CS 598: Communication Cost Analysis of Algorithms

Lecture 7: parallel algorithms for QR factorization

Edgar Solomonik

University of Illinois at Urbana-Champaign

September 14, 2016

Review of Householder QR

Represent orthogonal matrix as $Q = I - YTY^T$

- Y is tall-and-skinny, lower trapezoidal
- T is upper triangular and satisfies $T^{-1} + T^{-T} = -Y^T Y$
- therefore, given Y, we can compute T
- we can combine Householder transformations

$$Q_1 = I - Y_1 T_1 Y_1^T$$
$$Q_2 = I - Y_2 T_2 T_2^T$$
$$Q_1 Q_2 = I - YTY^T$$
$$Y = \begin{bmatrix} Y_1 & 0 \\ \vdots & Y_2 \end{bmatrix}$$
$$T^{-1} + T^{-T} = -Y^T Y$$

1D Householder QR

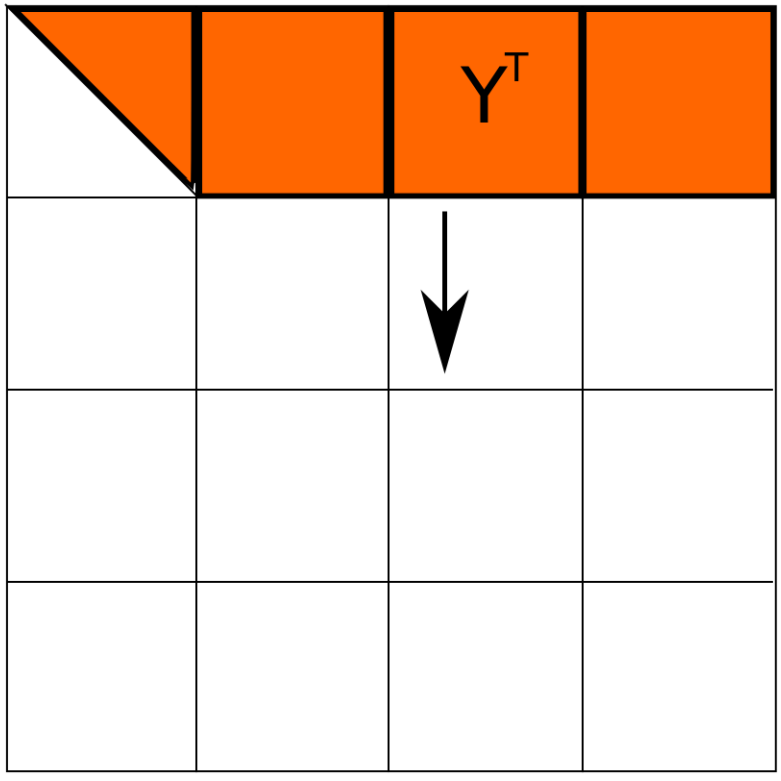
Lets start with a simple parallel QR algorithm

- assign processor $\Pi(i)$ columns $A_i = A(:, in/P + 1 : (i+1)n/P)$
- factorize A_0 using $\Pi(0)$
  - get Householder vector y_1, apply $(I - 2y_1 y_1^T)A_0$, move to next column
  - collect these vectors into trapezoidal matrix $Y_0 = [y_1, \ldots, y_{n/P}]$
  - compute $S_0 = Y_0^T Y_0$, extract lower-triangular part of S_0 and invert it to get T_0
  - so we obtain Y_0 and T_0 such that $(I - Y_0 T_0 Y_0^T)^T A_0 = R_0$
- broadcast Y_0, T_0 to $\Pi(1 : P-1)$
- compute $(I - Y_0 T_0 Y_0^T)^T A_i$ with $\Pi(i)$ for $i \in [1, P-1]$
- Q: how many operations does the above update require on each processor, $O(n^3/P)$ or $O(n^3/P^2)$? Recall, Y_0 is $n \times n/P$.
- A: the latter, via $A_i - (Y_0(T_0^T(Y_0^T A_i)))$
- continue with $\Pi(1)$ on lower $n - n/P$ rows of A_1, etc.
- Q: what is the communication cost of this algorithm?
- A: $O(P \cdot \log(P) \cdot \alpha + n^2 \cdot \beta)$

