INVESTIGATION AND EXPERIMENTS ON STRASSEN'S ALGORITHM

SEOK WON LEE

Abstract. In this paper, we prove theorems about the optimality of Strassen's algorithm on 2×2 matrices and the computational complexity of the algorithm. We also present some experimental results of Strassen's algorithm. The results show that Strassen's algorithm is still effective on modern multi-core architectures.

1. Introduction

There are many well-known techniques for fast multiplication. For multiplication of integers and polynomials, Karatsuba [7] came up with an algorithm requiring $O(n^{\log_2 3})$ operations where n is the number of words in an integer or the degree of a polynomial. Fast Fourier Transform, which traces back to Gauss and was implemented on a computer by Cooley & Tukey [2], can reduce the complexity of multiplying two polynomial to $O(n \log n)$ where n is the degree of the polynomials. Schönhage & Strassen [11] reduced the complexity of integer multiplication to $O(n \log n \log \log n)$ where n is the number of words in the integers.

Fast multiplication has also been invented in the area of matrix multiplication. In 1969, Strassen [12] published an explicit algorithm for the multiplication of two 2×2 matrices using 18 ring additions and 7 ring multiplications, which can multiply two $n \times n$ matrices with the complexity of $O(n^{2.807})$ by recursion. Afterwards Pan [9], Bini *et al.* [1] and Schönhage [10] reduced the exponent one after another, and Coppersmith & Winograd [3] reached $O(n^{2.376})$, followed by Le Gall [5]'s $O(n^{2.374})$. The holy grail of the complexity of matrix multiplication is hoped to be $O(n^2)$.

This paper aims to investigate theorems and analyze experimental results regarding an aforementioned fast matrix multiplication method, Strassen's algorithm. In particular, we want to prove the following theorems:

Theorem 1.1. *Two 2×2 matrices can be multiplied with 7 ring multiplications.*

Theorem 1.2. *The multiplication of two 2×2 matrices requires at least 7 ring multiplications even if the base ring is a field.*

Therefore, we can say that Strassen's algorithm is optimal in multiplying two 2×2 matrices in terms of the number of ring multiplications. We also prove the following theorem regarding the complexity of Strassen's algorithm:

Theorem 1.3. *Given $n \in \mathbb{Z}_+$, Strassen's algorithm multiplies two $n \times n$ matrices with the complexity of $O(n^{\log_2 7}) \approx O(n^{2.807})$.*

In Section 2, we take a look at Strassen's algorithm and prove theorems about its computational complexity and optimality. In Section 3, we present experimental results of classical and Strassen's matrix multiplication on CPU and GPU, which shows that that Strassen's algorithm is still effective on modern multi-core architectures.

2. Strassen's Algorithm

In this section, we introduce Strassen's matrix multiplication and analyze the multiplication complexity of the algorithm. Then, we prove that the algorithm is optimal for the multiplication of two 2×2 matrices in terms of the number of ring multiplications.

2.1. **Strassen's Algorithm.**

Theorem 2.1 (Strassen)**.** *Let A, B and C be 2×2 matrices and M_i be 1×1 matrices for $i = 1, 2, \ldots, 7$, each of which defined by*

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},\ B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},\ C = AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22})B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) \\ M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12})B_{22} \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

Then, the following equations hold:

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6. \end{aligned}$$

Proof. It is a computation. □

Corollary 2.2. *The multiplication of two 2×2 matrices can be computed with 7 ring multiplications.*

Proof. In Theorem 2.1, 7 ring multiplications are used to compute M_i's for $i = 1, 2, \ldots, 7$ and they are all multiplications that are used. □

Remark 2.3. Winograd invented Strassen-Winograd algorithm that multiplies two 2×2 matrices with 15 additions and 7 multiplications. See Algorithm 12.1 of [6].

Theorem 2.4. *For $n \in \mathbb{Z}_+$, Strassen's algorithm computes the product of two $n \times n$ matrices with at most $49n^{\log_2 7} \in O(n^{\log_2 7})$ ring operations.*

Proof. Let $m = 2^{\lceil \log_2 n \rceil}$ so that $n \leq m < 2n$. By applying Lemma 8.2 from [6] with $b = 7$, $c = 4$, $d = 1$ and $S(n) = \frac{9}{2}n^2$, we have $T(n) \leq T(m) \leq m^{\log_2 7} + 6m^2(m^{\log \frac{7}{4}} - 1) \leq 7m^{\log_2 7} < 49n^{\log_2 7}$. □

2.2. **Optimality of Strassen's Algorithm.** The content of this subsection is largely taken from Winograd's work [14].

