

# Cache-optimierte QR-Zerlegung

## Bachelorkolloquium

Florian Krötz

Universität Ulm

15. Oktober 2018

## Cache-optimierte QR-Zerlegung

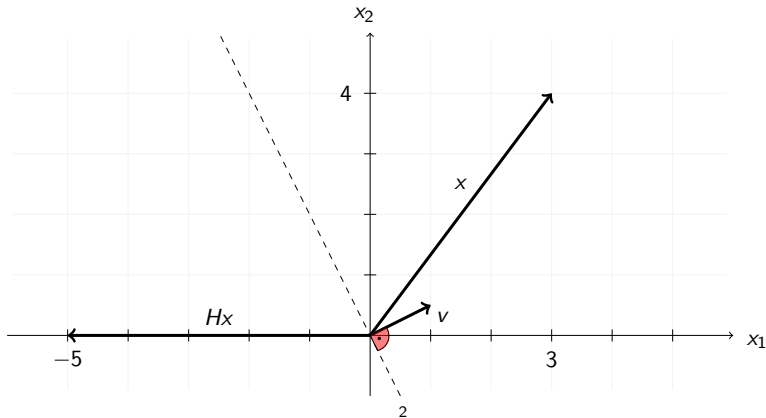
►  $A = QR$

$$\left( \begin{array}{|c|} \hline A \\ \hline \end{array} \right) = \left( \begin{array}{|c|} \hline Q \\ \hline \end{array} \right) * \left( \begin{array}{|c|} \hline \begin{array}{c} \text{ } \end{array} \\ \hline \end{array} \right)$$

- QR-Zerlegung mittels Householder-Transformation
- Cache-optimierten Algorithmus implementieren
- Anwendungen:
  - LGS  $Ax = b$  lösen mit QR
  - Lineares Ausgleichsproblem mittels kleinstem Fehlerquadrat
  - Kernoperation im QR-Verfahren

## Householder-Transformation

$$H = I - 2 \frac{vv^T}{v^T v}$$



# Householder-Transformation

- ▶ Householder-Vektor berechnen

- ▶ Normieren  $v_1 = 1$

- ▶  $\tau = \frac{2}{v^T v} \implies H = I - 2 \frac{vv^T}{v^T v} = I - \tau vv^T$

- ▶ Householder-Transformation anwenden

$$HA = (I - \tau vv^T)A = A - \tau(vv^T)A = A - \tau v(v^T A)$$

- ▶ Vektor-Rechenoperationen und Vektor-Matrix-Rechenoperationen

## QR-Zerlegung mittels Householder-Transformation

►  $A = QR$

$$H_1 A = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$



$$H_1 = (\hat{H}_1) \quad , \quad H_2 = \left( \begin{array}{c|c} I_1 & 0 \\ \hline 0 & \hat{H}_2 \end{array} \right) \quad , \quad H_i = \left( \begin{array}{c|c} I_{i-1} & 0 \\ \hline 0 & \hat{H}_i \end{array} \right)$$

## QR-Zerlegung mittels Householder-Transformation

►  $A = QR$

$$H_2 H_1 A = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix}$$



$$H_1 = (\hat{H}_1) \quad , \quad H_2 = \left( \begin{array}{c|c} I_1 & 0 \\ \hline 0 & \hat{H}_2 \end{array} \right) \quad , \quad H_i = \left( \begin{array}{c|c} I_{i-1} & 0 \\ \hline 0 & \hat{H}_i \end{array} \right)$$

## QR-Zerlegung mittels Householder-Transformation

►  $A = QR$

$$H_3 H_2 H_1 A = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix}$$



$$H_1 = (\hat{H}_1) \quad , \quad H_2 = \left( \begin{array}{c|c} I_1 & 0 \\ \hline 0 & \hat{H}_2 \end{array} \right) \quad , \quad H_i = \left( \begin{array}{c|c} I_{i-1} & 0 \\ \hline 0 & \hat{H}_i \end{array} \right)$$

## QR-Zerlegung mittels Householder-Transformation

►  $A = QR$

$$H_3 H_2 H_1 A = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\ v_2^{(1)} & r_{2,2} & r_{2,3} & r_{2,4} \\ v_3^{(1)} & v_3^{(2)} & r_{3,3} & r_{3,4} \\ v_4^{(1)} & v_4^{(2)} & v_4^{(3)} & r_{4,4} \end{pmatrix}$$



$$H_1 = (\hat{H}_1) \quad , \quad H_2 = \left( \begin{array}{c|c} l_1 & 0 \\ \hline 0 & \hat{H}_2 \end{array} \right) \quad , \quad H_i = \left( \begin{array}{c|c} l_{i-1} & 0 \\ \hline 0 & \hat{H}_i \end{array} \right)$$