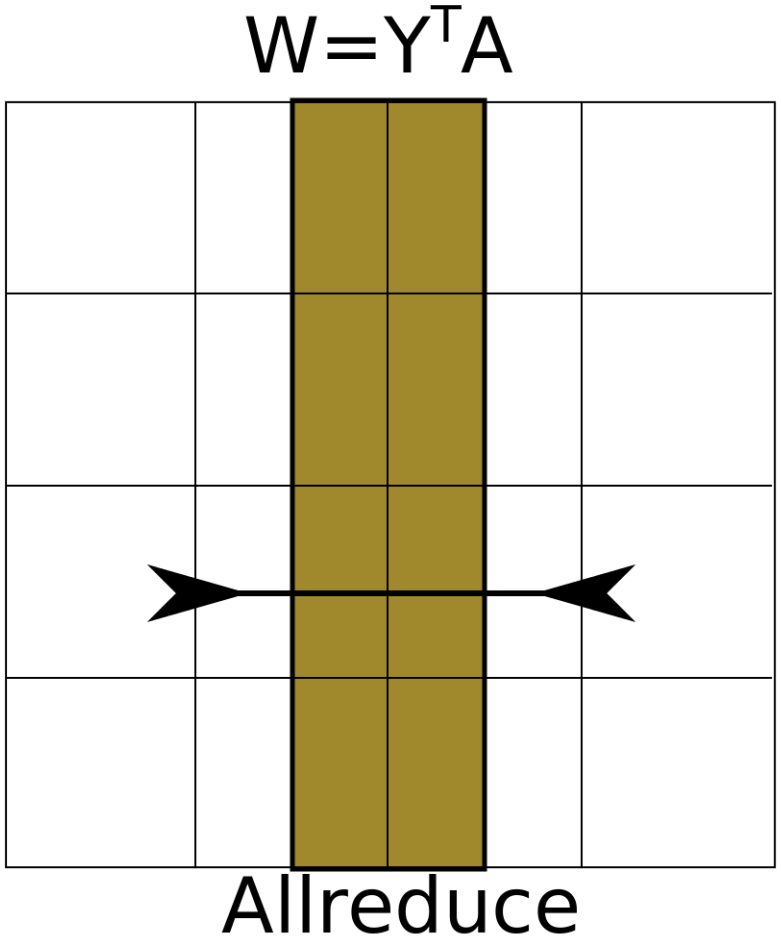
2D Householder QR

- column $\Pi(:, i)$ owns columns $A_i = A(:, in/P + 1 : (i+1)n/P)$
- factorize A_0 using $\Pi(:, 0)$
  - 1-element reduction/broadcast to compute each Householder vector
  - applying $(I - 2y_1 y_1^T)A_0$ needs allreduce of size up to $n/\sqrt{P}$ for $y_1^T A_0$
  - computing T_0 has cost $O(\beta \cdot n^2/P)$
- broadcast T_0 everywhere $(\Pi(:, :))$
- send Y_0 blocks so they are distributed along $\Pi(:, 0)$ and $\Pi(0, :)$
- broadcast Y_0 along processor grid columns and rows so its distributed on each $\Pi(:, i)$ and each $\Pi(i, :)$
- compute $(I - Y_0 T_0 Y_0^T)^T A_i$ with $\Pi(i)$ for $i \in [1, P-1]$
- Q: what communication do we need for $W_0 = Y_0^T A_i$?
- A: a reduce or scatter-reduce along processor grid rows
- then transpose and broadcast W_0 so that each processor to compute $Y_0 T_0^T Y_0^T A_i$

