LU factorization with Panel Rank Revealing Pivoting and its Communication Avoiding version

Amal Khabou
Advisor: Laura Grigori

Université Paris Sud 11, INRIA Saclay France

SIAMPP12 February 17, 2012
Collaborators

- Jim Demmel, UC Berkeley
- Ming Gu, UC Berkeley
Plan

- Motivation: just work, less talk
- Communication Avoiding LU (CALU)
- LU with Panel Rank Revealing Pivoting (LU_PRRP)
- Communication Avoiding LU_PRRP (CALU_PRRP)
- Conclusions and future work
Let $M = \text{local/fast memory size}$, general lower bounds: [Ballard et al., 2011]

$$\text{# words moved} = \Omega \left( \frac{\# \text{flops}}{\sqrt{M}} \right)$$

$$\text{# messages} = \Omega \left( \frac{\# \text{flops}}{M^2} \right)$$
Lower bound for all direct linear algebra

Let $M =$ local/ fast memory size, general lower bounds: [Ballard et al., 2011]

$$\text{# words moved} = \Omega \left( \frac{\# \text{flops}}{\sqrt{M}} \right)$$

$$\text{# messages} = \Omega \left( \frac{\# \text{flops}}{M^2} \right)$$

Goal: reorganize linear algebra to:

- minimize # words moved
- minimize # messages
- conserve numerical stability (better: to improve)
- don’t increase the flop count (too much)
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors
For $ib = 1$ to $n - 1$ step b $A^{(ib)} = A(ib : n, ib : n)$

1. Compute panel factorization
   - find pivot in each column, swap rows.
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors
For $ib = 1$ to $n - 1$ step b $A^{(ib)} = A(ib : n, ib : n)$

1. Compute panel factorization # messages=$O(n \log_2 P_r)$
   - find pivot in each column, swap rows.
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors
For $ib = 1$ to $n - 1$ step b \[ A^{(ib)} = A(ib : n, ib : n) \]

1. Compute panel factorization \# messages$= O(n \log_2 P_r)$
   - find pivot in each column, swap rows.

2. Apply all row permutations
   - broadcast pivot information along the rows
   - swap rows at left and right.
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors

For $ib = 1$ to $n - 1$ step b  
$A^{(ib)} = A(ib : n, ib : n)$

1. Compute panel factorization  
   # messages=$O(n \log_2 P_r)$
   - find pivot in each column, swap rows.

2. Apply all row permutations  
   # messages=$O(n/b(\log_2 P_r + \log_2 P_c))$
   - broadcast pivot information along the rows
   - swap rows at left and right.
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a \( P = P_r \times P_c \) grid of processors
For \( ib = 1 \) to \( n - 1 \) step \( b \)  \( A^{(ib)} = A( ib : n, ib : n ) \)

1. Compute panel factorization  # messages=\( O(n \log_2 P_r) \)
   - find pivot in each column, swap rows.

2. Apply all row permutations  # messages=\( O(n/b( \log_2 P_r + \log_2 P_c )) \)
   - broadcast pivot information along the rows
   - swap rows at left and right.

3. Compute block row of \( U \)
   - broadcast right diagonal block of \( L \) of current panel.
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors
For $ib = 1$ to $n - 1$ step b $A^{(ib)} = A(ib : n, ib : n)$

1. Compute panel factorization # messages=$O(n \log_2 P_r)$
   - find pivot in each column, swap rows.

2. Apply all row permutations # messages=$O(n/b(\log_2 P_r + \log_2 P_c))$
   - broadcast pivot information along the rows
   - swap rows at left and right.

3. Compute block row of U # messages=$O(n/b \log_2 P_c)$
   - broadcast right diagonal block of L of current panel.
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors

For $ib = 1$ to $n - 1$ step b  

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

1. **Compute panel factorization**  
   # messages=$O(n \log_2 P_r)$  
   - find pivot in each column, swap rows.

2. **Apply all row permutations**  
   # messages=$O(n/b(\log_2 P_r + \log_2 P_c))$  
   - broadcast pivot information along the rows  
   - swap rows at left and right.

3. **Compute block row of U**  
   # messages=$O(n/b \log_2 P_c)$  
   - broadcast right diagonal block of L of current panel.