Definition 2.5. *The evaluation of an N-step algorithm α is a function $e_\alpha : \{1, 2, \ldots, N\} \to \mathbb{Q}[a_{11}, \ldots, a_{22}, b_{11}, \ldots, b_{22}]$ such that either $e_\alpha(j) \in \mathbb{Q} \cup \{a_{11}, \ldots, a_{22}, b_{11}, \ldots, b_{22}\}$ or $e_\alpha(j)$ can be obtained by adding, subtracting or multiplying $e_\alpha(j_1)$ and $e_\alpha(j_2)$ where $j_1, j_2 < j$.*

We say that the algorithm can compute the product of the two matrices if there exist j_1, j_2, j_3 and j_4 such that

$$\begin{aligned} e_\alpha(j_1) &= a_{11}b_{11} + a_{12}b_{21}, \\ e_\alpha(j_2) &= a_{11}b_{12} + a_{12}b_{22}, \\ e_\alpha(j_3) &= a_{21}b_{11} + a_{22}b_{21}, \\ e_\alpha(j_4) &= a_{21}b_{12} + a_{22}b_{22}. \end{aligned}$$

We present another definition and then move on to lemmas.

Definition 2.6. *If $e_\alpha(j) \in \mathbb{Q} \cup \{a_{11}, \dots, a_{22}, b_{11}, \dots, b_{22}\}$, then j-th step of the algorithm is a data step. Otherwise, it is a computation step.*

Lemma 2.7. *Let α be an algorithm for multiplying two 2×2 matrices and $p_1, p_2, \dots, p_m$ be the results of the computation steps which are multiplications. Then for every step k there exist rational numbers r_i, $r_{i,j}$ and $r'_{i,j}$ depending on k such that*

$$e_\alpha(j) = r_0 + \sum_{i=1}^{m} r_i p_i + \sum_{i,j} r_{i,j} a_{i,j} + \sum_{i,j} r'_{i,j} b_{i,j}.$$

Proof. The assertion holds for $j = 1$ by definition as $e_\alpha(1) \in \mathbb{Q} \cup \{a_{11}, \dots, a_{22}, b_{11}, \dots, b_{22}\}$. Assume that the assertion hols for all steps smaller than j. If $e_\alpha(j) \in \mathbb{Q} \cup \{a_{11}, \dots, a_{22}, b_{11}, \dots, b_{22}\}$ or $e_\alpha(j) = p_k$ for some k, then the assertion holds evidently. Else if $e_\alpha(j) = e_\alpha(j_1) \pm e_\alpha(j_2)$ where $j_1, j_2 < j$, then the assertion also holds as both $e_\alpha(j_1)$ and $e_\alpha(j_2)$ are of the form $r_0 + \sum_{i=1}^{m} r_i p_i + \sum_{i,j} r_{i,j} a_{i,j} + \sum_{i,j} r'_{i,j} b_{i,j}$. Therefore, the assertion holds for every step k by induction. □

Lemma 2.8. *Let α be an algorithm for computing the product of two 2×2 matrices which has m multiplication steps excluding multiplications by a rational number. Then there exists an algorithm α' requiring only m steps such that the arguments for each multiplication are of the form $\sum_{i,j} r_{i,j} a_{i,j} + \sum_{i,j} r'_{i,j} b_{i,j}$.*

Proof. Let $S = \{a_{11}, \dots, a_{22}, b_{11}, \dots, b_{22}\}$ for the sake of convenience.. Let $u \in \mathbb{Q}[S]$, and $L_i : \mathbb{Q}[S] \to \mathbb{Q}[S]$ be defined for $i = 0, 1, 2, 3$ such that $L_0(u)$ equals the constant term of u, $L_1(u)$ equals the linear term of u, $L_2(u)$ equals the quadratic term of u and $L_3(u) = u - L_0(u) - L_1(u) - L_2(u)$.

