

Matrizenmultiplikation

Wozu?

- Geometrie: Matrix = lineare Abbildung
Matrixprodukt = Hintereinanderausführung
- Graphen: Matrix = Adjazenzmatrix
Multiplikation z.B. für transitive Hülle
- Signale: Filter, Diskrete Fourier Transformation

Matrizenmultiplikation

Standardalgorithmus:

$$A = (a_{ij}), \quad B = (b_{ij}), \quad C = AB = (c_{ij})$$

wobei: $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$
 n Multiplikationen je Matrixelement und $n-1$ Additionen je Element

Gesamtaufwand ist n^3 Multiplikationen und $n^2(n-1)$ Additionen, also $O(n^3)$

Multiplikation nach Winograd

$$c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} b_{2k-1,j} + a_{i,2k} b_{2k,j})$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} b_{2k-1,j} + a_{i,2k} b_{2k,j} + a_{i,2k-1} a_{i,2k} + b_{2k,j} b_{2k-1,j} - a_{i,2k-1} a_{i,2k} - b_{2k,j} b_{2k-1,j})$$

$$c_{i,j} = \sum_{k=1}^{n/2} ((a_{i,2k-1} + b_{2k,j})(a_{i,2k} b_{2k-1,j}) - a_{i,2k-1} a_{i,2k} - b_{2k,j} b_{2k-1,j})$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} b_{2k-1,j}) - \sum_{k=1}^{n/2} a_{i,2k-1} a_{i,2k} - \sum_{k=1}^{n/2} b_{2k,j} b_{2k-1,j}$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} b_{2k-1,j}) - A_i - B_j$$

Multiplikation nach Winograd

$$c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} b_{2k-1,j} + a_{i,2k} b_{2k,j})$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} b_{2k-1,j} + a_{i,2k} b_{2k,j})$$

$$B_j = \sum_{k=1}^{n/2} b_{2k-1,j} b_{2k,j}$$

$$A_i = \sum_{k=1}^{n/2} a_{i,2k-1} a_{i,2k}$$

$$\sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j}) \cdot (a_{i,2k} + b_{2k-1,j})$$

$$= \sum_{k=1}^{n/2} \underbrace{(a_{i,2k-1} a_{i,2k})}_{A_i} + (a_{i,2k-1} b_{2k-1,j}) + (b_{2k,j} a_{i,2k}) + \underbrace{(b_{2k,j} b_{2k-1,j})}_{B_j}$$

$$= c_{i,j} + A_i + B_j$$

Multiplikation nach Winograd

Aufwandsabschätzung für Winograds Algorithmus:

Berechnung eines $A_i = \sum_{k=1}^{n/2} a_{i,2k-1} a_{i,2k}$ benötigt $\frac{1}{2}n$ Multiplikationen. Ebenso B_j .

A_i und B_j müssen nur einmal berechnet werden => Aufwand für alle A_i und B_j ist n^2 Multiplikationen.

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} b_{2k-1,j}) - A_i - B_j$$

benötigt $n^2 \cdot \frac{n}{2}$ Multiplikationen, insgesamt also $\frac{n^3}{2} + n^2$

Matrixmultiplikation nach Strassen

Entwurf eines Teile-und-herrsche-Algorithmus:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\left(\left[\begin{matrix} A_{11} \\ A_{12} \\ A_{21} \\ A_{22} \end{matrix} \right] + \left[\begin{matrix} A_{12} \\ A_{21} \\ A_{22} \\ A_{11} \end{matrix} \right] + \left[\begin{matrix} A_{21} \\ A_{11} \\ A_{12} \\ A_{22} \end{matrix} \right] + \left[\begin{matrix} A_{22} \\ A_{12} \\ A_{11} \\ A_{21} \end{matrix} \right] \right) \cdot \left(\left[\begin{matrix} B_{11} \\ B_{12} \\ B_{21} \\ B_{22} \end{matrix} \right] + \left[\begin{matrix} B_{12} \\ B_{21} \\ B_{22} \\ B_{11} \end{matrix} \right] + \left[\begin{matrix} B_{21} \\ B_{11} \\ B_{12} \\ B_{22} \end{matrix} \right] + \left[\begin{matrix} B_{22} \\ B_{12} \\ B_{11} \\ B_{21} \end{matrix} \right] \right)$$

