T2,
2020

# SIT315 - M2.T1P

## SEQUENTIAL VERSUS PARALLEL PROGRAMMING

CAMERON BOYD | STUDENT #218275923

# INTRODUCTION

This document will cover the following with regards to a C++ matrix multiplication program:

- Planning for parallelisation;
- Comparing the sequential program to a pthread multithreaded one;
- Comparing pthread multithreading with two and four threads; and
- Comparing the sequential program to a multithreaded pthread and OpenMP program.

# PLANNING PARALLELISATION

All subtasks and whether they should be completed in sequence or if they can be completed in parallel:

| TASK | SEQUENCE OR PARALLEL |
|---|---|
| Create matrices A and B | Sequence |
| Print matrix A | Sequence |
| Print matrix B | Sequence |
| Calculate matrix C | Parallel |
| Print matrix C | Sequence |
| Print calculation time | Sequence |

My reasoning for creating matrices A and B in sequence as opposed to in parallel is that this experiment will be focusing on the time taken to calculate matrix C. Additionally, the minute time saved by multithreading the creation of matrix A and B does not outweigh the overhead of implementation.

# PARALLELISATION WITH PTHREAD

To test whether multithreading improves the program's efficiency, I tested how long it takes both a sequential and multithreaded program to multiply two 10x10, 100x100, 1,000x1,000 and 2,000x2,000 matrices.

Each was run three times to get a reasonable average of the program's calculation time. The following results were obtained by using two threads for the multithreaded program.

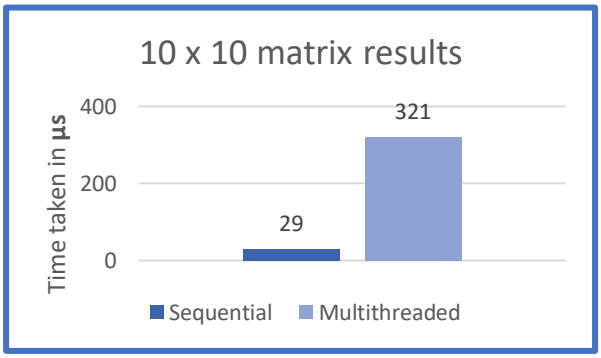


*Figure 2: 10 x 10 matrix results for single vs multithreaded*

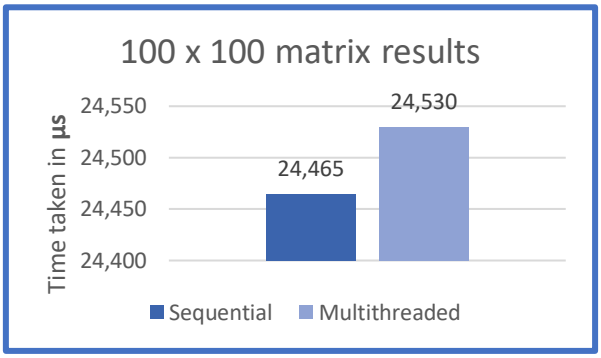


*Figure 1: 100 x 100 matrix results for single vs multithreaded*

Figure 1 indicates that the sequential program was significantly faster than the multithreaded one.

Figure 2 shows that the sequential was slightly faster in its execution than the multithreaded variant.

