Communication avoiding algorithms in dense linear algebra

Laura Grigori
ALPINES
INRIA Rocquencourt - LJLL, UPMC
On sabbatical at UC Berkeley

Plan

- Motivation
- Selected past work on reducing communication
- Communication complexity of linear algebra operations
- Communication avoiding for dense linear algebra
  - **LU, QR**, Rank Revealing QR factorizations
  - Progressively implemented in ScaLAPACK and LAPACK
  - Algorithms for multicore processors
- Conclusions

The role of numerical linear algebra

- Challenging applications often rely on solving linear algebra problems
- Linear systems of equations

  Solve $Ax = b$, where $A \in \boldsymbol{R}^{nxn}$, $b \in \boldsymbol{R}^n$, $x \in \boldsymbol{R}^n$

  - Direct methods

    $PA = LU$, then solve $P^T LUx = b$

    LU factorization is backward stable,

    $$\left\| PA - \widehat{L} \cdot \widehat{U} \right\|_\infty \text{ is small, close to machine epsilon in practice}$$

  - Iterative methods
    - Find a solution x_k from $x_0 + K_k (A, r_0)$, where $K_k (A, r_0) = span\ \{r_0, A\, r_0, \ldots, A^{k-1}\, r_0\}$ such that the Petrov-Galerkin condition $b - Ax_k \perp L_k$ is satisfied,
      where L_k is a subspace of dimension k and $r_0 = Ax_0 - b$.
    - Convergence depends on $\kappa(A)$ and the eigenvalue distribution (for SPD matrices).

Least Square (LS) Problems

- Given $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^{m}$, solve $\min_x \|Ax - b\|_2$.
- Any solution of the LS problem satisfies the normal equations: $A^T Ax = A^T b$
- Given the QR factorization of A

$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix} \quad \begin{matrix} A \text{ is } m \times n \text{ real matrix}, m \geq n \\ \text{where } R \text{ is } n \times n \text{ upper triangular matrix} \\ Q \text{ is } m \times m \text{ orthogonal matrix} \end{matrix}$$

if *rank(A)* = *rank(R)* = *n*, then the LS solution is given by $Rx = \left(Q^T b\right)(1:n)$

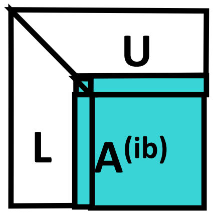
- The QR factorization is column-wise backward stable

$\left\|A - \hat{Q}\hat{R}\right\|_2$ is small, close to machine epsilon in practice

Evolution of numerical libraries

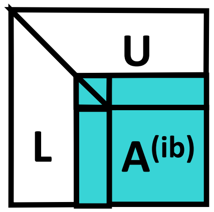
LINPACK (70's)

- vector operations, uses BLAS1/2
- HPL benchmark based on Linpack LU factorization



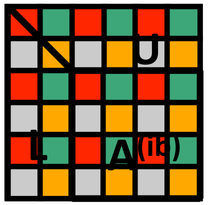
LAPACK (80's)

- Block versions of the algorithms used in LINPACK
- Uses BLAS3



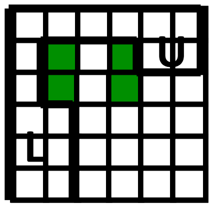
ScaLAPACK (90's)

- Targets distributed memories
- 2D block cyclic distribution of data
- PBLAS based on message passing



PLASMA (2008): new algorithms

- Targets many-core
- Block data layout
- Low granularity, high asynchronicity

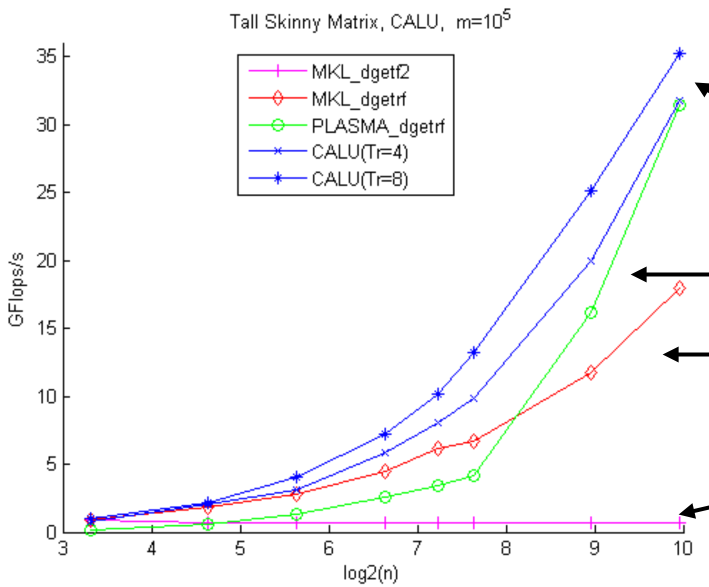


Project developed by U Tennessee Knoxville, UC Berkeley, other collaborators.

Source: inspired from J. Dongarra, UTK, J. Langou, CU Denver

Evolution of numerical libraries

- Did we need new algorithms?
  - Results on two-socket, quad-core Intel Xeon EMT64 machine, 2.4 GHz per core, peak performance 76.5 Gflops/s
  - LU factorization of an m-by-n matrix, m=10^5 and n varies from 10 to 1000



LU using communication avoiding approach

LU from PLASMA using operations on tiles

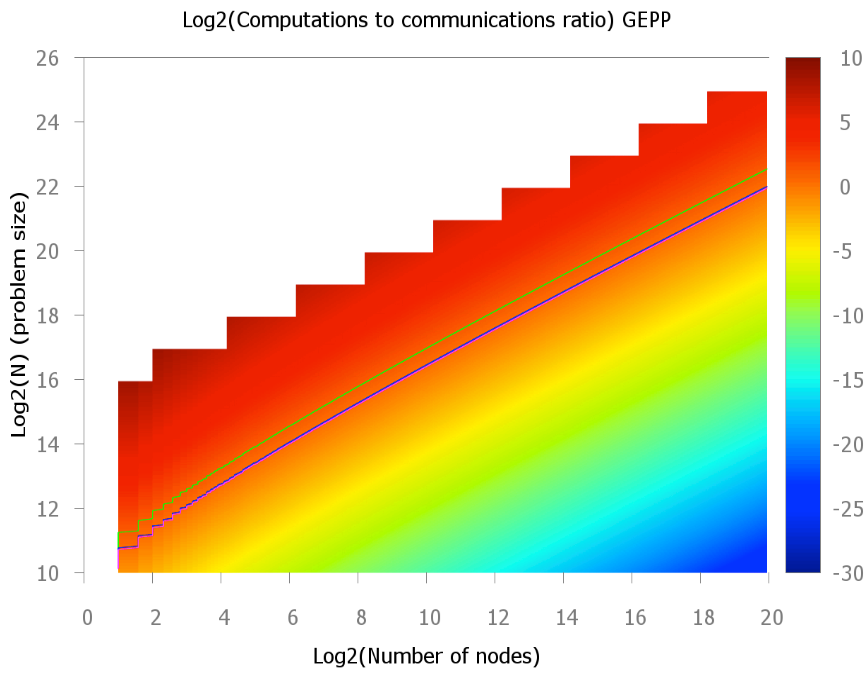
LU from Intel MKL using lapack block operations
Lapack will not get even 1/2 of this performance

LU using vector operations

Approaches for reducing communication

- Tuning
  - Overlap communication and computation, at most a factor of 2 speedup

- Same numerical algorithm, different schedule of the computation
  - Block algorithms for NLA
    - Barron and Swinnerton-Dyer, 1960
    - ScaLAPACK, Blackford et al 97
  - Cache oblivious algorithms for NLA
    - Gustavson 97, Toledo 97, Frens and Wise 03, Ahmed and Pingali 00



- Same algebraic framework, different numerical algorithm
  - The approach used in CA algorithms
  - More opportunities for reducing communication, may affect stability

Motivation

- The communication problem needs to be taken into account higher in the computing stack
- A paradigm shift in the way the numerical algorithms are devised is required
- Communication avoiding algorithms - a novel perspective for numerical linear algebra
  - Minimize volume of communication
  - Minimize number of messages
  - Minimize over multiple levels of memory/parallelism
  - Allow redundant computations (preferably as a low order term)

Communication Complexity of Dense Linear Algebra

- Matrix multiply, using $2n^3$ flops (sequential or parallel)
  - Hong-Kung (1981), Irony/Tishkin/Toledo (2004)
  - Lower bound on Bandwidth = Ω (#flops / $M^{1/2}$)
  - Lower bound on Latency = Ω (#flops / $M^{3/2}$)

- Same lower bounds apply to LU using reduction
  - Demmel, LG, Hoemmen, Langou 2008

$$\begin{pmatrix} I & & -B \\ A & I & \\ & & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ & & I \end{pmatrix} \cdot \begin{pmatrix} I & & -B \\ & I & AB \\ & & I \end{pmatrix}$$

- And to almost all direct linear algebra [Ballard, Demmel, Holtz, Schwartz, 09]

Sequential algorithms and communication bounds

Algorithm	Minimizing #words (not #messages)	Minimizing #words and #messages
Cholesky	LAPACK	[Gustavson, 97] [Ahmed, Pingali, 00]
LU	LAPACK (few cases) [Toledo,97], [Gustavson, 97] both use partial pivoting	[LG, Demmel, Xiang, 08] [Khabou, Demmel, LG, Gu, 12] uses tournament pivoting
QR	LAPACK (few cases) [Elmroth,Gustavson,98]	[Frens, Wise, 03], 3x flops [Demmel, LG, Hoemmen, Langou, 08] [Ballard et al, 14]
RRQR		[Demmel, LG, Gu, Xiang 11] uses tournament pivoting, 3x flops

- Only several references shown for block algorithms (LAPACK), cache-oblivious algorithms and communication avoiding algorithms
- CA algorithms exist also for SVD and eigenvalue computation

2D Parallel algorithms and communication bounds

- If memory per processor = n^2 / P, the lower bounds become
  #words_moved $\geq \Omega (n^2 / P^{1/2})$, #messages $\geq \Omega (P^{1/2})$

Algorithm	Minimizing #words (not #messages)	Minimizing #words and #messages
Cholesky	ScaLAPACK	ScaLAPACK
LU	ScaLAPACK uses partial pivoting	[LG, Demmel, Xiang, 08] [Khabou, Demmel, LG, Gu, 12] uses tournament pivoting
QR	ScaLAPACK	[Demmel, LG, Hoemmen, Langou, 08] [Ballard et al, 14]
RRQR	ScaLAPACK	[Demmel, LG, Gu, Xiang 13] uses tournament pivoting, 3x flops

- Only several references shown, block algorithms (ScaLAPACK) and communication avoiding algorithms
- CA algorithms exist also for SVD and eigenvalue computation

Scalability of communication optimal algorithms

- 2D communication optimal algorithms, $M = 3 \cdot n^2/P$

  (matrix distributed over a $P^{1/2}$-by- $P^{1/2}$ grid of processors)

  $T_P = O(n^3/P)\gamma + \Omega(n^2/P^{1/2})\beta + \Omega(P^{1/2})\alpha$

  - Isoefficiency: **$n^3 \propto P^{1.5}$** and **$n^2 \propto P$**
  - For GEPP, **$n^3 \propto P^{2.25}$** [Grama et al, 93]

- 3D communication optimal algorithms, $M = 3 \cdot P^{1/3}(n^2/P)$

  (matrix distributed over a $P^{1/3}$-by- $P^{1/3}$-by- $P^{1/3}$ grid of processors)

  $T_P = O(n^3/P)\gamma + \Omega(n^2/P^{2/3})\beta + \Omega(\log(P))\alpha$

  - Isoefficiency: **$n^3 \propto P$** and **$n^2 \propto P^{2/3}$**

- 2.5D algorithms with $M = 3 \cdot c \cdot (n^2/P)$, and 3D algorithms exist for matrix multiplication and LU factorization
  - References: Dekel et al 81, Agarwal et al 90, 95, Johnsson 93, McColl and Tiskin 99, Irony and Toledo 02, Solomonik and Demmel 2011

2.5D algorithms for LU, QR

- Assume c>1 copies of data, memory per processor is $M \approx c \cdot (n^2/P)$

- For matrix multiplication
  - The bandwidth is reduced by a factor of $c^{1/2}$
  - The latency is reduced by a factor of $c^{3/2}$
  - Perfect Strong Scaling regime, given P such that $M = 3n^2 /P$

