Parallel LU Decomposition
(PSC §2.3)
Designing a parallel algorithm

- Main question: how to distribute the data?
- What data? The matrix $A$ and the permutation $\pi$.
- Data distribution + sequential algorithm $\rightarrow$ computation supersteps.
- Design backwards: insert preceding communication supersteps following the need-to-know principle.
Data distribution for the matrix $A$

- The bulk of the work in the sequential computation is the update

$$a_{ij} := a_{ij} - a_{ik}a_{kj}$$

for matrix elements $a_{ij}$ with $i, j \geq k + 1$, taking $2(n - k - 1)^2$ flops.

- The other operations take only $n - k - 1$ flops. Thus, the data distribution is chosen mainly by considering the matrix update.

- Elements $a_{ij}, a_{ik}, a_{kj}$ may not be on the same processor. Who does the update?

- Many elements $a_{ij}$ must be updated in stage $k$, but only few elements $a_{ik}, a_{kj}$ are used, all from column $k$ or row $k$ of the matrix. Moving those elements around causes less traffic.

- Therefore, the owner of $a_{ij}$ computes the new value $a_{ij}$ using communicated values of $a_{ik}, a_{kj}$. 
Matrix update by operation \( a_{ij} := a_{ij} - a_{ik} a_{kj} \)

Update of row \( i \) uses only one value, \( a_{ik} \), from column \( k \). If we distribute row \( i \) over only \( N \) processors, then \( a_{ik} \) needs to be sent to at most \( N - 1 \) processors.
Matrix distribution

- A matrix distribution is a mapping

\[ \phi : \{(i,j) : 0 \leq i, j < n\} \rightarrow \{(s,t) : 0 \leq s < M \land 0 \leq t < N\} \]

from the set of matrix index pairs to the set of processor identifiers. The mapping function \( \phi \) has two coordinates,

\[ \phi(i,j) = (\phi_0(i,j), \phi_1(i,j)). \]

- Here, we number the processors in 2D fashion, with \( p = MN \). This is just a numbering!

- Processor numberings have no physical meaning. BSPlib randomly renumbers the processors at the start.

- A processor row \( P(s,\ast) \) is a group of \( N \) processors \( P(s,t) \) with \( 0 \leq t < N \). A processor column \( P(\ast,t) \) is a group of \( M \) processors \( P(s,t) \) with \( 0 \leq s < M \).
A matrix distribution is called **Cartesian** if \( \phi_0(i,j) \) is independent of \( j \) and \( \phi_1(i,j) \) is independent of \( i \):

\[
\phi(i,j) = (\phi_0(i), \phi_1(j)).
\]
Parallel matrix update

(8) \[ \text{if } \phi_0(k) = s \land \phi_1(k) = t \text{ then put } a_{kk} \text{ in } P(*, t); \]

(9) \[ \text{if } \phi_1(k) = t \text{ then for all } i : k < i < n \land \phi_0(i) = s \text{ do} \]
\[ a_{ik} := a_{ik}/a_{kk}; \]
Parallel matrix update

(8) \[ \text{if } \phi_0(k) = s \land \phi_1(k) = t \text{ then put } a_{kk} \text{ in } P(\ast, t); \]

(9) \[ \text{if } \phi_1(k) = t \text{ then for all } i : k < i < n \land \phi_0(i) = s \text{ do} \]
\[ a_{ik} := a_{ik}/a_{kk}; \]

(10) \[ \text{if } \phi_1(k) = t \text{ then for all } i : k < i < n \land \phi_0(i) = s \text{ do} \]
\[ \text{put } a_{ik} \text{ in } P(s, \ast); \]
\[ \text{if } \phi_0(k) = s \text{ then for all } j : k < j < n \land \phi_1(j) = t \text{ do} \]
\[ \text{put } a_{kj} \text{ in } P(\ast, t); \]

(11) \[ \text{for all } i : k < i < n \land \phi_0(i) = s \text{ do} \]
\[ \text{for all } j : k < j < n \land \phi_1(j) = t \text{ do} \]
\[ a_{ij} := a_{ij} - a_{ik}a_{kj}; \]
Parallel pivot search

(0) \( \textbf{if} \ \phi_1(k) = t \ \textbf{then} \ \ r_s := \arg\max(|a_{ik}| : k \leq i < n \ \land \ \phi_0(i) = s); \)

(1) \( \textbf{if} \ \phi_1(k) = t \ \textbf{then} \ \text{put } r_s \ \text{and } a_{rs,k} \ \text{in } P(\ast, t); \)
Parallel pivot search

(0) if $\phi_1(k) = t$ then $r_s := \arg\max(|a_{ik}| : k \leq i < n \land \phi_0(i) = s)$;

(1) if $\phi_1(k) = t$ then put $r_s$ and $a_{rs,k}$ in $P(*, t)$;

(2) if $\phi_1(k) = t$ then

$s_{\text{max}} := \arg\max(|a_{rq,k}| : 0 \leq q < M)$;

$r := r_{s_{\text{max}}}$;

(3) if $\phi_1(k) = t$ then put $r$ in $P(s, *)$;
Two parallelisation methods

▶ The need-to-know principle: exactly those nonlocal data that are needed in a computation superstep should be fetched in preceding communication supersteps.

