Case One: Communication-avoiding Factorizations

Kirk M. Soodhalter

School of Mathematics
Trinity College Dublin,
the University of Dublin
http://math.soodhalter.com

January 20, 2025

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Overview

- Solving dense linear systems
- LU- and QR-factorizations
- Stability of LU
- Parallel LU
- Communication-avoiding LU

Solving dense linear systems...

Linear systems of equations

- Solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ with $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b} \in \mathbb{R}^n$ and $\boldsymbol{A}$ dense
- Iterative methods for *sparse matrices* or *matrix-free* problems to follow
- We consider direct methods (LU-factorization)
    - Factorize $\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$: $\boldsymbol{L}$, $\boldsymbol{U}$ lower/upper triangular, $\boldsymbol{P}$ is a row exchange matrix
    - Solve $\boldsymbol{P}^T\boldsymbol{L}\boldsymbol{U}\boldsymbol{x} = \boldsymbol{b}$
    - Serial LU-factorization with partial pivoting is backwards stable
    - $\|\boldsymbol{P}\boldsymbol{A} - \boldsymbol{L}\boldsymbol{U}\|_\infty$ small (close to ε_{mach}) in practice

Linear least-squares (LS) problems

- Given $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in: \mathbb{R}^m$, solve $\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$ $Ax \approx b$
- LS solutions satisfy normal equations: $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$ $b \notin \mathcal{R}(A)$
- For $m \geq n$, we denote as the "~~skinny~~ narrow" QR-factorization of $\boldsymbol{A}$

different than Gram-Schmidt version.

$$\boldsymbol{A} = \boldsymbol{Q}\begin{bmatrix} \boldsymbol{R} \\ \underline{0} \end{bmatrix}$$

with $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ orthogonal and $\boldsymbol{R} \in \mathbb{R}^{n \times n}$ upper-triangular

- If rank $\boldsymbol{A}$ = rank $\boldsymbol{R} = n$, then LS solution given by

$$\boldsymbol{R}\boldsymbol{x} = \left(\boldsymbol{Q}^T\boldsymbol{b}\right)_{1:n}$$

- Serial QR-factorization (with appropriate orthogonalization routine) is backwards stable
- $\left\| \boldsymbol{A} - \boldsymbol{Q}\begin{bmatrix} \boldsymbol{R} \\ \underline{0} \end{bmatrix} \right\|_\infty$ small (close to ε_{mach}) in practice

Communication lower bounds

The same communication lower bounds shown for serial and parallel matrix-matrix multiplication have been proven to extend to other dense/direct linear algebra algorithms and more generally to many procedures involving three nested loops.

Quick LU-factorization (without pivoting) example I

- $\boldsymbol{A} = \begin{bmatrix} 3 & 1 & 3 \\ 6 & 7 & 3 \\ 9 & 12 & 3 \end{bmatrix}$
- Scale first column: Let $\boldsymbol{S}_1 = \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$;

  $$\boldsymbol{S}_1 \boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 6 & 7 & 3 \\ 9 & 12 & 3 \end{bmatrix}$$
- Eliminating first column: let $\boldsymbol{L}_1 = \begin{bmatrix} 1 & & \\ -6 & 1 & \\ -9 & & 1 \end{bmatrix}$, then

  $$\boldsymbol{L}_1 \boldsymbol{S}_1 \boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 5 & -3 \\ 0 & 9 & -6 \end{bmatrix}$$

Quick LU-factorization (without pivoting) example II

- $$\boldsymbol{L}_1\boldsymbol{S}_1\boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 5 & -3 \\ 0 & 9 & -6 \end{bmatrix}$$
- Scale second column: Let $\boldsymbol{S}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$;
$$\boldsymbol{S}_2\boldsymbol{L}_1\boldsymbol{S}_1\boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 1 & -\frac{3}{5} \\ 0 & 9 & -6 \end{bmatrix}$$
- Eliminating second column: let $\boldsymbol{L}_2 = \begin{bmatrix} 1 & & \\ & 1 & \\ & -9 & 1 \end{bmatrix}$, then
$$\boldsymbol{L}_2\boldsymbol{S}_2\boldsymbol{L}_1\boldsymbol{S}_1\boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 1 & -\frac{3}{5} \\ 0 & 0 & -\frac{3}{5} \end{bmatrix}$$

Quick LU-factorization (without pivoting) example II

- $\boldsymbol{L}_2\boldsymbol{S}_2\boldsymbol{L}_1\boldsymbol{S}_1\boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 1 & -\frac{3}{5} \\ 0 & 0 & -\frac{3}{5} \end{bmatrix}$
- Scale third column: Let $\boldsymbol{S}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -5/3 \end{bmatrix}$;

  $\boldsymbol{S}_3\boldsymbol{L}_2\boldsymbol{S}_2\boldsymbol{L}_1\boldsymbol{S}_1\boldsymbol{A} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 1 & -\frac{3}{5} \\ 0 & 0 & 1 \end{bmatrix}$
- We set $\boldsymbol{L} = \boldsymbol{S}_3\boldsymbol{L}_2\boldsymbol{S}_2\boldsymbol{L}_1\boldsymbol{S}_1$ and $\boldsymbol{U} = \begin{bmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 1 & -\frac{3}{5} \\ 0 & 0 & 1 \end{bmatrix}$, and

  $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$

Why do we need pivoting?

