Algorithmen für die Speicherhierarchie

Lineare Algebra: untere Schranken

Riko Jacob

Lehrstuhl für Effiziente Algorithmen
Fakultät für Informatik
Technische Universität München

Vorlesung 12. November 2007

Technische Universität München

Gliederung

Zusätzliche Überlegungen

Erinnerung Permutieren

Elemente sind Atome (nur bewegen, kopieren, löschen)

Multiplikation erfordert mehr Modellbildung:

- Zwischenergebnisse nötig
- Struktur der Zahlen kann Aufgabe trivialisieren (0 oder $\mathbb{R}$)

$\Rightarrow$ Semiring I/O-Maschine:

- Nur Addieren und Multiplizieren, kein Minus oder geteilt (inverse Elemente)
- Rechnen nur im Hauptspeicher, wird nicht gezählt
- Speicherplatz in Anzahl von Zahlen
- Indizes und Adressen für umsonst

Zusätzliche Überlegungen

Erinnerung Permutieren

Elemente sind Atome (nur bewegen, ~~kopieren, löschen~~)

Multiplikation erfordert mehr Modellbildung:

- Zwischenergebnisse nötig
- Struktur der Zahlen kann Aufgabe trivialisieren (0 oder $\mathbb{R}$)

$\Rightarrow$ Semiring I/O-Maschine:

- Nur Addieren und Multiplizieren, kein Minus oder geteilt (inverse Elemente)
- Rechnen nur im Hauptspeicher, wird nicht gezählt
- Speicherplatz in Anzahl von Zahlen
- Indizes und Adressen für umsonst

Zusätzliche Überlegungen

Erinnerung Permutieren

Elemente sind Atome (nur bewegen, ~~kopieren, löschen~~)

Multiplikation erfordert mehr Modellbildung:

- Zwischenergebnisse nötig
- Struktur der Zahlen kann Aufgabe trivialisieren (0 oder $\mathbb{R}$)

$\Rightarrow$ Semiring I/O-Maschine:

- Nur Addieren und Multiplizieren, kein Minus oder geteilt (inverse Elemente)
- Rechnen nur im Hauptspeicher, wird nicht gezählt
- Speicherplatz in Anzahl von Zahlen
- Indizes und Adressen für umsonst

Zusätzliche Überlegungen

Erinnerung Permutieren

Elemente sind Atome (nur bewegen, ~~kopieren, löschen~~)

Multiplikation erfordert mehr Modellbildung:

- Zwischenergebnisse nötig
- Struktur der Zahlen kann Aufgabe trivialisieren (0 oder $\mathbb{R}$)

$\Rightarrow$ Semiring I/O-Maschine:

- Nur Addieren und Multiplizieren, kein Minus oder geteilt (inverse Elemente)
- Rechnen nur im Hauptspeicher, wird nicht gezählt
- Speicherplatz in Anzahl von Zahlen
- Indizes und Adressen für umsonst

Diskussion Semiring I/O-Maschine

- Alle Zwischenergebnisse haben Form

Matrix-Vektor	x_i,	a_{ij},	$x_j \cdot a_{ij}$,	$\sum_{j \in S} x_j \cdot a_{ij}$
Matrix-Matrix	a_{ij},	b_{ij},	$a_{ik} \cdot b_{kj}$,	$\sum_{k \in S} a_{ik} \cdot b_{kj}$
Skalarprodukt	x_i, y_j,	a_{ij},	$x_i \cdot a_{ij}, a_{ij} \cdot y_j$,	$\sum x_i a_{ij} y_j$

  sind so klassifizierbar
- Verzweigungen im Programm helfen nicht
- Indizes sind Teil des Programms
- Wie “I/O-Pebble Game / independent evaluation”
  [Hong, Kung 1981]
- Fokus auf Datenfluss (statt Algebra)

Technische Universität München

Diskussion Semiring I/O-Maschine

- Alle Zwischenergebnisse haben Form

Matrix-Vektor	x_i,	a_{ij},	$x_j \cdot a_{ij}$,	$\sum_{j \in S} x_j \cdot a_{ij}$
Matrix-Matrix	a_{ij},	b_{ij},	$a_{ik} \cdot b_{kj}$,	$\sum_{k \in S} a_{ik} \cdot b_{kj}$
Skalarprodukt	x_i, y_j,	a_{ij},	$x_i \cdot a_{ij}, a_{ij} \cdot y_j$,	$\sum x_i a_{ij} y_j$

  sind so klassifizierbar
- Verzweigungen im Programm helfen nicht
- Indizes sind Teil des Programms

- Wie “I/O-Pebble Game / independent evaluation”
  [Hong, Kung 1981]

- Fokus auf Datenfluss (statt Algebra)

Technische Universität München

Diskussion Semiring I/O-Maschine

- Alle Zwischenergebnisse haben Form

Matrix-Vektor	x_i,	a_{ij},	$x_j \cdot a_{ij}$,	$\sum_{j \in S} x_j \cdot a_{ij}$
Matrix-Matrix	a_{ij},	b_{ij},	$a_{ik} \cdot b_{kj}$,	$\sum_{k \in S} a_{ik} \cdot b_{kj}$
Skalarprodukt	x_i, y_j,	a_{ij},	$x_i \cdot a_{ij}, a_{ij} \cdot y_j$,	$\sum x_i a_{ij} y_j$

  sind so klassifizierbar
- Verzweigungen im Programm helfen nicht
- Indizes sind Teil des Programms
- Wie “I/O-Pebble Game / independent evaluation”
  [Hong, Kung 1981]
- Fokus auf Datenfluss (statt Algebra)

Technische Universität München

Vollbesetzte Matrix mal Vektor

Problem

$$y = A \cdot x\,, \quad y_i = \sum_{k=1..N} a_{ik} \cdot x_k$$

Direkter Algorithmus

Total $O(N^2/B)$ I/Os

Annahme

a_{ik} ohne I/O verfügbar

Algorithm 2: Blockweiser Algorithmus

- Teile y in Blöcke der Größe $s = M - B$
- $\frac{N}{s}$ mal x scannen

$$O\left(\frac{N}{M}\frac{N}{B}\right) = O\left(\frac{N^2}{MB}\right) \text{ I/Os}$$

Vollbesetzte Matrix mal Vektor

Problem

$$y = A \cdot x\,, \quad y_i = \sum_{k=1..N} a_{ik} \cdot x_k$$

Direkter Algorithmus

Total $O(N^2/B)$ I/Os

Annahme

a_{ik} ohne I/O verfügbar

Algorithm 2: Blockweiser Algorithmus

- Teile y in Blöcke der Größe $s = M - B$
- $\frac{N}{s}$ mal x scannen

$$O\left(\frac{N}{M}\frac{N}{B}\right) = O\left(\frac{N^2}{MB}\right) \text{ I/Os}$$

Vollbesetzte Matrix mal Vektor

Problem

$$y = A \cdot x\,, \quad y_i = \sum_{k=1..N} a_{ik} \cdot x_k$$

Direkter Algorithmus

Total $O(N^2/B)$ I/Os

Annahme

a_{ik} ohne I/O verfügbar

Algorithm 2: Blockweiser Algorithmus

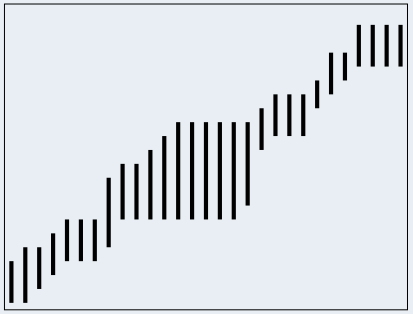
- Teile y in Blöcke der Größe $s = M - B$
- $\frac{N}{s}$ mal x scannen

$$O\left(\frac{N}{M}\frac{N}{B}\right) = O\left(\frac{N^2}{MB}\right) \text{ I/Os}$$

Skalarprodukt mit Bandmatrix

Problem

- Alle Einträge $a_{ij} = 1$ oder $a_{ij} = 0$
- Band: unterer und oberer Rand monoton
- k: Anzahl der 1-Einträge
- Berechne $y^T Ax$



Anwendung

Suche gemessene in synthetisierten Massenspektren

[Roos, Jacob, Grossmann, Fischer, Buhmann, Gruissem, Baginsky, Widmayer, 2007]

Algorithmus

Arbeite in Streifen der Breite $M - B$

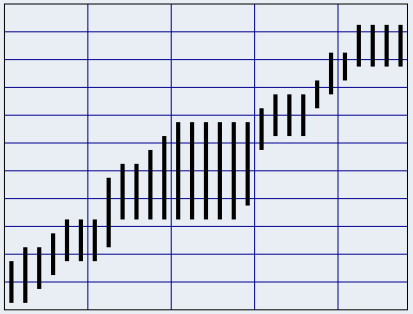
$$O\left(\frac{N_x+N_y}{B} + \frac{k}{BM}\right) \text{ I/Os}$$

N_x ist Dimension von x, N_y von y

Skalarprodukt mit Bandmatrix

Problem

- Alle Einträge $a_{ij} = 1$ oder $a_{ij} = 0$
- Band: unterer und oberer Rand monoton
- k: Anzahl der 1-Einträge
- Berechne $y^T Ax$



Anwendung

Suche gemessene in synthetisierten Massenspektren

[Roos, Jacob, Grossmann, Fischer, Buhmann, Gruissem, Baginsky, Widmayer, 2007]

