The way through ‘complexity’ towards an optimal algorithm for matrix multiplication

KASHAN HASAN - 180123022

TANMAY JAIN -180123050

Abstract

In this report, we analyze the complexity of the problem of multiplying two matrices. We review the various developments in matrix multiplication algorithms taken over time. We shall look at Strassen, and it is parallel implementation to reduce the execution time significantly. Also, we analyze Frievald's algorithms, a quadratic time verifier of the problem. We take a glance at one of the latest algorithms (Coppersmith and Winograd) and finally conclude by our understanding and intuition.

Introduction

Matrix multiplication is one of the most fundamental operations in mathematics, which finds its applications in almost every area, such as network theory, perspective projections, linear equations, formulation of the Lorentz transformation from special relativity, to mention a few. Often a bottleneck for many crucial algorithms and a fast algorithm will increase the functionality of these algorithms. Though the optimal solution is hypothesized to be of complexity $O(n^2)$, we are yet to find such an algorithm.

Background

Naive Algorithm :

If A is an m x n matrix and B is an n x r matrix, then the product C = AB is an m x r matrix. The (i, j) entry of the product is computed as follows:

$$C_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$$

The standard method for multiplying n × n matrices requires $O(n^3)$ multiplications.

Divide-and -Conquer & Strassen’s approach

When we look at Strassen's method of finding the product of two 2n x 2n matrices, the basic idea behind this is to make partition of the given nxn matrix into smaller four n/2 x n/2 matrices and reducing the numbers of multiplications required to 7.

(*He probably derived the intuition from this from Gauss: While multiplying complex numbers (a + bi) and(c + di) , the result (ac – bd) + (bc + ad)i which apparently consists of 4 multiplications can be rewritten using only 3 multiplications because bc +ad = (a+b)(c+d) - ac-bd !)*

Parallel Implementation of Strassen's Algorithm

Every algorithm's running time is dependent on the computer architecture used to implement it. Parallel implementation using dynamic multithreading reduces the execution time by performing concurrent operations. To explain it, we first introduce some terms.

- Work

  Total time to execute everything on a single processor.

- Span:

  Longest time to execute the threads along any path.

Note: If each thread takes unit time, span is the length of the critical path.

- ❖ $T_{1} =$ work, T_∞ = span
- ❖ P = number of (identical processors)
- ❖ T_P = running time using P processors

Strassen's algorithm(Parallelised)

1. Partition each of the matrices into four n/2 x n/2 submatrices.

(This step takes Θ(1) work and span by index calculations.)

2. Create 10 matrices $S_1, S_2, \ldots, S_{10}$. Each is n/2 x n/2 and is the sum or difference of two matrices created in the previous step.

*(Can create all 10 matrices with $\Theta(n^2)$work and Θ(log n) span using the doubly nested **parallel for** loops).*

3. Recursively compute 7 matrix products $P_1, P_2, \ldots, P_7$, each n/2 x n/2.

*(Recursively **spawn** the computation of the seven products.)*

4. Compute n/2 x n/2 submatrices of C by adding and subtracting various combinations of the P_i.

Results achieved

- Speed-up: T_1 / T_P (Maximum speedup bounded by P)
- Parallelism: T_1 / T_∞ (Maximum speed for ∞ processors !)
- $T_1(n) = (n^{\log 7})$
- $T_\infty(n) = (\log_2 n)$

Current State-of-the-Art

Now that we have some idea about the problem and basic algorithms to solve it, it brings us to the natural question, "What exactly is the complexity of this problem ?" The exponent for matrix multiplication has proved so elusive that it has been given a name: ω. Formally, ω is the smallest number such that $O(n^{\omega + \varepsilon})$ multiplications suffice for all $\varepsilon > 0$. Because all n^2 entries must be part of any computation, ω is at least 2. Nevertheless, does a quadratic-time algorithm really exist? Before we address this question, we will discuss an algorithm that can verify whether n× n matrices A, B, and C satisfy the condition AB = C, and interestingly it can do so in quadratic time!

Freivalds' Algorithm (1977)

An essential step in any algorithm is verifying the correctness of it. Matrix multiplication over the years has seen numerous practical applications in various fields. Hence, the correctness of output is critical. Frievald's Algorithm provides an elegant solution. It gives an $O(n^2)$ time randomized Algorithm with a bounded error probability. We will now be looking at the process involved.

Pseudocode taking input as n×n matrices A, B and C.

