LU factorization and its communication avoiding version

Laura Grigori

EPFL and PSI

October 17, 2023

Plan

LU factorization

Block LU factorization

Communication avoiding LU factorization

Norms and other notations

$$\begin{aligned}
\|A\|_F &= \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2} \\
\|A\|_2 &= \sigma_{max}(A) \\
\|A\|_\infty &= \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| \\
\|A\|_1 &= \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|
\end{aligned}$$

Inequalities $|x| \le |y|$ and $|A| \le |B|$ hold componentwise.

Plan

LU factorization

Block LU factorization

Communication avoiding LU factorization

Algebra of the LU factorization

LU factorization

Compute the factorization PA = LU

Example

Given the matrix

$$A = \begin{pmatrix} 3 & 1 & 3 \\ 6 & 7 & 3 \\ 9 & 12 & 3 \end{pmatrix}$$

Let

$$M_1 = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -3 & & 1 \end{pmatrix}, \quad M_1 A = \begin{pmatrix} 3 & 1 & 3 \\ 0 & 5 & -3 \\ 0 & 9 & -6 \end{pmatrix}$$

Algebra of the LU factorization

- In general

$$\begin{aligned} A^{(k+1)} &= M_k A^{(k)} := \begin{pmatrix} I_{k-1} & & & & \\ & 1 & & & \\ & -m_{k+1,k} & 1 & & \\ & \dots & & \ddots & \\ & -m_{n,k} & & & 1 \end{pmatrix} A^{(k)}, \text{where} \\ M_k &= I - m_k e_k^T, \quad M_k^{-1} = I + m_k e_k^T \end{aligned}$$

where e_k is the k-th unit vector, $m_k = (0, \dots, 0, 1, m_{k+1,k}, \dots, m_{n,k})^T$, $e_i^T m_k = 0, \forall i \leq k$

- The factorization can be written as

$$M_{n-1} \dots M_1 A = A^{(n)} = U$$

Algebra of the LU factorization

- We obtain

$$
\begin{aligned}
A &= M_1^{-1} \dots M_{n-1}^{-1} U \\
&= (I + m_1 e_1^T) \dots (I + m_{n-1} e_{n-1}^T) U \\
&= \left(I + \sum_{i=1}^{n-1} m_i e_i^T \right) U \\
&= \begin{pmatrix} 1 & & & \\ m_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ m_{n1} & m_{n2} & \dots & 1 \end{pmatrix} U = LU
\end{aligned}
$$

The need for pivoting

- For stability, avoid division by small diagonal elements
- For example

$$A = \begin{pmatrix} 0 & 3 & 3 \\ 3 & 1 & 3 \\ 6 & 2 & 3 \end{pmatrix} \tag{1}$$

has an LU factorization if we permute the rows of matrix A

$$PA = \begin{pmatrix} 6 & 2 & 3 \\ 0 & 3 & 3 \\ 3 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & & \\ & 1 & \\ 0.5 & & 1 \end{pmatrix} \cdot \begin{pmatrix} 6 & 2 & 3 \\ & 3 & 3 \\ & & 1.5 \end{pmatrix} \tag{2}$$

- Partial pivoting allows to bound the multipliers $m_{ik} \leq 1$ and hence $|L| \leq 1$
- Factorization referred to as LU with partial pivoting (LUPP) or also Gaussian elimination with partial pivoting GEPP

Wilkinson's backward error stability result

Growth factor g_W defined as

$$g_W = \frac{\max_{i,j,k} |a_{ij}^k|}{\max_{i,j} |a_{ij}|}$$

Note that

$$|u_{ij}| = |a_{ij}^i| \leq g_W \max_{i,j} |a_{ij}|$$

Theorem (Wilkinson's backward error stability result, see also [N.J.Higham, 2002] for more details)

Let $A \in \mathbb{R}^{n \times n}$ and let $\hat{x}$ be the computed solution of $Ax = b$ obtained by using GEPP. Then

$$(A + \Delta A)\hat{x} = b, \qquad \|\Delta A\|_\infty \leq n^2 \gamma_{3n} g_W(n) \|A\|_\infty,$$

where $\gamma_n = nu/(1 - nu)$, u is machine precision and assuming $nu < 1$.

