# SIT315 – Programming Paradigms

TaskM2.T1P: Parallel Matrix Multiplication

### Greg McIntyre

### 218356779

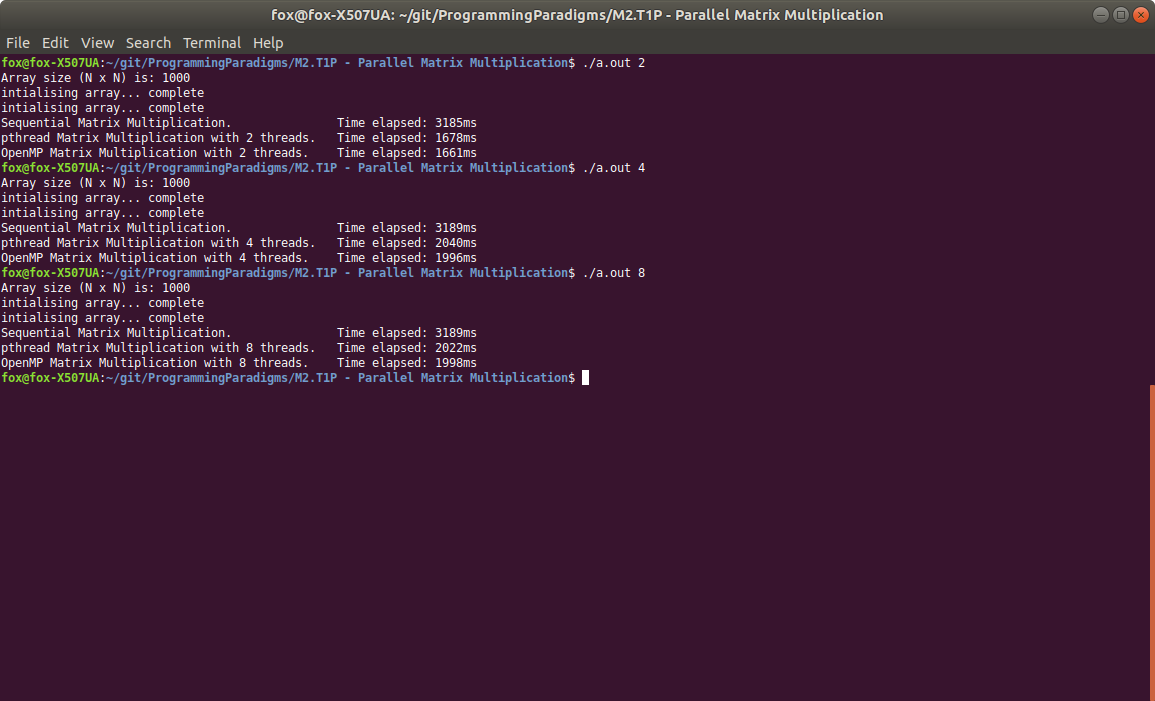
### 1. Implement a sequential matrix multiplication program in C or C++.

Written in C++.

<https://github.com/gregorymcintyre/ProgrammingParadigms/tree/master/M2.T1P%20-%20Parallel%20Matrix%20Multiplication>

or See Appendix B for a static version

### 2. At the end of the program, please print the execution time.



### 3. Once you have completed and tested the program, please review your code and develop a roadmap to parallelise your code.

To parallelise my code, I would like to have the following loop happen in parallel:

void SequentialMatrixMultiplication()

{

int value;

for (int i = 0; i < N; i++)

{

//code

I will implement it to perform the column functions independently, as results are not dependent on each other I should not need to implement mutex with this method, but I will assess as the program develops.

The array values are independent of each other and should be able to be implemented parallel. This would mean that all array values would be calculated concurrently, this should improve the performance of the program significantly.

### 4. Implement your parallel algorithm in C or C++ using pthread library

Same Git

<https://github.com/gregorymcintyre/ProgrammingParadigms/tree/master/M2.T1P%20-%20Parallel%20Matrix%20Multiplication>

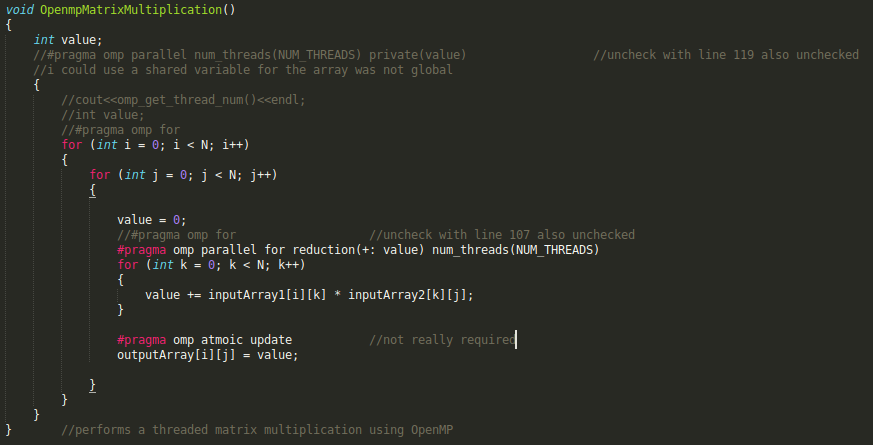
### 5. Evaluate the performance of your program

Program performed well, I expected to see a more variable result when adding more threads, but results show that the systems did not improve significantly in runtime when using more than 2 threads.

### 6. Modify your sequential program to use OpenMP to achieve parallelism

Same Git

<https://github.com/gregorymcintyre/ProgrammingParadigms/tree/master/M2.T1P%20-%20Parallel%20Matrix%20Multiplication>



I worked through a lot if iterations to make this satisfy the directions from MD, but I think this is a much more elaborate solution. It uses private ‘value’ and a ‘reduction’ as well as a ‘num\_threads’ and ‘atomic update’ to make the program more complex and demonstrate understanding.

I have also left commented code to show more understanding and demonstrate all the topics covered in the lecture. This reduced performance slightly (creating, destroying, reduction, atomic) so I have not updated the following comparisons.

### 7. Evaluate the performance of the OpenMP implementation vs pthread implementation vs the sequential program

The OpenMP Did not yield a decrease in runtime, it was much simpler to implement. But implementing a #pragma omp for yielded the no improvement on the pthread method.

### 8. Submit your task as detailed on the submission details section above to OnTrack

Submitted

## Appendix A: Raw Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **500** | **1000** | **1500** | **2000** |
| **SequentialMatrixMultiplication** | 334 | 3378 | 16949 | 36512 |
| **PthreadMatrixMultiplication (1)** | 341 | 3613 | 17104 | 37353 |
| **PthreadMatrixMultiplication (2)** | 172 | 1808 | 8584 | 19338 |
| **PthreadMatrixMultiplication (4)** | 173 | 2106 | 7652 | 20028 |
| **PthreadMatrixMultiplication (8)** | 172 | 2023 | 8087 | 20910 |
| **OpenMPMatrixMultiplication (1)** | 331 | 3544 | 17654 | 36649 |
| **OpenMPMatrixMultiplication (2)** | 193 | 1757 | 8385 | 20226 |
| **OpenMPMatrixMultiplication (4)** | 162 | 2110 | 7893 | 20362 |
| **OpenMPMatrixMultiplication (8)** | 169 | 2027 | 8000 | 21932 |