# **CA670 Concurrent Programming**

|  |  |
| --- | --- |
| Name | Anubhav Gupta |
| Student no. | 19211171 |
| Programme | MSc. in Computing |
| Module Code | CA670 |
| Assignment title | Assignment 2 – Matrix Multiplication in OpenMP |
| Submission Date | 17th April, 2020 |
| Module Coordinator | David Sinclair |

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. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. 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.

I have read and understood the referencing guidelines found recommended in the assignment guidelines.

Name: Anubhav Gupta Date: 17thApril, 2020

EFFICIENT MATRIX MULTIPLICATION IN OpenMP

**Introduction**

OpenMP better stands for **Open Multi-processing** is used to convert a sequential program into multi-threaded, shared memory program. It supports C / C++ or FORTRAN language to run codes in parallel threads.

It has a header file #include <omp.h> used in C/C++. By using this we can create number of threads which act as a processing main in our system. All the threads created run parallelly to achieve an output and to speed it. OpenMP is easily accessible over Linux, Windows or MacOS. It is started with #pragma omp parallel. It parallel world system in which all the threads that are initialized are running in a parallel world. This is a basic example of any OpenMP program

#include <omp.h>

#include <stdio.h>

Int main(int argc, char \*argv[]){

/\* program logic \*/

#pragma omp parallel {

}

return 0;

}

This assignment is based on OpenMP, I have used OpenMP to solve large matrix Multiplications, I have used C language for this whole assignment. A maximum of 1000 \* 1000 square matrices have been used and random function is used to generate matrices also time series have been used to get the average time for each calculation as output.

**Code**

In my code I have defined dimension of the matrices to be large to be 1000x1000 and the number of iterations to be provided by the user. I have used OpenMP in creating random matrices and then used in multiplying the matrices. I have provided 50 threads in the matrix multiplication. I

#pragma omp parallel shared(matrix3) private(i, j, k, l, anu, gupta) num\_threads(50)

I have used rand() to generate the random numbers in a particular matrix and seeding it with time to generate new combinations every time the program runs

srand( time(0)+clock()+rand );

    #pragma omp parallel for

    for(int i=0; i<dimension; i++){

        for(int j=0; j<dimension; j++){

            matrix[i][j] = rand() % MAX\_VALUE + MIN\_VALUE;

For efficiency I used gettimeofday(), this will retrieve the total time it will take for the for loop to run.

gettimeofday(&t1, 0);

    double elapsed = (t1.tv\_sec-t0.tv\_sec) \* 1.0f + (t1.tv\_usec - t0.tv\_usec) / 1000000.0f;

Parallelized for loops for fast computation of matrix multiplication

double MatrixMultiply(TYPE\*\* matrix1, TYPE\*\* matrix2, TYPE\*\* matrix3, int dimension){

    int i, j, k, l, anu;

    TYPE gupta;

    struct timeval t0, t1;

    gettimeofday(&t0, 0);

    convert(matrix1, matrix2, dimension);

    #pragma omp parallel shared(matrix3) private(i, j, k, l, anu, gupta) num\_threads(50)

    {

        #pragma omp for schedule(static)

        for(i=0; i<dimension; i++){

            l = i \* dimension;

            for(j=0; j<dimension; j++){

                anu = j \* dimension;

                gupta = 0;

                for(k=0; k<dimension; k++){

                    gupta += flatA[l + k] \* flatB[anu + k];

                }

                matrix3[i][j] = gupta;

            }

        }

    }

    gettimeofday(&t1, 0);

    double elapsed = (t1.tv\_sec-t0.tv\_sec) \* 1.0f + (t1.tv\_usec - t0.tv\_usec) / 1000000.0f;

**Efficiency**

To increase the efficiency of the program I have used openmp for initiating random matrices that will do the loop parallelization and will initiate matrices more efficiently. Also I have created 2 containers that will convert the data stored in heap memory to the stack memory. A total of 50 threads have been used to efficiently calculate the matrix multiplication. As it will efficiently calculate the matrix multiplication and we will be getting total time taken for the loop to be executed.