4. **Update trailing matrix**  
   - broadcast right block column of L.  
   - broadcast down block row of U.
LU factorization on a $P = P_r \times P_c$ grid of processors
For $ib = 1$ to $n - 1$ step $b$ \quad $A^{(ib)} = A(ib : n, ib : n)$

1. Compute panel factorization \quad # messages=$O(n \log_2 P_r)$
   - find pivot in each column, swap rows.

2. Apply all row permutations \quad # messages=$O(n/b(\log_2 P_r + \log_2 P_c))$
   - broadcast pivot information along the rows
   - swap rows at left and right.

3. Compute block row of $U$ \quad # messages=$O(n/b \log_2 P_c)$
   - broadcast right diagonal block of $L$ of current panel.

4. Update trailing matrix \quad # messages=$O(n/b(\log_2 P_r + \log_2 P_c))$
   - broadcast right block column of $L$.
   - broadcast down block row of $U$. 
LU factorization (ScaLAPACK) upper bounds

2D algorithm \( P = P_r \times P_c \), \( M = O \left( \frac{n^2}{P} \right) \), Lower bounds:

[Ballard et al., 2011]

\[
\begin{align*}
\# \text{ words moved} & = \Omega \left( \frac{n^2}{\sqrt{P}} \right) \\
\# \text{ messages} & = \Omega \left( \sqrt{P} \right)
\end{align*}
\]
LU factorization (ScaLAPACK) upper bounds

2D algorithm $P = P_r \times P_c$, $M = O \left( \frac{n^2}{P} \right)$, Lower bounds: [Ballard et al., 2011]

- # words moved $= \Omega \left( \frac{n^2}{\sqrt{P}} \right)$
- # messages $= \Omega \left( \sqrt{P} \right)$

LU with partial pivoting (ScaLAPACK) $P = \sqrt{P} \times \sqrt{P}$, $b = \frac{n}{\sqrt{P}}$ (upper bound):

<table>
<thead>
<tr>
<th>Factor exceeding lower bounds for #words moved</th>
<th>Factor exceeding lower bounds for #messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log P$</td>
<td>$\left( \frac{n}{\sqrt{P}} \right) \log P$</td>
</tr>
</tbody>
</table>
LU factorization (ScaLAPACK) upper bounds

2D algorithm $P = P_r \times P_c$, $M = O\left(\frac{n^2}{P}\right)$, Lower bounds:

[Ballard et al., 2011]

- # words moved $= \Omega\left(\frac{n^2}{\sqrt{P}}\right)$
- # messages $= \Omega\left(\sqrt{P}\right)$

LU with partial pivoting (ScaLAPACK) $P = \sqrt{P} \times \sqrt{P}$, $b = \frac{n}{\sqrt{P}}$ (upper bound):

<table>
<thead>
<tr>
<th>Factor exceeding lower bounds for</th>
<th>Factor exceeding lower bounds for</th>
</tr>
</thead>
<tbody>
<tr>
<td>#words moved</td>
<td>#messages</td>
</tr>
<tr>
<td>$\log P$</td>
<td>$\left(\frac{n}{\sqrt{P}}\right) \log P$</td>
</tr>
</tbody>
</table>

The lower bounds are attained by CALU: LU factorization with tournament pivoting [Grigori et al., 2011]
Consider the growth factor [Wilkinson, 1961]

\[ gw = \frac{\max_{i,j,k} |a_{i,j}^{(k)}|}{\max_{i,j} |a_{i,j}|} \]

where \( a_{i,j}^{(k)} \) denotes the entry in position \((i, j)\) obtained after \(k\) steps of elimination.
Consider the growth factor [Wilkinson, 1961]

\[
g_W = \frac{\max_{i,j,k} |a_{i,j}^{(k)}|}{\max_{i,j} |a_{i,j}|}
\]

where \(a_{i,j}^{(k)}\) denotes the entry in position \((i, j)\) obtained after \(k\) steps of elimination.

Worst case growth factor:

- Partial pivoting: \(g_W \leq 2^{n-1}\)
Consider the growth factor \[\text{[Wilkinson, 1961]}\]

\[
g_W = \frac{\max_{i,j,k} |a_{i,j}^{(k)}|}{\max_{i,j} |a_{i,j}|}
\]

where \(a_{i,j}^{(k)}\) denotes the entry in position \((i, j)\) obtained after \(k\) steps of elimination.

