King Saud University.

College of Computer and Information Sciences.

Department of Computer Science.

CSC 453 introduction to parallel processing.

Instructor: Dr. Mohammed A. Al-Abdulkareem.

Fall 2020.

Self-Report

Student Name: Hassan Jaafar.

Student ID: 438105959.

Team Members:

Name	ID#
Majed Al-khraiji	438104708
Faisal Al-dhuwayhi	438102142
Ahmad Al-Mosallam	438103307
Fahad Al-deham	438100439

In this short report, I am going to write my own findings and important notes after working in the project with my team

Findings:

RUN#	p	M	T1	Tp	Speedup	Efficiency
1	1	100	0.0018	0.0018	1	1
2	2	100	0.0018	0.001	1.8	0.9
3	3	100	0.0018	0.001	1.8	0.6
4	4	100	0.0018	0.001	1.8	0.45
5	1	200	0.0114	0.0114	1	1
6	2	200	0.0114	0.0058	1.965517241	0.982758621
7	3	200	0.0114	0.0042	2.714285714	0.904761905
8	4	200	0.0114	0.0036	3.166666667	0.791666667

small sample

First of all, it is clear that the parallel code was very efficient for solving the matrix multiplication as it gives a good speedup and efficiency except when the matrix is small in size where whenever we increase the number of processing units (2,3,4) the speedup still the same and of course with decreasing in the efficiency since it is a small problem and all different processing units will result in almost the same time.

Also, I tried to increase the number of processing units from 4 to 8 on size 2000 and see if there is a difference or not and this what I found

```
MXM
 C/OpenMP version.

 Matrix multiplication tests.

 Number of processors available = 8
 Max number of threads = 8
 Working number of threads = 4

R8_MXM matrix multiplication timing.
 A(LxN) = B(LxM) * C(MxN).
 L = 2000
 M = 2000
 N = 2000
 Floating point OPS roughly -1179869184
 Elapsed time dT = 12.256000
 Rate = MegaOPS/dT = -96.268699

MXM:
 Normal end of execution.
```

```
MXM
 C/OpenMP version.

 Matrix multiplication tests.

 Number of processors available = 8
 Max number of threads = 8
 Working number of threads = 8

R8_MXM matrix multiplication timing.
 A(LxN) = B(LxM) * C(MxN).
 L = 2000
 M = 2000
 N = 2000
 Floating point OPS roughly -1179869184
 Elapsed time dT = 9.866000
 Rate = MegaOPS/dT = -119.589417

MXM:
 Normal end of execution.
```

As shown above running on 4 gives us a speedup of 3.13 and efficiency of 0.78 on the other hand running it on 8 gives us 3.89 with efficiency equal to 0.49 which is clearly implies that we don't utilize the power of increasing the processing units, because there is a limitation that can't be exceeded after a certain number of processors and this limitation could happen for some reasons for example, the nature of the algorithm or limited computational resources.

Potential enhancements:

There are different ways I want to try to see if it can improve the code:

1- using padding solution to make sure that the elements for one processing unit is in one cache for preventing false sharing

2- optimizing the memory could help significantly in increasing the parallel algorithm performance for example, Cache friendly algorithm implementation.