- Stability: avoid division by zero or **small elements**
  - $\|\boldsymbol{A} - \boldsymbol{LU}\|$ can be large due to roundoff
- Example: $\boldsymbol{A}_1 = \begin{bmatrix} 0 & 3 & 3 \\ 3 & 1 & 3 \\ 6 & 2 & 3 \end{bmatrix}$ has no LU-factorization
- Example: $\boldsymbol{A}_2 = \begin{bmatrix} 10^{-20} & 3 & 3 \\ 3 & 1 & 3 \\ 6 & 2 & 3 \end{bmatrix}$ has an unstable

  LU-factorization, with $\boldsymbol{L}_1 = \begin{bmatrix} 1 & & \\ -3 \times 10^{20} & 1 & \\ -6 \times 10^{20} & & 1 \end{bmatrix}$
- Partial pivoting to move col-max to diagonal:

$$\widehat{\boldsymbol{P}}\boldsymbol{A}_1 = \begin{bmatrix} 6 & 2 & 3 \\ 3 & 1 & 3 \\ 0 & 3 & 3 \end{bmatrix} \qquad \widehat{\boldsymbol{P}}\boldsymbol{A}_2 = \begin{bmatrix} 6 & 2 & 3 \\ 3 & 1 & 3 \\ 10^{-20} & 3 & 3 \end{bmatrix}$$

Worst-case instability of LU with partial pivoting

Theorem

Let $\boldsymbol{PA} = \boldsymbol{LU}$ be the LU factorization with partial pivoting in exact arithmetic. Then the computed versions of these factors $\widetilde{\boldsymbol{P}}$, $\widetilde{\boldsymbol{L}}$, and $\widetilde{\boldsymbol{U}}$ satisfy

$$\widetilde{\boldsymbol{L}}\widetilde{\boldsymbol{U}} = \widetilde{\boldsymbol{P}}\boldsymbol{A} + \delta\boldsymbol{A}$$

where $\frac{\|\delta\boldsymbol{A}\|}{\boldsymbol{A}} = \mathcal{O}(\rho \cdot \varepsilon_{mach})$, *and* $\rho = \frac{\max_{i,j}|u_{ij}|}{\max_{i,j}|a_{ij}|}$.

ρ **is called the growth factor.** $\rho \approx 1 \implies$ **stable factorization**

Example of the worst case

Example (Worst-case instability)

In spite of partial pivoting yielding backwards stability, ρ can be huge. Let $\boldsymbol{A} = \begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}$ then

$$\boldsymbol{U} = \begin{bmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4\cancel{3} \\ & & & 1 & 8 \\ & & & & 16 \end{bmatrix}$$

- Scaling this example up to $2^m \times 2^m$, we see that the largest entry of $\boldsymbol{U}$ will be 2^{m-1}!

Worst-case rare for serial LU with pivoting

- Previous example is pathological. "Never" happens in practice for serial case
- Can be justified with careful statistical arguments[1]
  - Each step of Gaussian elimination introduces a rank-one correction to remaining submatrix
  - On average, this implies statistical relationships between entries of remaining submatrix tending to retard growth
- **Important:** this average stability relies on operations introducing *rank-one* corrections.
  - When moving to HPC/Parallel setting where we favor BLAS-3 and block operations (i.e., higher-rank corrections), average stability can no longer be assumed.

[1] Trefethen and Schreiber. *Average case stability of Gaussian elimination.* 1990

Block LU-factorization - obtained by delaying updates

panel

- Matrix $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \in \mathbb{R}^{n\times n}$ with $\boldsymbol{A}_{11} \in \mathbb{R}^{b\times b}$
- First step computes LU with partial pivoting of the first block:

$$\underset{n\times n}{\boldsymbol{P}_1} \underset{n\times b}{\begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix}} = \underset{n\times b}{\begin{bmatrix} \boldsymbol{L}_{11} \\ \boldsymbol{L}_{21} \end{bmatrix}} \underset{b\times b}{\boldsymbol{U}_{11}}$$

- We obtain the factorization

$$\boldsymbol{P}_1\boldsymbol{A} = \begin{bmatrix} \boldsymbol{L}_{11} & \\ \boldsymbol{L}_{21} & \boldsymbol{I}_{n-b} \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ & \boldsymbol{A}_{22}^{(1)} \end{bmatrix}$$

  with $\boldsymbol{U}_{12} = \boldsymbol{L}_{11}^{-1}\boldsymbol{A}_{12}$ and $\boldsymbol{A}_{22}^{(1)} = \boldsymbol{A}_{22} - \boldsymbol{L}_{21}\boldsymbol{U}_{12}$
- Algorithm is applied recursively on $\boldsymbol{A}_{22}^{(1)}$

Block LU-factorization - pseudocode

Compute LU of first **panel**: $\boldsymbol{P}_1 \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix} = \begin{bmatrix} \boldsymbol{L}_{11} \\ \boldsymbol{L}_{21} \end{bmatrix} \boldsymbol{U}_{11}$

Apply pivoting matrix $\boldsymbol{P}_1$ to full matrix $\overline{\boldsymbol{A}} = \boldsymbol{P}_1 \boldsymbol{A}$

Solve triangular system $\boldsymbol{U}_{12} = \boldsymbol{L}_{11}^{-1} \overline{\boldsymbol{A}}_{12}$

Update the **trailing matrix**: $\overline{\boldsymbol{A}}_{22}^{(1)} = \overline{\boldsymbol{A}}_{22} - \boldsymbol{L}_{21} \boldsymbol{U}_{12}$

Apply recursively on trailing matrix $\overline{\boldsymbol{A}}_{22}^{(1)}$

Parallel LU factorization overview

for $i = 0, b, 2b, \ldots n$ **do**

$\boldsymbol{A}^{(ib)} = \boldsymbol{A}(i:n, i:n)$

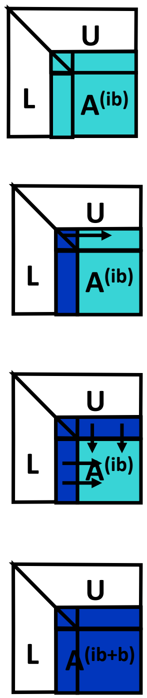
Factorize ith column panel – find a pivot for each column, do row swap