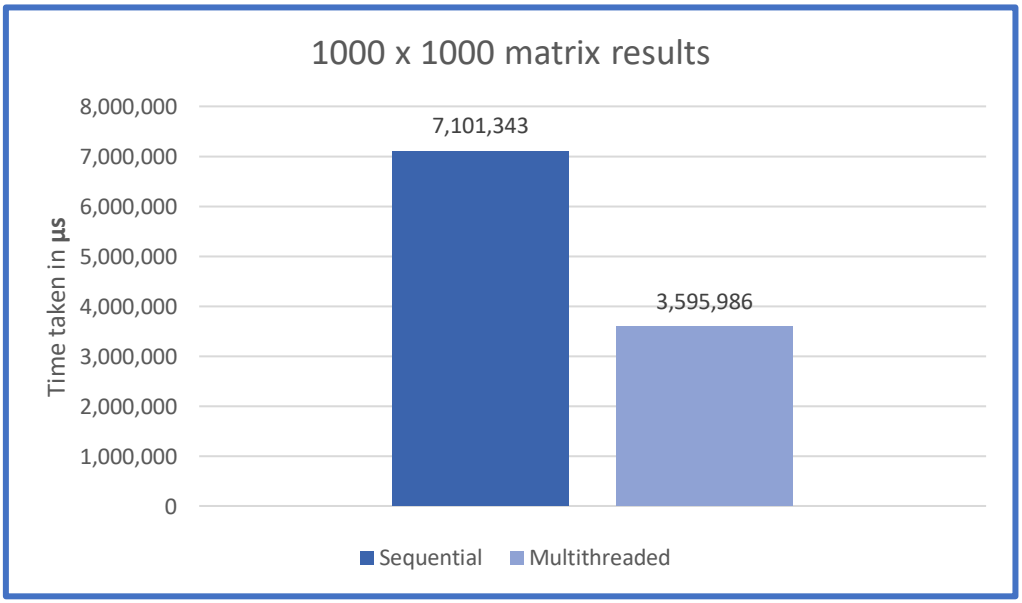


*Figure 3: 1000 x 1000 matrix results for single vs multithreaded*

Figure 3 shows that the sequential program took almost twice as long to execute than the multithreaded program.

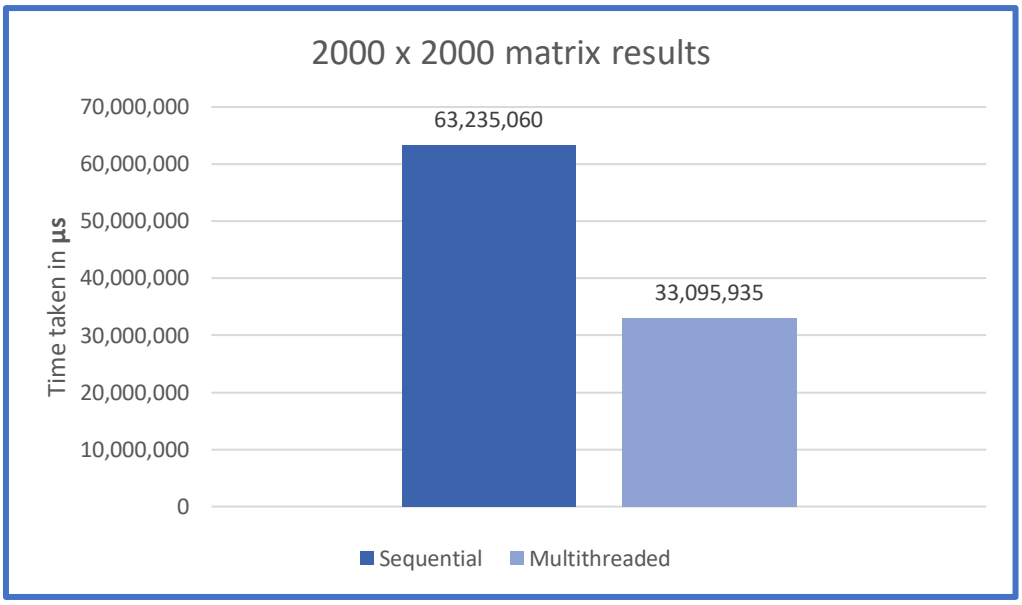


*Figure 4: 2000 x 2000 matrix results for single vs multithreaded*

Figure 4 shows that the sequential program, once again, took almost twice as long to execute than the multithreaded one, in this case an extra 30 seconds.

From these results, I can conclude that for small tasks, such as multiplying two 10x10 or 100x100 matrices, it is faster to complete the task sequentially. For larger tasks such as multiplying two 1,000x1,000 or 2,000x2,000 matrices, it is significantly faster to split the computation between multiple threads.

