MS87: Innovative Methods for High Performance Iterative Solvers
Organized by Marc Baboulin, Takeshi Fukaya, Takeshi Iwashita

ParILUT - A New Parallel Threshold ILU

Hartwig Anzt, Edmond Chow, Jack Dongarra

SIAM Conference on
Parallel Processing
for Scientific Computing
March 7-10, 2018
Waseda University
Tokyo, Japan

Karlsruher Institut für Technologie
The University of Tennessee Knoxville
Georgia Tech

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Exact LU Factorization

- Decompose system matrix into product $A = L \cdot U$.
- Based on Gaussian elimination.
- Triangular solves to solve a system $Ax = b$:

$$Ly = b \Rightarrow y \qquad \Rightarrow \qquad Ly = b \Rightarrow x$$

- De-Facto standard for solving dense problems.
- *What about sparse? Often significant fill-in...*

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Exact LU Factorization

- Decompose system matrix into product $A = L \cdot U$.
- Based on Gaussian elimination.
- Triangular solves to solve a system $Ax = b$:

$$Ly = b \Rightarrow y \qquad \Rightarrow \qquad Ly = b \Rightarrow x$$

- De-Facto standard for solving dense problems.
- *What about sparse? Often significant fill-in…*

Incomplete LU Factorization (ILU)

- Focused on restricting fill-in to a specific sparsity pattern $\mathcal{S}$.

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Exact LU Factorization

- Decompose system matrix into product $A = L \cdot U$.
- Based on Gaussian elimination.
- Triangular solves to solve a system $Ax = b$:

$$Ly = b \Rightarrow y \qquad \Rightarrow \qquad Ly = b \Rightarrow x$$

- De-Facto standard for solving dense problems.
- *What about sparse? Often significant fill-in…*

Incomplete LU Factorization (ILU)

- Focused on restricting fill-in to a specific sparsity pattern $\mathcal{S}$.
- For **ILU(0)**, $\mathcal{S}$ is the sparsity pattern of A.
  - Works well for many problems.
  - *Is this the best we can get for nonzero count?*

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Exact LU Factorization

- Decompose system matrix into product $A = L \cdot U$.
- Based on Gaussian elimination.
- Triangular solves to solve a system $Ax = b$:

$$Ly = b \Rightarrow y \qquad \Rightarrow \qquad Ly = b \Rightarrow x$$

- De-Facto standard for solving dense problems.
- *What about sparse? Often significant fill-in…*

Incomplete LU Factorization (ILU)

- Focused on restricting fill-in to a specific sparsity pattern $\mathcal{S}$.
- For **ILU(0)**, $\mathcal{S}$ is the sparsity pattern of A.
  - Works well for many problems.
  - *Is this the best we can get for nonzero count?*
- Fill-in in threshold ILU (**ILUT**) bases $\mathcal{S}$ on the significance of elements (e.g. magnitude).
  - Often better preconditioners than level-based ILU.
  - Difficult to parallelize.

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Rethink the overall strategy!

- Use a parallel iterative process to generate factors.

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Rethink the overall strategy!

- Use a parallel iterative process to generate factors.
- The preconditioner should have a moderate number of nonzero elements, *but we don't care too much about intermediate data.*

Motivation

We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).

- *Where should these nonzero elements be located?*
- *How can we compute the preconditioner in a highly parallel fashion?*

Rethink the overall strategy!

- Use a parallel iterative process to generate factors.
- The preconditioner should have a moderate number of nonzero elements,
  but we don't care too much about intermediate data.

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that*** $A \approx L \cdot U$ ***is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem…

