Right Looking GE

New blocking:

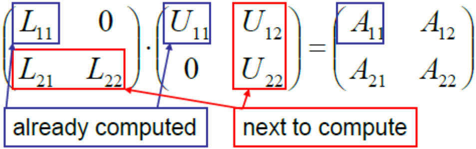
$$\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

already computed

next to compute

Right Looking GE

New blocking:

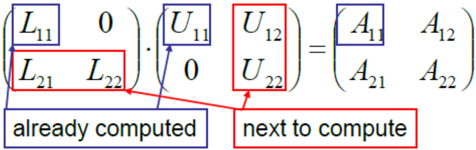


$$\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

- Start with $L_{11} U_{11} = A_{11}$ (small LU-decomposition).

Right Looking GE

New blocking:

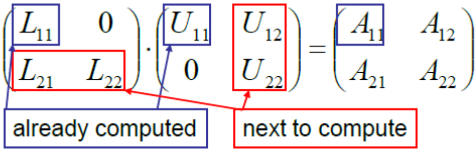


$$\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

- Start with $L_{11} U_{11} = A_{11}$ (small LU-decomposition).
- Equations $L_{21} U_{11} = A_{21}$ and $L_{11} U_{12} = A_{12}$ by triangular solves gives L_{21} and U_{12}.

Right Looking GE

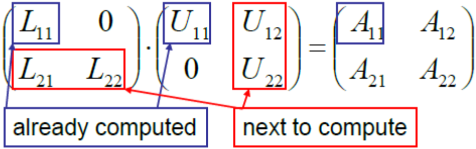
New blocking:



- Start with $L_{11}U_{11} = A_{11}$ (small LU-decomposition).
- Equations $L_{21}U_{11} = A_{21}$ and $L_{11}U_{12} = A_{12}$ by triangular solves gives L_{21} and U_{12}.
- It remains $L_{22}U_{22} = A_{22} - L_{21}U_{12} = \hat{A}_{22}$

Right Looking GE

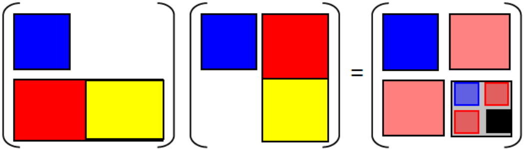
New blocking:



- Start with $L_{11} U_{11} = A_{11}$ (small LU-decomposition).
- Equations $L_{21} U_{11} = A_{21}$ and $L_{11} U_{12} = A_{12}$ by triangular solves gives L_{21} and U_{12}.
- It remains $L_{22} U_{22} = A_{22} - L_{21} U_{12} = \hat{A}_{22}$
- To compute the LU-decomposition of modified A_{22} repeat 2×2-blocking for A_{22} and apply recursively.

Block Structure

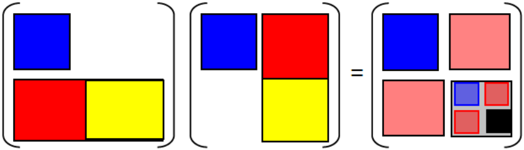
Intermediate block structure:



Solve for blue and both red blocks.

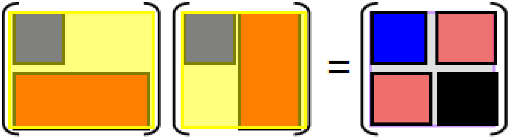
Block Structure

Intermediate block structure:



Solve for blue and both red blocks.

Reconfigure the block structure:



Repeat until done.

Comparison and Overview

- In comparison, all methods
  - have nearly same efficiency in parallel
  - but better performance (in sequential or parallel) than the unblocked variants because they are based on BLAS-3.

Comparison and Overview

- In comparison, all methods
  - have nearly same efficiency in parallel
  - but better performance (in sequential or parallel) than the unblocked variants because they are based on BLAS-3.
- Elementary steps of all blocking methods:
  - Matrix-Matrix product and sum (easy to parallelize)
  - Couple of triangular solves (easy to parallelize)
  - Small LU-decomposition (parallelizable for long rows)

Comparison and Overview

- In comparison, all methods
  - have nearly same efficiency in parallel
  - but better performance (in sequential or parallel) than the unblocked variants because they are based on BLAS-3.
- Elementary steps of all blocking methods:
  - Matrix-Matrix product and sum (easy to parallelize)
  - Couple of triangular solves (easy to parallelize)
  - Small LU-decomposition (parallelizable for long rows)
- Crout and right looking slightly better because more flops in matrix-updates and less triangular solves respectively *LU*-decompositions.