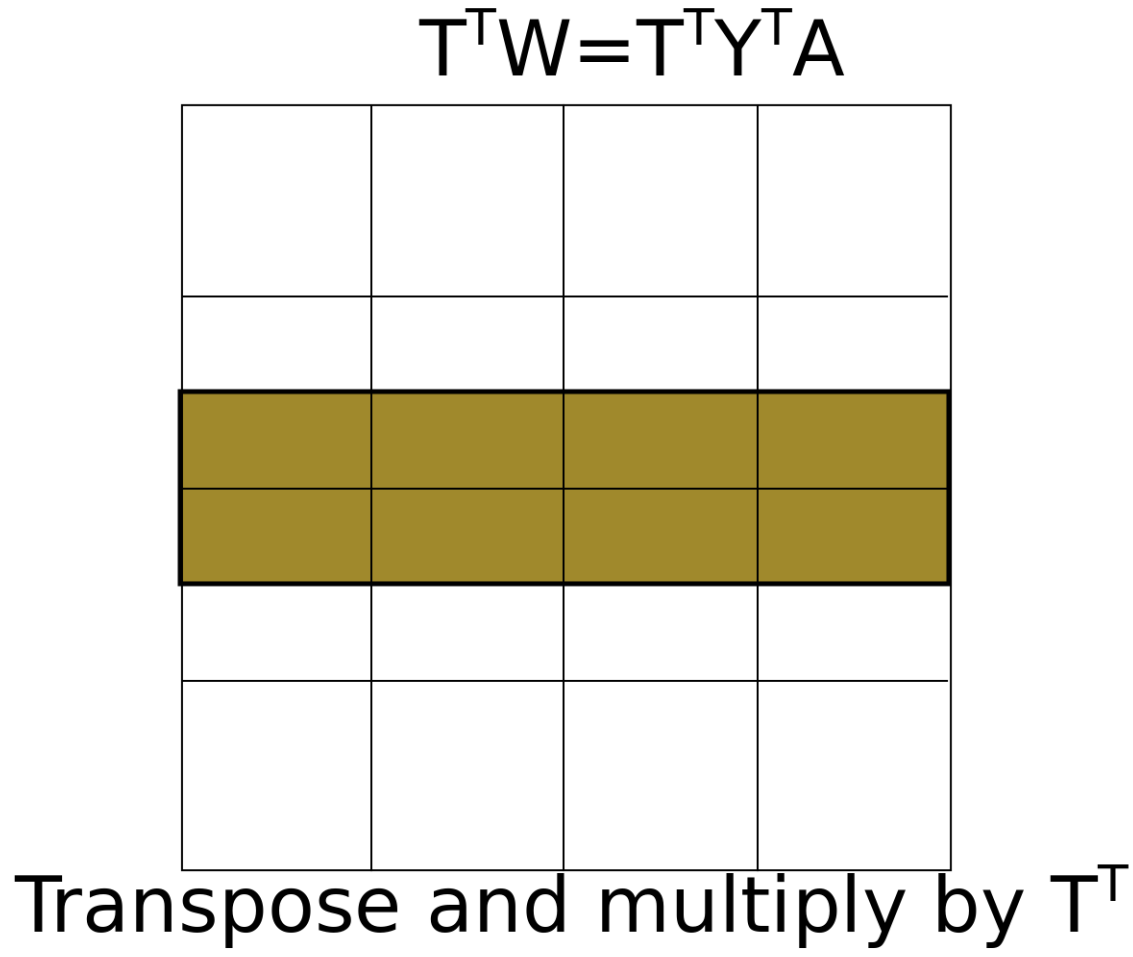
2D Householder QR, transpose and broadcast Y