    $T(cP) = T(P)/c$

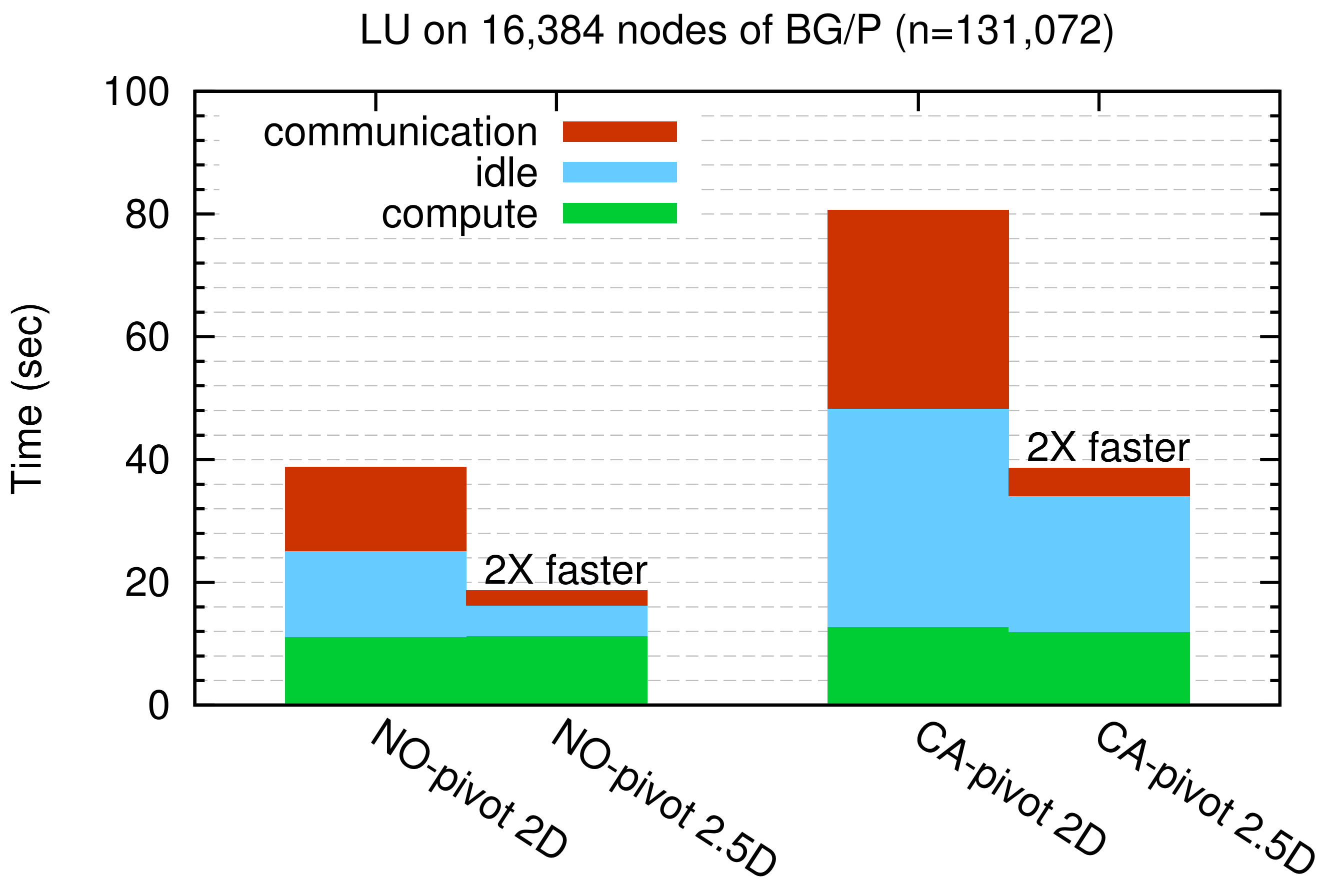
- For LU, QR
  - The bandwidth can be reduced by a factor of $c^{1/2}$
  - But then the latency will increase by a factor of $c^{1/2}$

  - Thm [Solomonik et al]: Perfect Strong Scaling impossible for LU, because

    Latency*Bandwidth = $\Omega(n^2)$
  - Conjecture: this applies to other factorizations as QR, RRQR, etc.

2.5D LU with and without pivoting

- 2.5D algorithms with $M = 3 \cdot c \cdot (n^2/P)$, and 3D algorithms exist for matrix multiplication and LU factorization
  - References: Dekel et al 81, Agarwal et al 90, 95, Johnsson 93, McColl and Tiskin 99, Irony and Toledo 02, Solomonik and Demmel 2011 (data presented below)



The algebra of LU factorization

- Compute the factorization PA = LU

- Given the matrix

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

Let

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

The need for pivoting

- For stability avoid division by small elements, otherwise ||A-LU|| can be large
  - Because of roundoff error
- For example

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

has an LU factorization if we permute the rows of A

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

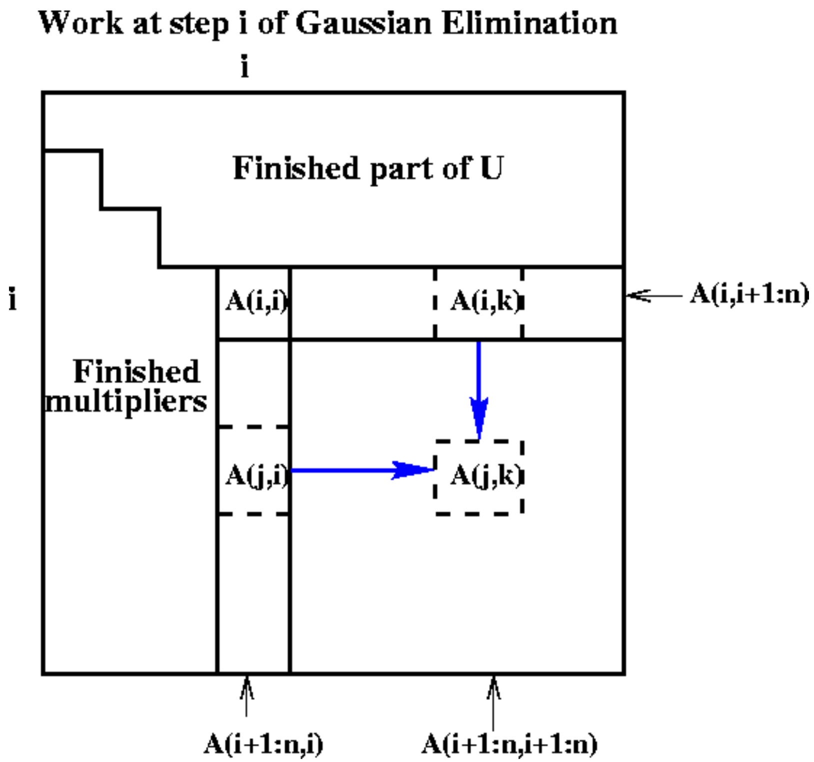
- Partial pivoting allows to bound all elements of L by 1.

LU with partial pivoting – BLAS 2 algorithm

```
for i = 1 to n-1
    Let A(j,i) be elt. of max magnitude in A(i+1:n,i)
    Permute rows i and j
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- Algorithm using BLAS 1/2 operations

