MATRIX MULTIPLICATION ALGORITHM COMPARISON

ARDEN RASMUSSEN

Abstract. Matrix multiplication is a key part of may different computer algorithms, and is widely used across the fields of computer science. Because of the important role that it plays, much work has been put into developing more efficient algorithms for matrix multiplication. A lot of progress has been made because of this research, there are now a number of different matrix multiplication algorithms, that are used for different purposes. This paper will compare the best, average, and worst case time and memory requirements for these different matrix multiplication algorithms, and provide some guidelines as to when the different methods are most useful.

1. Introduction

Matrix multiplication has industry and research applications in all mathematical or scientific fields. However, there were no known algorithms that improved on the naive one until 1969. Most of the faster matrix multiplication algorithms greatly improve the asymptotic complexity of matrix multiplication, but can also incurs a very large initial constant, meaning that for matrices smaller than a specified size, the algorithm would be less efficient than some other simpler algorithms.

This paper aims to provide an introduction and implementation details of the more common matrix multiplication algorithms. As well as comparing the complexities of the different algorithms, and determining a optimal range of matrices for the different algorithms.

1.1. Notation.

1.1.1. *Matrix.* In this paper we will denote a matrix A which is $N \times M$ to be the form

$$A = \begin{bmatrix} A_1^1 & A_2^1 & \cdots & A_N^1 \\ A_1^2 & A_2^2 & \cdots & A_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ A_1^M & A_2^M & \cdots & A_N^M \end{bmatrix}$$

.

Most of these algorithms can be used on any matrix, but for simplicity we will assume that a matrix is square $(N \times N)$ unless otherwise noted.

1.1.2. *Complexity.* We will use Big O notation to describe the limiting behavior of a function when the argument tends towards a particular value or infinity. This notation characterizes function according to their growth rates. This will be used for both our notation for time complexity, and space complexity.

$$\mathcal{O}_t \equiv \text{Time complexity}$$
$$\mathcal{O}_s \equiv \text{Space complexity}$$

1.1.3. *Operations.* Sometimes it is beneficial to talk about the number of operations that are required for something. We will use the term $flop$ for this, meaning Floating Point Operations. It is important to note that we consider all basic operations using a floating point as a $flop$, this means that addition, subtraction, multiplication, and division are all given the same weight. However, in reality this is not the case.

In [Int18] they provide latency for the different floating point operations. They claim that on some CPU architectures could be very significant. A table of the data is provided below.

Operation	Latency
Addition	3
Multiplication	5
Division	15

This is not always the case, many newer CPU architectures greatly improve upon this. However, it still must be noted that not all operations are the same. And that multiplication and division frequently require more time than simple addition or subtraction. Because of this it is often a good goal to reduce the number of complex operations within an algorithm, in order to improve the complexity of the algorithm.

Date: January 26, 2019.

2. Naive

This algorithm is the most basic algorithm, that directly follows the simplistic mathematical method for computing the matrix multiplication. This method is very easy to implement, but has many draw backs.

3. Strassen

4. Le Gall

References

[Int18] Intel. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, 2018.