$$\left[\begin{matrix} A_{11} B_{11} \\ A_{11} B_{21} \\ A_{12} B_{21} \\ A_{21} B_{21} \end{matrix} \right] + \left[\begin{matrix} A_{11} B_{12} \\ A_{11} B_{22} \\ A_{12} B_{22} \\ A_{21} B_{22} \end{matrix} \right] + \left[\begin{matrix} A_{12} B_{11} \\ A_{12} B_{21} \\ A_{21} B_{11} \\ A_{21} B_{21} \end{matrix} \right] + \left[\begin{matrix} A_{22} B_{11} \\ A_{22} B_{21} \\ A_{11} B_{11} \\ A_{11} B_{21} \end{matrix} \right] + \left[\begin{matrix} A_{22} B_{12} \\ A_{22} B_{22} \\ A_{12} B_{12} \\ A_{12} B_{22} \end{matrix} \right]$$

$$\left[\begin{matrix} C_{11} \\ C_{12} \\ C_{21} \\ C_{22} \end{matrix} \right]$$

Multiplikation nach Strassen

Teile-und-herrsche-Algorithmus macht also aus
 1 Multiplikation von $n \times n$ Matrizen
 8 Multiplikationen von $n/2 \times n/2$ Matrizen
 (wir lösen die Problematik für ungerade n ähnlich wie beim Potenzieren)

Basisfall: Multiplikation zweier 2×2 Matrizen:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Benötigt standardmäßig 8 Multiplikationen

Multiplikation nach Strassen

Aufwandsabschätzung: $T(n) = 8T(n/2) + O(n^2)$
 $= O(n^3)$ für Additionen
 (in jeder Rekursionsebene eine Addition je Element)
 $O(n^2)$ trägt nicht zur Asymptote bei.

Was wäre, wenn wir mit 7 Multiplikationen auskämen?

$$\begin{aligned} T(2) &= 8 + \dots \\ T(4) &= 8 \cdot T(2) + \dots = 64 \\ T(8) &= 8 \cdot T(4) + \dots = 512 \\ T(16) &= 8 \cdot T(8) + \dots = 4096 \\ &= O(n^3) \end{aligned}$$

$$\begin{aligned} T(2) &= 7 + \dots \\ T(4) &= 7 \cdot T(2) + \dots = 49 \\ T(8) &= 7 \cdot T(4) + \dots = 343 \\ T(16) &= 7 \cdot T(8) + \dots = 2401 \\ &= O(n^{\log_2 7}) \approx O(n^{2.81}) \end{aligned}$$

Multiplikation nach Strassen

Unter welchen Umständen < 8 Multiplikationen für 2×2 ?

$$\begin{bmatrix} a & a \\ b & b \end{bmatrix} \cdot \begin{bmatrix} c & e \\ d & f \end{bmatrix} = \begin{bmatrix} a(c+d) & a(e+f) \\ b(c+d) & b(e+f) \end{bmatrix} \quad 4 \text{ Multiplikationen}$$

$$\begin{bmatrix} a & a \\ b & c \end{bmatrix} \cdot \begin{bmatrix} b & a \\ c & a \end{bmatrix} = \begin{bmatrix} a(b+c) & 2a^2 \\ b^2+c^2 & a(b+c) \end{bmatrix} \quad 5 \text{ Multiplikationen}$$

$$\begin{bmatrix} a & a \\ a & a \end{bmatrix} \cdot \begin{bmatrix} a & a \\ a & a \end{bmatrix} = \begin{bmatrix} 2a^2 & 2a^2 \\ 2a^2 & 2a^2 \end{bmatrix} \quad 1 \text{ Multiplikation}$$

Frage: kann $A \cdot B$ als Summe $(A_1 + \dots + A_p) \cdot (B_1 + \dots + B_q)$ anderer Matrizen dargestellt werden, so daß insgesamt weniger Multiplikationen benötigt werden?

Multiplikation nach Strassen

$2 \times 2 \cdot 2 \times 2$ läßt sich auch darstellen als $4 \times 4 \cdot 4 \times 4$, nämlich

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} p & s \\ r & t \end{bmatrix} \Leftrightarrow \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} \cdot \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} p \\ r \\ s \\ t \end{bmatrix}$$

Das spart keine Rechenschritte ein, bietet aber evtl. Ansätze für Optimierungen.