- Our aim is to verify if A×B=C. For this, we choose an n×1 column vector r→∈{0,1}n, uniformly and at random.
- Next, we compute r ·AB and r ·C and compare the results obtained. Note that matrix multiplication is associative; i.e., we have (r · A) · B = r ·(AB) = r · The important point to be observed here is that r ·A can be computed in time $O(n^2)$ and since the result is again a 1 × n vector r, also r ·B and r ·C are computable in time $O(n^2)$.
- Now comes the output
  - If A · (B · r)≠C · r. Hence, we run the algorithm for a certain number ‘k’ of randomly chosen vectors from {0, 1} n and output “NO” if at least one of the tests fails. Otherwise, we output “YES.”
  - If A · (B · r)=C · r. Furthermore, it is obvious that no error occurs provided that AB = C. This implies the probability of correctness equal to 1.Consequently, it remains to analyze the probability of success in the case that AB ≠ C.

- **Correctness** : P(ABx = Cx | AB = C) = 1 . Let us assume AB ≠ C . Then AB - C ≠ 0, so there exists i, j with $(AB - C)_{ij} \neq 0$. Let $(d_1, d_2, \dots, d_n)$ be i-th row of AB - C ; $d_j \neq 0$ .Now P((AB - C) x = 0 | AB ≠C) ≤ P($\sum^{n}_{i=1} d_i x_i$ | AB ≠ C) = P(x_j = {− 1 /d_j $\sum_{i \neq j} d_i x_i$} | AB ≠ C) ≤ ½ P(ABx = Cx | AB ≠ C) ≤ ½ .This proves that the maximum possible error is ½.

Error Reduction in Frievald's Algorithm

If we run the algorithm 'k' times we have 'k' independent Bernoulli trials with failure probability at most 1/2. Then the probability to produce still in the incorrect result is upper bounded by $(½)^k$.
Suppose we wish to reduce the probability of error to ϵ. The straightforward approach involves running Algorithm for k independent trials, where $(1/2)^k < \epsilon$ or $k > \log_2(1/\epsilon)$. This strategy requires $\Omega(kn^2)$ operations and $\lceil \log_2(1/\epsilon) \rceil \lceil \log_2 n \rceil$ random bits.

Although Frievald's algorithm is incorrect with small probability, it is fast. The situation here is analogous to what we saw in NP-complete problems where problems not having currently known polynomial-time algorithm did have polynomial-time verifiers. The million-dollar problem of whether P = NP and here whether ω = 2 is in a way similar so this brings us to the original question: Is there a quadratic time algorithm for matrix multiplication?

Group-Theoretic Approach

Coppersmith-Winograd Algorithm

Though this algorithm is mathematically highly sophisticated involving rigorous use of group theory algebra(*that is currently beyond our understanding !)*, the basic idea is similar to .the Strassen algorithm. That is, for some p, it gives a way to multiply p × p matrices using n<<< p^3 multiplications, and apply the method recursively to show that ω < $\log_p n$ .It should also be noted that Coppersmith-Winograd is not so much of an algorithm than an existence proof and it is rarely used in practice because of large constant factors hidden in its running time. The time complexity of this algorithm is $O(n^{2.375477})$.

Conclusion

In the last 30 years, although developments have taken place in group-theoretic approaches, results in the light of time complexity still seem pretty stagnant. The recent algorithms are purely theoretical based and for now, don't have any practical utility at all. Also, the difference in running time is noticeable only for astronomical sizes of matrices which can't be processed by our current hardware technology. Still finding the optimal exponent for matrix multiplication even on theoretical grounds is an exciting problem. The whole research in this area can be best visualised as a journey from $O(n^3)$ to the destination $O(n^2)$ which is yet to be completed. Most researchers believe that the quadratic time algorithm exists. And we too believe that it's a matter of time before we find one.

References

- Grayson, Brian, and Robert Van De Geijn. "A high performance parallel Strassen implementation." *Parallel Processing Letters* 6.01 (1996): 3-12.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford(2009) [1990]. *Introduction to Algorithms* (3rd ed.). MIT Press and McGraw-Hill.
- Coppersmith, Don, and Shmuel Winograd. "Matrix multiplication via arithmetic progressions." *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 1987.
- Korec, Ivan, and Jiří Wiedermann. "Deterministic verification of integer matrix multiplication in quadratic time." *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, Cham, 2014.
- Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge (1995)