Communication avoiding algorithms for LU and QR factorizations

Laura Grigori

Alpines

INRIA Paris - LJLL, Sorbonne Universite

February 2018

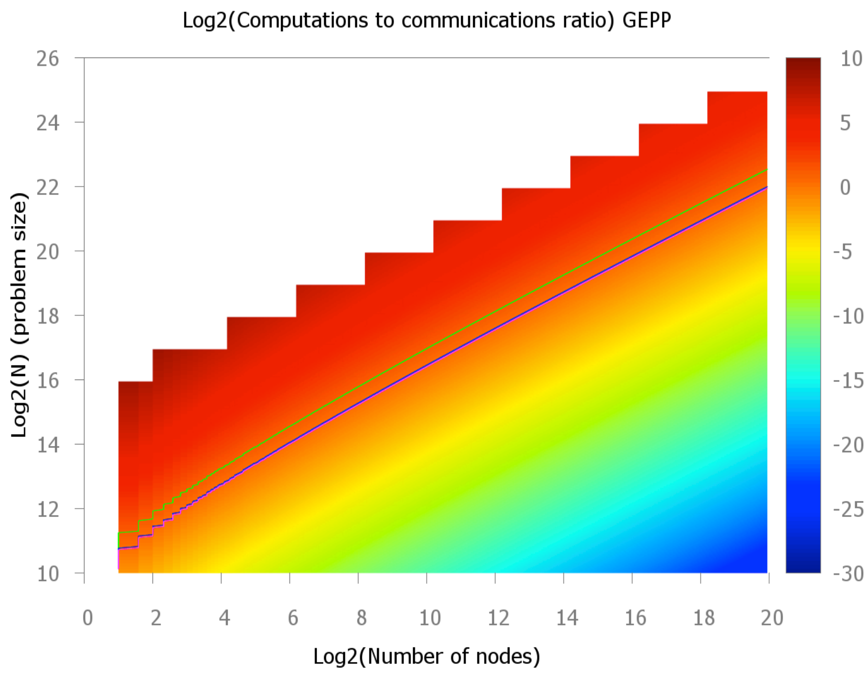
Plan

- Motivation
- Communication complexity of linear algebra operations
- Communication avoiding for dense linear algebra
  - LU, QR, Rank Revealing QR factorizations
  - Progressively implemented in ScaLAPACK, LAPACK
  - Algorithms for multicore processors
- Conclusions

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

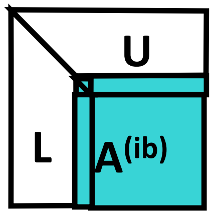


- Same algebraic framework, different numerical algorithm
  - The approach used in CA algorithms
  - More opportunities for reducing communication, may affect stability

Evolution of numerical libraries

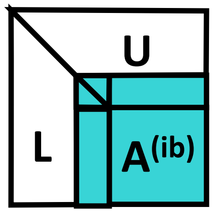
LINPACK (70's)

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



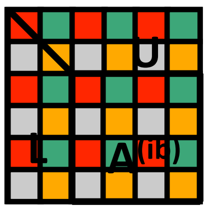
LAPACK (80's)

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



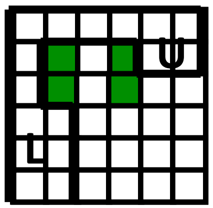
ScaLAPACK (90's)

- 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

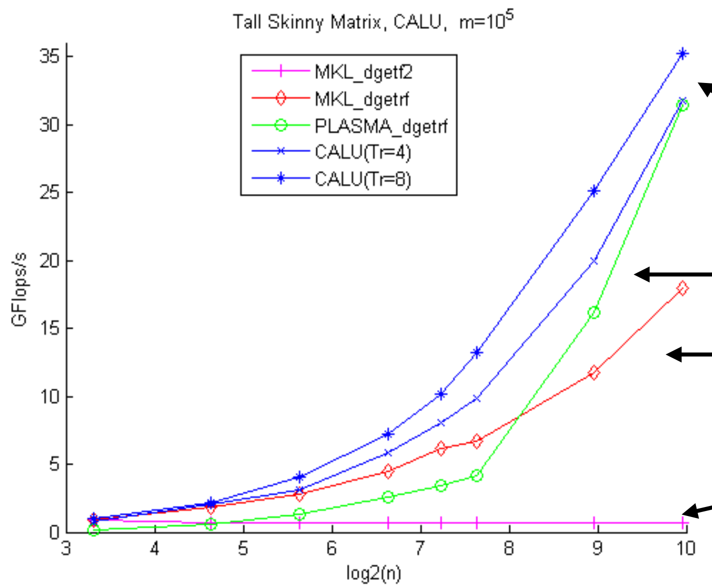


Project developed by U Tennessee Knoxville, UC Berkeley, other collaborators.

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

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]

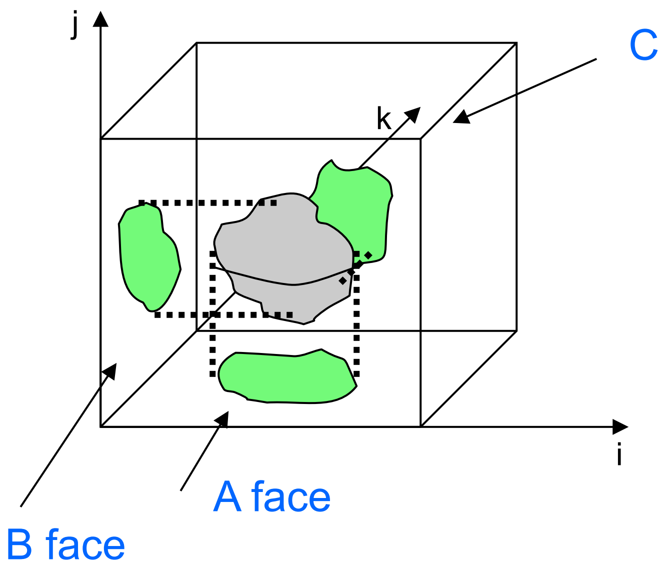
Lower bounds for linear algebra

- Computation modelled as an n-by-n-by-n set of lattice points
  (i,j,k) represents the operation *c(i,j) += f_{ij}(g_{ijk} (a(i,k)*b(k,j))))*
- The computation is divided in S phases
- Each phase contains exactly M (the fast memory size) load and store instructions
- Determine how many flops the algorithm can compute in each phase, by applying discrete Loomis-Whitney inequality:

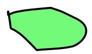
$$w^2 \le N_A N_B N_C$$

Algorithms in direct linear algebra:

$$
\begin{aligned}
&\mathit{for}\ i,j,k = 1:n \\
&\quad c(i,j) = f_{ij}(g_{ijk}(a(i,k),b(k,j))) \\
&\mathit{endfor}
\end{aligned}
$$



 - set of points in R^3, represent w arithmetics

 - orthogonal projections of the points onto coordinate planes N_A, N_B, N_C represent values of A, B, C

Lower bounds for matrix multiplication (contd)

- Discrete Loomis-Whitney inequality:

$$w^2 \leq N_A N_B N_C$$

- Since there are at most 2M elements of A, B, C in a phase, the bound is:

$$w \leq 2\sqrt{2} M^{3/2}$$

- The number of phases S is #flops/w, and hence the lower bound on communication is:

$$\#\, messages(S) \geq \frac{\#\, flops}{w} = \Omega\left(\frac{\#\, flops}{M^{3/2}}\right)$$

$$\#\, loads/stores \geq \Omega\left(\frac{\#\, flops}{M^{1/2}}\right)$$

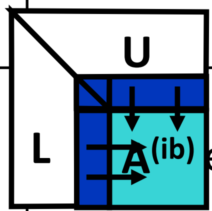
Sequential algorithms and communication bounds

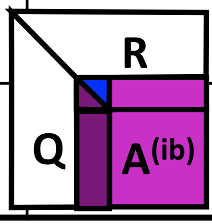
Algorithm	Minimizing #words (not #messages)	Minimizing #words and #messages
Cholesky	LAPACK	[Gustavson, 97] [Ahmed, Pingali, 00]
LU	LAPACK (few cases) [Toledo,97], [Gustavson, 97] both use partial pivoting	[LG, Demmel, Xiang, 08] [Khabou, Demmel, LG, Gu, 12] uses tournament pivoting
QR	LAPACK (few cases) [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 es 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

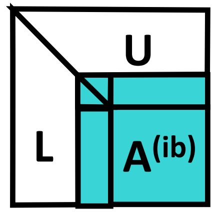
LU factorization (as in ScaLAPACK pdgetrf)

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

For ib = 1 to n-1 step b

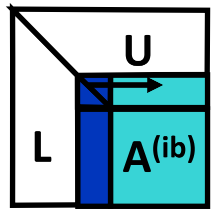
$A^{(ib)} = A(ib:n, ib:n)$

#messages



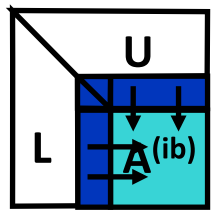
(1) Compute panel factorization $O(n \log_2 P_r)$

- 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

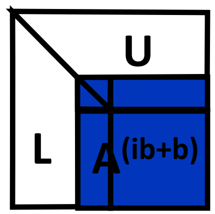


(3) Compute block row of U $O(n/b \log_2 P_c)$

- 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

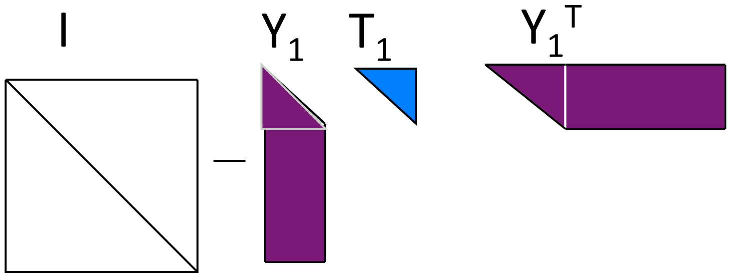


Block QR factorization

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

Block QR algebra:

1. Compute panel factorization:
$$\begin{pmatrix} \mathrm{A}_{11} \\ \mathrm{A}_{12} \end{pmatrix} = \mathrm{Q}_1 \begin{pmatrix} R_{11} \\ \\ \end{pmatrix}, \quad Q_1 = H_1 H_2 \ldots H_b$$
2. Compute the compact representation:
$$\mathrm{Q}_1 = I - Y_1 T_1 Y_1^T$$



