Parallel Numerical Algorithms

Chapter 5 – Eigenvalue Problems
Section 5.1 – QR Factorization

Michael T. Heath and Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512

Outline

QR Factorization

- For given $m \times n$ matrix $\boldsymbol{A}$, with $m > n$, *QR factorization* has form

$$\boxed{\boldsymbol{A} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix}}$$

  where matrix $\boldsymbol{Q}$ is $m \times n$ with orthonormal columns, and $\boldsymbol{R}$ is $n \times n$ and upper triangular

- Can be used to solve linear systems, least squares problems, and eigenvalue problems

- As with Gaussian elimination, zeros are introduced successively into matrix $\boldsymbol{A}$, eventually reaching upper triangular form, but using orthogonal transformations instead of elementary eliminators

Methods for QR Factorization

- Householder transformations (elementary reflectors)
- Givens transformations (plane rotations)
- Gram-Schmidt orthogonalization

Householder Transformations

- *Householder transformation* has form

$$\boldsymbol{H} = \boldsymbol{I} - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}$$

  where $\boldsymbol{v}$ is nonzero vector

- From definition, $\boldsymbol{H} = \boldsymbol{H}^T = \boldsymbol{H}^{-1}$, so $\boldsymbol{H}$ is both orthogonal and symmetric

- For given vector $\boldsymbol{a}$, choose $\boldsymbol{v}$ so that

$$\boldsymbol{H}\boldsymbol{a} = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \boldsymbol{e}_1$$

Householder Transformations

- Substituting into formula for $\boldsymbol{H}$, we see that we can take

$$\boldsymbol{v} = \boldsymbol{a} - \alpha \boldsymbol{e}_1$$

  and to preserve norm we must have $\alpha = \pm\|\boldsymbol{a}\|_2$, with sign chosen to avoid cancellation

Householder QR Factorization

for $k = 1$ **to** n

$$\alpha_k = -\text{sign}(a_{kk})\sqrt{a_{kk}^2 + \cdots + a_{mk}^2}$$

$$\boldsymbol{v}_k = \begin{bmatrix} 0 & \cdots & 0 & a_{kk} & \cdots & a_{mk} \end{bmatrix}^T - \alpha_k \boldsymbol{e}_k$$

$$\beta_k = \boldsymbol{v}_k^T \boldsymbol{v}_k$$

if $\beta_k = 0$ **then**

continue with next k

for $j = k$ **to** n

$$\gamma_j = \boldsymbol{v}_k^T \boldsymbol{a}_j$$

$$\boldsymbol{a}_j = \boldsymbol{a}_j - (2\gamma_j/\beta_k)\boldsymbol{v}_k$$

end

end

Basis-Kernel Representations

- A Householder matrix $\boldsymbol{H}$ is represented by $\boldsymbol{H} = \boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^T$, i.e. a rank-1 perturbation of the identity
- We can combine r Householder matrices $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_r$ into a rank-r peturbation of the identity

$$\bar{\boldsymbol{H}} = \prod_{i=1}^{r} \boldsymbol{H}_i = \boldsymbol{I} - \boldsymbol{Y}\boldsymbol{V}^T, \text{where } \boldsymbol{Y}, \boldsymbol{V} \in \mathbb{R}^{n \times r}$$

- Often, $\boldsymbol{V} = \boldsymbol{Y}\boldsymbol{T}$ where $\boldsymbol{T}$ is upper-triangular and $\boldsymbol{Y}$ is lower-triangular, yielding

$$\bar{\boldsymbol{H}} = \boldsymbol{I} - \boldsymbol{Y}\boldsymbol{T}^T\boldsymbol{Y}^T$$

- If $\boldsymbol{H}_i = \boldsymbol{I} - \boldsymbol{y}_i\boldsymbol{y}_i^T$, then the ith column of $\boldsymbol{Y}$ is $\boldsymbol{y}_i$, while $\boldsymbol{T}$ is defined by $\boldsymbol{T}^{-1} + \boldsymbol{T}^{-T} = \boldsymbol{Y}^T\boldsymbol{Y}$