The growth factor

- The LU factorization is backward stable if the growth factor is small (grows linearly with n).
- For partial pivoting, the growth factor $g(n) \leq 2^{n-1}$, and this bound is attainable.
- In practice it is on the order of $n^{2/3} - n^{1/2}$

Exponential growth factor for Wilkinson matrix

$$A = diag(\pm 1) \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \ldots & 0 & 1 \\ -1 & -1 & 1 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 & 1 \\ -1 & -1 & \cdots & -1 & 1 & 1 \\ -1 & -1 & \cdots & -1 & -1 & 1 \end{bmatrix}$$

Experimental results for special matrices

Several errror bounds for GEPP, the normwise backward error η and the componentwise backward error w $(r = b - Ax)$.

$$\eta = \frac{||r||_1}{||A||_1 \; ||x||_1 + ||b||_1},$$
$$w = \max_i \frac{|r_i|}{(|A| \; |x| + |b|)_i}.$$

matrix	cond(A,2)	g_W	$\|\|L\|\|_1$	cond$(U,1)$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b
hadamard	1.0E+0	4.1E+3	4.1E+3	5.3E+5	0.0E+0	3.3E-16	4.6E-15
randsvd	6.7E+7	4.7E+0	9.9E+2	1.4E+10	5.6E-15	3.4E-16	2.0E-15
chebvand	3.8E+19	2.0E+2	2.2E+3	4.8E+22	5.1E-14	3.3E-17	2.6E-16
frank	1.7E+20	1.0E+0	2.0E+0	1.9E+30	2.2E-18	4.9E-27	1.2E-23
hilb	8.0E+21	1.0E+0	3.1E+3	2.2E+22	2.2E-16	5.5E-19	2.0E-17

- Two reasons considered to be important for the average case stability [Trefethen and Schreiber, 90]:
  - the multipliers in L are small
  - the correction introduced at each elimination step is of rank 1

Plan

LU factorization

Block LU factorization

Communication avoiding LU factorization

Block formulation of the LU factorization

Partitioning of matrix A of size $n \times n$

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

where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$ and A_{22} is of size $(m-b) \times (n-b)$.

Block LU algebra

The first iteration computes the factorization:

$$P_1^T A = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & \\ L_{21} & I_{n-b} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} \\ & A^1 \end{bmatrix}$$

The algorithm continues recursively on the trailing matrix A^1.

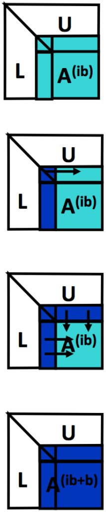
Block LU factorization - the algorithm

1. Compute the LU factorization with partial pivoting of the first block column
$$P_1 \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} U_{11}$$
2. Pivot by applying the permutation matrix P_1^T on the entire matrix,
$$\bar{A} = P_1^T A.$$
3. Solve the triangular system
$$L_{11} U_{12} = \bar{A}_{12}$$
4. Update the trailing matrix,
$$A^1 = \bar{A}_{22} - L_{21} U_{12}$$
5. Compute recursively the block LU factorization of A^1.