Worst case growth factor:

- Partial pivoting: \(g_W \leq 2^{n-1}\)

- For partial pivoting, the upper bound is attained for the Wilkinson matrix.
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

\[
A = \begin{bmatrix}
    b & n-b \\
    \underbrace{A_{11}} & \underbrace{A_{12}} \\
    A_{21} & A_{22}
\end{bmatrix}
\]

where, \[ W = \begin{bmatrix}
    A_{11} \\
    A_{21}
\end{bmatrix} \]
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ \overbrace{A_{11}} & \overbrace{A_{12}} \\ \underbrace{A_{21}} & \underbrace{A_{22}} \end{bmatrix}$$

where, $W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$

For each panel $W$
CALU: Tournament pivoting - the overall idea

Consider a block algorithm that factors an \( n \times n \) matrix \( A \) by traversing panels of \( b \) columns.

\[
A = \begin{bmatrix}
  b & n-b \\
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{bmatrix}
\]

where, \( W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \)

For each panel \( W \)

- Find at low communication cost good pivots for the LU factorization of the panel \( W \), return a permutation matrix \( \Pi \) (preprocessing step).
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ \overbrace{A_{11}} & \overbrace{A_{12}} \\ \overbrace{A_{21}} & \overbrace{A_{22}} \end{bmatrix} \quad \text{where,} \quad W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

For each panel $W$

- Find at low communication cost good pivots for the LU factorization of the panel $W$, return a permutation matrix $\Pi$ (preprocessing step).
- Apply $\Pi$ to the input matrix $A$. 
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix}
  b & n-b \\
  \begin{bmatrix} A_{11} & A_{12} \\
  A_{21} & A_{22} \end{bmatrix} & \begin{bmatrix} A_{11} \\
  A_{21} \end{bmatrix}
\end{bmatrix}$$

where, $W = \begin{bmatrix} A_{11} \\
  A_{21} \end{bmatrix}$

For each panel $W$

- Find at low communication cost good pivots for the LU factorization of the panel $W$, return a permutation matrix $\Pi$ (preprocessing step).
- Apply $\Pi$ to the input matrix $A$.
- Compute LU with no pivoting of $\Pi W$, update trailing matrix.
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{where,} \quad W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

For each panel $W$

- Find at low communication cost good pivots for the LU factorization of the panel $W$, return a permutation matrix $\Pi$ (preprocessing step).
- Apply $\Pi$ to the input matrix $A$.
- Compute LU with no pivoting of $\Pi W$, update trailing matrix.

**benefit:** $b$ times fewer messages overall for the panel factorization - faster
Stability of CALU

Worst case analysis of growth factor for a reduction tree of height $H$ : CALU vs GEPP

<table>
<thead>
<tr>
<th></th>
<th>matrix of size $m \times (b + 1)$</th>
<th>GEPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSLU(b,H)</td>
<td>$2^{b(H+1)}$</td>
<td>$2^b$</td>
</tr>
<tr>
<td>$g_W$ upper bound</td>
<td>$2^{n(H+1)-1}$</td>
<td>$2^{n-1}$</td>
</tr>
</tbody>
</table>

- The growth factor upper bound of CALU is worse than GEPP, but CALU is still stable in practice. [Grigori et al., 2011]
Motivation

- CALU does an optimal amount of communication.
- CALU is as stable as GEPP in practice. [Grigori et al., 2011]
- CALU is worse than GEPP in terms of theoretical growth factor upper bound.

- Our goal: to improve the stability of both GEPP and CALU.
To improve the stability of LU:

- Pivoting strategy based on the Strong RRQR factorization

\[ A^T \Pi = QR = Q \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \]
To improve the stability of LU:

- Pivoting strategy based on the Strong RRQR factorization

\[ A^T \Pi = QR = Q \begin{bmatrix} R_{11} & R_{12} \\ R_{22} \end{bmatrix} \]

This factorization satisfies three conditions [Gu and Eisenstat, 1996]:

- every singular value of \( R_{11} \) is large.
- every singular value of \( R_{22} \) is small.
- every element of \( R_{11}^{-1} R_{12} \) could be bounded by a given threshold \( \tau \).
Result: $W^T \Pi = QR$ with $\| R_{11}^{-1} R_{12} \|_{max} \leq \tau$

Compute $W^T \Pi = QR$ \text{ RRQR with column pivoting } ;

\textbf{while} there exist \textit{i} and \textit{j} such that $| R_{11}^{-1} R_{12} |_{ij} > \tau$ \textbf{do}

