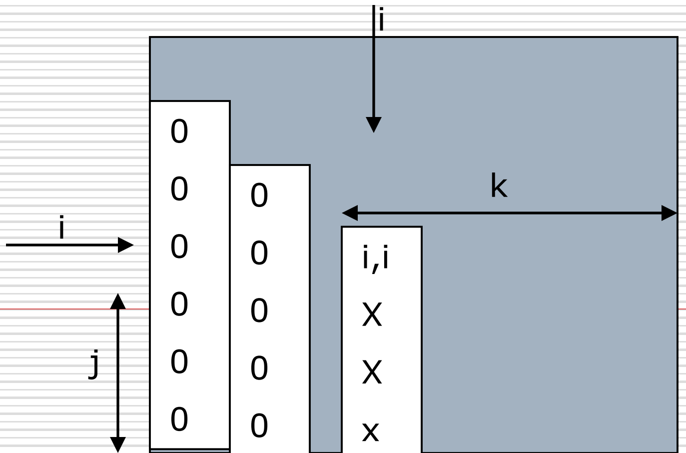
Parallel Linear Algebra (Linear System of Equations)

Sathish Vadhiyar

Gaussian Elimination - Review

Version 1

for each column i
  zero it out below the diagonal by adding multiples of row i to later rows
for i= 1 to n-1
  for each row j below row i
  for j = i+1 to n
    add a multiple of row i to row j
    for k = i to n
      A(j, k) = A(j, k) – A(j, i)/A(i, i) * A(i, k)

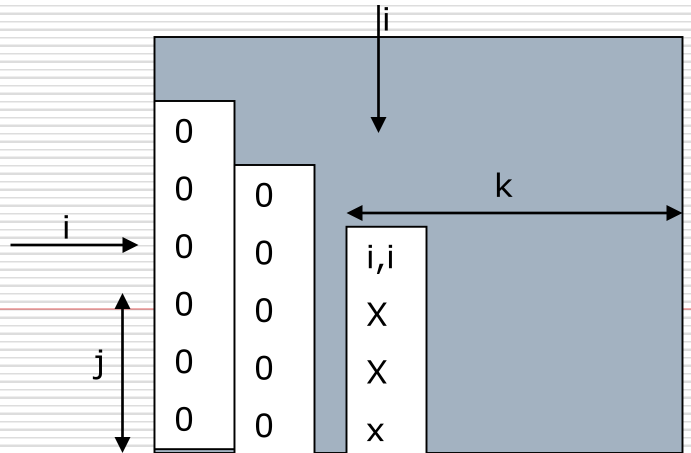


Gaussian Elimination - Review

Version 2 – Remove A(j, i)/A(i, i) from inner loop

for each column i
  zero it out below the diagonal by adding multiples of row i to later rows

```
for i= 1 to n-1
  for each row j below row i
  for j = i+1 to n
    m = A(j, i) / A(i, i)
    for k = i to n
      A(j, k) = A(j, k) – m* A(i, k)
```

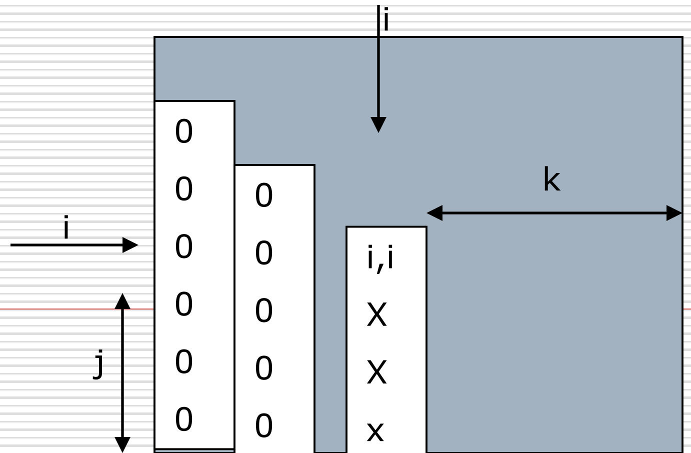


Gaussian Elimination - Review

Version 3 – Don't compute what we already know

for each column i
  zero it out below the diagonal by adding multiples of row i to later rows

```
for i= 1 to n-1
  for each row j below row i
  for j = i+1 to n
    m = A(j, i) / A(i, i)
    for k = i+1 to n
      A(j, k) = A(j, k) – m* A(i, k)
```



