Multi-threaded Matrix Multiplication

Gracie Bliss Jose Valencia

March 2023

Abstract

Matrices were first invented in the 1850s by Andrew Cayley, with the term first being introduced by James Sylvester. Since then matrices and matrix multiplication has been used in other forms of mathematics and in many professions. They have been used to make graphs, conduct studies, and calculate statistics, which can aid in almost all jobs. This means that matrices are a very important and integral part of mathematics and matrix multiplication is a necessity. Our project explores the idea that matrix multiplication is a task that can be parallelized, which will result in faster run times.

1 Introduction

Matrix multiplication has been established as the most important matrix operation due to it being commonly used to solve or approximate solutions across a wide range of problems and fields. It can be used to create graphs, calculate statistics, and conduct studies and research. Population growth and mortality rate are examples of statistics that can be represented by matrices and calculated using the help of matrix multiplication. This study aims to increase the efficiency of existing matrix multiplication algorithms by adapting them safely to a multi-threaded architecture, where tasks are running parallel to each other instead of in a linear fashion. The usual method of matrix multiplication relies on the dot product operation to multiply various combinations of rows and columns. The calculations involved in this process are well suited for parallelization due to the lack of a need to modify shared resources. In other words, each dot product operation works independently of each other, which allows most of the process to be parallelized, resulting in a significant speed-up. The program will use already established alternative algorithms like the Strassen algorithm and to compare them to the the method of paralleliz-ing the dot product operations in the usual matrix multiplication method. The results in this paper will demonstrate the dot product algorithm parallelized and benchmark them against other methods and its own linear counterpart. The idea here is that since alternative methods of multiplying matrices are only slightly more efficient, parallelizing the dot product method well enough should result in a faster time than this alternative methods up to a certain large size of matrices.

2 Problem Statement

The current algorithms to solve matrix multiplication run at $O(n^3)$, with the Strassen algorithm running at $O(n^{log7})$ at best. Both of these time complexities ultimately, for a single-threaded algorithm, leads to a long run time when working with large matrices. Because matrix multiplication is a necessity for so many computations, making a faster algorithm would be beneficial. With this research we do not intend to create new algorithms that deal with matrix multiplication in a better time complexity, but instead change the implementation of the dot product algorithm from linear to parallel in the most efficient way possible which would theoretically cause a speed up in the running time and then compare this method to see if it is faster than using a slightly more efficient method like the Strassen algorithm in a linear fashion. Also, if the dot product method being parallelized results faster than a method like Strassen or divide and conquer, up to what size of matrices is would it be faster.

3 Methodology

As mentioned before, the usual dot product method of matrix multiplication is already well suited to be adapted to a multi-threaded environment. This section will go more in depth about why this is true and how exactly is this method going to be parallelized and compared to alternative methods.

First note that for matrix multiplication to work with the usual dot product method, the number of columns in the first column must equal the number of rows in the second one. If the two matrices being multiplied satisfy this requirement, we can multiply the matrices by taking the dot product of each horizontal vector in the first matrix with each vertical vector on the second matrix. The results of these operations are then arranged in a result matrix. If the first matrix has dimensions of $m \times n$ and the second matrix has dimensions of $n \times p$, then the resulting matrix will be of size $m \times p$. This method can be easily parallelized by calculating each of this dot product operations on a separate threads. Although the two initial matrices are a shared resources, the threads will only be reading from them so there is no problem here. The resultant matrix is also a shared resources, but when using an array each inividual element is safe to be accessed by different threads at the same time, which is exactly the case we run into when each thread records the result of its operation to the result matrix.

With this new parallel method of running the dot product algorithm working. It is important to explain how it will be compared to other alternative, slightly more efficient, specifically the divide and conquer method and the Strassen method. Both of this methods work by splitting big matrices into smaller ones recursively. The Strassen method also has its owns set of formulas for dealing with multiplying the smaller matrices and rejoining them into the result matrix. However both of this methods only work correctly when the input matrices are of dimensions that are a power of two. Because of this when comparing our parallelized dot prod-

uct method to the these two other alternate methods, we will work with matrices that fulfills this requirements. Also, as the main idea in these two alternate methods is to split bigger matrices into small ones, we will only be testing with matrices bigger than 2×2. We will not only be testing to see if the parallelized dot product method is faster on single separate sizes of matrices, but up to what size of matrices being multiplied is the parallelized dot product method faster.