3.3. *QR*-Decomposition with Householder Matrices

3.3.1. QR-decomposition

- Gaussian elimination $\rightarrow$ *LU*-decomposition: sometimes numerically not stable, over/underdetermined systems

3.4. QR-Decomposition with Householder Matrices

3.4.1. QR-decomposition

- Gaussian elimination $\rightarrow$ LU-decomposition: sometimes numerically not stable, over/underdetermined systems
- Improvement:

  QR-decomposition $A = QR$ with Q orthogonal, R triangular,
  Solve linear system Ax=b numerically stable via

$$b = Ax = QRx \Leftrightarrow Rx = Q^T b$$

  by cheap matrix-vector multiplication and triangular solve.

Overdetermined Systems

- $Ax = b$ with

  A being $m \times n$ matrix, $n \ll m$

  x vector of length n

  b vector of length m

Overdetermined Systems

- $Ax = b$ with

  A being $m \times n$ matrix, $n \ll m$
  x vector of length n
  b vector of length m

- Best **approximate** solution by solving minimization

$$\min_x \|Ax - b\|_2^2 = \min_x (x^T A^T Ax - 2x^T A^T b + b^T b)$$

- Gradient equal zero $\Leftrightarrow A^T Ax = A^T b$ (normal equations)

Overdetermined Systems

- $Ax = b$ with

  A being $m \times n$ matrix, $n \ll m$

  x vector of length n

  b vector of length m

- Best **approximate** solution by solving minimization

$$\min_x \|Ax - b\|_2^2 = \min_x (x^T A^T Ax - 2x^T A^T b + b^T b)$$

- Gradient equal zero $\Leftrightarrow A^T Ax = A^T b$ (normal equations)

- Solution by considering linear system $A^T A$, but condition number worse:

$$\text{cond}(A^T A) = \text{cond}(A)^2$$

Advantage of QR-Decomposition

$$A = QR, \quad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, \quad \text{cond}(R_1) = \text{cond}(A), \quad \hat{b} = Q^T b = \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix}$$

$$A^T A x = A^T b \;\Leftrightarrow\; (QR)^T (QR) x = (QR)^T b \;\Leftrightarrow$$

$$R^T R x = R^T (Q^T b) \;\Leftrightarrow\; (R_1^T \;\; 0) \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x = (R_1^T \;\; 0) \hat{b} \;\Leftrightarrow$$

$$R_1^T R_1 x = (R_1^T \;\; 0) \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix} \;\Leftrightarrow\; R_1^T R_1 x = R_1^T \hat{b}_1 \;\Leftrightarrow\; R_1 x = \hat{b}_1$$

Advantage of QR-Decomposition

$$A = QR, \quad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, \quad \text{cond}(R_1) = \text{cond}(A), \quad \hat{b} = Q^T b = \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix}$$

$$A^T A x = A^T b \;\Leftrightarrow\; (QR)^T (QR) x = (QR)^T b \;\Leftrightarrow$$

$$R^T R x = R^T (Q^T b) \;\Leftrightarrow\; (R_1^T \;\; 0) \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x = (R_1^T \;\; 0) \hat{b} \;\Leftrightarrow$$

$$R_1^T R_1 x = (R_1^T \;\; 0) \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix} \;\Leftrightarrow\; R_1^T R_1 x = R_1^T \hat{b}_1 \;\Leftrightarrow\; R_1 x = \hat{b}_1$$

- Instead of solving the normal equations we only have to consider the triangular system in R_1 .
- Cheap and better condition number.