Gaussian Elimination - Review

Version 4 – Store multipliers m below diagonals

for each column i

  zero it out below the diagonal by adding multiples of row i to later rows

for i= 1 to n-1

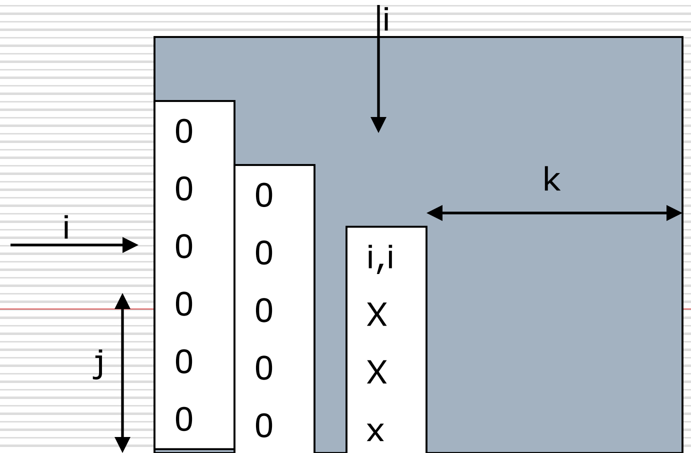
  for each row j below row i

  for j = i+1 to n

    A(j, i) = A(j, i) / A(i, i)

    for k = i+1 to n

      A(j, k) = A(j, k) – A(j, i)* A(i, k)



GE - Runtime

- Divisions

  $1 + 2 + 3 + \ldots (n-1) = n^2/2$ (approx.)

- Multiplications / subtractions

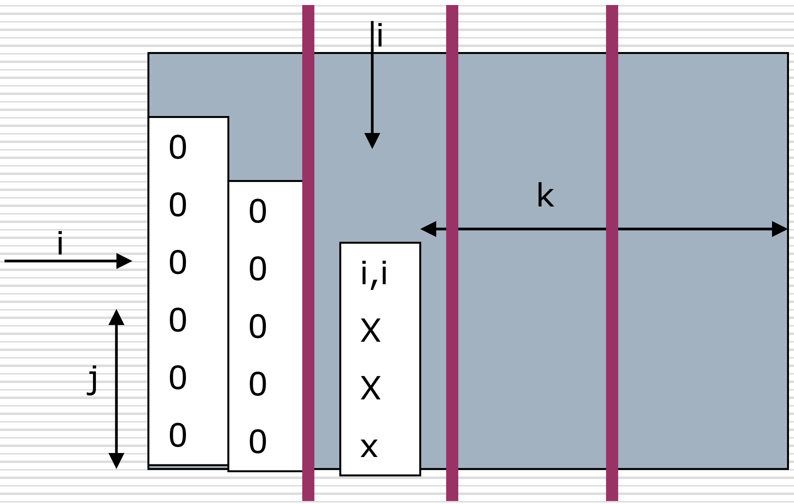
  $1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \ldots. (n-1)^2 = n^3/3 - n^2/2$

- Total

  $2n^3/3$

Parallel GE

- 1^{st} step – 1-D block partitioning along blocks of n columns by p processors



1D block partitioning - Steps

1. Divisions

$n^2/2$

2. Broadcast

$x\log(p) + y\log(p-1) + z\log(p-3) + \ldots \log 1 < n^2\log p$

3. Multiplications and Subtractions

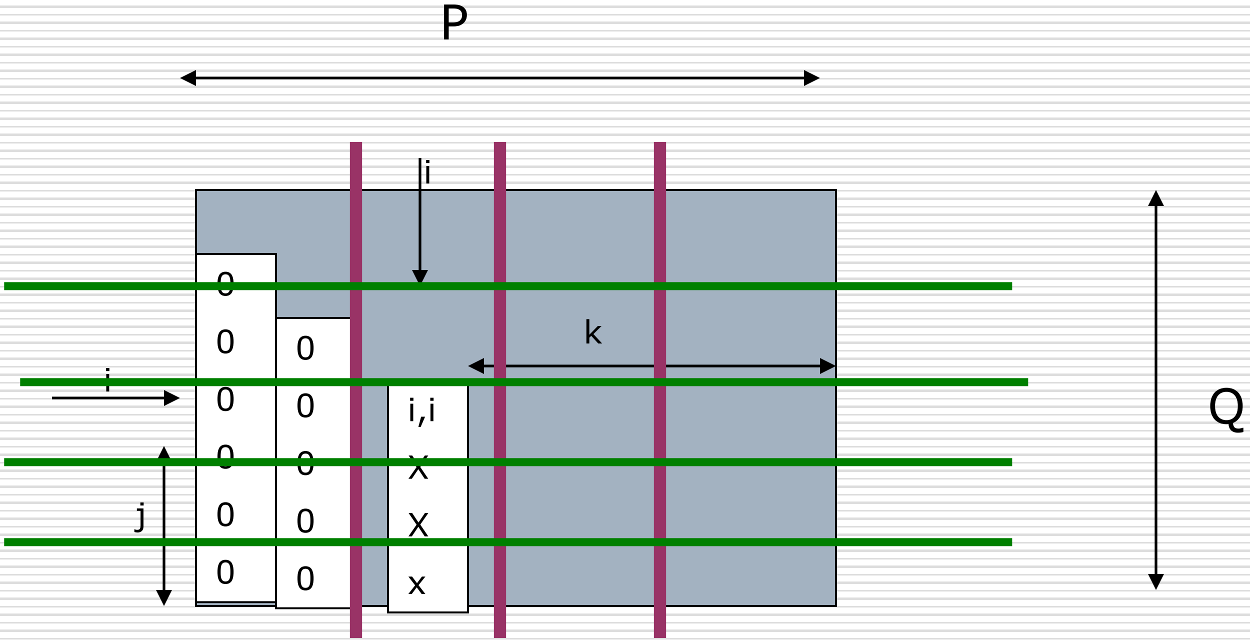
$(n-1)n/p + (n-2)n/p + \ldots.\ 1\mathrm{x}1 = n^3/p$ (approx.)