As the above results were obtained using two threads, I decided to test the speed at which four threads would complete this same task. As the 10x10 and 100x100 matrices weren't faster with two threads, I decided to focus solely on the 1,000x1,000 and 2,000x2,000 matrices for this next test.

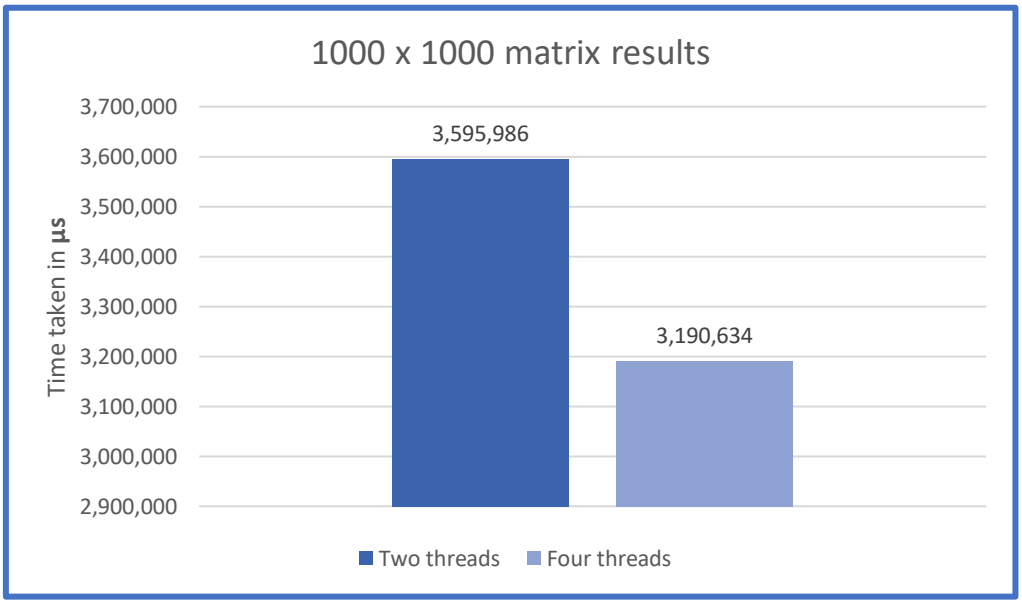


*Figure 5: 1000 x 1000 matrix results for two vs four threads*

Figure 5 shows that two threads took ~3.5 seconds, whereas four threads took ~3.1 seconds.