Let $e_\alpha(j) = u_1 u_2$. Then, $e_\alpha(j) = L_0(u_1)L_0(u_2) + L_0(u_1)(u_2 - L_0(u_2)) + L_0(u_2)(u_1 - L_0(u_1)) + (u_1 - L_0(u_1))(u_2 - L_0(u_2))$ holds. Since the first term is constant, which is not counted, and the next two terms are multiplication by a rational number, which are also not counted, we can obtain $e_\alpha(j)$ by computing one multiplication $(u_1 - L_0(u_1))(u_2 - L_0(u_2))$ and additionally computing other three terms. Thus, we can obtain an algorithm α^* which has the same number of multiplications as α and $L_0(u_1) = L_0(u_2) = 0$ if $e_{\alpha^*}(j) = u_1 u_2$.

By Lemma 2.7, $\sum_{k=1}^{2} a_{ik} b_{kj} = r_0 + \sum_{i=1}^{m} r_i p_i + \sum_{i,j} r_{i,j} a_{i,j} + \sum_{i,j} r'_{i,j} b_{i,j}$ for $1 \le i, j \le 2$. By applying L_2 to both sides, we obtain $\sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{i=1}^{m} r_i L_2(p_i)$. Assuming $p_i = u_i u'_i$, we obtain

$$\sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{i=1}^{m} r_i L_1(u_i) L_1(u'_i)$$

since $L_0(u_1) = L_0(u_2) = 0$ if $e_{\alpha^*}(j) = u_1 u_2$.

To construct a desired algorithm α', first compute $L_1(u_i)$ and $L_1(u'_i)$ for $i = 1, 2, \dots, m$. No multiplication counted is required in this step. Then compute $L_1(u_i)L_1(u'_i)$'s using m counted multiplications, and finally we have $\sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{i=1}^{m} r_i L_1(u_i) L_1(u'_i)$ with additions and multiplications by a rational number, which are not counted. □

On the other hand, we need the following lemma from [13] to move forward. Specifically, the inverse of the condition (b) is used in the next lemma following this one.

Lemma 2.9 (Winograd)**.** *Let $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$ be two disjoint sets of indeterminates and $k \in \mathbb{Z}_+$. Let $\psi_i = \sum_{j=1}^{n} \phi_{i,j} x_j$ where $\phi_{i,j} \in \mathbb{Q}(y_1, \dots, y_m)$ for $1 \le i \le k$. If*
(a) *$\exists\ j$ such that $\phi_{i,j} \notin \mathbb{Q}$ for every i,*
(b) *no nontrivial linear combination of $\phi_{i',j}$ is in $\mathbb{Q}$, where i' ranges over all indices such that $\phi_{i,j} \notin \mathbb{Q}$ for every j,*
then every algorithm which computes ψ's has at least n multiplications/divisions. Moreover it still holds even if we only count multiplications between a and b such that $a, b \notin \mathbb{Q}$ and either $a \notin \mathbb{Q}(y_1, \dots, y_m)$ or $b \notin \mathbb{Q}(y_1, \dots, y_m)$.

Proof. See [13].

□

Lemma 2.10. *Let* $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ *and* $(\alpha_1', \alpha_2', \alpha_3', \alpha_4')$ *be two linearly independent vectors in* $\mathbb{Q}^4$*. Then any algorithm which computes both* f_1 *and* f_2 *such that*

$$f_1 = a_{11}(\alpha_1 b_{11} + \alpha_2 b_{12}) + a_{12}(\alpha_1 b_{21} + \alpha_2 b_{22}) + a_{21}(\alpha_3 b_{11} + \alpha_4 b_{12}) + a_{22}(\alpha_3 b_{21} + \alpha_4 b_{22}),$$
$$f_2 = a_{11}(\alpha_1' b_{11} + \alpha_2' b_{12}) + a_{12}(\alpha_1' b_{21} + \alpha_2' b_{22}) + a_{21}(\alpha_3' b_{11} + \alpha_4' b_{12}) + a_{22}(\alpha_3' b_{21} + \alpha_4' b_{22}).$$

requires at least 4 *multiplications.*

Proof. We can rewrite f_1 and f_2 as follows:

$$f_3 = f_1 = b_{11}(\alpha_1 a_{11} + \alpha_3 a_{21}) + b_{12}(\alpha_2 a_{11} + \alpha_4 a_{21}) + b_{21}(\alpha_1 a_{12} + \alpha_3 a_{22}) + b_{22}(\alpha_2 a_{12} + \alpha_4 a_{22}),$$
$$f_4 = f_2 = b_{11}(\alpha_1' a_{11} + \alpha_3' a_{21}) + b_{12}(\alpha_2' a_{11} + \alpha_4' a_{21}) + b_{21}(\alpha_1' a_{12} + \alpha_3' a_{22}) + b_{22}(\alpha_2' a_{12} + \alpha_4' a_{22}).$$