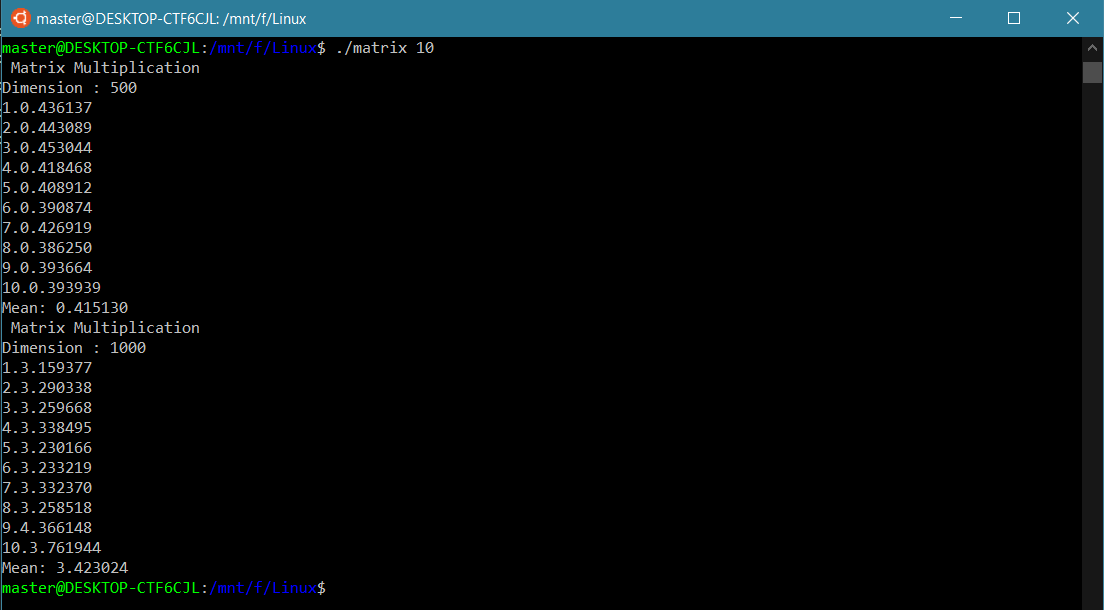


Figure 1 Execution of the OpenMP to run and multiply the matrices

I have used a total of 1000x1000 dimension for matrix but executed it in 2 parts one with 500x500 and then 1000x1000. As shown in figure 1 the time taken in 500x500 gave avg time of about 0.415 sec to complete the computation while in 1000x1000 took 3.4 sec to complete the computation.

To test the efficiency of this OpenMP program I tried to compare it with the sequential program of multiplying matrices and to test created another program that doesn’t run any parallelizing threads that will calculate the matrix multiplication. I implemented the same program but didn’t used any OpenMP loop to run any of the matrix multiplication. This program also created random matrix with seeding and it was also created with 1000x1000 dimensions implemented in two groups one with 500x500 and other 1000x1000 dimensions. As you can see in figure 2 the mean time taken to compute 500x500 dimension matrix multiplication was 0.8 seconds and for 1000x1000 matrix it came out to be over 11 seconds.

