Matrix

16-by-16 matrix
Blocked layout
Block-cyclic layout

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

block-cyclic layout
Cyclic layout

0 1 0 1 0 1 0 1 1
2 3 2 3 2 3 2 3
0 1 0 1 0 1 0 1 1
2 3 2 3 2 3 2 3
0 1 0 1 0 1 0 1 1
2 3 2 3 2 3 2 3
0 1 0 1 0 1 0 1 1
2 3 2 3 2 3 2 3
0 1 0 1 0 1 0 1 1
2 3 2 3 2 3 2 3

cyclic layout
Recursion with cyclic layout

All processors work until base-case

\[ T(n, P) = 2T(n/2, P) + O(\alpha \cdot \log(P) + n^2/P \cdot \beta), \]

\[ T(n_0, P) = O(\alpha \cdot \log(P) + n_0^2 \cdot \beta) \]
Rectangular QR

Consider QR factorization of $m \times n$ matrix $A$ when $m \geq n$

- so far we have focused on $m = n$
- we will first consider $m \geq nP$, then the more general case
- we can decompose $Q$ and $R$ in $A = QR$ as follows

\[ A = Q_1 Q_2 \]

\[ 0 = Q_1 \]

- $Q$: given $Q_1 R_1 = A$ and $Q_1^T Q_1 = I$, would choosing $Q_2 = 0$ yield a valid QR decomposition of $A$?
- $A$: no, it would not satisfy the orthogonality criterion, $Q^T Q = I$
- we need $Q_1 Q_2^T = 0$ and $Q_2 Q_2^T = I$; given $Q_1$, $Q_2$ is not unique
Rectangular QR for least squares

Given $m \times n$ matrix $A$ with $m \leq n$, compute $\arg \min_{x \in \mathbb{R}^n} (\|Ax - b\|_2)$

- solve $Rx = Q^T b$, i.e. $x = R^+ Q^T b$
- $R^+ = [R_1^{-1} \ 0]$ where $R_1$ is $n \times n$
- Q: given $Q_1$ (the first $n$ columns of $Q$), do we need $Q_2$?
- A: No, since, $Q^T b = \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix}$ and $[R_1^{-1} \ 0] \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix} = R_1^{-1} Q_1^T b$

- rectangular QR factorizations are also used in iterative methods such as block-Arnoldi (orthogonalization is used implicitly in many others)
- in these methods it typically suffices to have $Q_1$
Rectangular QR within square QR

QR of tall and skinny matrices is also a subroutine in square matrix factorizations

- in the last lecture, we utilized QR factorizations of matrix panels within 2D QR
- panel QR factorizations are also done for SVD and eigenvalue decompositions
- in the case of 2D QR, we apply the full $m \times m$ transformation $Q^T$
- Q: what representation could we use for $Q$ computed from an $m \times n$ matrix $A$, to compute $Q^T B$ where $B$ is $n \times k$, with $mnk$ computation?
- A: Householder: $Q = (I - YTY^T)$, where $Y$ is $m \times n$
Tall-skinny QR (TSQR)

Given an $m \times n$ matrix $A$, distributed over $P$ processors so that $\Pi(i)$ owns $A(in/P + 1 : (i + 1)n/P, :)$

- we can use Householder QR, but this requires $n \log(P)$ synchronizations
- there are a few alternative algorithms that achieve require $O(\log(P))$ synchronizations
- the simplest is probably Cholesky-QR
  - compute symmetric matrix $B = A^T A$
  - factorize $B$ using Cholesky $B = LL^T = R_1^T R_1$
  - perform ‘TRSM’ (back-substitution) $Q_1 = AR_1^{-1}$
  - cheap but not stable, $\text{cond}(B) = \text{cond}(A)^2$, so radical instability when $\text{cond}(A) \geq 1/\sqrt{\epsilon_{\text{mach}}}$
  - orthogonality of $Q$ is often poor
**Cholesky QR2**

Cholesky-QR can be made more stable [Yamamoto et al 2014]

