Report on Project 4, Part 2

Parallelization of Matrix Multiplication

CS415

April 12, 2017

Evan Su

Version 2.0

Table of Contents

Overview:

Computing matrix multiplication is known to have a long calculation time. Matrix multiplication itself is not necessarily difficult but as the size of the matrix grows, more calculations are needed. Computing matrix multiplication has a run time of n^3. The purpose of this experiment is to show the speedup achieved with parallelization.

Test Methodology:

Two different programs are written to test the matrix multiplication time, sequential and parallel. The sequential code will be used as a controlled variable. The code will be tested at various sizes starting at 360 by 360 sized matrix. For more detail of the sequential and parallel code, please see below.

Sequential

The sequential code first allocates spaces for three matrixes, 2 for multiplicand and 1 for product. The two multiplicand matrixes are filled. After being filled, the two matrixes are multiplied together and the results are stored in the product matrix. The time starts when the calculation starts and ends when the calculation ends.

Parallel

The parallel code first allocates spaces for three tiles in all the cores, 2 for multiplicand and 1 for product. The size of the tiles is determined by the size of the matrix and the number of cores. The two multiplicand matrixes are filled. After being filled, all the cores will follow Cannon's algorithm to multiple the two matrixes (spread across all the tiles) together. The time starts when the calculation starts and ends when the calculation ends.

Code Revisions:

Addressing Comments:

I have received the following concerns over my initial parallel implementation and addressed them accordingly. I grouped similar concerns together so that I can address them all at once.

- Code is very dense. Spacing out code statements with new line would make is easier to read.
- Code is very dense and bit difficult to read. This is partly due to it being written in C but there is a lack of auxiliary functions to make the code much more readable. Also line spacing would be nice to see.

  Resulting Changes:
  I added more line space between different lines of code. I also added additional functions for complex row and column id calculations. I added section headers for long functions to add readability.

- The code would also benefit from some more functions to make the code easier to read. For instance, a allocateMatrix function would be nice to see. It would also be nice to see a functions for sending and receiving.
- Although the functionality could be split into subroutines a little more, the overall code structure is solid.
- Everything was nice and understandable. I might put some of the repeated things in a function(matrix memory allocation, for example).

  Resulting Changes:
  I added an allocate matrix function, free matrix function and calculate neighbor function to reduce the amount of repeating code. I did not added send and receive functions because the function would only be used once at the beginning with a very specific set of circumstances to work.

- Instead of the switch statement it would nice to see just the sqrt() function in math.h and then some checking. It is possible to run this code on the cluster with 49 cores.
- On a more style-related note, you could make your switch statement at the beginning check for a remainder when you divide the size by it's square root, which would make it a bit less verbose and help it handle the general case.

  Resulting Changes:
  I change the switch statement to a square root function

- I really like the print flag.

  Resulting Changes:
  I added more flags for printing matrix and printing time

- I'm not sure what the output time means. I'm assuming its in micro seconds. That would be nice to see in the documentation.

  Resulting Changes:
  I added additional notes in the documentation explicitly stating that the time printout is in microseconds. I did not change the printout format because I wanted to keep the output in csv format

- The only markdown is mostly from not collecting the final result into one matrix.

  Resulting Changes:
  I added a function that collects all the data from the tiles and puts them into one matrix

- Not that it's super important, but you might clean this[makefile] up a bit(there's still stuff that looks like it's from PA1). It's really just for your own benefit, but it might help if you have to debug something in you makefile.

  Resulting Changes:
  I glad this person noticed the relics from previous projects in my makefile. I am not making any immediate changes to the makefile in case of the off chance of needing to use old code again.

- You might speed up your initial matrix communication time by transposing things to begin with to avoid the whole doubly-nested-for-loop inside a triply-nested-for-loop thing.

  Resulting Changes:
  I did not change my matrix communication method because I do not possess the knowledge to implement the feature of transposing matrices.

Data Analysis:

The raw data from the tests can be found in the file project4Analysis.

Sequential

The run time of the sequential code exhibits a runtime of O(n^3) as shown in Graph 1.1. The execution time grows exponentially as the length of the matrix grows linearly.

Parallel

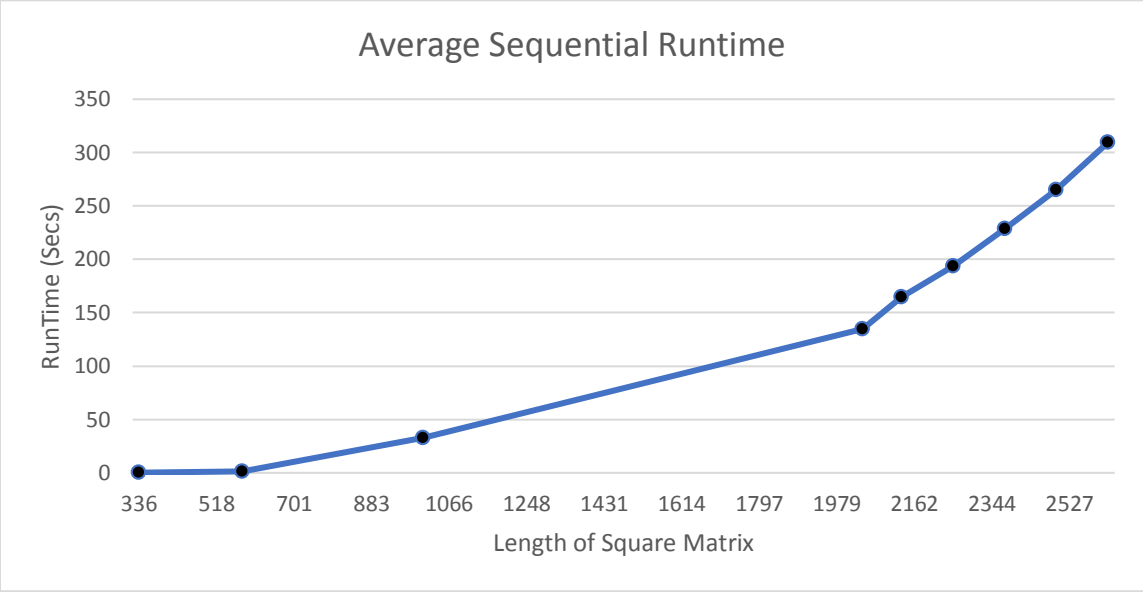
TODO

Conclusion:

TODO

Graphs and Tables

Graph 1.1



The graph shows the runtime of matrix multiplication is O(n^3).

Table 1.1

Sequential matrix runtime(Secs)									
Length of square matrix	360	600	1020	2040	2160	2280	2400	2520	2640
Average	0.178956	1.436156	33.01261	134.629	164.6035	193.7146	228.746	265.135	309.6805

The table is the average sequential runtime of matrix multiplication. The runtime is O(n^3) as the length increases, the runtime increases exponentially.