Parallel Householder QR

- A basis kernel representation of Householder transformations, allows us to update a trailing matrix $\boldsymbol{B}$ as

$$\bar{\boldsymbol{H}}\boldsymbol{B} = (\boldsymbol{I} - \boldsymbol{Y}\boldsymbol{T}^T\boldsymbol{Y}^T)\boldsymbol{B} = \boldsymbol{B} - \boldsymbol{Y}(\boldsymbol{T}^T(\boldsymbol{Y}^T\boldsymbol{B}))$$

  with cost $O(n^2r)$
- Performing such updates is essentially as hard as Schur complement updates in LU
- Forming Householder vector $\boldsymbol{v}_k$ is also analogous to computing multipliers in Gaussian elimination
- Thus, parallel implementation is similar to parallel LU, but with Householder vectors broadcast horizontally instead of multipliers

Panel QR Factorization

- Finding Householder vector $\boldsymbol{y}_i$ requires computation of the norm of the leading vector of the ith trailing matrix, creating a latency bottleneck much like that of pivot row selection in partial pivoting
- Other methods need $L = \Theta(\log(p))$ rather than $\Theta(n)$ msgs
- For example Cholesky-QR and Cholesky-QR2 perform $\boldsymbol{R} = \text{Cholesky}(\boldsymbol{A}^T\boldsymbol{A})$, $\boldsymbol{Q} = \boldsymbol{A}\boldsymbol{R}^{-1}$ (Cholesky-QR2 does one step of refinement), requiring only a single allreduce, but losing stability
- Unconditional stability and $O(\log(p))$ messages achieved by TSQR algorithm with row-wise recursion (akin to tournament pivoting)
- Basis-kernel representation can be recovered by constructing first r columns of $\bar{\boldsymbol{H}}$

Cholesky QR2

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

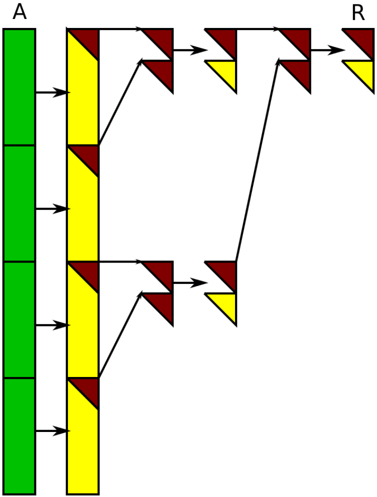
- As before, compute $\{\bar{\boldsymbol{Q}}, \bar{\boldsymbol{R}}\} =$ Cholesky-QR$(\boldsymbol{A})$
- Then, iterate $\{\boldsymbol{Q}, \hat{\boldsymbol{R}}\} =$ Cholesky-QR$(\bar{\boldsymbol{Q}})$
- $\boldsymbol{R} = \hat{\boldsymbol{R}}\bar{\boldsymbol{R}}$
- $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$
- Solution still bad when $\kappa(\boldsymbol{A}) \geq 1/\sqrt{\epsilon_{\text{mach}}}$
- But if $\kappa(\boldsymbol{A}) < 1/\sqrt{\epsilon_{\text{mach}}}$, it is numerically stable because $\kappa(\bar{\boldsymbol{Q}}) \approx 1$
- For QR of a tall-skinny A with $\kappa(A) < 1/\sqrt{\epsilon_{\text{mach}}}$, this algorithm is easy to implement, stable, and scalable

Recursive TSQR

Block Givens rotations yield another idea

- We can also employ a recursive scheme analogous to tournament pivoting for LU
- Subdivide $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_U \\ \boldsymbol{A}_L \end{bmatrix}$ and recursively compute $\{\boldsymbol{Q}_U, \boldsymbol{R}_U\} = \boldsymbol{QR}(\boldsymbol{A}_U)$, $\{\boldsymbol{Q}_L, \boldsymbol{R}_L\} = \boldsymbol{QR}(\boldsymbol{A}_L)$ concurrently with $P/2$ processors each
- We have $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{Q}_U\boldsymbol{R}_U \\ \boldsymbol{Q}_L\boldsymbol{R}_L \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_U & \\ & \boldsymbol{Q}_L \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_U \\ \boldsymbol{R}_L \end{bmatrix}$
- Gather $\boldsymbol{R}_U$ and $\boldsymbol{R}_L$ and compute sequentially, $\begin{bmatrix} \boldsymbol{R}_U \\ \boldsymbol{R}_L \end{bmatrix} = \tilde{\boldsymbol{Q}}\boldsymbol{R}$
- We now have $\boldsymbol{A} = \boldsymbol{QR}$ where $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_U & \\ & \boldsymbol{Q}_L \end{bmatrix} \tilde{\boldsymbol{Q}}$

