**SIT315 Program Paradigms**

Task M2.T1P Parallel Matrix Multiplication

**Documentation**

**Roadmap to parallelisation**

Decomposition of my program into sub tasks.

The sub-tasks that I plan to do in sequence are:

Building matrix B and C

Printing matrix B and C

Printing matrix A

Printing the execution time in microseconds

The sub-tasks that I plan to do in parallel is:

Matrix multiplication of B and C

A screenshot of a cell phone

Description automatically generated

The sequential matrix multiplication code can be parallelised by nesting two loops and adding new threads in each loop which has variable i. This will do each row at the same time. However, we don’t require to thread each multiplication because the processor is efficient enough to make the values insignificant.

**Performance Evaluation**

In order to prove that multithreading improves the efficiency of the program we can use the execution time in microseconds as the metric for performance comparison. For my matrix sizes I’ve used 20x20, 50x50, 100x100, 500x500 and 1000x1000 for sequential program, pthread implementation and openMP implementation. Here I will be multithreading with 4 threads for both pthread and openMP. For each program I collected data in three iterations in order for me to find the average time elapsed and for better accuracy of my results.

**Results**

FILENAME: MatrixMul\_seq.cpp

20x20 🡪 34, 32, 37 (Average: 34.33)

50x50 🡪 894, 721, 825 (813.33)

100x100 🡪 4643, 5148, 4432 (4741)

500x500 🡪 943810, 950039, 876967 (923605.33)

1000x1000 🡪 6079931, 5751262, 6101305 (5977499.33)

FILENAME: MatrixMul\_pthread.cpp

20x20 🡪 189, 177, 187 (184.33)

50x50 🡪 679, 556, 433 (556)

100x100 🡪 2498, 2535, 3720 (2917.66)

500x500 🡪 212159, 225994, 211953 (216702)

1000x1000 🡪 3063290, 2642030, 2056909 (2587409.66)

FILENAME: MatrixMul\_openMP.cpp

20x20 🡪 43, 39, 38 (40)

50x50 🡪 515, 539, 530 (528)

100x100 🡪 4305, 4278, 4707 (4430)

500x500 🡪 812198, 832458, 816405 (820353.66)

1000x1000 🡪 5990445, 5797674, 5756119 (5848079.33)

Sequential vs pthread implementation

As we can see when comparing pthread implementation with sequential program for matrix size of 100x100 the execution time with multithreading is almost twice as faster than that of sequential. And as the matrix size increases the execution time ranges from 2 times to 5 times faster than sequential program.

Sequential vs openMP implementation

However, when comparing sequential program with openMP implementation, the execution time for matrices with larger sizes are close and doesn’t have a huge difference like that of pthread implementation but nevertheless it is still 2-3% faster.

**Conclusion**

From the acquired data it simply concludes that matrices with small size such as 20x20 is faster to complete using the sequential matrix. Whereas, for matrices with bigger sizes such as 500x500 and 1000x1000 multithreading performance shows a significant difference as compared to that of sequential. And overall, pthread implementation is much faster as compared to sequential and openMP implementation for larger matrix sizes