Matrixmultiplikation von Strassen

Stephan Epp

2. Juli 2025

Beim Algorithmus von Strassen für die Multiplikation von zwei $n \times n$ Matrizen lautet die Rekurrenz zur Ermittlung der Laufzeit $T(n)$ des Algorithmus

$$T(n) = 7\,T(\tfrac{n}{2}) + c\,n^2.$$

Der Algorithmus halbiert in jedem rekursiven Aufruf die beiden $n \times n$ Matrizen zu vier $\frac{n}{2} \times \frac{n}{2}$ Matrizen. Substituieren wir n durch $\frac{n}{2}$, erhalten wir für

$$T(\tfrac{n}{2}) = 7\,T(\tfrac{n}{4}) + c(\tfrac{n}{2})^2 = 7\,T(\tfrac{n}{4}) + \tfrac{c}{4}n^2.$$

Nach dem *ersten* rekursiven Aufruf erhalten wir mit $T(\frac{n}{2})$ eingesetzt in $T(n)$ dann

$$\begin{aligned} T(n) &= 7\,(7\,T(\tfrac{n}{4}) + \tfrac{c}{4}\,n^2) + cn^2 \\ &= 7^2\,T(\tfrac{n}{4}) + \tfrac{7}{4}\,cn^2 + cn^2. \end{aligned}$$

Mit dem *zweiten* rekursiven Aufruf werden die vier $\frac{n}{2} \times \frac{n}{2}$ Matrizen wieder halbiert zu acht $\frac{n}{4} \times \frac{n}{4}$ Matrizen. Damit ist

$$T(\tfrac{n}{4}) = 7\,T(\tfrac{n}{8}) + c(\tfrac{n}{4})^2 = 7\,T(\tfrac{n}{8}) + \tfrac{c}{16}n^2.$$

Wird $T(\frac{n}{4})$ eingesetzt in $T(n)$ ergibt sich

$$\begin{aligned} T(n) &= 7^2\,(7\,T(\tfrac{n}{8}) + \tfrac{c}{16}n^2) + \tfrac{7}{4}cn^2 + cn^2 \\ &= 7^3\,T(\tfrac{n}{2^3}) + \tfrac{7^2}{4^2}cn^2 + \tfrac{7}{4}cn^2 + cn^2. \end{aligned}$$

Betrachten wir nun den k-ten rekursiven Aufruf finden wir für

$$T(n) = 7^k\,T\left(\tfrac{n}{2^k}\right) + cn^2 \sum_{i=0}^{k-1} \left(\frac{7}{4}\right)^i.$$

Zur Vereinfachung belassen wir es bei dem k-ten rekursiven Aufruf auch in $T(n)$ bei k und nicht $k+1$. Kleinere Matrizen als 1×1 Matrizen gibt es nicht, daher können die $n \times n$ Matrizen nur k mal halbiert werden. Der größte Wert, den k annehmen kann, ist $k = \log_2 n$. Damit ist

$$\begin{aligned} T(n) &= 7^{\log_2 n}\,T\Big(\frac{n}{2^{\log_2 n}}\Big) + cn^2 \sum_{i=0}^{\log_2 n - 1} \left(\tfrac{7}{4}\right)^i \\ &= n^{\log_2 7}\,T(1) + cn^2\,\frac{\left(\frac{7}{4}\right)^{\log_2 n} - 1}{\frac{7}{4} - 1} \\ &= O\left(n^{2.8074}\right) + cn^2\left(\tfrac{7}{4}\right)^{\log_2 n} \\ &= O\left(n^{2.8074}\right) + cn^2 \cdot n^{\log_2 \frac{7}{4}} \\ &= O\left(n^{2.8074}\right) + cn^{2.8074} \\ &= O\left(n^{2.8074}\right). \end{aligned}$$

1 Algorithmus

Es folgt der Algorithmus 2 von STRASSEN als Pseudocode. Als Eingabe erhalten wir zwei $n \times n$ Matrizen. Der Einfachheit halber wird angenommen, dass n eine Zweierpotenz ist. Zu Beginn prüfen wir, ob die Größe der Matrizen bereits den Wert 1 hat. Haben die Matrizen den Wert 1, geben wir das Produkt AB zurück. In Zeile 4 und 5 werden die Matrizen A

Algorithmus 1 STRASSEN(A, B)

Eingabe: $\langle A, B \rangle$, mit $n \times n$ Matrizen A, B, $n = 2^k$, $k \in \mathbb{N}$
Ausgabe: $\langle C \rangle$, mit Produktmatrix $C = AB$

