RYERSON UNIVERSITY

Faculty of Electrical and Computer Engineering

Department of Electrical and Computer Engineering

Program: Computer Engineering

Course Number	**EE 8218 – 011**
Section Number	**01**
Course Title	**Parallel Computing**
Semester/Year	Fall 2015

Instructor	**Nagi Mekhiel**

ASSIGNMENT No. 1

Assignment Title	**Introduction to OpenMP**

Submission Date	**September 29, 2015**
Due Date	**September 29, 2015**

Student Name	**Ismail Sheikh**
Student ID	Xxxx89867
Signature*	**M.Ismail**

(Note: Remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/policies/pol60.pdf.*

Objective:

This lab is about installing and getting familiarize with the open MP extension of Microsoft Visual Studio software. Similarly, visualize the performance improvement through parallel program with increased number of threads.

Introduction:

Open MP is an API for multi-platform shared-Memory parallel programming in C/C++ [1]. Open MP is heavily used in many real life applications to introduce multiple threading in the software algorithms. Multiple threads execution provide the ability to the application to divide the processor workload equally in all thread and execute all at same time. This eventually improves the overall performance of the application. In this report, the performance of an application using parallel programming as well as standard programming with be thoroughly discussed.

Experiment:

In order to visualize and compare the performance of a multiple processing, a Microsoft Visual Studio application was implemented using Open MP API. In this application, two square matrices with the size of 1000, 2000, and 3000, 4000 respectively initialized. Further, both square matrices where multiplied together using sequential algorithm as well as parallel computing algorithm.

The result of the experiment is as follow:

Size of Matrices	Sequential Algorithm Time	Parallel Algorithm Time
1000 x 1000	2000 milliseconds	0 milliseconds
2000 x 2000	13,000 milliseconds	7,000 milliseconds
3000 x 3000	50,000 milliseconds	18,000 milliseconds
4000 x 4000	85,000 milliseconds	40,000 milliseconds

Table 1 above represents the size of two square matrices used for the application and the time the system took to multiply those two matrices using sequential algorithm as well as parallel algorithm. It is very clear from the above table that the performance of parallel algorithms is much better than sequential algorithm.

Further, as the size of the matrices becomes bigger and bigger there are more values to execute and sequential algorithm waits for the first commands to complete before executing the next command which results in a really slow response. Similarly, the parallel algorithm divide the processor workload into equal thread and execute multiple application at the same time (depending on running thread) and results in much faster response.

Hence, parallel computing performance is much more efficient and fast as compare to sequential algorithms.

Appendix A:

Appendix A represents the screenshots of Microsoft Visual Studio software by running the application.

