CS 598: Communication Cost Analysis of Algorithms

Lecture 6: LU factorization with pivoting and intro to parallel QR

Edgar Solomonik

University of Illinois at Urbana-Champaign

September 12, 2016

Partial pivoting

When computing each $U(i, i)$ partial pivoting selects the row with the largest leading element and rotates it to the top

- achieves stability in practice
- pivot growth bounds are still exponential $O(2^n)$ and numerical examples can attain them, in particular the Wilkinson matrix
- complete pivoting guarantees stability but is even more expensive
- partial pivoting requires message/synchronization for each column when the matrix is blocked in both dimensions

Partial pivoting

Lets consider the cost of partial pivoting on a $m \times n$ matrix A, in a 1D layout ($\Pi(i)$ owns $A(im/P + 1 : (i + 1)m/P, :)$)

- selecting each pivot required an (all)reduce of size $O(1)$
- the cost of pivoting is therefore at least

$$O\Big(n \log(P) \cdot \alpha\Big)$$

- the synchronization cost of $n \log(P) \cdot \alpha$ is problematic, especially within 2D LU

Parallel pivoting

Parallel pivoting is a naive way to exploit more parallelism within pivoting

- perform pivoted LU on each pair of rows and recurse with $n/2$ rows
- worst-case pivot growth bound $O(2^n)$, same as partial pivoting
- exponential pivot growth in the average case
- unstable in practice

Pairwise pivoting

One alternative technique to partial pivoting is pairwise pivoting

- it looks like Givens rotations in QR, so we will return to it in more detail then
- basic idea is to perform 2-by-2 LU factorizations with partial pivoting, to ‘zero-out’ one matrix entry at a time
- this strategy is more ‘local’ than partial pivoting
- theoretical pivot growth bound is $O(4^n)$, worse than partial pivoting by factor of 2^n
- average case growth (empirical) is $O(n)$ rather than $O(n^{2/3})$ for partial pivoting
- in practical tests, somewhat more numerical error is indeed observed
- see Sorensen 1985 for details on stability

Tournament pivoting

Tournament pivoting is a stable approach for ‘block-pivoting’

- performs a tournament to determine best pivot row candidates
- rotates up 'best rows' of A
- does not perform LU while doing pivoting, so is different from naive version of blocked pairwise pivoting

Tournament pivoting

Consider $m \times n$ matrix A (seeking the leading n rows)

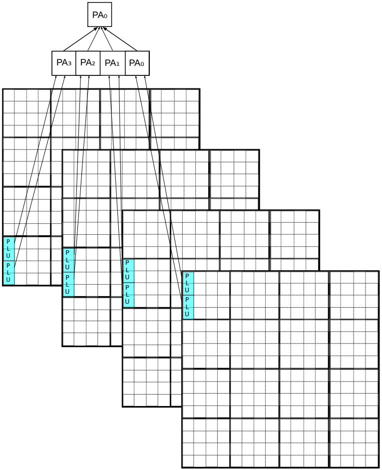
- partition $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ into two $m/2 \times n$ blocks
- recursively find best n candidate rows R_1 from A_1 and R_2 from A_2
- do LU with partial pivoting sequentially on $PLU = \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$
- return best b rows of A as top n rows of $P^T \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$
- Q: why might this be more stable than parallel pivoting?
- A: we pass up rows of A, so error of intermediate factorizations is not blown up
- best known worst-case error bound, $O(2^{n\log(P)})$, but observed to be more stable than pairwise pivoting empirically

Cost analysis of tournament pivoting

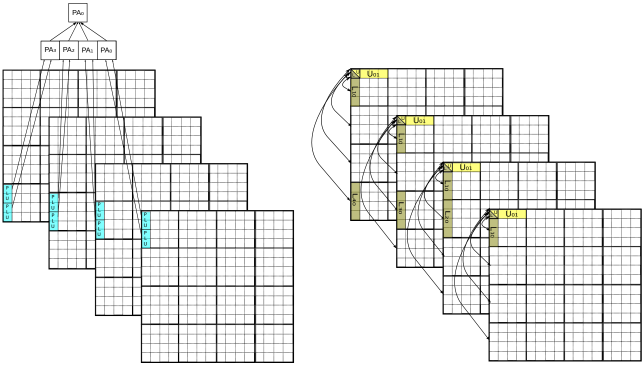
Consider tournament pivoting on a $m \times n$ matrix with $m \geq nP$. If γ is the cost of a floating point operation, then