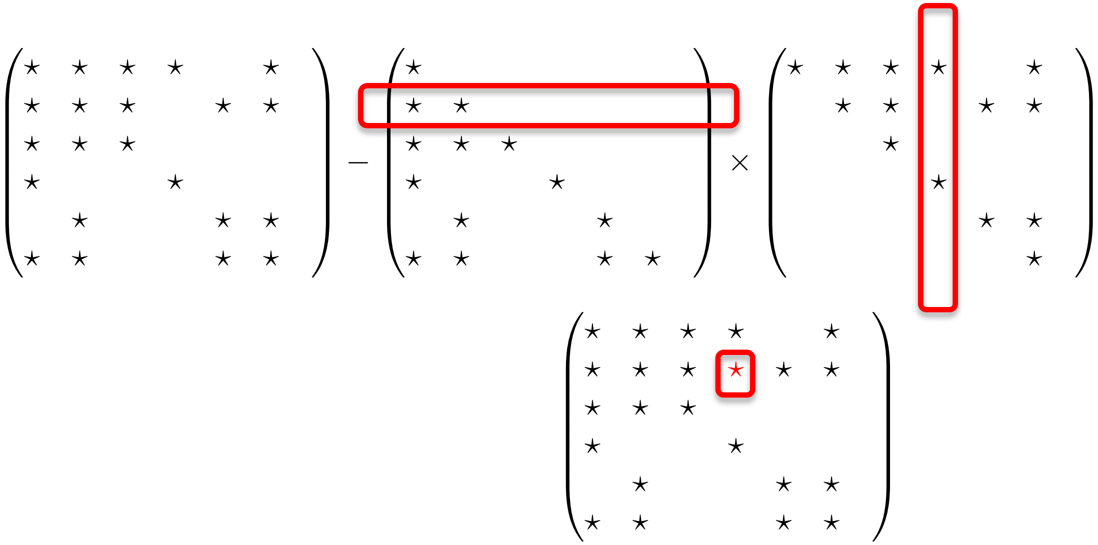
$$
\begin{pmatrix}
\star & \star & \star & \star & & \star \\
\star & \star & \star & & \star & \star \\
\star & \star & \star & & & \\
\star & & & \star & & \\
& \star & & & \star & \star \\
\star & \star & & & \star & \star
\end{pmatrix}
-
\begin{pmatrix}
\star & & & & & \\
\star & \star & & & & \\
\star & \star & \star & & & \\
\star & & & \star & & \\
& \star & & & \star & \\
\star & \star & & & \star & \star
\end{pmatrix}
\times
\begin{pmatrix}
\star & \star & \star & \star & & \star \\
& \star & \star & & \star & \star \\
& & \star & & & \\
& & & \star & & \\
& & & & \star & \star \\
& & & & & \star
\end{pmatrix}
$$

ILU residual $R = A - L \times U$

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

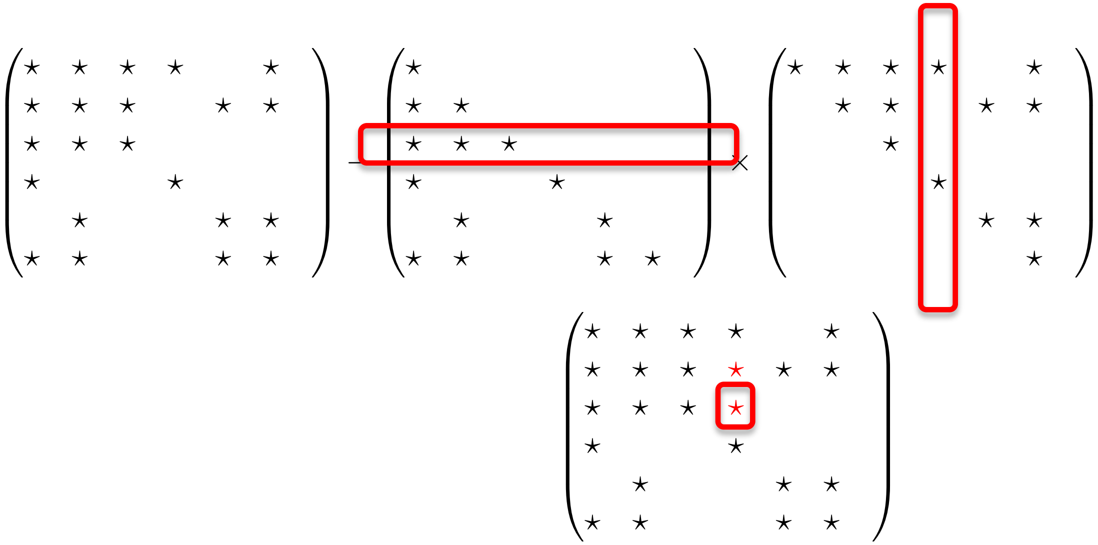
- This is an optimization problem…



Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem…



Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem…

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \\ \star & \star & & & \star & \star \end{pmatrix} \times \begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ & & \star & & & \\ & & & \star & & \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}$$

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ \star & \star & \star & \star & \star & \star \end{pmatrix}$$

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that*** $A \approx L \cdot U$ ***is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem…

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \\ \star & \star & & & \star & \star \end{pmatrix} \times \begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ & & \star & & & \\ & & & \star & & \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}$$

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & {\color{red}\star} & \star & \star \\ \star & \star & \star & {\color{red}\star} & {\color{red}\star} & {\color{red}\star} \\ \star & {\color{red}\star} & {\color{red}\star} & \star & & {\color{red}\star} \\ & \star & {\color{red}\star} & & \star & \star \\ \star & \star & {\color{red}\star} & {\color{red}\star} & \star & \star \end{pmatrix}$$

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that*** $A \approx L \cdot U$ ***is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem…

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \\ \star & \star & & & \star & \star \end{pmatrix} \times \begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ & & \star & & & \\ & & & \star & & \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}$$

ILU residual matrix pattern

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ \star & \star & \star & \star & \star & \star \end{pmatrix} = \begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ \star & \star & \star & \star & \star & \star \end{pmatrix}$$

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem with ${\color{red} nnz(A - L \cdot U)}$ equations and ${\color{blue} nnz(L + U)}$ variables.