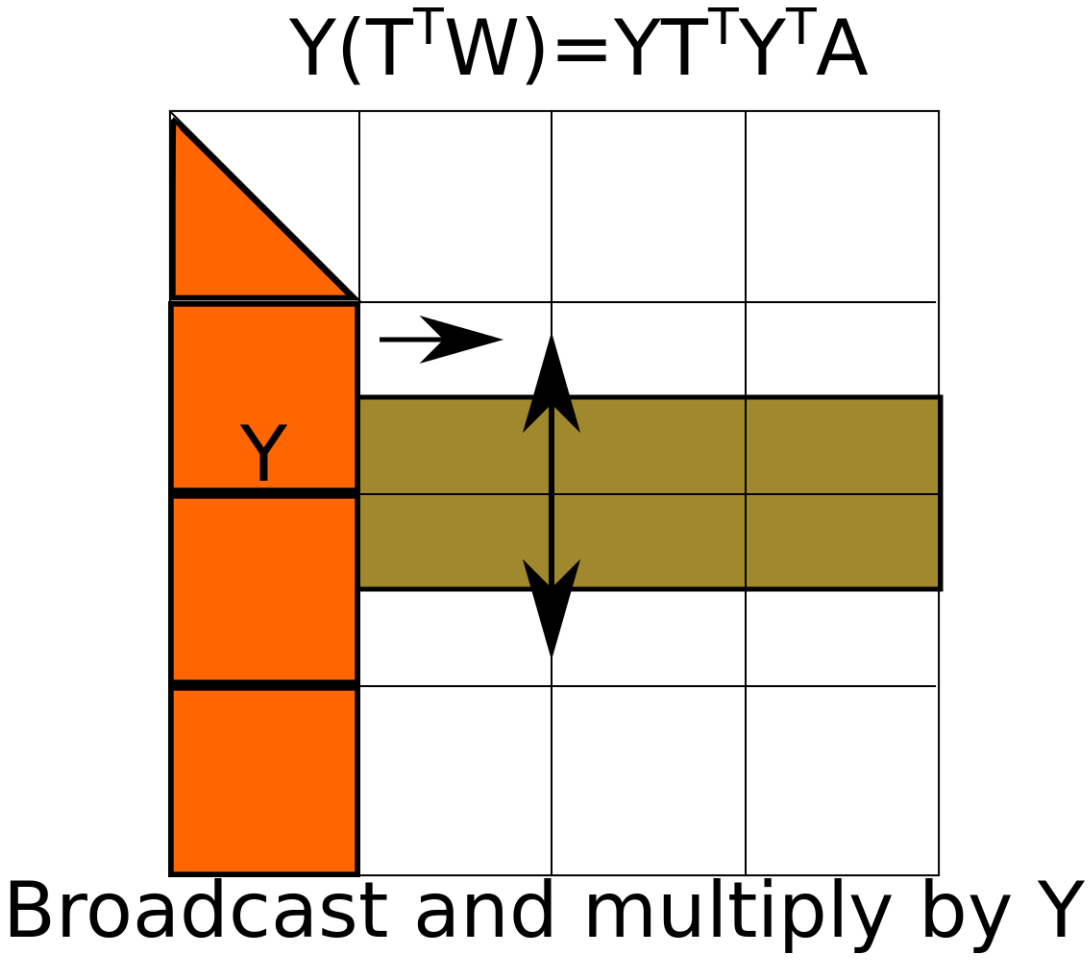
2D householder QR, reduce $W = Y^T A$



2D householder QR, transpose W and compute $T^T W$



2D householder QR, compute YT^TY^TA and subsequently $Q^TA = A - YT^TY^TA$



2D Householder QR analysis

Q: What is the bandwidth cost of the 2D Householder QR algorithm?

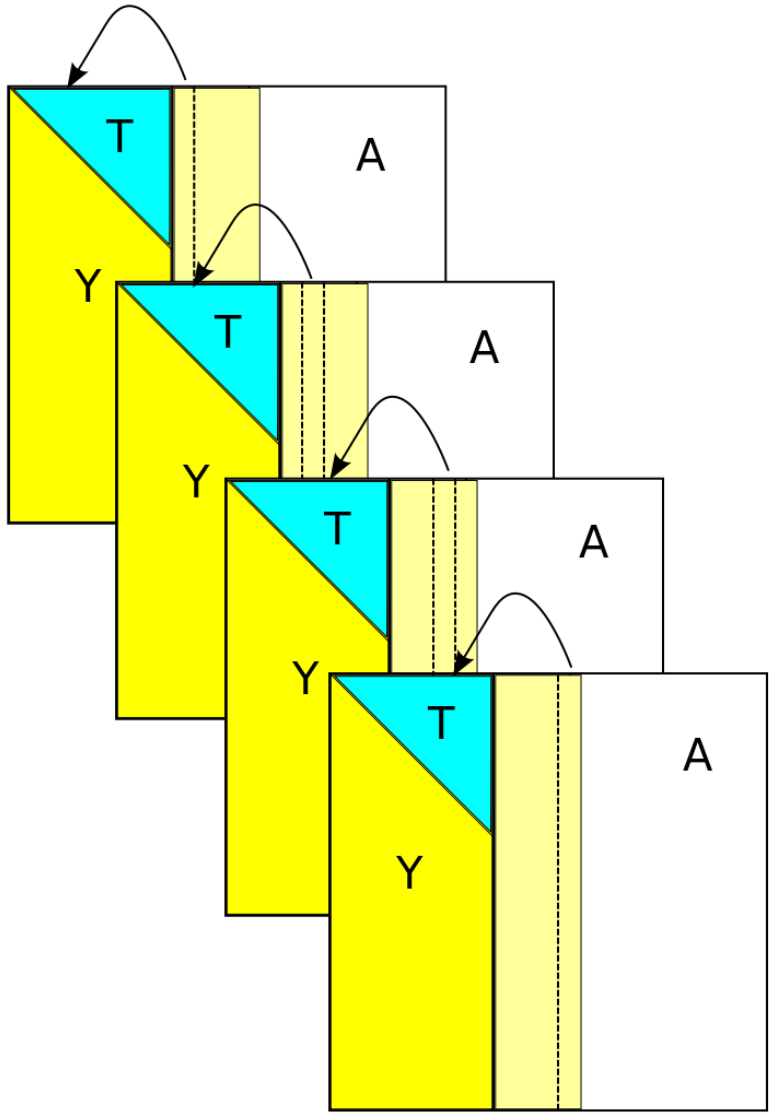
- Hint: we have $\sqrt{P}$ steps where we work with $n \times n/\sqrt{P}$ panels distributed over $\sqrt{P}$ processors
- A: $O(\frac{n^2}{\sqrt{P}} \cdot \beta)$, $O(\sqrt{P})$ collectives with $n/\sqrt{P} \times n/\sqrt{P}$ blocks
- Q And what is the synchronization cost?
- Hint: is it dominated by communicating panels or computing Householder vectors?
- A: $O(n \log(P) \cdot \alpha)$
- Recall that the 1D cost was $O(P \cdot \log(P) \cdot \alpha + n^2 \cdot \beta)$, the 2D algorithm has a lesser bandwidth cost, but a higher latency cost