```
for i = 1 to n-1
    Let A(j,i) be of elt. of max magnitude in A(i+1:n,i)
    Permute rows i and j
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
            … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
            - A(i+1:n , i) * A(i , i+1:n)
            … BLAS 2 (rank-1 update)
```



Source slide: J. Demmel

Block LU factorization – obtained by delaying updates

- Matrix A of size *nxn* is partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \text{ where } A_{11} \text{ is } b \times b$$

- The first step computes LU with partial pivoting of the first block:

$$P_1 \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} U_{11}$$

- The factorization obtained is:

$$P_1 A = \begin{pmatrix} L_{11} & \\ L_{21} & I_{n-b} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & A_{22}^1 \end{pmatrix}, \text{ where } \begin{aligned} U_{12} &= L_{11}^{-1} A_{12} \\ A_{22}^1 &= A_{22} - U_{12} \end{aligned}$$

- The algorithm continues recursively on the trailing matrix A_{22}^1

Block LU factorization – the algorithm

1. Compute LU with partial pivoting of the first panel

$$P_1 \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} U_{11}$$

2. Pivot by applying the permutation matrix P_1 on the entire matrix

$$P_1 A = \overline{A}$$

3. Solve the triangular system to compute a block row of U

$$U_{12} = L_{12}^{-1} \overline{A}_{12}$$

4. Update the trailing matrix

$$\overline{A}_{22}^{1} = \overline{A}_{22} - L_{21} U_{12}$$

5. The algorithm continues recursively on the trailing matrix $\overline{A}_{22}^{1}$

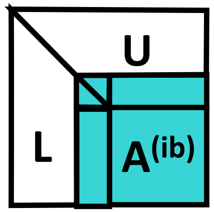
LU factorization (as in ScaLAPACK pdgetrf)

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

For ib = 1 to n-1 step b

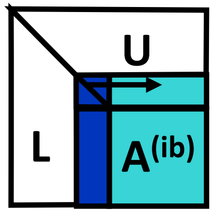
$A^{(ib)} = A(ib:n, ib:n)$

#messages



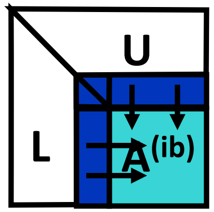
(1) Compute panel factorization $O(n \log_2 P_r)$

- find pivot in each column, swap rows



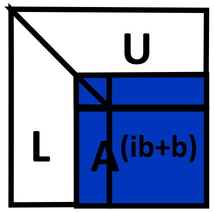
(2) Apply all row permutations $O(n/b(\log_2 P_c + \log_2 P_r))$

- broadcast pivot information along the rows
- swap rows at left and right



(3) Compute block row of U $O(n/b \log_2 P_c)$

- broadcast right diagonal block of L of current panel



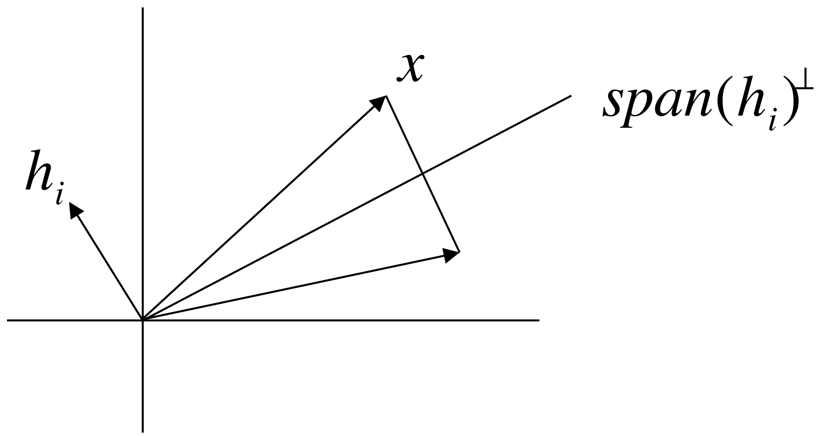
(4) Update trailing matrix $O(n/b(\log_2 P_c + \log_2 P_r))$

- broadcast right block column of L
- broadcast down block row of U

General scheme for
QR factorization by Householder transformations

The Householder matrix

$$H_i = I - \tau_i h_i h_i^T$$



has the following properties:

- is symmetric and orthogonal,

  $H_i^2 = I$,
- is independent of the scaling of v,
- it reflects *x* about the hyperplane $span(h_i)^{\perp}$

- For QR, we choose a Householder matrix that allows to annihilate the elements of a vector x, except first one.

General scheme for QR factorization by Householder transformations

- Apply Householder transformations to annihilate subdiagonal entries

$$A = \begin{pmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{pmatrix} = H_1 \begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{pmatrix} = H_1 \begin{pmatrix} 1 & \\ & \tilde{H}_2 \end{pmatrix} \begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{pmatrix} =$$

$$= H_1 H_2 \begin{pmatrix} 1 & & \\ & 1 & \\ & & \tilde{H}_3 \end{pmatrix} \begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{pmatrix} = H_1 H_2 H_3 R = QR$$

- For A of size mxn, the factorization can be written as:

$$H_n H_{n-1} \ldots H_2 H_1 A = R \rightarrow A = (H_n H_{n-1} \ldots H_2 H_1)^T R$$

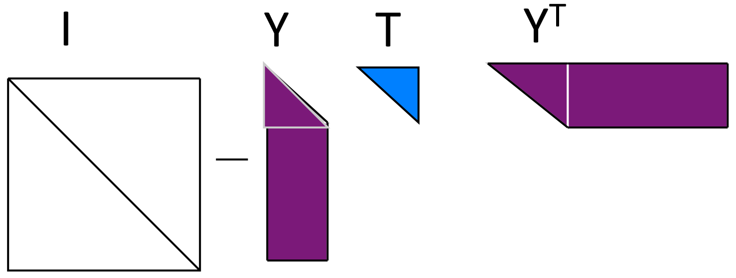
$$Q = H_1 H_2 \ldots H_n$$

Compact representation for Q

- Orthogonal factor *Q* can be represented implicitly as

$$Q = H_1 H_2 \ldots H_b = (I - \tau_1 h_1 h_1^T)\ldots(I - \tau_b h_b h_b^T) = I - YTY^T, \text{ where}$$

$$Y = \begin{pmatrix} h_1 & h_2 & \ldots & h_b \end{pmatrix}$$



- Example for *b*=2:

$$Y = (h_1 | h_2), \qquad \mathrm{T} = \begin{pmatrix} \tau_1 & -\tau_1 h_1^T h_2 \tau_2 \\ & \tau_2 \end{pmatrix}$$

Algebra of block QR factorization

Matrix A of size *nxn* is partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \text{ where } A_{11} \text{ is } b \times b$$