Assuming that only 3 multiplications are needed to computed $f_1, f_2, f_3\ (= f_1)$ and $f_4\ (= f_2)$, then by Lemma 2.9 both sets of vectors

$$\left\{ \begin{bmatrix} \alpha_1 b_{11} + \alpha_2 b_{12} \\ \alpha_1' b_{11} + \alpha_2' b_{12} \end{bmatrix}, \begin{bmatrix} \alpha_1 b_{21} + \alpha_2 b_{22} \\ \alpha_1' b_{21} + \alpha_2' b_{22} \end{bmatrix}, \begin{bmatrix} \alpha_3 b_{11} + \alpha_4 b_{12} \\ \alpha_3' b_{11} + \alpha_4' b_{12} \end{bmatrix}, \begin{bmatrix} \alpha_3 b_{11} + \alpha_4 b_{12} \\ \alpha_3' b_{11} + \alpha_4' b_{12} \end{bmatrix} \right\},$$
$$\left\{ \begin{bmatrix} \alpha_1 a_{11} + \alpha_3 a_{21} \\ \alpha_1' a_{11} + \alpha_3' a_{21} \end{bmatrix}, \begin{bmatrix} \alpha_2 a_{11} + \alpha_4 a_{21} \\ \alpha_2' a_{11} + \alpha_4' a_{21} \end{bmatrix}, \begin{bmatrix} \alpha_1 a_{12} + \alpha_3 a_{22} \\ \alpha_1' a_{12} + \alpha_3' a_{22} \end{bmatrix}, \begin{bmatrix} \alpha_2 a_{12} + \alpha_4 a_{22} \\ \alpha_2' a_{11} + \alpha_4' a_{22} \end{bmatrix} \right\}$$

are linearly dependent as a contrary to condition (b).

There exist c_1, c_2, c_3 and c_4 such that

$$c_1 \begin{bmatrix} \alpha_1 b_{11} + \alpha_2 b_{12} \\ \alpha_1' b_{11} + \alpha_2' b_{12} \end{bmatrix} + c_2 \begin{bmatrix} \alpha_1 b_{21} + \alpha_2 b_{22} \\ \alpha_1' b_{21} + \alpha_2' b_{22} \end{bmatrix} + c_3 \begin{bmatrix} \alpha_3 b_{11} + \alpha_4 b_{12} \\ \alpha_3' b_{11} + \alpha_4' b_{12} \end{bmatrix} + c_4 \begin{bmatrix} \alpha_3 b_{11} + \alpha_4 b_{12} \\ \alpha_3' b_{11} + \alpha_4' b_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$= b_{11} \begin{bmatrix} c_1\alpha_1 + c_3\alpha_3 \\ c_1\alpha_1' + c_3\alpha_3' \end{bmatrix} + b_{12} \begin{bmatrix} c_1\alpha_2 + c_3\alpha_4 \\ c_1\alpha_2' + c_3\alpha_4' \end{bmatrix} + b_{21} \begin{bmatrix} c_2\alpha_1 + c_4\alpha_3 \\ c_2\alpha_1' + c_4\alpha_3' \end{bmatrix} + b_{22} \begin{bmatrix} c_2\alpha_2 + c_4\alpha_4 \\ c_2\alpha_2' + c_4\alpha_4' \end{bmatrix},$$

and then the following equations hold:

$$c_1 \begin{bmatrix} \alpha_1 \\ \alpha_1' \end{bmatrix} + c_3 \begin{bmatrix} \alpha_3 \\ \alpha_3' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad c_1 \begin{bmatrix} \alpha_2 \\ \alpha_2' \end{bmatrix} + c_3 \begin{bmatrix} \alpha_4 \\ \alpha_4' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Likewise, there exists d_1 and d_2 such that

$$d_1 \begin{bmatrix} \alpha_1 \\ \alpha_1' \end{bmatrix} + d_2 \begin{bmatrix} \alpha_2 \\ \alpha_2' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad d_1 \begin{bmatrix} \alpha_3 \\ \alpha_3' \end{bmatrix} + d_2 \begin{bmatrix} \alpha_4 \\ \alpha_4' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Therefore, the space spanned by

$$\begin{bmatrix} \alpha_1 \\ \alpha_1' \end{bmatrix}, \begin{bmatrix} \alpha_2 \\ \alpha_2' \end{bmatrix}, \begin{bmatrix} \alpha_3 \\ \alpha_3' \end{bmatrix}, \begin{bmatrix} \alpha_4 \\ \alpha_4' \end{bmatrix}$$

is one-dimensional, which is contradictory to the assumption that $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ and $(\alpha_1', \alpha_2', \alpha_3', \alpha_4')$ are linearly independent.