Broadcast pivot information along all rows, do swaps

Broadcast $\boldsymbol{L}_{11}$ along row to compute $\boldsymbol{U}_{12}$

Broadcast $\boldsymbol{L}_{21}$ *along rows* and $\boldsymbol{U}_{12}$ *down columns* to update trailing matrix

end



[2] Bottleneck

- Getting pivot information from entire column across all processors

[2] Source: slides from Laura Grigori: `https://people.eecs.berkeley.edu/~demmel/cs267_Spr15/Lectures/lecture13_densela2_CommAvoid_UCB_Grigori_v3.pdf`

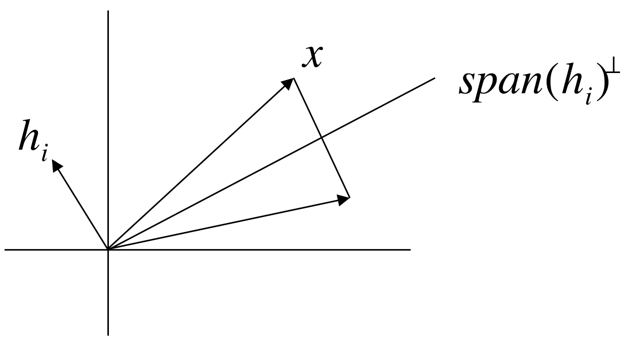
QR-factorization via Householder reflections

For $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $m \geq n$, w compute the "skinny" QR factorization

$$\begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix} \xrightarrow{\text{reflect}} \begin{pmatrix} * \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\boldsymbol{A} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R} \\ \underline{0} \end{bmatrix}$$

- Orthogonal triangularization (Householder) vs. triangular orthogonalization (Gram-Schmidt)
- Householder matrix: $\boldsymbol{H}_i = \boldsymbol{I} - \tau_i \boldsymbol{h}_i \boldsymbol{h}_i^T$
- Reflect a vector across a hyperplane
- Choose $\boldsymbol{h}_i$ (the hyperplane) so that $\boldsymbol{x} \xrightarrow[\boldsymbol{H}_i]{} \|\boldsymbol{x}\| \boldsymbol{e}_1$
- $\boldsymbol{h}_i = \pm \left(\|x\| \boldsymbol{e}_1 - \boldsymbol{x} \right)$; *sign* chosen for stability



QR-factorization via Householder reflections: an example

Apply Householder transformations to annihilate subdiagonal entries

$$\boldsymbol{A} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} = \boldsymbol{H}_1 \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}$$

$$= \boldsymbol{H}_1 \begin{bmatrix} 1 & \\ & \widetilde{\boldsymbol{H}}_2 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ & r_{22} & r_{23} & r_{24} \\ & & \boldsymbol{l}* & \boldsymbol{l}* \\ & & \boldsymbol{l}* & \boldsymbol{l}* \end{bmatrix} = \boldsymbol{H}_1 \begin{bmatrix} 1 & \\ & \widetilde{\boldsymbol{H}}_2 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & \widetilde{\boldsymbol{H}}_3 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ & r_{22} & r_{23} & r_{24} \\ & & r_{33} & r_{34} \\ & & & r_{44} \end{bmatrix}$$

$$\underbrace{\boldsymbol{H}_1 \boldsymbol{H}_2 \boldsymbol{H}_3}_{\boldsymbol{Q}} \boldsymbol{R}$$

For general $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, the factorization is

$$\boldsymbol{R} = \boldsymbol{H}_n \boldsymbol{H}_{n-1} \boldsymbol{H}_{n-2} \cdots \boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{A}$$

$$\iff (\boldsymbol{H}_n \boldsymbol{H}_{n-1} \boldsymbol{H}_{n-2} \cdots \boldsymbol{H}_2 \boldsymbol{H}_1)^T \boldsymbol{R} = \boldsymbol{A},$$

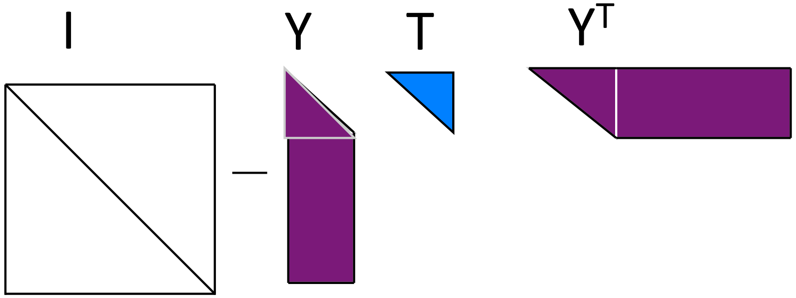
and $\boldsymbol{Q} = \boldsymbol{H}_1^T \boldsymbol{H}_2^T \cdots \boldsymbol{H}_{n-1}^T \boldsymbol{H}_n^T$

Householder reflections: a compact form

Represent Q implicitly

$$\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_b = \left(\boldsymbol{I} - \tau_1 \boldsymbol{h}_1 \boldsymbol{h}_1^T\right) \cdots \left(\boldsymbol{I} - \tau_b \boldsymbol{h}_b \boldsymbol{h}_b^T\right) = \boldsymbol{I} - \boldsymbol{Y}_b \boldsymbol{T}_b \boldsymbol{Y}_b^T$$

where $\boldsymbol{Y}_b = \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_b \end{bmatrix}$ is lower triangular and $\boldsymbol{T}_b$ is upper triangular:



Example (for $b = 2$)

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 \end{bmatrix}, \; \boldsymbol{T} = \begin{bmatrix} \tau_1 & -\tau_2 \boldsymbol{h}_1^T \boldsymbol{h}_2 \tau_2 \\ & \tau_2 \end{bmatrix}$$

Block QR-factorization

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \text{where} \boldsymbol{A}_{11} \in \mathbb{R}^{b \times b}$$

Block QR Algebra

- Compute the first panel QR-factorization
  $\boldsymbol{Q}_1^T \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{11} \\ \underline{0} \end{bmatrix}$
- Update $\boldsymbol{Q}_1^T \boldsymbol{A} = \begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ & \widetilde{\boldsymbol{A}}_{22} \end{bmatrix}$ and apply algorithm recursively on trailing matrix $\widetilde{\boldsymbol{A}}_{22}$