- as before, compute \( \{ \bar{Q}_1, \bar{R}_1 \} = \text{Cholesky-QR}(A) \)
- then, iterate! \( \{ Q_1, \hat{R}_1 \} = \text{Cholesky-QR}(\bar{Q}_1) \)
- \( R_1 = \hat{R}_1 \bar{R}_1 \)
- \( A = Q_1 R_1 \)
- solution still bad when \( \text{cond}(A) \geq 1/\sqrt{\epsilon_{\text{mach}}} \)
- but if \( \text{cond}(A) < 1/\sqrt{\epsilon_{\text{mach}}} \), it is numerically stable because \( \text{cond}(\bar{Q}_1) \approx 1 \)
- parallel Cholesky-QR2
  1. perform \( A^T A \) using an allreduce of size \( n^2/2 \)
  2. compute Cholesky redundantly and TRSM to get \( \bar{Q}_1 \) and \( \bar{R}_1 \)
  3. perform \( \bar{Q}_1^T \bar{Q}_1 \) using an allreduce of size \( n^2/2 \)
  4. compute Cholesky redundantly, TRSM, and \( R_1 = \hat{R}_1 \bar{R}_1 \) to get \( Q_1, R_1 \)
  5. \( T_{\text{Cholesky-QR2}}(m, n, P) = 2T_{\text{allred}}(n^2/2, P) = 2n^2 \cdot \beta + 4 \log_2(P) \cdot \alpha \)

- for QR of a tall-skinny \( A \) with \( \text{cond}(A) < 1/\sqrt{\epsilon_{\text{mach}}} \), this algorithm is trivial to implement, stable, and very fast
Recursive TSQR

Block Givens rotations yield another idea

- we can also employ a recursive scheme analogous to tournament pivoting for LU
- subdivide \( A = \begin{bmatrix} A_U \\ A_L \end{bmatrix} \) and recursively compute \( \{ Q_U, R_U \} = QR(A_U), \{ Q_L, R_L \} = QR(A_L) \) concurrently with \( P/2 \) processors each
- we have \( A = \begin{bmatrix} Q_U & R_U \\ Q_L & R_L \end{bmatrix} = \begin{bmatrix} Q_U & 0 \\ 0 & Q_L \end{bmatrix} \begin{bmatrix} R_U \\ R_L \end{bmatrix} \)
- (all)gather \( R_U \) and \( R_L \) and compute sequentially, \( \begin{bmatrix} R_U \\ R_L \end{bmatrix} = \tilde{Q} R \)
- we now have \( A = QR \) where \( Q = \begin{bmatrix} Q_U & 0 \\ 0 & Q_L \end{bmatrix} \tilde{Q} \)
Recursive TSQR, binary tree (binomial comm. pattern)

Householder vectors are denoted in yellow ($R$ is $R_1$)
Recursive TSQR, butterfly, redundant $R$ computation

Householder vectors are denoted in yellow ($R$ is $R_1$)
Cost analysis of recursive TSQR, butterfly

We can subdivide the cost into base cases (tree leaves) and internal nodes

- let the cost per flop be $\gamma$
- every processor computes a QR of their $m/P \times n$ leaf matrix block

$$T_{\text{Rec-TSQR}}(m_0, n, 1) = m_0 n^2 \cdot \gamma$$

- Q: what cost do we incur at every tree node

$$T_{\text{Rec-TSQR}}(m, n, P) = T_{\text{Rec-TSQR}}(m/2, n, P/2) + O(?)$$

- A: $O(n^3 \cdot \gamma + n^2 \cdot \beta + \alpha)$, for a total cost of

$$T_{\text{Rec-TSQR}}(m, n, P) = O([mn^2/P + n^3 \log(P)] \cdot \gamma + n^2 \log(P) \cdot \beta + \log(P) \cdot \alpha)$$

- Q: How does this bandwidth cost compare to Cholesky-QR2?
- Hint: the communication cost of Cholesky-QR2 is $2T_{\text{allreduce}}(n^2/2, P)$
- A: The cost of recursive TSQR is a factor of $O(\log(P))$ greater.
Computing $Q_1$ in recursive TSQR

Let's now consider how to compute the $m \times n$ set of orthonormal columns $Q_1$ such that $A = Q_1 R_1$ for $n \times n$ upper-triangular $R_1$