$$\begin{aligned} T_{\mathrm{TP}}(m, n, P) &= T_{\mathrm{TP}}(m/2, n, P/2) + O(\alpha + n^2 \cdot \beta + n^3 \cdot \gamma) \\ T_{\mathrm{TP}}(m_0, n, 1) &= O(m_0 n^2 \cdot \gamma) \\ &= O(\log(P) \cdot \alpha + n^2 \log(P) \cdot \beta + (n^3 \log(P) + mn^2/P) \cdot \gamma) \end{aligned}$$

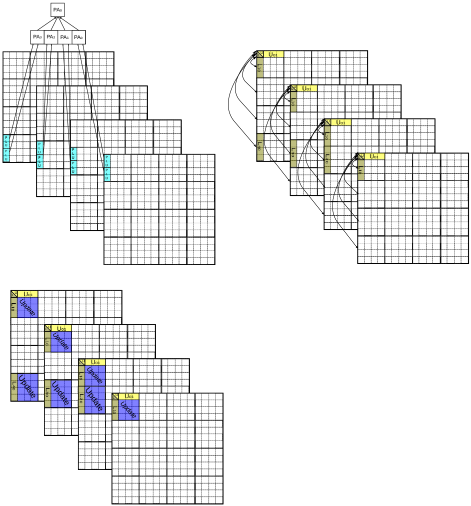
2.5D LU with tournament pivoting



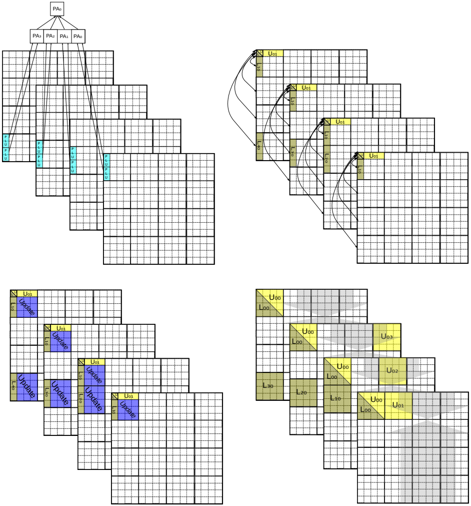
2.5D LU factorization with tournament pivoting



2.5D LU factorization with tournament pivoting



2.5D LU factorization with tournament pivoting



Summary of 2.5D LU

A slightly modified summary of 2.5D LU:

- 2D LU on rectangular panels of dimensions at most $n \times n/c$ with a $\sqrt{Pc} \times \sqrt{P/c}$ grid of processors
- a transposition of the computed panel of L on the processor grid, so each processor owns blocks of dimension $n/\sqrt{Pc^3} \times n/\sqrt{P/c}$
- the dual operation to compute a panel of U
- broadcasts of L and U panels to aggregate Schur complement
- scatter-reduce of aggregates to obtain the next panel of Schur complement

2D rectangular LU with tournament pivoting

The $n \times n/c$ panel factorizations are done with $\sqrt{Pc} \times \sqrt{P/c}$ processors

- tournament pivoting on each subpanel of dims (at most) $n \times b$ is

$$T_{\text{TP}}(n, b, \sqrt{Pc}) = O\Big(\log(P)\cdot\alpha + b^2\log(P)\cdot\beta + \Big(b^3\log(P) + \frac{nb^2}{\sqrt{Pc}}\Big)\cdot\gamma\Big)$$

- computing subpanel of U of dims $b \times n/c$ is less expensive
- the Schur complement broadcast has cost

$$T_{\text{Schur-2D}}(n, b, P, c) = O(\log(P)\cdot\alpha + \frac{nb}{\sqrt{Pc}}\cdot\beta + \frac{n^2 b}{cP}\cdot\gamma)$$