LU Factorization as in ScaLAPACK

LU factorization on a P = Pr x Pc grid of processors
For ib = 1 to n-1 step b
$A(ib) = A(ib:n, ib:n)$

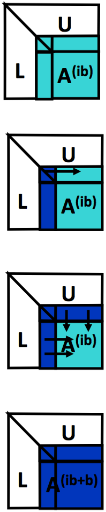
1. Compute panel factorization
   - find pivot in each column, swap rows
2. Apply all row permutations
   - broadcast pivot information along the rows
   - swap rows at left and right
3. Compute block row of U
   - broadcast right diagonal block of L of current panel
4. Update trailing matrix
   - broadcast right block column of L
   - broadcast down block row of U



Cost of LU Factorization in ScaLAPACK

LU factorization on a P = Pr x Pc grid of processors
For ib = 1 to n-1 step b
$A(ib) = A(ib:n, ib:n)$

1. Compute panel factorization
   - $\#messages = O(n \log_2 P_r)$
2. Apply all row permutations
   - $\#messages = O(n/b(\log_2 P_r + \log_2 P_c))$
3. Compute block row of U
   - $\#messages = O(n/b \log_2 P_c)$
4. Update trailing matrix
   - $\#messages = O(n/b(\log_2 P_r + \log_2 P_c)$



Cost of parallel block LU

Consider that we have a $\sqrt{P} \times \sqrt{P}$ grid, block size b

$$\gamma \cdot \left(\frac{2/3n^3}{P} + \frac{n^2 b}{\sqrt{P}} \right) + \beta \cdot \frac{n^2 \log P}{\sqrt{P}} + \alpha \cdot \left(1.5 n \log P + \frac{3.5 n}{b} \log P \right).$$

Plan

LU factorization

Block LU factorization

Communication avoiding LU factorization

The LU factorization of a tall skinny matrix

First try the obvious generalization of TSQR.

$$
\begin{aligned}
W &= \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \begin{pmatrix} \Pi_{00} L_{00} U_{00} \\ \Pi_{10} L_{10} U_{10} \\ \Pi_{20} L_{20} U_{20} \\ \Pi_{30} L_{30} U_{30} \end{pmatrix} \\
&= \begin{pmatrix} \Pi_{00} & & & \\ & \Pi_{10} & & \\ & & \Pi_{20} & \\ & & & \Pi_{30} \end{pmatrix} \begin{pmatrix} L_{00} & & & \\ & L_{10} & & \\ & & L_{20} & \\ & & & L_{30} \end{pmatrix} \left(\begin{array}{c} U_{00} \\ U_{10} \\ \hline U_{20} \\ U_{30} \end{array}\right) = \Pi_0 L_0 \left(\begin{array}{c} U_{00} \\ U_{10} \\ \hline U_{20} \\ U_{30} \end{array}\right) \\
\left(\begin{array}{c} U_{00} \\ U_{10} \\ \hline U_{20} \\ U_{30} \end{array}\right) &= \left(\begin{array}{c} \Pi_{01} L_{01} U_{01} \\ \hline \Pi_{11} L_{11} U_{11} \end{array}\right) = \left(\begin{array}{c|c} \Pi_{01} & \\ \hline & \Pi_{11} \end{array}\right) \left(\begin{array}{c|c} L_{01} & \\ \hline & L_{11} \end{array}\right) \left(\begin{array}{c} U_{01} \\ \hline U_{11} \end{array}\right) = \Pi_1 L_1 \left(\begin{array}{c} U_{01} \\ \hline U_{11} \end{array}\right) \\
\left(\begin{array}{c} U_{01} \\ \hline U_{11} \end{array}\right) &= \Pi_{02} \cdot L_{02} \cdot U_{02}
\end{aligned}
$$

The final factorization is:

$$
W = \Pi_0 L_0 \Pi_1 L_1 \Pi_{02} L_{02} U_{02}
$$

Obvious generalization of TSQR to LU

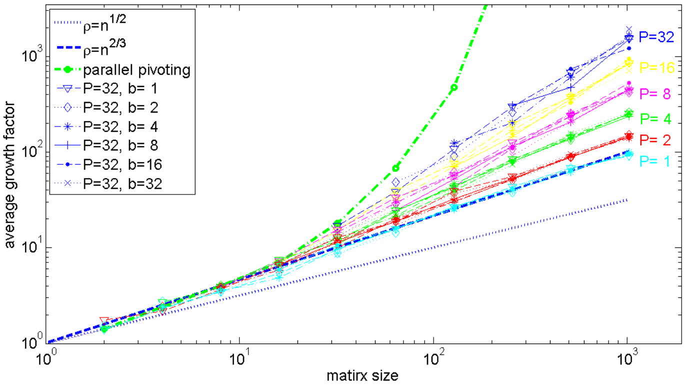
- Block parallel pivoting:

$$\begin{tikzcd}
W_0 \arrow[r] & U_{00} \arrow[dr] & & \\
& & U_{01} \arrow[ddr] & \\
W_1 \arrow[r] & U_{10} \arrow[ur] & & \\
& & & U_{02} \\
W_2 \arrow[r] & U_{20} \arrow[dr] & & \\
& & U_{11} \arrow[uur] & \\
W_3 \arrow[r] & U_{30} \arrow[ur] & &
\end{tikzcd}$$

- Block pairwise pivoting:
  - uses a flat tree and is optimal in the sequential case
  - introduced by Barron and Swinnerton-Dyer, 1960: block LU factorization used to solve a system with 100 equations on EDSAC 2 computer using an auxiliary magnetic-tape
  - used in PLASMA for multicore architectures and FLAME for out-of-core algorithms and for multicore architectures

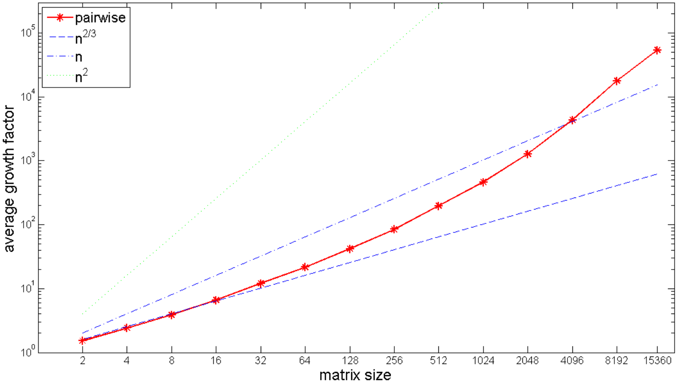
$$\begin{tikzcd}
W_0 \arrow[r] & U_{00} \arrow[r] & U_{01} \arrow[r] & U_{02} \arrow[r] & U_{03} \\
W_1 \arrow[urr] & & & & \\
W_2 \arrow[uurrr] & & & & \\
W_3 \arrow[uuurrrr] & & & &
\end{tikzcd}$$

Block parallel pivoting



- Unstable for large number of processors P
- When P=number rows, it corresponds to parallel pivoting, known to be unstable (Trefethen and Schreiber, 90)

Block pairwise pivoting



- Results shown for random matrices
- Will become unstable for large matrices

Tournament pivoting - the overall idea

- At each iteration of a block algorithm

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad \text{where} W = \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$$

  where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$ and A_{22} is of size $(m-b) \times (n-b)$.

  - Preprocess W to find at low communication cost good pivots for the LU factorization of W, return a permutation matrix P
  - Permute the pivots to top, ie compute PA
  - Compute LU with no pivoting of W, update trailing matrix, obtain

$$PA = \begin{pmatrix} L_{11} & \\ L_{21} & I_{n-b} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & A_{22} - L_{21} U_{12} \end{pmatrix}$$

- For details see [Grigori et al., 2011]

Tournament pivoting for a tall skinny matrix

1. Compute GEPP factorization of each W_i, find permutation Π_0

$$W = A(:, 1:b) \quad = \quad \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \begin{pmatrix} \Pi_{00} L_{00} U_{00} \\ \Pi_{10} L_{10} U_{10} \\ \Pi_{20} L_{20} U_{20} \\ \Pi_{30} L_{30} U_{30} \end{pmatrix}, \quad \begin{array}{l} \text{Pick b pivot rows, form } A_{00} \\ \text{Same for } A_{10} \\ \text{Same for } A_{20} \\ \text{Same for } A_{30} \end{array}$$