1: **if** $n = 1$ **then** $C = AB$
2: **return** C
3: **end if**
4: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$
5: $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$
6: $P_1 = \text{STRASSEN}(A_{11} + A_{22}, B_{11} + B_{22})$
7: $P_2 = \text{STRASSEN}(A_{21} + A_{22}, B_{11})$
8: $P_3 = \text{STRASSEN}(A_{11}, B_{12} - B_{22})$
9: $P_4 = \text{STRASSEN}(A_{22}, B_{21} - B_{11})$
10: $P_5 = \text{STRASSEN}(A_{11} + A_{12}, B_{22})$
11: $P_6 = \text{STRASSEN}(A_{21} - A_{11}, B_{11} + B_{12})$
12: $P_7 = \text{STRASSEN}(A_{12} - A_{22}, B_{21} + B_{22})$
13: $C_{11} = P_1 + P_4 - P_5 + P_7$
14: $C_{12} = P_3 + P_5$
15: $C_{21} = P_2 + P_4$
16: $C_{22} = P_1 - P_2 + P_3 + P_6$
17: **return** $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

und B so definiert, dass in den Zeilen 6 bis 12 die sieben Matrixmultiplikationen jeweils durchgeführt werden. In den Zeilen 13 bis 16 werden 4 Matrizen C_{ij} durch Addition und Subtraktion der Matrizen A_{ij} berechnet und in Zeile 17 Matrix C als Ergebnis zurückgegeben.

Als Idee zum Beweis der Korrektheit betrachten wir das Produkt $A \cdot B$ der zwei Matrizen A und B mit

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{und} \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

Zur Vereinfachung beinhalten die Matrizen nur skalare Werte. Das Ergebnis $C = A \cdot B$ ist

$$C = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}.$$

Der Algorithmus von STRASSEN berechnet P_i mit

$$\begin{aligned}
P_1 &= \text{STRASSEN}(a + d, e + h) && = ae + ah + de + dh \\
P_2 &= \text{STRASSEN}(c + d, e) && = ce + de \\
P_3 &= \text{STRASSEN}(a, f - h) && = af - ah \\
P_4 &= \text{STRASSEN}(d, g - e) && = dg - de \\
P_5 &= \text{STRASSEN}(a + b, h) && = ah + bh \\
P_6 &= \text{STRASSEN}(c - a, e + f) && = ce + cf - ae - af \\
P_7 &= \text{STRASSEN}(b - d, g + h) && = bg + bh - dg - dh.
\end{aligned}$$

Dann werden C_{ij} berechnet mit

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7 && = ae + bg \\ C_{12} &= P_3 + P_5 && = af + bh \\ C_{21} &= P_2 + P_4 && = ce + dg \\ C_{22} &= P_1 - P_2 + P_3 + P_6 && = cf + dh, \end{aligned}$$

wobei z.B. $C_{11} = (ae + ah + de + dh) + (dg - de) - (ah + bh) + (bg + bh - dg - dh) = ae + bg$. Für einen formalen Beweis der Korrektheit verzichten wir auf die vollständige Induktion über $n \in \mathbb{N}$ der $n \times n$ Matrizen.

2 Experimentelle Ergebnisse in Python

Um die theoretische Laufzeitanalyse von Strassens Algorithmus zu überprüfen, wurde eine Python-Implementierung des Algorithmus erstellt und deren Performance mit der einer Standard-Matrixmultiplikation verglichen. Die Experimente wurden für Matrizen unterschiedlicher Größe (n) durchgeführt, wobei n von 4 bis 256 in Schritten von 4 variiert wurde. Für jede Matrixgröße wurden 5 Messungen (Trials) durchgeführt und die durchschnittliche Laufzeit ermittelt. Ein **interner Schwellenwert (threshold)** von 32 wurde