3.4.2. Householder Method

- Define special orthogonal and simple matrices H called Householder matrices (compare Givens):

$$u \in \mathbb{R}^n,\ \|u\|_2 = 1: \quad H = I - 2uu^T$$

3.4.3. Householder Method

- Define special orthogonal and simple matrices H called Householder matrices (compare Givens):

$$u \in \mathbb{R}^n,\ \|u\|_2 = 1 : \quad H = I - 2uu^T$$

- H as rank-1 perturbation of the identity is symmetric, idempotent and orthogonal:

$$H^T = I - 2uu^T = H$$

$$H^T H = H^2 = (I-2uu^T)(I-2uu^T) = I-2uu^T-2uu^T+4u\underbrace{u^T u}_{=1}u^T = I$$

3.4.4. Householder Method

- Define special orthogonal and simple matrices H called Householder matrices (compare Givens):

$$u \in \mathbb{R}^n,\ \|u\|_2 = 1: \quad H = I - 2uu^T$$

- H as rank-1 perturbation of the identity is symmetric, idempotent and orthogonal:

$$H^T = I - 2uu^T = H$$

$$H^T H = H^2 = (I - 2uu^T)(I - 2uu^T) = I - 2uu^T - 2uu^T + 4u \underbrace{u^T u}_{=1} u^T = I$$

- For complex problems:
  orthogonal $\rightarrow$ unitary, symmetric $\rightarrow$ hermitian

Householder Method (cont.)

- Use H_1 with appropriate vector u_1 to eliminate first column of A

$$H_1 A = (I - 2u_1 u_1^T)(a_1 \ \cdots \ a_m) = (a_1 - 2(u_1^T a_1)u_1 \ \cdots \ *) = \begin{pmatrix} \alpha & * \\ 0 & * \\ \vdots & \vdots \\ 0 & * \end{pmatrix}$$

Householder Method (cont.)

- Use H_1 with appropriate vector u_1 to eliminate first column of A

$$H_1 A = (I - 2u_1 u_1^T)(a_1 \ \cdots \ a_m) = (a_1 - 2(u_1^T a_1)u_1 \ \cdots \ *) = \begin{pmatrix} \alpha & * \\ 0 & * \\ \vdots & \vdots \\ 0 & * \end{pmatrix}$$

- To satisfy this equation we have to find a vector u_1 of length 1 with

$$a_1 - 2(u_1^T a_1)u_1 = \alpha e_1$$

Householder Method (cont.)

- Use H_1 with appropriate vector u_1 to eliminate first column of A

$$H_1A = (I - 2u_1u_1^T)(a_1 \; \cdots \; a_m) = (a_1 - 2(u_1^Ta_1)u_1 \; \cdots \; *) = \begin{pmatrix} \alpha & * \\ 0 & * \\ \vdots & \vdots \\ 0 & * \end{pmatrix}$$

- To satisfy this equation we have to find a vector u_1 of length 1 with

$$a_1 - 2(u_1^Ta_1)u_1 = \alpha e_1$$

- Because H_1 is orthogonal it holds:

$$\|H_1a_1\|_2 = \|a_1\|_2 = \|\alpha e_1\|_2 = |\alpha| \Rightarrow \alpha = \pm\|a_1\|_2, \text{ e.g. } \alpha = \|a_1\|_2$$

$$u_1 = \frac{a_1 - \|a_1\|_2 e_1}{2(u_1^Ta_1)} = \frac{a_1 - \|a_1\|_2 e_1}{\|a_1 - \|a_1\|_2 e_1\|_2}$$

Householder Method (cont. 2)

- Repeat for all columns of A

$$H_1 A = H_1 A_1 = (I - 2u_1 u_1^T)A = \left(\begin{array}{c|ccc} \|a_1\|_2 & * & \cdots & * \\ \hline 0 & & & \\ \vdots & & A_2 & \\ 0 & & & \end{array}\right)$$

Householder Method (cont. 2)

- Repeat for all columns of A

$$H_1 A = H_1 A_1 = (I - 2u_1 u_1^T)A = \left(\begin{array}{c|ccc} \|a_1\|_2 & * & \cdots & * \\ \hline 0 & & & \\ \vdots & & A_2 & \\ 0 & & & \end{array}\right)$$

- Apply the same procedure on A_2, $(n-1) \times (m-1)$ matrix.

$$\tilde{H}_2 A_2 = (I - 2\tilde{u}_2 \tilde{u}_2^T)A_2 = \left(\begin{array}{c|ccc} \alpha_2 & * & \cdots & * \\ \hline 0 & & & \\ \vdots & & A_3 & \\ 0 & & & \end{array}\right)$$

Householder Method (cont. 2)