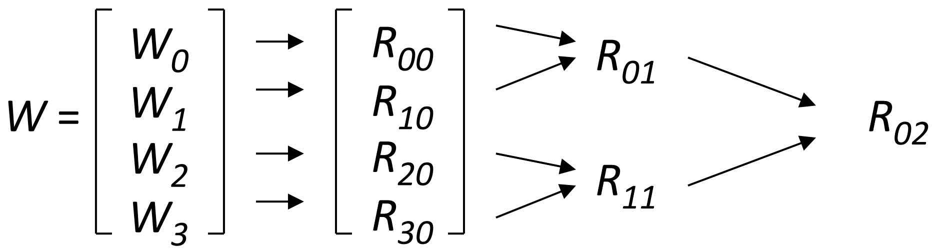
3. Update the trailing matrix:
$$\left(I - Y_1 T_1^T Y_1^T\right)\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - Y_1\left(T_1^T\left(Y_1^T\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}\right)\right) = \begin{pmatrix} R_{12} \\ A_{22}^1 \end{pmatrix}$$
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

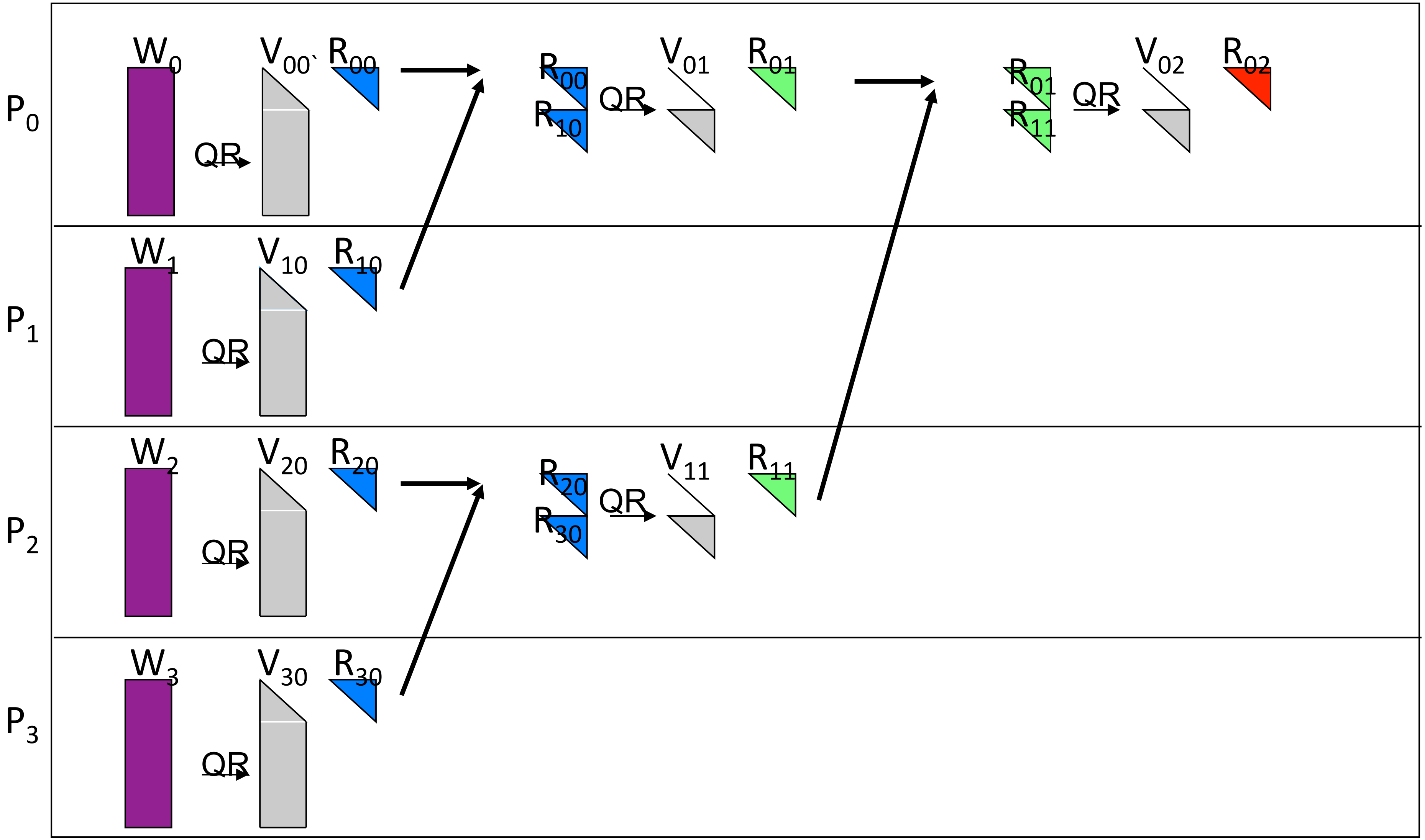
- Classic Parallel Algorithm
  - Compute Householder vector for each column
  - Number of messages ∝ b log P

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



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

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

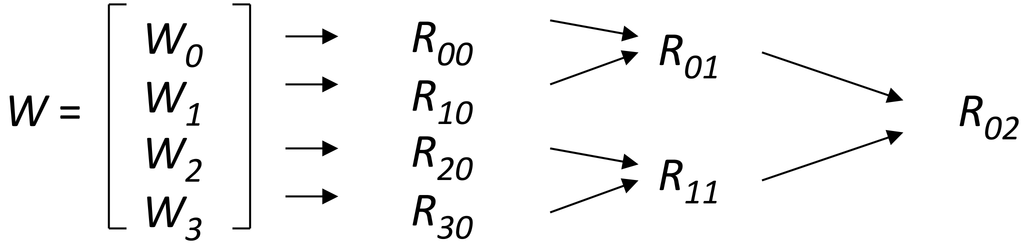
$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}R_{01} \\ \hline Q_{11}R_{11} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ \hline & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} \qquad \begin{pmatrix} R_{01} \\ \hline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

Q is represented implicitly as a product

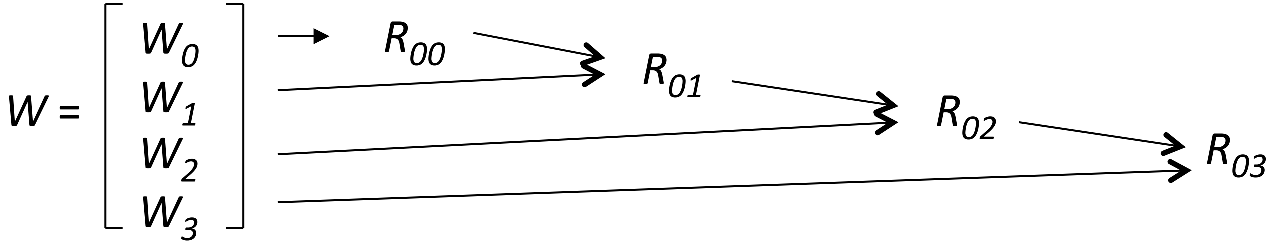
Output: $\{Q_{00}, Q_{10}, Q_{00}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02}\}$

Flexibility of TSQR and CAQR algorithms

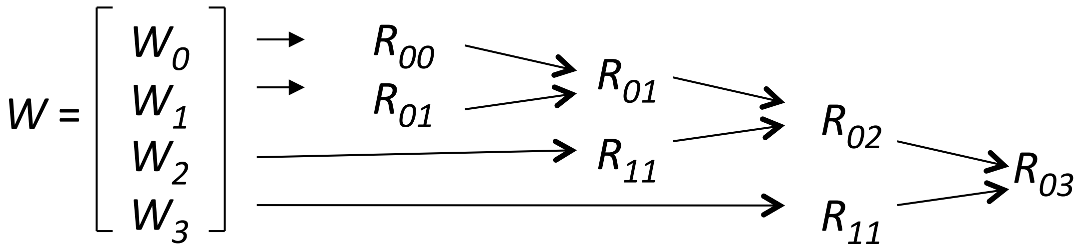
Parallel:



Sequential:



Dual Core:



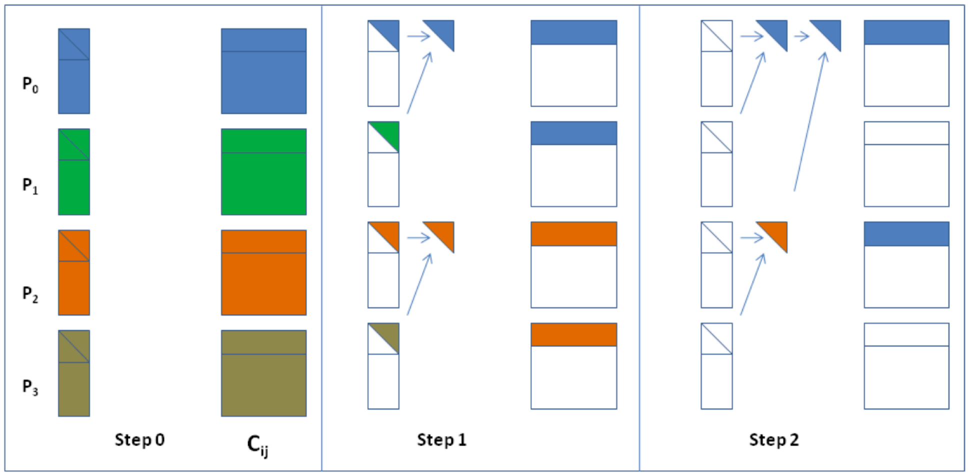
Reduction tree will depend on the underlying architecture, could be chosen dynamically

Algebra of TSQR

Parallel:

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

CAQR



QR for General Matrices

- 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: $2.5\ n \log P$ / **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:

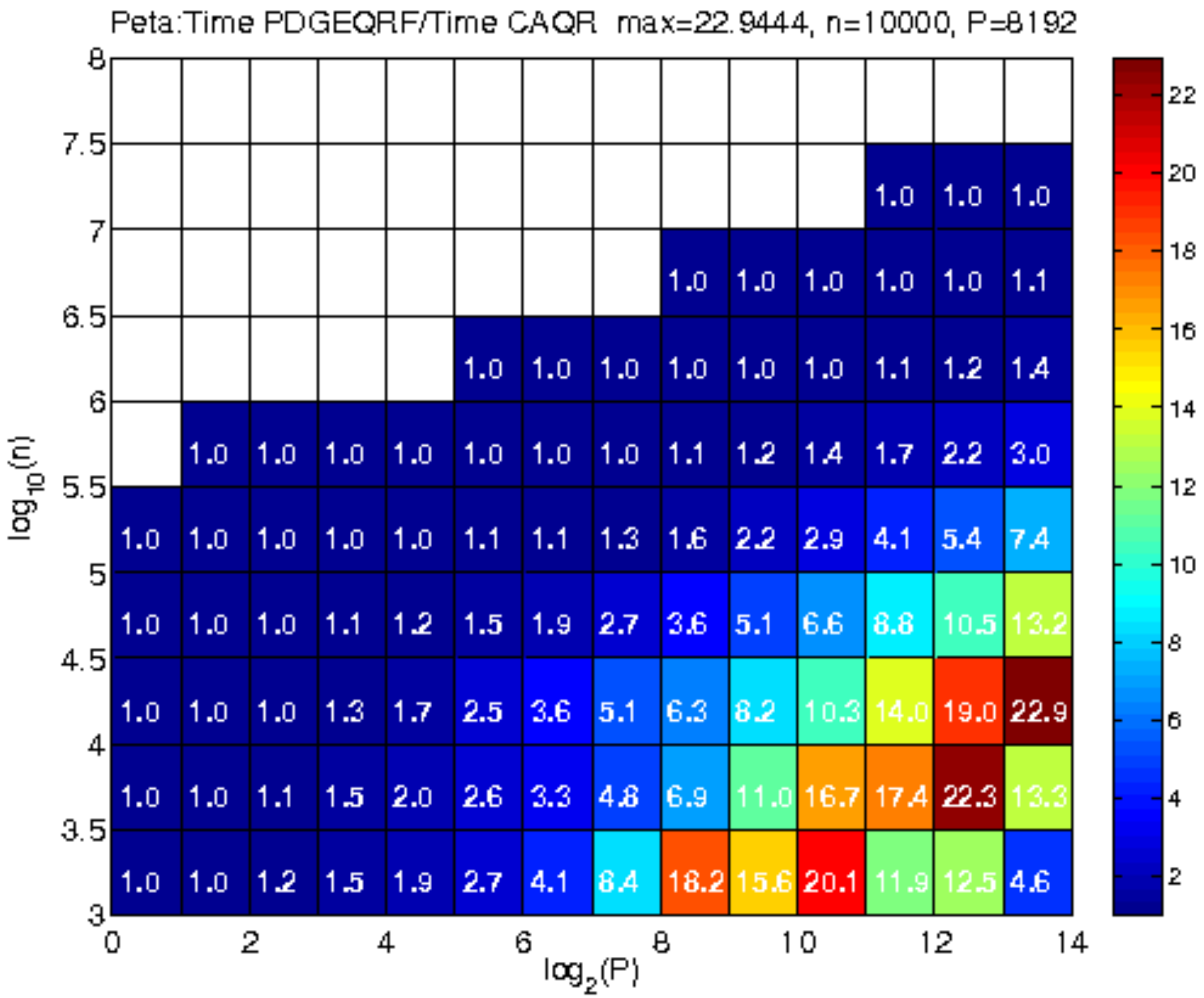
    $\Omega(P^{1/2})$ – just polylog(P) smaller

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.

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

Impact

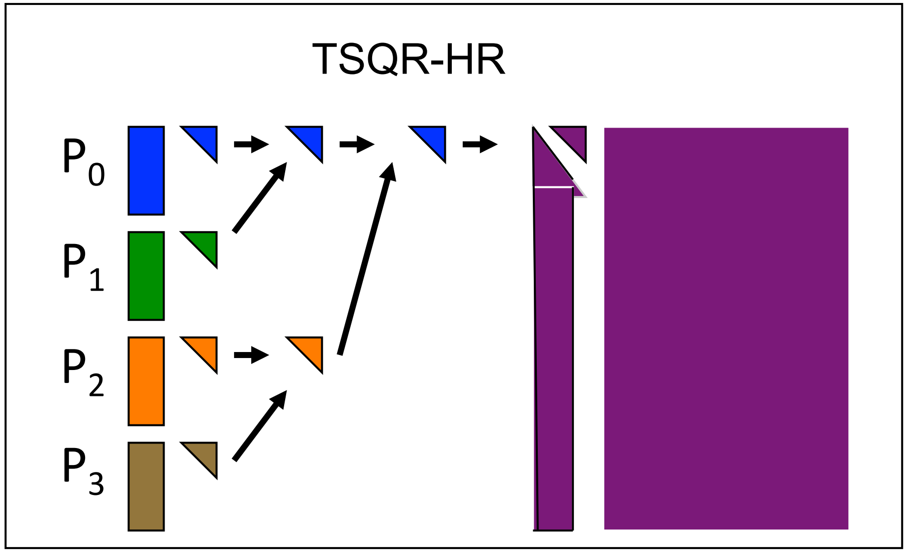
- TSQR/CAQR implemented in
  - Intel Data analytics library
  - GNU Scientific Library
  - ScaLAPACK
  - Spark for data mining

- CALU implemented in
  - Cray's libsci
  - To be implemented in lapack/scapalack

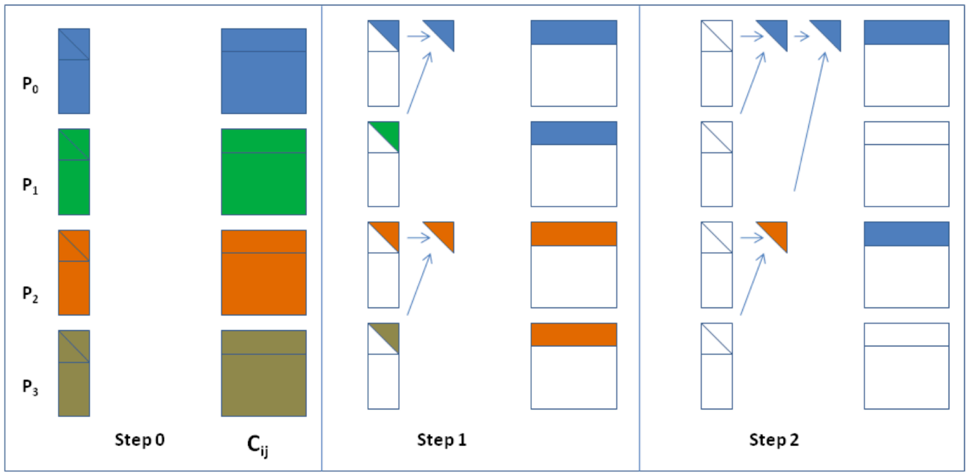
Algebra of TSQR

Parallel:

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



CAQR



Reconstruct Householder vectors from TSQR

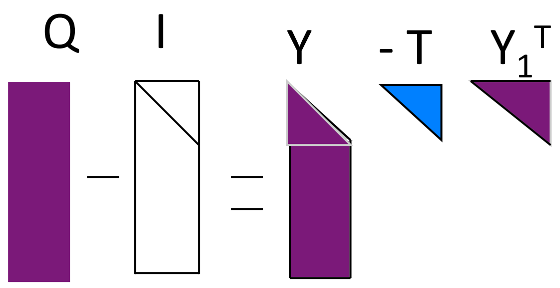
The QR factorization using Householder vectors