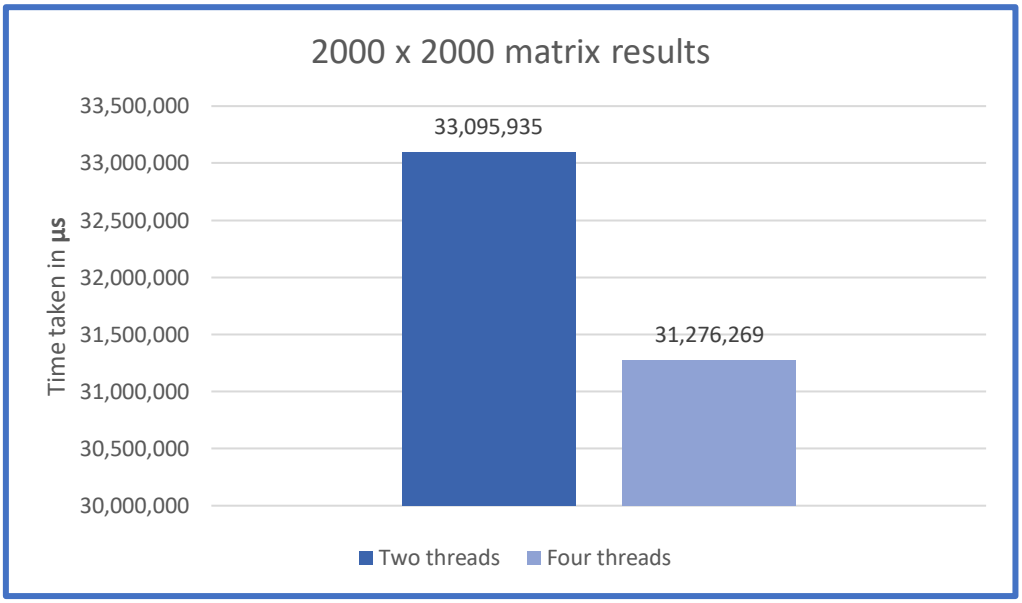


*Figure 6: 2000 x 2000 matrix results for two vs four threads*

Figure 6 shows that two threads took ~33.1 seconds, whereas four took ~31.2 seconds.

As the above results indicate, it is (unsurprisingly) faster to use four threads, rather than two. Although only a small decrease in time measured in this test; the bigger the matrix, the bigger the noticeable difference in computation time.

# PARALLELISATION WITH OPENMP

In this section, I will be comparing the difference in time taken to multiply two 10x10, 100x100, 1000x1000 and 2000x2000 matrices by a sequential, pthread and OpenMP program. Note, both multithreaded programs were tested using four threads.

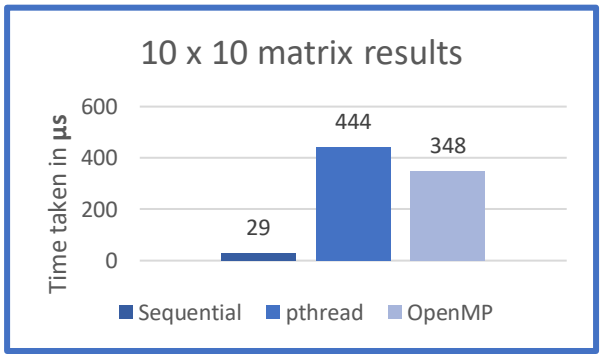


*Figure 7: 10 x 10 matrix results for sequential vs pthread vs OpenMP*

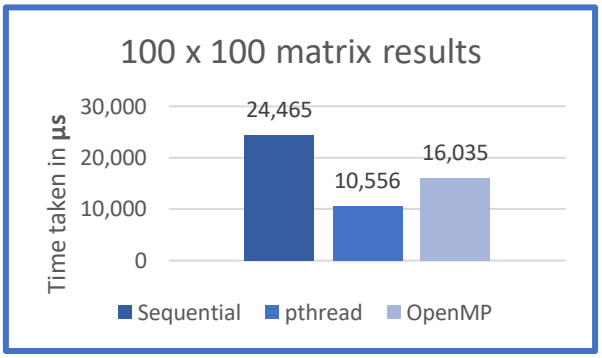


*Figure 8: 100 x 100 matrix results for sequential vs pthread vs OpenMP*

Figure 7 shows that, for a small program, sequential is still the best for performance. Additionally, if multithreading is used, it is better to use OpenMP.

Figure 8 indicates that multithreading with pthread is faster than with OpenMP, and both multithreading options are faster than a sequential program.