- we had the recurrence $Q = \begin{bmatrix} Q_U & 0 \\ 0 & Q_L \end{bmatrix} \tilde{Q}$
- these orthogonal factors: $Q_L, Q_U, \tilde{Q}$ have a lot of structure, especially if represented with Householder vectors or Givens rotations
- Q: how do we compute $Q$ when performing regular Givens rotations?
- A: by applying them to an identity matrix, similar idea here...
- instead of computing the full $m \times m$ matrix $Q$ (which really, we never want explicitly), we can apply the implicit representation of $Q$ to $\begin{bmatrix} I \\ 0 \end{bmatrix}$ where $I$ is $n \times n$ to get $Q_1$
- this has the same cost as the tree for computing $R$, except now we do it backwards
Computing $Q_1$ in recursive TSQR
Short pause
Homeworks and projects

- any questions on homework problems?
- office hours Tuesday 3-4
- posts on Piazza on late Tuesday evening may not get a response until Wednesday morning
- first project proposal due Sep 28th, email me or stop by to discuss preliminary ideas
Recursive TSQR within a 2D algorithm

Consider using recursive TSQR for $n \times b$ panel factorizations to factorize an $n \times n$ matrix using a 2D algorithm

- each of $n/b$ TSQRs would have cost

$$T_{\text{Rec-TSQR}}(n, b, \sqrt{P}) = O(b^2 \log(P) \cdot \beta + \log(P) \cdot \alpha)$$

- Q: if we want to achieve a bandwidth cost of $O(n^2/\sqrt{P} \cdot \beta)$ in the entire 2D algorithm, how does Rec-TSQR restrict our choice of $b$?

  A: $b \leq \frac{n}{\sqrt{P} \log(P)}$

- to perform trailing matrix updates, we need to multiply by $Q^T$, where we can again use its implicit tree representation

- Q: would we need to traverse the tree from the leaves to the root, as we did when computing $R$, or from the root to the leaves as we did for computing $Q_1$?

  A: from the leaves to the root, since

$$Q^T = \left( \begin{bmatrix} Q_U & 0 \\ 0 & Q_L \end{bmatrix} \tilde{Q} \right)^T = \tilde{Q}^T \begin{bmatrix} Q_U^T & 0 \\ 0 & Q_L^T \end{bmatrix}$$
Apply implicit $Q^T$ via binary tree
Cost analysis of applying $Q^T$ via binary tree

We need to apply $Q^T$ for each panel, $n/b$ times

- every time, we need to update up to $n - b = O(n)$ columns
- the cost of the update done in the tree leaves is
  \[
  O\left(\frac{n^2 b}{P} \cdot \gamma + \frac{nb}{\sqrt{P}} \cdot \beta + \log(P) \cdot \alpha\right)
  \]

- for every tree node, we need to communicate the $b$ updated rows, a block of dimension proportional to $b \times n/\sqrt{P}$
- Q: what is then the bandwidth cost of whole tree update?
- A: $O(nb \log(P) / \sqrt{P} \cdot \beta)$, the tree nodes cost:
  \[
  O\left(\frac{nb^2 \log(P)}{P} \cdot \gamma + \frac{nb \log(P)}{\sqrt{P}} \cdot \beta + \log(P) \cdot \alpha\right)
  \]

- since there are $n/b$ such updates, the 2D algorithm would have a bandwidth cost of at least $O\left(\frac{n^2 \log(P)}{P} \cdot \beta\right)$
Apply implicit $Q^T$ via butterfly

Subdivide updated columns recursively to keep all processors busy

$$T(b, n, P) = T(b, n/2, P/2) + O\left(\frac{nb^2}{P} \cdot \gamma + \beta \cdot \frac{nb}{\sqrt{P}} + \alpha\right)$$
Apply implicit $Q^T$ via butterfly

After recursion, return the columns back to owner, for a total cost of

$$T(b, n, P) = O\left(\frac{nb^2}{P} \cdot \gamma + \beta \cdot \frac{nb}{\sqrt{P}} + \alpha \cdot \log(P)\right)$$
Motivation for Householder reconstruction

The trailing matrix update in Householder QR is still the most efficient
- consists of $O(1)$ matrix multiplications
- requires standard collective communication, rather than an algorithmic tree
- compliant with standard libraries (ScaLAPACK returns $Y$ not $Q$ for dgeqrf)
- moreover, how do we do a trailing matrix update with Cholesky-QR2?
Householder reconstruction

Given $m \times n$ matrix $Q_1$, we can construct $Y$ such that $Q = (I - YTY^T) = [Q_1, Q_2]$ and $Q$ is orthogonal