Block QR algebra

The first step of the block QR factorization algorithm computes:

$$Q_1^T A = \begin{bmatrix} R_{11} & R_{12} \\ & A_{22}^1 \end{bmatrix}$$

The algorithm continues recursively on the trailing matrix A_{22}^1

Block QR factorization

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = Q_1 \begin{pmatrix} R_{11} & R_{12} \\ & A_{22}^{1} \end{pmatrix}$$

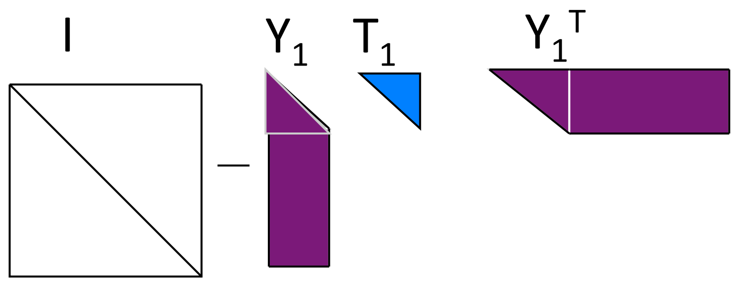
Block QR algebra:

1. Compute panel factorization:

$$\begin{pmatrix} A_{11} \\ A_{12} \end{pmatrix} = Q_1 \begin{pmatrix} R_{11} \\ \end{pmatrix}$$

2. Compute the compact representation:

$$Q_1 = I - Y_1 T_1 Y_1^T$$



3. Update the trailing matrix:

$$\left(I - Y_1 T_1^T Y_1^T\right)\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - Y_1\left(T_1^T\left(Y_1^T\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}\right)\right) = \begin{pmatrix} R_{12} \\ A_{22}^{1} \end{pmatrix}$$

4. The algorithm continues recursively on the trailing matrix.

TSQR: QR factorization of a tall skinny matrix using Householder transformations

- QR decomposition of m x b matrix W, m >> b
  - P processors, block row layout

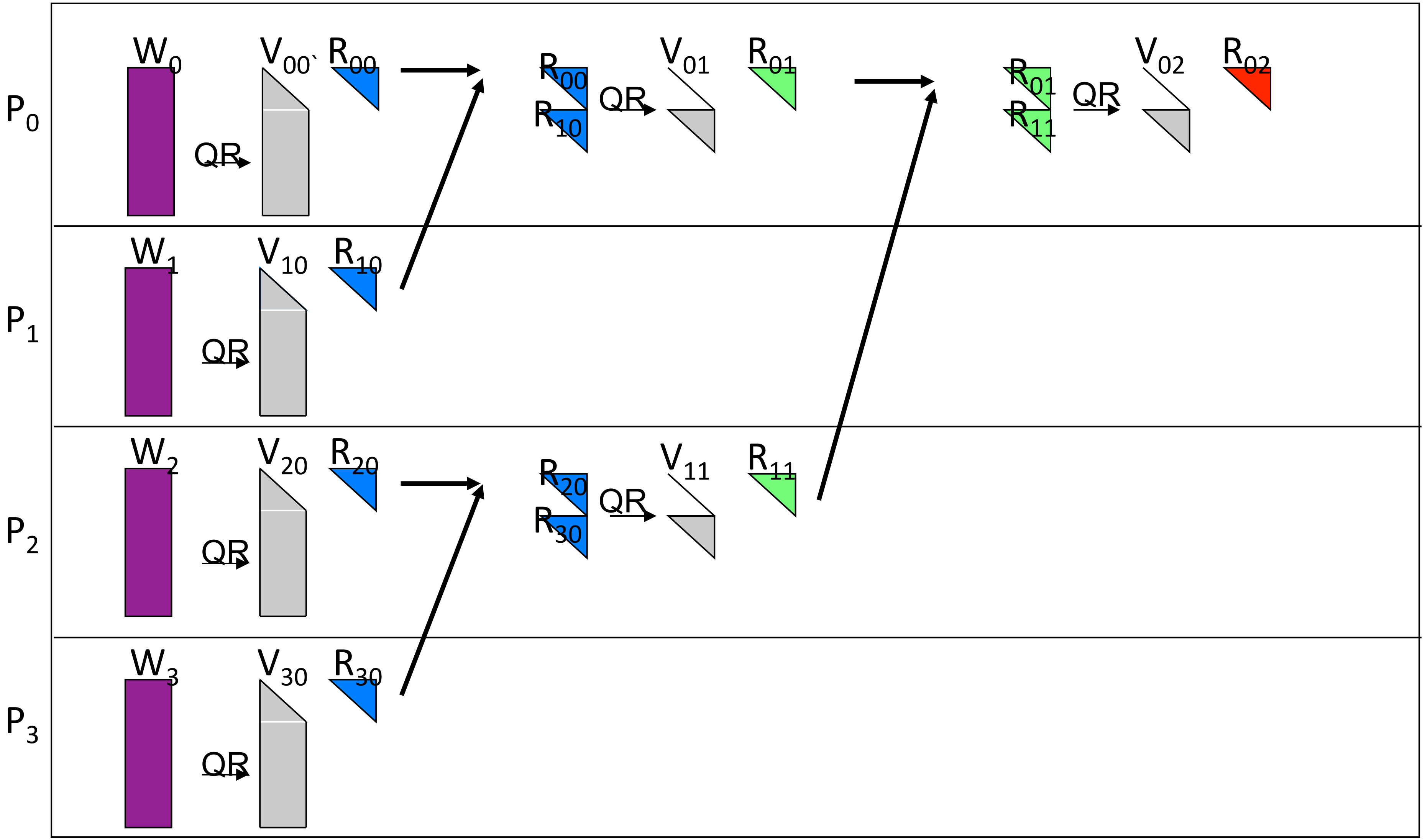
- Classic Parallel Algorithm
  - Compute Householder vector for each column
  - Number of messages ∝ b log P

- Communication Avoiding Algorithm
  - Reduction operation, with QR as operator
  - Number of messages ∝ log P

$$W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \begin{matrix} \to \\ \to \\ \to \\ \to \end{matrix} \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix} \begin{matrix} \searrow \\ \nearrow \\ \searrow \\ \nearrow \end{matrix} \begin{matrix} R_{01} \\ \\ R_{11} \end{matrix} \begin{matrix} \searrow \\ \nearrow \end{matrix} R_{02}$$

J. Demmel, LG, M. Hoemmen, J. Langou, 08

Parallel TSQR



References: Golub, Plemmons, Sameh 88, Pothen, Raghavan, 89, Da Cunha, Becker, Patterson, 02

Algebra of TSQR

Parallel:

$$W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \begin{array}{c} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{array} \begin{array}{c} \searrow \\ \nearrow \\ \searrow \\ \nearrow \end{array} \begin{array}{c} R_{01} \\ \\ R_{11} \end{array} \begin{array}{c} \searrow \\ \nearrow \end{array} R_{02}$$

$$W = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \begin{pmatrix} Q_{00}R_{00} \\ \hline Q_{10}R_{10} \\ \hline Q_{20}R_{20} \\ \hline Q_{30}R_{30} \end{pmatrix} = \begin{pmatrix} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{pmatrix} \cdot \begin{pmatrix} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ \hline R_{30} \end{pmatrix}$$

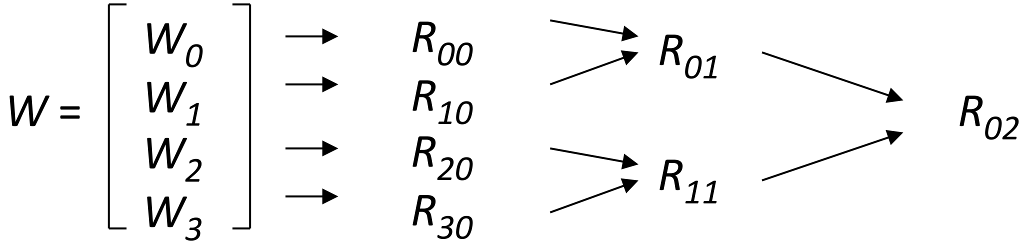
$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}R_{01} \\ \hline Q_{11}R_{11} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ \hline & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} \qquad \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} = Q_{02}R_{02}$$

Q is represented implicitly as a product

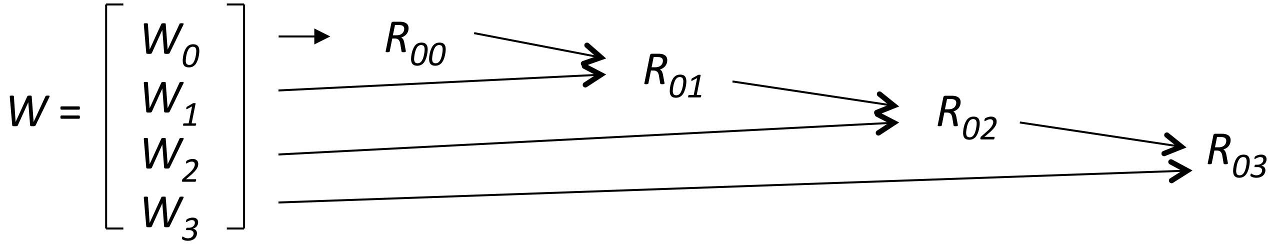
Output: $\{Q_{00}, Q_{10}, Q_{00}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02}\}$

Flexibility of TSQR and CAQR algorithms

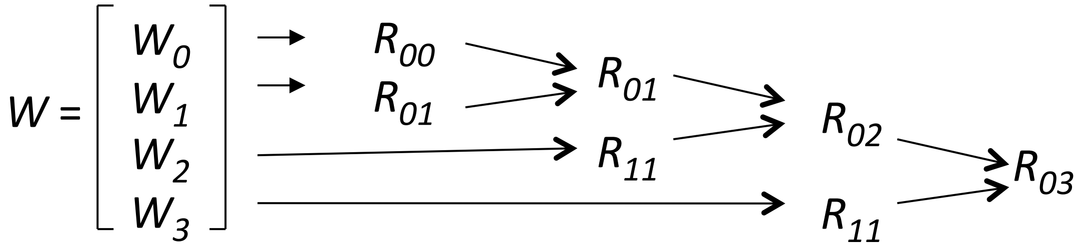
Parallel:



Sequential:



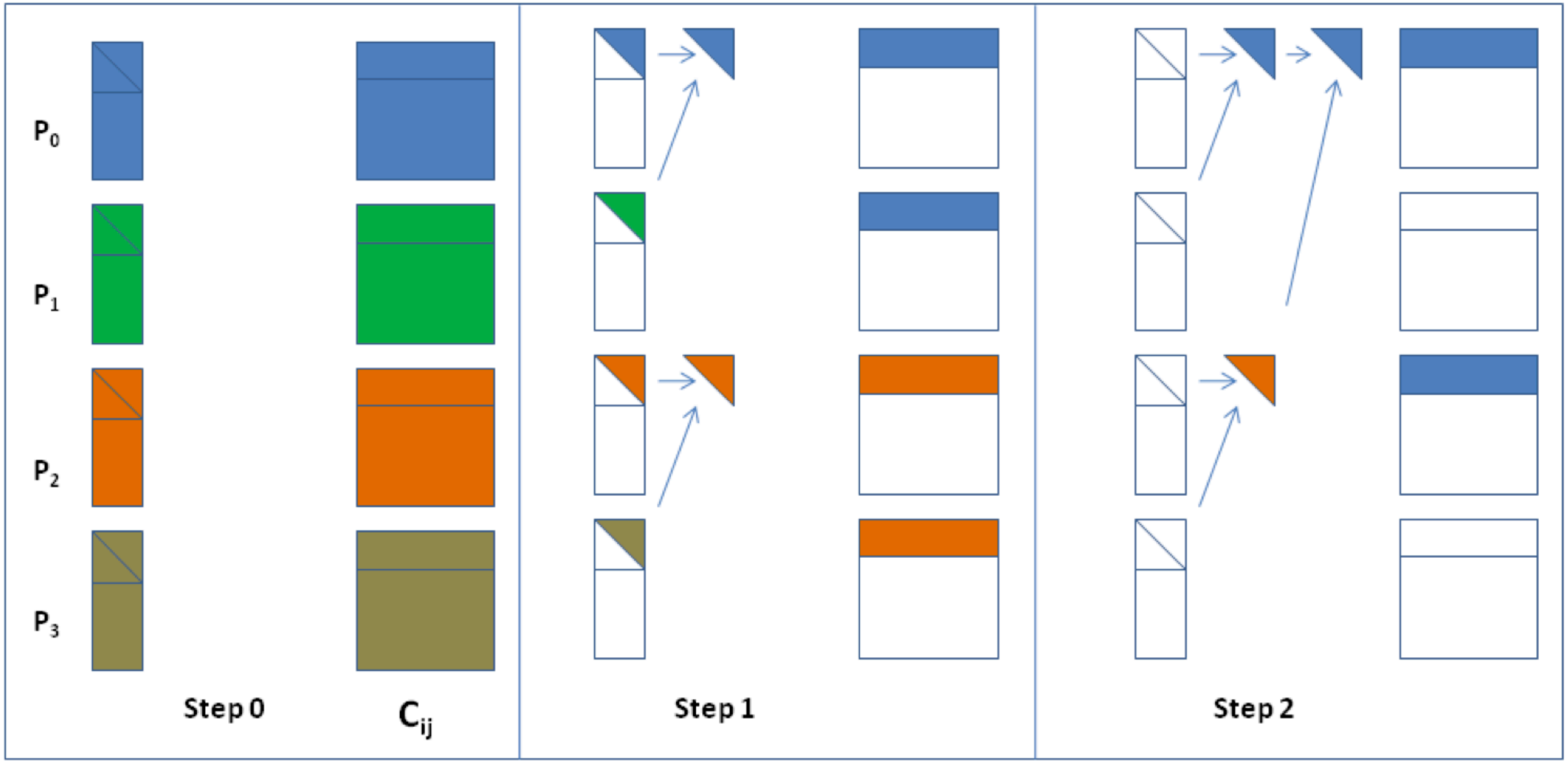
Dual Core:



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

CAQR for general matrices

- Use TSQR for panel factorizations
- Update the trailing matrix - triggered by the reduction tree used for the panel factorization



QR for General Matrices

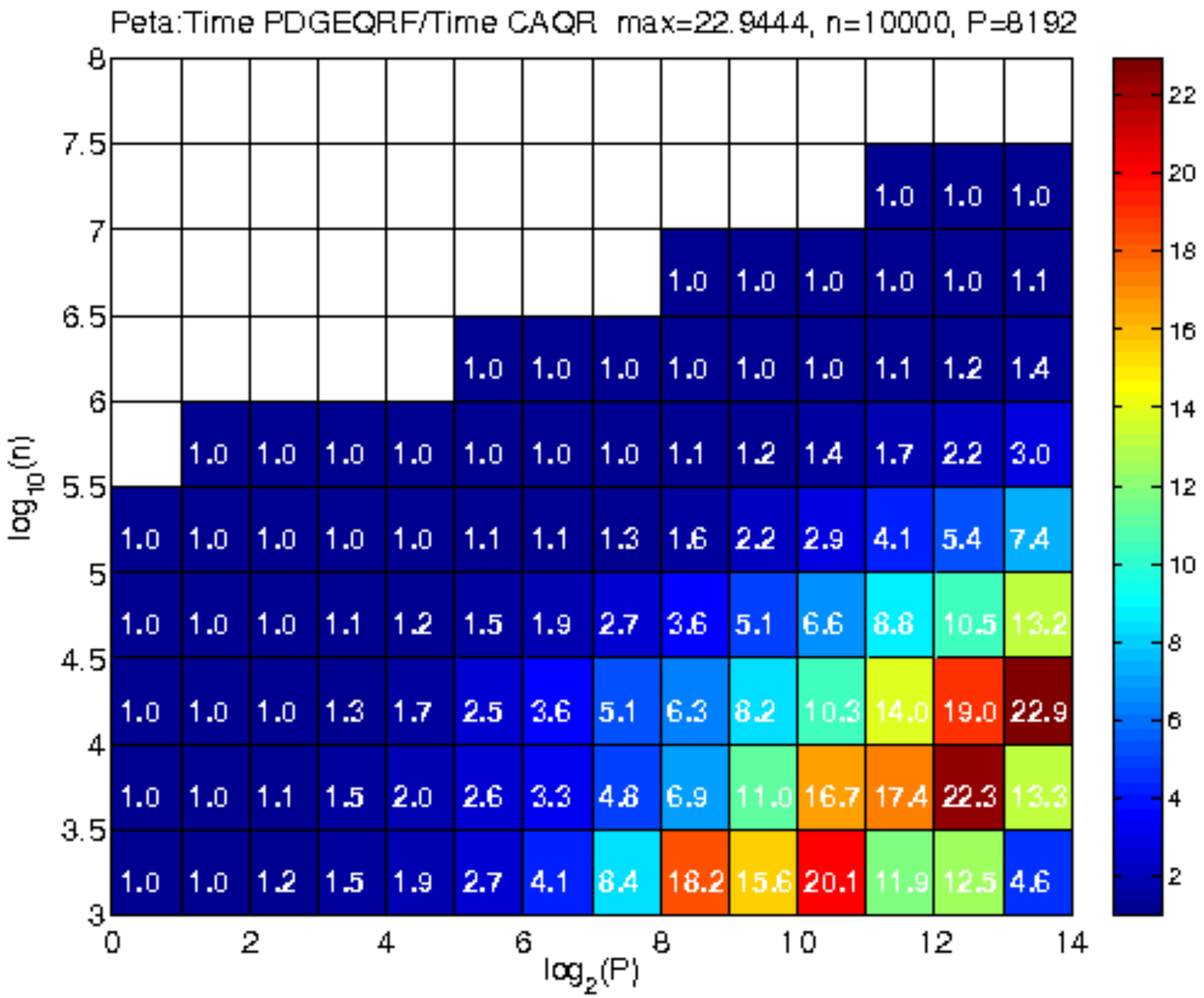
- CAQR – Communication Avoiding QR for general A
  - Use TSQR for panel factorizations
  - Apply to rest of matrix
- Cost of CAQR vs ScaLAPACK's PDGEQRF
  - n x n matrix on $P^{1/2}$ x $P^{1/2}$ processor grid, block size b
  - Flops: $(4/3)n^3/P + (3/4)n^2 b \log P/P^{1/2}$ vs $(4/3)n^3/P$
  - Bandwidth: $(3/4)n^2 \log P/P^{1/2}$ vs same
  - Latency: $2.5\, n \log P / \mathbf{b}$ vs $1.5\, n \log P$
- Close to optimal (modulo log P factors)
  - Assume: $O(n^2/P)$ memory/processor, $O(n^3)$ algorithm,
  - Choose b near $n / P^{1/2}$ (its upper bound)
  - Bandwidth lower bound: $\Omega(n^2 / P^{1/2})$ – just log(P) smaller