□

Now we are ready to present the theorem about the lower bound of the number of field multiplications in 2×2 matrix multiplication.

Theorem 2.11 (Winograd)**.** *Every algorithm for multiplying two* 2×2 *matrices requires at least* 7 *field multiplications excluding multiplications by a rational number.*

Proof. Assume two 2×2 matrices can be multiplied with 6 field multiplications. Then there exists products $p_1, p_2, \ldots, p_6$ and $r_{i,j}^k \in \mathbb{Q}$ for $1 \le i, j \le 2$ and $k = 1, 2, \ldots, 6$ such that

$$c_{i,j} = \sum_{k=1}^{2} a_{ik} b_{kj} = \sum_{k=1}^{6} r_{i,j}^k p_k$$

.

Let R, p and c be defined by

$$R = \begin{pmatrix} r_{1,1}^1 & r_{1,1}^2 & \cdots & r_{1,1}^6 \\ r_{1,2}^1 & \ddots & & \\ r_{2,1}^1 & & \ddots & \\ r_{2,2}^1 & & & r_{2,2}^6 \end{pmatrix},\ p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_6 \end{bmatrix},\ c = Rp \begin{bmatrix} c_{1,1} \\ c_{1,2} \\ c_{2,1} \\ c_{2,2} \end{bmatrix}.$$

Since the rank of R is 4, as there is no nontrivial linear combination of $c_{i,j}$'s, there is a 4×4 matrix $D = (d_{i,j})_{1 \le i,j \le 4}$ such that

$$R' = DR = \begin{pmatrix} 1 & 0 & 0 & 0 & r_1 & s_1 \\ 0 & 1 & 0 & 0 & r_2 & s_2 \\ 0 & 0 & 1 & 0 & r_3 & s_3 \\ 0 & 0 & 0 & 1 & r_4 & s_4 \end{pmatrix}.$$

Let $c' = (c'_{1,1}, c'_{1,2}, c'_{2,1}, c'_{2,2})^T = R'p = Dc$. Then while

$$\begin{aligned} c'_{i,j} = a_{1,1}(d_{m,1}b_{1,1} + d_{m,2}b_{1,2}) + a_{1,2}(d_{m,1}b_{2,1} + d_{m,2}b_{2,2}) \\ +a_{2,1}(d_{m,3}b_{1,1} + d_{m,4}b_{1,2}) + a_{2,2}(d_{m,3}b_{2,1} + d_{m,4}b_{2,2}) \end{aligned}$$

where $m = 2i + j - 2$, each $c'_{i,j}$ can be computed with 3 multiplications due to the the form of R'. It means that $\{d_{m,1}b_{1,1} + d_{m,2}b_{1,2}, d_{m,1}b_{2,1} + d_{m,2}b_{2,2}, d_{m,3}b_{1,1} + d_{m,4}b_{1,2}, d_{m,3}b_{2,1} + d_{m,4}b_{2,2}\}$ is a linearly dependent set and we have

$$\begin{vmatrix} d_{m,1} & d_{m,2} \\ d_{m,3} & d_{m,4} \end{vmatrix} = 0. \tag{2.1}$$

By Lemma 2.10, at most one of the r_i's can be 0 and at most one of the s_i's can be 0, as otherwise two of $c'_{i,j}$'s can be computed with 3 multiplications, which is contradictory to the lemma. Without loss of generality, assume $r_1, r_2, r_3 \neq 0$. Then either

$$\begin{pmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \\ d_{3,1} & d_{3,2} \end{pmatrix} = 0 \quad \text{or} \quad \begin{pmatrix} d_{1,3} & d_{1,4} \\ d_{2,3} & d_{2,4} \\ d_{3,3} & d_{3,4} \end{pmatrix} = 0$$

as D is invertible. Again without loss of generality, assume the former equation holds and $(d_{1,1}, d_{1,2})$ and $(d_{2,1}, d_{2,2})$ are linearly independent. We can also assume that $(d_{3,1}, d_{3,2}) \neq 0$, as otherwise replacing D by

$$D' = \frac{1}{s_1} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -s_2 & s_1 & 0 & 0 \\ -s_3 & 0 & s_1 & 0 \\ -s_4 & 0 & 0 & s_1 \end{pmatrix} D$$

is sufficient. For the last we assume $(d_{1,1}, d_{1,2})$ and $(d_{3,1}, d_{3,2})$ are linearly independent without loss of generality.