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \\ \star & \star & & & \star & \star \end{pmatrix} \times \begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ & & \star & & & \\ & & & \star & & \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}$$

$${\color{red}\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ \star & \star & \star & \star & \star & \star \end{pmatrix}} = \begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - {\color{blue}\left(\begin{matrix} {\color{black}\star} & & & & & \\ \star & {\color{black}\star} & & & & \\ \star & \star & {\color{black}\star} & & & \\ \star & & & {\color{black}\star} & & \\ & \star & & & {\color{black}\star} & \\ \star & \star & & & \star & {\color{black}\star} \end{matrix}\right)} \times {\color{blue}\begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ & & \star & & & \\ & & & \star & & \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}}$$

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that*** $A \approx L \cdot U$ ***is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem with $nnz(A - L \cdot U)$ equations and $nnz(L + U)$ variables.
- We may want to compute the values in L, U such that $R = A - L \cdot U = 0|_{\mathcal{S}}$, the approximation being exact in the locations included in $\mathcal{S}$, *but not outside!*

$nnz(L + U)$ equations
$nnz(L + U)$ variables

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & \color{red}\star & \star & \star \\ \star & \star & \star & \color{red}\star & \color{red}\star & \color{red}\star \\ \star & \color{red}\star & \color{red}\star & \star & & \color{red}\star \\ & \star & \color{red}\star & & \star & \star \\ \star & \star & \color{red}\star & \color{red}\star & \star & \star \end{pmatrix} = \begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \\ \star & \star & & & \star & \star \end{pmatrix} \times \begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & & \star & \star \\ & & \star & & & \\ & & & \star & & \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}$$

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that*** $A \approx L \cdot U$ ***is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem with $nnz(A - L \cdot U)$ equations and $nnz(L + U)$ variables.
- We may want to compute the values in L, U such that $R = A - L \cdot U = 0|_{\mathcal{S}}$, the approximation being exact in the locations included in $\mathcal{S}$, *but not outside!*
- This is the underlying idea of Edmond Chow's parallel ILU algorithm[1]:

$$F(L,U) = \begin{cases} \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right), & i > j \\ a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, & i \leq j \end{cases}$$

- Converges in the asymptotic sense towards incomplete factors L, U such that $R = A - L \cdot U = 0|_{\mathcal{S}}$

[1] Chow and Patel. **"Fine-grained Parallel Incomplete LU Factorization".** *In: SIAM J. on Sci. Comp. (2015).*

Considerations

1. *Select a set of nonzero locations.*
2. ***Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.***
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

- This is an optimization problem with $nnz(A - L \cdot U)$ equations and $nnz(L + U)$ variables.
- We may want to compute the values in L, U such that $R = A - L \cdot U = 0|_{\mathcal{S}}$, the approximation being exact in the locations included in $\mathcal{S}$, *but not outside!*
- This is the underlying idea of Edmond Chow's parallel ILU algorithm[1]:

$$F(L,U) = \begin{cases} \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right), & i > j \\ a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, & i \leq j \end{cases}$$

- We may not need high accuracy here, because we may change the pattern again…
- One single fixed-point sweep.

Fixed-point sweep approximates incomplete factors.

[1]Chow and Patel. **"Fine-grained Parallel Incomplete LU Factorization".** *In: SIAM J. on Sci. Comp. (2015).*

Considerations

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. ***Repeat until the preconditioner quality stagnates.***

Fixed-point sweep approximates incomplete factors.

Considerations

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. ***Repeat until the preconditioner quality stagnates.***

- Comparing sparsity patterns extremely difficult.
- Maybe use the ILU residual as convergence check.

Compute ILU residual & check convergence.

Fixed-point sweep approximates incomplete factors.

Considerations

1. ***Select a set of nonzero locations.***
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. *Maybe change some locations in favor of locations that result in a better preconditioner.*
4. *Repeat until the preconditioner quality stagnates.*

Compute ILU residual & check convergence.

- The sparsity pattern of A might be a good initial start for nonzero locations.

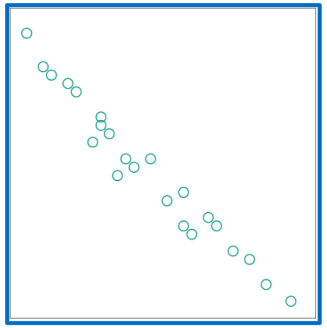
Fixed-point sweep approximates incomplete factors.

Considerations

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

Identify locations with nonzero ILU residual.