\hspace{1em} Compute the QR factorization of $R \Pi_{ij}$ (QR updates) ;

\textbf{end}

\textbf{Algorithm:} Strong RRQR of the panel $W$
Strong Rank Revealing QR

**Result:** \( W^T \Pi = QR \) with \( \| R_{11}^{-1} R_{12} \|_{max} \leq \tau \)

Compute \( W^T \Pi = QR \) RRQR with column pivoting;

**while** there exist \( i \) and \( j \) such that \( |R_{11}^{-1} R_{12}_{ij}| > \tau \) **do**

Compute the QR factorization of \( R \Pi_{ij} \) (QR updates);

**end**

**Algorithm:** Strong RRQR of the panel \( W \)

- After each swap the determinant of \( R_{11} \) increases by at least \( \tau \).
- There are only a finite number of permutations.
- Strong RRQR does \( O(mb^2) \) flops in the worst case.
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

\[
A = \begin{bmatrix}
    \underbrace{b} & \underbrace{n-b} \\
    \underbrace{A_{11}} & \underbrace{A_{12}} \\
    \underbrace{A_{21}} & \underbrace{A_{22}}
\end{bmatrix}
\]

where, $W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

\[
A = \begin{bmatrix}
\text{b} & \text{n} - \text{b} \\
\text{A}_{11} & \text{A}_{12} \\
\text{A}_{21} & \text{A}_{22}
\end{bmatrix}
\]

where, $W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$

For each panel $W$
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ \widehat{A_{11}} & \widehat{A_{12}} \\ \widehat{A_{21}} & \widehat{A_{22}} \end{bmatrix} \text{ where, } W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

For each panel $W$

- perform the Strong RRQR factorization of the transpose of the panel $W$, find a permutation matrix $\Pi$

$$W^T \Pi = Q \begin{bmatrix} R_{11} & R_{12} \end{bmatrix} \text{ such as } \|L_{21}\| = \|R_{12}^T(R_{11}^{-1})^T\| \leq \tau$$
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ \widehat{A_{11}} & \widehat{A_{12}} \\ \widehat{A_{21}} & \widehat{A_{22}} \end{bmatrix} \text{ where, } W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

For each panel $W$

- perform the Strong RRQR factorization of the transpose of the panel $W$, find a permutation matrix $\Pi$

$$W^T \Pi = Q \begin{bmatrix} R_{11} & R_{12} \end{bmatrix} \text{ such as } \|L_{21}\| = \|R_{12}^T (R_{11}^{-1})^T \| \leq \tau$$

- Apply $\Pi^T$ to the input matrix $A$. 

13
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix}
\begin{array}{c}
\sum_{i=1}^{b} A_{11} \\
\sum_{i=1}^{n-b} A_{12} \\
A_{21} \\
A_{22}
\end{array}
\end{bmatrix} \text{ where, } W = \begin{bmatrix}
A_{11} \\
A_{21}
\end{bmatrix}$$

For each panel $W$

- perform the Strong RRQR factorization of the transpose of the panel $W$, find a permutation matrix $\Pi$

$$W^T \Pi = Q \begin{bmatrix}
R_{11} & R_{12}
\end{bmatrix} \text{ such as } \|L_{21}\| = \|R_{12}(R_{11}^{-1})^T\| \leq \tau$$

- Apply $\Pi^T$ to the input matrix $A$.
- Update the trailing matrix as follows.

$$\hat{A} = \Pi^T A = \begin{bmatrix}
I_b \\
L_{21} \\
I_{m-b}
\end{bmatrix} . \begin{bmatrix}
\hat{A}_{11} & \hat{A}_{12} \\
\hat{A}_{22}
\end{bmatrix}$$

where

$$\hat{A}_{22}^s = \hat{A}_{22} - L_{21} \hat{A}_{12}$$
Perform an additional GEPP on the $b \times b$ diagonal block $\hat{A}_{11}$. 
Perform an additional GEPP on the $b \times b$ diagonal block $\hat{A}_{11}$.

Update the corresponding trailing matrix $\hat{A}_{12}$.

$$
\hat{A} = \begin{bmatrix}
I_b \\
L_{21} & I_{m-b}
\end{bmatrix} \cdot \begin{bmatrix}
L_{11} & \\
& I_{m-b}
\end{bmatrix} \cdot \begin{bmatrix}
U_{11} & U_{12} \\
& \hat{A}_{s22}
\end{bmatrix}
$$
Perform an additional GEPP on the $b \times b$ diagonal block $\hat{A}_{11}$.