- Q: what maximal choice of b gives $T_{\text{TP}} \le T_{\text{Schur-2D}}$?
- A: $b = \frac{n}{\sqrt{Pc}\log(P)}$, for a total cost of (over all c panels)

$$\frac{n}{b}T_{\text{Schur-2D}}(n, b, P, c) = O\Big(\frac{n\log(P)}{b}\cdot\alpha + \frac{n^2}{\sqrt{Pc}}\cdot\beta + \frac{n^3}{cP}\cdot\gamma\Big)$$

- the synchronization cost becomes $O(\sqrt{Pc}\log(P)\cdot\alpha)$

Communication in 2.5D LU with tournament pivoting

- 2D LU on rectangular panels ✓: $O\left(\sqrt{Pc}\log(P)\cdot\alpha + \frac{n^2}{\sqrt{Pc}}\cdot\beta\right)$
- a transposition of the computed panel of L on the processor grid, so each processor owns blocks of dimension $n/\sqrt{P/c} \times n/\sqrt{Pc}$

$$O\left(c\log(P)\cdot\alpha + n^2\log(P)/P\cdot\beta\right)$$

- the dual operation to compute a panel of U (less than above)
- broadcasts of L and U panels to aggregate Schur complement
  - each panel block needs to be multiplied by $\sqrt{P/c}$ other blocks, naturally expressed by a $\sqrt{P/c}\times\sqrt{P/c}\times c$ processor grid
  - so each processor obtains $\sqrt{P/c}$ blocks of dims $n/\sqrt{Pc^3}\times n/\sqrt{P/c}$

$$O\left(c\log(P)\cdot\alpha + \frac{n^2}{\sqrt{Pc}}\cdot\beta\right)$$

- scatter-reduce of aggregates (each processor owns $n/\sqrt{P/c}\times n/\sqrt{Pc}$ block) to obtain the next panel of Schur complement

$$O\left(c\log(P)\cdot\alpha + \frac{cn^2}{P}\cdot\beta\right)$$

Short pause

QR factorization

QR factorization $A = QR$ where A is orthogonal and R is upper-triangular is a robust method with applications including linear systems $Ax = b$

- given a pivoted LU factorization $A = PLU$, we can compute $x = U^{-1}L^{-1}P^Tb$
- given a QR factorization $A = QR$, we can compute $x = R^{-1}Q^Tb$
- **for overdetermined systems (tall and skinny A),**

$$\hat{x} = R^+Q^Tb \textbf{ minimizes } \quad ||A\hat{x} - b||_2$$

- QR factorization is *unconditionally stable* for $||A - QR||_2$ (LU is only with complete pivoting), but not necessarily row-wise stable
  - applying orthogonal transformations is numerically stable, $\text{cond}(Q) = 1$
- QR is used to compute eigenvalue and singular value decompositions, as well as within iterative methods
- QR with column-pivoting is “rank-revealing”, its cost is proportional to the rank of A

Householder QR

Householder QR is a stable approach to computing the factorization

- Gram-Schmidt is not stable for $A - QR$, Modified Gram-Schmidt is not stable for $I - Q^T Q$, Givens rotations we will come back to later
- A Householder rotation is a symmetric orthogonal matrix $P = I - 2uu^T$
- u is picked to annihilate $n - 1$ entries in A and to have $||u||_2 = 1$
- Householder QR is stable, because multiplying by P corresponds to multiplying by an orthogonal matrix
- if we compute Q_i for the ith column of A (after updating), we obtain

$$\prod_{i=1}^{n-1} Q_i = Q$$

Aggregated Householder QR

We can aggregated k Householder transformations of dimensions n as $Q = I - YTY^T$

- where Y is $n \times k$, lower trapezoidal, unit-diagonal, all entries ≤ 1
- T is upper triangular and satisfies $T^{-1} + T^{-T} = -Y^T Y$
- its easy to check that this is true for $n = 1$, $T = [2]$, $T^{-1} = T^{-T} = 1/2$ and $Y^T Y = u^T u = ||u||_2^2 = 1$
- given $Q_1 = I - Y_1 T_1 Y_1^T$ and $Q_2 = I - Y_2 T_2 T_2^T$,

$$Q_1 Q_2 = I - YTY^T$$

where

$$Y = \begin{bmatrix} Y_1 & 0 \\ \vdots & Y_2 \end{bmatrix} \quad \text{and} \quad T^{-1} + T^{-T} = Y^T Y$$