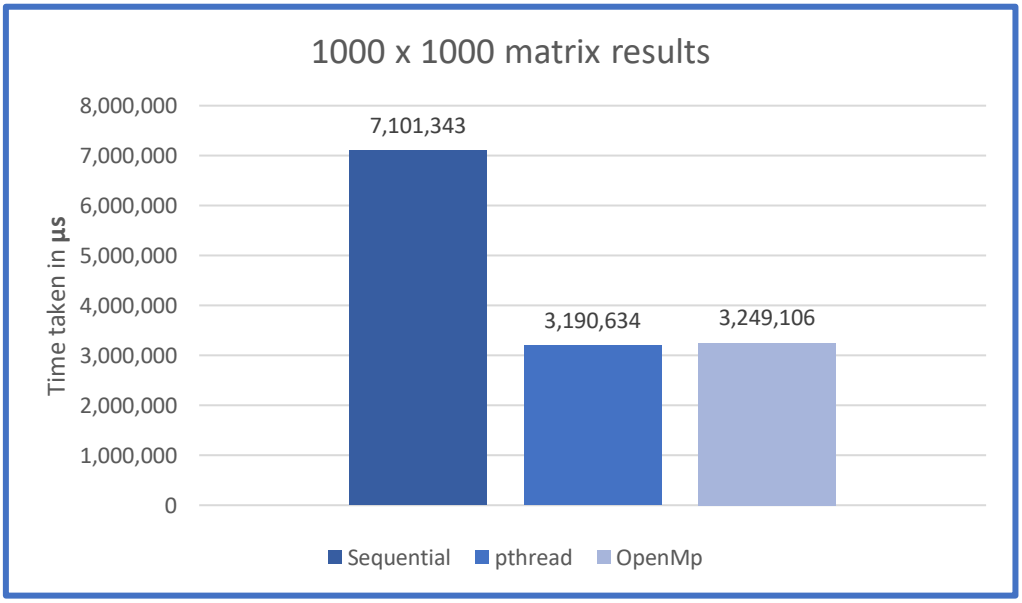


*Figure 9: 1000 x 1000 matrix results for sequential vs pthread vs OpenMP*

Figure 9 above shows that multithreading with pthread is slightly faster than OpenMP, and both more than half the time taken by a sequential program.

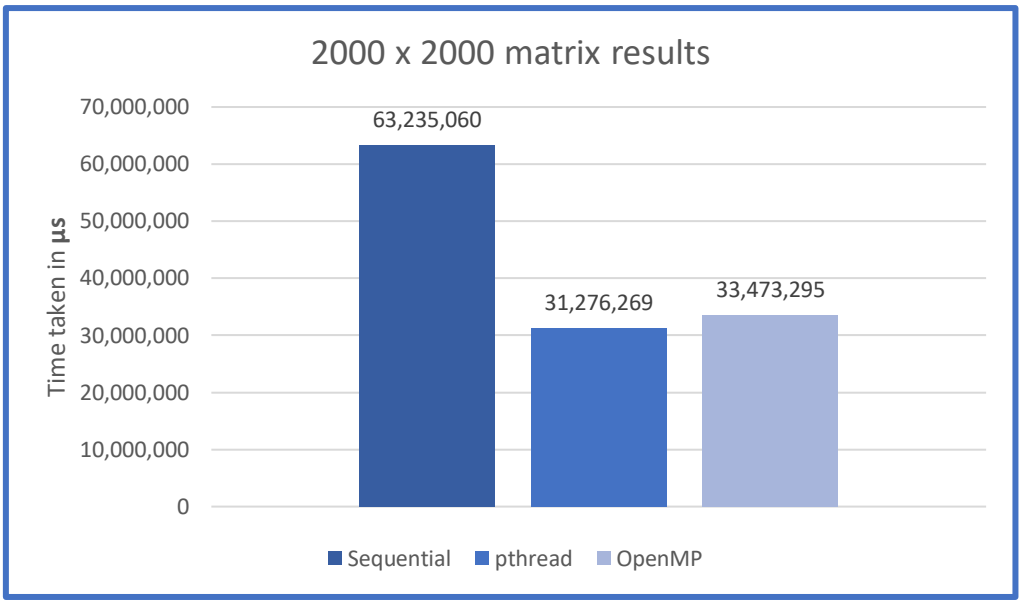


*Figure 10: 2000 x 2000 matrix results for sequential vs pthread vs OpenMP*

Figure 10 follows the same trend of Figure 9; pthread is slightly faster than OpenMP, but sequential is significantly far behind.

Overall, the results indicate that using the pthread multithreading module is marginally faster than that of the OpenMP module.

# CONCLUSION

The overall result of this experiment leaves me with a few takeaway points:

- Very small programs are better suited to a sequential program;
- The larger the task, the more effective multithreading becomes with regards to execution time; and
- Using pthread for multithreading is better than OpenMP.

These results and key points should of course be taken with a grain of salt, however. I am by no means the best C++ programmer and this was my first attempt at multithreaded programming. As such, these results could be skewed as my implementation of timing and thread creation/assignment.