Multiplikation nach Strassen

Besonders einfache Fälle von Matrix-Vektor-Multiplikationen:

$$\alpha \begin{bmatrix} a & a \\ a & a \end{bmatrix} \cdot \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a(e+f) \\ a(e+f) \end{bmatrix} \quad 1 \text{ Multiplikation}$$

$$\beta \begin{bmatrix} a & a \\ -a & -a \end{bmatrix} \cdot \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a(e+f) \\ -a(e+f) \end{bmatrix} \quad 1 \text{ Multiplikation}$$

$$\gamma \begin{bmatrix} a & 0 \\ a-b & b \end{bmatrix} \cdot \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae \\ ae+b(f-e) \end{bmatrix} \quad 2 \text{ Multiplikationen}$$

$$\delta \begin{bmatrix} a & b-a \\ 0 & b \end{bmatrix} \cdot \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a(e-f)+bf \\ bf \end{bmatrix} \quad 2 \text{ Multiplikationen}$$

Multiplikation nach Strassen

2×2 Matrix (z.B. vom Typ γ)

$$\begin{bmatrix} a & 0 \\ a-b & b \end{bmatrix} \cdot \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae \\ ae+b(f-e) \end{bmatrix}$$

läßt sich durch Auffüllen mit Nullen auf 4×4 aufblasen:

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ a-b & 0 & b & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e \\ x \\ f \\ y \end{bmatrix} = \begin{bmatrix} ae \\ 0 \\ ae+b(f-e) \\ 0 \end{bmatrix} \quad \text{auch vom Typ } \gamma, \text{ auch nur 2 Multipl.}$$

Multiplikation nach Strassen

Wir versuchen
$$\begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

zu schreiben als Summe von Matrixprodukten mit Matrizen nur der Typen α, β, γ oder δ , so daß die Zahl der Multiplikationen unter 8 bleibt, z.B:

$$\alpha + \alpha + \beta + \gamma + \delta$$

würde

$$1 + 1 + 1 + 2 + 2 = 7$$

Multiplikationen benötigen.

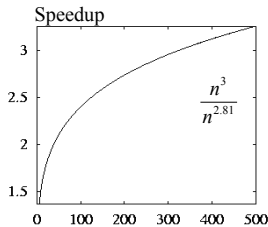
Multiplikation nach Strassen

$$\begin{bmatrix} b & b & 0 & 0 \\ b & b & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{\alpha} + \begin{bmatrix} a-b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c-b & 0 & a-c & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{\gamma} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ c-b & 0 & 0 & c-b \\ b-c & 0 & 0 & b-c \\ 0 & 0 & 0 & 0 \end{bmatrix}_{\beta}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & c & c \\ 0 & 0 & c & c \end{bmatrix}_{\alpha} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & d-b & 0 & b-c \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & d-c \end{bmatrix}_{\delta} = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

Multiplikation nach Strassen

Strassens Algorithmus bringt nur für sehr große Matrizen eine nennenswerte Verbesserung:



Praktische Erfahrung:

Strassen hat soviel Overhead, daß erst ab ca. $n=100$ eine Verbesserung zum Standard eintritt.

Multiplikation nach Strassen

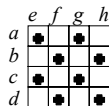
Weitere Nachteile von Strassens Algorithmus:

- geringere numerische Stabilität (viele Rechenoperationen und Rundungen => Fehler)
- schlechtere Parallelisierbarkeit
- erhöhter Programmieraufwand (Verschwendung von Informatikerzeit)

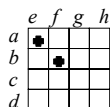
Matrixmultiplikation nach Strassen

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix}$$

Markieren wir die benötigten Produkte in einer Matrix:



Wir können so auch andere Terme darstellen, z.B.

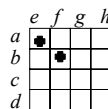


$$ae+bf=r$$

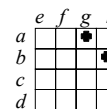
Matrixmultiplikation nach Strassen

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix}$$

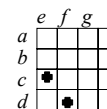
Standardalgorithmus:



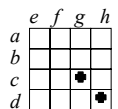
$$ae+bf=r$$



$$ag+bh=s$$



$$ce+df=t$$



$$cg+dh=u$$

4 mal 2 = 8 Multiplikationen

Matrixmultiplikation nach Strassen

Andere Terme (mit je einer Multiplikation) könnten z.B. sein:

	e	f	g	h
a				+
b				
c				
d				

$$(a+b)h$$

	e	f	g	h
a			+	-
b				
c				
d				

$$a(g-h)$$

	e	f	g	h
a				
b				
c				
d	-	+		

$$d(f-e)$$

Fragestellung: Ist es möglich, Terme zu finden, die alle benötigten Produkte abdecken, aber zusammen weniger als 8 Multiplikationen benötigen?

