CS 598: Communication Cost Analysis of Algorithms
Lecture 4: communication avoiding algorithms for LU factorization

Edgar Solomonik

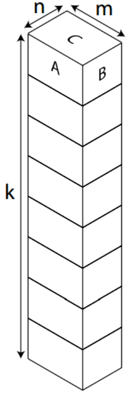
University of Illinois at Urbana-Champaign

August 31, 2016

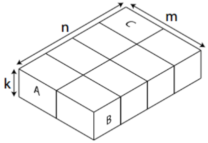
Review of Matrix Multiplication Algorithms

- SUMMA algorithm
  - 2D partitioning, bandwidth cost $O(n^2/\sqrt{P})$
  - based on broadcast/reduce or allgather/scatter-reduce
  - 3 variants: stationary C, stationary A, stationary B
- Cannon's algorithm
  - 2D partitioning, bandwidth cost $O(n^2/\sqrt{P})$
  - based on point-to-point messages
  - also has 3 variants: stationary C, stationary A, stationary B
  - can be done in-place, but is difficult to implement for nonsquare processor grids
- 3D algorithm
  - communicates all three matrices, achieves cost $O(n^2/P^{2/3})$
  - uses a factor of $P^{1/3}$ more memory
  - 2.5D algorithm achieves $O(n^2/\sqrt{cp})$ communication with a factor of c more memory

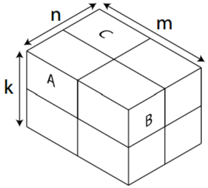
Rectangular Matrix Multiplication



(a) One large dimension



(b) Two large dimensions



(c) Three large dimensions

[Demmel et al, *Communication-optimal parallel recursive rectangular matrix multiplication*, 2013]

For multiplication of $m \times k$ and $k \times n$ matrices the bandwidth cost is

$$W(m,n,k,P) = O\left(\min_{p_1p_2p_3=P}\left[\frac{mk}{p_1p_2}+\frac{kn}{p_1p_3}+\frac{mn}{p_2p_3}\right]-\frac{mk+kn+mn}{P}\right)$$

Relevance of Loomis-Whitney inequality

Theorem (Loomis-Whitney (3D version), 1949)

Let V be a set of 3-tuples $V \subseteq [1,n]^3$

$$|V| \leq \sqrt{|\pi_1(V)||\pi_2(V)||\pi_3(V)|}$$

where

$$\pi_1(V) = \{(i_2, i_3) : \exists i_1, (i_1, i_2, i_3) \in V\}$$
$$\pi_2(V) = \{(i_1, i_3) : \exists i_2, (i_1, i_2, i_3) \in V\}$$
$$\pi_3(V) = \{(i_1, i_2) : \exists i_3, (i_1, i_2, i_3) \in V\}$$

To minimize comm. in MM, minimize $\Pi = \pi_1(V) \cup \pi_2(V) \cup \pi_3(V)$

$$|V| < |\Pi|^{3/2} \quad \Rightarrow \quad |\Pi| > |V|^{2/3}$$

when $|V| = n^3/P$, we see that $|\Pi| > n^2/P^{2/3}$

LU factorization

The LU factorization algorithm provides a stable (when combined with pivoting) replacement for computing the inverse of a n-by-n matrix A,

$$A = L \cdot U$$

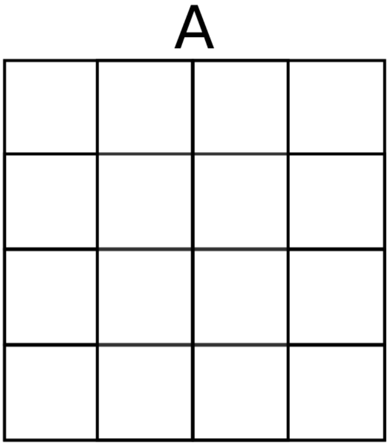
where L is lower-triangular and U is upper-triangular is computed via Gaussian elimination: for $k = 1$ to n,

- set $L(k, k) = 1$ and $U(k, k : n] = A(k, k : n)$
- divide $L(k+1 : n, k) = A(k+1 : n, k)/U(k, k)$
- update Schur complement

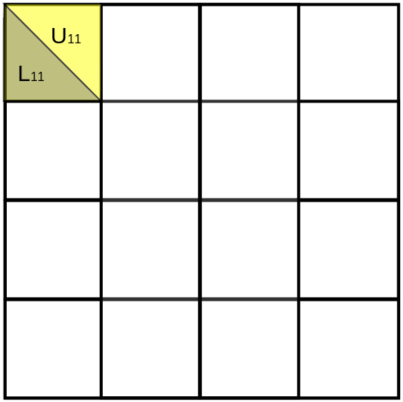
$$A(k+1 : n, k+1 : n) = A(k+1 : n, k+1 : n) - L(k+1 : n, k) \cdot U(k, k+1 : n)$$