Block QR-factorization

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} = \boldsymbol{Q}_1 \begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ & \widetilde{\boldsymbol{A}}_{12} \end{bmatrix}$$

Compute panel factorization: $\boldsymbol{Q}_1^T \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{11} \\ \underline{0} \end{bmatrix}$

Compute compact representation: $\boldsymbol{Q}_1 = \boldsymbol{I} - \boldsymbol{Y}_1 \boldsymbol{T}_1 \boldsymbol{Y}_1^T$

Update trailing matrix:

$$\boldsymbol{Q}_1^T \begin{bmatrix} \boldsymbol{A}_{12} \\ \boldsymbol{A}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{12} \\ \boldsymbol{A}_{22} \end{bmatrix} - \boldsymbol{Y}_1 \left(\boldsymbol{T}_1^T \left(\boldsymbol{Y}_1^T \begin{bmatrix} \boldsymbol{A}_{12} \\ \boldsymbol{A}_{22} \end{bmatrix} \right) \right) = \begin{bmatrix} \boldsymbol{R}_{12} \\ \widetilde{\boldsymbol{A}}_{22} \end{bmatrix}$$

Continue recursively on trailing matrix

Bottleneck

- We must use the entire column to build Householder vector

Tall Skinny QR (TSQR) in Parallel

- We compute the QR-factorization of $\boldsymbol{W} \in \mathbb{R}^{m \times b}$ with $b \ll m$
- p processors: each processor owns a block of rows of $\boldsymbol{W}$
- Classic Parallel TSQR (ScaLAPACK)
  - Compute Householder vector for each column (Bottleneck!!)
  - Compute norm of each column: $\mathcal{O}(b \log p)$ messages
- Communication Avoiding Algorithm
  - Perform local QR-factorizations, then reduce; repeat
  - $\mathcal{O}(\log p)$ messages

Tall Skinny QR (TSQR) in Parallel

- We compute the QR-factorization of $\boldsymbol{W} \in \mathbb{R}^{m\times b}$ with $b \ll m$
- p processors: each processor owns a block of rows of $\boldsymbol{W}$
- Classic Parallel TSQR (ScaLAPACK)
  - Compute Householder vector for each column (Bottleneck!!)
  - Compute norm of each column: $\mathcal{O}(b\log p)$ messages
- Communication Avoiding Algorithm
  - Perform local QR-factorizations, then reduce; repeat
  - $\mathcal{O}(\log p)$ messages