Recursive TSQR, Binary (Binomial) Tree



Cost Analysis of Recursive TSQR

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

- Every processor computes a QR of their $m/P \times n$ leaf matrix block

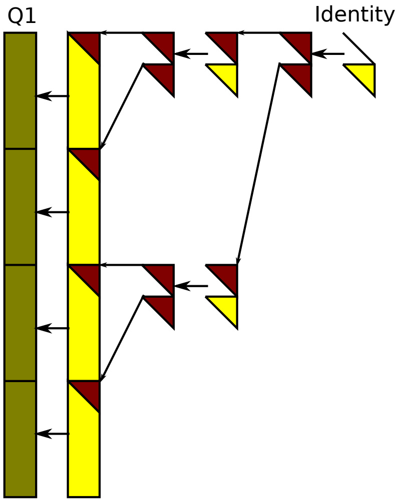
$$T_{\text{Rec-TSQR}}(m,n,P) = T_{\text{Rec-TSQR}}(nP,n,1) + (m/P)n^2 \cdot \gamma$$

- Subsequently for each tree node, each processor we sends/receives a message of size $O(n^2)$ and performs $O(n^3)$ work to factorize $2n \times n$ matrix
- The total cost is

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

- Communication cost is higher than of Cholesky-QR2, which is $2T_{\text{allreduce}}(n^2/2,P) = O(n^2\beta + \log(P)\alpha)$

Recovering Q in Recursive TSQR



Householder Reconstruction

Given $m \times n$ matrix $\boldsymbol{Q}_1$, we can construct $\boldsymbol{Y}$ such that $\boldsymbol{Q} = (\boldsymbol{I} - \boldsymbol{Y}\boldsymbol{T}\boldsymbol{Y}^T) = [\boldsymbol{Q}_1, \boldsymbol{Q}_2]$ and $\boldsymbol{Q}$ is orthogonal

- note that in the Householder representation, we have $\boldsymbol{I} - \boldsymbol{Q} = \boldsymbol{Y} \cdot \boldsymbol{T}\boldsymbol{Y}^T$, where $\boldsymbol{Y}$ is lower-trapezoidal and $\boldsymbol{T}\boldsymbol{Y}^T$ is upper-trapezoidal
- Let $\boldsymbol{Q}_1 = \begin{bmatrix} \boldsymbol{Q}_{11} \\ \boldsymbol{Q}_{21} \end{bmatrix}$ where $\boldsymbol{Q}_{11}$ is $n \times n$, compute

$$\{\boldsymbol{Y}, \boldsymbol{T}\boldsymbol{Y}_1^T\} = \text{LU}\left(\begin{bmatrix} \boldsymbol{I} - \boldsymbol{Q}_{11} \\ \boldsymbol{Q}_{21} \end{bmatrix} \right),$$

where $\boldsymbol{Y}_1$ is the upper-triangular $n \times n$ leading block of $\boldsymbol{Y}^T$

Householder Reconstruction Stability

Householder reconstruction can be done with unconditional stability

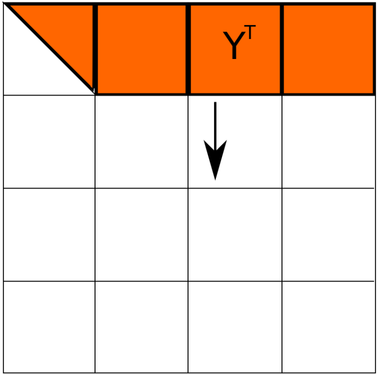
- We need to be just a little more careful

$$\{\boldsymbol{Y}, \boldsymbol{T}\boldsymbol{Y}_1^T\} = \mathsf{LU}\left(\begin{bmatrix} \boldsymbol{S} - \boldsymbol{Q}_{11} \\ \boldsymbol{Q}_{21} \end{bmatrix} \right),$$

  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 $\boldsymbol{Q}$ are ≤ 1, pivoting is unnecessary (partial pivoting would do nothing)
- Since $\kappa(\boldsymbol{Q}) \approx 1$, Householder reconstruction is stable