this algorithm can be blocked analogously to matrix multiplication

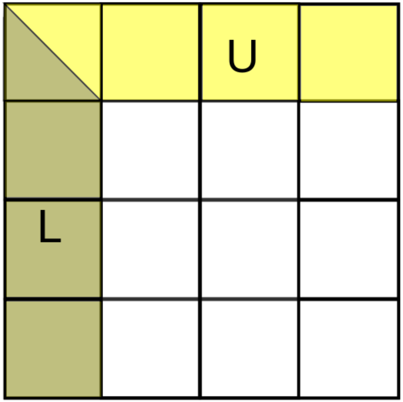
Blocked LU factorization



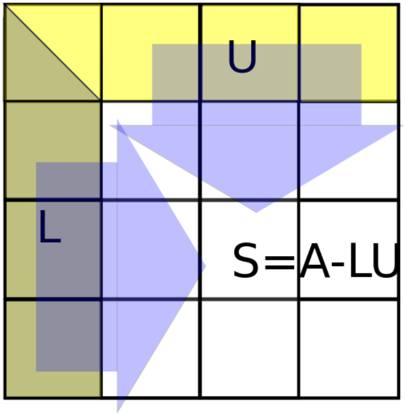
Blocked LU factorization



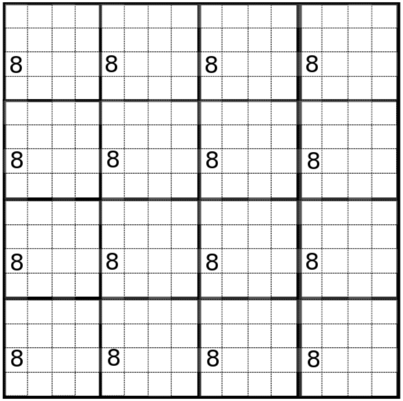
Blocked LU factorization



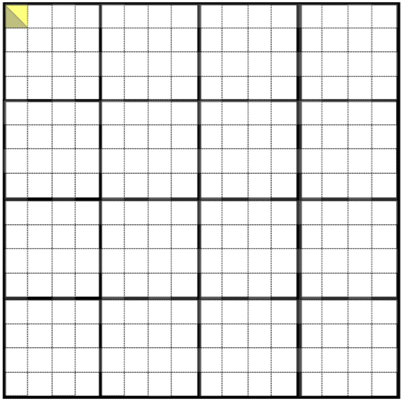
Blocked LU factorization



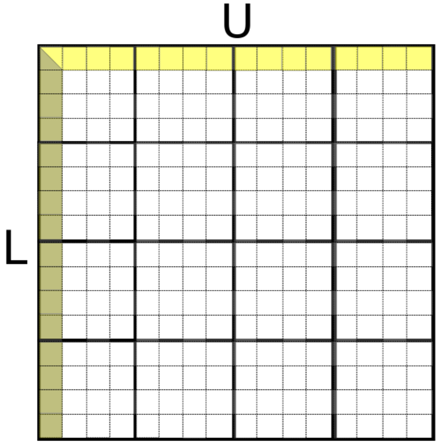
Block-cyclic LU factorization



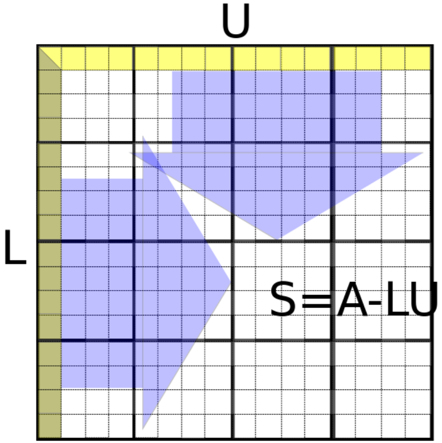
Block-cyclic LU factorization



Block-cyclic LU factorization



Block-cyclic LU factorization



Analysis of 2D LU algorithm

We are working on a $\sqrt{P} \times \sqrt{P}$ processor grid, with block size $b \leq n/\sqrt{P}$

The algorithm requires n/b steps, broadcasting the L and U factors of the diagonal blocks is less expensive than the Schur complement update, so we get the cost

$$\begin{aligned} T_{\mathrm{2D-LU}}(n, b, P) &= (n/b)\, T_{\mathrm{bcast}}(nb/\sqrt{P}, \sqrt{P}) \\ &= O((n/b)\log(P) \cdot \alpha + n^2/\sqrt{P} \cdot \beta) \end{aligned}$$

To minimize latency, we would pick a maximal block size: $b = n/\sqrt{P}$.

Q: Then why do we bother with block-cyclic layouts for 2D LU?