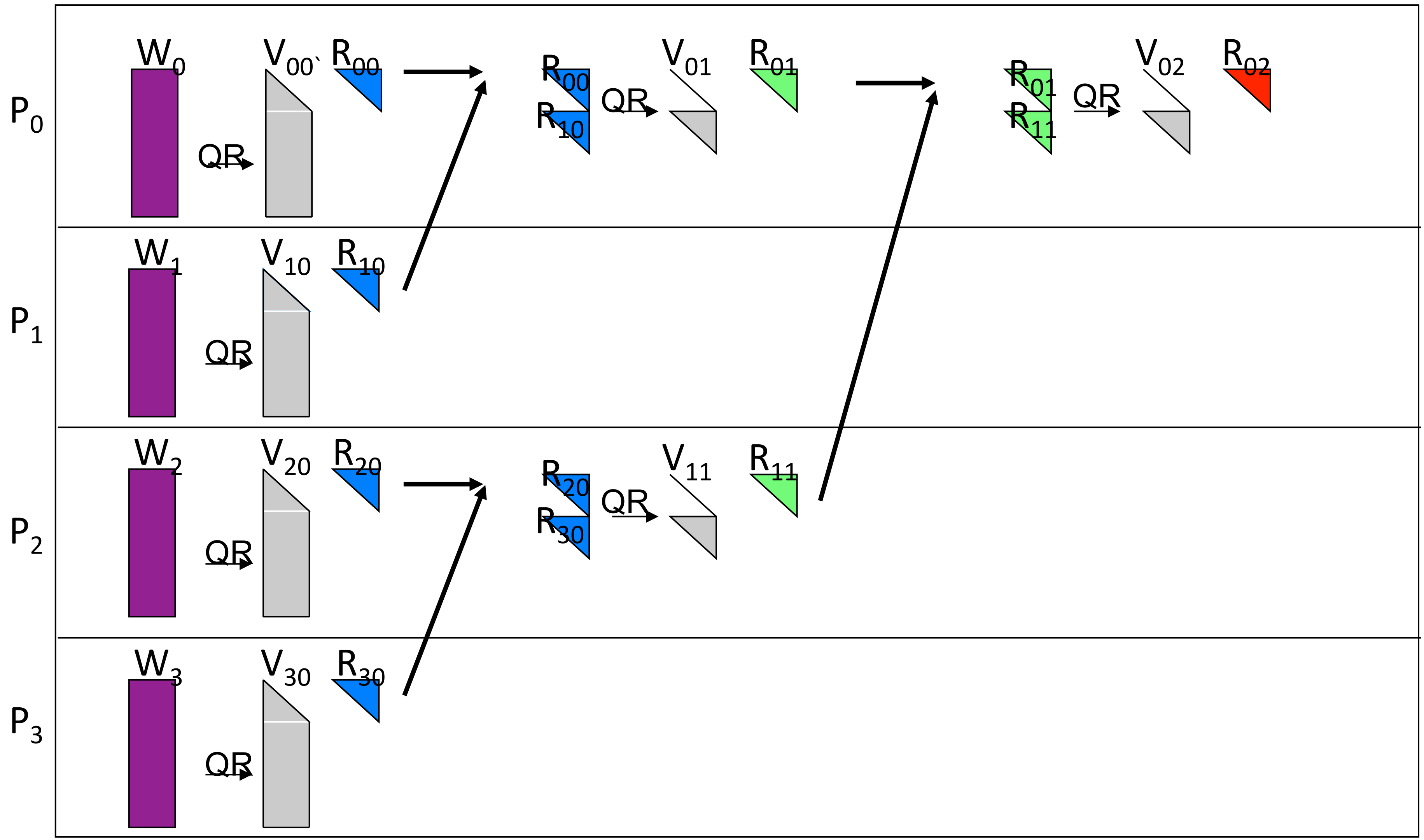
Tall Skinny QR (TSQR) in Parallel

- We compute the QR-factorization of $\boldsymbol{W} \in \mathbb{R}^{m \times b}$ with $b \ll m$
- p processors: each processor owns a block of rows of $\boldsymbol{W}$
- Classic Parallel TSQR (ScaLAPACK)
  - Compute Householder vector for each column (Bottleneck!!)
  - Compute norm of each column: $\mathcal{O}(b \log p)$ messages
- Communication Avoiding Algorithm
  - Perform local QR-factorizations, then reduce; repeat
  - $\mathcal{O}(\log p)$ messages

Tall Skinny QR (TSQR) in Parallel

- We compute the QR-factorization of $\boldsymbol{W} \in \mathbb{R}^{m \times b}$ with $b \ll m$
- p processors: each processor owns a block of rows of $\boldsymbol{W}$
- Classic Parallel TSQR (ScaLAPACK)
  - Compute Householder vector for each column (Bottleneck!!)
  - Compute norm of each column: $\mathcal{O}(b \log p)$ messages
- Communication Avoiding Algorithm
  - Perform local QR-factorizations, then reduce; repeat
  - $\mathcal{O}(\log p)$ messages

Communication Avoiding TSQR Example



3

Source: slides from Laura Grigori: https://people.eecs.berkeley.edu/~demmel/cs267_Spr15/Lectures/lecture13_densela2_CommAvoid_UCB_Grigori_v3.pdf

Communication Avoiding TSQR Example II

- Stage 1:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_0 \\ \hline \boldsymbol{W}_1 \\ \hline \boldsymbol{W}_2 \\ \hline \boldsymbol{W}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{00}\boldsymbol{R}_{00} \\ \hline \boldsymbol{Q}_{10}\boldsymbol{R}_{10} \\ \hline \boldsymbol{Q}_{20}\boldsymbol{R}_{20} \\ \hline \boldsymbol{Q}_{30}\boldsymbol{R}_{30} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{00} & & & \\ \hline & \boldsymbol{Q}_{10} & & \\ \hline & & \boldsymbol{Q}_{20} & \\ \hline & & & \boldsymbol{Q}_{30} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{R}_{00} \\ \hline \boldsymbol{R}_{10} \\ \hline \boldsymbol{R}_{20} \\ \hline \boldsymbol{R}_{30} \end{bmatrix}$$

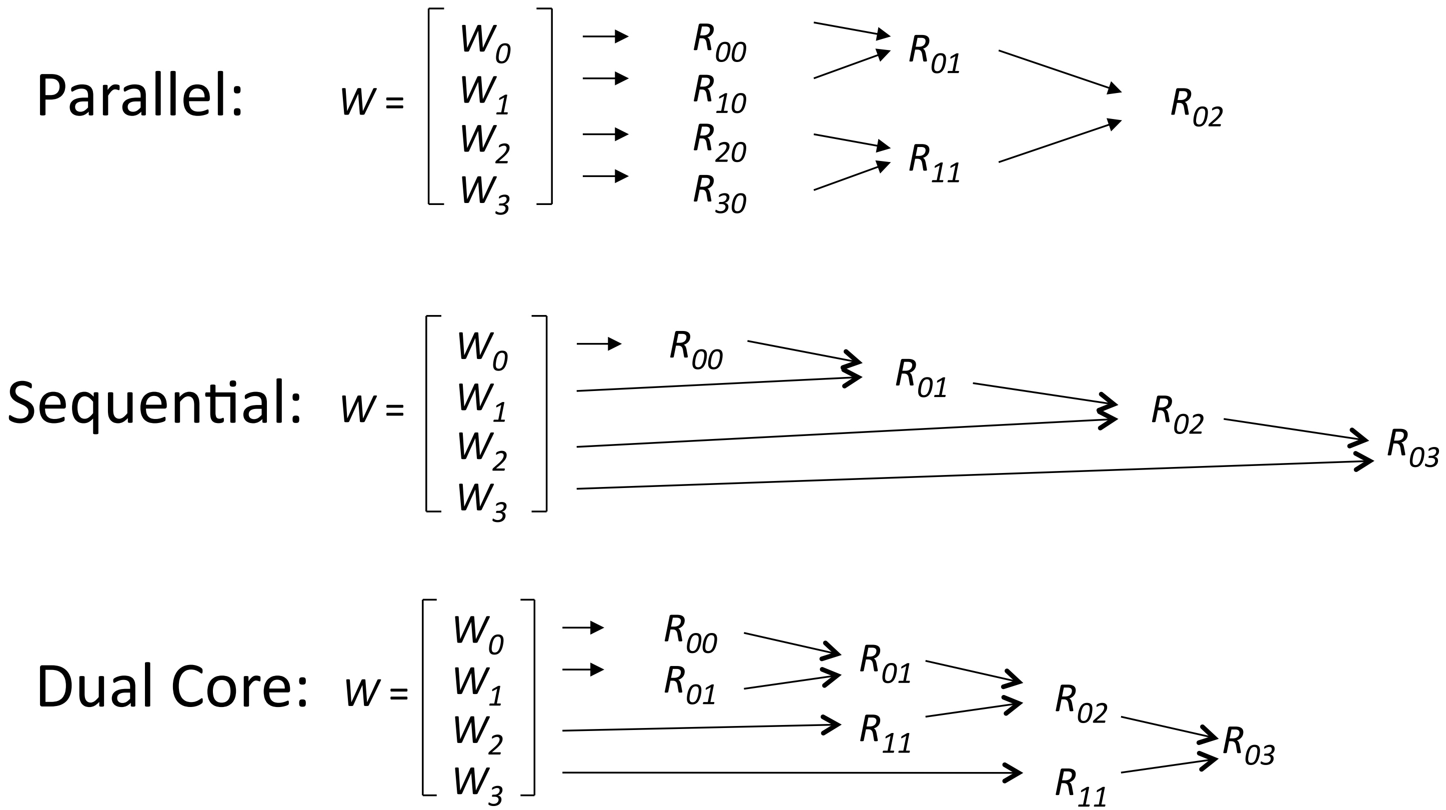
$$b = \begin{pmatrix} b_0 \\ \hline b_1 \\ \hline b_2 \\ \hline b_3 \end{pmatrix}$$