2. Perform $\log_2 P$ times GEPP factorization of selected $2b \times b$ rows, find permutations Π_1, Π_2

$$\begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix} = \begin{pmatrix} \Pi_{01} L_{01} U_{01} \\ \Pi_{11} L_{11} U_{11} \end{pmatrix}, \quad \begin{array}{l} \text{Pick b pivot rows, form } A_{01} \\ \text{Same for } A_{11} \end{array}$$

$$\begin{pmatrix} A_{01} \\ A_{11} \end{pmatrix} = \left(\Pi_{02} L_{02} U_{02} \right), \quad \begin{array}{l} \text{Pick b pivot rows that will} \\ \text{be used as pivots to factor } W \end{array}$$

3. Perform LU factorization with no pivoting of the permuted matrix

$$\Pi_2^T \Pi_1^T \Pi_0^T W = LU$$

Tournament pivoting

P_0: $W_0 = \begin{pmatrix} 2 & 4 \\ 0 & 1 \\ 2 & 0 \\ 1 & 2 \end{pmatrix} = \Pi_0 L_0 U_0$, $\quad \Pi_0^T W_0 = \begin{pmatrix} 2 & 4 \\ 2 & 0 \end{pmatrix}$

P_1: $W_1 = \begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 4 & 1 \\ 1 & 0 \end{pmatrix} = \Pi_1 L_1 U_1$, $\quad \Pi_1^T W_1 = \begin{pmatrix} 4 & 1 \\ 2 & 0 \end{pmatrix}$

P_2: $W_2 = \begin{pmatrix} 0 & 1 \\ 1 & 4 \\ 0 & 0 \\ 0 & 2 \end{pmatrix} = \Pi_2 L_2 U_2$, $\quad \Pi_2^T W_2 = \begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix}$

P_3: $W_3 = \begin{pmatrix} 2 & 1 \\ 0 & 2 \\ 1 & 0 \\ 4 & 2 \end{pmatrix} = \Pi_3 L_3 U_3$, $\quad \Pi_3^T W_3 = \begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix}$

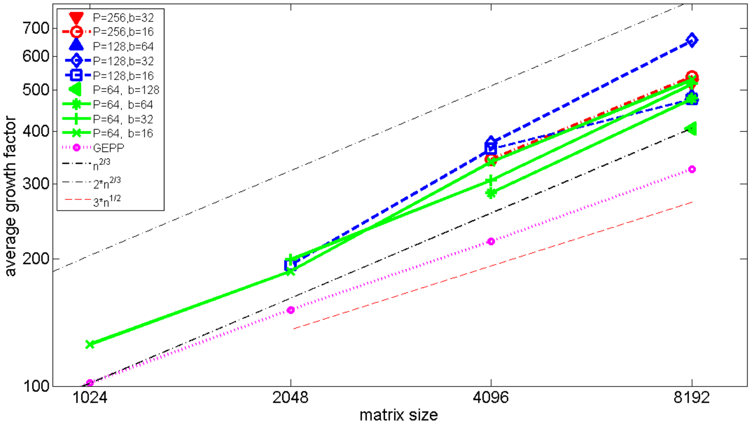
$\overline{W}_0 = \begin{pmatrix} 2 & 4 \\ 2 & 0 \\ 4 & 1 \\ 2 & 0 \end{pmatrix} = \overline{\Pi}_0 \overline{L}_0 \overline{U}_0$, $\quad \overline{\Pi}_0^T \overline{W}_0 = \begin{pmatrix} 4 & 1 \\ 2 & 4 \end{pmatrix}$

