Project 2 – Threading Introduction

Julio Anacleto | Abdullah Almaroof

Device Specification: Raspberry Pi 5 – 4 Standard Cores

Parallel decomposition: Performed a standard dot product operation in which we perform the dot product using the standard row by column method. Later we learned about the concept of transposing the matrix and multiplying row by row while still getting the correct result for a dot product. This method would be faster than the standard operation as the data stored in memory of the column values that we are fetching are closer in proximity to each other.

Challenges Faced: When creating the single thread applications as a base before designing the multithreaded application, we had to learn the use and application of double pointers and how it translates to 2 dimensional arrays. Had trouble understanding how double pointers and memory allocation could create a 2-dimensional array initially but realized later that we are creating an array of pointers with each pointer element pointing to an array of integer values. From there we learned how to implement that design into a multithreaded application that will take the shared matrix between multiple threads and operate on different regions of the matrix.

Future Improvements: If we were to improve this in a future iteration of the same project, we would transpose one of the matrixes so that we can multiply row element by row element rather than jumping between large gaps in memory for column values.

Locking: The reason that we do not need to implement locks into this project is because the dot product on 2 matrixes is considered “embarrassingly parallel” meaning that we can perform the operations without two instances operating in the same region. By ensuring that our code does not allow threads to operate in the same region we have no reason to implement locks into our program.

Contributions:

- Abdullah – Wrote single threaded application and created Makefile for compilation.
- Julio – Wrote multi-threaded application, researched double pointers, Created report.

# of Threads	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Avg. Runtime	Avg, Speed Up
1	21.321	21.289	21.271	21.28	21.24	21.279	21.24	21.295	21.276875	1
2	16.45	16.44	16.557	17.689	17.513	17.693	17.712	16.457	17.063875	1.246895855
4	14.395	7.893	14.159	10.897	7.903	7.949	14.303	14.709	11.526	1.845989502
8	12.707	12.711	12.188	13.192	11.733	11.818	12.197	12.131	12.334625	1.724971371
16	12.742	12.727	12.488	12.06	13.207	12.854	12.597	12.817	12.6865	1.677127261