$$W = QR = (I - YTY_1^T)R$$

can be re-written as an LU factorization

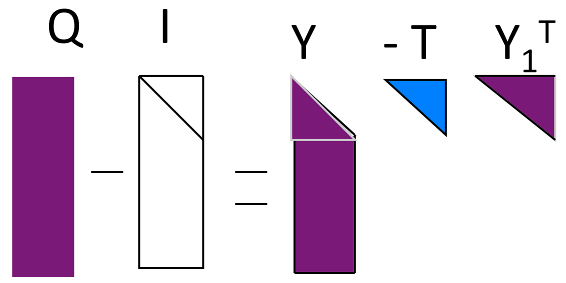
$$W - R = Y(-TY_1^T)R$$

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



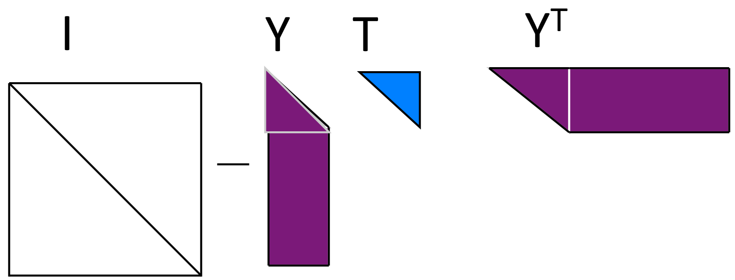
Reconstruct Householder vectors TSQR-HR

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

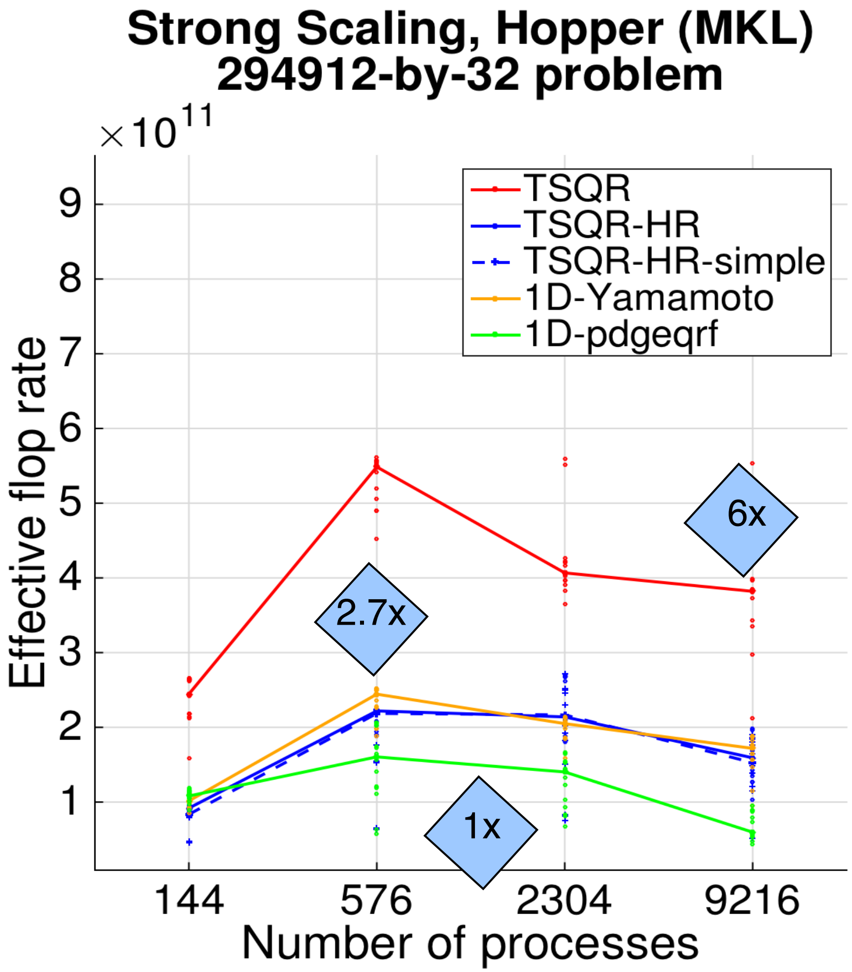


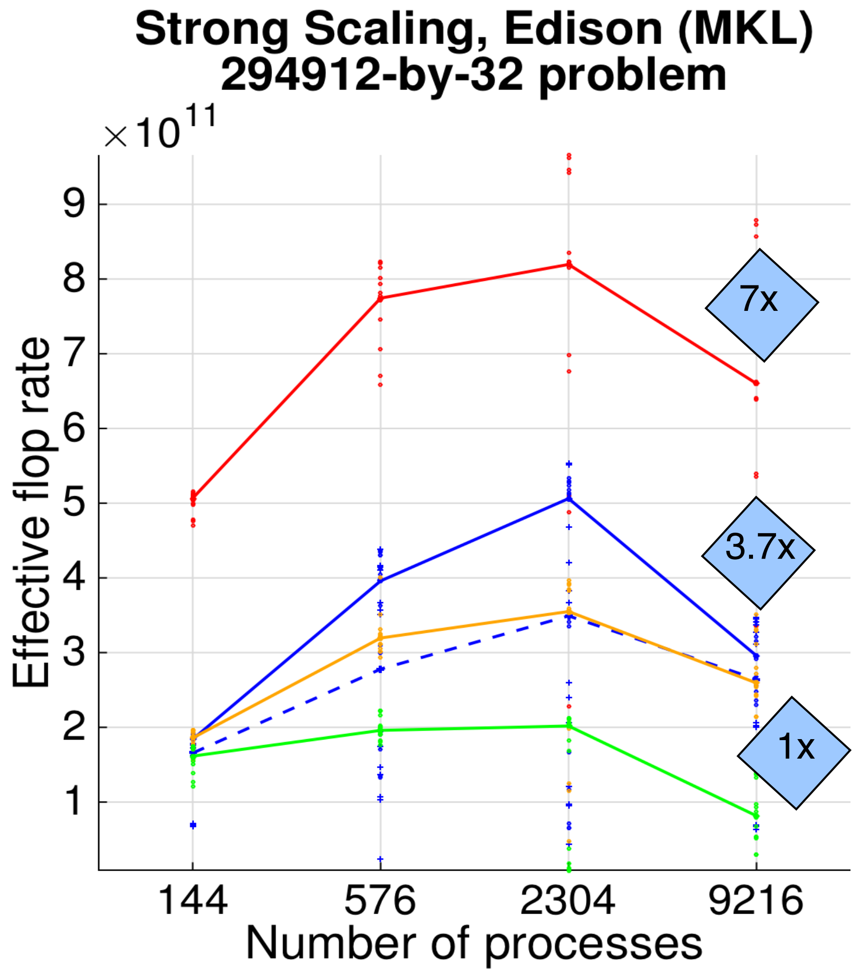
4. Set $Y = L$
5. Set $T = -U\, Y_1^{-T}$

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



Strong scaling





- Hopper: Cray XE6 (NERSC) – 2 x 12-core AMD Magny-Cours (2.1 GHz)
- Edison: Cray CX30 (NERSC) – 2 x 12-core Intel Ivy Bridge (2.4 GHz)
- Effective flop rate, computed by dividing $2mn^2 - 2n^3/3$ by measured runtime

Ballard, Demmel, LG, Jacquelin, Knight, Nguyen, and Solomonik, 2015.

The LU factorization of a tall skinny matrix

First try the obvious generalization of TSQR.

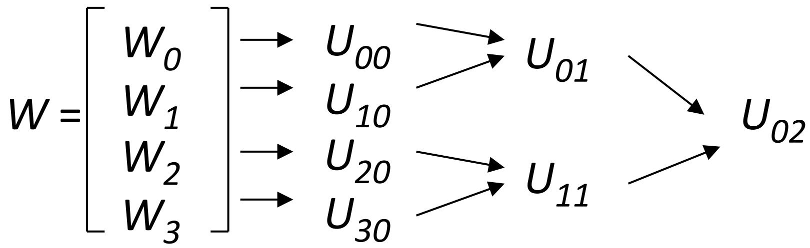
$$W = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \underbrace{\begin{pmatrix} \Pi_{00} & & & \\ & \Pi_{10} & & \\ & & \Pi_{20} & \\ & & & \Pi_{30} \end{pmatrix}}_{\Pi_0} \cdot \begin{pmatrix} L_{00} & & & \\ & L_{10} & & \\ & & L_{20} & \\ & & & L_{30} \end{pmatrix} \cdot \begin{pmatrix}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

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

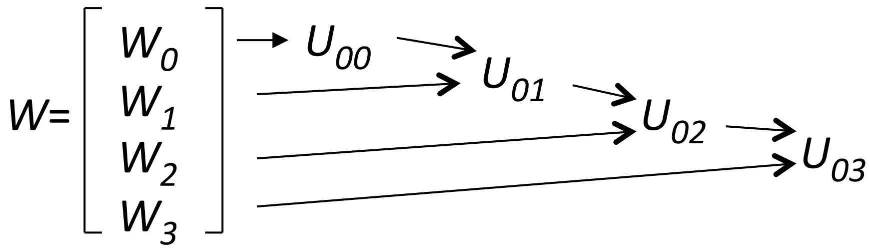
$$\begin{pmatrix} U_{01} \\ U_{11} \end{pmatrix} = \underbrace{\Pi_{02}}_{\Pi_2} L_{02} U_{02}$$

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

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

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.

- $g_W \leq 2^{n-1}$, attained for Wilkinson matrix

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

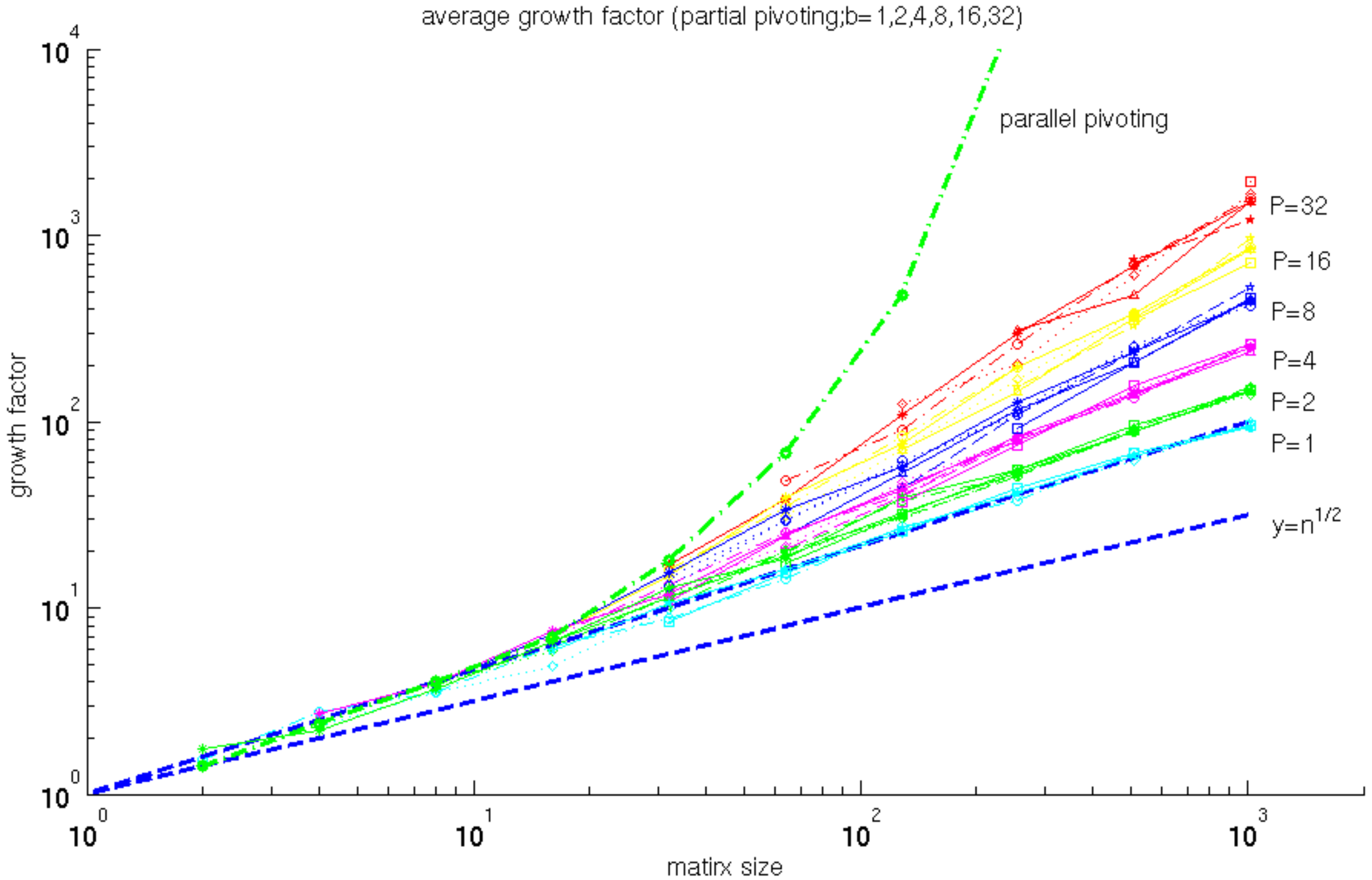
$$A = diag(\pm 1)\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & & \cdots & 0 & 1 \\ -1 & -1 & 1 & \ddots & 0 & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ -1 & -1 & \cdots & -1 & 1 & 1 \\ -1 & -1 & \cdots & -1 & -1 & 1 \end{pmatrix}$$

- 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.