▶ Matrix update uses first parallelisation method: look at lhs (left-hand side) of assignment, owner computes.

▶ Pivot search uses second method: look at rhs of assignment, compute what can be done locally, reduce the number of data to be communicated.

▶ In pivot search: first a local search, then communication of the local winner to all processors, finally a redundant (replicated) search for the global winner.

▶ Broadcast of $r$ in (3) is needed later in (4). Designing backwards, we formulate (4) first and then insert (3).
Distribution for permutation $\pi$

- Store $\pi_k$ together with row $k$, somewhere in processor row $P(\phi_0(k), *)$.
- We choose $P(\phi_0(k), 0)$. This gives a true distribution.
- We could also have chosen to replicate $\pi_k$ in processor row $P(\phi_0(k), *)$. This would save some if-statements in our programs.
Index and row swaps

(4) \[\text{if } \phi_0(k) = s \land t = 0 \text{ then put } \pi_k \text{ as } \hat{\pi}_k \text{ in } P(\phi_0(r), 0); \]
\[\text{if } \phi_0(r) = s \land t = 0 \text{ then put } \pi_r \text{ as } \hat{\pi}_r \text{ in } P(\phi_0(k), 0); \]

(5) \[\text{if } \phi_0(k) = s \land t = 0 \text{ then } \pi_k := \hat{\pi}_r; \]
\[\text{if } \phi_0(r) = s \land t = 0 \text{ then } \pi_r := \hat{\pi}_k; \]
Index and row swaps

(4) \[\text{if } \phi_0(k) = s \land t = 0 \text{ then put } \pi_k \text{ as } \hat{\pi}_k \text{ in } P(\phi_0(r), 0);\]
\[\text{if } \phi_0(r) = s \land t = 0 \text{ then put } \pi_r \text{ as } \hat{\pi}_r \text{ in } P(\phi_0(k), 0);\]

(5) \[\text{if } \phi_0(k) = s \land t = 0 \text{ then } \pi_k := \hat{\pi}_r;\]
\[\text{if } \phi_0(r) = s \land t = 0 \text{ then } \pi_r := \hat{\pi}_k;\]

(6) \[\text{if } \phi_0(k) = s \text{ then for all } j : 0 \leq j < n \land \phi_1(j) = t \text{ do}
\text{put } a_{kj} \text{ as } \hat{a}_{kj} \text{ in } P(\phi_0(r), t);\]
\[\text{if } \phi_0(r) = s \text{ then for all } j : 0 \leq j < n \land \phi_1(j) = t \text{ do}
\text{put } a_{rj} \text{ as } \hat{a}_{rj} \text{ in } P(\phi_0(k), t);\]

(7) \[\text{if } \phi_0(k) = s \text{ then for all } j : 0 \leq j < n \land \phi_1(j) = t \text{ do}
\text{a}_{kj} := \hat{a}_{rj};\]
\[\text{if } \phi_0(r) = s \text{ then for all } j : 0 \leq j < n \land \phi_1(j) = t \text{ do}
\text{a}_{rj} := \hat{a}_{kj};\]
Optimising the matrix distribution

- We have chosen a Cartesian matrix distribution $\phi$ to limit the communication.
- We now specify $\phi$ further to achieve a good computational load balance and to minimise the communication.
- Maximum number of local matrix rows with index $\geq k$:

$$R_k = \max_{0 \leq s < M} | \{ i : k \leq i < n \land \phi_0(i) = s \} |.$$

Maximum number of local matrix columns with index $\geq k$:

$$C_k = \max_{0 \leq t < N} | \{ j : k \leq j < n \land \phi_1(j) = t \} |.$$

- The computation cost of the largest superstep, the matrix update (11), is then $2R_{k+1}C_{k+1}$. 
Example