$\overline{W}_2 = \begin{pmatrix} 1 & 4 \\ 0 & 2 \\ 4 & 2 \\ 0 & 2 \end{pmatrix} = \overline{\Pi}_2 \overline{L}_2 \overline{U}_2$, $\quad \overline{\Pi}_2^T \overline{W}_2 = \begin{pmatrix} 4 & 2 \\ 1 & 4 \end{pmatrix}$

$\underline{W}_0 = \begin{pmatrix} 4 & 1 \\ 2 & 4 \\ 4 & 2 \\ 1 & 4 \end{pmatrix} = \underline{\Pi}_0 \underline{L}_0 \underline{U}_0$, $\quad \underline{\Pi}_0^T \underline{W}_0 = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$

Good pivots for factorizing W

Growth factor for binary tree based CALU



- Random matrices from a normal distribution
- Same behaviour for all matrices in our test, and $|L| \leq 4.2$

Our "proof of stability" for CALU

- CALU as stable as GEPP in following sense:
  In exact arithmetic, CALU process on a matrix A is equivalent to GEPP process on a larger matrix G whose entries are blocks of A and zeros.
- Example of one step of tournament pivoting:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix},$$

$$\begin{tikzcd} A_{11} \arrow[rr] & & \bar{A}_{11} \\ A_{21} \arrow[r] & A_{21} \arrow[ur] & \\ A_{31} \arrow[ur] & & \end{tikzcd}$$

$$G = \begin{pmatrix} \bar{A}_{11} & & \bar{A}_{12} \\ A_{21} & A_{21} & \\ & -A_{31} & A_{32} \end{pmatrix}$$

- Proof possible by using original rows of A during tournament pivoting (not the computed rows of U).

Outline of "proof of stability"

- After the factorization of first panel by CALU, A_{32}^s (the Schur complement of A_{32}) is not bounded as in GEPP,

$$\begin{pmatrix} \Pi_{11} & \Pi_{12} & \\ \Pi_{21} & \Pi_{22} & \\ & & I \end{pmatrix} \cdot \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{L}_{11} & & \\ \bar{L}_{21} & I_b & \\ \bar{L}_{31} & & I_{m-2b} \end{pmatrix} \cdot \begin{pmatrix} \bar{U}_{11} & \bar{U}_{12} \\ & \bar{A}_{22}^s \\ & A_{32}^s \end{pmatrix}$$

- but A_{32}^s can be obtained by GEPP on larger matrix G formed from blocks of A

$$G = \begin{pmatrix} \bar{A}_{11} & & \bar{A}_{12} \\ A_{21} & A_{21} & \\ & -A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{L}_{11} & & \\ A_{21}\bar{U}_{11}^{-1} & L_{21} & \\ & -L_{31} & I_{m-2b} \end{pmatrix} \begin{pmatrix} \bar{U}_{11} & & \bar{U}_{12} \\ & U_{21} & -L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} \\ & & A_{32}^s \end{pmatrix}$$

- GEPP on G does not permute and

$$\begin{aligned} L_{31}L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s &= L_{31}U_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s = A_{31}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s \\ &= \bar{L}_{31}\bar{U}_{12} + A_{32}^s = A_{32} \end{aligned}$$

Growth factor in exact arithmetic

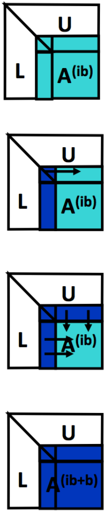
- Matrix of size m-by-n, reduction tree of height $H = \log_2(P)$
- In practice growth factor for GEPP and CALU is on the order of $n^{2/3} - -n^{1/2}$

	matrix of size $m \times (b+1)$		
	TSLU(b,H)		GEPP
	upper bound	attained	upper bound
$\|L\|$	2^{bH}	$2^{(b-2)H-(b-1)}$	1
g_W	$2^{b(H+1)}$	2^b	2^b
	matrix of size $m \times n$		
	CALU(b,H)		GEPP
	upper bound	attained	upper bound
$\|L\|$	2^{bH}	$2^{(b-2)H-(b-1)}$	1
g_W	$2^{n(H+1)-1}$	2^{n-1}	2^{n-1}

