PA4 – Matrix Multiplication Documentation

Brian Conway

13 Apr. 2017

CS415

Matrix Multiplication - Sequential

This sequential implementation of matrix multiplication involves taking the dot product of each row in matrix A and with each column of matrix B in order to calculate the values of each cell in matrix C.

Methodology

The file sequential.cpp is responsible for generating random values for square matrices A and B based on the dimensions specified, matrix multiplying them, and outputting statistics about the timing as well as optionally writing the three matrices to a file.

The program has one mandatory command line parameter for the dimensions of the square matrix, and an optional parameter for if you want the matrices output to a file. If the second command line parameter is “y”, then the results are output to the file “output.txt”.

The program first calls generateNumbers(), a function which uses c++11 random devices to generate uniformly distributed values from 0 to 999,999 for all of the cells of matrices A and B. The generator is seeded appropriately so that this sequential version generates the same numbers as the eventual parallel one.

Afterwards, the program calls the matrixMult() function, which loops through each row of A and column of B, then loops through each element in that row/column combination to sum up the multiplication of all corresponding elements and store the result in the corresponding cell in C. For example, to calculate the cell in row 3, column 2 of C, it takes the dot product of row 3 of matrix A and column 2 of matrix B.

It times this using the Timer class, and it repeats this process the amount of times specified by the NUM_MEASUREMENTS constant. Afterwards, it calls the calcStatistics() function to get an average of the time taken to multiply the matrices and outputs it to the console.

Timing is done using the custom Timer class, which has start, stop, and resume functions similar to a stopwatch. Timer makes use of timevals and the gettimeofday() function in order to get an accurate reading of seconds and milliseconds at certain instants. When the elapsed time is to be given, the Timer takes the difference in seconds and milliseconds between when the timer was started and when the timer was stopped. It then combines the seconds and milliseconds reading into a double precision floating point variable.

Results – Sequential

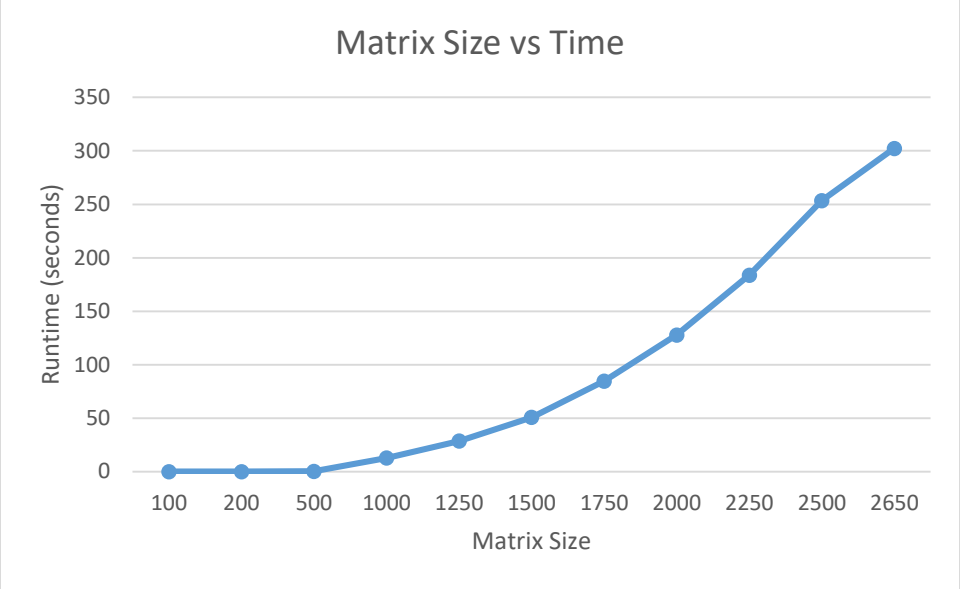

Fig. 1 Graph of sequential runtimes with different sizes of the matrices.

100	200	500	1000	1250	1500	1750	2000	2250	2500	2650
0.002366	0.011048	0.33475	12.641	28.5228	50.7558	84.7339	127.827	183.79	253.556	302.379

Table 1: Timings matrix multiplication calculations, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

Analysis

Sequential

As for the sequential algorithm, the main goal was to find out how the runtimes would look with different square matrix sizes, up to five minutes. With the $O(n^3)$ algorithm being used for matrix multiplication, it got to five minute runtimes at around the 2600x2600 matrix sizes.