## Benchmark

- ▶ Aufwand QR-Zerlegung mittels Householder-Transformation  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$

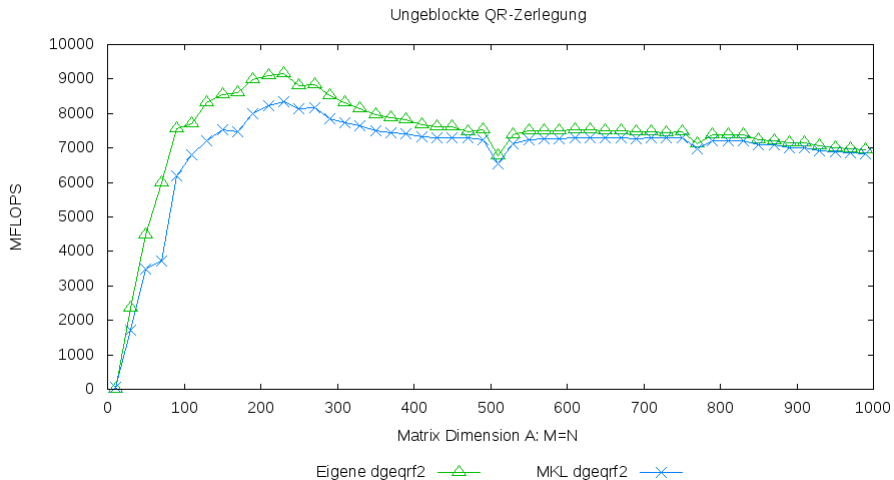
$$\#QR = n \cdot \left( \frac{23}{6} + m + \frac{n}{2} + n \cdot \left( m - \frac{n}{3} \right) + \frac{5}{6} + n \cdot \left( \frac{1}{2} + m - \frac{n}{3} \right) \right) = \mathcal{O}(n^2 m)$$



$$\text{FLOPS} = \frac{\text{Aufwand}}{\Delta t}$$

- ▶ *Peak performance*  
25,6 GFLOPS auf dem Testsystem mit i5-3470-CPU

# Ungeblockte QR-Zerlegung



## Mehrere Householder-Transformationen anwenden

- ▶ Ansatz

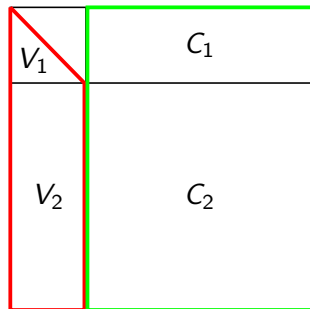
$$\hat{H} = H_1 H_2 \dots H_k = I - VTV^T \quad \text{mit} \quad H_i = I - \tau_i v_i v_i^T$$

