CS3014 Lab 1: Parallel Matrix Multiplication

Killian McCabe and Samuel Mardirosian

# Researching

We began this project by researching common algorithms applied to matrix multiplication. We learned about the Strassen algorithm and the algorithms that improved on Strassens like the Coppersmith-Winograd algorithm. These algorithms require both matrices that are being multiplies to be 2^n X 2^n. The resizing of the input matrices will create a large overhead. We found it would be best to apply Strassens only to extremely large matrices where their dimensions are either exactly or only slightly smaller than 2^n X 2^n. We decided then to implement Strassens but only apply it the input matrices satisfy those two conditions.

We decided to parallelize the naïve method and apply that to matrices that didn’t meet the requirements for Strassens. And on very small matrices we would use the non-parallel naïve method.

# Our Implementation

An estimation of work required in multiplying the matrices is made by multiplying the dimensions first input matrix to retrieve its size. We then choose the most efficient multiplication method according to the amount of work required.

If the matrix is small (i.e. less than 80x80) we calculate the result through a simple non-parallel procedure.

Elements of a medium to large sized matrix will be calculated in parallel. For this we found it best to only parallelize the outermost for-loop. Parallelizing the inner for-loop only resulted in a slower procedure. Next we began to experiment with loop-unrolling. Unrolling the inner for-loop achieved a small speed-up on very large scale matrices.

As we began to implement the Strassen algorithm it became apparent why it’s only efficient on extremely large matrices. Firstly input matrices dimensions need to be expanded up to the next highest power of 2. This means creating 2 new larger matrices and copying over their elements. Next we noticed that each step of the recursion requires both input matrices to be split into 4 parts and for 7 more matrices to be created to hold the calculated values in that level of recursion. This meant 15 matrices have to be created and populated at each step of recursion. The advantage of Strassens is that it saves on one multiply operation at each level of recursion, though it adds several additions and subtractions.

Once we had Strassens algorithm complete it become apparent that there was no clear point at which we could apply it. Strassens was still much slower than naïve with matrices of size 4096x4096. With matrices of that size it was taking too long to test the next power of 2. This meant we had to give up on the application of our Strassens method as we could find no valid point at which it would become effective. According to some online sources Strassens will never pay off. There are two proposed reasons for this. Some say Strassens would only be worthwhile on matrices too large to be handled by present-day computers. Others propose that Strassens was designed to run on systems where floating point multiplication took much longer than addition but the difference between those two times is now much less.