By (2.1), we have $(d_{m,3}, d_{m,4}) = w_m(d_{m,1}, d_{m,1})$ for $m = 1, 2, 3$. On the other hand, since each $c'_{i,j}$ can be computed with 3 multiplications, so are $c'_{1,2} - \frac{s_2}{s_1}c'_{1,1}$ and $c'_{2,1} - \frac{s_3}{s_1}c'_{1,1}$. Then again by (2.1),

$$\begin{vmatrix} d_{2,1} - \frac{s_2}{s_1}d_{1,1} & d_{2,2} - \frac{s_2}{s_1}d_{1,2} \\ d_{2,3} - \frac{s_2}{s_1}d_{1,3} & d_{2,4} - \frac{s_2}{s_1}d_{1,4} \end{vmatrix} = 0 \quad \text{and} \quad \begin{vmatrix} d_{3,1} - \frac{s_3}{s_1}d_{1,1} & d_{3,2} - \frac{s_3}{s_1}d_{1,2} \\ d_{3,3} - \frac{s_3}{s_1}d_{1,3} & d_{3,4} - \frac{s_3}{s_1}d_{1,4} \end{vmatrix} = 0. \tag{2.2}$$

The left equation of (2.2) holds *iff* $w_1 = w_2$ and the right equation of (2.2) holds *iff* $w_1 = w_3$, making $w_1 = w_2 = w_3$. However, if $(d_{3,1}, d_{3,2}) = \alpha(d_{1,1}, d_{1,2}) + \beta(d_{2,1}, d_{2,2})$, then $(d_{3,3}, d_{3,4}) = \alpha(d_{1,3}, d_{1,4}) + \beta(d_{2,3}, d_{2,4})$, meaning that D is not invertible. Thus, we obtained a contradiction from the assumption that two 2×2 matrices can be multiplied with 6 multiplications.

□

Finally, we have the following desired theorem from Corollary 2.2 and Theorem 2.11.

Theorem 2.12. *Strassen's algorithm is optimal in multiplying two 2×2 matrices in terms of the number of ring multiplications.*

3. Experimental Results[1]

In this section, we analyze the performance of classical and Strassen's matrix multiplication on CPU and GPU. We firstly find out the theoretical threshold of Strassen's algorithm and then check running time of two algorithms with various data types. Basic data types `float`, `double` along with polynomials are used. For multi-core library, OpenMP 4.0 [4] and CUDA 9.0 [8] are used for CPU and GPU, respectively.

Experiments are conducted on a single computer with Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz and GeForce GTX TITAN X. Running time is checked via `omp_get_wtime()` on CPU and `cudaEvent_t` on GPU and thresholds are chosen with coarse hyperparameter search. Results are averaged over 50 experiments.

For the sake of convenience, we assume that we want $C = AB$.

3.1. Theoretical Threshold of Strassen's.

Here we try to find the threshold for Strassen's algorithm to switch to classical matrix multiplication. We assume that the operation complexities of the addition and multiplication are the same.

Given two $n \times n$ matrices, the number of operations required for classical matrix multiplication and Strassen's algorithm with recursion depth of 1 are as follows:

$$T_{classical}(n) = 2n^3,$$
$$T_{Strassen}(n) = \frac{7}{4}n^3 + \frac{9}{2}n^2.$$

Solving a simple inequality $T_{Strassen}(n) \leq T_{classical}(n)$, we have $n \geq 18$. Therefore, the theoretical threshold for Strassen's algorithm is 32, as n should be the power of 2.

3.2. Experiment on CPU.

Before we start, one thing to note is that 2-D array is stored in memory in row-major order in C++. Thus, while accessing $(i, j+1)$ after (i, j) and $(i+1, j)$ after (i, j) are logically same, the former has a higher cache hit rate than the latter and is way efficient. Classical matrix multiplication in this section for primitive data types are implemented with cache-friendly by fixing one element in B and iterating elements of A and C horizontally; naive implementation is used for polynomials.

Table 1 shows experimental results of matrix multiplication with primitive data types on CPU. Thresholds for Strassen's algorithm are 128 in both cases.