Compute ILU residual & check convergence.



- The sparsity pattern of A might be a good initial start for nonzero locations.
- Then, the approximation will be exact for all locations $\mathcal{S}_0 = \mathcal{S}(L_0 + U_0)$ and nonzero in locations $\mathcal{S}_1 = (\mathcal{S}(A) \cup \mathcal{S}(L_0 \cdot U_0)) \setminus \mathcal{S}(L_0 + U_0)$[1].

$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \color{blue}\star & \star & & & & \\ \color{blue}\star & \color{blue}\star & \star & & & \\ \color{blue}\star & & & \star & & \\ & \color{blue}\star & & & \star & \\ \color{blue}\star & \color{blue}\star & & & \color{blue}\star & \star \end{pmatrix} \times \begin{pmatrix} \color{blue}\star & \color{blue}\star & \color{blue}\star & \color{blue}\star & & \color{blue}\star \\ & \color{blue}\star & \color{blue}\star & & \color{blue}\star & \color{blue}\star \\ & & \color{blue}\star & & & \\ & & & \color{blue}\star & & \\ & & & & \color{blue}\star & \color{blue}\star \\ & & & & & \color{blue}\star \end{pmatrix} = \begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & \color{red}\star & \star & \star \\ \star & \star & \star & \color{red}\star & \color{red}\star & \color{red}\star \\ \star & \color{red}\star & \color{red}\star & \star & & \color{red}\star \\ & \star & \color{red}\star & & \star & \star \\ \star & \star & \color{red}\star & \color{red}\star & \star & \star \end{pmatrix}$$

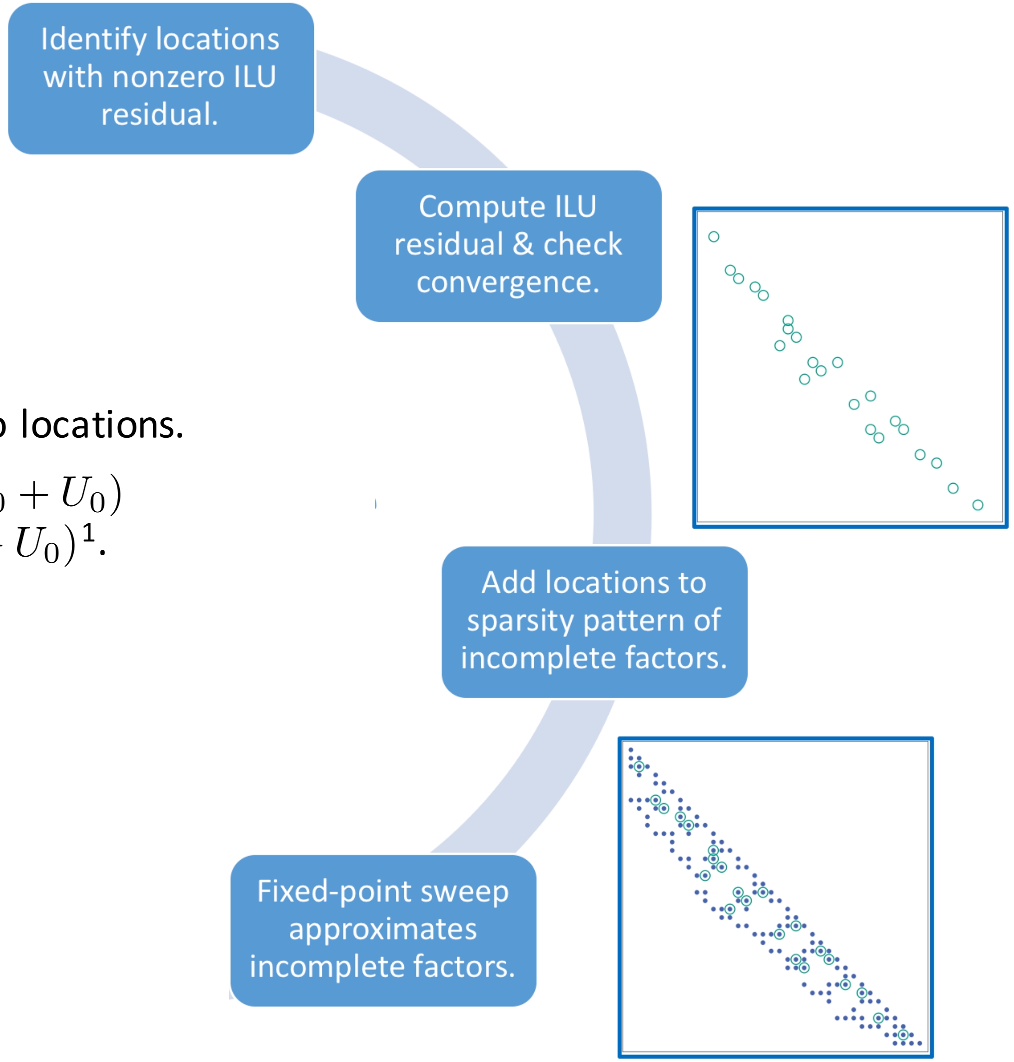
Fixed-point sweep approximates incomplete factors.