Update the corresponding trailing matrix $\hat{A}_{12}$.

\[
\hat{A} = \begin{bmatrix}
I_b \\
L_{21} & I_{m-b}
\end{bmatrix} \cdot \begin{bmatrix}
L_{11} & \\
& I_{m-b}
\end{bmatrix} \cdot \begin{bmatrix}
U_{11} & U_{12} \\
& \hat{A}_{22}^s
\end{bmatrix}
\]

LU_PRRP has the same memory requirements as the standard LU decomposition.

The total cost is about $\frac{2}{3} n^3 + O(n^2 b)$ flops.

Is only $O(n^2 b)$ flops more than GEPP.
Worst case analysis of the growth factor: LU_PRRP vs GEPP.

<table>
<thead>
<tr>
<th></th>
<th>LU_PRRP</th>
<th>GEPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_W$ upper bound</td>
<td>$1 + \tau b$</td>
<td>$2^b$</td>
</tr>
<tr>
<td>matrix of size $m \times (b + 1)$</td>
<td>LU_PRRP</td>
<td>GEPP</td>
</tr>
<tr>
<td>$g_W$ upper bound</td>
<td>$(1 + \tau b)^n/b$</td>
<td>$2^{n-1}$</td>
</tr>
</tbody>
</table>

LU_PRRP is more stable than GEPP in terms of theoretical growth factor upper bound.
The upper bound for different panel sizes with a parameter $\tau = 2$

For the different panel sizes, LU_PRRP is more stable than GEPP.

<table>
<thead>
<tr>
<th>$b$</th>
<th>$g_W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$(1.425)^n$</td>
</tr>
<tr>
<td>16</td>
<td>$(1.244)^n$</td>
</tr>
<tr>
<td>32</td>
<td>$(1.139)^n$</td>
</tr>
<tr>
<td>64</td>
<td>$(1.078)^n$</td>
</tr>
<tr>
<td>128</td>
<td>$(1.044)^n$</td>
</tr>
</tbody>
</table>
LU_PRRP is more resistant to pathological matrices on which GEPP fails:
LU_PRRP is more resistant to pathological matrices on which GEPP fails:

- the Wilkinson matrix.
LU_PRRP is more resistant to pathological matrices on which GEPP fails:

- the Wilkinson matrix.

| n    | b | $g_W$ | $||U||_1$ | $||U^{-1}||_1$ | $||L||_1$ | $||L^{-1}||_1$ | $\frac{||PA-LU||_F}{||A||_F}$ |
|------|---|-------|---------|---------------|---------|-------------|------------------|
| 2048 | 128 | 1     | 1.02e+03 | 6.09e+00     | 1       | 1.95e+00    | 4.25e-20         |
|      | 64  | 1     | 1.02e+03 | 6.09e+00     | 1       | 1.95e+00    | 5.29e-20         |
|      | 32  | 1     | 1.02e+03 | 6.09e+00     | 1       | 1.95e+00    | 8.63e-20         |
|      | 16  | 1     | 1.02e+03 | 6.09e+00     | 1       | 1.95e+00    | 1.13e-19         |
|      | 8   | 1     | 1.02e+03 | 6.09e+00     | 1       | 1.95e+00    | 1.57e-19         |
LU_PRRP is more resistant to pathological matrices on which GEPP fails:

- the Wilkinson matrix.

| n   | b | $g_W$ | $||U||_1$ | $||U^{-1}||_1$ | $||L||_1$ | $||L^{-1}||_1$ | $||PA-LU||_F$ |
|-----|---|-------|-----------|----------------|-----------|----------------|---------------|
| 2048| 128 | 1     | 1.02e+03  | 6.09e+00      | 1         | 1.95e+00       | 4.25e-20      |
|     | 64  | 1     | 1.02e+03  | 6.09e+00      | 1         | 1.95e+00       | 5.29e-20      |
|     | 32  | 1     | 1.02e+03  | 6.09e+00      | 1         | 1.95e+00       | 8.63e-20      |
|     | 16  | 1     | 1.02e+03  | 6.09e+00      | 1         | 1.95e+00       | 1.13e-19      |
|     | 8   | 1     | 1.02e+03  | 6.09e+00      | 1         | 1.95e+00       | 1.57e-19      |