Cost of LU Factorization in ScaLAPACK

LU factorization on a P = Pr x Pc grid of processors
For ib = 1 to n-1 step b
$A(ib) = A(ib:n, ib:n)$

1. Compute panel factorization
   - $\#messages = O(n/b \log_2 P_r)$
2. Apply all row permutations
   - $\#messages = O(n/b(\log_2 P_r + \log_2 P_c))$
3. Compute block row of U
   - $\#messages = O(n/b \log_2 P_c)$
4. Update trailing matrix
   - $\#messages = O(n/b(\log_2 P_r + \log_2 P_c)$



CALU based on TSLU

Cost of CALU vs ScaLAPACK's PDGETRF

- $n \times n$ matrix on $\sqrt{P} \times \sqrt{P}$ processor grid, block size b
- Flops: $(2/3)n^3/P + 3/2n^2 b \log_2 P/\sqrt{P}$ vs $(2/3)n^3/P + n^2 b/P^{1/2}$
- Bandwidth: $n^2 \log_2 P/\sqrt{P}$ vs same
- Latency: $3n \log_2 P/b$ vs $1.5n \log_2 P$

Close to optimal (modulo log P factors)

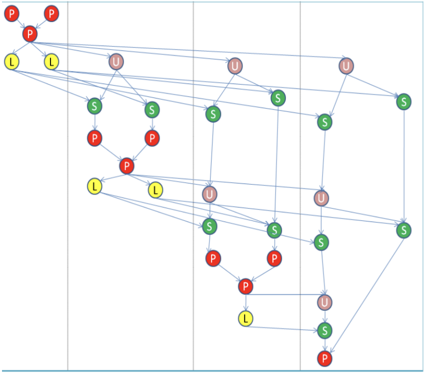
- Assume $O(n^2/P)$ memory/processor, $O(n^3)$ algorithm
- Choose b near $n/\sqrt{P}$ (its upper bound)
- Bandwidth lower bound: $\Omega(n^2/\sqrt{P})$ – just $\log_2 P$ smaller
- Latency lower bound: $\Omega(\sqrt{P})$ – just polylog(P) smaller

Performance vs ScaLAPACK

- Parallel TSLU (LU on tall-skinny matrix)
  - IBM Power 5
    Up to 4.37x faster (16 procs, $1M \times 150$)
  - Cray XT4
    Up to 5.52x faster (8 procs, $1M \times 150$)
- Parallel CALU (LU on general matrices)
  - Intel Xeon (two socket, quad core)
    Up to 2.3x faster (8 cores, $10^6 \times 500$)
  - IBM Power 5
    Up to 2.29x faster (64 procs, 1000×1000)
  - Cray XT4
    Up to 1.81x faster (64 procs, 1000×1000)
- Details in SuperComputing'08 (LG, Demmel, Xiang), IPDPS'10 (S. Donfack, LG).

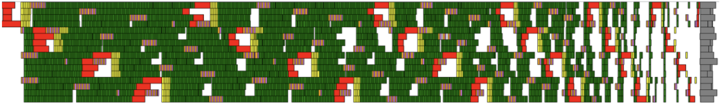
CALU and its task dependency graph

- The matrix is partitioned into blocks of size $T \times b$
- The computation of each block is associated with a task.