Runtime:

$< n^2/2 + n^2\log p + n^3/p$

2-D block

- To speedup the divisions



2D block partitioning - Steps

1. Broadcast of (k,k)

   $\log Q$

2. Divisions

   n^2/Q (approx.)

3. Broadcast of multipliers

   $x\log(P) + y\log(P-1) + z\log(P-2) + \ldots. = n^2/Q \log P$

4. Multiplications and subtractions

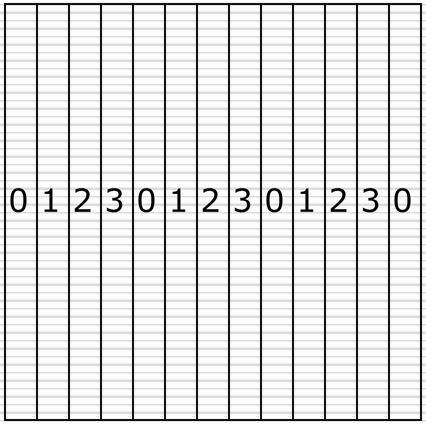
   n^3/PQ (approx.)

Problem with block partitioning for GE

- Once a block is finished, the corresponding processor remains idle for the rest of the execution
- Solution? -

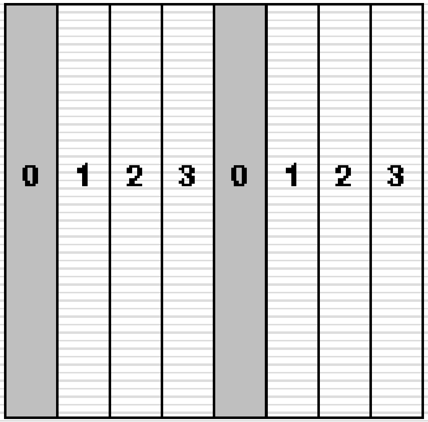
Onto cyclic

- The block partitioning algorithms waste processor cycles. No load balancing throughout the algorithm.
- Onto cyclic



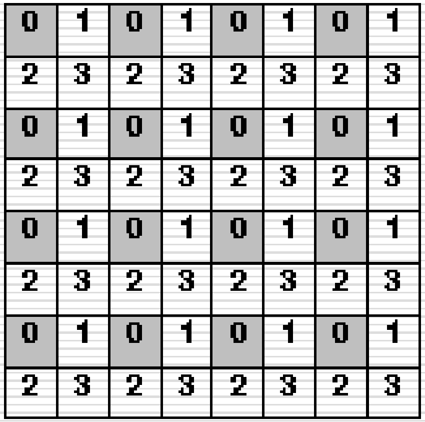
cyclic

Load balance



1-D block-cyclic

Load balance, block operations, but column factorization bottleneck



2-D block-cyclic

Has everything

Block cyclic

- Having blocks in a processor can lead to block-based operations (block matrix multiply etc.)
- Block based operations lead to high performance

GE: Miscellaneous
GE with Partial Pivoting

- 1D block-column partitioning: which is better? Column or row pivoting

•Column pivoting does not involve any extra steps since pivot search and exchange are done locally on each processor. O(n-i-1)

•The exchange information is passed to the other processes by piggybacking with the multiplier information

- Row pivoting
- Involves distributed search and exchange – O(n/P)+O(logP)

- 2D block partitioning: Can restrict the pivot search to limited number of columns