Algorithmus

Arbeite in Streifen der Breite $M - B$

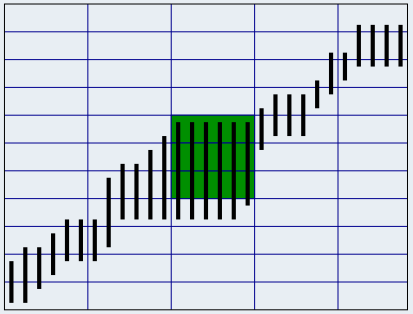
$$O\left(\frac{N_x+N_y}{B} + \frac{k}{BM}\right) \text{ I/Os}$$

N_x ist Dimension von x, N_y von y

Skalarprodukt mit Bandmatrix

Problem

- Alle Einträge $a_{ij} = 1$ oder $a_{ij} = 0$
- Band: unterer und oberer Rand monoton
- k: Anzahl der 1-Einträge
- Berechne $y^T Ax$



Anwendung

Suche gemessene in synthetisierten Massenspektren

[Roos, Jacob, Grossmann, Fischer, Buhmann, Gruissem, Baginsky, Widmayer, 2007]

Algorithmus

Arbeite in Streifen der Breite $M - B$

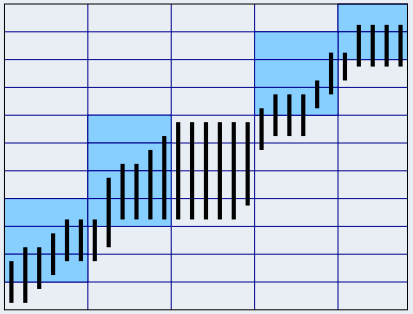
$$O\left(\frac{N_x+N_y}{B} + \frac{k}{BM}\right) \text{ I/Os}$$

N_x ist Dimension von x, N_y von y

Skalarprodukt mit Bandmatrix

Problem

- Alle Einträge $a_{ij} = 1$ oder $a_{ij} = 0$
- Band: unterer und oberer Rand monoton
- k: Anzahl der 1-Einträge
- Berechne $y^T Ax$



Anwendung

Suche gemessene in synthetisierten Massenspektren

[Roos, Jacob, Grossmann, Fischer, Buhmann, Gruissem, Baginsky, Widmayer, 2007]

Algorithmus

Arbeite in Streifen der Breite $M - B$

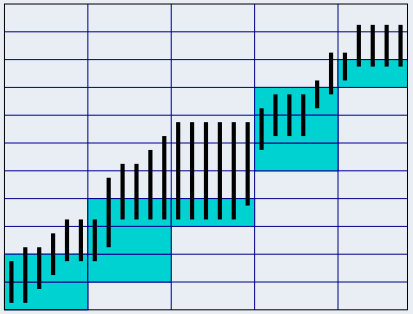
$$O\left(\frac{N_x + N_y}{B} + \frac{k}{BM}\right) \text{ I/Os}$$

N_x ist Dimension von x, N_y von y

Skalarprodukt mit Bandmatrix

Problem

- Alle Einträge $a_{ij} = 1$ oder $a_{ij} = 0$
- Band: unterer und oberer Rand monoton
- k: Anzahl der 1-Einträge
- Berechne $y^T Ax$



Anwendung

Suche gemessene in synthetisierten Massenspektren

[Roos, Jacob, Grossmann, Fischer, Buhmann, Gruissem, Baginsky, Widmayer, 2007]

Algorithmus

Arbeite in Streifen der Breite $M - B$

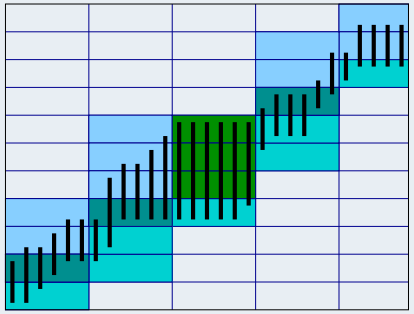
$$O\left(\frac{N_x+N_y}{B} + \frac{k}{BM}\right) \text{ I/Os}$$

N_x ist Dimension von x, N_y von y

Skalarprodukt mit Bandmatrix

Problem

- Alle Einträge $a_{ij} = 1$ oder $a_{ij} = 0$
- Band: unterer und oberer Rand monoton
- k: Anzahl der 1-Einträge
- Berechne $y^T Ax$



Anwendung

Suche gemessene in synthetisierten Massenspektren

[Roos, Jacob, Grossmann, Fischer, Buhmann, Gruissem, Baginsky, Widmayer, 2007]

Algorithmus

Arbeite in Streifen der Breite $M - B$

$$O\left(\frac{N_x+N_y}{B} + \frac{k}{BM}\right) \text{ I/Os}$$

N_x ist Dimension von x, N_y von y

Gradschranke

Annahme

Alle elementaren Produkte haben Grad $\leq D$

Ein elementares Produkt ist D-Tupel von Eingabevariablen.
Nach einem Input sind B neue Variablen im Speicher, diese erlauben höchstens $M^{D-1}B$ neue Tupel zu bilden.

Beispiel

Bandmatrix Skalarprodukt / Vollbesetzte Matrix–Vektor Alle elementaren Produkte haben Grad 2
Nach einem I/O höchstens MB neue $\Rightarrow \Omega\left(\frac{k}{MB}\right)$ I/Os

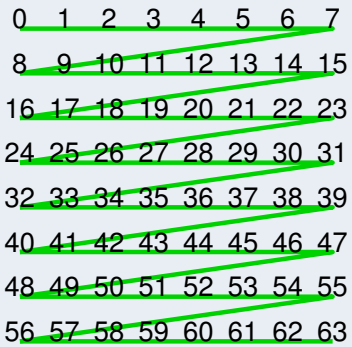
Zusammen mit Scanning-Bound:
vorgestellte Algorithmen sind optimal

Matrix Multiplikation

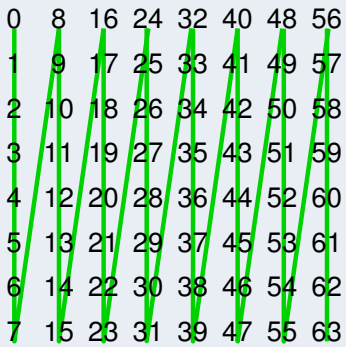
Problem

$$C = A \cdot B\,, \quad c_{ij} = \sum_{k=1..N} a_{ik} \cdot b_{kj}$$

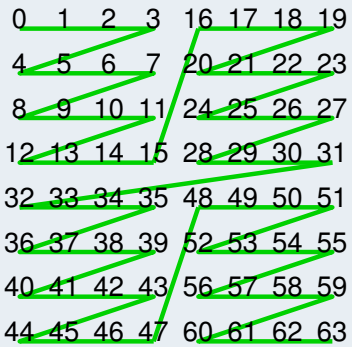
Layout von Matrizen



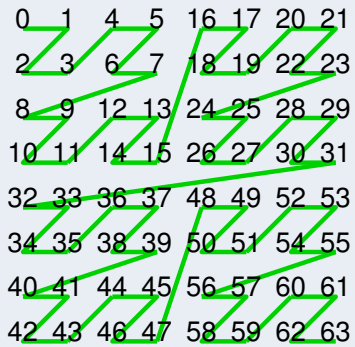
Zeilen



Spalten



4×4-Blöcke



Bit interleaved

Vollbesetzte Matrix mal Matrix

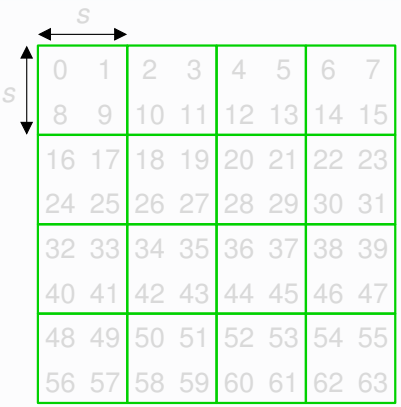
Algorithmus 1: Nested loops

- Zeilen Layout
- Lesen einer Spalte von B benötigt N I/Os
- Total $O(N^3)$ I/Os

```
for i = 1 to N
  for j = 1 to N
    c_ij = 0
    for k = 1 to N
      c_ij = c_ij + a_ik · b_kj
```

Algorithm 2: Blockweiser Algorithmus

- Teile A und B in $s \times s$ Blöcke mit $s = \Theta(\sqrt{M})$
- Algorithm 1 für die $\frac{N}{s} \times \frac{N}{s}$ Matrizen Elemente sind $s \times s$ Matrizen
- $s \times s$-Blöcke oder $\left(\text{Zeilen und } M = \Omega(B^2)\right)$



$$O\left(\left(\frac{N}{s}\right)^3 \cdot \frac{s^2}{B}\right) = O\left(\frac{N^3}{s \cdot B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right) \text{ I/Os}$$