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung
 $P_1 = ag - ah = a(g - h)$ 1

	e	f	g	h
a			+	-
b				
c				
d				

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$P_2 = ah + bh = (a + b)h$$

1

	e	f	g	h
a				+
b				+
c				
d				

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$-P_2 = -ah - bh$$

0

	e	f	g	h
a				-
b				-
c				
d				

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$P_3 = ce + de = (c + d)e$$

1

	e	f	g	h
a				
b				
c	+			
d	+			

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$-P_3 = -ce - de$$

0

	e	f	g	h
a				
b				
c	-			
d	-			

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$P_4 = df - de = d(f - e) \quad 1$$

	e	f	g	h
a				
b				
c				
d	-	+		

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$P_5 = ae + ah + de + dh = (a + d)(e + h) \quad 1$$

	e	f	g	h
a	+			+
b				
c				
d	+			+

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$P_6 = bf + bh - df - dh = (b - d)(f + h) \quad 1$$

	e	f	g	h
a				
b		+		+
c				
d		-		-

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$P_7 = ae + ag - ce - cg = (a - c)(e + g) \quad 1$$

	e	f	g	h
a	+		+	
b				
c	-		-	
d				

Matrixmultiplikation nach Strassen

Term Multiplikationen Darstellung

$$-P_7 = -ae - ag + ce + cg \quad 0$$

	e	f	g	h
a	-		-	
b				
c	+		+	
d				

Matrixmultiplikation nach Strassen

$$r = ae + bf = P_5 + P_4 - P_2 + P_6$$

Matrixmultiplikation nach Strassen

$$s = ag + bh = P_1 + P_2$$

Matrixmultiplikation nach Strassen

$$t = ce + df = P_3 + P_4$$

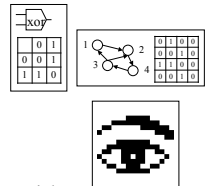
Matrixmultiplikation nach Strassen

$$u = cg + dh = P_5 + P_1 - P_3 - P_7$$

Rechnen mit Booleschen Matrizen

Boolesche Matrizen sind z.B:

- Schalttafeln für elektronische Gatter
- Adjazenzmatrizen für Graphen
- schwarz-weiße Pixelbilder
- Definitionen beliebiger Relationen
- etc.



Multiplikation von Booleschen Matrizen entspricht Hintereinanderausführung der durch sie definierten Relation.

Erinnern: Berechnung der transitiven Hülle eines Graphen.

Rechnen mit Booleschen Matrizen

Boolesche Operatoren und Grundrechenarten:

AND entspricht Multiplikation

OR entspricht Addition

·	0	1
0	0	0
1	0	1

+	0	1
0	0	1
1	1	1

Was aber ist mit Subtraktion?

0+1=1 und 1+1=1

Multiplikation Boolescher Matrizen

bei transitiver Hülle hatten wir: $(i, j) \in H \Leftrightarrow A_j^n \neq 0$

Kante (i, j) ist in Hülle, wenn die n -te Potenz der Adjazenzmatrix (mit Elementen aus \mathbb{N}) an der stelle i, j nicht 0 ist.

D.h. wir können die Booleschen Werte 0 und 1 wie natürliche Zahlen behandeln, aber interpretieren am Ende

=0 ist falsch
≠0 ist wahr

Leichte Modifikation für Strassen-Algorithmus: Rechne im Ring Modulo $(n+1)$, d.h. kein Überlauf, Subtraktion geht. Also obere Schranke ist $O(n^{2.81})$.

Multiplikation Boolescher Matrizen

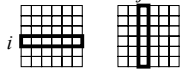
Gesehen: Boolesche Matrizenmultiplikation kann mit Strassen-Algorithmus in $O(n^{2.81})$ berechnet werden.

Aber Rechnen mit Natürlichen Zahlen statt Bits ist Overkill.

Fragen: Kann man irgendwie ausnutzen, daß die Operationen mit Bits und Bitvektoren einfacher zu machen sind?

Kann man ausnutzen, daß fast alle CPUs mehrere Bits auf einmal manipulieren können?