- the Foster matrix: a concrete physical example that arises from using the quadrature method to solve a certain Volterra integral equation [Foster, 1994].
Stability of LU_PRRP (4/4)

| n  | b  | $g_W$ | $||U||_1$ | $||U^{-1}||_1$ | $||L||_1$ | $||L^{-1}||_1$ | $||PA-LU||_F/||A||_F$ |
|----|----|-------|-----------|---------------|---------|---------------|-----------------|
| 2048 | 128 | 2.66  | 1.28e+03 | 1.87e+00     | 1.92e+03 | 1.92e+03     | 4.67e-16        |
|     | 64  | 2.66  | 1.19e+03 | 1.87e+00     | 1.98e+03 | 1.79e+03     | 2.64e-16        |
|     | 32  | 2.66  | 4.33e+01 | 1.87e+00     | 2.01e+03 | 3.30e+01     | 2.83e-16        |
|     | 16  | 2.66  | 1.35e+03 | 1.87e+00     | 2.03e+03 | 2.03e+00     | 2.38e-16        |
|     | 8   | 2.66  | 1.35e+03 | 1.87e+00     | 2.04e+03 | 2.02e+00     | 5.36e-17        |
the Wright matrix: two-point boundary value problems, the multiple shooting algorithm [WRIGHT, 1993].
the Wright matrix: two-point boundary value problems, the multiple shooting algorithm [WRIGHT, 1993].

| n  | b  | $g_W$ | $||U||_1$ | $||U^{-1}||_1$ | $||L||_1$ | $||L^{-1}||_1$ | $||PA-LU||_F/||A||_F$ |
|----|----|-------|----------|----------------|----------|----------------|------------------------|
| 2048 | 128 | 2.66  | 1.28e+03 | 1.87e+00      | 1.92e+03 | 1.92e+03      | 4.67e-16               |
|     | 64  | 2.66  | 1.19e+03 | 1.87e+00      | 1.98e+03 | 1.79e+03      | 2.64e-16               |
|     | 32  | 2.66  | 4.33e+01 | 1.87e+00      | 2.01e+03 | 3.30e+01      | 2.83e-16               |
|     | 16  | 2.66  | 1.35e+03 | 1.87e+00      | 2.03e+03 | 2.03e+00      | 2.38e-16               |
|     | 8   | 2.66  | 1.35e+03 | 1.87e+00      | 2.04e+03 | 2.02e+00      | 5.36e-17               |

| n  | b  | $g_W$ | $||U||_1$ | $||U^{-1}||_1$ | $||L||_1$ | $||L^{-1}||_1$ | $||PA-LU||_F/||A||_F$ |
|----|----|-------|----------|----------------|----------|----------------|------------------------|
| 2048 | 128 | 1     | 3.25e+00 | 8.00e+00      | 2.00e+00 | 2.00e+00      | 4.08e-17               |
|     | 64  | 1     | 3.25e+00 | 8.00e+00      | 2.00e+00 | 2.00e+00      | 4.08e-17               |
|     | 32  | 1     | 3.25e+00 | 8.00e+00      | 2.05e+00 | 2.07e+00      | 6.65e-17               |
|     | 16  | 1     | 3.25e+00 | 8.00e+00      | 2.32e+00 | 2.44e+00      | 1.04e-16               |
|     | 8   | 1     | 3.40e+00 | 8.00e+00      | 2.62e+00 | 3.65e+00      | 1.26e-16               |
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\]

where,

\[
W = \begin{bmatrix}
A_{11} \\
A_{21}
\end{bmatrix}
\]
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where, $W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$

For each panel $W$
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ \underbrace{A_{11}} & \underbrace{A_{12}} \\ \underbrace{A_{21}} & A_{22} \end{bmatrix}$$

where, $W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$

For each panel $W$

- Find at low communication cost $b$ pivot rows for the QR factorization of $W^T$, return a permutation matrix $\Pi$ (preprocessing step).
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & n-b \\ A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{where}, \quad W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

For each panel $W$

- Find at low communication cost $b$ pivot rows for the QR factorization of $W^T$, return a permutation matrix $\Pi$ (preprocessing step).
- Apply $\Pi^T$ to the input matrix $A$. 
Consider a block algorithm that factors an \( n \times n \) matrix \( A \) by traversing panels of \( b \) columns.

\[
A = \begin{bmatrix}
  b \\
  \overbrace{A_{11}}^{n-b} \\
  A_{21} \\
  A_{22}
\end{bmatrix}
\]

where, \( W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \)