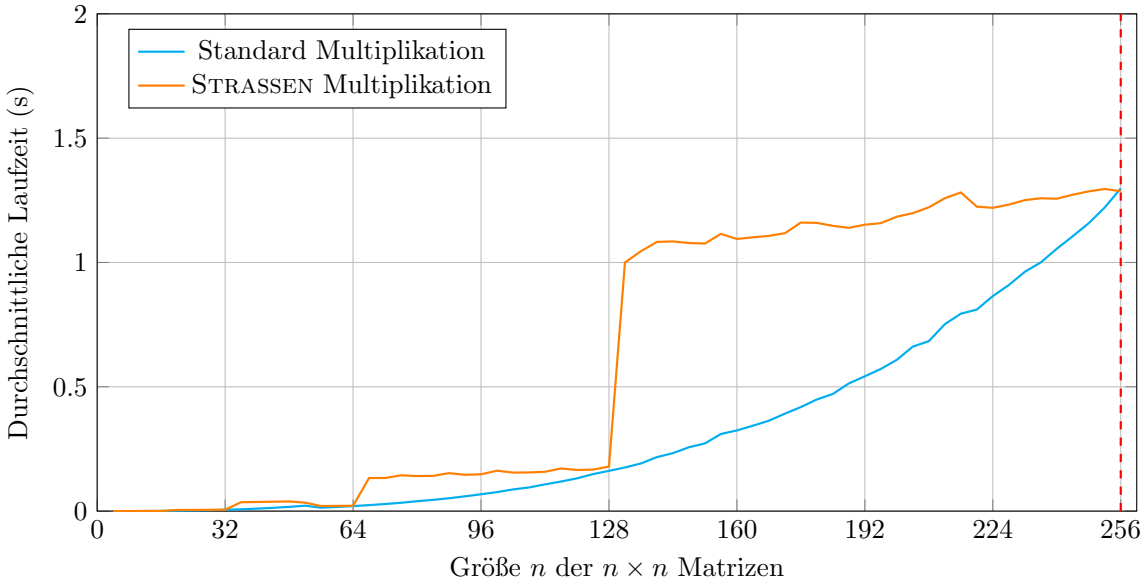
Tabelle 1: Vergleich der Laufzeiten von Standard- und Strassen-Matrixmultiplikation

n	**Standard Avg Time (s)**	**Strassen Avg Time (s)**
4	0.000027	0.000045
8	0.000101	0.000142
12	0.000312	0.000698
16	0.000708	0.000808
20	0.001350	0.004538
24	0.002304	0.004827
28	0.003644	0.005236
⋮	⋮	⋮
52	0.021834	0.034057
56	0.013749	0.020460
60	0.016938	0.020990
⋮	⋮	⋮
232	0.962290	1.250792
236	1.000335	1.258274
240	1.055068	1.256394
244	1.105579	1.272495
248	1.158183	1.285980
252	1.221187	1.295652
256	1.298067	1.286444

für Strassens Algorithmus festgelegt, was bedeutet, dass für Matrizen, deren Größe kleiner oder gleich 32 ist, auf die Standard-Matrixmultiplikation umgeschaltet wird, um den Overhead der Rekursion zu reduzieren. Die Systemauslastung vor Beginn der Experimente betrug 0,0% CPU-Auslastung bei 3,38 GB verwendetem RAM von insgesamt 7,01 GB. Nach Abschluss der Experimente stieg die CPU-Auslastung auf 3,8% und der verwendete RAM auf 3,43 GB, was auf eine moderate Systemauslastung während der Messungen hinweist. Die Messergebnisse sind in Tabelle 1 zusammengefasst.

Aus der Tabelle lässt sich ablesen, dass für kleine Matrizen (z.B. $n \leq 64$) die Standard-Matrixmultiplikation tendenziell schneller war oder eine vergleichbare Performance wie Strassens Algorithmus aufwies. Dies ist auf den **Overhead der Rekursion und der zusätzlichen Matrixadditionen/-subtraktionen** bei Strassens Algorithmus zurückzuführen. Insbesondere ist der Sprung in der Laufzeit des Strassen-Algorithmus bei $n = 36$ (nach dem eingestellten Schwellenwert von $n = 32$) auffällig, was den Wechsel von der optimierten Basis-Multiplikation zur rekursiven Struktur widerspiegelt. Mit zunehmender Matrixgröße, insbesondere ab etwa $n \approx 52 - 56$ und deutlicher ab $n > 128$, zeigt sich, dass Strassens Algorithmus eine geringfügig bessere oder zumindest wettbewerbsfähige Laufzeit im Vergleich zur Standard-Matrixmultiplikation erreicht, was der theoretischen Annahme von $O(n^{\log_2 7})$ gegenüber $O(n^3)$ entspricht. Ab $n = 256$ übertrifft Strassen die Standardmultiplikation leicht. Die praktische Performance wird jedoch stark vom gewählten Schwellenwert und der Effizienz der Implementierung der Basisfälle beeinflusst.

