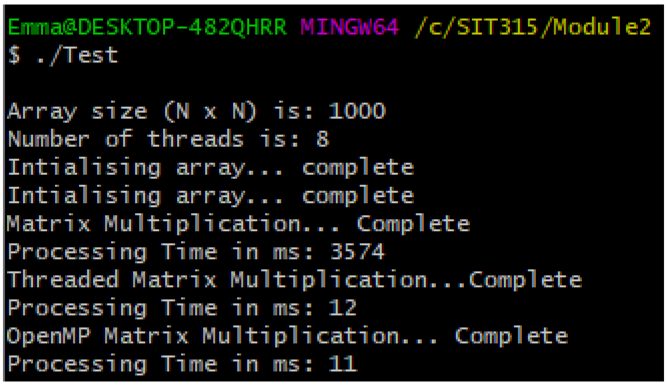
TaskM2.T1P

SIT315

- Link to code on GitHub:

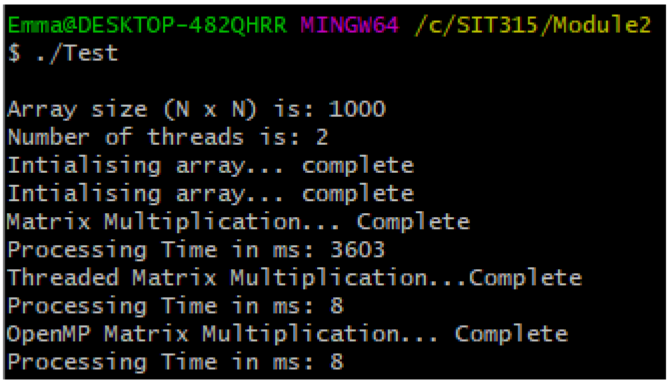
https://github.com/elpeacey/SIT315/blob/master/Module%202/MatrixMulti.cpp

- Execution times:



```
Emma@DESKTOP-482QHRR MINGW64 /c/SIT315/Module2
$ ./Test

Array size (N x N) is: 1000
Number of threads is: 8
Intialising array... complete
Intialising array... complete
Matrix Multiplication... Complete
Processing Time in ms: 3574
Threaded Matrix Multiplication...Complete
Processing Time in ms: 12
OpenMP Matrix Multiplication... Complete
Processing Time in ms: 11
```



```
Emma@DESKTOP-482QHRR MINGW64 /c/SIT315/Module2
$ ./Test

Array size (N x N) is: 1000
Number of threads is: 2
Intialising array... complete
Intialising array... complete
Matrix Multiplication... Complete
Processing Time in ms: 3603
Threaded Matrix Multiplication...Complete
Processing Time in ms: 8
OpenMP Matrix Multiplication... Complete
Processing Time in ms: 8
```

Array size: 1000x1000	2 Threads	4 Threads	8 Threads
Sequential	3603ms	3559ms	3574ms
pThread	8ms	8ms	12ms
OpenMP	8ms	7ms	11ms
Array size: 500x500			
Sequential	390ms	374ms	349ms
pThread	2ms	2ms	3ms
OpenMP	1ms	2ms	3ms

- Roadmap for parallelising code

Array values are independent therefore it is possible for the values to be calculated in parallel. I will update my code to implement this by running the first loop (bellow) in parallel

```
int num;
```

```
for (int i = 0; i < N; i++)
{
//code
}
```

- Evaluate:

  Both the OpenMP and the threaded matrix multiplication were noticeably faster than the sequential algorithm. I did expect to see more variability, particularly when I was using the same array size and number of thread. I was also surprised that the OpenMP was consistently slightly faster than the threaded matrix multiplication, especially when I tested the program with larger array sizes, I had assumed that they would both have the same processing time.