For each panel \( W \)

- Find at low communication cost \( b \) pivot rows for the QR factorization of \( W^T \), return a permutation matrix \( \Pi \) (preprocessing step).
- Apply \( \Pi^T \) to the input matrix \( A \).
- Compute QR with no pivoting of \( W^T \Pi \), update trailing matrix as for the LU_PRRP algorithm.
Consider a block algorithm that factors an $n \times n$ matrix $A$ by traversing panels of $b$ columns.

$$A = \begin{bmatrix} b & \overbrace{n-b}^{n-b} \\ \underbrace{A_{11}}_{A_{11}} & \underbrace{A_{12}}_{A_{12}} \\ A_{21} & A_{22} \end{bmatrix} \text{ where, } W = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

For each panel $W$
- Find at low communication cost $b$ pivot rows for the QR factorization of $W^T$, return a permutation matrix $\Pi$ (preprocessing step).
- Apply $\Pi^T$ to the input matrix $A$.
- Compute QR with no pivoting of $W^T \Pi$, update trailing matrix as for the LU_PRRP algorithm.
- Perform GEPP on the $b \times b$ diagonal block and update the corresponding trailing matrix (to obtain the LU decomposition).
Consider the panel $W$, partitioned over $P = 4$ processors as

$$W = \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix},$$
Consider the panel $W$, partitioned over $P = 4$ processors as

$$W = \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix},$$

The panel factorization uses the following binary tree:

```graph
A_{00} -> A_{01} -> A_{02}
|      |
A_{10} | A_{11} |
|      |
A_{20} | A_{21} |
|      |
A_{30} | A_{31} |
```

Compute Strong RRQR factorization of the transpose of each block $A_i$ of the panel $W$, find a permutation $\Pi$:

$$A_T^{\Pi_0} = \begin{pmatrix} Q_{00} & R_{00} \\ Q_{10} & R_{10} \\ Q_{20} & R_{20} \\ Q_{30} & R_{30} \end{pmatrix}$$

Perform $\log P$ times Strong RRQR factorizations of $2 \times b$ blocks, find permutations $\Pi_1$ and $\Pi_2$:

$$A_T^{\Pi_{01}} = \begin{pmatrix} (A_T^{\Pi_0})(:,1:b); (A_T^{\Pi_1})(:,1:b) \end{pmatrix}^T$$

$$A_T^{\Pi_{11}} = \begin{pmatrix} (A_T^{\Pi_2})(:,1:b); (A_T^{\Pi_1})(:,1:b) \end{pmatrix}^T$$

$$A_T^{\Pi_{02}} = \begin{pmatrix} (A_T^{\Pi_0})(:,1:b); (A_T^{\Pi_2})(:,1:b) \end{pmatrix}^T$$

$$A_T^{\Pi_{02}} = \begin{pmatrix} (A_T^{\Pi_0})(:,1:b); (A_T^{\Pi_2})(:,1:b) \end{pmatrix}^T$$
Consider the panel $W$, partitioned over $P = 4$ processors as

$$W = \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix},$$

The panel factorization uses the following binary tree:

1. Compute Strong RRQR factorization of the transpose of each block $A_{i0}$ of the panel $W$, find a permutation $\Pi_0$

$$\begin{bmatrix} A_{00}^T \Pi_{00} \\ A_{10}^T \Pi_{10} \\ A_{20}^T \Pi_{20} \\ A_{30}^T \Pi_{30} \end{bmatrix} = \begin{bmatrix} Q_{00} R_{00} \\ Q_{10} R_{10} \\ Q_{20} R_{20} \\ Q_{30} R_{30} \end{bmatrix}$$
CALU_PRRP: preprocessing step

Consider the panel \( W \), partitioned over \( P = 4 \) processors as

\[
W = \begin{pmatrix}
A_{00} \\
A_{10} \\
A_{20} \\
A_{30}
\end{pmatrix},
\]

The panel factorization uses the following binary tree:

1. Compute Strong RRQR factorization of the transpose of each block \( A_{i0} \) of the panel \( W \), find a permutation \( \Pi_0 \)

\[
\begin{bmatrix}
A^T_{00} \Pi_0 \\
A^T_{10} \Pi_1 \\
A^T_{20} \Pi_2 \\
A^T_{30} \Pi_3
\end{bmatrix} = \begin{bmatrix}
Q_{00} R_{00} \\
Q_{10} R_{10} \\
Q_{20} R_{20} \\
Q_{30} R_{30}
\end{bmatrix}
\]