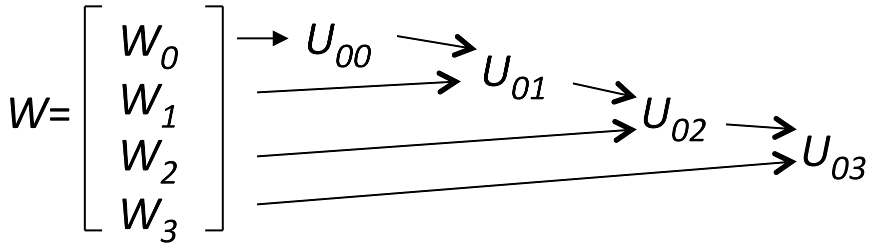
Block parallel pivoting

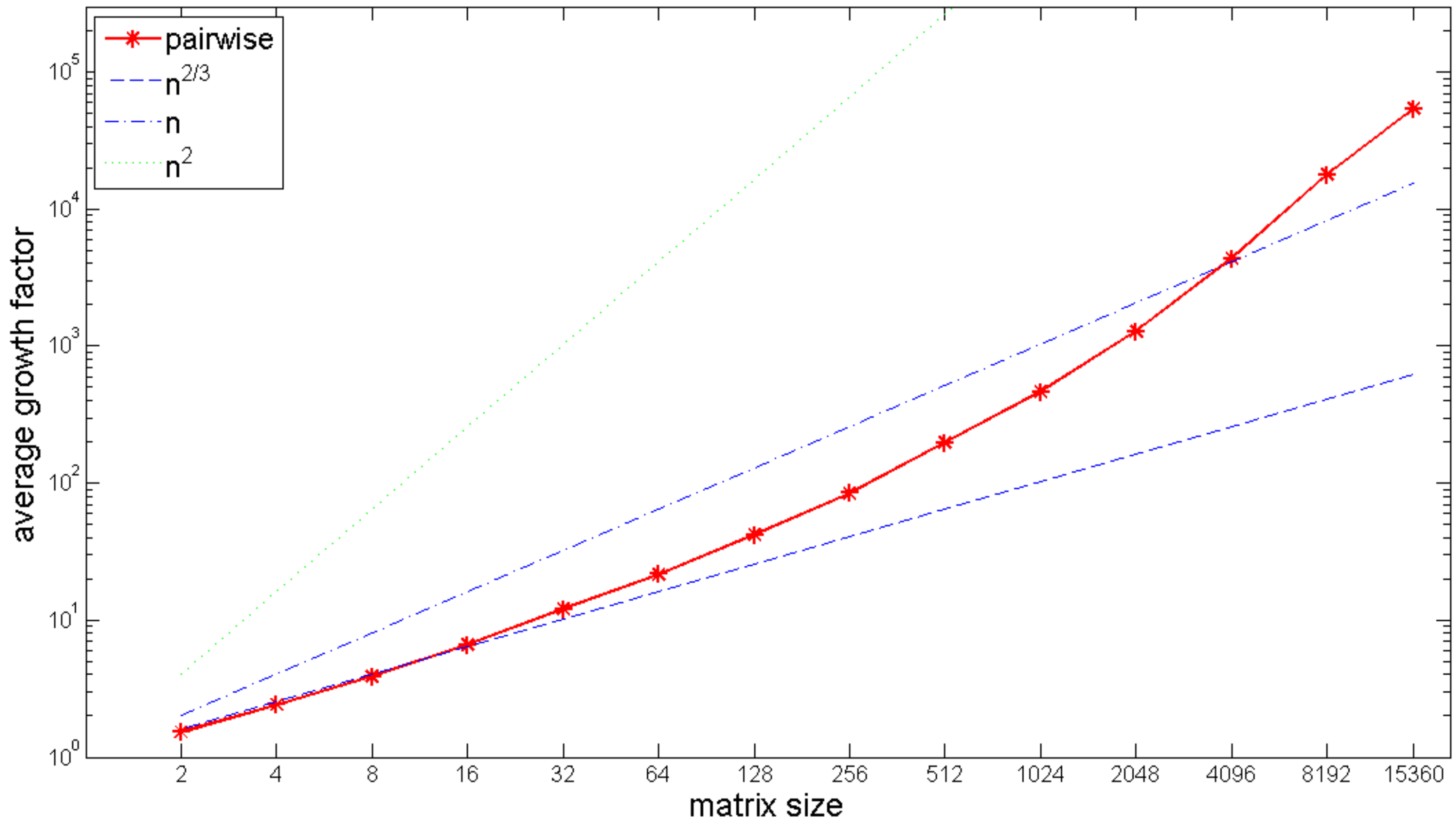


- Unstable for large number of processors P
- When P=number rows, it corresponds to parallel pivoting, known to be unstable (Trefethen and Schreiber, 90)

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

$$A = \begin{array}{cc} \begin{array}{cc} b & n-b \end{array} & \\ \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} & \begin{array}{l} \}\ b \\ \}\ n-b \end{array} \end{array} \text{, where } W = \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$$

  - 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

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

Pick b pivot rows, form A_{01}

Same for A_{11}

$$\begin{pmatrix} A_{01} \\ A_{11} \end{pmatrix} = \underbrace{\Pi_{02}}_{\Pi_2} L_{02} U_{02}$$

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$_0$

$$\overset{W_0}{\begin{pmatrix}2&4\\0&1\\2&0\\1&2\end{pmatrix}} = \Pi_0 L_0 U_0 \qquad \overset{\Pi_0^T W_0}{\begin{pmatrix}2&4\\2&0\end{pmatrix}} \longrightarrow \overset{\overline{W}_0}{\begin{pmatrix}2&4\\2&0\\4&1\\2&0\end{pmatrix}} = \overline{\Pi}_0 \overline{L}_0 \overline{U}_0 \qquad \overset{\overline{\Pi}_0^T \overline{W}_0}{\begin{pmatrix}4&1\\2&4\end{pmatrix}} \longrightarrow \overset{\underline{W}_0}{\begin{pmatrix}4&1\\2&4\\4&2\\1&4\end{pmatrix}} = \underline{\Pi}_0 \underline{L}_0 \underline{U}_0 \qquad \overset{\underline{\Pi}_0^T \underline{W}_0}{\begin{pmatrix}4&1\\1&4\end{pmatrix}}$$

Good pivots for factorizing W

P$_1$

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

P$_2$

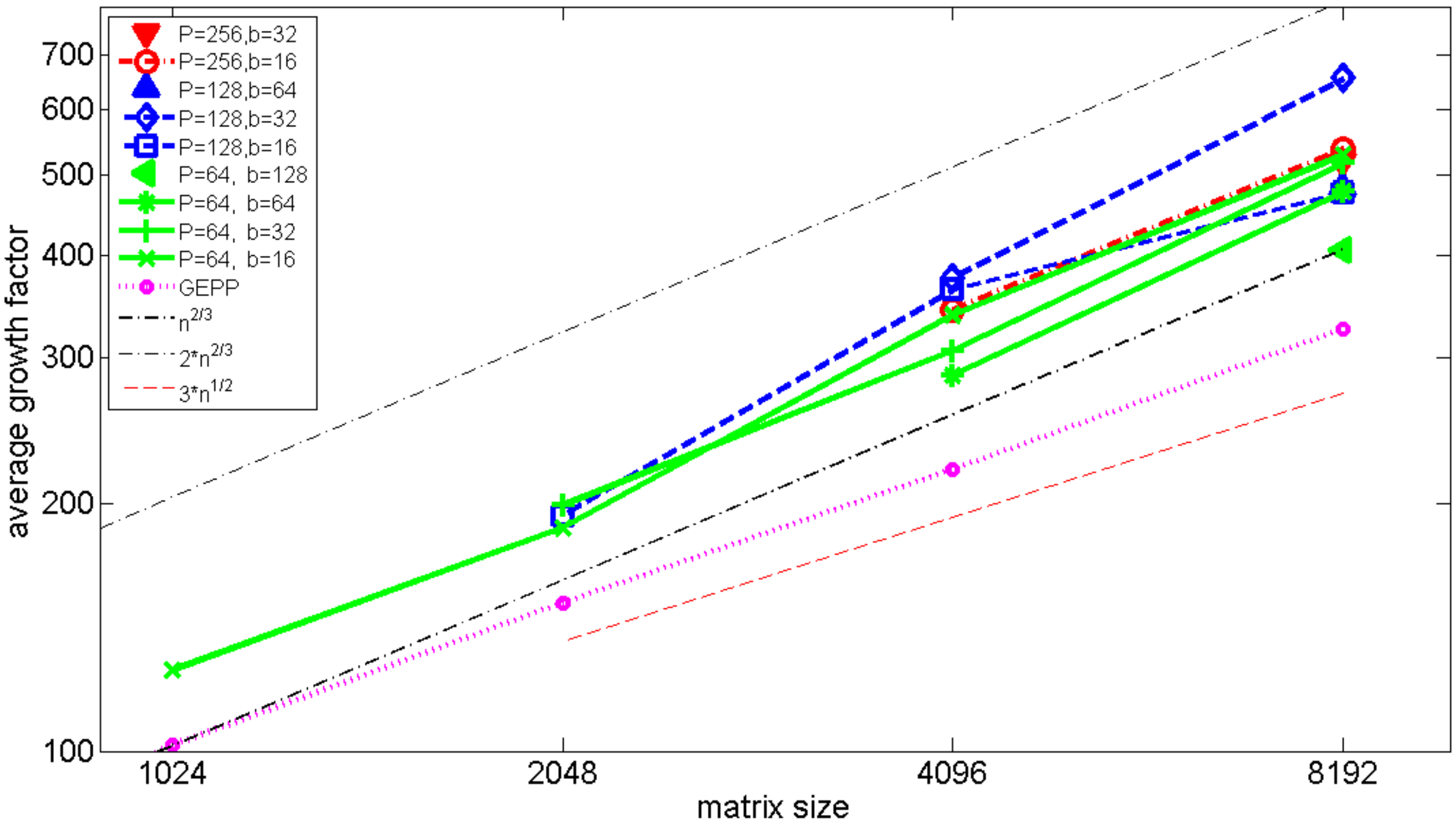
$$\overset{W_2}{\begin{pmatrix}0&1\\1&4\\0&0\\0&2\end{pmatrix}} = \Pi_2 L_2 U_2 \qquad \overset{\Pi_2^T W_2}{\begin{pmatrix}1&4\\0&2\end{pmatrix}} \longrightarrow \overset{\overline{W}_2}{\begin{pmatrix}1&4\\0&2\\4&2\\0&2\end{pmatrix}} = \overline{\Pi}_2 \overline{L}_2 \overline{U}_2 \qquad \overset{\overline{\Pi}_2^T \overline{W}_2}{\begin{pmatrix}4&2\\1&4\end{pmatrix}}$$