- Repeat for all columns of A

$$H_1 A = H_1 A_1 = (I - 2u_1 u_1^T)A = \left(\begin{array}{c|ccc} \|a_1\|_2 & * & \cdots & * \\ \hline 0 & & & \\ \vdots & & A_2 & \\ 0 & & & \end{array}\right)$$

- Apply the same procedure on A_2, $(n-1) \times (m-1)$ matrix.

$$\tilde{H}_2 A_2 = (I - 2\tilde{u}_2 \tilde{u}_2^T)A_2 = \left(\begin{array}{c|ccc} \alpha_2 & * & \cdots & * \\ \hline 0 & & & \\ \vdots & & A_3 & \\ 0 & & & \end{array}\right)$$

- Extend

$$u_2 := \begin{pmatrix} 0 \\ \tilde{u}_2 \end{pmatrix}, \qquad H_2 := I - 2u_2 u_2^T = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \tilde{H}_2 & \\ 0 & & & \end{pmatrix}$$

Householder Method (cont. 3)

- For column $1, 2, \ldots, m$ this gives Householder matrices $H_1, \ldots, H_m$ with

$$\underbrace{H_m \cdots H_2 H_1}_{=Q^T} \cdot A = H_m \cdots H_3 \cdot \left(\begin{array}{cc|ccc} \alpha_1 & * & * & \cdots & * \\ 0 & \alpha_2 & * & \cdots & * \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & A_3 & \\ 0 & 0 & & & \end{array}\right) = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} =: R$$

Householder Method (cont. 3)

- For column $1, 2, \ldots, m$ this gives Householder matrices $H_1, \ldots, H_m$ with

$$\underbrace{H_m \cdots H_2 H_1}_{= Q^T} \cdot A = H_m \cdots H_3 \cdot \left(\begin{array}{cc|ccc} \alpha_1 & * & * & \cdots & * \\ 0 & \alpha_2 & * & \cdots & * \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & A_3 & \\ 0 & 0 & & & \end{array} \right) = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} =: R$$

- Hence:

$$A = QR, \qquad Q := (H_m \cdots H_2 H_1)^T = H_1 H_2 \cdots H_m$$

Householder Method (cont. 3)

- For column $1, 2, \ldots, m$ this gives Householder matrices $H_1, \ldots, H_m$ with

$$\underbrace{H_m \cdots H_2 H_1}_{= Q^T} \cdot A = H_m \cdots H_3 \cdot \left(\begin{array}{cc|ccc} \alpha_1 & * & * & \cdots & * \\ 0 & \alpha_2 & * & \cdots & * \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & A_3 & \\ 0 & 0 & & & \end{array}\right) = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} =: R$$

- Hence:

$$A = QR, \qquad Q := (H_m \cdots H_2 H_1)^T = H_1 H_2 \cdots H_m$$

- Remark: for $m = n$: $H_1, \ldots, H_{m-1}$ is enough, because last column is scalar.

3.4.5. Householder Method in Parallel - Blockwise

Idea: work again blockwise. Allows BLAS3, matrix-times-matrix operations.

- In a first step compute u_1 and the application of H_1 on the first k columns of A. Do not compute $H_1 A$ fully!

3.4.6. Householder Method in Parallel - Blockwise

Idea: work again blockwise. Allows BLAS3, matrix-times-matrix operations.

- In a first step compute u_1 and the application of H_1 on the first k columns of A. Do not compute $H_1 A$ fully!
- Then compute $u_2, \ldots, u_k$ and the application of $H_1 \ldots H_k$ on the first columns of A.

$$H_k \cdots H_1 (A_1 \quad A_2) = (H_k \cdots H_1 A_1 \quad (H_k \cdots H_1) A_2) = (A_1^{(k)} \quad VA_2)$$

3.4.7. Householder Method in Parallel - Blockwise

Idea: work again blockwise. Allows BLAS3, matrix-times-matrix operations.

- In a first step compute u_1 and the application of H_1 on the first k columns of A. Do not compute $H_1 A$ fully!
- Then compute $u_2, \ldots, u_k$ and the application of $H_1 \ldots H_k$ on the first columns of A.

$$H_k \cdots H_1 (A_1 \quad A_2) = (H_k \cdots H_1 A_1 \quad (H_k \cdots H_1) A_2) = (A_1^{(k)} \quad VA_2)$$

- Still to compute: VA_2.
- How can we take advantage of parallelism in this computation? $\rightarrow$ represent V in special form that allows fast and parallel evaluation of VA_2.