Vollbesetzte Matrix mal Matrix

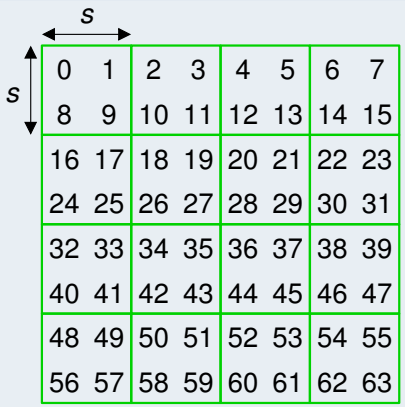
Algorithmus 1: Nested loops

- Zeilen Layout
- Lesen einer Spalte von B
  benötigt N I/Os
- Total $O(N^3)$ I/Os

```
for i = 1 to N
  for j = 1 to N
    c_ij = 0
    for k = 1 to N
      c_ij = c_ij + a_ik · b_kj
```

Algorithm 2: Blockweiser Algorithmus

- Teile A und B in $s \times s$ Blöcke
  mit $s = \Theta(\sqrt{M})$
- Algorithm 1 für die $\frac{N}{s} \times \frac{N}{s}$ Matrizen
  Elemente sind $s \times s$ Matrizen
- $s \times s$-Blöcke oder $\left(\text{Zeilen und } M = \Omega(B^2)\right)$



$$O\left(\left(\frac{N}{s}\right)^3 \cdot \frac{s^2}{B}\right) = O\left(\frac{N^3}{s \cdot B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right) \text{ I/Os}$$

Vollbesetzte Matrix mal Matrix

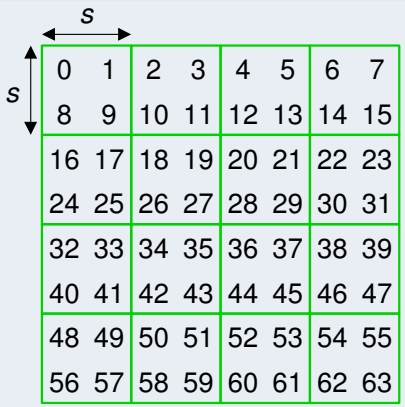
Algorithmus 1: Nested loops

- Zeilen Layout
- Lesen einer Spalte von B
  benötigt N I/Os
- Total $O(N^3)$ I/Os

```
for i = 1 to N
  for j = 1 to N
    c_ij = 0
    for k = 1 to N
      c_ij = c_ij + a_ik · b_kj
```

Algorithm 2: Blockweiser Algorithmus

- Teile A und B in $s \times s$ Blöcke
  mit $s = \Theta(\sqrt{M})$
- Algorithm 1 für die $\frac{N}{s} \times \frac{N}{s}$ Matrizen
  Elemente sind $s \times s$ Matrizen
- $s \times s$-Blöcke oder $\left(\text{Zeilen und } M = \Omega(B^2)\right)$



$$O\left(\left(\frac{N}{s}\right)^3 \cdot \frac{s^2}{B}\right) = O\left(\frac{N^3}{s \cdot B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right) \text{ I/Os}$$

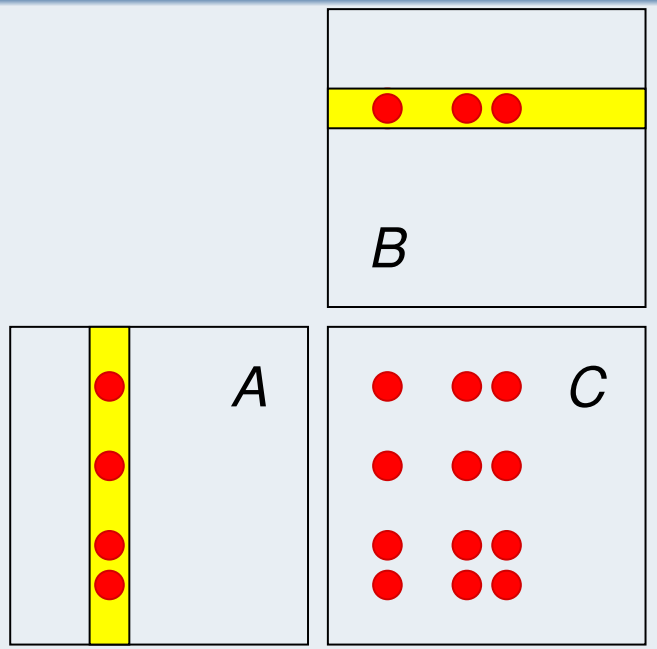
Vollbesetzte Matrix–Matrix

Berechnung in Runden [Hong, Kung 1981]

Für jede (M, B)-I/O Berechnung mit ℓ I/Os gibt es eine $(2M, B)$-I/O Berechnung mit höchstens 2ℓ I/Os, der Gestalt (Speicher leer, laden, speichern, Speicher leer)*

untere Schranke: elementare Produkte pro Runde

- Sei c #Ergebnisse (C, Output)
- a_i # Elemente in Spalte i von A
- b_i # Elemente in Zeile i von B
- Speicher: $c, \sum_i a_i + b_i \leq M$
- Produkte: $\sum_i \min\{a_i \cdot b_i, c\}$
- Maximal für $a_i = b_i = \sqrt{M}, c = M$
  #Produkte $\leq M\sqrt{M}$



untere Schranke: vollbesetzte Matrix–Matrix

Theorem [Hong, Kung 1981]

Ein Semiring I/O Programm, dass eine vollbesetzte $N_1 \times N_2$ mit einer $N_2 \times N_3$ Matrix multipliziert benötigt

$$\Omega\left(\frac{N_1 N_2 N_3}{B\sqrt{M}}\right)$$

Beweis:

Es gibt $N_1 N_2 N_3$ elementare Produkte, also (vorige Folie) $\Omega\left(\frac{N_1 N_2 N_3}{M\sqrt{M}}\right)$ Runden.
Eine Runde hat M/B I/Os.