Dimension		128	256	512	1024	2048
`float`	**Classical**	164.94 μs	1.1594 ms	8.6097 ms	83.190 ms	788.79 ms
	Strassen	-	1.0959 ms	8.1365 ms	55.947 ms	407.45 ms
`double`	**Classical**	345.64 μs	2.2565 ms	18.417 ms	169.08 ms	1.6997 s
	Strassen	-	2.2276 ms	14.674 ms	107.99 ms	785.33 ms

Table 1. Running time on CPU with primitive data types

Theoretically the number of operations of classical matrix multiplications grows by a factor of 8 and that of Strassen's matrix multiplication grows by a factor of 7 as the dimension of matrices doubles. However, we can see that growth rates not always around 8 and 7 in classical and Strassen's matrix multiplications in these cases, respectively. On the other hand, it should be mentioned that the threshold for Strassen's algorithm with primitive data types are quite practical.

Table 2 shows experimental results of matrix multiplications with `Polynomial` data type on CPU. `Polynomial` is an implementation of $R[x]/(x^n - 1)$ via `std::array` in C++, with R replaced

[1]Source code used for the experiment is available on https://github.com/ijleesw/matmul-omp-cuda.

by `float` or `double`. n is set to 4 and 16 to put more weight on addition and even more on multiplication. Thresholds are set to 64 when $n = 4$ and to 16 when $n = 16$.

Dimension		32	64	128	256	512
`float`, $n = 4$	**Classical**	44.455 μs	371.68 μs	2.4761 ms	18.400 ms	162.16 ms
	Strassen	-	-	2.1866 ms	15.123 ms	103.89 ms
`double`, $n = 4$	**Classical**	48.045 μs	545.51 μs	3.3183 ms	27.063 ms	230.39 ms
	Strassen	-	-	3.2832 ms	21.954 ms	150.97 ms
`float`, $n = 16$	**Classical**	2.1214 ms	15.682 ms	124.34 ms	991.93 ms	-
	Strassen	1.8765 ms	12.470 ms	85.546 ms	623.13 ms	-
`double`, $n = 16$	**Classical**	2.1615 ms	16.629 ms	132.68 ms	1.0627 s	-
	Strassen	1.8993 ms	13.132 ms	91.922 ms	637.92 ms	-

TABLE 2. Running time on CPU with polynomial data types

As aforementioned, more weights are put on operations when dealing with polynomials. Since multiplication needs more base ring operations than addition in this case, thresholds for Strassen's algorithm gets smaller as the degree of the generator of the ideal increases, even lower than the theoretical threshold if the degree is high enough. It can be easily seen that the dimension of matrices that Strassen's algorithm gets effective is relatively low compared to the case with primitive data types, making the use of the algorithm more appealing. Also as the degree increases, running times of the algorithm grow almost precisely by a factor of 8 and 7.

3.3. **Experiment on GPU.** GPU consists of global memory, L1 and L2 cache, shared memory and multiple streaming multiprocessors(SMs) which consists of multiple threads. Shared memory, which can be accessed by all threads in one SM, plays a crucial role in GPU performance like cache in CPU performance. Thus if we divide A and B into blocks and store the result of multiplications of these blocks in shared memory, time consumption of memory access decreases. Classical matrix multiplications using `float` are implemented with this method, while naive implementation is used for the rest of data types.

Table 3 shows experimental results of matrix multiplication with primitive data types on GPU. Thresholds are set to 1024 in both cases.

Dimension		256	512	1024	2048	4096	8192
`float`	**Classical**	48.18 μs	302.64 μs	2.004 ms	17.182 ms	148.32 ms	1.2485 s
	Strassen	-	-	-	17.124 ms	126.02 ms	958.83 ms
`double`	**Classical**	263.05 μs	1.6514 ms	11.147 ms	88.416 ms	747.89 ms	5.6286 s
	Strassen	-	-	-	76.319 ms	598.50 ms	4.3077 s

TABLE 3. Running time on GPU with primitive data types

First to mention is that due to high parallelism of GPU, the running time of the algorithm on GPU is remarkably short comparing to that of CPU. Also, due to that high parallelism, threshold for Strassen's algorithm is high. On the other hand, similar to Table 1, we can see that growth rates are fluctuating, not being precisely 8 or 7.