  - Latency lower bound: $\Omega(P^{1/2})$ – just polylog(P) smaller

Performance of TSQR vs Sca/LAPACK

- Parallel
  - Intel Xeon (two socket, quad core machine), 2010
    - Up to **5.3x speedup** (8 cores, 10^5 x 200)
  - Pentium III cluster, Dolphin Interconnect, MPICH, 2008
    - Up to **6.7x speedup** (16 procs, 100K x 200)
  - BlueGene/L, 2008
    - Up to **4x speedup** (32 procs, 1M x 50)
  - Tesla C 2050 / Fermi (Anderson et al)
    - Up to **13x** (110,592 x 100)
  - Grid – **4x** on 4 cities vs 1 city (Dongarra, Langou et al)
  - QR computed locally using recursive algorithm (Elmroth-Gustavson) – enabled by TSQR

- Results from many papers, for some see [Demmel, LG, Hoemmen, Langou, SISC 12], [Donfack, LG, IPDPS 10].

Modeled Speedups of CAQR vs ScaLAPACK



Petascale

up to 22.9x

IBM Power 5

up to 9.7x

“Grid”

up to 11x

Petascale machine with 8192 procs, each at 500 GFlops/s, a bandwidth of 4 GB/s.

$$\gamma = 2 \cdot 10^{-12} s, \alpha = 10^{-5} s, \beta = 2 \cdot 10^{-9} s / word.$$

The LU factorization of a tall skinny matrix

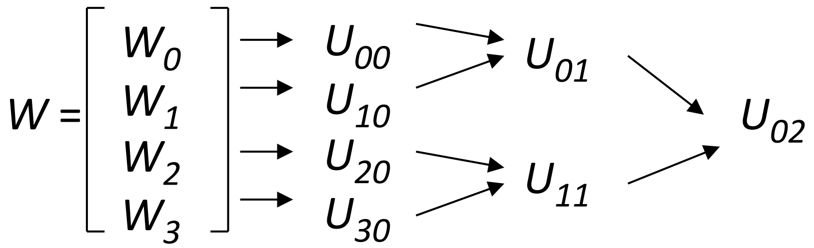
First try the obvious generalization of TSQR.

$$W = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \underbrace{\begin{pmatrix} \Pi_{00} & & & \\ & \Pi_{10} & & \\ & & \Pi_{20} & \\ & & & \Pi_{30} \end{pmatrix}}_{\Pi_0} \cdot \begin{pmatrix} L_{00} & & & \\ & L_{10} & & \\ & & L_{20} & \\ & & & L_{30} \end{pmatrix} \cdot \begin{pmatrix} U_{00} \\ U_{10} \\ U_{20} \\ U_{30} \end{pmatrix}$$

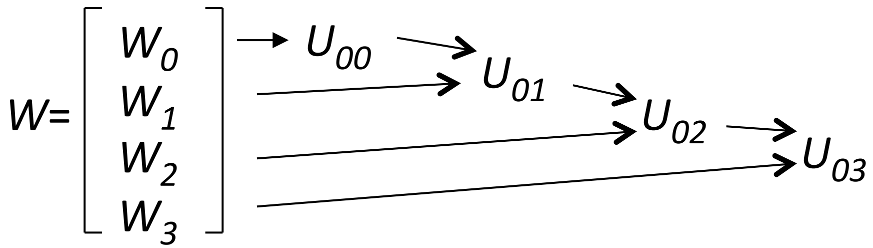
$$\begin{pmatrix} U_{00} \\ U_{10} \\ U_{20} \\ U_{30} \end{pmatrix} = \underbrace{\begin{pmatrix} \Pi_{01} & \\ & \Pi_{11} \end{pmatrix}}_{\Pi_1} \cdot \begin{pmatrix} L_{01} & \\ & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{01} \\ U_{11} \end{pmatrix} \qquad \begin{pmatrix} U_{01} \\ U_{11} \end{pmatrix} = \underbrace{\Pi_{02}}_{\Pi_2} L_{02} U_{02}$$

Obvious generalization of TSQR to LU

- Block parallel pivoting:
  - uses a binary tree and is optimal in the parallel case



- 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



Stability of the LU factorization

- The backward stability of the LU factorization of a matrix A of size n-by-n

$$\left\| |\hat{L}| \cdot |\hat{U}| \right\|_\infty \le (1 + 2(n^2 - n) g_w) \|A\|_\infty$$

depends on the growth factor

$$g_W = \frac{\max_{i,j,k} |a_{ij}^k|}{\max_{i,j} |a_{ij}|}$$ where a_{ij}^k are the values at the k-th step.

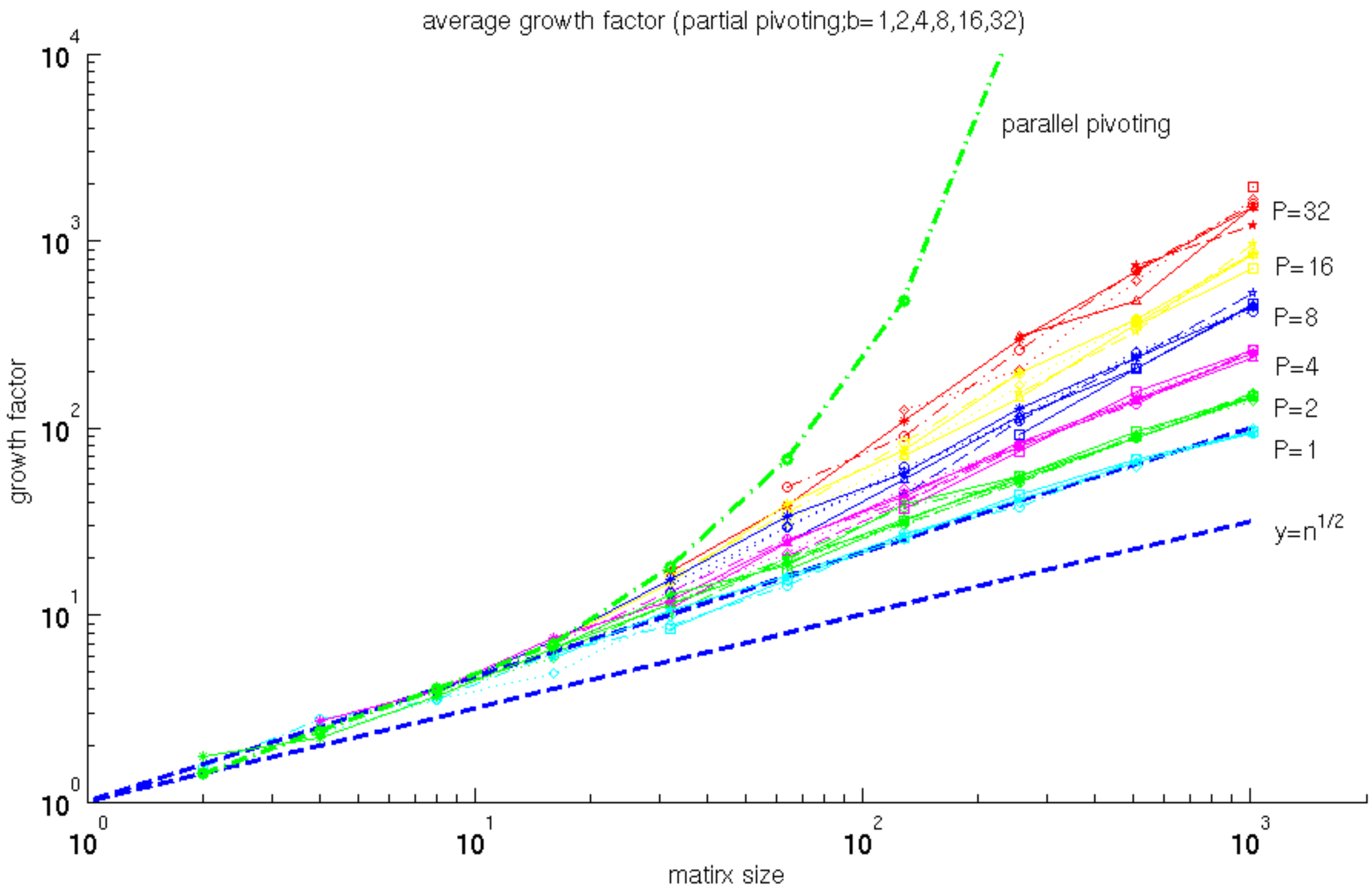
- $g_W \le 2^{n-1}$, but in practice it is on the order of $n^{2/3}$ -- $n^{1/2}$

- 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.

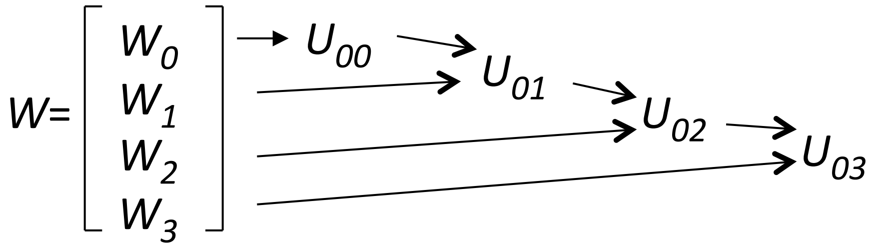
Block parallel pivoting

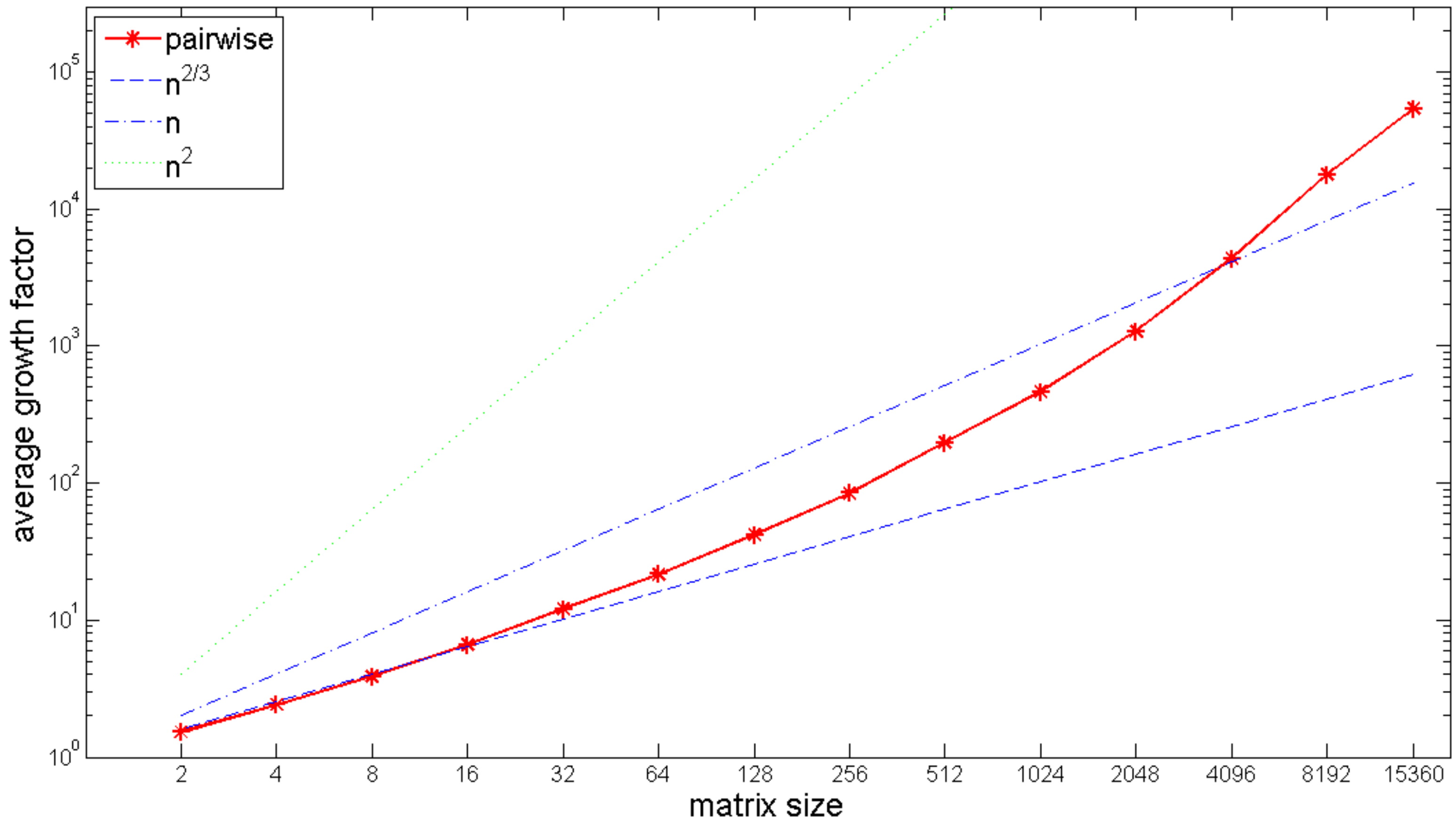


- Unstable for large number of processors P
- When P=number rows, it corresponds to parallel pivoting, known to be unstable (Trefethen and Schreiber, 90)

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

$$
A = \begin{array}{c} \begin{array}{cc} b & n-b \end{array} \\ \begin{pmatrix} \overbrace{A_{11}} & \overbrace{A_{21}} \\ A_{21} & A_{22} \end{pmatrix} \end{array} \begin{array}{l} \}\ b \\ \}\ n-b \end{array} \text{, where} \quad W = \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}
$$

  - 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.

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

Tournament pivoting for a tall skinny matrix

1) Compute GEPP factorization of each $W_{i.}$, find permutation Π_0

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix} = \begin{pmatrix} \Pi_{00} L_{00} U_{00} \\ \hline \Pi_{10} L_{10} U_{10} \\ \hline \Pi_{20} L_{20} U_{20} \\ \hline \Pi_{30} L_{30} U_{30} \end{pmatrix},$$

Pick b pivot rows, form A_{00}

Same for A_{10}

Same for A_{20}

Same for A_{30}

2) Perform $\log_2(P)$ times GEPP factorizations of 2b-by-b rows, find permutations Π_1, Π_2

$$\begin{pmatrix} A_{00} \\ A_{10} \\ \hline A_{20} \\ A_{30} \end{pmatrix} = \begin{pmatrix} \prod_{01} L_{01} U_{01} \\ \hline \prod_{11} L_{11} U_{11} \end{pmatrix}$$

Pick b pivot rows, form A_{01}

Same for A_{11}

$$\begin{pmatrix} A_{01} \\ A_{11} \end{pmatrix} = \underbrace{\prod_{02}}_{\prod_2} L_{02} U_{02}$$