Multiplizieren mit einer dünn besetzten Matrix

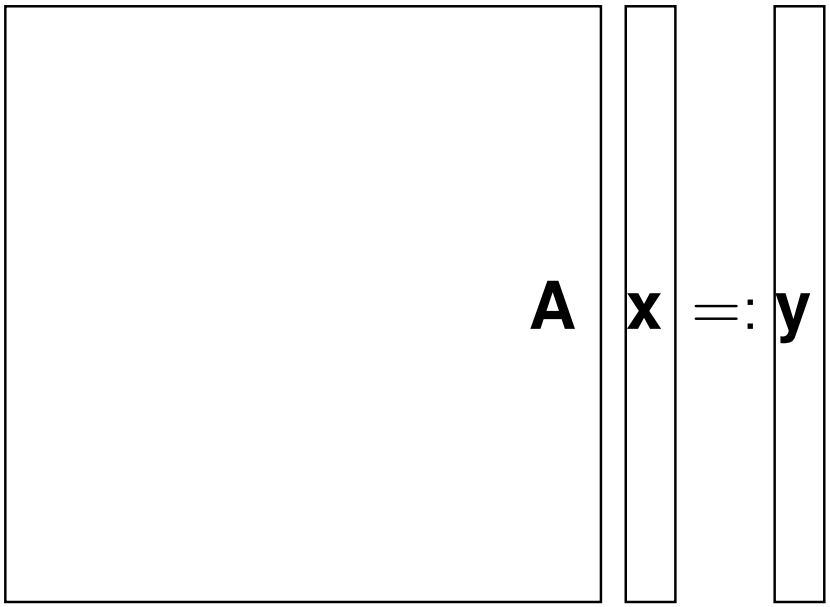
Berechne

$$\mathbf{A}\ \mathbf{x} =: \mathbf{y}$$

Multiplizieren mit einer dünn besetzten Matrix

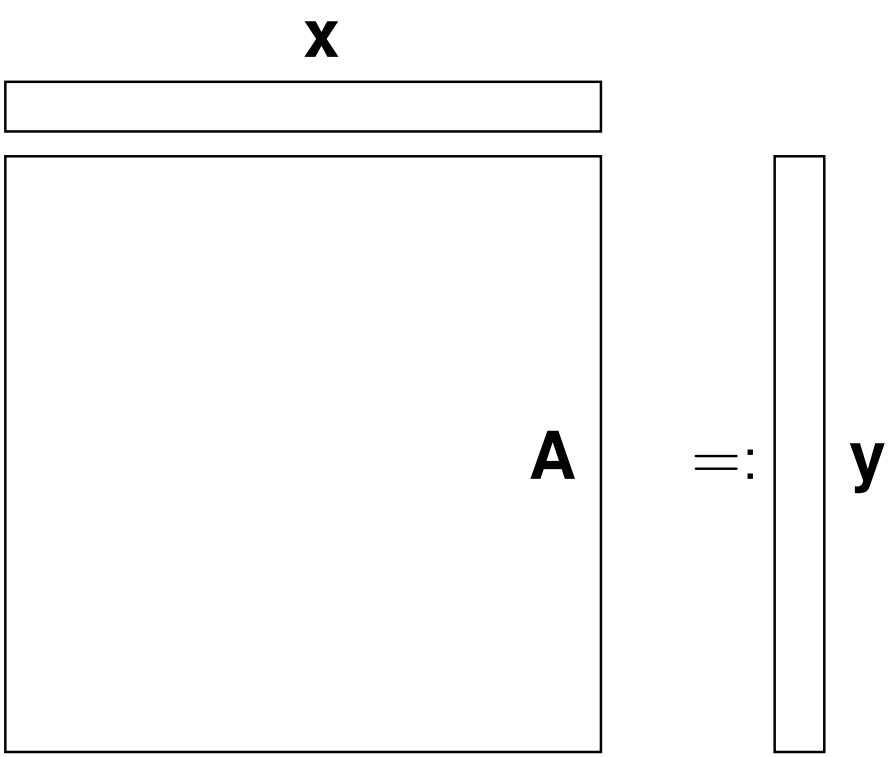
Berechne $\mathbf{Ax} =: \mathbf{y}$,

$\mathbf{A}$ ist $N \times N$ Matrix



Multiplizieren mit einer dünn besetzten Matrix

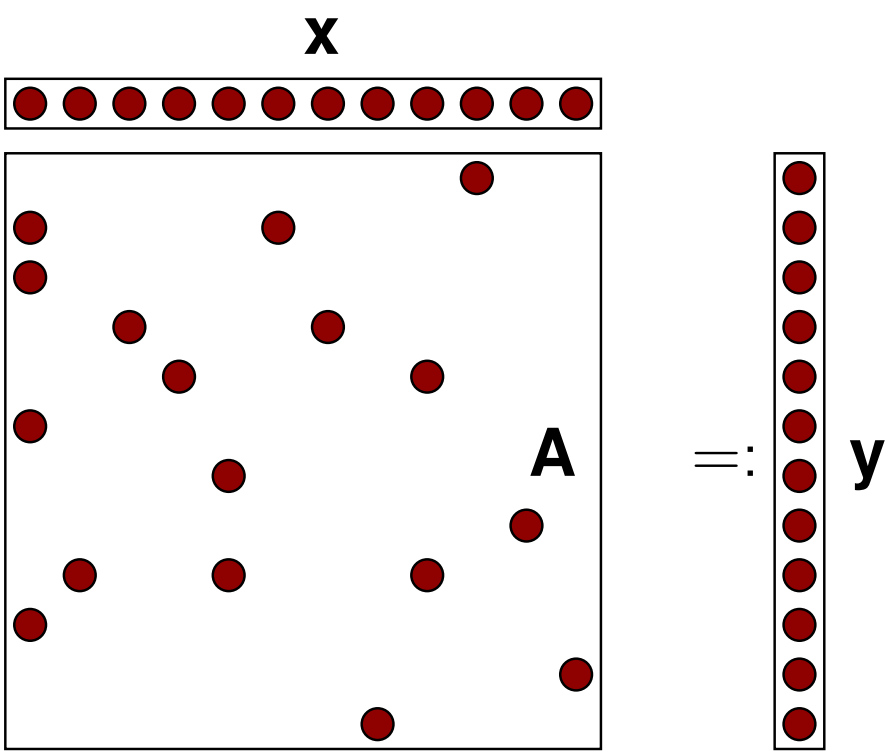
Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix

Multiplizieren mit einer dünn besetzten Matrix

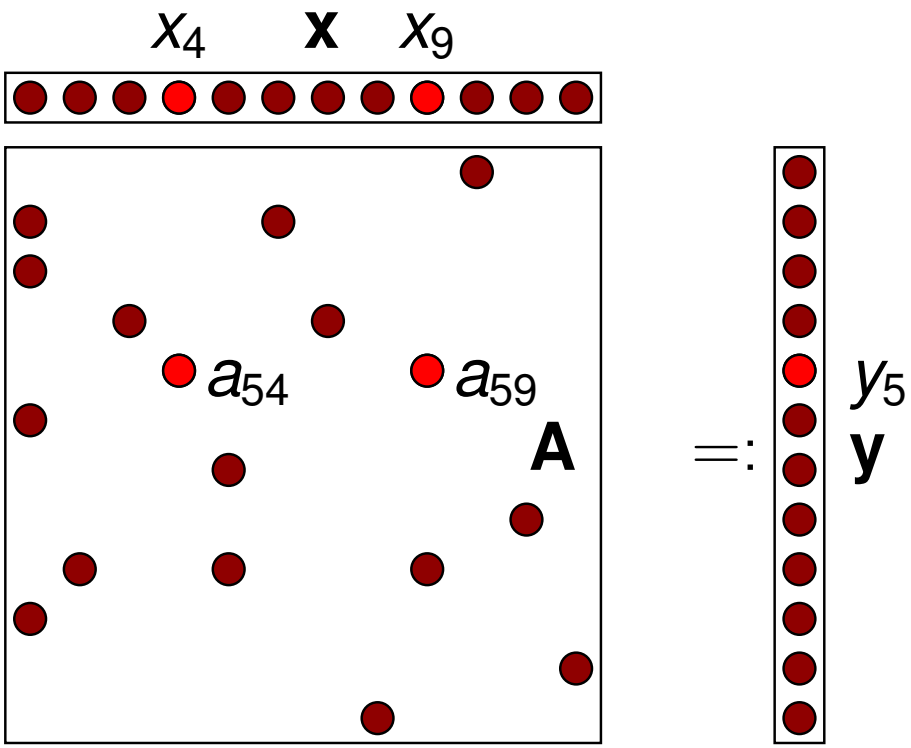
Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,



$\mathbf{A}$ ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

$y_i = \sum_{j=1}^{N} a_{ij} x_j$

Permutieren ist Spezialfall:
$k = 1$, eine 1 pro Zeile und pro Spalte

Anwendungen

dünn besetzte Matrizen:

- Diskretisierung von Differentialgleichungen
- Modifikation von digitalen Bildern
- Adjazenzmatrix eines Graphen

Operationen:

- Iteratives Lösen eines Gleichungssystems
- Eigenwerte und Eigenvektoren
- Googles PageRank

Programmierung:

- Software Bibliothek: “Sparse Matrix Kernel”
- Googles “MapReduce”

Multiplikations-Programme

Direkt $\mathbf{x} \mapsto \mathbf{Ax}$

for $i = 1$ *to* N **do**
  $y_i := 0$;
  for $i = 1$ *to* N **do**
    $y_i := y_i + a_{ij}x_j$;
  end
end

Kontrollstruktur hängt nur von **A** ab

$\Rightarrow$ Eigentlich ein langes Programm ohne Verzweigung

i, j sind dann Konstanten

Technische Universität München

Multiplikations-Programme

Direkt $\mathbf{x} \mapsto \mathbf{Ax}$

for $i = 1$ *to* N **do**
  $y_i := 0$;
  for $i = 1$ *to* N **do**
    if $a_{ij} \neq 0$ **then**
      $y_i := y_i + a_{ij} x_j$;
    end
  end
end

Kontrollstruktur hängt nur von **A** ab

$\Rightarrow$ Eigentlich ein langes Programm ohne Verzweigung

i, j sind dann Konstanten

Technische Universität München

Multiplikations-Programme

Direkt $\mathbf{x} \mapsto \mathbf{Ax}$

for $i = 1$ *to* N **do**
  $y_i := 0$;
  for $i = 1$ *to* N **do**
    if $a_{ij} \neq 0$ **then**
      $y_i := y_i + a_{ij} x_j$;
    end
  end
end

A ist Liste $L = [(i, j, a_{ij})]$

for $i = 1$ *to* N **do**
  $y_i := 0$;
end
for $(i, j, a_{ij}) \in L$ **do**
  $y_i := y_i + a_{ij} x_j$;
end

Kontrollstruktur hängt nur von **A** ab

$\Rightarrow$ Eigentlich ein langes Programm ohne Verzweigung

i, j sind dann Konstanten

Technische Universität München

Multiplikations-Programme

Direkt $\mathbf{x} \mapsto \mathbf{Ax}$

for $i = 1$ *to* N **do**
  $y_i := 0$;
  for $i = 1$ *to* N **do**
    if $a_{ij} \neq 0$ **then**
      $y_i := y_i + a_{ij} x_j$;
    end
  end
end

A ist Liste $L = [(i, j, a_{ij})]$

for $i = 1$ *to* N **do**
  $y_i := 0$;
end
for $(i, j, a_{ij}) \in L$ **do**
  $y_i := y_i + a_{ij} x_j$;
end

Kontrollstruktur hängt nur von **A** ab

$\Rightarrow$ Eigentlich ein langes Programm ohne Verzweigung

i, j sind dann Konstanten

Technische Universität München

Multiplikations-Programme

Direkt $\mathbf{x} \mapsto \mathbf{Ax}$

for $i = 1$ *to* N **do**
  $y_i := 0$;
  for $i = 1$ *to* N **do**
    if $a_{ij} \neq 0$ **then**
      $y_i := y_i + a_{ij} x_j$;
    end
  end
end

A ist Liste $L = \left[(i, j, a_{ij})\right]$

for $i = 1$ *to* N **do**
  $y_i := 0$;
end
for $(i, j, a_{ij}) \in L$ **do**
  $y_i := y_i + a_{ij} x_j$;
end