A: The computation cost of factorizing the diagonal blocks is proportional to $(n/b) \cdot b^3 = nb^2$, for the triangular solves to $(n/b) \cdot nb/\sqrt{P} = n^2 b/\sqrt{P}$, which would raise the leading order cost unless $b \ll n/\sqrt{P}$

Recursive LU factorization (Tiskin, 2002)

LU factorization has the form

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

and can be computed recursively via $[L, U] = \mathrm{LU}(A)$:

$$\begin{aligned} [L_{11}, U_{11}] &= \mathrm{LU}(A_{11}) \\ L_{21} &= A_{21} \cdot U_{11}^{-1} \\ U_{12} &= L_{11}^{-1} \cdot A_{12} \\ [L_{22}, U_{22}] &= \mathrm{LU}(A_{22} - L_{21} \cdot U_{12}) \\ L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} & \quad U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \end{aligned}$$

Q: But how to we obtain inversed matrices L_{11}^{-1} and U_{11}^{-1}?
Note: Triangular inversion is much more stable than dense matrix inversion but must be done with care (see Du Croz and Higham, 1992)

A: just compute them?

$$\begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}^{-1} = \begin{bmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} \cdot L_{21} \cdot L_{11}^{-1} & L_{22}^{-1} \end{bmatrix}$$

- invert L_{11}
- invert L_{22}
- perform matrix multiplications to obtain $L_{22}^{-1} \cdot L_{21} \cdot L_{11}^{-1}$
- Q: we had two recursive calls for LU, so is inversion as expensive?
- A: no, because in inversion the recursive calls are independent!
- Q: could we do better by building this algorithm into LU?

Recursive LU factorization

A: Yes! **Generalize the problem** $[L, U, L^{-1}, U^{-1}] = \mathrm{LU}(A)$:

$$
\begin{aligned}
[L_{11}, U_{11}, L_{11}^{-1}, U_{11}^{-1}] &= \mathrm{LU}(A_{11}) \\
L_{21} &= A_{21} \cdot U_{11}^{-1} \\
U_{12} &= L_{11}^{-1} \cdot A_{12} \\
[L_{22}, U_{22}, L_{22}^{-1}, U_{22}^{-1}] &= \mathrm{LU}(A_{22} - L_{21} \cdot U_{12}) \\
L &= \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \quad U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \\
L^{-1} &= \begin{bmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} \cdot L_{21} \cdot L_{11}^{-1} & L_{22}^{-1} \end{bmatrix} \\
U^{-1} &= \begin{bmatrix} U_{11}^{-1} & -U_{11}^{-1} \cdot U_{12} \cdot U_{22}^{-1} \\ 0 & U_{22}^{-1} \end{bmatrix}
\end{aligned}
$$

Try it yourself!

Our LU algorithm makes two recursive calls to LU with $n/2$ and P processors, and performs 7 matrix multiplications at each recursive level with matrix dimension $n/2$ and P processors

What bandwidth and synchronization costs can it achieve?

Recursive LU factorization: analysis

The two recursive calls within LU factorization must be done in sequence, so we perform them with all the processors. We have to also pay for the cost of matrix multiplications at each level

$$\begin{aligned} T_{\mathrm{LU}}(n,P) &= 2T_{\mathrm{LU}}(n/2,P) + O(T_{\mathrm{MM}}(n,P)) \\ &= 2T_{\mathrm{LU}}(n/2,P) + O\left(\log(P)\cdot\alpha + \frac{n^2}{P^{2/3}}\cdot\beta\right) \end{aligned}$$

with base-case cost (sequential execution)

$$T_{\mathrm{LU}}(n_0,P) = O(\log(P)\cdot\alpha + n_0^2\cdot\beta)$$

the bandwidth cost decreases geometrically at each level, the base cases can be done sequentially with $n_0 = n/P^{2/3}$, giving a total cost of

$$T_{\mathrm{LU}}(n,P) = O(P^{2/3}\cdot\log(P)\cdot\alpha) + O\left(\frac{n^2}{P^{2/3}}\cdot\beta\right)$$

Recursive triangular inversion: analysis

The two recursive calls within triangular inversion are independent, so we can perform them simultaneously with half of the processors

$$\begin{aligned} T_{\mathrm{Tri-Inv}}(n,P) &= 2T_{\mathrm{Tri-Inv}}(n/2,P/2) + O(T_{\mathrm{MM}}(n,P)) \\ &= T_{\mathrm{Tri-Inv}}(n/2,P/2) + O\left(\log(P)\cdot\alpha + \frac{n^2}{P^{2/3}}\cdot\beta\right) \end{aligned}$$

with base-case cost (sequential execution)

$$T_{\mathrm{Tri-Inv}}(n_0,P) = O(\log(P)\cdot\alpha) + O(n_0^2\cdot\beta)$$

the bandwidth cost goes down at each level and we can execute the base-case sequentially when $n_0 = n/P^{2/3}$, with a total cost of

$$T_{\mathrm{Tri-Inv}}(n,P) = O(\log(P)^2\cdot\alpha) + O\left(\frac{n^2}{P^{2/3}}\cdot\beta\right)$$

So triangular inversion has *logarithmic depth* while LU has *polynomial depth*, but using inversion within LU naively would raise the LU latency by another log factor

Short pause

Course projects

- the choice of project will be flexible
- doing something in your current research area is encouraged
- first proposal deadline pushed back a week to Sep 28
- setting up a meeting with me prior to first proposal is recommended
  - especially if you are not sure what you want to do
  - can give you feedback on your ideas (gauge difficulty) or suggest others
  - send an email or just show up at office hours