Triangular Solve – Unit Upper triangular matrix

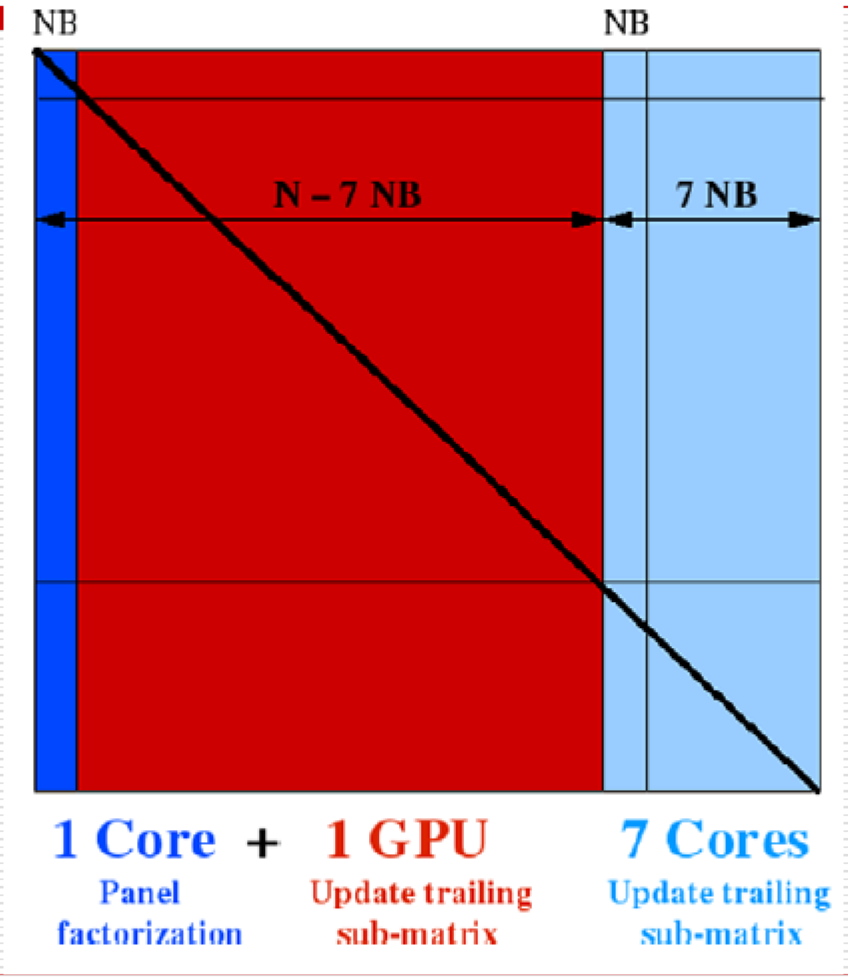
- Sequential Complexity - $O(n^2)$
- Complexity of parallel algorithm with 2D block partitioning ($P^{0.5}*P^{0.5}$) $O(n^2)/P^{0.5}$
  - Thus (parallel GE / parallel TS) < (sequential GE / sequential TS)
  - Overall (GE+TS) – $O(n^3/P)$

- Dense LU on GPUs

LU for Hybrid Multicore + GPU Systems
(Tomov et al., Parallel Computing, 2010)

- Assume the CPU host has 8 cores
- Assume NxN matrix; Divided into blocks of size NB;
- Split such that the first N-7NB columns are on GPU memory
- Last 7NB on the host

Load Splitting for Hybrid LU



Steps

- Current panel is downloaded to CPU; the dark blue part in the figure
- Panel factored on CPU and result sent to GPU to update trailing sub-matrix; the red part;
- GPU updates the first NB columns of the trailing submatrix
- Updated panel sent to CPU; Asynchronously factored on CPU while the GPU updates the rest of the trailing submatrix
- The rest of the 7 host cores update the last 7 NB host columns

Parallel Dense Matrix Computations – Tile-Based Cholesky and QR

- For heterogeneous architectures
- Communication-avoiding algorithms

Heterogeneous Tile Algorithms for Heterogeneous Architectures General Strategy

- Hetrogeneous tile algorithms
- Heterogeneous multi-level block-cyclic data distribution

- Source:
- Paper: “F. Song, S. Tomov, and J. Dongarra. Enabling and Scaling Matrix Computations on Heterogeneous Multi-core and Multi-

General Strategy

- Small tiles on the host, large tiles on the GPUs
- A two-level 1-D block-cyclic method
- First map a matrix to only CPUs using a 1-D column block cyclic distribution, and then cut out slices for GPUs

Hybrid Tile Data Layout

- Divide a matrix into a set of small and large tiles
- At the top level, divid
  large square tiles of s
- Subdivide each top le
  BxB
  into a number of small rectangular tiles of size Bxb, and a remaining tile

(a) (b)

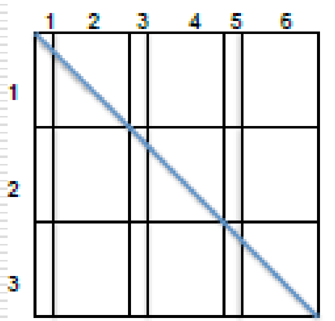
- Figure (a) – divide the 12x12 matrix into four 6x6 tiles, then divide each 6x6 tiles into two 6x1 and one 6x4