2.5D Householder QR

2D Householder QR looks similar to 2D LU, so we can hope to do 2.5D Householder QR

- we'll come back to synchronization cost, first lets try to get $O(n^2/\sqrt{cp})$ bandwidth
- we can adapt the same idea of factorizing panels of sizes $n \times n/c$ using a $\sqrt{cP} \times \sqrt{P/c}$ processor grid
- however, how do we aggregate Householder updated like Schur complement updates?
- to get $Y_0^T \cdot (T_0^T Y_0^T A)$, we first need to fully compute $Y_0^T A$
- moreover, we need the fully updated $(I - Y_0 T_0^T Y_0^T)^T A$ to compute the next update by Y_1^T
- solution: delay the trailing matrix update and aggregate Y further
- terminology: left-looking algorithms (as opposed to right looking), also possible in 2D and for LU

2.5D Householder QR



A problem with some 2.5D/3D algorithms

The outlined scheme achieves $O(n^2/\sqrt{cp})$ communication like 2.5D LU

- however, it has an overhead due to a type of communication we have for now ignored...
- we are trying to multiply a large replicated Y matrix by skinny panels
- 2D algorithms multiply relatively square matrices
- if the large matrices do not fit into cache, memory bandwidth can become a bottleneck
- 2.5D matrix multiplication did not have this problem, neither did the recursive LU algorithm
- 2.5D LU as described had this problem, but it can be addressed by delaying computation of the Schur complement update aggregates
- next, we will study a 2.5D QR algorithm that uses recursion to avoid memory-bandwidth overheads

Short pause

Pairwise rotations

Givens rotations provide an alternative to Householder transformations

- a Givens rotation is given by a 2×2 matrix $Q = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$, with c, s chosen so that $QQ^T = I$ and for a given vector a, $Q^T \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$
- to obtain c and s, we can note that the desired elimination yields the condition $\begin{bmatrix} -a_1 & a_2 \\ a_2 & a_1 \end{bmatrix} \begin{bmatrix} c \\ s \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$
- LU combined with the orthogonalization condition provides a closed-form solution: $c = a_1/\sqrt{a_1^2 + a_2^2}$, $s = -a_2/\sqrt{a_1^2 + a_2^2}$
- Givens rotations have the ability to preserve sparsity and can be more parallelizable
- LU with pairwise pivoting is similar, it generates matrix $L^{-1} = \begin{bmatrix} 1 & 0 \\ -a_2/a_1 & 1 \end{bmatrix}$, so that $L^{-1}a = [\beta \quad 0]^T$
- both of these rotation matrices can be embedded into larger matrices to act on neighboring elements of a column as well as ones separated by any number of rows

Pairwise rotations

Pairwise rotations may be blocked in a few convenient ways

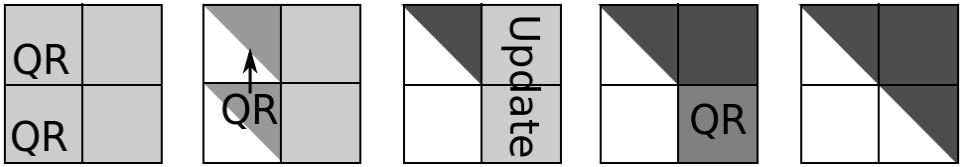
- for $n \times n$ matrix A, we can compute $A = TR$ with upper-triangular R and T orthogonal or lower triangular, by either $n(n-1)/2$ rotations or a standard method, e.g. Householder
- for $2n \times n$ matrix $\begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$, we can compute $A = T \begin{bmatrix} R \\ 0 \end{bmatrix}$ where T can be represented with $n(n+1)/2$ rotations
- these two steps suggest a recursive scheme for QR or LU with pairwise pivoting (due to Frens and Wise 2003)

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \rightarrow \begin{bmatrix} U_{11} & A_{12} \\ U_{21} & A_{22} \end{bmatrix} \rightarrow \begin{bmatrix} R_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$$

first factorize the right $n \times n/2$ submatrix via two recursive QR calls, then update the right half and compute the QR of the lower right submatrix

$$\rightarrow \begin{bmatrix} R_{11} & R_{21} \\ 0 & B_{22} \end{bmatrix} \rightarrow \begin{bmatrix} R_{11} & R_{21} \\ 0 & R_{22} \end{bmatrix}$$

Frens and Wise recursive QR



- requires four recursive calls to QR
- Q: which of these calls can be done concurrently?
- A: only the first two, three of them must be done in sequence
- therefore, the parallel cost is at least