Table 4 shows experimental results of matrix multiplication with `Polynomial` data type on GPU. The highest dimensional vectors that CUDA supports are 4-dim data types (`float4`, `double4`), so polynomials are set to $R[x]/(x^4 - 1)$ only with R replaced by `float` or `double`. Thresholds are set to 512 with $R =$`float` and 128 with $R =$`double`.

Now we can see that the thresholds are lower than the previous case, which means that Strassen's algorithm can be effective even on GPU. One interesting thing to note is that while the running time tend to grow precisely a factor of 8 or 7 when CPU is used with polynomial data types, it is not the case when GPU is used with polynomials.

Dimension		256	512	1024	2048	4096
`float`, $n = 4$	**Classical**	171.61 μs	1.2576 ms	9.0476ms	69.209 ms	591.48 ms
	Strassen	-	-	8.7405 ms	66.204 ms	545.84 ms
`double`, $n = 4$	**Classical**	3.6098 ms	22.020 ms	174.25ms	1.4847 s	-
	Strassen	3.0133 ms	20.613 ms	148.28 ms	1.1590 s	-

TABLE 4. Running time on GPU with polynomial data types

4. Conclusion

So far we have checked theorems about the complexity and optimality of Strassen's algorithm. In addition we have obtained some experimental results of the algorithm on CPU and GPU. Strassen's algorithm, which computes the multiplication of two 2×2 matrices with 7 ring multiplications, can compute multiplication of two $n \times n$ matrices with the computation complexity of $O(n^{\log_2 7})$ by recursion while classical matrix multiplication needs $O(n^3)$ operations. We also showed that Strassen's algorithm is optimal in terms of the number of ring multiplications by proving that the multiplication of two 2×2 matrices requires at least 7 ring multiplications.

Experimental results show that while multi-core architectures are good with dealing matrix multiplications over primitive data types, Strassen's algorithm beat classical matrix multiplication when the data type gets complex. $R[x]/(x^4-1)$ with R replaced by `float` or `double` had thresholds of 64 on CPU and 512/128 on GPU, and $R[x]/(x^{16}-1)$ with R replaced by `float` or `double` had thresholds of 16 on CPU. Also Strassen's algorithm is faster than the classical one on both CPU and GPU even with primitive data types if the dimension of the matrices is high enough. Both `float` and `double` had thresholds of 128 on CPU and 1024 on GPU.

For future work, it would be nice to explain Strassen's algorithm in a nice and elegant way. The algorithm's formula is complicated and seems like it came out from nowhere, so explaining it will yield a high contribution. Also, conducting experiments with more diverse data types on various architectures will be useful.

References

1. Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti, *$O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication*, Inf. Process. Lett. **8** (1979), 234–235.
2. James W. Cooley and John W. Tukey, *An Algorithm for the Machine Calculation of Complex Fourier Series*, Math. Comput. **19** (1965), 297–301.
3. Don Coppersmith and Shmuel Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation **9** (1990), no. 3, 251–280.
4. Leonardo Dagum and Ramesh Menon, *Openmp: An industry-standard api for shared-memory programming*, IEEE Comput. Sci. Eng. **5** (1998), no. 1, 46–55.
5. François Le Gall, *Powers of tensors and fast matrix multiplication*, CoRR **abs/1401.7714** (2014).
6. Joachim von zur Gathen, *Modern computer algebra / joachim von zur gathen, juÌrgen gerhard.*, 3rd ed.. ed., 2013 (eng).
7. Anatolii Karatsuba and Ofman Yu, *Multiplication of many-digital numbers by automatic computers*, Dokl. Akad. Nauk SSSR **145** (1962), no. 2, 293–294.
8. John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron, *Scalable parallel programming with cuda*, Queue **6** (2008), no. 2, 40–53.
9. V. Y. Pan, *Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations*, 19th Annual Symposium on Foundations of Computer Science (sfcs 1978), Oct 1978, pp. 166–176.
10. A. Schönhage, *Partial and total matrix multiplication*, SIAM Journal on Computing **10** (1980), no. 3, 434–455.
11. A. Schönhage and V. Strassen, *Schnelle multiplikation großer zahlen*, Computing **7** (1971), no. 3, 281–292.
12. Volker Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik **13** (1969), no. 4, 354–356.
13. Shmuel Winograd, *On the number of multiplications necessary to compute certain functions*, Communications on Pure and Applied Mathematics **23** (1970), no. 2, 165–179.
14. ———, *On multiplication of 2×2 matrices*, Linear Algebra and Its Applications **4** (1971), no. 4, 381–388.