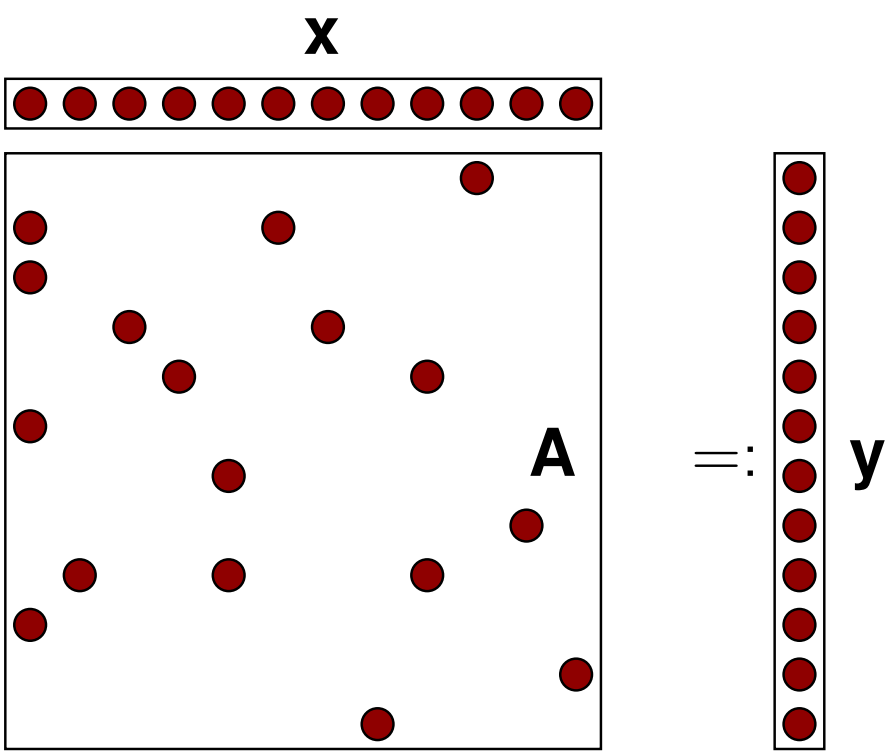
Kontrollstruktur hängt nur von **A** ab

$\Rightarrow$ Eigentlich ein langes Programm ohne Verzweigung
i, j sind dann Konstanten

Technische Universität München

Multiplizieren mit einer dünn besetzten Matrix

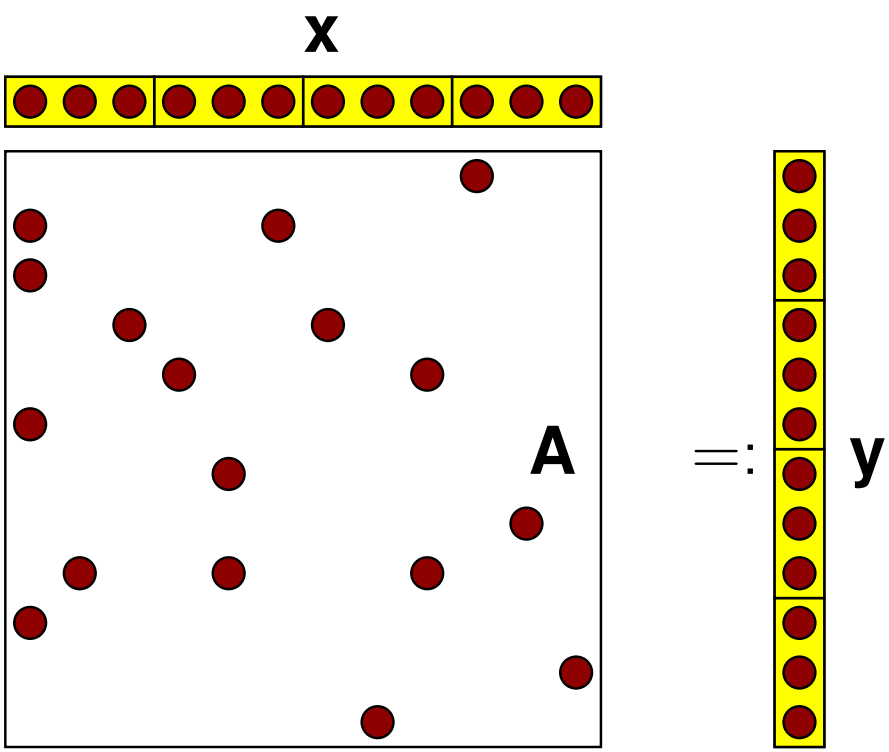
Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Multiplizieren mit einer dünn besetzten Matrix

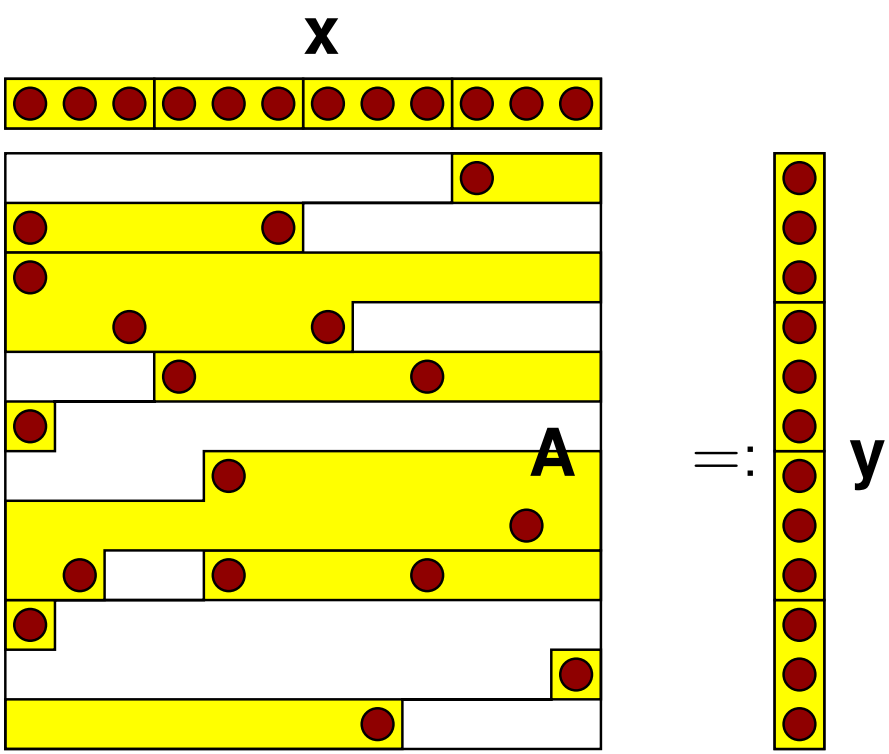
Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,

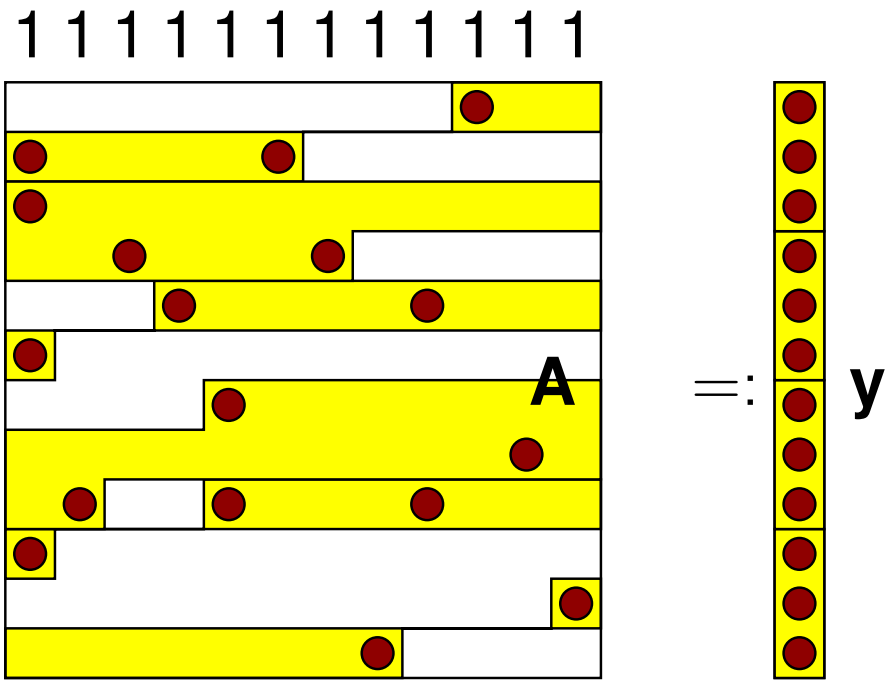


A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Row-major Layout

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,

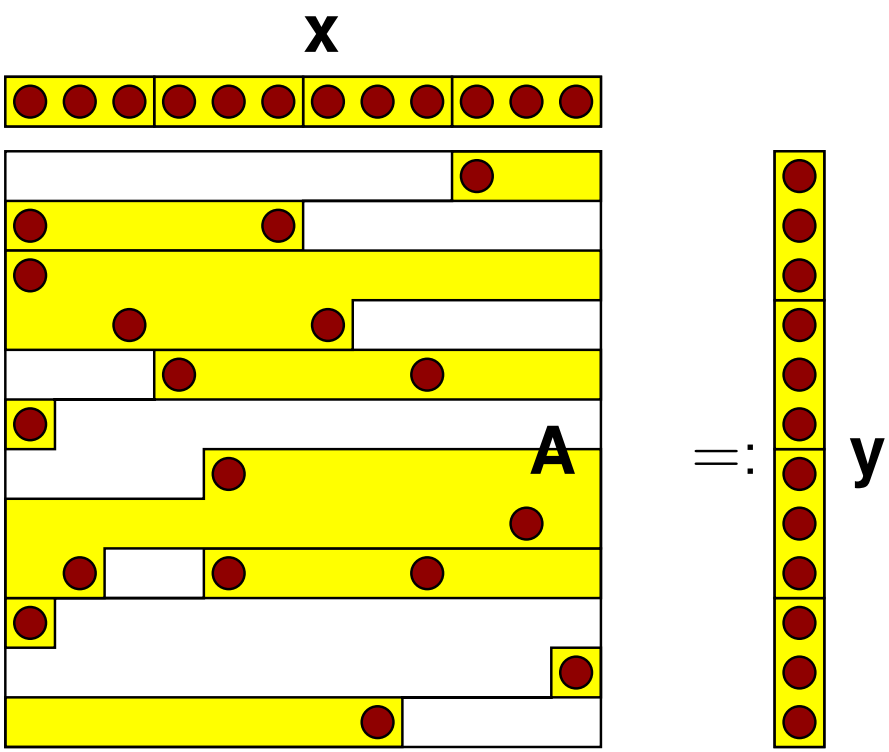


A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Row-major Layout
Zeilensummen
$\Theta(kN/B)$ I/Os

