Bildentrauschung (A220)

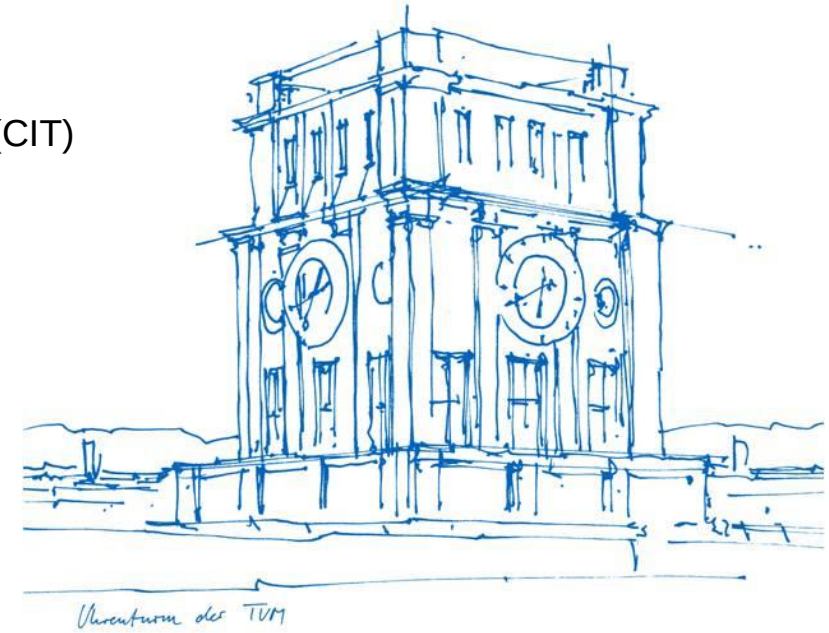
Feres Ben Fraj, Oussama Jeddou, Michael Ries

Technische Universität München

TUM School of Computation Information and Technology (CIT)

Lehrstuhl für Rechnerarchitektur und Parallele Systeme

Garching, 12. März 2024

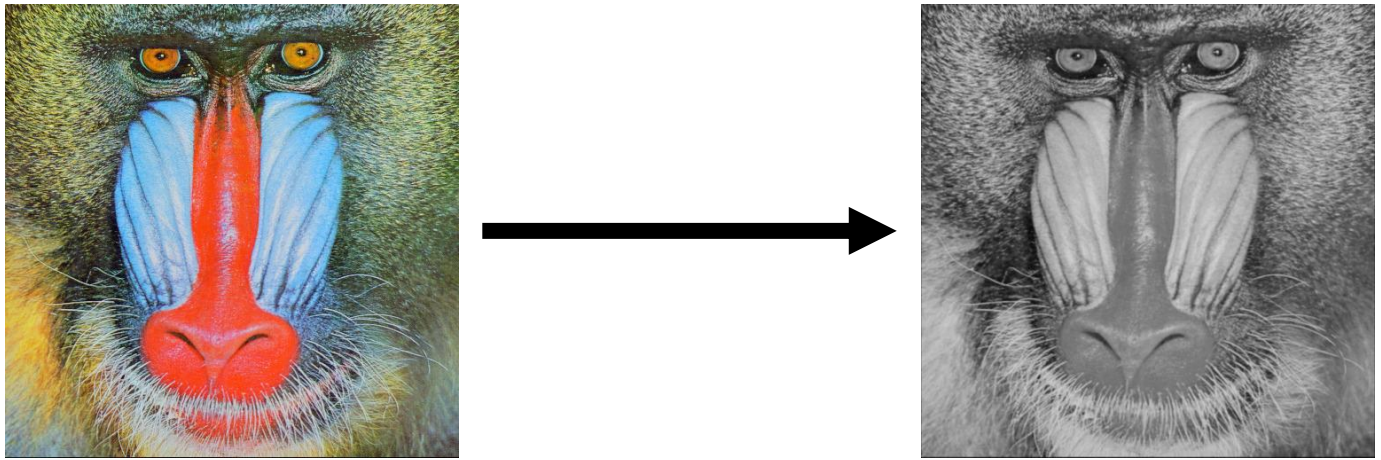


Lehrstuhl für Rechnerarchitektur und Parallele Systeme
TUM School of Computation, Information and Technology
Technische Universität München