Property of Householder matrices

Theorem 3: For Householder matrices $H_k, \ldots, H_i$ it holds

$$H_k \cdots H_i = (I - 2u_k u_k^T) \cdots (I - 2u_i u_i^T) = I - \underbrace{\begin{pmatrix} u_k & \cdots & u_i \end{pmatrix}}_{=:\, Y} T_i \begin{pmatrix} u_k^T \\ \vdots \\ u_i^T \end{pmatrix}$$

with T_i being upper triangular.

Property of Householder matrices

Theorem 3: For Householder matrices $H_k, \dots, H_i$ it holds

$$H_k \cdots H_i = (I - 2u_k u_k^T) \cdots (I - 2u_i u_i^T) = I - \underbrace{\begin{pmatrix} u_k & \cdots & u_i \end{pmatrix}}_{=:\, Y} T_i \begin{pmatrix} u_k^T \\ \vdots \\ u_i^T \end{pmatrix}$$

with T_i being upper triangular.

Proof by induction:

Representation obviously fulfilled for one Householder mtx $i = k$.

Assume, representation holds for $H_k, \dots, H_i$. Then ... (next slide)

Property of Householder matrices (cont.)

$$\left[(I - 2u_k u_k^T)\cdots(I - 2u_i u_i^T)\right](I - 2u_{i-1}u_{i-1}^T) =$$

$$= \left[I - \begin{pmatrix} u_k & \cdots & u_i \end{pmatrix} T_i \begin{pmatrix} u_k^T \\ \vdots \\ u_i^T \end{pmatrix}\right] \cdot (I - 2u_{i-1}u_{i-1}^T) =$$

$$= I - 2u_{i-1}u_{i-1}^T - \begin{pmatrix} u_k & \cdots & u_i \end{pmatrix} T_i \begin{pmatrix} u_k^T \\ \vdots \\ u_i^T \end{pmatrix} + 2\begin{pmatrix} u_k & \cdots & u_i \end{pmatrix} \underbrace{T_i \begin{pmatrix} u_k^T u_{i-1} \\ \vdots \\ u_i^T u_{i-1} \end{pmatrix}}_{=:\, y} u_{i-1}^T =$$

$$= I - \begin{pmatrix} u_k & \cdots & u_i & u_{i-1} \end{pmatrix} \cdot \begin{pmatrix} T_i & -2y \\ 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} u_k^T \\ \vdots \\ u_i^T \\ u_{i-1}^T \end{pmatrix}$$

Algorithm for parallel Householder

Computation of $H_k \cdots H_i A = V_{ki} A = (I - YTY^T)A$ in the form

$$V_{ki} A = V_{ki} (A_1 \quad A_2) = (* \quad V_{ki} A_2)$$

and

$$V_{ki} A_2 = (I - YTY^T) A_2 = A_2 - Y[T(Y^T A_2)]$$

Algorithm for parallel Householder

Computation of $H_k \cdots H_i A = V_{ki} A = (I - YTY^T)A$ in the form

$$V_{ki} A = V_{ki}(A_1 \quad A_2) = (* \quad V_{ki} A_2)$$

and

$$V_{ki} A_2 = (I - YTY^T) A_2 = A_2 - Y[T(Y^T A_2)]$$

Algorithm:

- Compute u_1 and $H_1 A_1$; u_2 and $H_2 A_1$; $\ldots$; u_k and $H_k A_1$ (sequential)
- Compute VA_2 (parallel as matrix-times-matrix BLAS3)
- Repeat with indices $k+1, \ldots, 2k$; $2k+1, \ldots, 3k$; $\ldots$

Algorithm for parallel Householder

Computation of $H_k \cdots H_i A = V_{ki} A = (I - YTY^T)A$ in the form

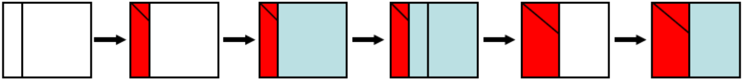
$$V_{ki} A = V_{ki}(A_1 \quad A_2) = (* \quad V_{ki} A_2)$$

and

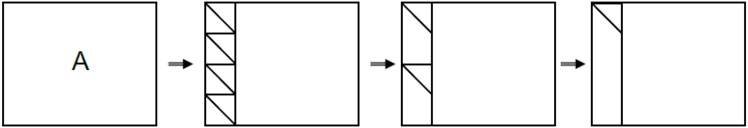
$$V_{ki} A_2 = (I - YTY^T)A_2 = A_2 - Y[T(Y^T A_2)]$$

Algorithm:

- Compute u_1 and $H_1 A_1$; u_2 and $H_2 A_1$; ...; u_k and $H_k A_1$ (sequential)
- Compute VA_2 (parallel as matrix-times-matrix BLAS3)
- Repeat with indices $k+1, \ldots, 2k$; $2k+1, \ldots, 3k$; ...