Heterogeneous Tile Cholesky Factorization

$$\left(\begin{array}{c|c|c} \overbrace{\begin{matrix} a_{11} & a_{12} \dots A_{1s} \\ a_{21} & a_{22} \dots A_{2s} \\ & \vdots \\ a_{p1} & a_{p2} \dots A_{ps} \end{matrix}}^{B} & \overbrace{\begin{matrix} a_{1(s+1)} & a_{1(s+2)} \dots A_{1(2s)} \\ a_{2(s+1)} & a_{2(s+2)} \dots A_{2(2s)} \\ & \vdots \\ a_{p(s+1)} & a_{p(s+2)} \dots A_{p(2s)} \end{matrix}}^{B} & \begin{matrix} \dots \\ \dots \\ \ddots \\ \dots \end{matrix} \end{array}\right), \text{ where}$$

- Each a_{ij} represents a small tile of size Bxb, and each A_{ij} represents a large tile of size Bx(B-b(s-1))

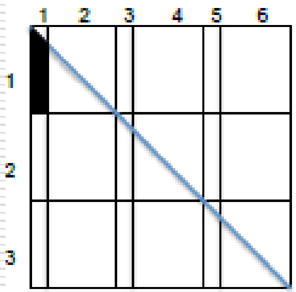
Heterogeneous Tile Cholesky Factorization - Illustration



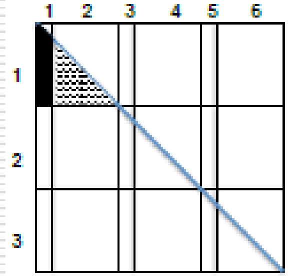
(a)

- Matrix divided into 3x3 tiles, i.e., p=3
- Each tile divided into one small and one large tile, i.e., s=2
- Factorization goes through six (pxs) iterations

Illustration Continued



(b)



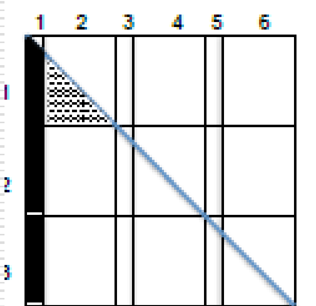
(c)

- Solve L … te its right A12

POTF2'(A_{tk}, L_{tk}): Given a matrix A_{tk} of $m \times n$ and $m \geq n$, we let $A_{tk} = \binom{A_{tk1}}{A_{tk2}}$ such that A_{tk1} is of $n \times n$, and A_{tk2} is of $(m-n) \times n$. We also let $L_{tk} = \binom{L_{tk1}}{L_{tk2}}$. POTF2' computes $\binom{L_{tk1}}{L_{tk2}}$ by solving $L_{tk1} = \text{Cholesky}(A_{tk1})$ and $L_{tk2} = A_{tk2} L_{tk1}^{-T}$.

GSMM(L_{ik}, L_{jk}, A_{ij}) computes $A_{ij} = A_{ij} - L_{ik} L_{jk}^T$.

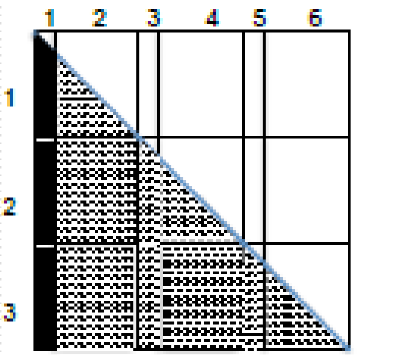
Illustration Continued



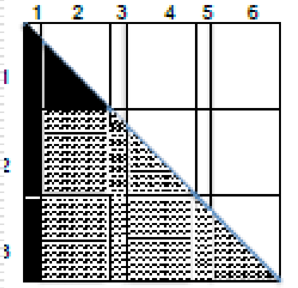
(d)

$\mathrm{TRSM}(L_{tk}, A_{ik}, L_{ik})$ computes $L_{ik} = A_{ik} L_{tk}^{-T}$. the two tiles below L11

Illustration Continued



(c)



(f)

- Apply GSMMs to update all tiles Second iteration to the right of the first tile column Start from the second tile column

Heterogeneous 1D Column Block Cyclic Distribution

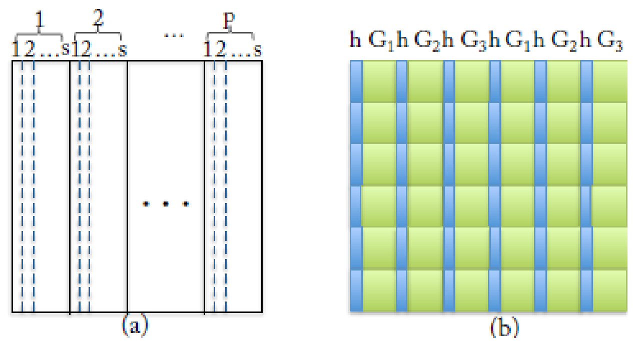


Figure 7: Heterogeneous 1-D column block cyclic data distribution. **(a)** The matrix A divided by a two-level partitioning method. (p, s) determines a matrix partition. **(b)** Allocation of a matrix of 6×12 rectangular tiles (i.e., p=6, s=2) to a host and three GPUs: `h`, $\mathtt{G}_1$, $\mathtt{G}_2$, and $\mathtt{G}_3$.