Bildentrauschung (A220)

Feres Ben Fraj, Oussama Jeddou, Michael Ries

Garching, 12. März 2024



Graustufenkonvertierung

1. Einzelne Pixel werden mit den 3 Farbwerten R (rot), G (grün), B (blau) dargestellt: $P_{(x,y)} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$ mit

   $(x, y) \in \mathbb{D} = \{0, \dots, Breite - 1\} \times \{0, \dots, Höhe - 1\}$

2. Drei Koeffizienten a, b, c gewichten die Farbwerte. Verschiedene Koeffizienten erzeugen unterschiedliche Bildeffekte.

3. Jeder Pixel wird in Graustufe konvertiert durch: $D = \frac{a \times R + b \times G + c \times B}{a + b + c} \quad \forall \quad P_{(x,y)}$

4. Resultierendes Graustufenbild definiert als $Q_{(x,y)} = D$

Laplace-Filter

Bsp.: $Q^L_{(2,2)}, Q^L_{(5,5)}, Q^L_{(2,5)}$

1,1	1,2	1,3	1,4	1,5	1,6
2,1	2,2	2,3	2,4	2,5	2,6
3,1	3,2	3,3	3,4	3,5	3,6
4,1	4,2	4,3	4,4	4,5	4,6
5,1	5,2	5,3	5,4	5,5	5,6
6,1	6,2	6,3	6,4	6,5	6,6

1. Definition der Faltungsmatrix $M^L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

2. Definieren des Faltoperators * : $(f * g)[n] = \sum_{m \in \mathbb{D}} f[m]g[n-m]$

3. Berechnung des Laplace-Filters $Q^L \approx M^L * Q$

4. Durch Symmetrie und geringer Größe der Faltungsmatrix kann die Berechnung vereinfacht werden:
   $Q^L_{(x,y)} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} M_{(1+i,1+j)} \times Q_{(x+i,y+i)}$

Weichzeichnung

1. Definition der Faltungsmatrix $M^W = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

2. Berechnung der Weichzeichnung $Q^W = \frac{1}{16} \times M^W * Q$

Entrauschtes Graustufenbild

Resultierendes entrauschtes Graustufenbild ergibt sich durch:

$$Q'_{(x,y)} = \frac{|Q^L_{(x,y)}|}{1020} \times Q_{(x,y)} + \left(1 - \frac{|Q^L_{(x,y)}|}{1020}\right) * Q^W_{(x,y)}$$

mit:

- $Q_{(x,y)} = D = \frac{a \times R + b \times G + c \times B}{a + b + c} \quad \forall \quad P_{(x,y)}$
- $Q^L_{(x,y)} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} M_{(1+i,1+j)} \times Q_{(x+i,y+i)}$ mit $M^L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$
- $Q^W = \frac{1}{16} \times M^W * Q$ mit $M^W = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

Überblick Ansätze

1. Naiv

2. Optimierung durch Rechenreihenfolge

3. SIMD Operationen

4. Parallelisierung mit Threads

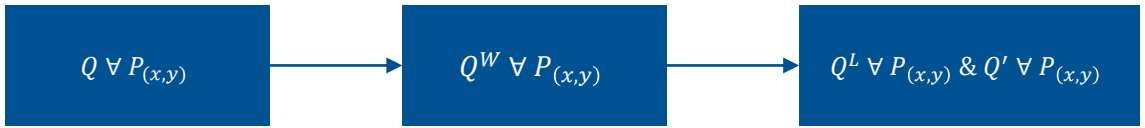
Im folgenden definieren wir n = breite x höhe = Anzahl an Pixeln

1. Naive Implementation

Programm:	Anzahl Zugriffe:	Speicher:
1. berechne $Q \forall P_{(x,y)}$	n	tmp1
2. berechne $Q^W \forall P_{(x,y)}$	n	tmp2
3. berechne $Q^L \forall P_{(x,y)}$ & berechne $Q' \forall P_{(x,y)}$	n	result

Optimierungen: keine

Laufzeit: 90-95 ns pro Pixel



2. Optimierung durch Rechenreihenfolge

Programm:	Anzahl Zugriffe:	Speicher:
1. berechne $Q \forall P_{(x,y)}$	n	tmp1
2. berechne Q^W & Q^L => $Q' \forall P_{(x,y)}$	n	result