- Stages 2 & 3:

$$\underbrace{\begin{bmatrix} \boldsymbol{R}_{00} \\ \boldsymbol{R}_{10} \\ \hline \boldsymbol{R}_{20} \\ \boldsymbol{R}_{30} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{01}\boldsymbol{R}_{01} \\ \hline \boldsymbol{Q}_{11}\boldsymbol{R}_{11} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{01} & \\ \hline & \boldsymbol{Q}_{11} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{R}_{01} \\ \hline \boldsymbol{R}_{11} \end{bmatrix}}_{\text{Stage2}} ; \underbrace{\begin{bmatrix} \boldsymbol{R}_{01} \\ \boldsymbol{R}_{11} \end{bmatrix} = \boldsymbol{Q}_{02}\boldsymbol{R}_{02}}_{\text{Stage3}}$$

- $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_{00} & & & \\ & \boldsymbol{Q}_{10} & & \\ & & \boldsymbol{Q}_{20} & \\ & & & \boldsymbol{Q}_{30} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{Q}_{01} & \\ & \boldsymbol{Q}_{11} \end{bmatrix} \cdot \boldsymbol{Q}_{02}$ and $\boldsymbol{R} = \boldsymbol{R}_{02}$
- **Important:** The QR-factorization is unique up to the sign of the columns of $\boldsymbol{Q}$.

Flexibility of TSQR and CAQR algorithms



Reduction tree will depend on the underlying architecture, could be chosen dynamically

Communication-avoiding QR for general matrices

Solving LS: $Rx = (Q^T b)_{1:n}$

What if I want to keep Q & R?

- $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, how do we best lay out on processors? What mapping?
- In some of your readings, they show how any processor layout can accomodation communication-avoiding factorization.
- Optimal block size b?
- Use communication-avoiding TSQR for panel factorization
- Broadcast $\boldsymbol{Q}_{ij}$ across appropriate rows after each panel factorization?

$Q = AR^{-1}$

Keep only R.

If you need Q

Analogous communication avoiding LU-fact. (CALU)?

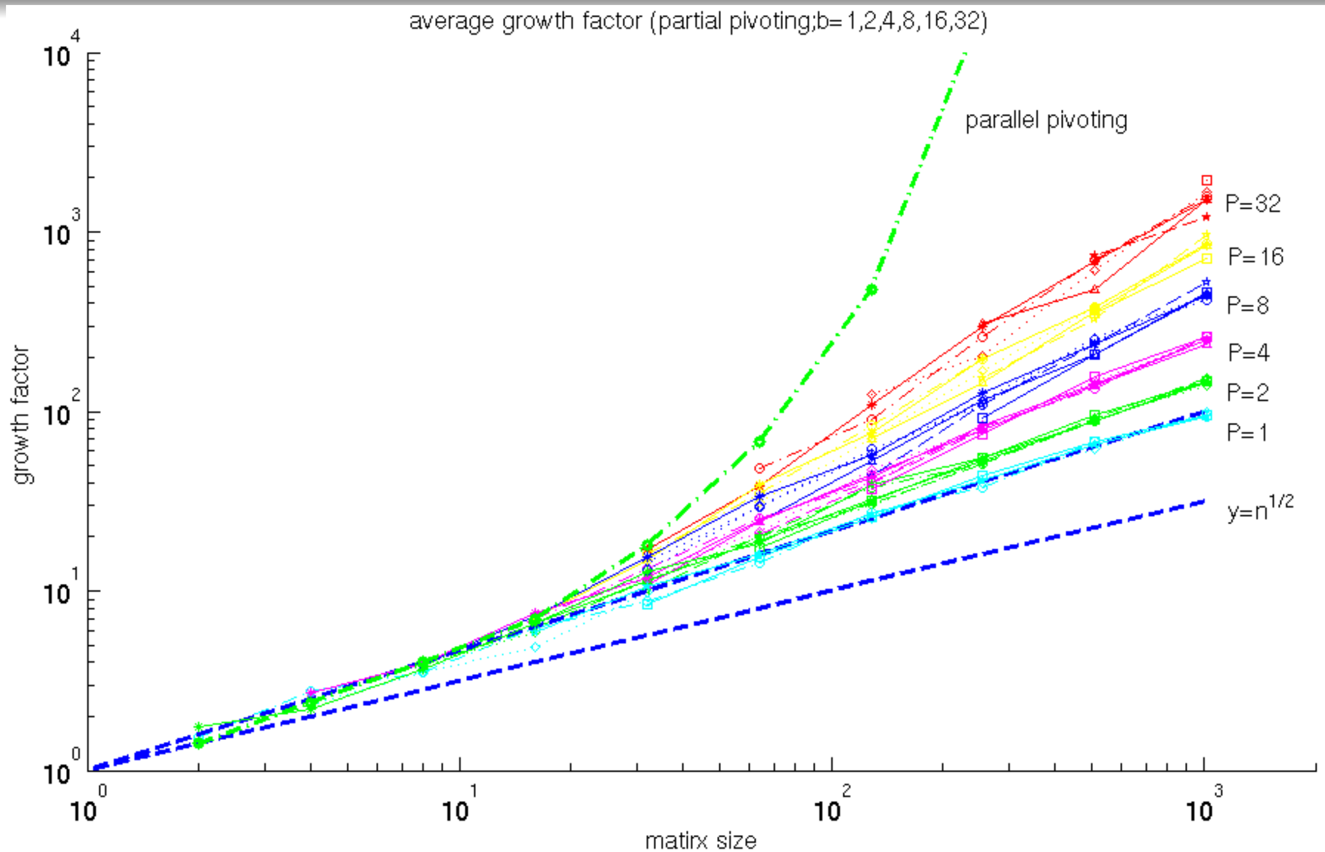
Tall skinny matrix $\boldsymbol{W}$:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_0 \\ \boldsymbol{W}_1 \\ \boldsymbol{W}_2 \\ \boldsymbol{W}_3 \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{\Pi}_{00} & & & \\ & \boldsymbol{\Pi}_{10} & & \\ & & \boldsymbol{\Pi}_{20} & \\ & & & \boldsymbol{\Pi}_{30} \end{bmatrix}}_{\boldsymbol{\Pi}_0} \cdot \begin{bmatrix} \boldsymbol{L}_{00} & & & \\ & \boldsymbol{L}_{10} & & \\ & & \boldsymbol{L}_{20} & \\ & & & \boldsymbol{L}_{30} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{U}_0 \\ \boldsymbol{U}_1 \\ \boldsymbol{U}_2 \\ \boldsymbol{U}_3 \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{U}_0 \\ \boldsymbol{U}_1 \\ \boldsymbol{U}_2 \\ \boldsymbol{U}_3 \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{\Pi}_{01} & \\ & \boldsymbol{\Pi}_{11} \end{bmatrix}}_{\boldsymbol{\Pi}_1} \cdot \begin{bmatrix} \boldsymbol{L}_{01} & \\ & \boldsymbol{L}_{11} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{U}_{01} \\ \boldsymbol{U}_{11} \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{U}_{01} \\ \boldsymbol{U}_{11} \end{bmatrix} = \boldsymbol{\Pi}_2 \boldsymbol{L}_{02} \boldsymbol{U}_{02}$$