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,

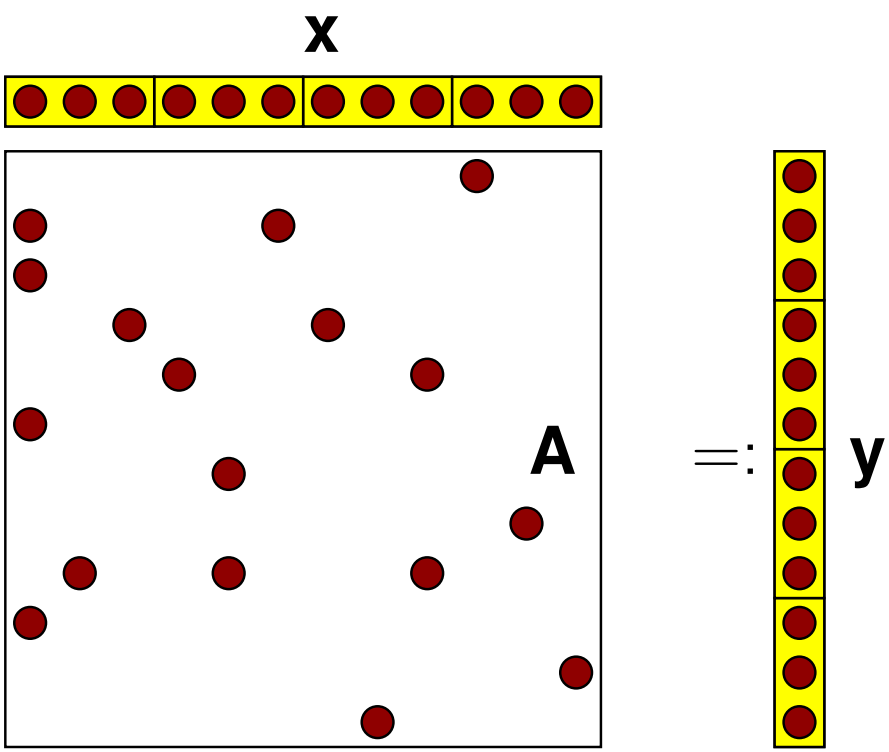


A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Row-major Layout
Direkter Alg.
$\Theta(kN)$ I/Os

Multiplizieren mit einer dünn besetzten Matrix

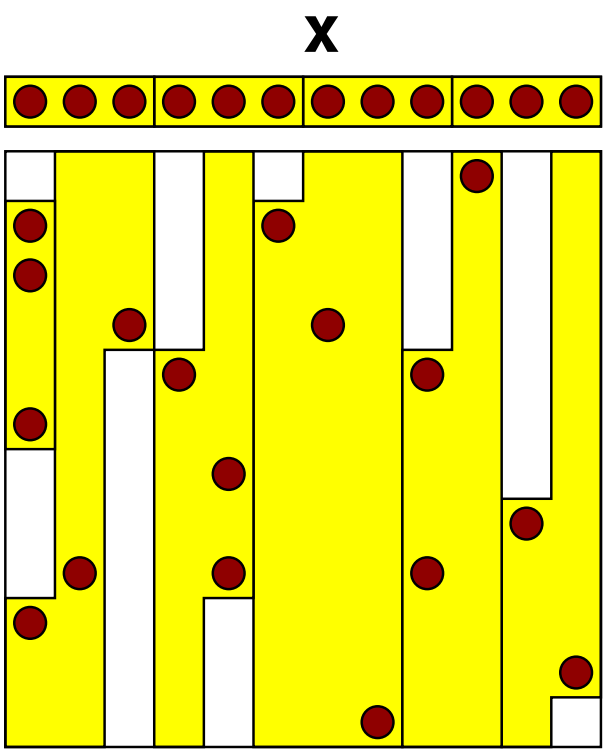
Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Multiplizieren mit einer dünn besetzten Matrix

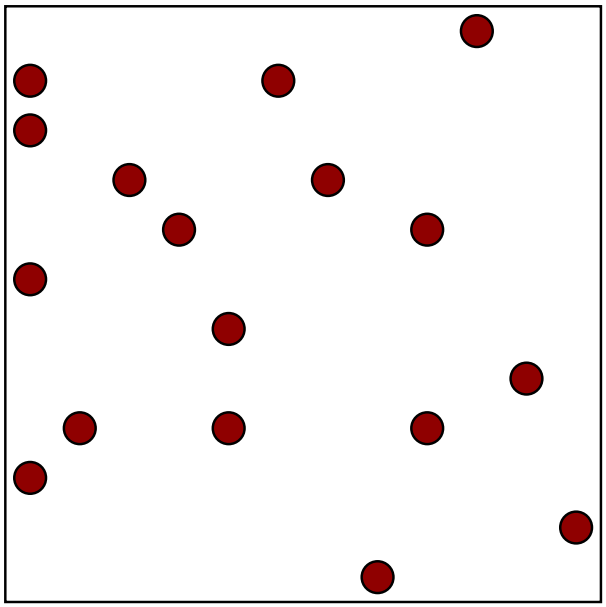
Berechne $\mathbf{Ax} =: \mathbf{y}$,

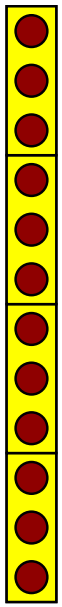


A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,





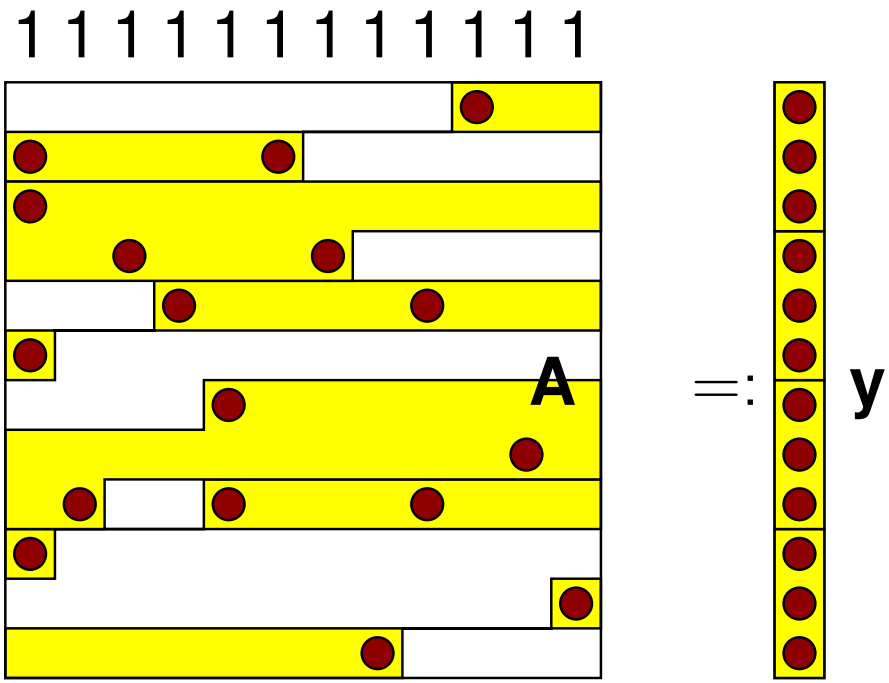
$\mathbf{y}$

$\mathbf{A}$ ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Einträge sind
Produkte $a_{ij}x_j$

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Row-major Layout
Zeilensummen
$\Theta(kN/B)$ I/Os

Sortierbasiertes Multiplizieren

A ist Liste $L = \left[(i, j, a_{ij})\right]$, Z ist gleichartige Liste

for $i = 1$ *to* N **do** $y_i := 0$;
Sortiere L nach Spalten (j);
for $(i, j, a_{ij}) \in L$ **do** $z_{ij} := a_{ij} x_j$; // j aufsteigend
Sortiere Z nach Zeilen (i);
for $(i, j, z_{ij}) \in Z$ **do** $y_i := y_i + z_{ij}$; // i aufsteigend