3) Compute LU factorization with no pivoting of the permuted matrix:

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

Tournament pivoting

P$_0$

$$W_0 = \begin{pmatrix} 2 & 4 \\ 0 & 1 \\ 2 & 0 \\ 1 & 2 \end{pmatrix} = \Pi_0 L_0 U_0 \qquad \Pi_0^T W_0 = \begin{pmatrix} 2 & 4 \\ 2 & 0 \end{pmatrix}$$

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

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

Good pivots for factorizing W

P$_1$

$$W_1 = \begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 4 & 1 \\ 1 & 0 \end{pmatrix} = \Pi_1 L_1 U_1 \qquad \Pi_1^T W_1 = \begin{pmatrix} 4 & 1 \\ 2 & 0 \end{pmatrix}$$

P$_2$

$$W_2 = \begin{pmatrix} 0 & 1 \\ 1 & 4 \\ 0 & 0 \\ 0 & 2 \end{pmatrix} = \Pi_2 L_2 U_2 \qquad \Pi_2^T W_2 = \begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix}$$

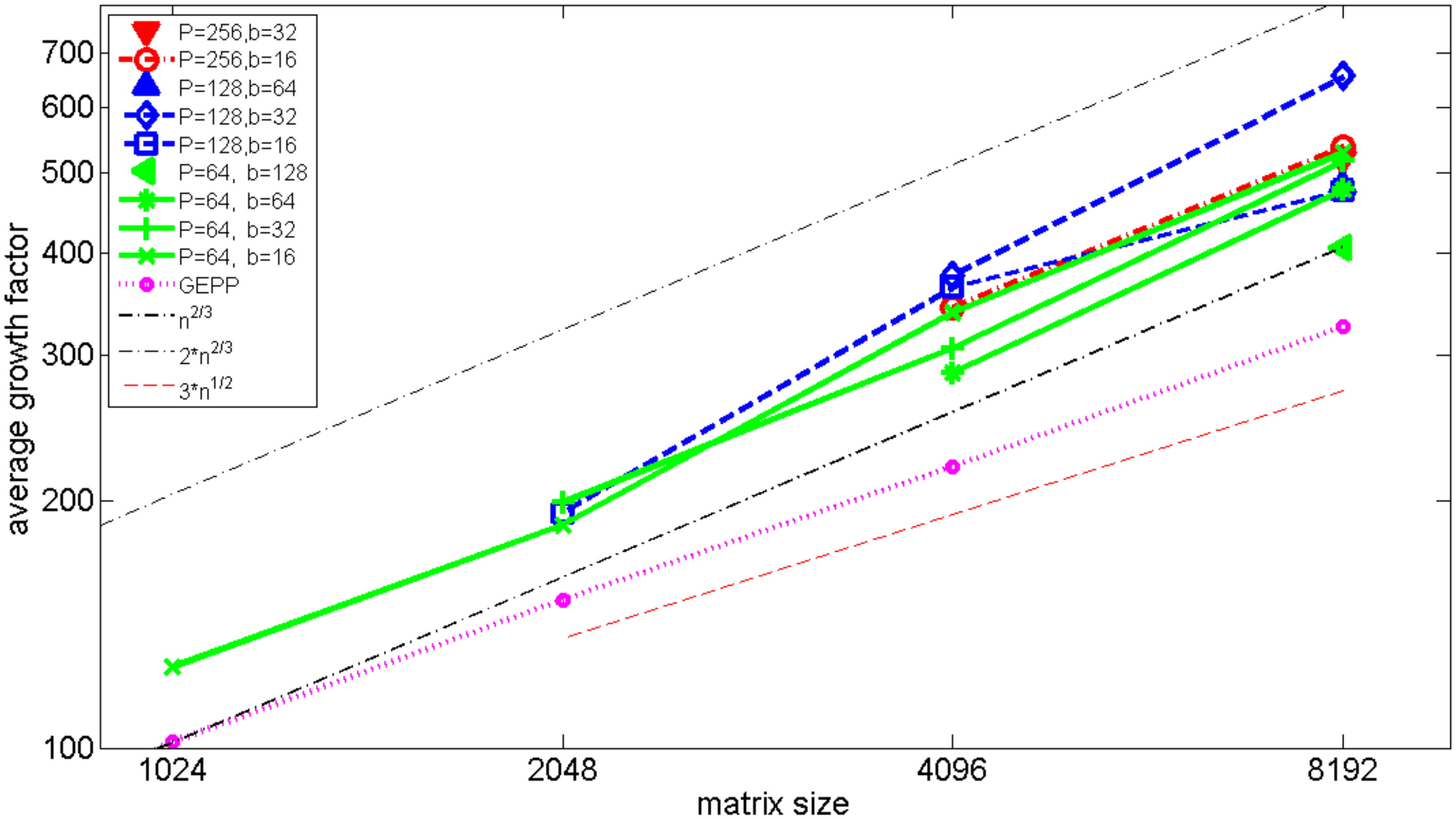
$$\overline{W}_2 = \begin{pmatrix} 1 & 4 \\ 0 & 2 \\ 4 & 2 \\ 0 & 2 \end{pmatrix} = \overline{\Pi}_2 \overline{L}_2 \overline{U}_2 \qquad \overline{\Pi}_2^T \overline{W}_2 = \begin{pmatrix} 4 & 2 \\ 1 & 4 \end{pmatrix}$$

P$_3$

$$W_3 = \begin{pmatrix} 2 & 1 \\ 0 & 2 \\ 1 & 0 \\ 4 & 2 \end{pmatrix} = \Pi_3 L_3 U_3 \qquad \Pi_3^T W_3 = \begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix}$$

time $\longrightarrow$

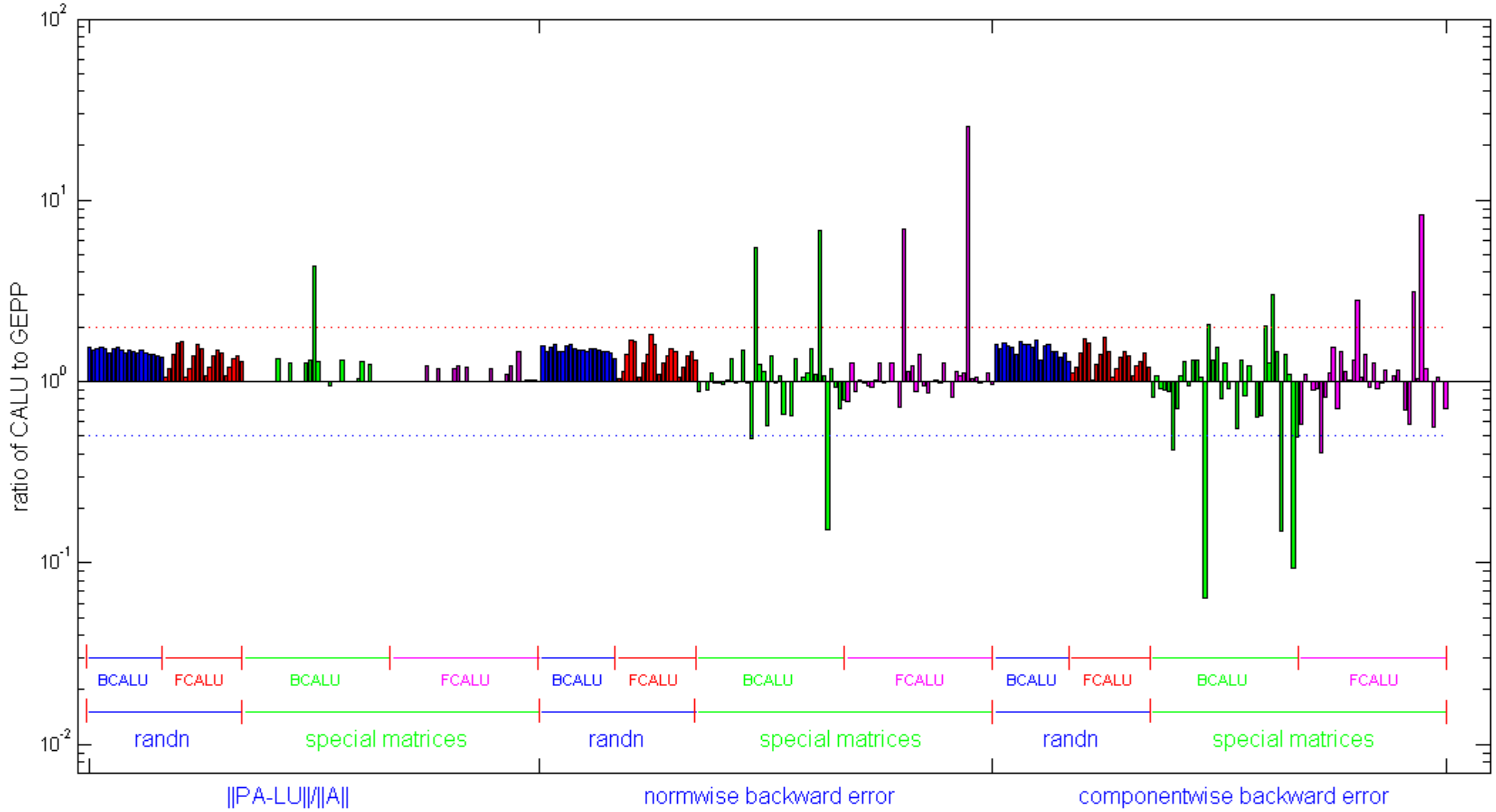
Growth factor for binary tree based CALU



- Random matrices from a normal distribution
- Same behaviour for all matrices in our test, and |L| <= 4.2

Stability of CALU (experimental results)

- Results show ||PA-LU||/||A||, normwise and componentwise backward errors, for random matrices and special ones
  - See [LG, Demmel, Xiang, SIMAX 2011] for details
  - BCALU denotes binary tree based CALU and FCALU denotes flat tree based CALU



Our "proof of stability" for CALU

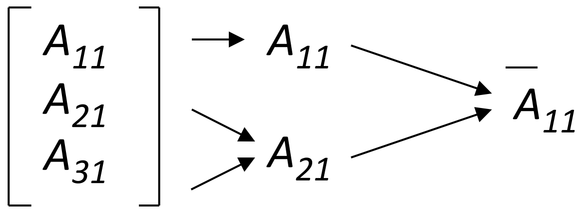
- CALU as stable as GEPP in following sense:

  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:

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

tournament pivoting:



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

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

Growth factor in exact arithmetic

- Matrix of size m-by-n, reduction tree of height H=log(P).
- (CA)LU_PRRP select pivots using strong rank revealing QR (A. Khabou, J. Demmel, LG, M. Gu, SIMAX 2013)
- "In practice" means observed/expected/conjectured values.

	CALU	GEPP	CALU_PRRP	LU_PRRP
Upper bound	$2^{n(\log(P)+1)-1}$	2^{n-1}	$(1+2b)^{(n/b)\log(P)}$	$(1+2b)^{(n/b)}$
In practice	$n^{2/3}$ -- $n^{1/2}$	$n^{2/3}$ -- $n^{1/2}$	$(n/b)^{2/3}$ -- $(n/b)^{1/2}$	$(n/b)^{2/3}$ -- $(n/b)^{1/2}$

$\longrightarrow$

Better bounds

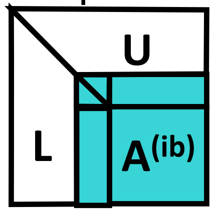
- For a matrix of size 10^7-by-10^7 (using petabytes of memory)

  $n^{1/2} = 10^{3.5}$

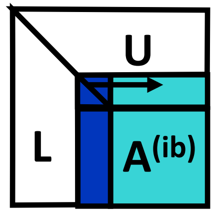
- When will Linpack have to use the QR factorization for solving linear systems ?

CALU – a communication avoiding LU factorization

- Consider a 2D grid of P processors P_r-by-P_c , using a 2D block cyclic layout with square blocks of size b.



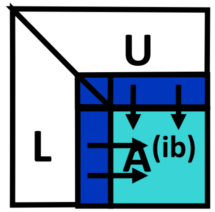
For ib = 1 to n-1 step b
    $A^{(ib)}$ = A(ib:n, ib:n)



(1) Find permutation for current panel using TSLU $O(n/b\log_2 P_r)$

(2) Apply all row permutations (pdlaswp) $O(n/b(\log_2 P_c + \log_2 P_r))$

    - broadcast pivot information along the rows of the grid

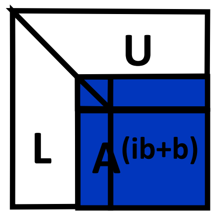


(3) Compute panel factorization (dtrsm)

$O(n/b\log_2 P_c)$

(4) Compute block row of U (pdtrsm)

    - broadcast right diagonal part of L of current panel



(5) Update trailing matrix (pdgemm) $O(n/b(\log_2 P_c + \log_2 P_r))$

    - broadcast right block column of L
    - broadcast down block row of U

LU for General Matrices

- Cost of CALU vs ScaLAPACK's PDGETRF
  - n x n matrix on $P^{1/2}$ x $P^{1/2}$ processor grid, block size b
  - Flops: $(2/3)n^3/P + (3/2)n^2b / P^{1/2}$ vs $(2/3)n^3/P + n^2b/P^{1/2}$
  - Bandwidth: $n^2 \log P/P^{1/2}$ vs same
  - Latency: $3 n \log P / \mathbf{b}$ vs $1.5 n \log P + 3.5 n \log P / b$

- Close to optimal (modulo log P factors)