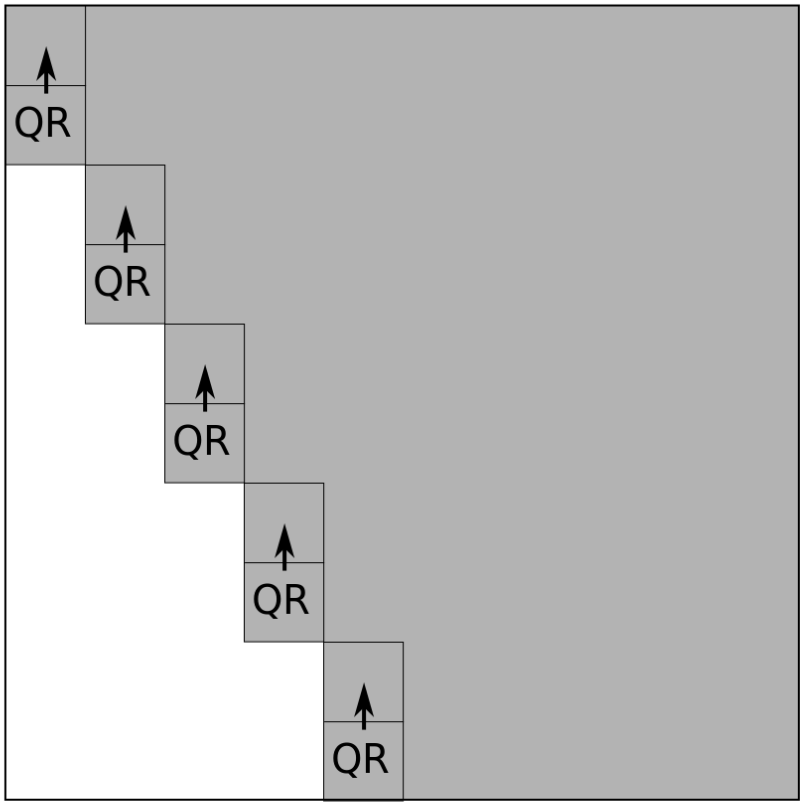
$$T_{\mathrm{FW}}(n,P) = 3T_{\mathrm{FW}}(n/2,P) + O(T_{\mathrm{MM}}(n/2,P))$$

- the cost $O(\beta \cdot n^2/\sqrt{cP})$ decreases with each level by factor of $4/3$
- after k recursive levels the base cases have a bandwidth cost of

$$T_{\mathrm{FW}}(n,P,k) \geq 3^k \cdot (n/2^k)^2 \cdot \beta = (3/4)^k n^2 \cdot \beta$$

- to get $T_{\mathrm{FW}}(n,P,k) \leq n^2/\sqrt{cP}$, we need $k = \log_{4/3}(\sqrt{cP})$
- the number of these base cases, and subsequently the synchronization cost is $2^{\log_{4/3}(\sqrt{cP})} = \sqrt{cP}^{\log_{4/3}(2)} \approx (cP)^{1.2}$
- ugly, but technically still better than $O(n \cdot \alpha)$

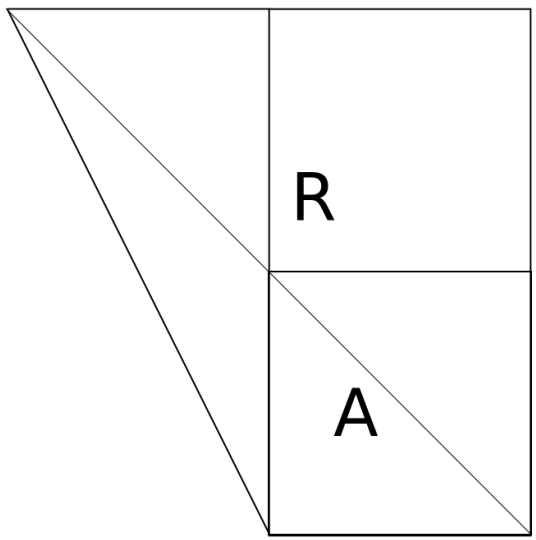
Can we do more block QR factorizations concurrently?