Analyse des Sortierbasierten Programms

Multiplikationen und Additionen: $O(kN)$

I/Os:

$$O\left(\frac{kN}{B}\log_{\frac{M}{B}}\frac{kN}{M}\right)$$

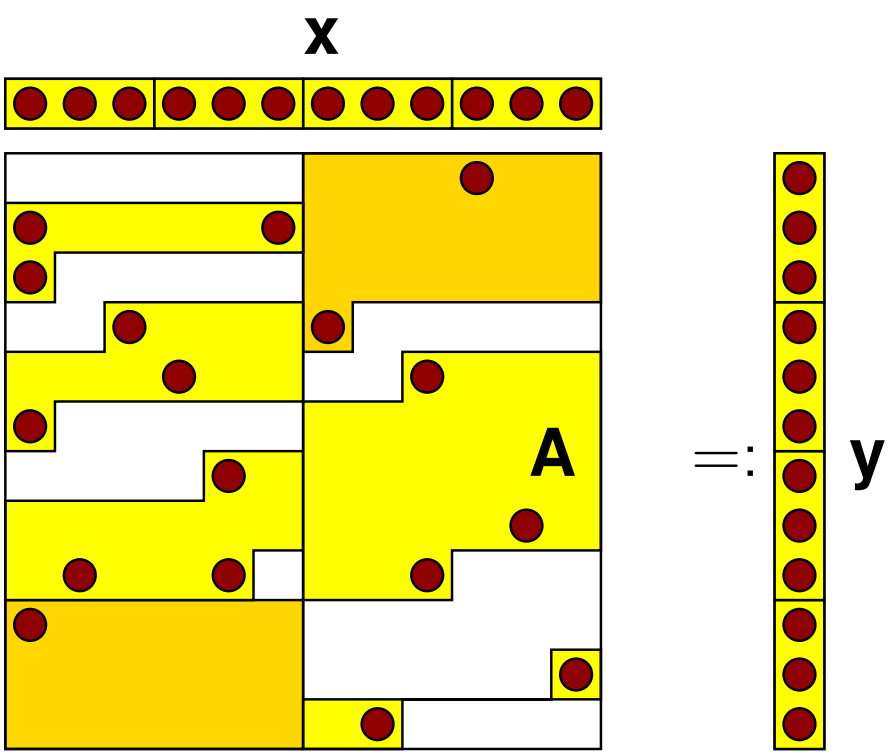
Sortierbasiertes Multiplizieren
Block Layout

A ist Liste $L = [(i, j, a_{ij})]$, Z ist gleichartige Liste

for $i = 1$ *to* N **do** $y_i := 0$;
~~Sortiere L nach Spalten (j);~~ L ist in Block-Row-Major Layout
for $(i, j, a_{ij}) \in L$ **do** $z_{ij} := a_{ij} x_j$; // j aufsteigend
Sortiere Z nach Zeilen (i);
for $(i, j, z_{ij}) \in Z$ **do** $y_i := y_i + z_{ij}$; // i aufsteigend

Multiplizieren mit einer dünn besetzten Matrix

Berechne $\mathbf{Ax} =: \mathbf{y}$,



A ist $N \times N$ Matrix
mit kN Einträgen $\neq 0$.
$(1 \leq k \leq \sqrt{N})$

Block-Column-major
Layout

Analyse des Sortierbasierten Programms

Multiplikationen und Additionen: $O(kN)$

I/Os:

$$O\left(\frac{kN}{B}\log_{\frac{M}{B}}\frac{kN}{M}\right)$$

$$O\left(\frac{kN}{B}\log_{\frac{M}{B}}\frac{N}{M}\right)$$

Sortierbasiertes Multiplizieren

Block Layout und Addieren beim Sortieren

A ist Liste $L = [(i, j, a_{ij})]$, Z ist gleichartige Liste

for $i = 1$ *to* N **do** $y_i := 0$;
~~Sortiere L nach Spalten (j);~~ L ist in Block-Row-Major Layout
for $(i, j, a_{ij}) \in L$ **do** $z_{ij} := a_{ij} x_j$; // j aufsteigend
~~Sortiere Z nach Zeilen (i);~~
begin Beim Sortieren von Z nach Zeilen
  if (i, j, z_{ij}) *und* (i, h, z_{ih}) *zugleich im Speicher* **then**
    $z_{ij} := z_{ij} + z_{ih}$, werfe z_{ih} weg;
  end
end
~~for $(i, j, z_{ij}) \in Z$ do $y_i := y_i + z_{ij}$; // i aufsteigend~~

Technische Universität München

Analyse des Sortierbasierten Programms

Multiplikationen und Additionen: $O(kN)$

I/Os:

$$O\left(\frac{kN}{B}\log_{\frac{M}{B}}\frac{kN}{M}\right)$$

$$O\left(\frac{kN}{B}\log_{\frac{M}{B}}\frac{N}{M}\right)$$

$$O\left(\frac{kN}{B}\log_{\frac{M}{B}}\frac{N}{kM}\right)$$

Technische Universität München

Dünnbesetzte Matrizen

k-Dünnbesetzte Matrix

Die $N \times N$ Matrix A hat kN Nicht-Null-Einträge, also im Schnitt k Einträge pro Spalte.

Bekannt: [Aggarwal, Vitter 1988]

Permutationsmatrizen sind Spezialfall:

$$\Omega\left(\min\left\{\frac{N}{B}\left(1+\log_{M/B}\frac{N}{M}\right), N\right\}\right)$$

Somit Faktor k schwächer als Algorithmen

Aussage der unteren Schranken

- Column-Major Layout

  $B > 2$, $M \geq 4B$ und $k \leq N^{1-\epsilon}$

$$\Theta\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{\max\{k,M\}}\right), kN\right\}\right)$$

- Freies Layout

  $B \geq 6$, $M \geq 3B$ und $k \leq \sqrt[3]{N}$

$$\Theta\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{kM}\right), kN\right\}\right)$$

Gilt für $k > 5$ auch wenn Eingabe- und Ausgabe-Vektor in beliebiger Form gespeichert werden können.

Technische Universität München

Beweisidee Untere Schranken

- Wie [Aggarwal, Vitter 1988]: Vergleiche Anzahl schneller Programme mit Anzahl von Aufgaben
- Normalisieren der Programme:
  - Mengen statt Listen
  - Ergebnisse rückwärts (nur Zeile statt auch Teilmenge von Spalten)
  - Berechnung so früh wie möglich

  $\Rightarrow$ kurze, eindeutige Beschreibung jedes Programms
- Sorgfältiges Auflösen der entstehenden Ungleichung

Eine ungewöhnliche Aufgabe

Aufgabe Gegeben $y_1, \ldots, y_N$, erzeuge eine Liste der Länge kN, interpretiert als N Blöcke von k Variablen.

$$(y_1, y_2, y_3) \rightsquigarrow ([y_1, y_3], [y_1, y_2], [y_2, y_3]) \equiv \begin{pmatrix} y_1 & y_1 & \\ & y_2 & y_2 \\ y_3 & & y_3 \end{pmatrix}$$

Ziel: Erzeuge alle k-regulären Matrizen in Column Major Layout.

- Annahme: Variablen können nur bewegt, kopiert oder gelöscht werden.
- Normalisiere: Ordnung im Speicher/Blöcken ist unwichtig $\Rightarrow$ Mengen von Indizes
- Dies ist Zeit-umgekehrt die Aufgabe Zeilen-Summen zu berechnen ($\mathbf{x} = \mathbf{1}$).

Technische Universität München

Eine ungewöhnliche Aufgabe

Aufgabe Gegeben $y_1, \ldots, y_N$, erzeuge eine Liste der Länge kN, interpretiert als N Blöcke von k Variablen.

$$(1,2,3) \quad \rightsquigarrow \quad (\{1,3\},\{1,2\},\{2,3\}) \quad \equiv \begin{pmatrix} y_1 & y_1 & \\ & y_2 & y_2 \\ y_3 & & y_3 \end{pmatrix}$$

Ziel: Erzeuge alle k-regulären Matrizen in Column Major Layout.

- Annahme: Variablen können nur bewegt, kopiert oder gelöscht werden.
- Normalisiere: Ordnung im Speicher/Blöcken ist unwichtig $\Rightarrow$ Mengen von Indizes
- Dies ist Zeit-umgekehrt die Aufgabe Zeilen-Summen zu berechnen ($\mathbf{x} = \mathbf{1}$).

Technische Universität München

Eine ungewöhnliche Aufgabe

Aufgabe Gegeben $y_1, \ldots, y_N$, erzeuge eine Liste der Länge kN, interpretiert als N Blöcke von k Variablen.

$$(1,2,3) \quad \rightsquigarrow \quad (\{1,3\},\{1,2\},\{2,3\}) \quad \equiv \begin{pmatrix} y_1 & y_1 & \\ & y_2 & y_2 \\ y_3 & & y_3 \end{pmatrix}$$

Ziel: Erzeuge alle k-regulären Matrizen in Column Major Layout.

- Annahme: Variablen können nur bewegt, kopiert oder gelöscht werden.
- Normalisiere: Ordnung im Speicher/Blöcken ist unwichtig $\Rightarrow$ Mengen von Indizes
- Dies ist Zeit-umgekehrt die Aufgabe Zeilen-Summen zu berechnen ($\mathbf{x} = \mathbf{1}$).