[1]Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition"**. *(2003).*

Considerations

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that* $A \approx L \cdot U$ *is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

- The sparsity pattern of A might be a good initial start for nonzero locations.
- Then, the approximation will be exact for all locations $\mathcal{S}_0 = \mathcal{S}(L_0 + U_0)$ and nonzero in locations $\mathcal{S}_1 = (\mathcal{S}(A) \cup \mathcal{S}(L_0 \cdot U_0)) \setminus \mathcal{S}(L_0 + U_0)$[1].
- Adding all these locations (level-fill!) might be good idea…

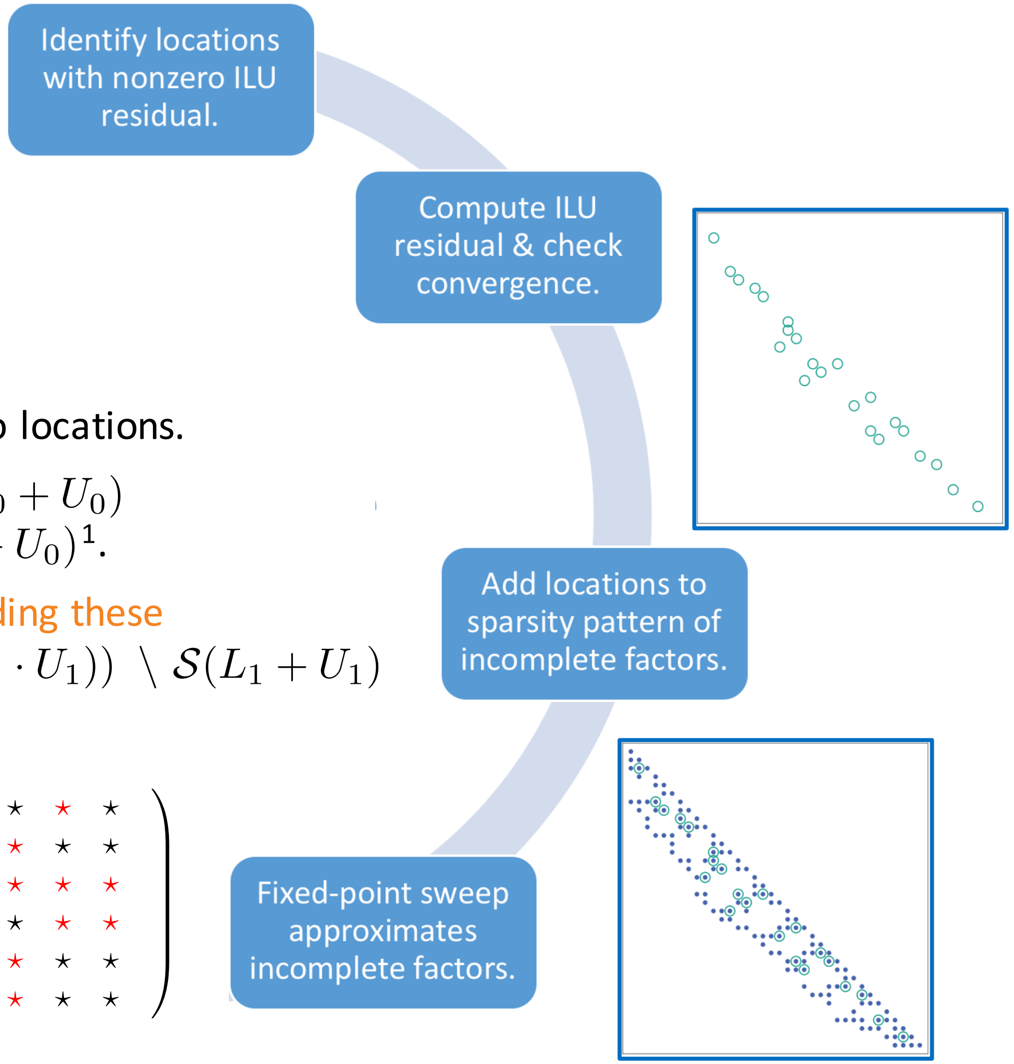


$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\ \star & \star & \star & & & \\ \star & \star & \star & \star & & \\ & \star & \star & & \star & \\ \star & \star & \star & \star & \star & \star \end{pmatrix} \times \begin{pmatrix} \star & \star & \star & \star & & \star \\ & \star & \star & \star & \star & \star \\ & & \star & \star & \star & \star \\ & & & \star & & \star \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix}$$