P$_3$

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

time $\longrightarrow$

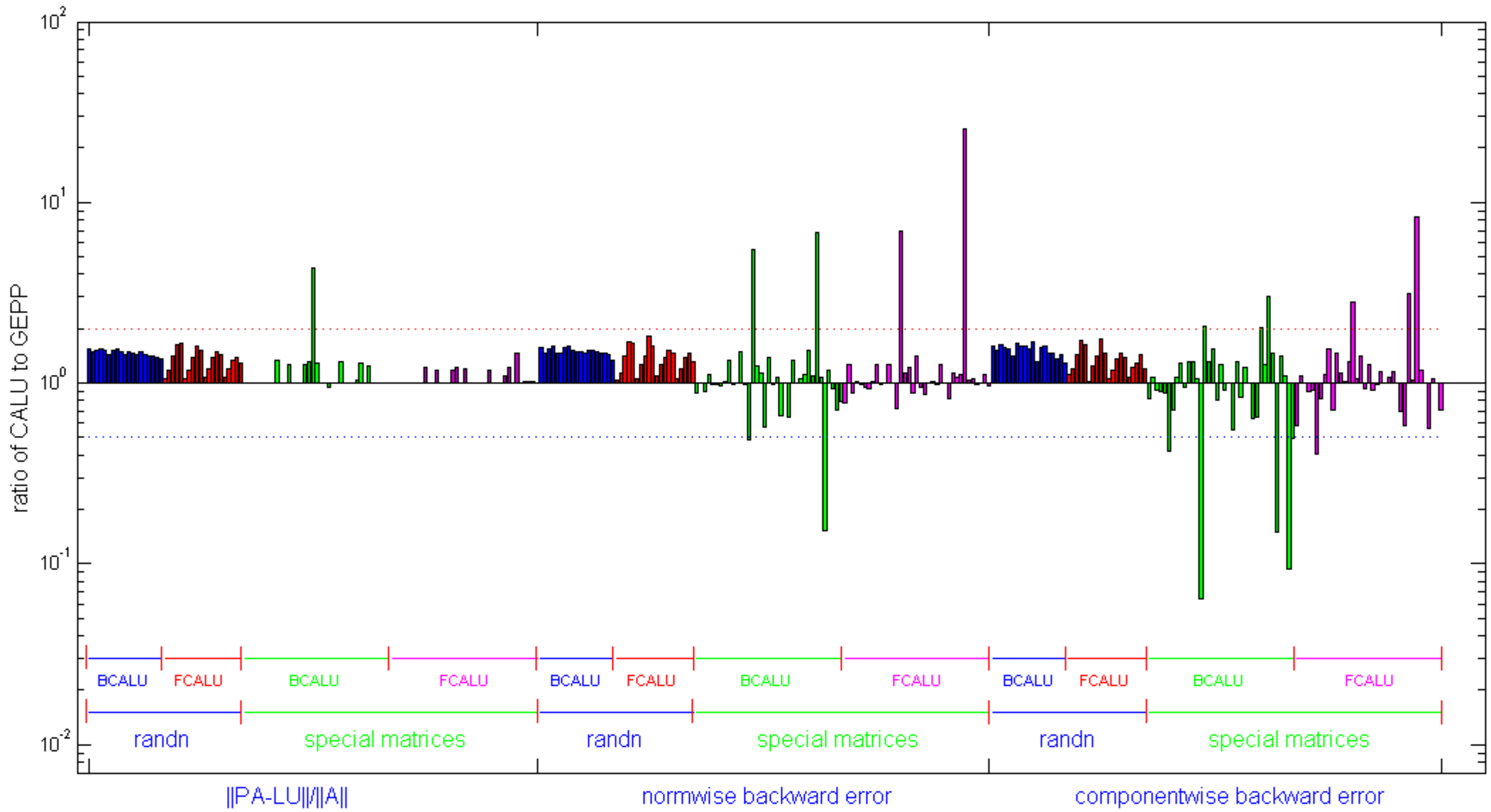
Growth factor for binary tree based CALU



- 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:

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

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

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

Outline of the proof of stability for CALU

- Consider $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix}$, and the result of TSLU as $\begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \end{bmatrix}$ $\begin{tikzcd} A_{11} \arrow[r] & A_{11} \arrow[dr] & \\ A_{21} \arrow[dr] & & \bar{A}_{11} \\ A_{31} \arrow[r] & A_{21} \arrow[ur] & \end{tikzcd}$

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

$$\begin{pmatrix} \Pi_{11} & \Pi_{12} & \\ \Pi_{21} & \Pi_{22} & \\ & & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{L}_{11} & & \\ \bar{L}_{21} & I & \\ \bar{L}_{31} & & I \end{pmatrix} \begin{pmatrix} \bar{U}_{11} & \bar{U}_{12} \\ & A^s_{22} \\ & A^s_{32} \end{pmatrix}$$

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

$$G = \begin{pmatrix} \bar{A}_{11} & & \bar{A}_{12} \\ A_{21} & A_{21} & \\ & -A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{L}_{11} & & \\ A_{21}\bar{U}_{11}^{-1} & L_{21} & \\ & -L_{31} & I \end{pmatrix} \begin{pmatrix} \bar{U}_{11} & & \bar{U}_{12} \\ & U_{21} & -L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} \\ & & A^s_{32} \end{pmatrix}$$

- GEPP on G does not permute and

$$L_{31}L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A^s_{32} = L_{31}U_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A^s_{32} = A_{31}\bar{U}_{11}^{-1}\bar{U}_{12} + A^s_{32} = \bar{L}_{31}\bar{U}_{12} + A^s_{32} = A_{32}$$

LU factorization and low rank matrices

- For low rank matrices, the factorization of A_1 computed as following might not be stable

  Compute PA=LU by using GEPP — L(k+1:end,k) = A(k+1:end,k)/A(k,k)

  Permute the matrix A_1=PA

  Compute LU with no pivoting $A_1=L_1U_1$ — L(k+1:end,k) = L(k+1:end,k)* (1/A(k,k))

- Example A = randn(6,3)*randn(3,5), max(abs(L)) = 1, max(abs(L_1)) = 10^{15}

After 4 steps of factorization of PA we obtain:

$$PA^4 = \begin{pmatrix} 1.0000 & & & & \\ 0.1729 & 1.0000 & & & \\ 0.6061 & 0.8608 & 1.0000 & & \\ 0.5776 & 0.0543 & 0.3264 & 1.0000 & \\ 0.4789 & -0.2877 & -0.1545 & 2.3333 & \boxed{2.3e-16} \\ -0.3264 & -0.7514 & -0.4597 & 1.7778 & \boxed{8.3e-17} \end{pmatrix} \cdot \begin{pmatrix} 4.4766 & 3.0163 & -4.7390 & 4.2180 & -0.8164 \\ & -1.5439 & -0.4703 & 1.9267 & 1.0925 \\ & & 1.6149 & 2.3623 & 0.3167 \\ & & & 9.9e-16 & 1.6e-16 \\ & & & & 1 \end{pmatrix}$$

Schur complement after 4 elimination steps

After 4 steps of factorization of A_1 we obtain:

$$A_1^4 = \begin{pmatrix} 1.0000 & & & & \\ 0.1729 & 1.0000 & & & \\ 0.6061 & 0.8608 & 1.0000 & & \\ 0.5776 & 0.0543 & 0.3264 & 1.0000 & \\ 0.4789 & -0.2877 & -0.1545 & 2.3333 & \boxed{4.9e-32} \\ -0.3264 & -0.7514 & -0.4597 & 1.7778 & \boxed{-7.4e-17} \end{pmatrix} \cdot \begin{pmatrix} 4.4766 & 3.0163 & -4.7390 & 4.2180 & -0.8164 \\ & -1.5439 & -0.4703 & 1.9267 & 1.0925 \\ & & 1.6149 & 2.3623 & 0.3167 \\ & & & 9.9e-16 & 1.6e-16 \\ & & & & 1 \end{pmatrix}$$

LU_PRRP: LU with panel rank revealing pivoting

- Pivots are selected by using strong rank revealing QR on each panel
- The factorization after one panel elimination is written as

$$PA = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I_b & \\ A_{21}A_{11}^{-1} & I_{n-b} \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$$

$A_{21} A_{11}^{-1}$ is computed through strong rank revealing QR
and $\max(|A_{21} A_{11}^{-1}|)_{ij} \leq f$

- LU_PRRP and CALU_PRRP stable for pathological cases (Wilkinson matrix) and matrices from two real applications (Voltera integral equation - Foster, a boundary value problem - Wright) on which GEPP fails.

A. Khabou, J. Demmel, LG, M. Gu, 2012

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	$n^{2/3}$ -- $n^{1/2}$	$n^{2/3}$ -- $n^{1/2}$	$(n/b)^{2/3}$ -- $(n/b)^{1/2}$	$(n/b)^{2/3}$ -- $(n/b)^{1/2}$

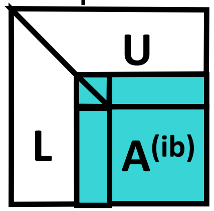
Better bounds

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

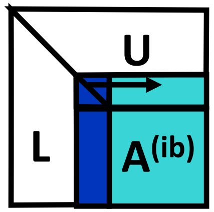
  $n^{1/2} = 10^{3.5}$

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.