Technische Universität München

Matrizen Erzeugen
Ideen

- Symbolische Representation: Zahlen $\leftrightarrow$ Indizes von Variablen
- $\mathcal{M} \subset [N]$, $|\mathcal{M}| \leq M$; $\mathcal{T}_i \subset [N]$, $|\mathcal{T}_i| \leq B$.
- Verfolge die Bewegung der Variablen
- Eindeutige Anfangskonfiguration $\mathcal{M} = \emptyset$, $\mathcal{T}_1 = \{1, ..., B\}, \ldots, \mathcal{T}_{N/B} = \{N - B + 1, ..., N\}$
- Endkonfiguration bestimmt die Gestalt der Matrix bis auf Anordnung in den Mengen

Matrizen Erzeugen
Ideen

- Symbolische Representation: Zahlen $\leftrightarrow$ Indizes von Variablen
- $\mathcal{M} \subset [N]$, $|\mathcal{M}| \leq M$; $\mathcal{T}_i \subset [N]$, $|\mathcal{T}_i| \leq B$.
- Verfolge die Bewegung der Variablen
- Eindeutige Anfangskonfiguration $\mathcal{M} = \emptyset$, $\mathcal{T}_1 = \{1, ..., B\}, \ldots, \mathcal{T}_{N/B} = \{N - B + 1, ..., N\}$
- Endkonfiguration bestimmt die Gestalt der Matrix bis auf Anordnung in den Mengen

Matrizen Erzeugen
Ideen

- Symbolische Representation: Zahlen $\leftrightarrow$ Indizes von Variablen
- $\mathcal{M} \subset [N]$, $|\mathcal{M}| \leq M$; $\mathcal{T}_i \subset [N]$, $|\mathcal{T}_i| \leq B$.
- Verfolge die Bewegung der Variablen
- Eindeutige Anfangskonfiguration $\mathcal{M} = \emptyset$, $\mathcal{T}_1 = \{1, ..., B\}, \ldots, \mathcal{T}_{N/B} = \{N - B + 1, ..., N\}$
- Endkonfiguration bestimmt die Gestalt der Matrix bis auf Anordnung in den Mengen

Technische Universität München

Matrizen Erzeugen
Ideen

- Symbolische Representation: Zahlen $\leftrightarrow$ Indizes von Variablen
- $\mathcal{M} \subset [N]$, $|\mathcal{M}| \leq M$; $\mathcal{T}_i \subset [N]$, $|\mathcal{T}_i| \leq B$.
- Verfolge die Bewegung der Variablen
- Eindeutige Anfangskonfiguration $\mathcal{M} = \emptyset$, $\mathcal{T}_1 = \{1, ..., B\}, \ldots, \mathcal{T}_{N/B} = \{N - B + 1, ..., N\}$
- Endkonfiguration bestimmt die Gestalt der Matrix bis auf Anordnung in den Mengen

Matrizen Erzeugen
Ideen

- Symbolische Representation: Zahlen $\leftrightarrow$ Indizes von Variablen
- $\mathcal{M} \subset [N]$, $|\mathcal{M}| \leq M$; $\mathcal{T}_i \subset [N]$, $|\mathcal{T}_i| \leq B$.
- Verfolge die Bewegung der Variablen
- Eindeutige Anfangskonfiguration $\mathcal{M} = \emptyset$, $\mathcal{T}_1 = \{1, ..., B\}, \ldots, \mathcal{T}_{N/B} = \{N - B + 1, ..., N\}$
- Endkonfiguration bestimmt die Gestalt der Matrix bis auf Anordnung in den Mengen

Zählen

Endkonfiguration

Eine Endkonfiguration entspricht einer bestimmten Anzahl von verschiedenen Matrizen.

Mehrdeutigkeit:

$$\begin{cases} 3^{kN}, & B < k \\ 1, & B = k \\ (2eB/k)^{kN}, & B > k \end{cases}$$

Anzahl verschiedener Programme (Datenfluss-Spur)

Anzahl Nachfolgekonfigurationen pro I/O-Operation:

$$\ell \binom{M+B}{B}$$

Zählen

Endkonfiguration

Eine Endkonfiguration entspricht einer bestimmten Anzahl von verschiedenen Matrizen.

Mehrdeutigkeit:

$$\begin{cases} 3^{kN}, & B < k \\ 1, & B = k \\ (2eB/k)^{kN}, & B > k \end{cases}$$

Anzahl verschiedener Programme (Datenfluss-Spur)

Anzahl Nachfolgekonfigurationen pro I/O-Operation:

$$\ell \binom{M+B}{B}$$

Theorem: Matrizen Erzeugen

Lemma

Für $B > 2$, $M \geq 4B$ und $k \leq N^{1-\varepsilon}$ gilt

$$\ell(k,N) \geq \min\left\{\kappa(\varepsilon)\frac{kN}{B}\log_{M/B}\frac{N}{\max\{k,M\}}\,,\ \frac{1}{8}\kappa'(\varepsilon)kN\right\}\,.$$

Proof.

Ergibt sich aus:

$$\binom{N}{k}^N \leq \left(\binom{M+B}{B}\ell\right)^\ell \cdot \max\{3, 2eB/k\}^{kN}$$

□

Theorem: Matrizen Erzeugen

Lemma

Für $B > 2$, $M \geq 4B$ und $k \leq N^{1-\varepsilon}$ gilt

$$\ell(k,N) \geq \min\left\{\kappa(\varepsilon)\frac{kN}{B}\log_{M/B}\frac{N}{\max\{k,M\}}\,,\ \frac{1}{8}\kappa'(\varepsilon)kN\right\}\,.$$

Proof.

Ergibt sich aus:

$$\binom{N}{k}^N \leq \left(\binom{M+B}{B}\ell\right)^\ell \cdot \max\{3, 2eB/k\}^{kN}$$

Theorem: Column Major Layout

Theorem

Für $B > 2$, $M \geq 4B$ und $k \leq N^{1-\varepsilon}$ gilt

$$\ell(k,N) \geq \min\left\{\kappa(\varepsilon)\frac{kN}{B}\log_{M/B}\frac{N}{\max\{k,M\}}, \frac{1}{8}\kappa'(\varepsilon)kN\right\}.$$

Proof.

Multiplizieren mit dem **1**-Vektor ist Zeit invers zum Erzeugen von Matrizen
(Teilsummen statt Variablen) □

Theorem: Column Major Layout

Theorem

Für $B > 2$, $M \geq 4B$ und $k \leq N^{1-\varepsilon}$ gilt

$$\ell(k,N) \geq \min\left\{\kappa(\varepsilon)\frac{kN}{B}\log_{M/B}\frac{N}{\max\{k,M\}}, \frac{1}{8}\kappa'(\varepsilon)kN\right\}.$$

Proof.

Multiplizieren mit dem **1**-Vektor ist Zeit invers zum Erzeugen von Matrizen
(Teilsummen statt Variablen) □

Erweiterung zu freiem Layout

Mehr Information nötig, um ein Programm zu beschreiben:

- Verfolge Zeit-vorwärts die Bewegung der Variablen (x_j);
- Verfolge Zeit-rückwärts die Bewegung von Teilsummen;
- Beobachte Multiplikationen ($a_{ij}x_j$) im Hauptspeicher.

▷ Dies bestimmt die Gestalt der Matrix.

$$\rightsquigarrow \quad \Omega\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{kM}\right), kN\right\}\right)$$

Erweiterung zu freiem Layout

Mehr Information nötig, um ein Programm zu beschreiben:

- Verfolge Zeit-vorwärts die Bewegung der Variablen (x_j);
- Verfolge Zeit-rückwärts die Bewegung von Teilsummen;
- Beobachte Multiplikationen ($a_{ij}x_j$) im Hauptspeicher.

▷ Dies bestimmt die Gestalt der Matrix.

$$\rightsquigarrow \quad \Omega\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{kM}\right), kN\right\}\right)$$

Ungleichung für freies Layout

Normalisieren: Elementare Produkte werden sofort erzeugt (ersetzen a_{ij}). Somit:

$$\binom{N}{k}^N \leq \left(\binom{M+B}{B}\ell\right)^{2\ell} \cdot \binom{2\ell BM}{kN}.$$

Löse nach ℓ mithilfe von

Lemma

Seien $b \geq 2$ und $s, t > 0$ mit $s \cdot t > 1$. Dann gilt für alle positiven reellen Zahlen x, dass $x \geq \frac{\log_b(s/x)}{t} \quad \Rightarrow \quad x \geq \frac{1}{2}\frac{\log_b(s \cdot t)}{t}$

Aussage der unteren Schranken

- Column-Major Layout

  $B > 2$, $M \geq 4B$ und $k \leq N^{1-\epsilon}$

$$\Theta\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{\max\{k,M\}}\right), kN\right\}\right)$$

- Freies Layout

  $B \geq 6$, $M \geq 3B$ und $k \leq \sqrt[3]{N}$

$$\Theta\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{kM}\right), kN\right\}\right)$$

Gilt für $k > 5$ auch wenn Eingabe- und Ausgabe-Vektor in beliebiger Form gespeichert werden können.

Technische Universität München

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen

Zusammenfassung der neuen Ergebnisse

- Es gibt dünn besetzte Matrizen, für die das sortierbasierte Programm optimal ist (bis auf eine Konstante)
- Fast alle dünn besetzten Matrizen sind schwierig
- Eine gleichverteilt zufällig gewählte Matrix ist schwierig

Erweiterungen:

- nicht quadratische Matrizen
- Skalarprodukte
- Parallelrechner mit Block-Kommunikation (statt I/O-Modell)

Offen:

- mit mehreren Vektoren gleichzeitig Multiplizieren
- Gestaltabhängige optimale Algorithmen