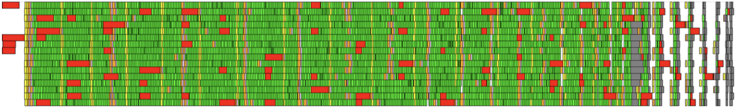


Scheduling CALU's Task Dependency Graph

- **Static scheduling**
  - \+ Good locality of data
  - \- Ignores noise



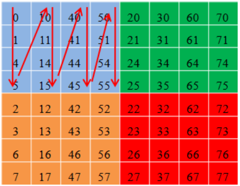
- **Dynamic scheduling**
  - \+ Keeps cores busy
  - \- Poor usage of data locality
  - \- Can have large dequeue overhead



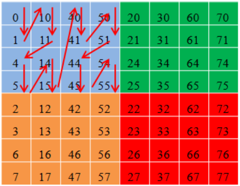
Lightweight scheduling

- Emerging complexities of multi- and mani-core processors suggest a need for self-adaptive strategies
  - One example is work stealing
- Goal
  - Design a tunable strategy that is able to provide a good trade-off between load balance, data locality, and dequeue overhead
  - Provide performance consistency
- Approach: combine static and dynamic scheduling
  - Shown to be efficient for regular mesh computation [B. Gropp and V. Kale]

Possible data layouts



Block cyclic layout (BCL)



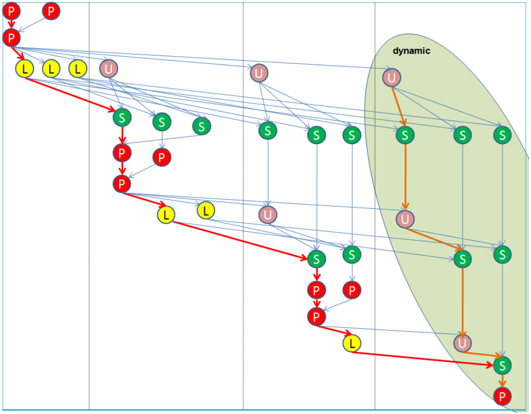
Two level block layout (2l-BL)

Data layout/scheduling	Static	Dynamic	Static(%dynamic)
Column Major Layout (CM)		x	
Block Cyclic Layout (BCL)	x	x	x
2-level Block Cyclic Layout (2l-BL)	x	x	x

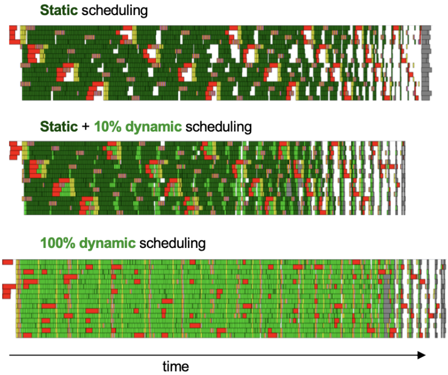
Lightweight scheduling

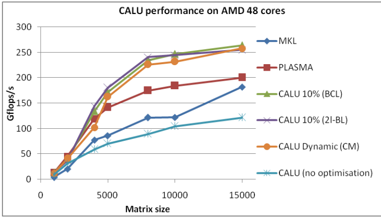
- A self-adaptive strategy to provide
  - A good trade-off between load balance, data locality, and dequeue overhead.
  - Performance consistency
  - Shown to be efficient for regular mesh computation [B. Gropp and V. Kale]

- Combined static/dynamic scheduling
  - A thread executes in priority its statically assigned tasks
  - When no task ready, it picks a ready task from the dynamic part
  - The size of the dynamic part is guided by a performance model



Best performance of CALU on multicore architectures





- Reported performance for PLASMA uses LU with block pairwise pivoting

Acknowledgement

- Some of the examples taken from [Golub and Van Loan, 1996]

References (1)

Golub, G. H. and Van Loan, C. F. (1996).
Matrix Computations (3rd Ed.).
Johns Hopkins University Press, Baltimore, MD, USA.

Grigori, L., Demmel, J., and Xiang, H. (2011).
CALU: a communication optimal LU factorization algorithm.
SIAM Journal on Matrix Analysis and Applications, 32:1317–1350.

N.J.Higham (2002).
Accuracy and Stability of Numerical Algorithms.
SIAM, second edition.