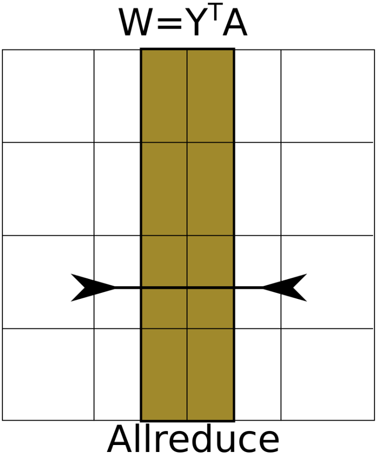
2D Householder QR, Basis-Kernel Representation

Transpose and Broadcast Y



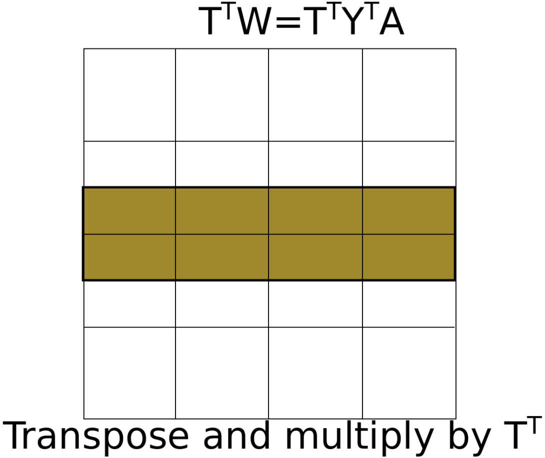
2D Householder QR, Basis-Kernel Representation

Reduce $\boldsymbol{W} = \boldsymbol{Y}^T\boldsymbol{A}$



2D Householder QR, Basis-Kernel Representation

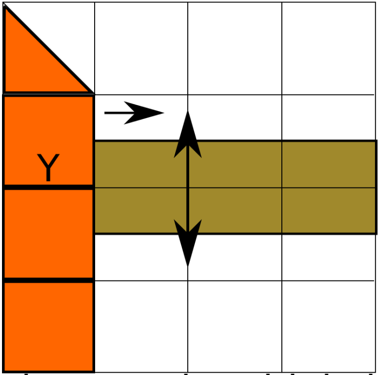
Transpose W and Compute $T^T W$



2D Householder QR, Trailing Matrix Update

Compute $\boldsymbol{Y}\boldsymbol{T}^T\boldsymbol{Y}^T\boldsymbol{A}$ and subsequently $\boldsymbol{Q}^T\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{Y}\boldsymbol{T}^T\boldsymbol{Y}^T\boldsymbol{A}$

$$Y(T^T W) = Y T^T Y^T A$$



Broadcast and multiply by Y

Elmroth-Gustavson Algorithm (3Dx2Dx1D)

One approach is to use column-recursion $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2]$

- Compute $\{\boldsymbol{Y}_1, \boldsymbol{T}_1, \boldsymbol{R}_1\} = \text{QR}(\boldsymbol{A}_1)$ recursively with P processors
- Perform rectangular matrix multiplications with communication-avoiding algorithms to compute $\boldsymbol{B}_2 = (\boldsymbol{I} - \boldsymbol{Y}_1\boldsymbol{T}_1\boldsymbol{Y}_1^T)^T\boldsymbol{A}_2$
- Compute $\{\boldsymbol{Y}_2, \boldsymbol{T}_2, \boldsymbol{R}_2\} = \text{QR}(\boldsymbol{B}_{22})$ where $\boldsymbol{B}_2 = \begin{bmatrix} \boldsymbol{R}_{12} \\ \boldsymbol{B}_{22} \end{bmatrix}$ recursively
- Concatenate $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ into $\boldsymbol{Y}$ and compute $\boldsymbol{T}$ from $\boldsymbol{Y}$ via rectangular matrix multiplication
- Output $\left\{\boldsymbol{Y}, \boldsymbol{T}, \begin{bmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_{12} \\ & \boldsymbol{R}_2 \end{bmatrix}\right\}$
- Pick an appropriate number of columns for a TSQR base-case

Givens Rotations

- *Givens rotation* operates on pair of rows to introduce single zero

- For given 2-vector $\boldsymbol{a} = [a_1 \;\; a_2]^T$, if

$$c = \frac{a_1}{\sqrt{a_1^2 + a_2^2}}, \qquad s = \frac{a_2}{\sqrt{a_1^2 + a_2^2}}$$

  then

$$\boldsymbol{Ga} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}$$

- Scalars c and s are cosine and sine of angle of rotation, and $c^2 + s^2 = 1$, so $\boldsymbol{G}$ is orthogonal