- key idea due to Yusaku Yamamoto (2013)
- note that in the Householder representation, we have $I - Q = Y \cdot TY^T$, where $Y$ is lower-trapezoidal and $TY^T$ is upper-trapezoidal

- let $Q_1 = \begin{bmatrix} Q_{11} \\ Q_{21} \end{bmatrix}$ where $Q_{11}$ is $n \times n$, compute

\[
\{Y, TY_1^T\} = LU\left(\begin{bmatrix} I - Q_{11} \\ Q_{21} \end{bmatrix}\right),
\]

where $Y_1$ is the upper-triangular $n \times n$ leading block of $Y^T$
Householder reconstruction stability

Householder reconstruction can be done with unconditional stability

- we need to be just a little more careful

\[
\{Y, TY_1^T\} = LU\left(\begin{bmatrix} \vphantom{-Q_{11}} S - Q_{11} \\ Q_{21} \end{bmatrix}\right),
\]

where $S$ is a sign matrix (each value in $\{-1, 1\}$) with values picked to match the sign of the diagonal entry within $LU$

- these are the sign choices we need to make for regular Householder factorization

- since all entries of $Q_1$ are $\leq 1$, pivoting is unnecessary (partial pivoting would do nothing)

- since $\text{cond}(Q) \approx 1$, Householder reconstruction is stable
Householder reconstruction for square matrix factorizations

Householder reconstruction provides a kind of abstraction between the panel factorization and trailing matrix update

- use algorithm of choice for panel QR, e.g. Cholesky-QR(2) or recursive TSQR
- construct $Q_1$ and reconstruct $Y$
  - construction of $Q_1$ should cost no more than the factorization itself
  - performing LU of $Q_1$ requires a sequential $n \times n$ LU and a broadcast of the $U$ factor for TRSM
- now perform trailing matrix update as if we had done Householder QR
- so we can achieve same bandwidth costs as in previous lecture, but lower synchronization cost ($O(\sqrt{cP} \cdot \alpha)$)
- for recursive TSQR, extra factor of $\log(P)$ in bandwidth cost requires a block size smaller by a factor of $\log(P)$, yielding $\log(P)$ higher synchronization cost than if we use Cholesky-QR2
QR for rectangular matrices

What if we want to factorize an $m \times n$ rectangular matrix, where $m > n$, but not $m \gg n$

- TSQR algorithms have cost factors of $O(n^3 \cdot \gamma + n^2 \cdot \beta)$ or higher, which may be problematic
- 2D and 3D algorithms have assumed $m = n$
- there are a couple of alternative approaches for the general case
- intuitively, we want to use processor grids that match the dimensions of the $m \times n \times n$ problem
Elmroth-Gustavson algorithm (3Dx2Dx1D)

One approach is to use column-recursion $A = [A_1, A_2]$

- compute $\{Y_1, T_1, R_1\} = \text{QR}(A_1)$ recursively with $P$ processors
- perform rectangular matrix multiplications with communication-avoiding algorithms to compute $B_2 = (I - Y_1T_1Y_1^T)^TA_2$
  - compute $\{Y_2, T_2, R_2\} = \text{QR}(B_{22})$ where $B_2 = \begin{bmatrix} R_{12} \\ B_{22} \end{bmatrix}$ recursively
- concatenate $Y_1$ and $Y_2$ into $Y$ and compute $T$ from $Y$ via rectangular matrix multiplication
- output $\{Y, T, \begin{bmatrix} R_1 & R_{12} \\ 0 & R_2 \end{bmatrix} \}$
- pick an appropriate number of columns for a TSQR base-case
Elmroth-Gustavson algorithm (1Dx2Dx3D)

Another approach is to use “row-recursion”

- perform recursive TSQR, where each node in the tree is factorized with $Pn/m$ processors (if $P \geq m/n$, a TSQR algorithm is the best option anyway)
- leaf nodes will require just a square QR
- tree nodes require QR of two stacked upper-triangular matrices
- interleave the rows of the upper-triangular matrices and you get a 2 : 1 ratio, i.e. slanted panel, so can use Tiskin’s QR algorithm without embedding!

- both of the proposed approaches achieve a bandwidth cost of $O\left(\left(\frac{mn^2}{P}\right)^{2/3} \log(P)\right)$ for $n \leq m \leq nP$