Communication Avoiding QR



- First level: Four independent QR-factorisations

$\downarrow$

- Second level: Two independent reduced QR-factorisations

$\downarrow$

- Last level: One reduced QR-factorisation

Tall Skinny QR

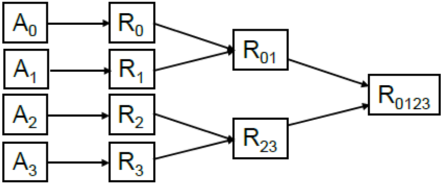
$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \begin{pmatrix} Q_0 R_0 \\ Q_1 R_1 \\ Q_2 R_2 \\ Q_3 R_3 \end{pmatrix} = \begin{pmatrix} Q_0 & & & \\ & Q_1 & & \\ & & Q_2 & \\ & & & Q_3 \end{pmatrix} \begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \end{pmatrix}$$

$$\begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} R_0 \\ R_1 \end{pmatrix} \\ \begin{pmatrix} R_2 \\ R_3 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} Q_{01} R_{01} \\ Q_{23} R_{23} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ & Q_{23} \end{pmatrix} \begin{pmatrix} R_{01} \\ R_{23} \end{pmatrix}$$

$$\begin{pmatrix} R_{01} \\ R_{23} \end{pmatrix} = Q_{0123} R_{0123}$$

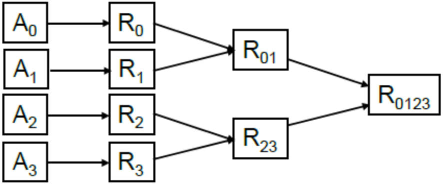
Tall Skinny QR (cont.)

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \left\{ \begin{pmatrix} Q_0 & & & \\ & Q_1 & & \\ & & Q_2 & \\ & & & Q_3 \end{pmatrix} \cdot \begin{pmatrix} Q_{01} & \\ & Q_{23} \end{pmatrix} \cdot Q_{0123} \right\} \cdot R_{0123}$$



Tall Skinny QR (cont.)

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \left\{ \begin{pmatrix} Q_0 & & & \\ & Q_1 & & \\ & & Q_2 & \\ & & & Q_3 \end{pmatrix} \cdot \begin{pmatrix} Q_{01} & \\ & Q_{23} \end{pmatrix} \cdot Q_{0123} \right\} \cdot R_{0123}$$



Advantage:
Messages in $\mathcal{O}(\log(P))$ compared to $\mathcal{O}(2n\log(P))$ for ScaLAPACK.

Cholesky QR Decomp. for Tall Skinny A

$$A^T A = LL^T$$

$$Q^T := L^{-1} A^T$$

Cholesky QR Decomp. for Tall Skinny A

$$A^T A = LL^T$$

$$Q^T := L^{-1} A^T$$

$$Q^T Q = L^{-1} A^T A L^{-T} = I$$

Cholesky QR Decomp. for Tall Skinny A

$$A^T A = LL^T$$

$$Q^T := L^{-1} A^T$$

$$Q^T Q = L^{-1} A^T A L^{-T} = I$$

$$A = QL^T = QR = \begin{pmatrix} Q & Q^\perp \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Cholesky QR Decomp. for Tall Skinny A

$$A^T A = LL^T$$

$$Q^T := L^{-1} A^T$$

$$Q^T Q = L^{-1} A^T A L^{-T} = I$$

$$A = QL^T = QR = \begin{pmatrix} Q & Q^{\perp} \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Advantage: Computation of $A^T A$ fully parallel, only small Cholesky decomposition L.