  - Assume: $O(n^2/P)$ memory/processor, $O(n^3)$ algorithm,
  - Choose b near $n / P^{1/2}$ (its upper bound)
  - Bandwidth lower bound: $\Omega(n^2 /P^{1/2})$ – just log(P) smaller
  - Latency lower bound: $\Omega(P^{1/2})$ – just polylog(P) smaller
  - Extension of Irony/Toledo/Tishkin (2004)

Performance vs ScaLAPACK

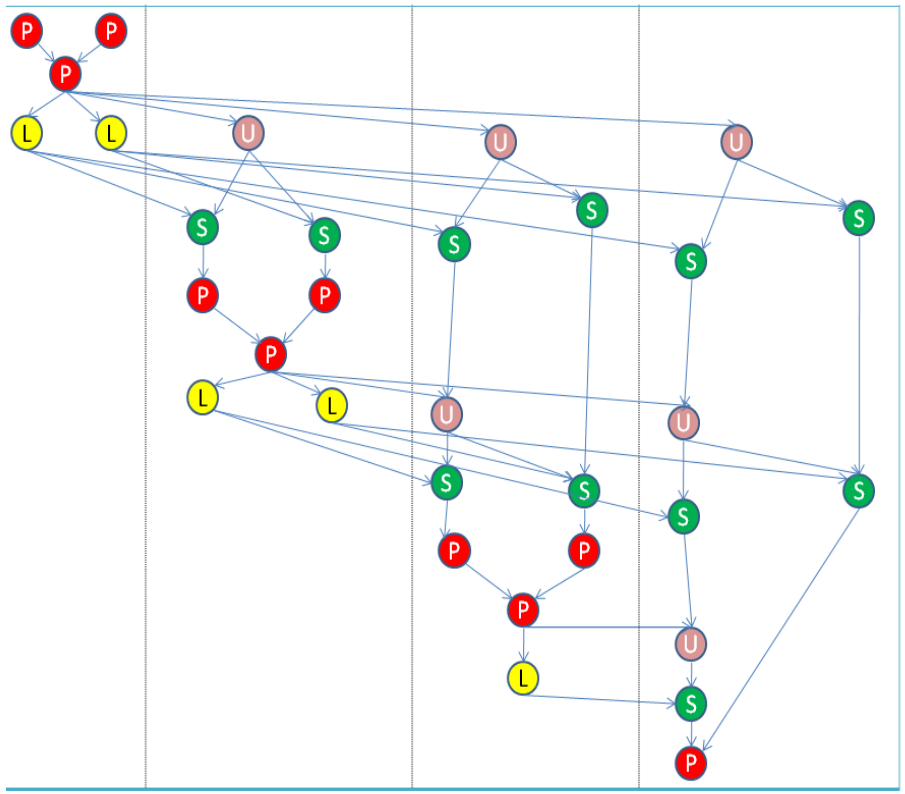
- Parallel TSLU (LU on tall-skinny matrix)
  - IBM Power 5
    - Up to **4.37x** faster (16 procs, 1M x 150)
  - Cray XT4
    - Up to **5.52x** faster (8 procs, 1M x 150)

- Parallel CALU (LU on general matrices)
  - Intel Xeon (two socket, quad core)
    - Up to **2.3x** faster (8 cores, 10^6 x 500)
  - IBM Power 5
    - Up to **2.29x** faster (64 procs, 1000 x 1000)
  - Cray XT4
    - Up to **1.81x** faster (64 procs, 1000 x 1000)

- Details in SC08 (LG, Demmel, Xiang), IPDPS'10 (S. Donfack, LG).

CALU and its task dependency graph

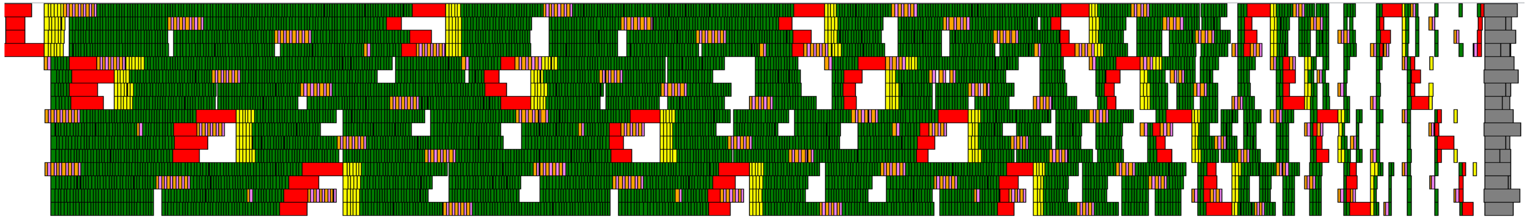
- The matrix is partitioned into blocks of size T x b.
- The computation of each block is associated with a task.



Scheduling CALU's Task Dependency Graph

- Static scheduling

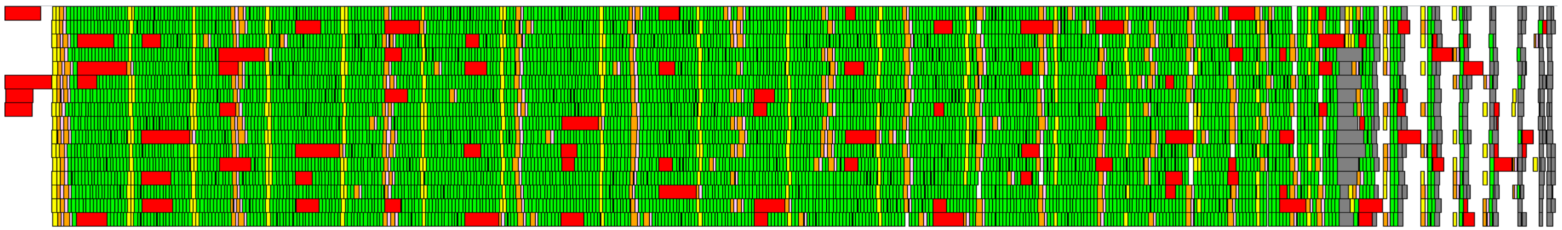
  \+ Good locality of data — - Ignores noise



- 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]

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

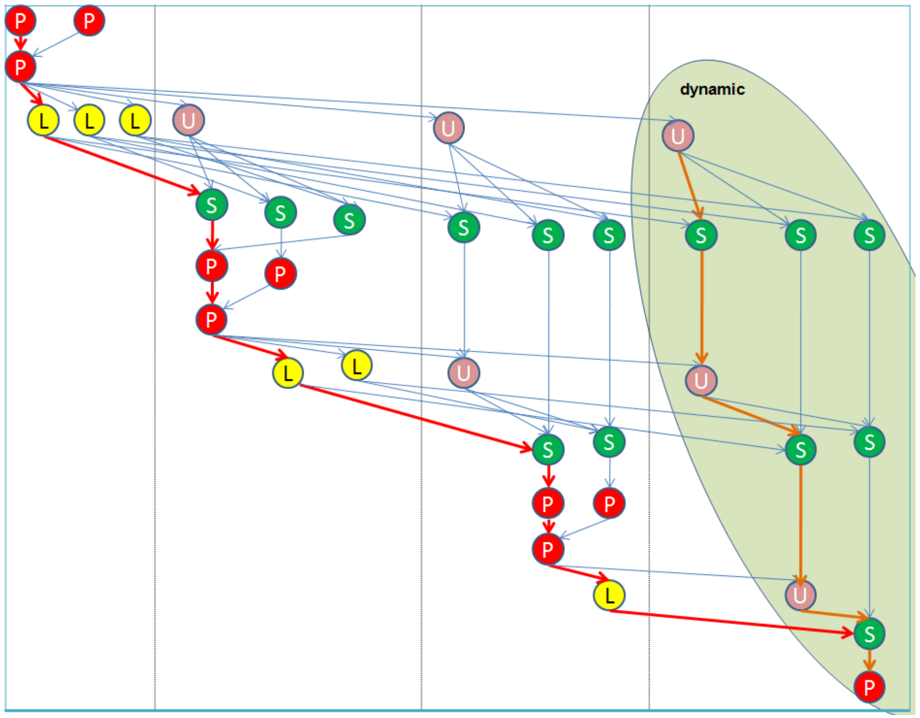
S. Donfack, LG, B. Gropp, V. Kale,IPDPS 2012

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:

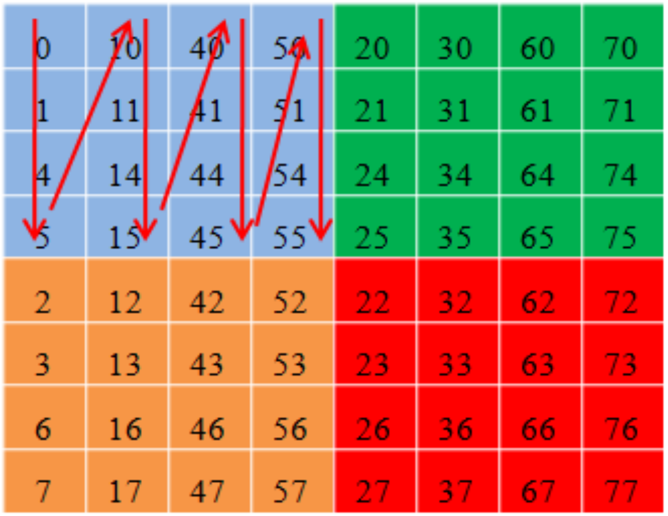
- A thread executes in priority its statically assigned tasks
- When no task ready, it picks a ready task from the dynamic part
- The size of the dynamic part is guided by a performance model



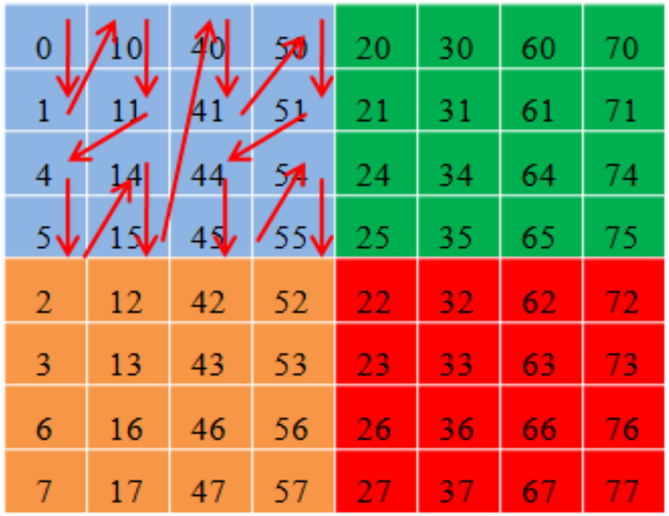
S. Donfack, LG, B. Gropp, V. Kale, 2012

Data layout and other optimizations

- Three data distributions investigated
  - CM : Column major order for the entire matrix
  - BCL : Each thread stores contiguously (CM) the data on which it operates
  - 2l-BL : Each thread stores in blocks the data on which it operates



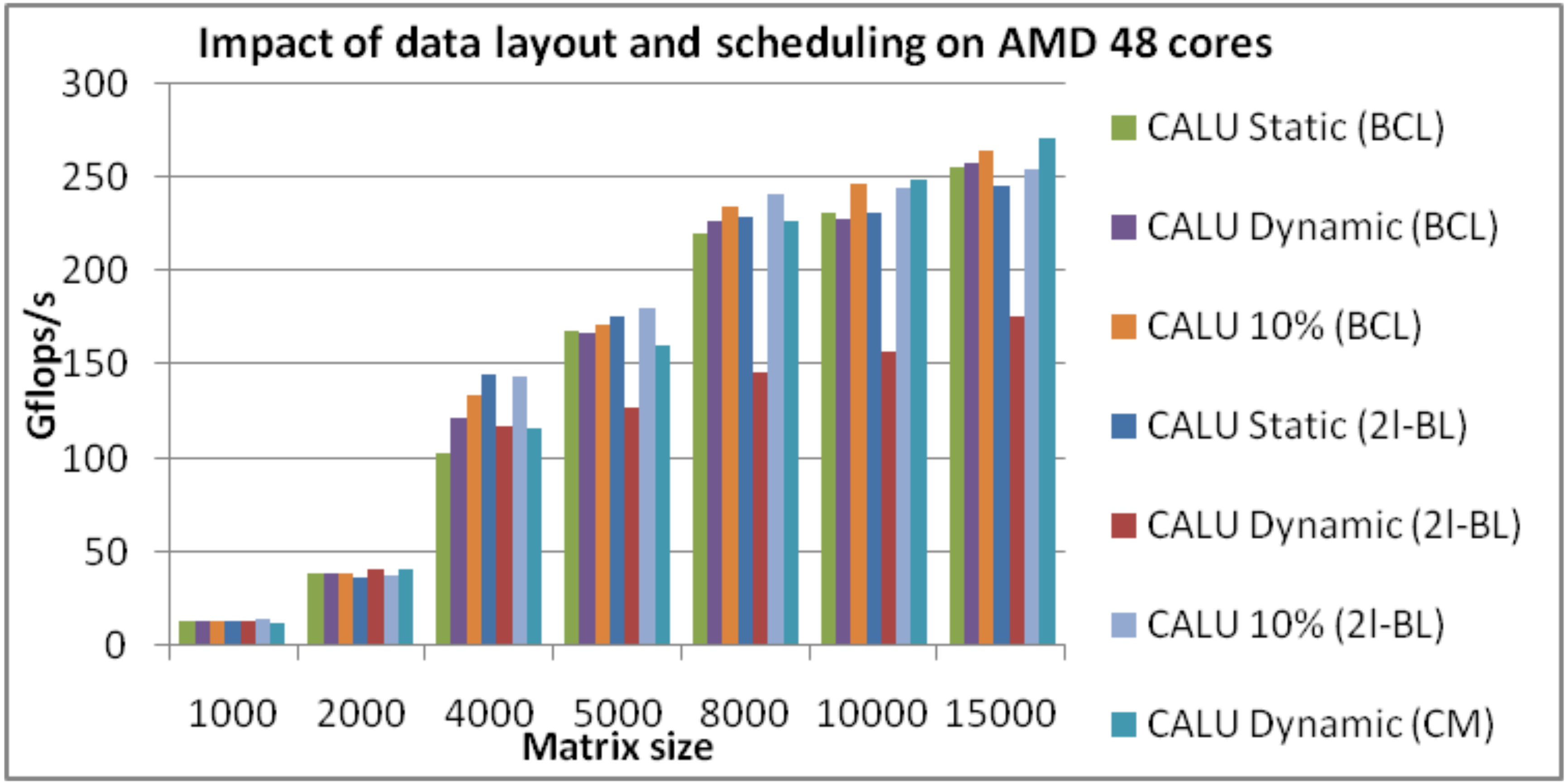
Block cyclic layout (BCL)



Two level block layout (2l-BL)

- And other optimizations
  - Updates (dgemm) performed on several blocks of columns (for BCL and CM layouts)

Impact of data layout

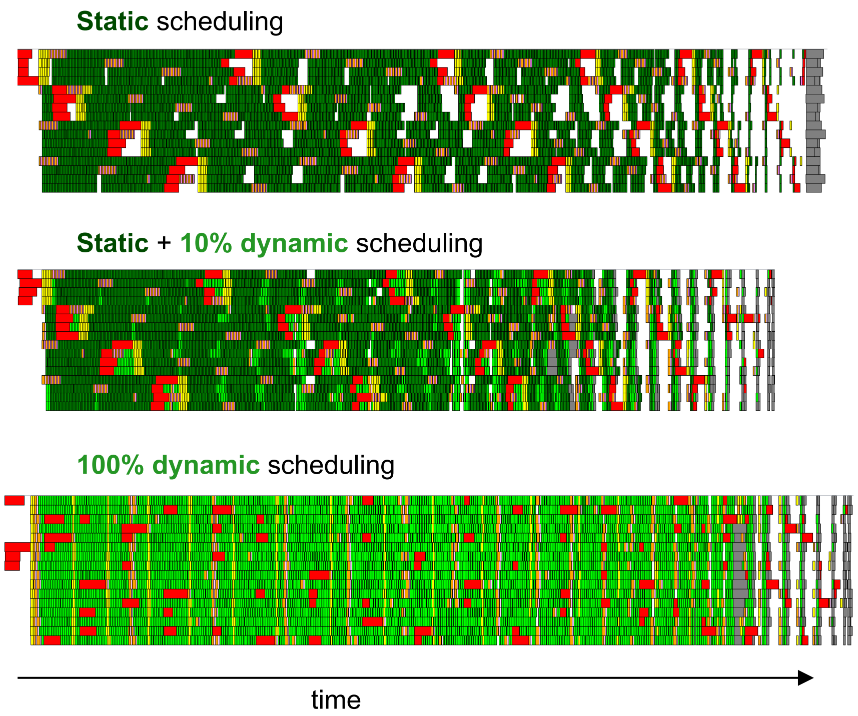


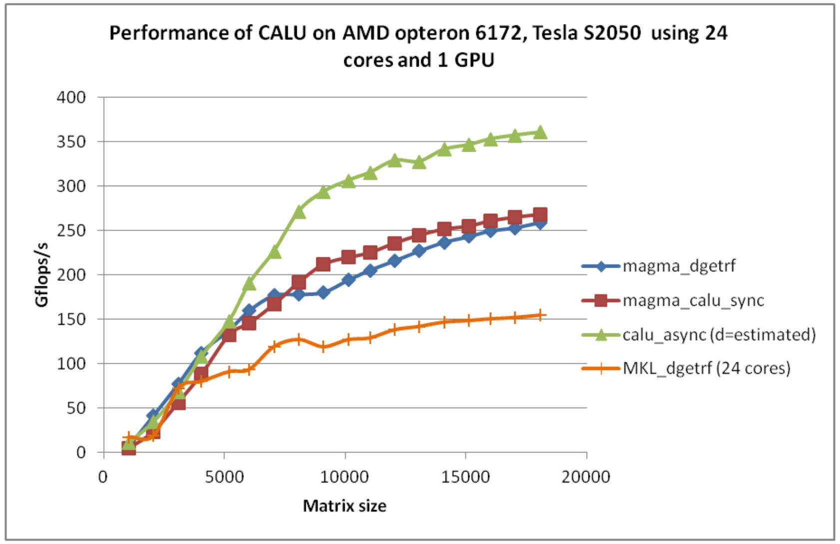
Eight socket, six core machine based on AMD Opteron processor (U. of Tennessee).

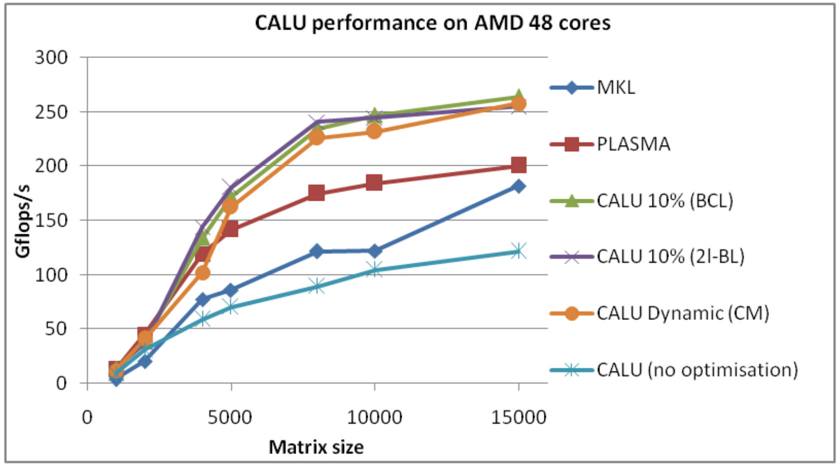
BCL : Each thread stores contiguously (CM) its data

2l-BL : Each thread stores in blocks its data

Best performance of CALU on multicore architectures







- Reported performance for PLASMA uses LU with block pairwise pivoting.
- GPU data courtesy of S. Donfack

Conclusions

- Introduced a new class of communication avoiding algorithms that minimize communication
  - Attain theoretical lower bounds on communication
  - Minimize communication at the cost of redundant computation
  - Are often faster than conventional algorithms in practice

- Remains a lot to do for sparse linear algebra
  - Communication bounds, communication optimal algorithms
  - Numerical stability of s-step methods
  - Alternatives as block iterative methods, pipelined iterative methods
  - Preconditioners - limited by memory and communication, not flops
- And BEYOND

Conclusions

- Many previous results
  - Only several cited, many references given in the papers
  - Flat trees algorithms for QR factorization, called tiled algorithms used in the context of
    - Out of core - Gunter, van de Geijn 2005
    - Multicore, Cell processors - Buttari, Langou, Kurzak and Dongarra (2007, 2008), Quintana-Orti, Quintana-Orti, Chan, van Zee, van de Geijn (2007, 2008)

Collaborators, funding

Collaborators:

- A. Branescu, Inria, S. Donfack, Inria, A. Khabou, Inria, M. Jacquelin, Inria, S. Moufawad, Inria, F. Nataf, CNRS, H. Xiang, University Paris 6, S. Yousef, Inria
- J. Demmel, UC Berkeley, B. Gropp, UIUC, M. Gu, UC Berkeley, M. Hoemmen, UC Berkeley, J. Langou, CU Denver, V. Kale, UIUC

Funding: ANR Petal and Petalh projects, ANR Midas, Digiteo Xscale NL, COALA INRIA funding