Optimierungen:

- Ersparnis einer Schleife durch Kombinierung der Matrizen Berechnung
- Verringerter Speicherbedarf (Kein tmp2 für Zwischenspeicherung von Q^W)

Laufzeit: 63-65 ns pro Pixel

3. SIMD-Operationen

Programm:

1. Initialisierung erste Linie in tmp1 mit 0
2. berechne $Q \forall P_{(x,y)}$ (Als floats)
3. berechne Q^W & Q^L => $Q' \forall P_{(x,y)}$ mit SIMD

Optimierungen:

- Nutzung von SIMD-Operationen für Matrizen Rechnungen:
  - -> 3 x tmp1 (statt 9)
  - -> 3 x M^L (statt 9)
  - -> 3 x mul (statt 9)
  - -> 2 x add (statt 8)

Laufzeit: 37-40 ns pro Pixel

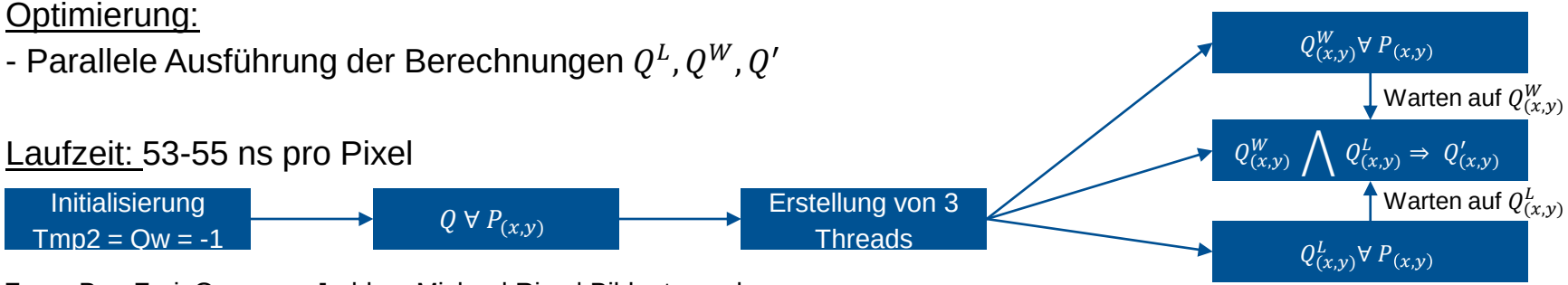
4. Parallelisierung mit Threads

Programm:	Anzahl Zugriffe:	Speicher:
1. Initialisierung tmp2, Qw (float buffer) mit -1	n	tmp2, Qw
2. berechne $Q \forall P_{(x,y)}$	n	tmp1
3. Starte 3 Threads: $Q^L Q^W Q'$		
3.1 berechne $Q^W \forall P_{(x,y)}$	n	Qw
3.2 berechne $Q^L \forall P_{(x,y)}$	n	tmp2
3.3 wenn $Q^W_{(x,y)} \wedge Q^L_{(x,y)} \Rightarrow$ berechne $Q'_{(x,y)}$	n	result

Optimierung:

- Parallele Ausführung der Berechnungen Q^L, Q^W, Q'