- yes, if we consider smaller blocks
- the wavefront of available concurrent work is slanted with a 2:1 ratio

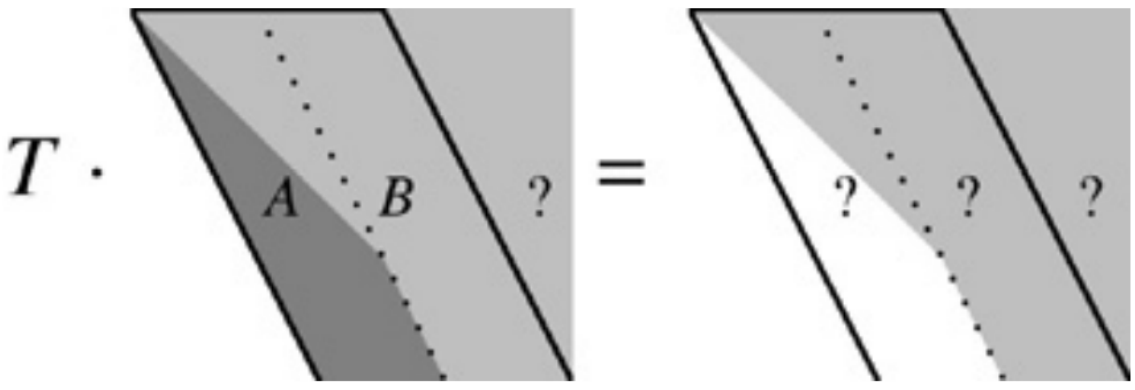
Tiskin's QR algorithm

Lets work with the 2:1 ratio directly by considering slanted panels! [Tiskin 2007]

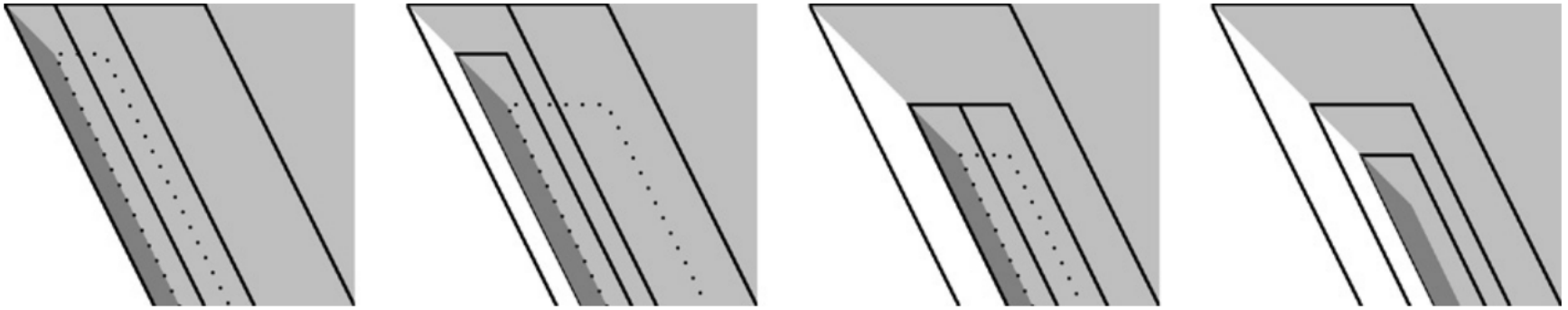


Embed the original matrix into a matrix with a slanted panel and perform QR on the slanted panel.
Represents problem so that the full wavefront (pipeline) is active.