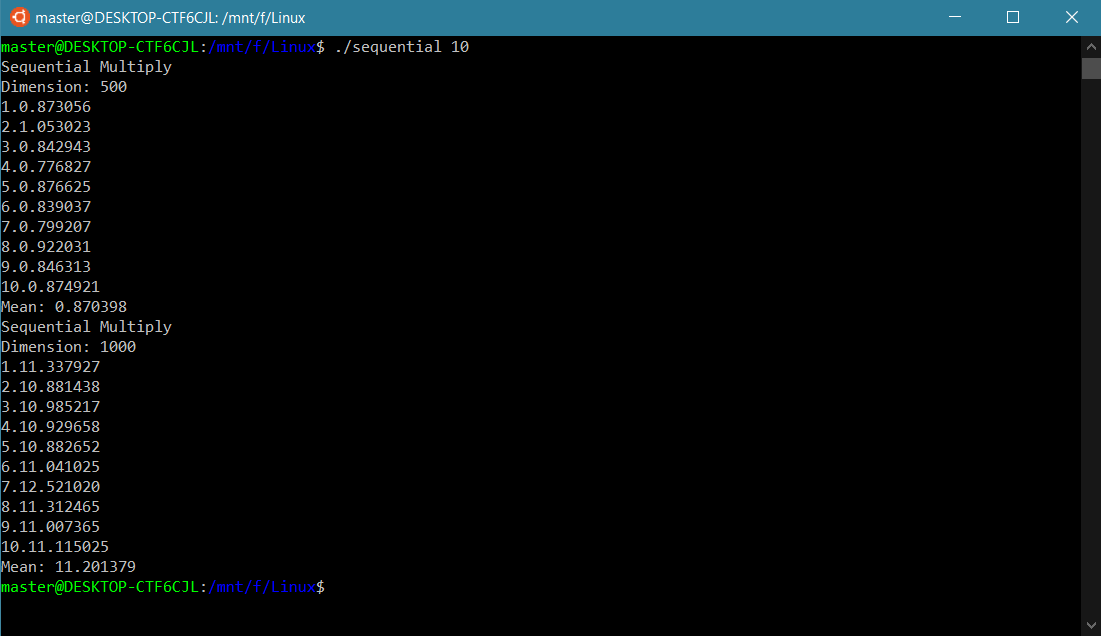


Figure 2 Sequential Multiplication of Matrix

|  |  |  |
| --- | --- | --- |
| Dimensions | Matrix Multiplication Using OpenMP | Matrix Multiplication without OpenMP |
| 500 | 0.415sec | 0.870sec |
| 1000 | 3.42 sec | 11.201sec |

As you can see in the table above the average time taken in 500x500 dimension matrix matrix multiplication without OpenMP took approximately double time than program with OpenMP took and in 1000x1000 matrix multiplication, Sequential multiplication took approximately 4 times than the parallelizing threading-based program took.

Sequential Multiplication was working fine when the dimension of matrix was not much larger and as we increase the size of the matrix the sequential multiplication method tend to get slower and slower in computation. This result tells that for larger number of matrices parallelizing threads play a big role and it also increases the efficiency of the computation. In the 500x500 multiplication the traditional method took double to what parallel loops took, in the 1000x1000 matrix multiplication the difference went up to 4 times. It will to 8 times in the next dimension and will increase the same way in further steps. This comparison successfully tells that the Matrix Multiplication written with parallel for loops using OpenMP is more efficient in multiplying large Matrices.

**References**

[1] <https://stackoverflow.com/questions/35613298/implicit-declaration-of-functions-srand-rand-and-system>

[2] <https://www.tutorialspoint.com/what-is-openmp>

[3] <https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/>

[4] <https://stackoverflow.com/questions/39710442/clock-gettime-vs-gettimeofday-for-measuring-openmp-execution-time>

[5] <https://stackoverflow.com/questions/10409032/why-am-i-getting-undefined-reference-to-sqrt-error-even-though-i-include-math>

[6] <https://stackoverflow.com/questions/19641597/what-is-segmentation-fault-core-dumped>

[7] <https://scriptverse.academy/tutorials/c-program-generate-random-matrix.html>

[8] <https://computing.llnl.gov/tutorials/parallel_comp/>

[9] <https://en.cppreference.com/w/c/language/typedef>

[10] <https://medium.com/tech-vision/parallel-matrix-multiplication-c-parallel-processing-5e3aadb36f27>