Andrew Gates

Professor Matthew Tolentino

TCES 420

October 30, 2016

Project #2 – Multi-Threaded Matrix Multiplication

Machine tested on: Raspberry Pi 3 with 1.2 GHz quad-core ARM Cortex-A53.

Challenges: The biggest challenge I encountered was trying to get the multi-threading to work properly. I had the matrix multiplication and the single thread working without too much trouble. But when I started on the multi-threading I got stuck for a while trying to figure out first how to create multi-threads, and second how to properly split up my matrix multiplication. I initially tested the multi-threading with a simple function that would print something, and then when I could get that working properly I decided to split my calculations for each thread into chunks of the array so that each thread had a similar size of the array to compute.

A way to further improve this implementation to increase performance would be adding in some way to check to see if a certain thread is done. Say there are 2 threads and thread 1 is going from rows 1-50 and thread 2 is going from 51-100. If thread 1 completes much quicker than thread 2, thread 1 could start working from the back of the matrix and complete part of what thread 2 was going to do. Another way would be to have your main function work as a thread instead of waiting for the threads to return. This would allow you to have an extra thread.

The reason that you don’t need locking in this case is since the various threads are not accessing data that is being changed. Each thread has their own section of the array which does not get modified by another thread. If the threads were dealing with data that was changing, then we would need locking.

Runtimes for 2000 matrix –

Single:

599 seconds

Multi:

1 Thread – 597 seconds

2 Threads – 297 seconds

4 Threads – 152 seconds

8 Threads – 173 seconds

16 Threads – 175 seconds

|  |  |
| --- | --- |
| # Threads | Speedup |
| 1 | 2 |
| 2 | 302 |
| 4 | 447 |
| 8 | 426 |
| 16 | 424 |