Multiplikation Boolescher Matrizen

Multiplikationsschema: $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$ 

D.h. c_{ij} ist das Skalarprodukt zweier Vektoren

$$A_i = (a_{i1}, \dots, a_{in}) \text{ und } B_j = (b_{1j}, \dots, b_{nj})^T$$

auch genannt inneres Produkt (inner product, dot product).

In Booleschen Operationen ausgedrückt:

$$p_k = a_{ik} b_{kj} \quad c_{ij} = \begin{cases} 0 & \text{falls } \forall k : p_k = 0 \\ 1 & \text{falls } \exists k : p_k = 1 \end{cases}$$

Multiplikation Boolescher Matrizen

Angenommen, die Wortbreite (Anzahl gleichzeitig verarbeitbarer Bits) unserer CPU sei mindestens n , dann können wir

$$p_k = a_{ik} b_{kj} \quad c_{ij} = \begin{cases} 0 & \text{falls } \forall k : p_k = 0 \\ 1 & \text{falls } \exists k : p_k = 1 \end{cases}$$

durch ein AND berechnen

durch einen Vergleich mit 0 berechnen

$$\text{also } P = A_i \text{ AND } B_j \quad c_{ij} = \begin{cases} 0 & \text{falls } P = 0 \\ 1 & \text{falls } P \neq 0 \end{cases}$$

d.h. ein inneres Produkt wäre in $O(1)$ berechenbar.

Multiplikation Boolescher Matrizen

Es ist zu erwarten, daß die Wortlänge k der CPU $< n$ ist. Was dann?

$$= \begin{matrix} a_1 & a_2 & \dots & a_n \\ a_1 & a_2 & \dots & a_k \\ a_{k+1} & a_{k+2} & \dots & a_{2k} \\ \dots \\ a_{n-k+1} & a_{n-k+2} & \dots & a_n \end{matrix} \cdot \begin{matrix} b_1 & b_2 & \dots & b_n \\ b_1 & b_2 & \dots & b_k \\ b_{k+1} & b_{k+2} & \dots & b_{2k} \\ \dots \\ b_{n-k+1} & b_{n-k+2} & \dots & b_n \end{matrix}$$

Wir zerlegen ein inneres Produkt der Länge n in n/k innere Produkte der Länge k .

=> Aufwand ist $O(n/k)$

Multiplikation Boolescher Matrizen

gesehen: Wenn CPU-Wortlänge $= k$, dann inneres Produkt der Länge n in $O(n/k)$ berechenbar.

=> Aufwand für $n \times n$ Matrixmultiplikation (d.h. n^2 innere Produkte) ist n^3/k .

Wenn $k = \log n$ (nicht untypisch, mit 8-Bit Wortlänge wären so 256×256 Matrizen machbar), dann ist der Aufwand für eine Boolesche $n \times n$ Matrixmultiplikation $O(n^3/\log n)$.

Frage: Was aber wenn wir uns nicht auf die Fähigkeit der CPU, k Bits gleichzeitig mit AND zu verknüpfen verlassen können?

Multiplikation Boolescher Matrizen

Vorberechnen aller inneren Produkte, und Ablegen in einer Tabelle:

$$k=2 \begin{matrix} \text{00} & \text{01} & \text{10} & \text{11} \end{matrix}$$

00	0	0	0	0
01	0	1	0	1
10	0	0	1	1
11	0	1	1	1

Speicheraufwand:

$$O(2^k) = O(n) \text{ für } k = (\log n)/2$$

Rechenaufwand:

$$O(k2^k) = O(n \log n) \text{ für } k = (\log n)/2$$

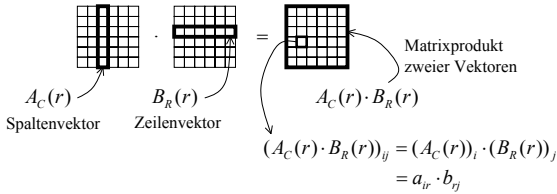
D.h.: wenn wir innere produkte der Länge n berechnen wollen, legen wir uns in $O(n \log n)$ Zeit eine Tabelle mit $O(n)$ Einträgen für innere Produkte der **gewählten** Länge $k = (\log n)/2$ an.

(Im obigen Beispiel wäre $n=16$.)

Alternativ: wähle $k = \log n$, dann steigt Speicheraufwand auf $O(n^2)$, aber k doppelt so groß => $2 \times$ weniger Operationen später. **Frage:** kann man $O(n^2)$ Speicher effizienter nutzen?