Further information:

http://www-rocq.inria.fr/who/Laura.Grigori/

References

Results presented from:

- J. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, UCB-EECS-2008-89, 2008, SIAM journal on Scientific Computing, Vol. 34, No 1, 2012.
- L. Grigori, J. Demmel, and H. Xiang, *Communication avoiding Gaussian elimination*, Proceedings of the IEEE/ACM SuperComputing SC08 Conference, November 2008.
- L. Grigori, J. Demmel, and H. Xiang, *CALU: a communication optimal LU factorization algorithm*, SIAM. J. Matrix Anal. & Appl., 32, pp. 1317-1350, 2011.
- M. Hoemmen's Phd thesis, *Communication avoiding Krylov subspace methods*, 2010.
- L. Grigori, P.-Y. David, J. Demmel, and S. Peyronnet, *Brief announcement: Lower bounds on communication for sparse Cholesky factorization of a model problem*, ACM SPAA 2010.
- S. Donfack, L. Grigori, and A. Kumar Gupta, *Adapting communication-avoiding LU and QR factorizations to multicore architectures*, Proceedings of IEEE International Parallel & Distributed Processing Symposium IPDPS, April 2010.
- S. Donfack, L. Grigori, W. Gropp, and V. Kale, *Hybrid static/dynamic scheduling for already optimized dense matrix factorization* , Proceedings of IEEE International Parallel & Distributed Processing Symposium IPDPS, 2012.
- A. Khabou, J. Demmel, L. Grigori, and M. Gu, *LU factorization with panel rank revealing pivoting and its communication avoiding version*, LAWN 263, SIAM Journal on Matrix Analysis, in revision, 2012.
- L. Grigori, S. Moufawad, *Communication avoiding ILU0 preconditioner*, Inria TR 8266, 2013.
- J. Demmel, L. Grigori, M. Gu, H. Xiang, Communication avoiding rank revealing QR factorization with column pivoting, 2013.

EXTRA SLIDES

EXTRA SLIDES:

RRQR

Rank revealing factorizations

- A rank revealing QR (RRQR) factorization is given as

$$A\Pi = QR = Q\begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}, \; R_{11} \text{ is } k-by-k$$

$$\text{with } \sigma_{\min}(R_{11}) \geq \frac{\sigma_k(A)}{p(k,n)}, \quad \sigma_{\max}(R_{22}) \leq \sigma_{k+1}(A) \cdot \; p(k,n),$$

$p(k,n)$ is a low degree polynomial in n and k, R_{11} is well conditioned, $\|R_{22}\|_2$ is small

strong RRQR if $\left|R_{11}^{-1}R_{12}\right| \leq c$

- Since $\sigma_{k+1}(A) \leq \sigma_{\max}\left(R_{22}\right) = \|R_{22}\|_2$, the numerical rank of *A* is *k*
- *Q(:,1:k)* forms an approximate orthogonal basis for the range of *A*
- $A\Pi\begin{pmatrix} R_{11}^{-1}R_{12} \\ -I \end{pmatrix} = Q\begin{pmatrix} 0 \\ -R_{22} \end{pmatrix}$, then $\Pi\begin{pmatrix} R_{11}^{-1}R_{12} \\ -I \end{pmatrix}$ are approximate null vectors
- Applications: subset selection and linear dependency analysis, rank determination, low rank approximation - solve $min_{rank(X)=k} \|A-X\|$

Reconstruct Householder vectors from TSQR

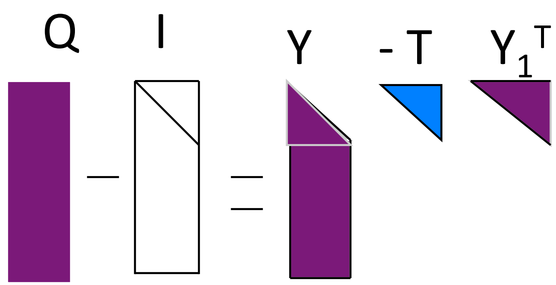
The QR factorization using Householder vectors

$$A = QR = (I - YTY^T)R$$

can be re-written as an LU factorization

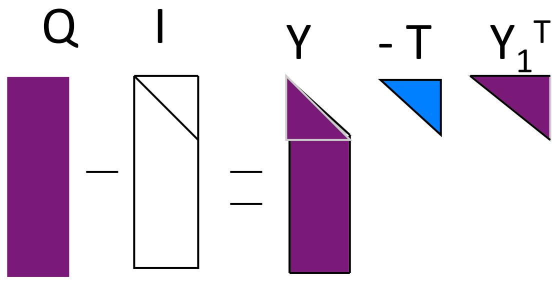
$$A - R = Y(-TY_1^T)R$$

$$Q - I = Y(-TY_1^T)$$



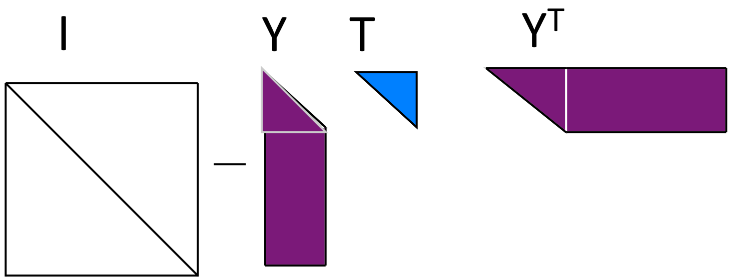
Reconstruct Householder vectors TSQR-HR

1. Perform TSQR
2. Form Q explicitly (tall-skinny orthonormal factor)
3. Perform LU decomposition: $Q - I = LU$

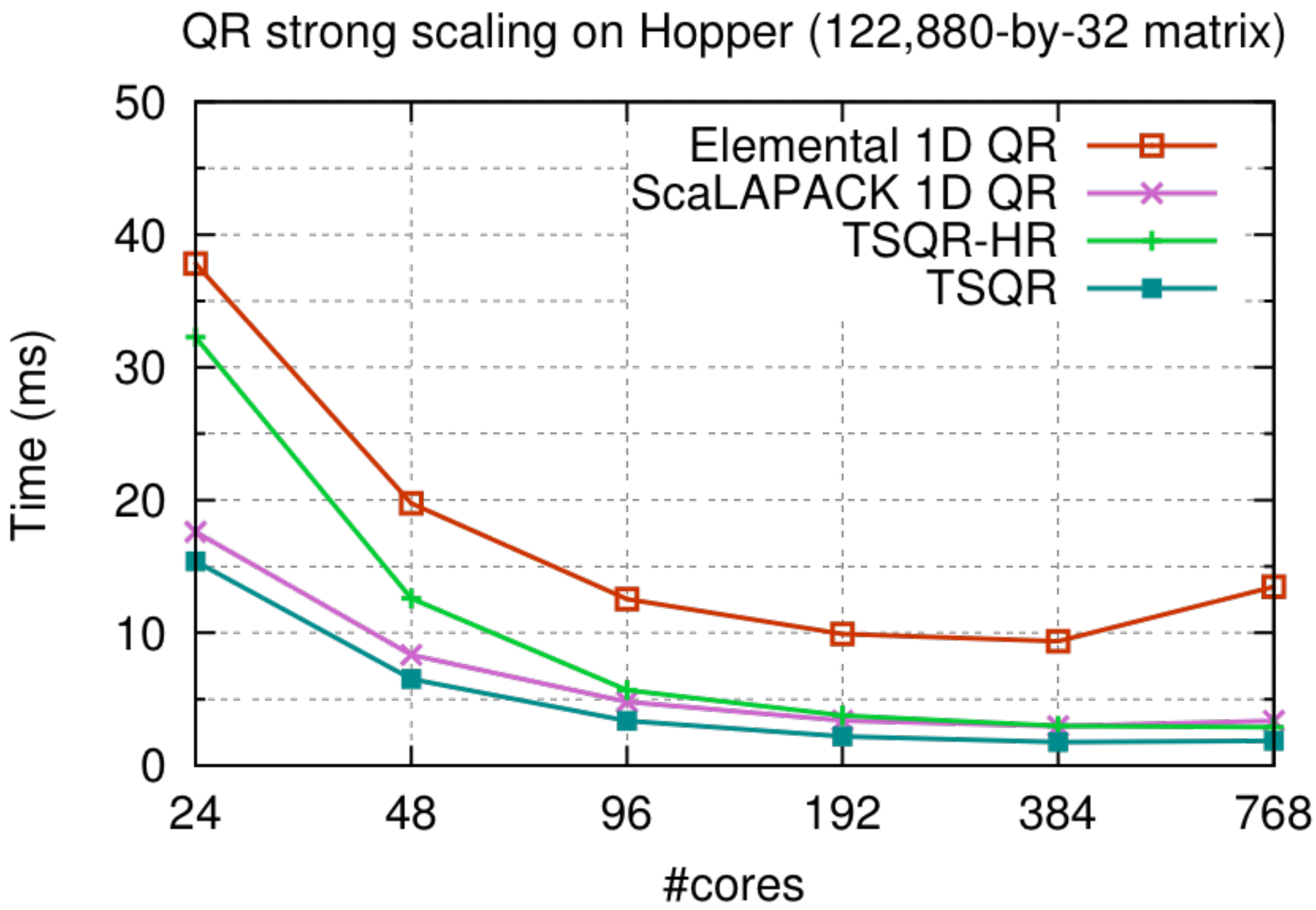


4. Set $Y = L$
5. Set $T = -U\, Y_1^{-T}$

$$I - YTY^T = I - \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} T \begin{bmatrix} Y_1^T & Y_2^T \end{bmatrix}$$

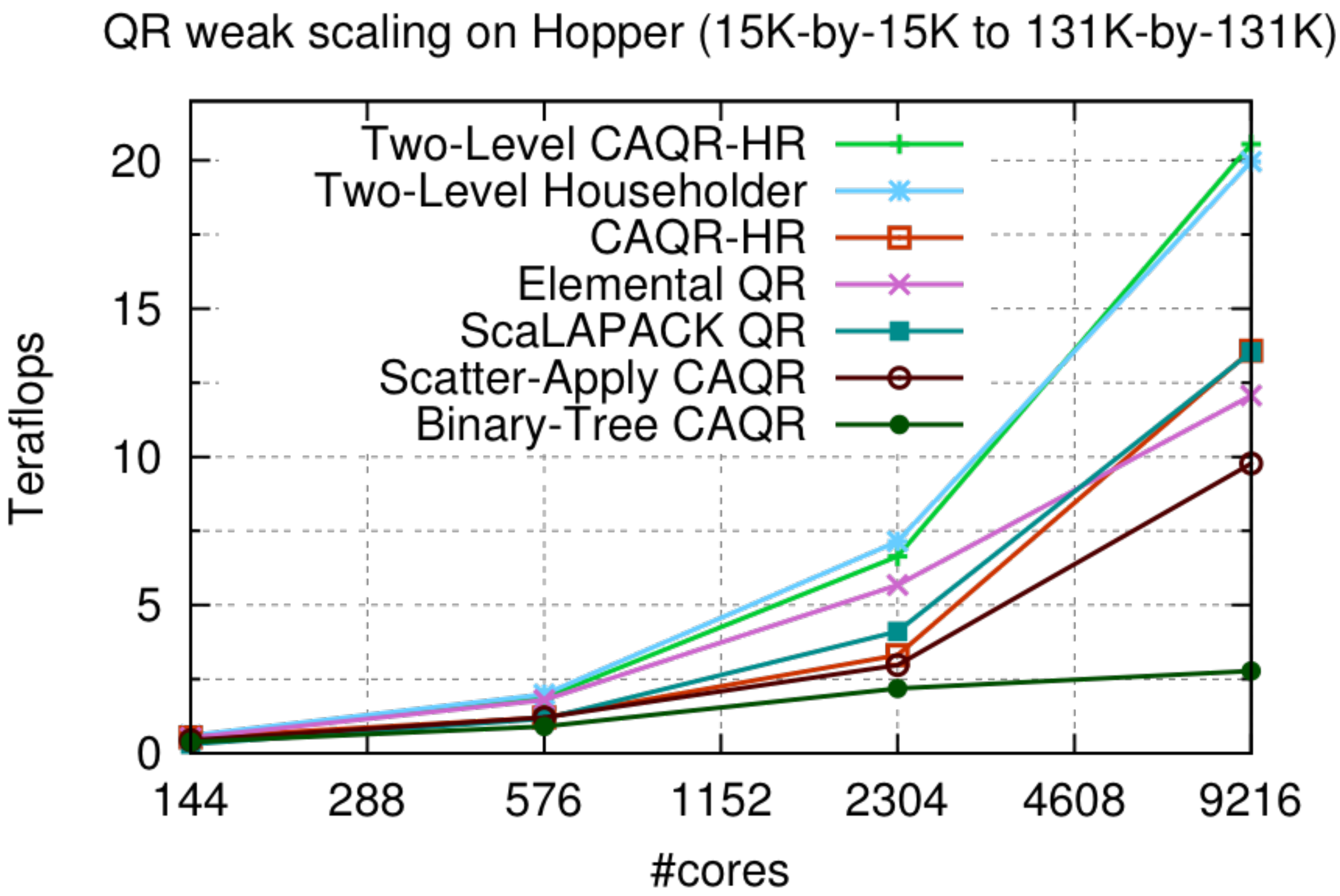


Strong scaling QR on Hopper



- Matrix of size 122,880-by-32
- Hopper: Cray XE6 supercomputer (NERSC) – dual socket 12-core Magny-Cours Opteron (2.1 GHz)

Weak scaling QR on Hopper

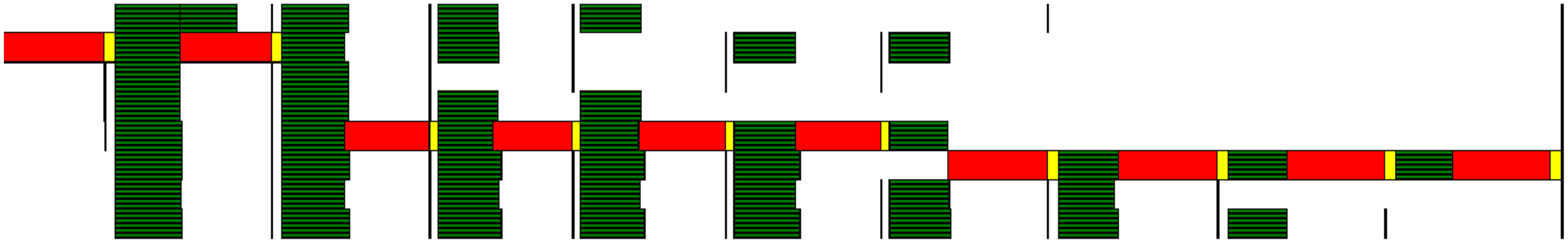


- Matrix of size 15K-by-15K to 131K-by-131K
- Hopper: Cray XE6 supercomputer (NERSC) – dual socket 12-core Magny-Cours Opteron (2.1 GHz)

CA(LU/QR) on multicore architectures

The panel factorization stays on the critical path, but it is much faster. Example of execution on Intel 8 cores machine for a matrix of size 10^5 x 1000, with block size b = 100.

CALU: one thread computes the panel factorizaton



CALU: 8 threads compute the panel factorizaton