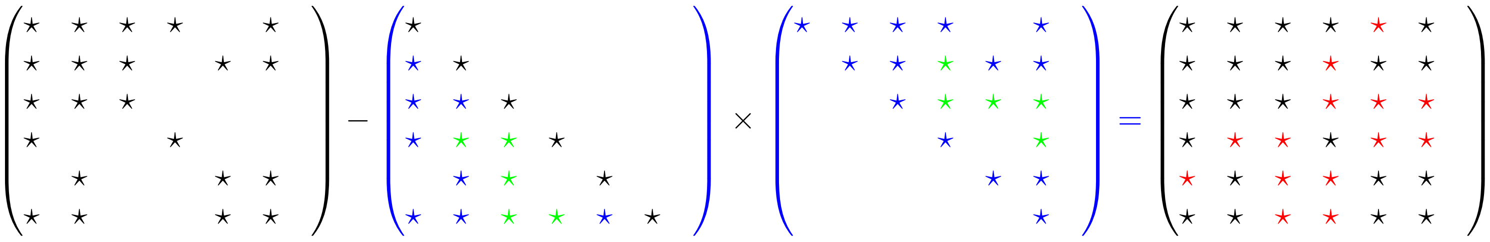
[1]Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition"**. *(2003).*

Considerations

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*



- The sparsity pattern of A might be a good initial start for nonzero locations.
- Then, the approximation will be exact for all locations $\mathcal{S}_0 = \mathcal{S}(L_0 + U_0)$ and nonzero in locations $\mathcal{S}_1 = (\mathcal{S}(A) \cup \mathcal{S}(L_0 \cdot U_0)) \setminus \mathcal{S}(L_0 + U_0)$[1].
- Adding all these locations (level-fill!) might be good idea, but adding these will again generate new nonzero residuals $\mathcal{S}_2 = (\mathcal{S}(A) \cup \mathcal{S}(L_1 \cdot U_1)) \setminus \mathcal{S}(L_1 + U_1)$