QR Factorization, TSQR, CAQR

- Sources, Credits, some slides taken from:
- Slides on “Communication Avoiding QR and LU”, CS 294 lecture slides, Laura Grigori, ALPINES INRIA Rocquencourt - LJLL, UPMC
- https://who.rocq.inria.fr/Laura.Grigori/TeachingDocs/CS-294_Spr2016/Slides_CS-294_Spr2016/CS294_Spr16_CALUQR.pdf

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:

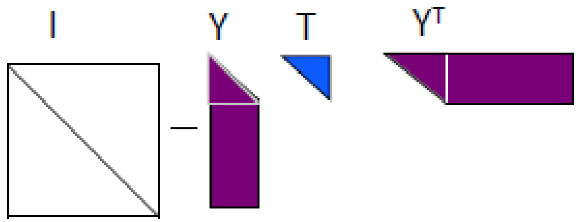
$$H_n H_{n-1} \ldots H_2 H_1 A = R \rightarrow A = \left(H_n H_{n-1} \ldots H_2 H_1\right)^T R$$
$$Q = H_1 H_2 \ldots H_n$$

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:

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

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

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

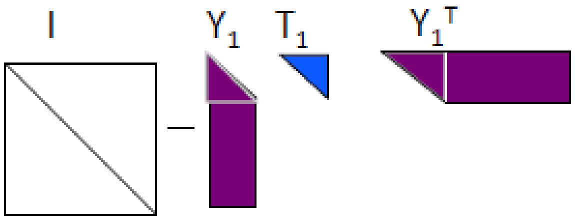
Block QR algebra:

1. Compute panel factorization:

   $$\begin{pmatrix} A_{11} \\ A_{12} \end{pmatrix} = Q_1 \begin{pmatrix} R_{11} \\ \\ \end{pmatrix}, \quad Q_1 = H_1 H_2 \ldots H_b$$

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.

QR Factorization for Tall and Skinny Matrices (TSQR)

- Parallelization using Binary Tree

Parallel TSQR Factorization on a Binary Tree of Four Processors

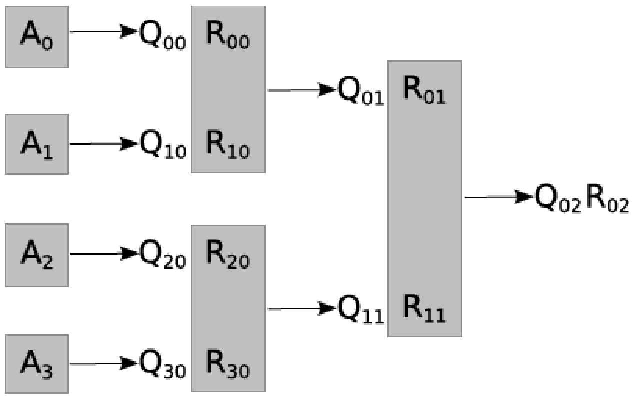
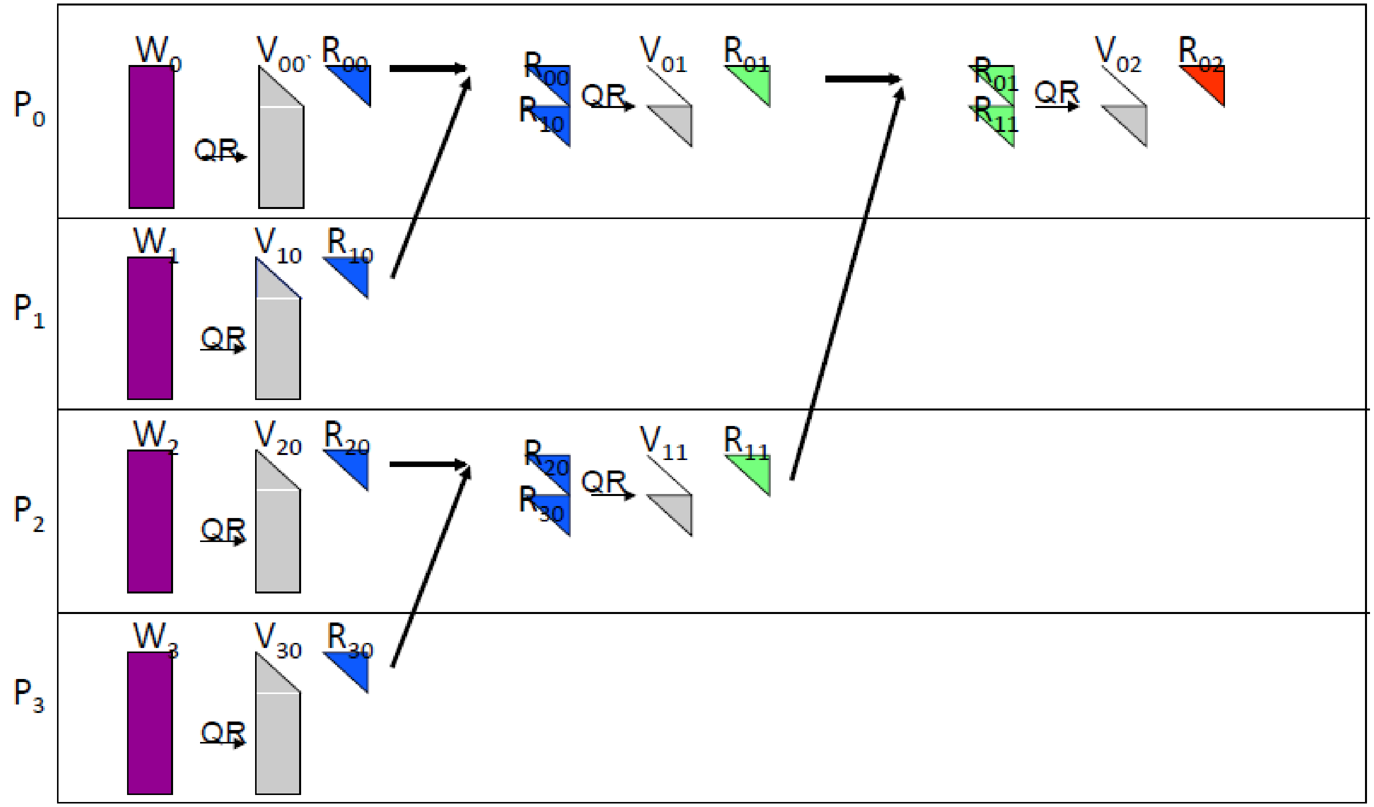