2. Perform log \( P \) times Strong RRQR factorizations of \( 2b \times b \) blocks, find permutations \( \Pi_1 \) and \( \Pi_2 \)

\[
A^T_{01} \Pi_1 = \begin{bmatrix}
(A^T_{00} \Pi_0)(::, 1:b); (A^T_{10} \Pi_{10})(::, 1:b)
\end{bmatrix}^T \Pi_{01} = Q_{01} R_{01}
\]

\[
A^T_{11} \Pi_1 = \begin{bmatrix}
(A^T_{20} \Pi_2)(::, 1:b); (A^T_{30} \Pi_3)(::, 1:b)
\end{bmatrix}^T \Pi_{11} = Q_{11} R_{11}
\]

\[
A^T_{02} \Pi_2 = \begin{bmatrix}
(A^T_{01} \Pi_1)(::, 1:b); (A^T_{11} \Pi_{11})(::, 1:b)
\end{bmatrix}^T \Pi_{02} = Q_{02} R_{02}
\]
Stability of CALU_PRRP

Worst case analysis of the growth factor for a reduction tree of height $H$ : CALU_PRRP vs CALU.

<table>
<thead>
<tr>
<th>$g_W$ upper bound</th>
<th>matrix of size $m \times (b + 1)$</th>
<th>$TSLU_PRRP(b,H)$</th>
<th>$TSLU(b,H)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + $\tau b)^{H+1}$</td>
<td>$2^{b(H+1)}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$g_W$ upper bound</th>
<th>matrix of size $m \times n$</th>
<th>$CALU_PRRP$</th>
<th>$CALU$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1 + \tau b)^{\frac{n}{b}(H+1)-1}$</td>
<td>$2^{n(H+1)-1}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stability of CALU_PRRP

Worst case analysis of the growth factor for a reduction tree of height $H$:

CALU_PRRP vs CALU.

<table>
<thead>
<tr>
<th></th>
<th>matrix of size $m \times (b + 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSLU_PRRP(b,H)</td>
<td>$(1 + \tau b)^{H+1}$</td>
</tr>
<tr>
<td>TSLU(b,H)</td>
<td>$2^{b(H+1)}$</td>
</tr>
</tbody>
</table>

| $g_{W}$ upper bound    | $\left(1 + \frac{\tau b}{b}\right)^{\frac{n}{b}(H+1)-1}$ |
|                        | $2^{n(H+1)-1}$                      |

- CALU_PRRP is more stable than CALU in terms of theoretical growth factor upper bound.
- For the binary reduction tree, it is more stable than GEPP when $b \geq \log(\tau b) \log P$ ($b = \frac{n}{\sqrt{P}}$)
CALU_PRRP: experimental results

- CALU_PRRP is as stable as GEPP in practice [Khabou et al., 2012].
- QR with column pivoting is sufficient to attain the bound $\tau$ in practice.
Overall summary

- **LU_PRRP**
  - more stable than GEPP in terms of growth factor upper bound.
  - more resistant to pathological matrices.
  - same memory requirements as the standard LU.

- **CALU_PRRP**
  - an optimal amount of communication at the cost of redundant computation.
  - more stable than CALU in terms of growth factor upper bound.
  - more stable than GEPP in terms of growth factor upper bound under certain conditions.
Future work

- Estimating the performance of CALU_PRRP on parallel machines based on multicore processors, and comparing it with the performance of CALU.
- Design of a communication avoiding algorithm that has smaller bounds on the growth factor than that of GEPP in general.
Minimizing communication in linear algebra.
*SIMAX*, 32:866–90.

Foster, L. V. (1994).
Gaussian Elimination with Partial Pivoting Can Fail in Practice.

CALU: A communication optimal LU factorization algorithm.
*SIAM Journal on Matrix Analysis and Applications*.

Efficient Algorithms For Computing A Strong Rank Reaviling QR Factorization.

LU factorization with panel rank revealing pivoting and its communication avoiding version.
Technical Report UCB/EECS-2012-15, EECS Department, University of California, Berkeley.

Error analysis of direct methods of matrix inversion.

A collection of problems for which gaussian elimination with partial pivoting is unstable.
*SIAM J. SCI. STATIST. COMPUT.*, 14:231–238.