Block parallel pivoting



[4]

- Recall: LU stability controlled by a growth factor ρ of the entries of $\boldsymbol{U}$
- Unstable for large numbers of processors
- p = # of rows corresponds to "parallel pivoting" which is known to be unstable [Trefethen and Schreiber, '90]

[4] Source: slides from Laura Grigori

Solution: Tournament Pivoting I

At each iteration, we have

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \text{with} \boldsymbol{W} = \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix} \in \mathbb{R}^{n \times b}$$

- Preprocess $\boldsymbol{W}$ to find good pivots, getting $\boldsymbol{P}$
- Perform all permutations, i.e., compute $\boldsymbol{PA}$
- Perform LU without pivots for updated $\boldsymbol{W}$, update trailing matrix

$$\boldsymbol{PA} = \begin{bmatrix} \boldsymbol{L}_{11} & \\ \boldsymbol{L}_{21} & \boldsymbol{I}_{n-b} \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ & \boldsymbol{A}_{22} - \boldsymbol{L}_{21}\boldsymbol{U}_{12} \end{bmatrix}$$

Solution: Tournament Pivoting II

- We do a reduction with LU at each step, **but only to get pivots**, and **we only pick the top b pivot rows** for each block to broadcast.
- Compute LU of each block $\boldsymbol{W}_i$ to get $\boldsymbol{\Pi}_0 = \operatorname{diag} \operatorname{curl} \boldsymbol{\Pi}_{00}, \boldsymbol{\Pi}_{10}, \boldsymbol{\Pi}_{20}, \boldsymbol{\Pi}_{30}$