Abbildung 1: Vergleich der Laufzeit der Matrixmultiplikationen



Die Abbildung 1 zeigt die Verläufe der Laufzeit beider Matrixmultiplikationen. Für $n = 56$ liefen sowohl die ursprüngliche als auch Strassens Multiplikation schneller als für vorherige $n < 56 = 7 \cdot 8$. Strassen verwendet 7 Matrixmultiplikationen, d. h., 14 Untermatrizen. Wie wäre es, wenn 15 Untermatrizen bzw. Strukturen für eine schnellere Matrixmultiplikation als die von Strassen verwendet würden? Die Idee besteht darin, 15 Untermatrizen bzw. Strukturen in den beiden $n \times n$ Matrizen zu finden, um Strassens Laufzeit zu verbessern. Betrachte dazu die Diagonale der Matrix B, um nur 5 statt 7 Matrixmultiplikationen durchzuführen, wodurch eine Laufzeit im Exponenten von n von $2,3219$ statt $2,807$ erreicht wird. Betrachtet man $P_2 = (c + d) \cdot e$ und $P_5 = (a + b) \cdot h$, dann fällt auf, dass P_2 durch P_6 und P_1 ermittelt werden kann und P_5 durch P_1 und P_7. Dazu müssen P_1, P_6 und P_7 berechnet werden bevor P_2 und P_5 ohne Multiplikation bestimmt werden können. Da nur 5 Matrixmultiplikationen verwendet werden kann das Ergebnis der Laufzeitanalyse wiederverwendet werden mit dem Unterschied, dass

$$T(n) = 5^{\log_2 n}\, T\Big(\frac{n}{2^{\log_2 n}}\Big) + cn^2 \sum_{i=0}^{\log_2 n - 1} \big(\tfrac{5}{4}\big)^i = n^{\log_2 5}\, T(1) + cn^2\, \frac{\big(\frac{5}{4}\big)^{\log_2 n} - 1}{\frac{5}{4} - 1} = O\left(n^{2.3219}\right).$$

Dazu wird der Algorithmus von Strassen zu Strassen-25 so geändert, dass P_6 und P_7 in Zeile 7 und 8 berechnet werden bevor P_2 und P_5 in Zeile 9 und 10 ermittelt werden.

Algorithmus 2 STRASSEN-25(A, B)

Eingabe: $\langle A, B\rangle$, mit $n \times n$ Matrizen A, B, $n = 2^k$, $k \in \mathbb{N}$
Ausgabe: $\langle C, A_{ij}, B_{ij}\rangle$, mit Produktmatrix $C = AB$ und A_{ij}, B_{ij}

1: **if** $n = 1$ **then** $C = AB$
2: **return** C
3: **end if**
4: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$
5: $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$
6: $P_1 = \text{STRASSEN-25}(A_{11} + A_{22}, B_{11} + B_{22})$, $P_1 = \langle C', A'_{11}, A'_{22}, B'_{11}, B'_{22}\rangle$
7: $P_6 = \text{STRASSEN-25}(A_{21} - A_{11}, B_{11} + B_{12})$, $P_6 = \langle C', A'_{21}, A'_{11}, B'_{11}, B'_{12}\rangle$
8: $P_7 = \text{STRASSEN-25}(A_{12} - A_{22}, B_{21} + B_{22})$, $P_7 = \langle C', A'_{12}, A'_{22}, B'_{21}, B'_{22}\rangle$
9: $P_2 = A'_{21}A'_{22} + A'_{22}B'_{11}$ von P_6, P_1
10: $P_5 = A'_{11}B'_{22} + A'_{12}B'_{22}$ von P_1, P_7
11: $P_3 = \text{STRASSEN-25}(A_{11}, B_{12} - B_{22})$, $P_3 = \langle C', A'_{11}, B'_{12}, B'_{22}\rangle$
12: $P_4 = \text{STRASSEN-25}(A_{22}, B_{21} - B_{11})$, $P_4 = \langle C', A'_{22}, B'_{21}, B'_{11}\rangle$
13: $C_{11} = P_1 + P_4 - P_5 + P_7$
14: $C_{12} = P_3 + P_5$
15: $C_{21} = P_2 + P_4$
16: $C_{22} = P_1 - P_2 + P_3 + P_6$
17: **return** $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Bei der Analyse der gemessenen Laufzeit des in Python implementierten Algorithmus STRASSEN-25(A, B) ist zu erwarten, dass mit 5 Matrixmultiplikationen weniger Laufzeit gemessen wird als mit STRASSEN(A, B).