Givens QR Factorization

- Givens rotations can be systematically applied to successive pairs of rows of matrix $\boldsymbol{A}$ to zero entire strict lower triangle
- Subdiagonal entries of matrix can be annihilated in various possible orderings (but once introduced, zeros should be preserved)
- Each rotation must be applied to all entries in relevant pair of rows, not just entries determining c and s
- Once upper triangular form is reached, product of rotations, $\boldsymbol{Q}$, is orthogonal, so we have QR factorization of $\boldsymbol{A}$

Parallel Givens QR Factorization

- With 1-D partitioning of A by columns, parallel implementation of Givens QR factorization is similar to parallel Householder QR factorization, with cosines and sines broadcast horizontally and each task updating its portion of relevant rows
- With 1-D partitioning of A by rows, broadcast of cosines and sines is unnecessary, but there is no parallelism unless multiple pairs of rows are processed simultaneously
- Fortunately, it is possible to process multiple pairs of rows simultaneously without interfering with each other

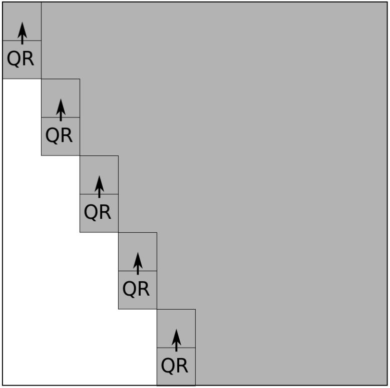
Parallel Givens QR Factorization

- Stage at which each subdiagonal entry can be annihilated is shown here for 8×8 example

$$\begin{bmatrix} \times & & & & & & & \\ 7 & \times & & & & & & \\ 6 & 8 & \times & & & & & \\ 5 & 7 & 9 & \times & & & & \\ 4 & 6 & 8 & 10 & \times & & & \\ 3 & 5 & 7 & 9 & 11 & \times & & \\ 2 & 4 & 6 & 8 & 10 & 12 & \times & \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & \times \end{bmatrix}$$

- Maximum parallelism is $n/2$ at stage $n-1$ for $n \times n$ matrix

Parallel Givens QR Wavefront



Parallel Givens QR Factorization

- Communication cost is high, but can be reduced by having each task initially reduce its entire local set of rows to upper triangular form, which requires no communication
- Then, in subsequent phase, task pairs cooperate in annihilating additional entries using one row from each of two tasks, exchanging data as necessary
- Various strategies can be used for combining results of first phase, depending on underlying network topology
- Parallel partitioning with slanted-panels (slope -2) achieve same scalablility as parallel algorithms for LU without pivoting (see [Tiskin 2007])

Parallel Givens QR Factorization

- With 2-D partitioning of A, parallel implementation combines features of 1-D column and 1-D row algorithms
- In particular, sets of rows can be processed simultaneously to annihilate multiple entries, but updating of rows requires horizontal broadcast of cosines and sines

References

- E. Chu and A. George, QR factorization of a dense matrix on a hypercube multiprocessor, *SIAM J. Sci. Stat. Comput.* 11:990-1028, 1990
- M. Cosnard, J. M. Muller, and Y. Robert, Parallel QR decomposition of a rectangular matrix, *Numer. Math.* 48:239-249, 1986
- M. Cosnard and Y. Robert, Complexity of parallel QR factorization, *J. ACM* 33:712-723, 1986
- E. Elmroth and F. G. Gustavson, Applying recursion to serial and parallel QR factorization leads to better performance, *IBM J. Res. Develop.* 44:605-624, 2000

References

- B. Hendrickson, Parallel QR factorization using the torus-wrap mapping, *Parallel Comput.* 19:1259-1271, 1993.
- F. T. Luk, A rotation method for computing the QR-decomposition, *SIAM J. Sci. Stat. Comput.* 7:452-459, 1986
- D. P. O'Leary and P. Whitman, Parallel QR factorization by Householder and modified Gram-Schmidt algorithms, *Parallel Comput.* 16:99-112, 1990.
- A. Pothen and P. Raghavan, Distributed orthogonal factorization: Givens and Householder algorithms, *SIAM J. Sci. Stat. Comput.* 10:1113-1134, 1989
- A. Tiskin, Communication-efficient parallel generic pairwise elimination. *Future Generation Computer Systems* 23.2 (2007): 179-188.