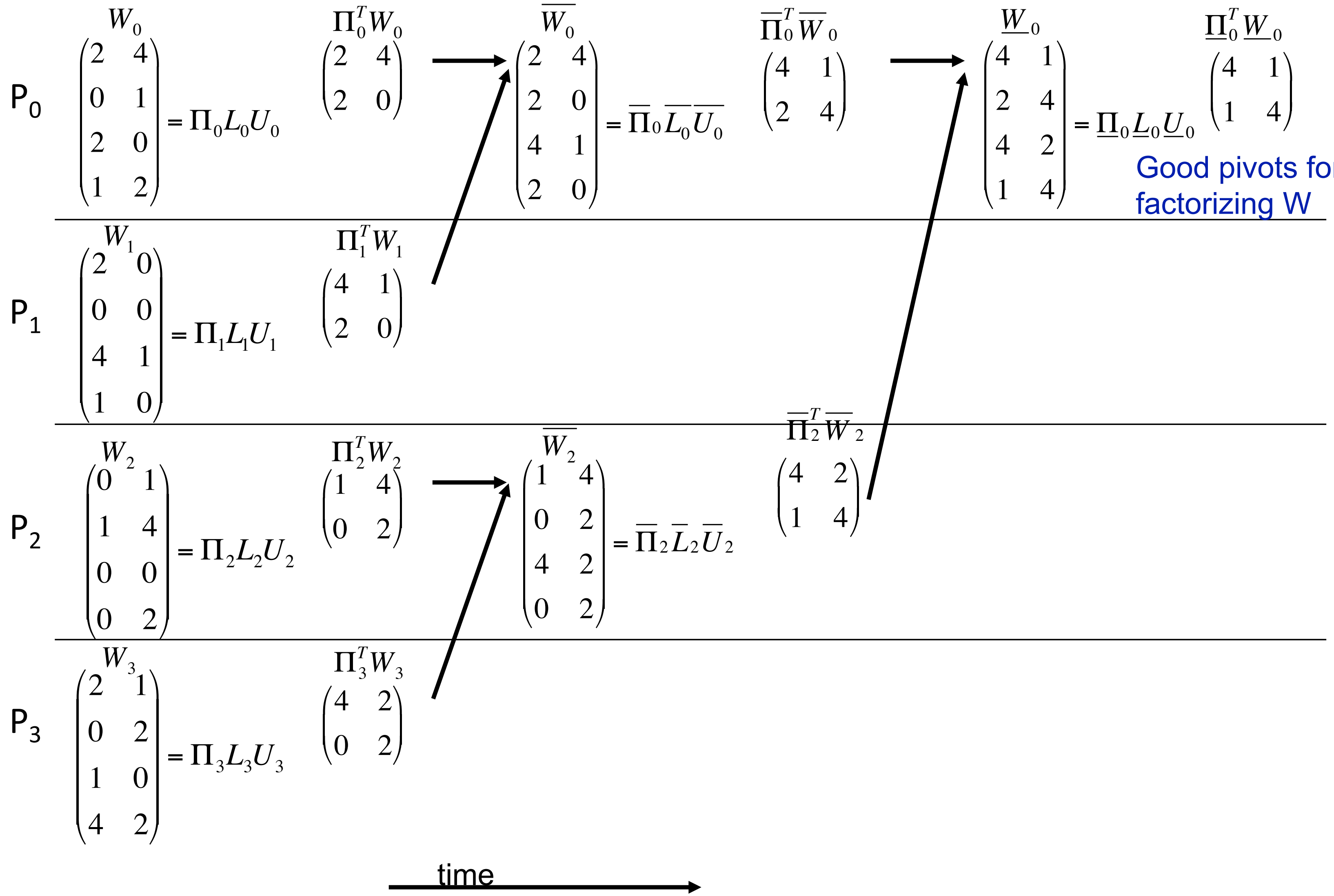
$$\boldsymbol{W} = \left[\begin{array}{c} \boldsymbol{W}_0 \\ \hline \boldsymbol{W}_1 \\ \hline \boldsymbol{W}_2 \\ \hline \boldsymbol{W}_3 \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\Pi}_{00}\boldsymbol{L}_{00}\boldsymbol{U}_{00} \\ \hline \boldsymbol{\Pi}_{10}\boldsymbol{L}_{10}\boldsymbol{U}_{10} \\ \hline \boldsymbol{\Pi}_{20}\boldsymbol{L}_{20}\boldsymbol{U}_{20} \\ \hline \boldsymbol{\Pi}_{30}\boldsymbol{L}_{30}\boldsymbol{U}_{30} \end{array}\right]$$

- Apply pivots to get $\boldsymbol{W}_{i0} = \left(\boldsymbol{\Pi}_{i0}^T \boldsymbol{W}_i\right)_{1:b}$. Get $\boldsymbol{\Pi}_1$

$$\left[\begin{array}{c} \boldsymbol{W}_{00} \\ \boldsymbol{W}_{10} \\ \hline \boldsymbol{W}_{20} \\ \boldsymbol{W}_{30} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\Pi}_{01}\boldsymbol{L}_{01}\boldsymbol{U}_{01} \\ \hline \boldsymbol{\Pi}_{11}\boldsymbol{L}_{11}\boldsymbol{U}_{11} \end{array}\right]$$

- Apply pivots to get $\boldsymbol{W}_{i1} = \left(\boldsymbol{\Pi}_{i1}^T \boldsymbol{W}_{i0}\right)_{1:b}$. Get $\boldsymbol{\Pi}_2$ with $\begin{bmatrix} \boldsymbol{W}_{i0} \\ \boldsymbol{W}_{i1} \end{bmatrix} = \boldsymbol{\Pi}_2 \boldsymbol{L}_{02} \boldsymbol{U}^{02}$.
- Computed the *unpivoted* LU factorization $\boldsymbol{\Pi}_2^T \boldsymbol{\Pi}_1^T \boldsymbol{\Pi}_0^T \boldsymbol{W} = \boldsymbol{L}\boldsymbol{U}$.

Example of Tournament Pivoting with $b = 2$ and $n = 4$



[5] Source: slides from Laura Grigori:
https://people.eecs.berkeley.edu/~demmel/cs267_Spr15/Lectures/lecture13_densela2_CommAvoid_UCB_Grigori_v3.pdf

Stability of CALU with Tournament Pivoting

- Experimentally: CALU has been demonstrated stable (with low growth factor ρ) for large random matrices with normally distributed entries
- Theoretically: CALU equivalent **in exact arithmetic** to BLAS-3 serial block LU with p.p. applied to an auxiliary matrix.
- Example: Let

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{32} \end{bmatrix}$$

- One round of tournament pivoting produces pivot matrices $\boldsymbol{\Pi}_0$ and $\boldsymbol{\Pi}_1$. Let $\begin{bmatrix} \overline{\boldsymbol{A}}_{11} & \overline{\boldsymbol{A}}_{12} \\ \overline{\boldsymbol{A}}_{21} & \overline{\boldsymbol{A}}_{22} \\ \overline{\boldsymbol{A}}_{31} & \overline{\boldsymbol{A}}_{32} \end{bmatrix} = \boldsymbol{\Pi}_1^T \boldsymbol{\Pi}_0^T \boldsymbol{A}$
- Equivalent to LU with p.p. applied to $\boldsymbol{G} = \begin{bmatrix} \overline{\boldsymbol{A}}_{11} & & \overline{\boldsymbol{A}}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{21} & \\ & -\boldsymbol{A}_{31} & \boldsymbol{A}_{32} \end{bmatrix}$

Case study

Example

You will be asked to implement a parallel communication-avoiding QR factorization code, use this code to solve some least-squares problems, and run some timing and scaling tests.

What's it all about?

- The two most widely used matrix factorizations LU and QR, which are used for directly solving, resp., linear systems and least squares problems
- The stability of LU with partial pivoting
- Parallel and then communication avoiding QR
- The naive analog of CALU followed by the stable version using tournament pivoting