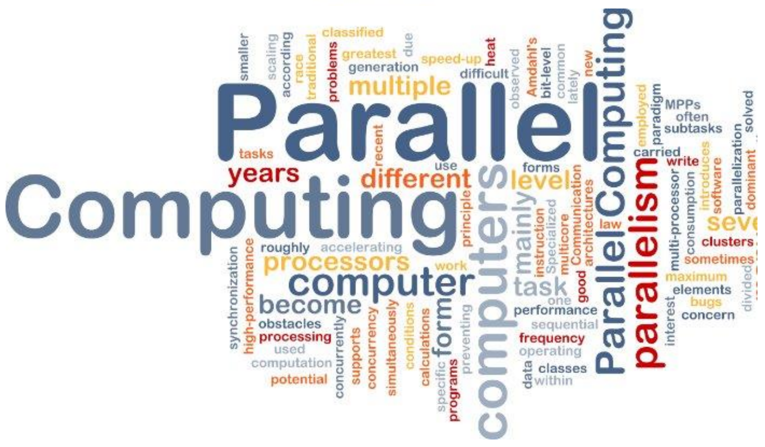

Hazem Alabiad

21403802

Project_1 Report

BBM442

Introduction:

- Most of the computers today benefits from parallel computing which rely on dividing the work among specific number of processes (real) and threads (virtual) to speed up the performance.

- A single core program or serial program works in serial so no data is being processed by more than a single thread at once. There is only one thread which manipulates the data part by part and in serial.

- At the other side, multithread program can make use of today's technology by letting various threads to manipulate our data in parallel by dividing and distributing the tasks between the threads and cores then by joining these threads we can have our job done most probably faster than a serial program.

- In some cases, we may have conflicts when many thread try to modify a shared variable and our job is to put an end to this kind of problems.

- Actually, many parallel programming specifies libraries and header files are available and easy to use. In this assignment we will use both <pthread.h> and <omp.h> and later we will compare the performance and see on which configuration <thread_number, problem_size> we can do the best.

*** Part 1:**

Pthread:

	N=200K	**N=500K**	**N=1M**	**N=2M**	**N=4M**
P=1	0.018026	0.053425	0.133651	0.330695	0.846021

P=2	0.012589	0.031546	0.071723	0.170103	0.435257
P=4	0.007645	0.027177	0.062539	0.146358	0.372468
P=8	0.010478	0.024720	0.060350	0.146388	0.374936

*** Part 2:**

<u>OpenMP</u>

	N=200K	**N=500K**	**N=1M**	**N=2M**	**N=4M**
P=1	0.017045	0.057736	0.127232	0.321988	0.847774
P=2	0.016718	0.032490	0.082634	0.205730	0.540851
P=4	0.008679	0.030077	0.058818	0.161613	0.402887
P=8	0.010394	0.022368	0.059718	0.149862	0.376564

* Part 3:

Here are the best configurations for both solutions:

1) For *<u>Pthread</u>*:

	N=200K	**N=500K**	**N=1M**	**N=2M**	**N=4M**
P=4	0.007384	0.026051	0.073749	0.152422	0.413968

2) For *<u>OpenMP</u>*:

	N=200K	**N=500K**	**N=1M**	**N=2M**	**N=4M**
P=8	0.010394	0.022368	0.059718	0.149862	0.376564

<u>Load Balance:</u>

- I tried to distribute the job fairly among threads by having a range (m) of values thereafter all threads will have equals ranges of values. That prevents chunking the range among threads randomly which may lead to what is called overloading when a thread has more tasks to finish than other threads. This would somehow guarantee that all threads will have the same data to process.

- In addition to that, I used "schedule(dynamic, 1)" which is provided by the header file <omp.h>. This statement helps us in dividing the tasks fairly among threads using a special algorithm and an internal work Queue.