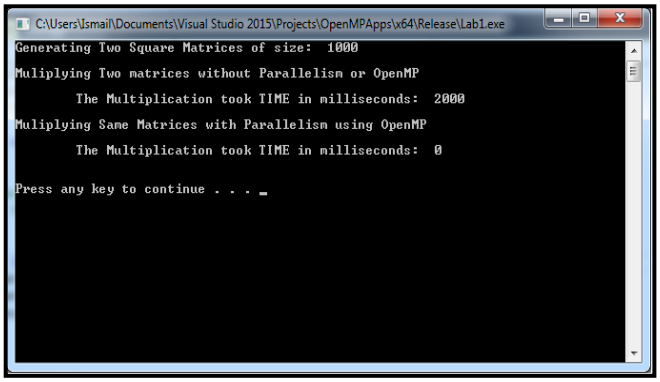

Figure 1: Result of Matrix 1000 x 1000

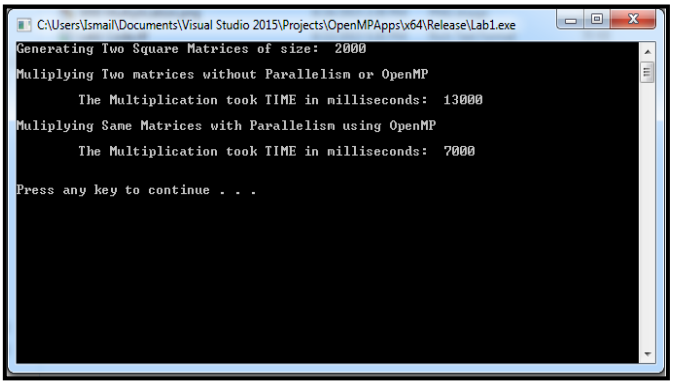

Figure 2: Result of Matrix 2000 x 2000

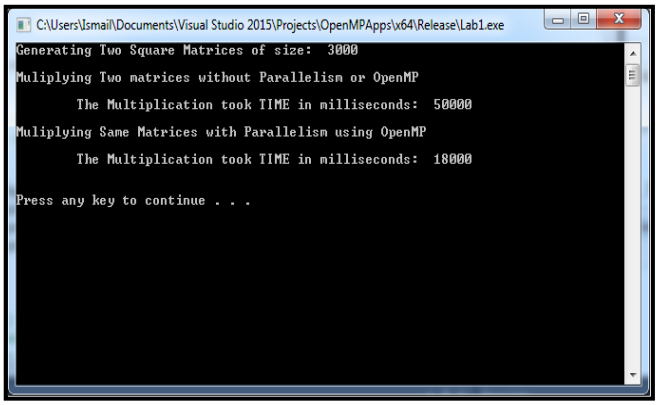

Figure 3: Result of Matrix 3000 x 3000

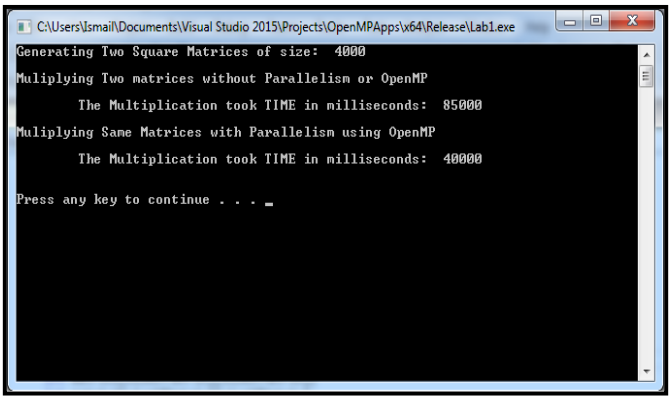

Figure 4: Result of Matrix 4000 x 4000

Appendix B:

Appendix B represent the actual code which was used the application implementation

```
/*
*Lab1:           Multiply two matrices using OpenMP
*Student Name:          Ismail Sheikh
*Course Name:    EE8128 Parallel Computing
*Instructor Name: Dr. Nagi Mekhiel

*/

#include "stdafx.h"
#include "iostream"
#include <omp.h>
#include <ctime>

using namespace std;
/*

*Instance Variables of Matrix1, Matrix2 and Resultant Matrix
*Also, the length of the square matrix

*/
const int sizeOfMatrix = 1000;
int matrix1[sizeOfMatrix][sizeOfMatrix];
int matrix2[sizeOfMatrix][sizeOfMatrix];
int result [sizeOfMatrix][sizeOfMatrix];

/*

*Void Initailize()
*THis method initailizes both matrices (MATRIX1 and MATRIX2) with Math random values 0-9
*Update the values of both matrices, Matrix1 and Matrix2

*/
void initialize() {
	for (int i = 0; i < sizeOfMatrix; i++) {
		for (int j = 0; j < sizeOfMatrix; j++) {
			matrix1[i][j] = rand() % 100;
			matrix2[i][j] = rand() % 100;
		}
	}
}

/*

*Void MultiplyMatrix
*THis method multiplies both matrices (MATRIX1 and MATRIX2) without using parallelism or OpenMP
*Store the result of multiplication into the resulant MATRIX.

*/

void multiplyMatrix() {
	for (int i = 0; i < sizeOfMatrix; i++) {
		for (int j = 0; j < sizeOfMatrix; j++) {
			for (int k = 0; k < sizeOfMatrix; k++) {
				result[i][j] += matrix1[i][k] * matrix2[k][j];
			}
		}
	}
```

```
}
/*

*Void DisplayMatrix
*optional method available to display any matrix
*Requires a matrix of specific size
*Display the whole matrix

*/

void displayMatrix(int matrixToDisplay[sizeOfMatrix][sizeOfMatrix]) {
	for (int i = 0; i < sizeOfMatrix; i++) {
		for (int j = 0; j < sizeOfMatrix; j++) {
			std::cout << matrixToDisplay[i][j] << " ";
		}
		std::cout << " \n" << endl;
	}
}

/*

*Void MultiplyMatrix
*THis method multiplies both matrices (MATRIX1 and MATRIX2) using parallelism or OpenMP
*Store the result of multiplication into the resulant MATRIX.

*/
void multiplyParallel() {
	omp_set_num_threads(4);
#pragma omp parallel for

	for (int i = 0; i < sizeOfMatrix; i++) {
		for (int j = 0; j < sizeOfMatrix; j++) {
			for (int k = 0; k < sizeOfMatrix; k++) {
				result[i][j] += matrix1[i][k] * matrix2[k][j];
			}
		}
	}

}
/*
*Main of the program which is executed on RUNTIME
*/

int main(int argc, _TCHAR* argv[])
{

	//Commands for initialization of Matrices
	cout << "Generating Two Square Matrices of size:  " << sizeOfMatrix;
	initialize();


	//Commands for Multiplying matrices without Parallelism and time the algoritham
	cout << "\n\nMuliplying Two matrices without Parallelism or OpenMP" << endl;
	time_t startTime = time(0);
	multiplyMatrix();
	time_t endTime = time(0);
	double timeTaken = difftime(endTime, startTime)*1000;
	cout << "\n\tThe Multiplication took TIME in milliseconds:  " << timeTaken;
```

```
	//Commands for Multiplying matrices with Parallelism and time the algoritham
	cout << "\n\nMuliplying Same Matrices with Parallelism using OpenMP" << endl;
	time_t startTimeParallel = time(0);
	multiplyParallel();
	time_t endTimeParallel = time(0);
	double timeTakenParallel = difftime(endTimeParallel, startTimeParallel) * 1000;
	cout << "\n\tThe Multiplication took TIME in milliseconds:  " << timeTakenParallel;
	std::cout << "\n\n" << endl;

	system("pause");

return 0;

}
```