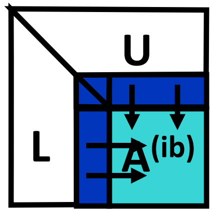


For ib = 1 to n-1 step b
  $A^{(ib)} = A(ib{:}n,\ ib{:}n)$



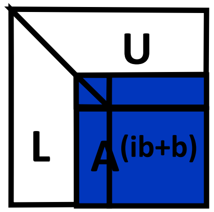
(1) Find permutation for current panel using TSLU $O(n/b \log_2 P_r)$

(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)

(4) Compute block row of U (pdtrsm) $O(n/b \log_2 P_c)$
  - 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: $(2/3)n^3/P + (3/2)n^2 b / P^{1/2}$ vs $(2/3)n^3/P + n^2 b/P^{1/2}$
  - 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

Performance vs ScaLAPACK

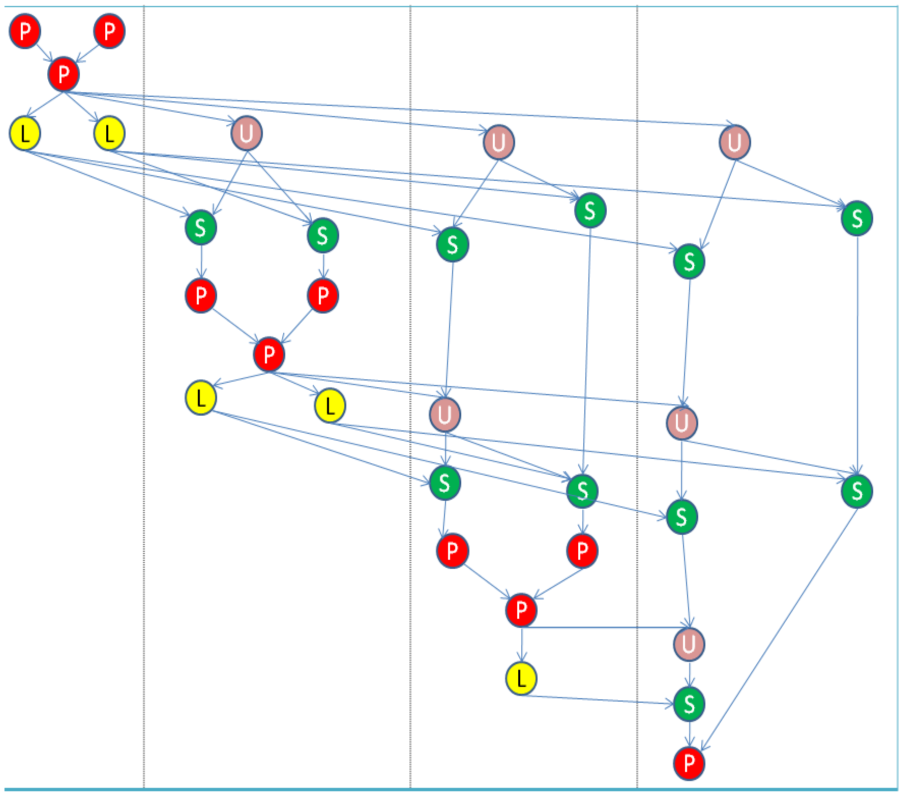
- Parallel TSLU (LU on tall-skinny matrix)
  - IBM Power 5
    - Up to **4.37x** faster (16 procs, 1M x 150)
  - Cray XT4
    - Up to **5.52x** faster (8 procs, 1M x 150)

- 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).

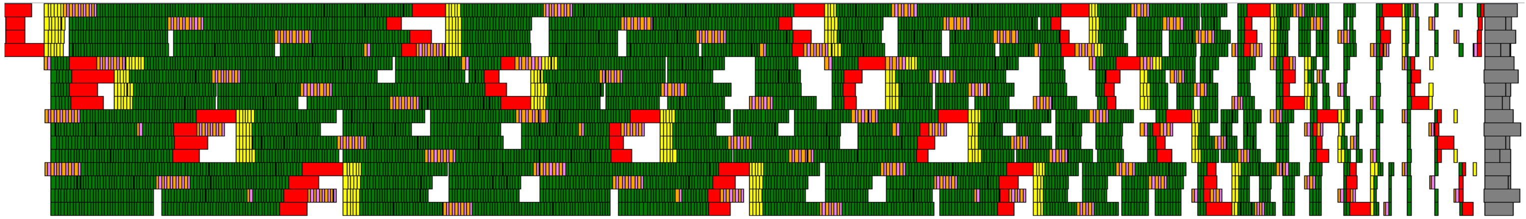
CALU and its task dependency graph

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

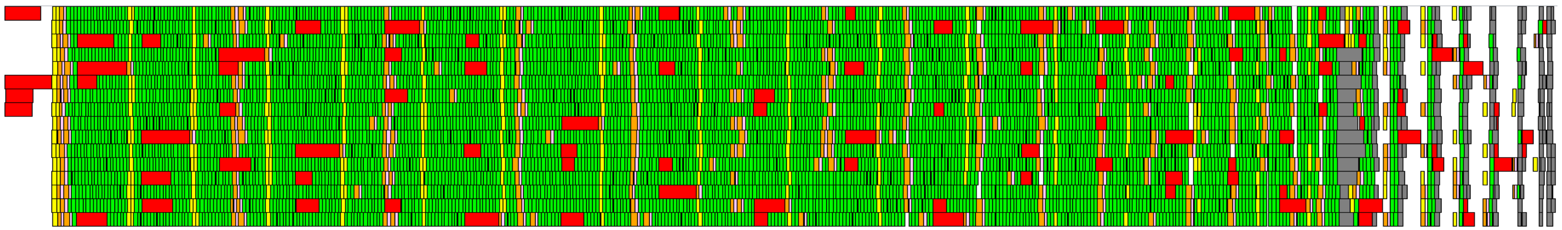


Scheduling CALU's Task Dependency Graph

- 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)	√	√	√

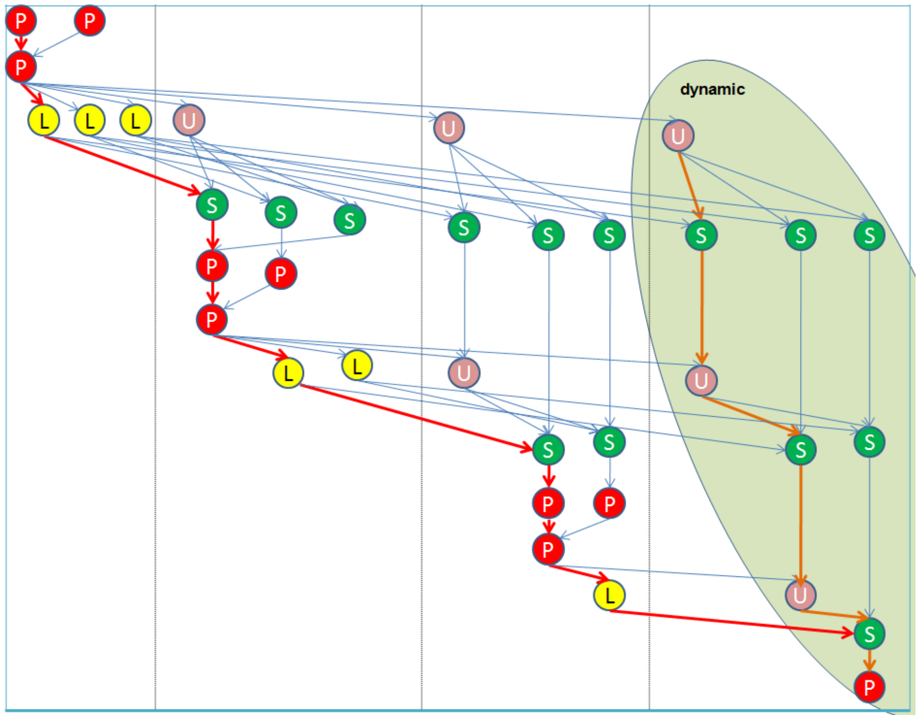
S. Donfack, LG, B. Gropp, V. Kale,IPDPS 2012

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:

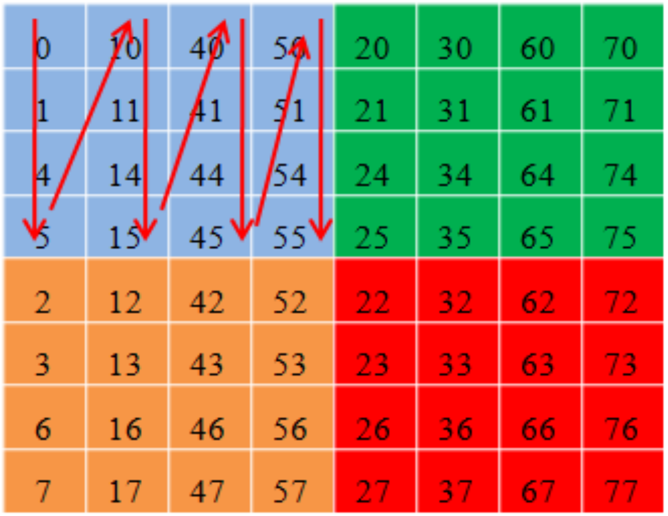
- A thread executes in priority its statically assigned tasks
- When no task ready, it picks a ready task from the dynamic part
- The size of the dynamic part is guided by a performance model