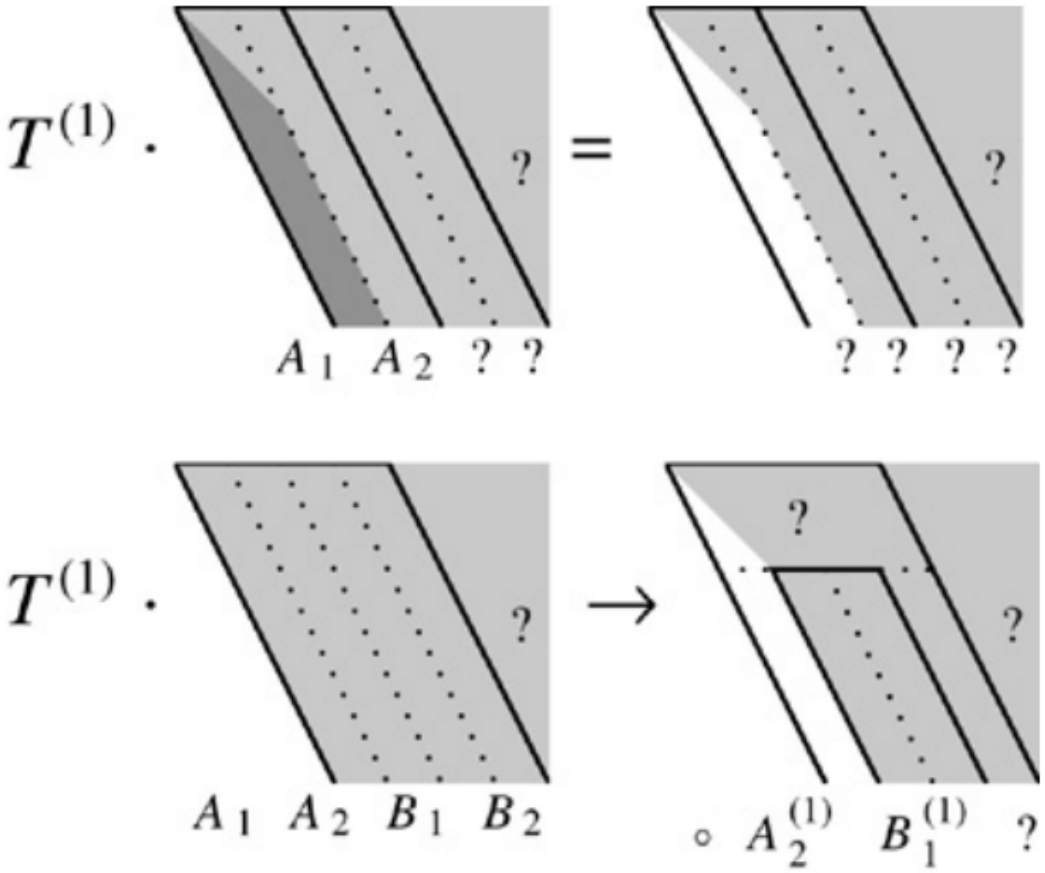
A generalized subproblem



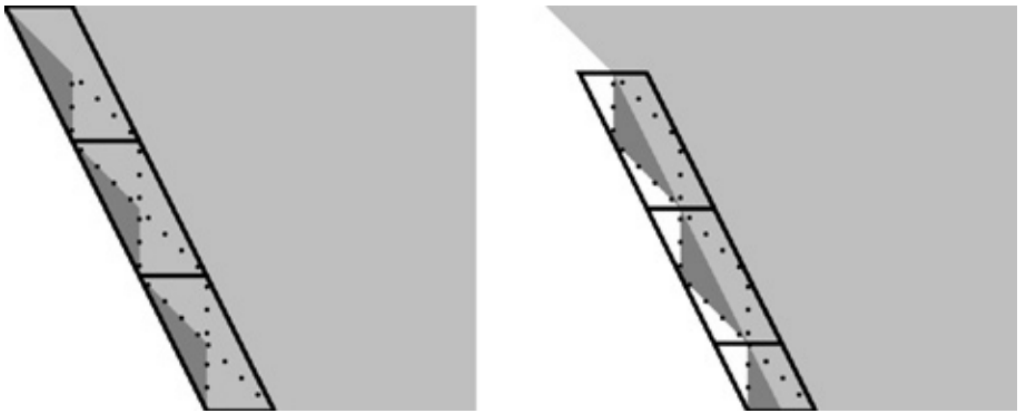
Slanted panel recursion



Slanted panel recursion, different view



Slanted panel base case



Details on Tiskin's QR algorithm

The subproblem is a slanted panel with row dimension n and width m

- the algorithm has two recursive calls with subpanels of half the width (one has a few less rows)
- the orthogonal transformation formed at each recursive step has the same structure as the eliminated subpanel
- therefore T is a matrix of dimension n with bandwidth $m/2$
- to get the next T we need to multiply the Ts obtained from subproblems
- we also need to update the trailing subpanel
- these correspond to multiplications of banded matrices, a slanted $n \times m \times m/2$ cuboid
- its possible to subdivide these multiplications into P blocks with area $O((nm^2/P)^{2/3})$

Cost analysis on Tiskin's QR algorithm

Two recursive calls and a matrix multiplication every step

- the base cases can be done in just two stages with $O(1)$ syncs.
- this is where keeping the slanted panel structure pays off, obtaining a QR factorization of a rectangular matrix with the same number of elements as slanted panel is more expensive
- overall we get the recursion with BSP cost

$$T(n, m, P) = 2T(n, m/2, P) + O(\alpha + (nm^2/P)^{2/3} \cdot \beta)$$

- bandwidth cost decreases by a factor of $2^{2/3}$ at each recursive level
- the base-cases have cost $T(n, m_0, P_0) = O(\alpha + nm_0/P_0 \cdot \beta)$, so long as $m_0 \leq n/P_0$
- we can select $m_0 = n/P^{2/3}$ and $P_0 = P^{2/3}$ to obtain the total cost

$$T_{\text{QR}} = O(P^{2/3} \cdot \alpha + n^2/P^{2/3} \cdot \beta)$$

- by doing the multiplications differently and having a larger base case we can get from $P^{2/3}$ to $\sqrt{cP}$ in both terms