Figure 1: Execution of the parallel TSQR factorization on a binary tree of four processors. The gray boxes indicate where local QR factorizations take place. The Q and R factors each have two subscripts: the first is the sequence number within that stage, and the second is the stage number.

Parallel TSQR



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

Steps – Stage 0

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix}.$$

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \begin{pmatrix} Q_{00}R_{00} \\ Q_{10}R_{10} \\ Q_{20}R_{20} \\ Q_{30}R_{30} \end{pmatrix}.$$

$$A = \begin{pmatrix} Q_{00}R_{00} \\ Q_{10}R_{10} \\ Q_{20}R_{20} \\ Q_{30}R_{30} \end{pmatrix} = \left(\begin{array}{c|c|c|c} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ \hline R_{30} \end{array}\right).$$

Steps

- Stagae 1 –

$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} R_{00} \\ R_{10} \end{pmatrix} \\ \hline \begin{pmatrix} R_{20} \\ R_{30} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} Q_{01} R_{01} \\ \hline Q_{11} R_{11} \end{pmatrix}.$$

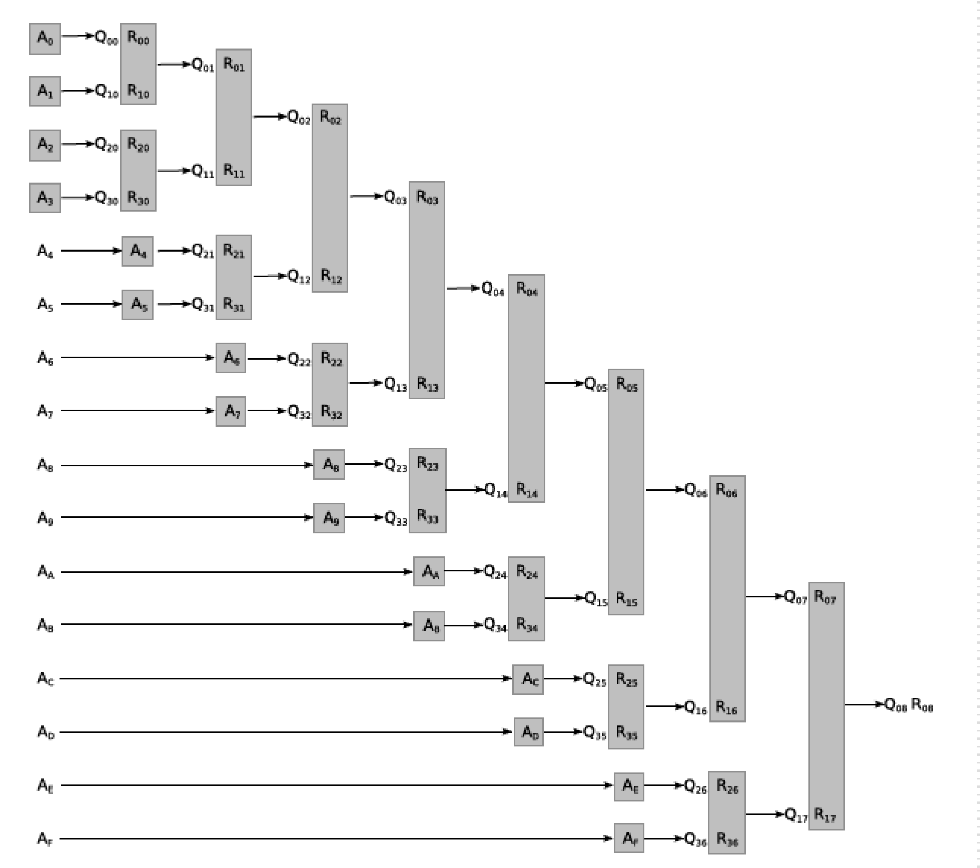
- Stag 2

$$\begin{pmatrix} R_{01} \\ R_{11} \end{pmatrix} = Q_{02} R_{02}.$$

- Compl

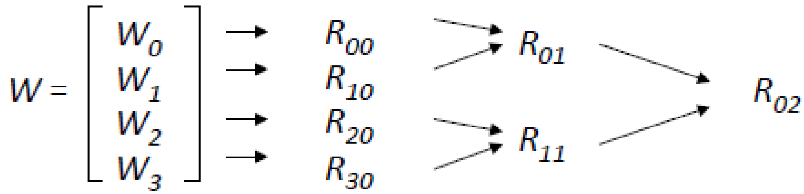
$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \left(\begin{array}{c|c|c|c} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c|c} Q_{01} & \\ \hline & Q_{11} \end{array}\right) \cdot Q_{02} \cdot R_{02},$$