Cholesky QR Decomp. for Tall Skinny A

$$A^T A = LL^T$$

$$Q^T := L^{-1} A^T$$

$$Q^T Q = L^{-1} A^T A L^{-T} = I$$

$$A = QL^T = QR = \begin{pmatrix} Q & Q^\perp \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Advantage: Computation of $A^T A$ fully parallel, only small Cholesky decomposition L.

Disadvantage: Numerical stability.

Cholesky QR Decomp. for Tall Skinny A

$$A^T A = LL^T$$

$$Q^T := L^{-1} A^T$$

$$Q^T Q = L^{-1} A^T A L^{-T} = I$$

$$A = QL^T = QR = \begin{pmatrix} Q & Q^{\perp} \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Advantage: Computation of $A^T A$ fully parallel, only small Cholesky decomposition L.

Disadvantage: Numerical stability.

Compromise:
Use Cholesky-QR only for well-conditioned A.

Parallel Numerics, WT 2016/2017

4 Sparse Matrices

Contents

4.1. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3) \rightarrow$ too costly for most applications, esp. for fine discretization (large n)

4.2. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3) \rightarrow$ too costly for most applications, esp. for fine discretization (large n)
- Formulate given problem in clever way that leads to a linear system that is sparse: $\mathcal{O}(n)$, solution $\mathcal{O}(n)$?
  - (that is structured: storage $\mathcal{O}(n)$, solution $O(n \log(n))$, e.g., FFT)
  - (that is dense, but reduced from, e.g., 3D to 2D)
  - (based on sparse grids)
  - (based on tensor approximations)

4.3. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3)$ $\rightarrow$ too costly for most applications, esp. for fine discretization (large n)
- Formulate given problem in clever way that leads to a linear system that is sparse: $\mathcal{O}(n)$, solution $\mathcal{O}(n)$?
  - (that is structured: storage $\mathcal{O}(n)$, solution $O(n\log(n))$, e.g., FFT)
  - (that is dense, but reduced from, e.g., 3D to 2D)
  - (based on sparse grids)
  - (based on tensor approximations)
- Examples:
  - tridiagonal matrix
  - banded matrix
  - block band matrix

Sparse Matrix Example

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

Additionally we need to store:

- the size of the matrix $n = 5$
- the number of nonzero entries $\text{nnz} = 12$

4.3.1. Storage in Coordinate Form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

To store:

- n
- nnz
- $2 \cdot$ nnz integer for row and column indices in JR and JC
- nnz float in AA

4.3.2. Storage in Coordinate Form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

To store:

- n
- nnz
- $2 \cdot$ nnz integer for row and column indices in JR and JC
- nnz float in AA

No sorting included. Redundant information.

Storage in Coordinate Form (cont.)

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Storage in Coordinate Form (cont.)

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Pseudocode for computing $c = A \cdot b$:

$c = 0;$

$\texttt{for } j = 1 \ : \ \mathrm{nnz}(A)$

$\quad c_{JR(j)} = c_{JR(j)} + \underbrace{AA(j)}_{A_{JR(j),JC(j)}} * \ b_{JC(j)};$

$\texttt{end}$

Storage in Coordinate Form (cont.)

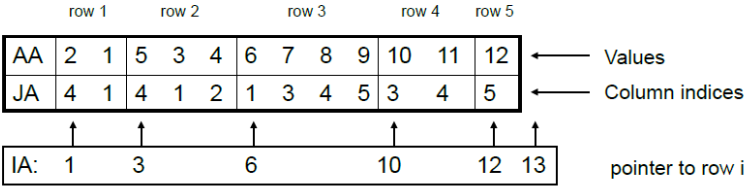
values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Pseudocode for computing $c = A \cdot b$:

$c = 0$;
for $j = 1 : \text{nnz}(A)$
$$c_{JR(j)} = c_{JR(j)} + \underbrace{AA(j)}_{A_{JR(j),JC(j)}} * b_{JC(j)};$$
end

- Disadvantage: Indirect addressing (indexing) in vector c and $b \rightarrow$ jumps in memory
- Advantage: No difference between columns and rows (A and A^T), simple.

4.3.3. Compressed Sparse Row Format: CSR



Storage:

- n and nnz
- $n +$ nnz $+1$ integer
- nnz float

Compressed Sparse Row: CSR (cont.)

	row 1		row 2			row 3				row 4		row 5	
AA	2	1	5	3	4	6	7	8	9	10	11	12	← Values
JA	4	1	4	1	2	1	3	4	5	3	4	5	← Column indices

IA:	1	3	6	10	12	13	pointer to row i

Compressed Sparse Row: CSR (cont.)