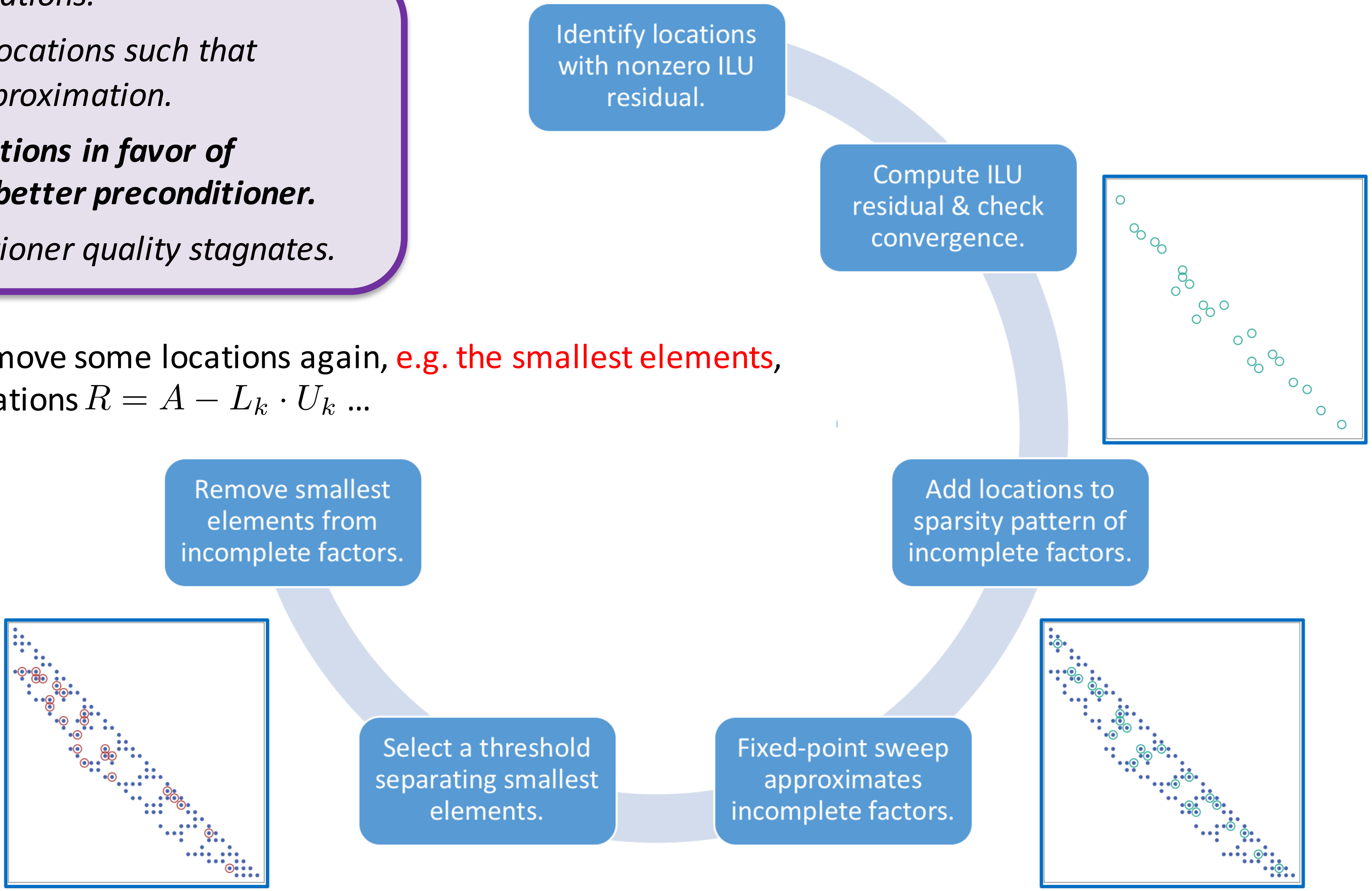


[1] Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition".** *(2003).*

Considerations

1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

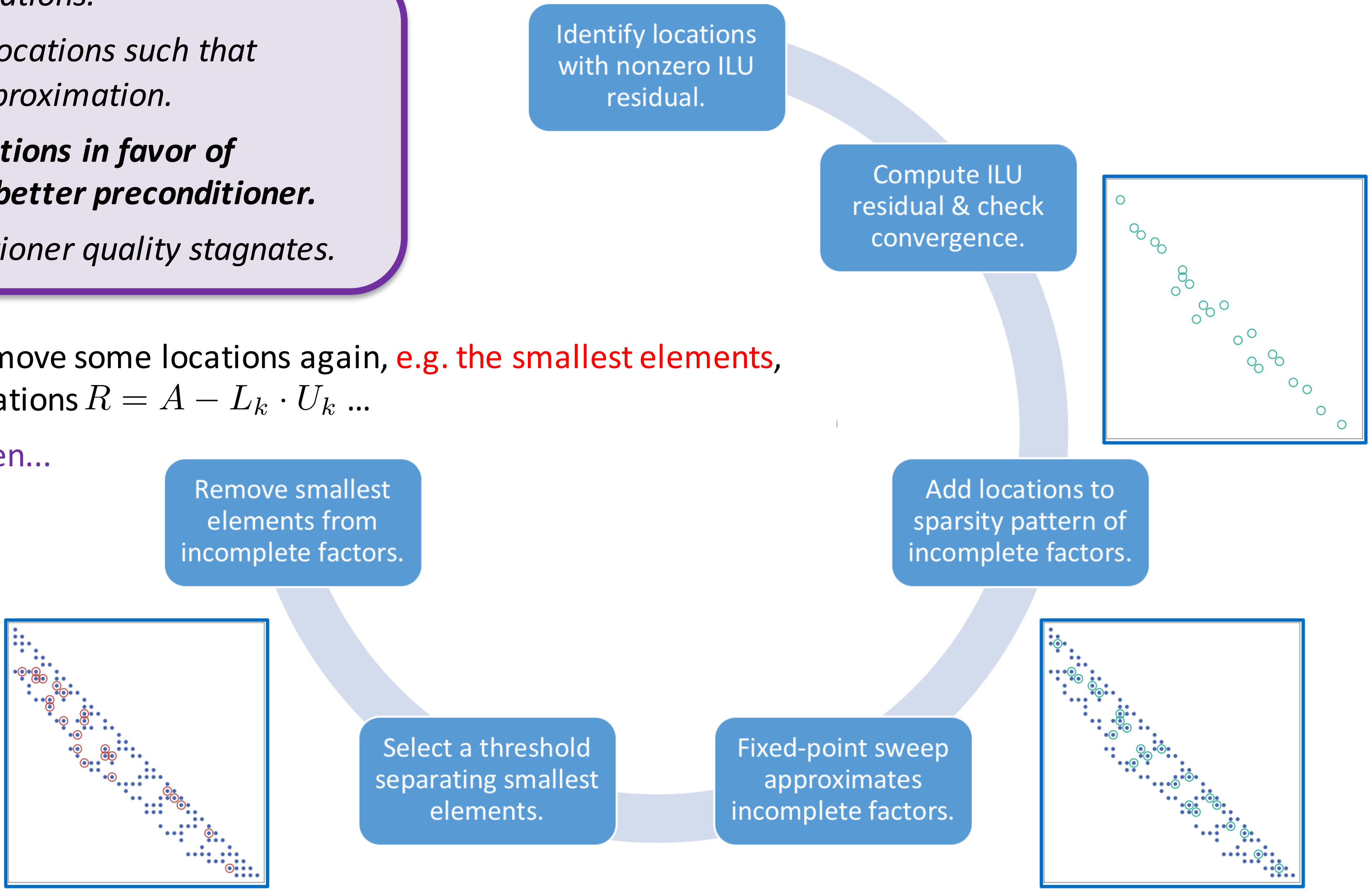
- At some point we should remove some locations again, e.g. the smallest elements, and start over looking at locations $R = A - L_k \cdot U_k$ ...