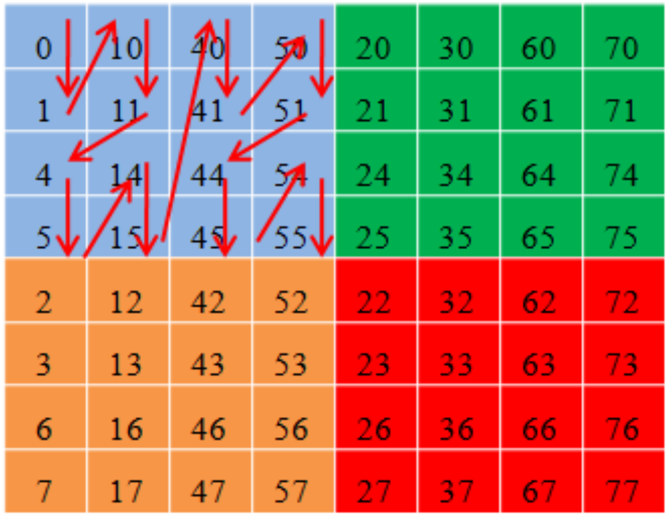
S. Donfack, LG, B. Gropp, V. Kale, 2012

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



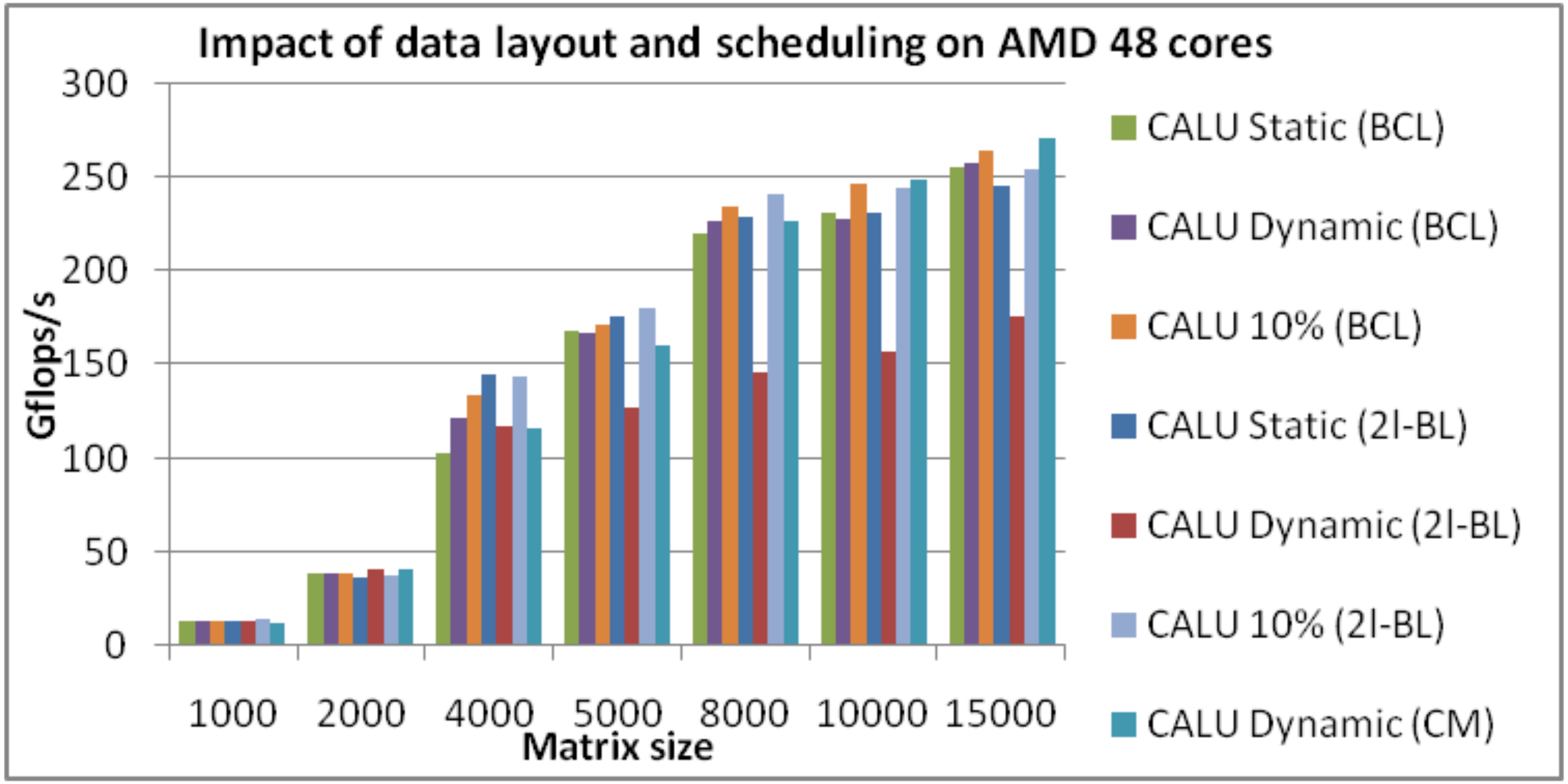
Block cyclic layout (BCL)



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

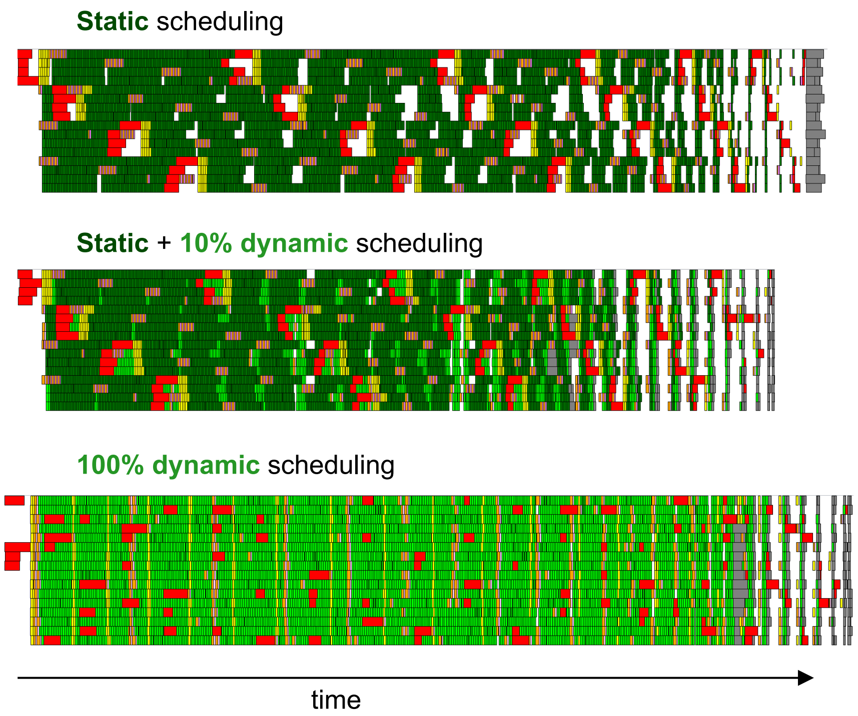


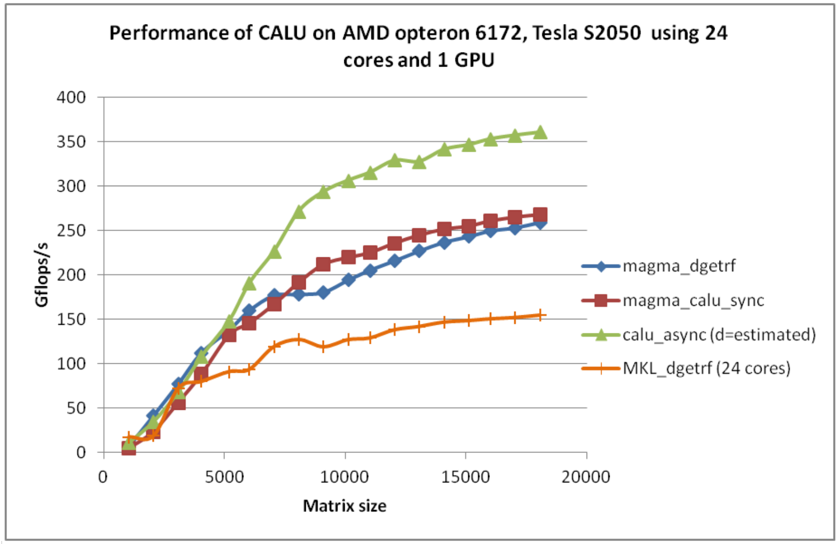
Eight socket, six core machine based on AMD Opteron processor (U. of Tennessee).

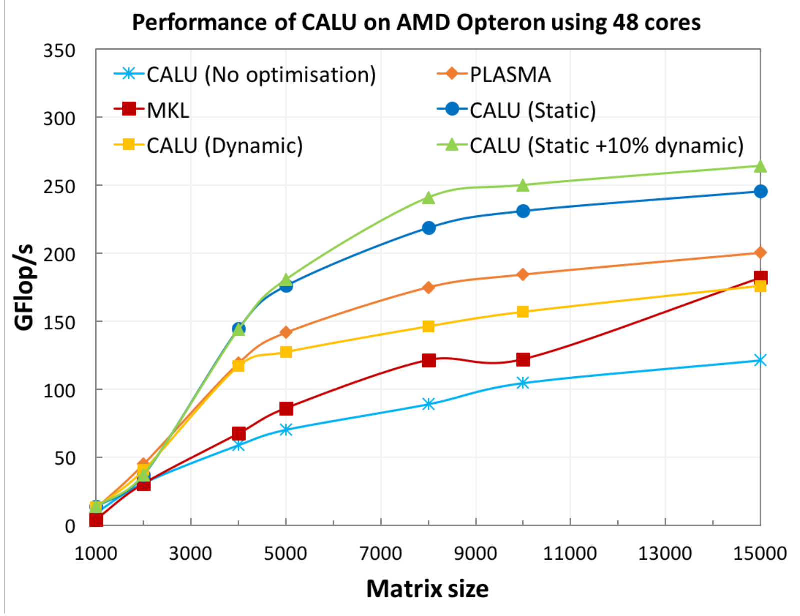
BCL : Each thread stores contiguously (CM) its data

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

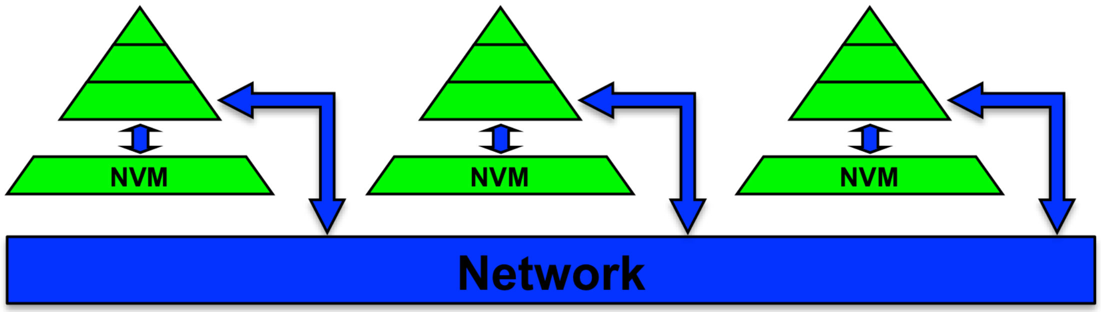
Parallel write avoiding algorithms

Need to avoid writing suggested by emerging memory technologies, as NVMs:

- Writes more expensive (in time and energy) than reads
- Writes are less reliable than reads

Some examples:

- Phase Change Memory: Reads 25 us latency
  Writes: 15x slower than reads (latency and bandwidth)
  consume 10x more energy
- Conductive Bridging RAM - CBRAM
  Writes: use more energy (1pJ) than reads (50 fJ)
- Gap improving by new technologies such as XPoint and other FLASH alternatives, but not eliminated



Parallel write-avoiding algorithms

- Matrix A does not fit in DRAM (of size M), need to use NVM (of size n^2 / P)

- Two lower bounds on volume of communication
  - Interprocessor communication: $\Omega(n^2 / P^{1/2})$
  - Writes to NVM: n^2 / P

- Result: any three-nested loop algorithm (matrix multiplication, LU,..), must asymptotically exceed at least one of these lower bounds
  - If $\Omega(n^2 / P^{1/2})$ words are transferred over the network, then $\Omega(n^2 / P^{2/3})$ words must be written to NVM !

- Parallel LU: choice of best algorithm depends on hardware parameters

	#words interprocessor comm.	#writes NVM
Left-looking	$O((n^3 \log^2 P) / (P M^{1/2}))$	$O(n^2 / P)$
Right-looking	$O((n^2 \log P) / P^{1/2})$	$O((n^2 \log^2 P) / P^{1/2})$