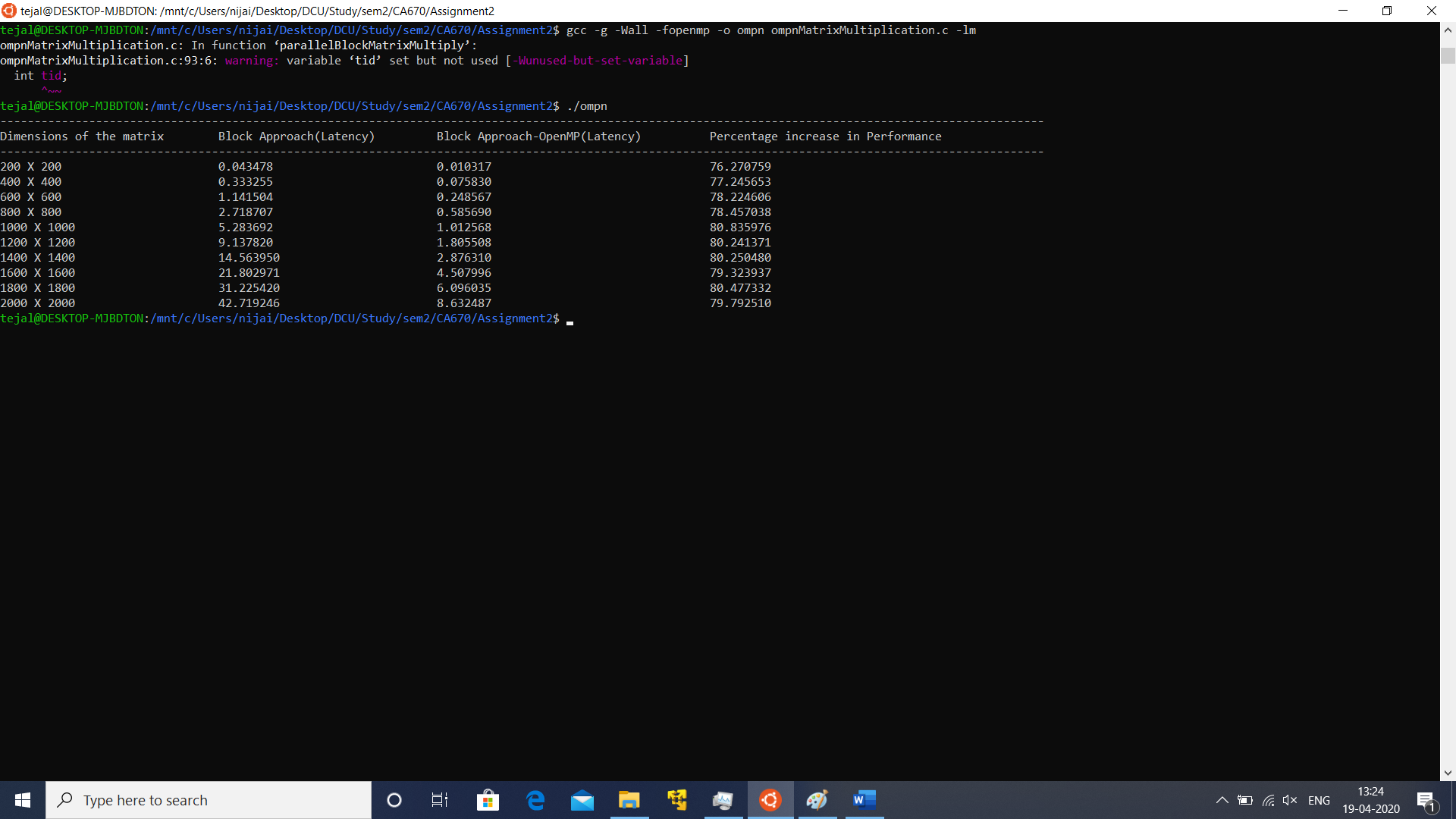
Thus, above directives given to compiler make the process more efficient than normal multiplication. Using OpenMP, it becomes possible to make reuse of local memory for repetitive addition and multiplicative operations of the small blocks and simultaneously work with all available threads. On an average, multiplication with OpenMP is giving around 80% of performance improvement for increasing size of matrices.

The graph below displays the time elapsed for both methods indicating the decrease in elapsed time for calculation using OpenMP:

The percentage efficiency increase graph for multiplication of normal block matrix multiplication and with OpenMP is as follows:

**3.Evidence of Program Output:**





**4.References:**

1.<https://github.com/roshanmadhushanka/Parallel-Matrix-Multiply> 2.<https://github.com/dmitrydonchenko/Block-Matrix-Multiplication-OpenMP/blob/master/block_matrix/Source.cpp>