Multiplikation Boolescher Matrizen

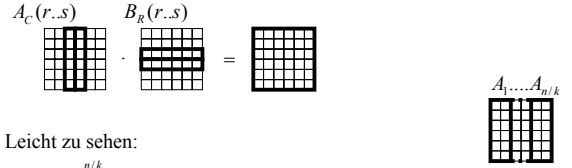
Andere Schreibweise für Matrixmultiplikation:



leicht zu sehen: $A \cdot B = \sum_{r=1}^n A_C(r) \cdot B_R(r)$

Multiplikation Boolescher Matrizen

$A \cdot B = \sum_{r=1}^n A_C(r) \cdot B_R(r)$ funktioniert nicht nur für einzelne Spalten (Zeilen), sondern auch für Gruppen von diesen:



Leicht zu sehen:

$$A \cdot B = \sum_{i=1}^{n/k} A_i B_i \quad \text{mit} \quad A = [A_1 A_2 \dots A_{n/k}], B = [B_1 B_2 \dots B_{n/k}]$$

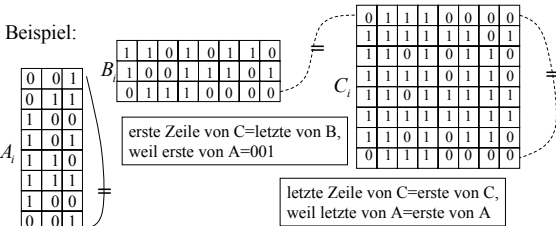
also $A_i = A_C(k(i-1)+1..ki), B_i = B_R(k(i-1)+1..ki)$

Multiplikation Boolescher Matrizen

$$A \cdot B = \sum_{i=1}^{n/k} A_i B_i = \sum_{i=1}^{n/k} C_i$$

Frage: wie berechnet man C_i am schnellsten?

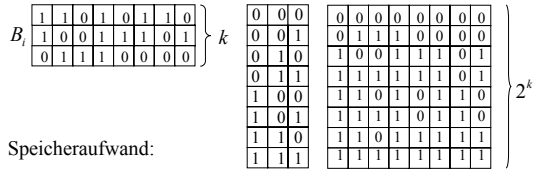
Beispiel:



Multiplikation Boolescher Matrizen

Feststellung: Es gibt maximal 2^k verschiedene Kombinationen (Potenzmenge) von Reihen aus B_i .

Frage: Lohnt es sich, die Summe jeder Kombination vorzuberechnen?



Speicheraufwand:

$$O(2^k \cdot 2^k) = O(n^2) \quad \text{für } k = \log n$$

Rechenaufwand: ?

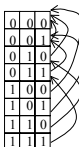
Multiplikation Boolescher Matrizen

Rechenaufwand für Vorbereitung aller Kombinationssummen:

Beginne mit Kombination 00...0 (Summe ist trivialerweise 00...0),

Induktion:

Berechne Kombination, die der Dualdarstellung von i entspricht.



Es gibt eine Zahl kleiner i , die sich von i nur durch ein Bit unterscheidet (z.B. die ohne die erste 1 in i).

Nimm diese und addiere die der ersten 1 entsprechende Reihe von B_i .

Aufwand:

Addition von n Bits für jede der 2^k Kombinationen $\Rightarrow O(n \cdot 2^k)$

Multiplikation Boolescher Matrizen

Algorithmus der vier Russen

gegeben: Boolesche $n \times n$ Matrizen A und B , Wortlänge k (teilt n)

$$A = [A_1 A_2 \dots A_{n/k}], B = [B_1 B_2 \dots B_{n/k}] \quad \text{gesucht: } C = AB$$

initialisiere $C=0$

für alle $i=1..n/k$

berechne alle 2^k Summen von Kombinationen aus Reihen von B_i für alle $j=1..n$

addiere zu C die Summe der Kombination, die der j -ten Zeile von A_i entspricht

Aufwand: n/k Multiplikationen für $A_i \cdot B_i$ mit jeweiligem Aufwand von $O(n^2)$ und $O(n \cdot 2^k)$ fürs Erstellen der Kombinationentabelle, gesamt also:

$$O(n^3 / k + 2^k \cdot n^2 / k) = O(n^3 / \log n) \quad \text{für } k = \log n$$