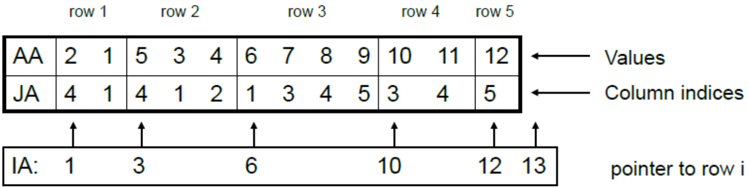
	row 1		row 2			row 3				row 4		row 5	
AA	2	1	5	3	4	6	7	8	9	10	11	12	← Values
JA	4	1	4	1	2	1	3	4	5	3	4	5	← Column indices

IA:	1	3	6	10	12	13	pointer to row i

Pseudocode for computing $c = A \cdot b$:

$c = 0$;
for $i = 1 \;:\; n$
  for $j = IA(i) \;:\; IA(i+1) - 1$
    $c_i = c_i + AA(j) \;*\; b_{JA(j)}$;
  end
end

Compressed Sparse Row: CSR (cont.)



Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
  for j = IA(i) : IA(i+1) - 1
    c_i = c_i + AA(j) * b_JA(j);
  end
end
```

$$c = 0;\quad \text{for } i = 1 : n \quad \text{for } j = IA(i) : IA(i+1)-1 \quad c_i = c_i + AA(j) * b_{JA(j)};\quad \text{end} \quad \text{end}$$

- Indirect addressing only in b.
- Columnwise $\rightarrow$ compressed sparse column format.

4.3.4. CSR with Extracted Main Diagonal

main diagonal entries | nondiagonal entries in CSR

AA	1	4	7	11	12	*	2	3	5	6	8	9	10
JA	7	8	10	13	14	14	4	1	4	1	4	5	3

Pointer to begin of *i*th row

Storage:

- n and nnz
- nnz $+1$ integer
- nnz $+1$ float

CSR with Extracted Main Diagonal (cont.)

main diagonal entries | nondiagonal entries in CSR

AA	1	4	7	11	12	*	2	3	5	6	8	9	10
JA	7	8	10	13	14	14	4	1	4	1	4	5	3

CSR with Extracted Main Diagonal (cont.)

main diagonal entries | nondiagonal entries in CSR

	main diagonal entries						nondiagonal entries in CSR						
AA	1	4	7	11	12	*	2	3	5	6	8	9	10
JA	7	8	10	13	14	14	4	1	4	1	4	5	3

Pseudocode for computing $c = A \cdot b$:

$c = 0$;
for $i = 1 \; : \; n$
  $c_i = AA_i \; * \; b_i$;
  for $j = JA(i) \; : \; JA(i+1) - 1$
    $c_i = c_i + AA_j \; * \; b_{JA(j)}$;
  end
end

4.3.5. Diagonalwise Storage

$$\begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix}$$

New matrix *A*!
Different matrix to slides before!

Diagonal numbers: $-1 \;\; 0 \;\; 2$

Values in:

$$\text{DIAG} = \begin{pmatrix} * & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & * \\ 11 & 12 & * \end{pmatrix}, \quad \text{IOFF} = (-1 \;\; 0 \;\; 2)$$

Storage: n, $nd :=$ #diagonals, nd integers for IOFF and $n \cdot nd$ float

4.3.6. Rectangular Storage Scheme by Pressing from the Right

$$\left(\begin{array}{ccccc|} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{array}\right. \leftarrow \text{pressing from right}$$

gives

$$\text{COEF} = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 0 \\ 11 & 12 & 0 \end{pmatrix} \qquad \text{JCOEF} = \begin{pmatrix} 1 & 3 & * \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & * \\ 4 & 5 & * \end{pmatrix}$$

Storage: n, $n \cdot nl$ integer and float ($nl :=$ nnz of longest row)

Rectangular Storage Scheme by Pressing from the Right (cont.)

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
  for j = 1 : nl
    c_i = c_i + COEF(i, j) * b(JCOEF(i, j));
  end
end
```

This format was used in ELLPACK (package of subroutines for elliptic PDEs).