Laufzeit: 53-55 ns pro Pixel



Theoretische Laufzeitanalyse

1. Graustufenberechnung:

$$T_{graustufe} = \lambda_Q \times n$$

mit λ_x = Zeit zur Berechnung von x für 1 Pixel

$$T_{graustufe}(SIMD) = \lambda_Q \times n + k \times n$$

mit k: zusätzliche Laufzeit zum Nullen

2. Berechnung von Q^L, Q^W, Q'

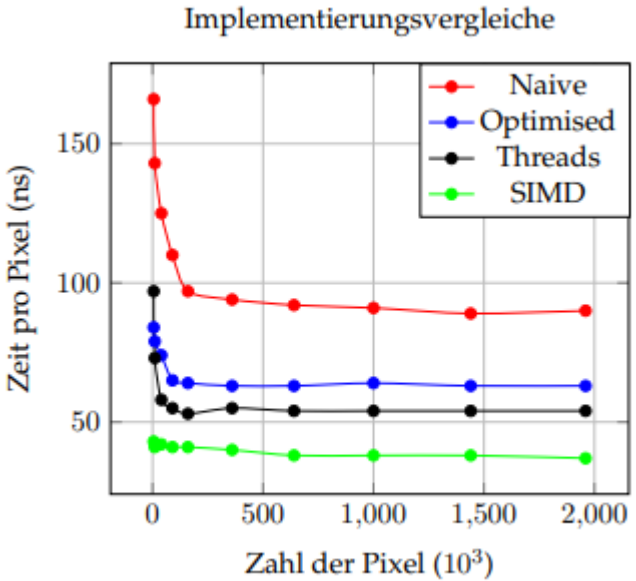
Naive: $T_{Q^L+Q^W+Q'} = \left(\lambda_{Q^W} + \lambda_{Speicherung\ Q^W}\right) \times n + (\lambda_{Q^L} + \lambda_{Q'} + \lambda_{Zugriff\ Q^W}) \times n$

Optimiert: $T_{Q^L+Q^W+Q'} = \left(\lambda_{Q^W} + \lambda_{Q^L} - 9 \times \lambda_{Zugriff\ Q} + \lambda_{Q'}\right) \times n$

SIMD: $T_{Q^L+Q^W+Q'} = \left(\frac{\lambda_{Q^W}+\lambda_{Q^L}-9 \times \lambda_{Zugriff\ Q}}{3} + \lambda_{Q'}\right) \times n$

Threads: $T_{Q^L+Q^W+Q'} = \left(\max\left(\lambda_{Q^W}, \lambda_{Q^L}, \lambda_{Q'}\right) + \lambda_{Speicher}\right) \times n + \lambda_{init} \times n$

Vergleich reelle Laufzeit & Speicherverbrauch

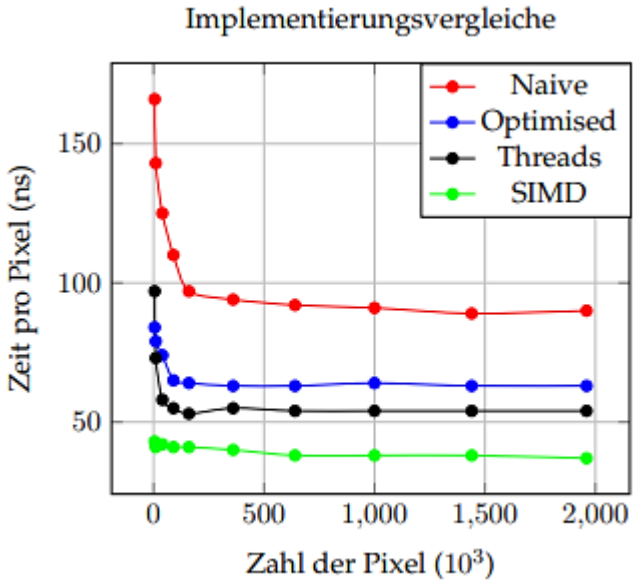


	Naive	Optimiert	SIMD	Threads
Img	3n	3n	3n	3n
Result	n	n	n	n
Tmp1	n	n	4n	n
Tmp2	n	0	0	n
Zusätzlich	0	0	0	Qw = 4n
Summe	6n	5n	8n	13n

Vergleich Laufzeiten

1. Naive 90-95 ns
2. Optimised 63-65 ns
3. Threads 53-55 ns
4. SIMD 37-40 ns

Wie zu erwarten SIMD schnellster -> $\sim \frac{1}{3}$ von Naive



Vergleich Speicherverbrauch

- SIMD & Threads verbrauchen meisten Speicher
  Grund: Double Nutzung anstelle von uint8

- Optimised verbraucht am wenigsten Speicher

	Naive	Optimised	SIMD	Threads
Img	3n	3n	3n	3n
Result	n	n	n	n
Tmp1	n	n	4n	n
Tmp2	n	0	0	n
Zusätzlich	0	0	0	Qw = 4n
Summe	6n	5n	8n	13n

Zusammenfassung

- SIMD generell am besten = große Laufzeitverbesserung mit leicht größerem Speicherverbrauch
- Threads = große Laufzeitverbesserung ABER großer Speicherverbrauch
- Optimised = sehr Speichereffizient & guter Laufzeitverbesserung