- ▶ Vektor-Matrix-Rechenoperationen

- ▶ Householder-Transformationen anwenden

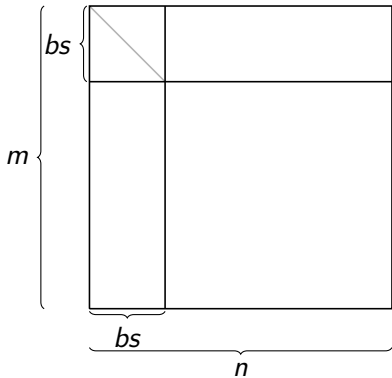
$$C \leftarrow \hat{H}C = C - VTV^T C$$

- ▶ Matrix-Matrix-Produkte



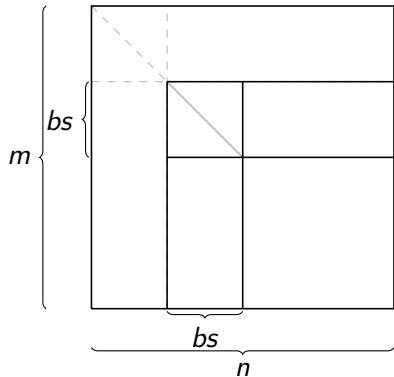
## Geblockte QR-Zerlegung

► Matrix A



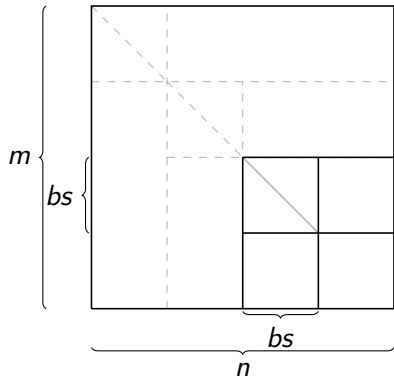
# Geblockte QR-Zerlegung

► Matrix A



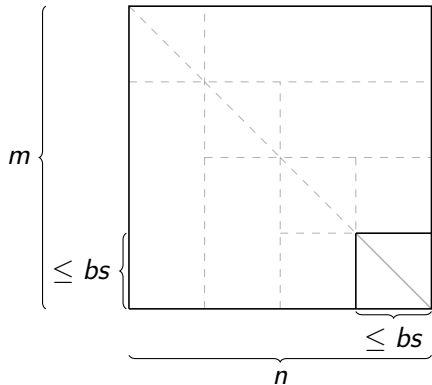
# Geblockte QR-Zerlegung

► Matrix A

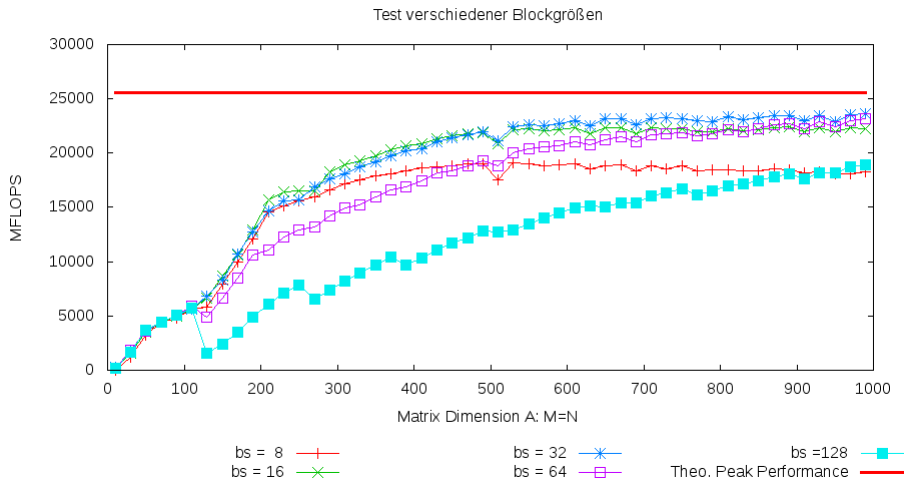


# Geblockte QR-Zerlegung

## ► Matrix A

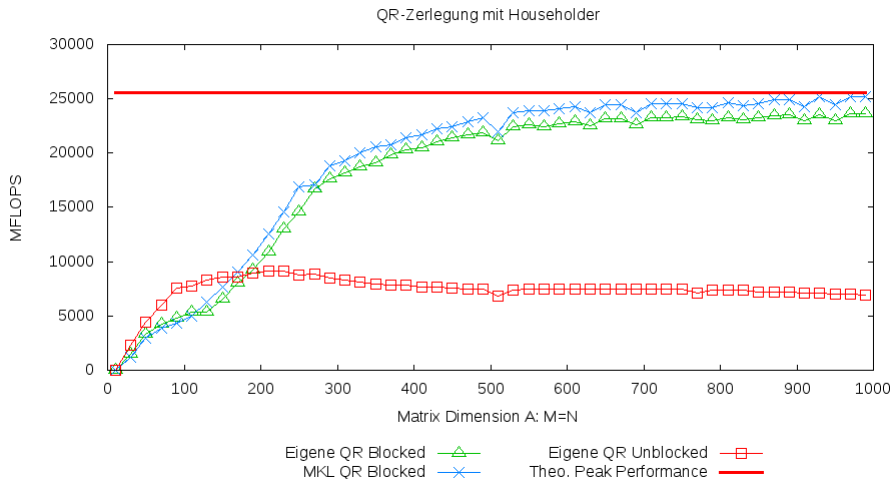


# Test verschiedener Blockgrößen





# Geblockte QR-Zerlegung



## Fazit

- ▶ Der ungeblockte Algorithmus erreicht nicht die *peak performance*.
- ▶ Der eigene ungeblockte Algorithmus ist etwa 5% schneller als MKL.
- ▶ Der geblockte Algorithmus erreicht fast die *peak performance*.
- ▶ Der geblockte Algorithmus *dgeqrf* der MKL ist etwas schneller. Der selbst implementierte Algorithmus erreicht bis zu 94% die Performance der MKL.
- ▶ Der geblockte Algorithmus ist um den Faktor 3 schneller als der ungeblockte.

## Ausblick

- ▶ Ist es möglich mit einem rekursiven geblockten Algorithmus die Performance der MKL zu erreichen oder sogar zu übertreffen?
- ▶ Welche Blas-Funktionen haben den größten Einfluss auf die Performance?