QR on General Trees using 16 blocks and 4 processors

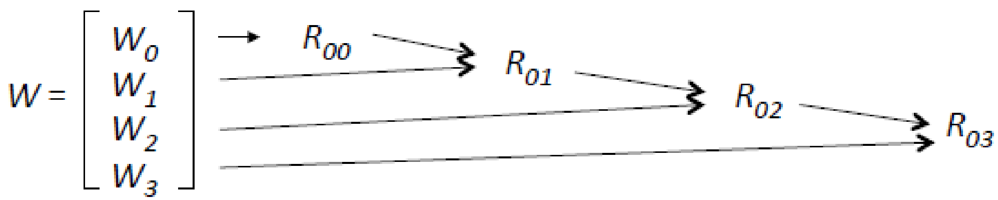


Flexibility of TSQR and CAQR algorithms

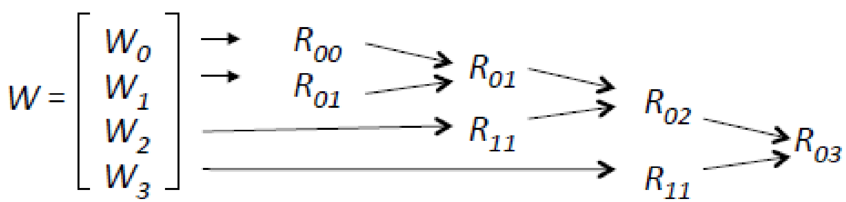
Parallel:



Sequential:



Dual Core:



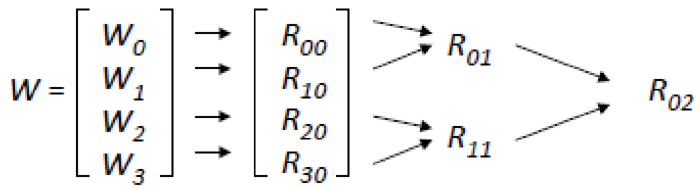
Reduction tree will depend on the underlying architecture, could be chosen dynamically

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 $\propto$ b log P

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



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

Parallel CAQR (Communication-avoiding QR)

- Uses parallel TSQR
- mxn matrix distributed in 2D block-cyclic distribution with block size b
- At each step of factorization, TSQR is used to factor a panel of columns
- Followed by trailing matrix update – applying the Householder vectors to the rest of the matrix
- The update corresponding to the QR factorization at the leaves of the TSQR tree is performed locally on every processor
- The updates corresponding to the upper levels of the TSQR tree are performed between groups of neighboring trailing matrix processors

Parallel CAQR

- Only one of the trailing matrix processors in each neighbour group continues to be involved in successive trailing matrix updates
- Allows overlap of computation and communication – uninvolved processors can finish their computations in parallel with successive reduction stages

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