\[
\begin{array}{ccccccc}
  & t = 0 & 2 & 1 & 2 & 0 & 1 & 0 \\
 s = 0 & 00 & 02 & 01 & 02 & 00 & 01 & 00 \\
 0 & 00 & 02 & 01 & 02 & 00 & 01 & 00 \\
 1 & 10 & 12 & 11 & 12 & 10 & 11 & 10 \\
 0 & 00 & 02 & 01 & 02 & 00 & 01 & 00 \\
 1 & 10 & 12 & 11 & 12 & 10 & 11 & 10 \\
 0 & 00 & 02 & 01 & 02 & 00 & 01 & 00 \\
 1 & 10 & 12 & 11 & 12 & 10 & 11 & 10 \\
\end{array}
\]

\[R_0 = 4, \ C_0 = 3 \text{ and } R_4 = 2, \ C_4 = 2\]
Bound for \( R_k \)

\[
R_k \geq \left\lceil \frac{n - k}{M} \right\rceil.
\]

Proof: Assume this is untrue, so that \( R_k < \left\lfloor \frac{n - k}{M} \right\rfloor \). Because \( R_k \) is integer, we even have \( R_k < \frac{n - k}{M} \). Hence all \( M \) processor rows together hold less than \( M \cdot \frac{n - k}{M} = n - k \) matrix rows. But they hold all matrix rows \( k \leq i < n \). Contradiction. \( \square \)
2D cyclic distribution attains bound

\[
\begin{array}{cccccccc}
   & 0 & 1 & 2 & 0 & 1 & 2 & 0 \\
 s = 0 & 00 & 01 & 02 & 00 & 01 & 02 & 00 \\
 1 & 10 & 11 & 12 & 10 & 11 & 12 & 10 \\
 0 & 00 & 01 & 02 & 00 & 01 & 02 & 00 \\
 1 & 10 & 11 & 12 & 10 & 11 & 12 & 10 \\
 0 & 00 & 01 & 02 & 00 & 01 & 02 & 00 \\
 1 & 10 & 11 & 12 & 10 & 11 & 12 & 10 \\
 0 & 00 & 01 & 02 & 00 & 01 & 02 & 00 \\
\end{array}
\]

\[
\phi_0(i) = i \mod M, \quad \phi_1(j) = j \mod N.
\]

\[
R_k = \left\lceil \frac{n - k}{M} \right\rceil, \quad C_k = \left\lceil \frac{n - k}{N} \right\rceil.
\]
Cost of main computation superstep (matrix update)

\[ T_{(11),\text{cyclic}} = 2 \left\lceil \frac{n - k - 1}{M} \right\rceil \left\lceil \frac{n - k - 1}{N} \right\rceil \geq \frac{2(n - k - 1)^2}{p}. \]

\[ T_{(11),\text{cyclic}} < 2 \left( \frac{n - k - 1}{M} + 1 \right) \left( \frac{n - k - 1}{N} + 1 \right) \]
\[ = \frac{2(n - k - 1)^2}{p} + \frac{2(n - k - 1)}{p}(M + N) + 2. \]

The upper bound is minimal for \( M = N = \sqrt{p} \). The second-order term \( 4(n - k - 1)/\sqrt{p} \) is the additional computation cost caused by load imbalance.
Load balance for the square block distribution

For $k \geq 4$, only the yellow processors works.
Load balance for the square cyclic distribution

For $k = 4, 5, 6$, all processors work.
Cost of main communication superstep

The cost of the broadcast of row $k$ and column $k$ in (10) is

$$T_{(10)} = (R_{k+1}(N-1) + C_{k+1}(M-1))g$$

$$\geq \left( \left\lceil \frac{n-k-1}{M} \right\rceil (N-1) + \left\lceil \frac{n-k-1}{N} \right\rceil (M-1) \right) g$$

$$= T_{(10),cyclic}.$$

$$T_{(10),cyclic} < \left( \left( \frac{n-k-1}{M} + 1 \right) N + \left( \frac{n-k-1}{N} + 1 \right) M \right) g$$

$$= \left( (n-k-1) \left( \frac{N}{M} + \frac{M}{N} \right) + M + N \right) g.$$

The upper bound is again minimal for $M = N = \sqrt{p}$. The resulting communication cost is about $2(n-k-1)g$. 

Lecture 2.3 Parallel LU
Summary

- We determined the matrix distribution, first by restricting it to be Cartesian, then by choosing the 2D cyclic distribution, based on a careful analysis of the main computation and communication supersteps, and finally by showing that a square $\sqrt{p} \times \sqrt{p}$ distribution is best.

- Developing the algorithm goes hand in hand with the cost analysis.

- We now have a correct algorithm and a good distribution, but the overall BSP cost may not be minimal yet. Wait and see ...