Considerations

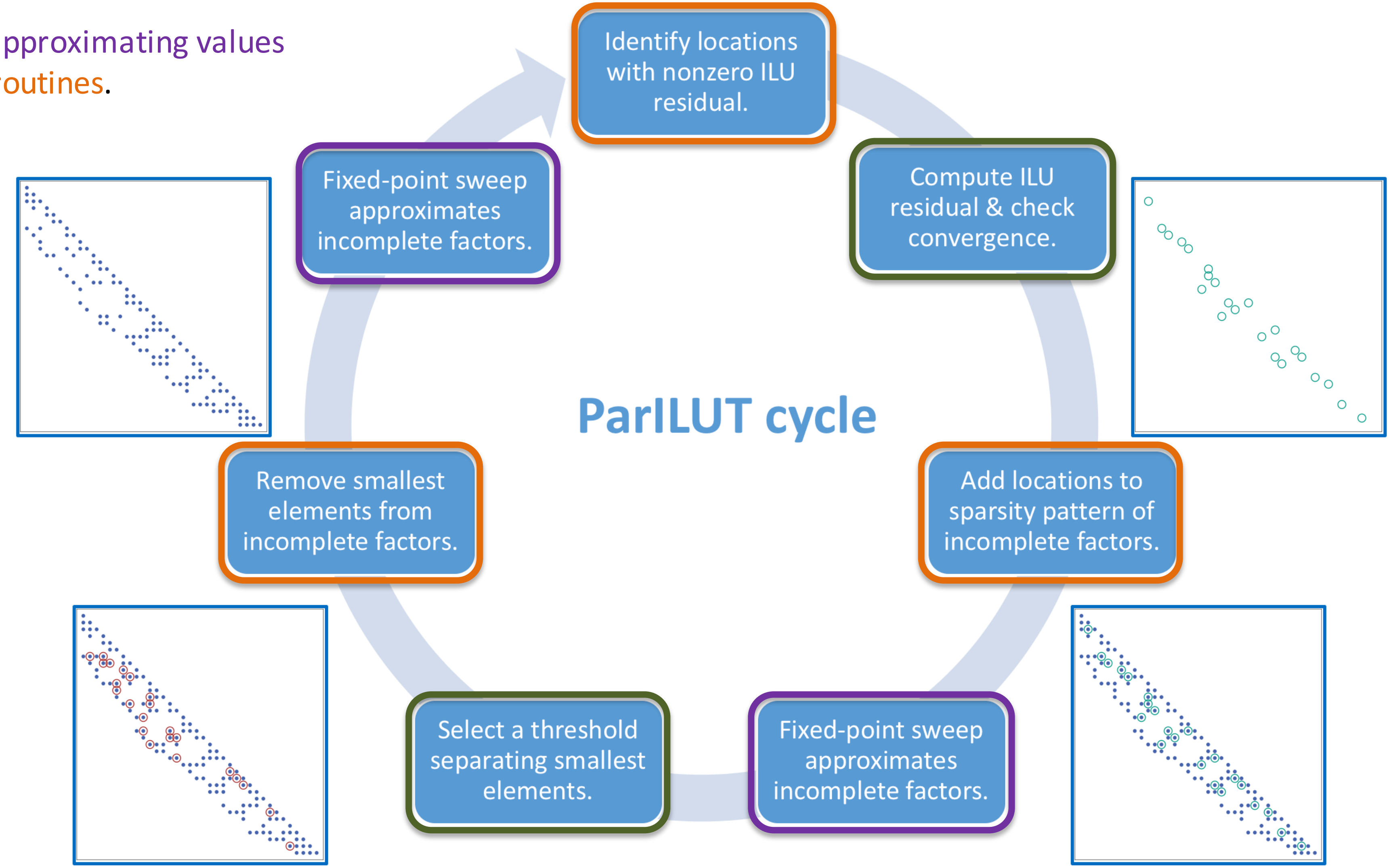
1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

- At some point we should remove some locations again, e.g. the smallest elements, and start over looking at locations $R = A - L_k \cdot U_k$ ...
- We need another sweep, then...



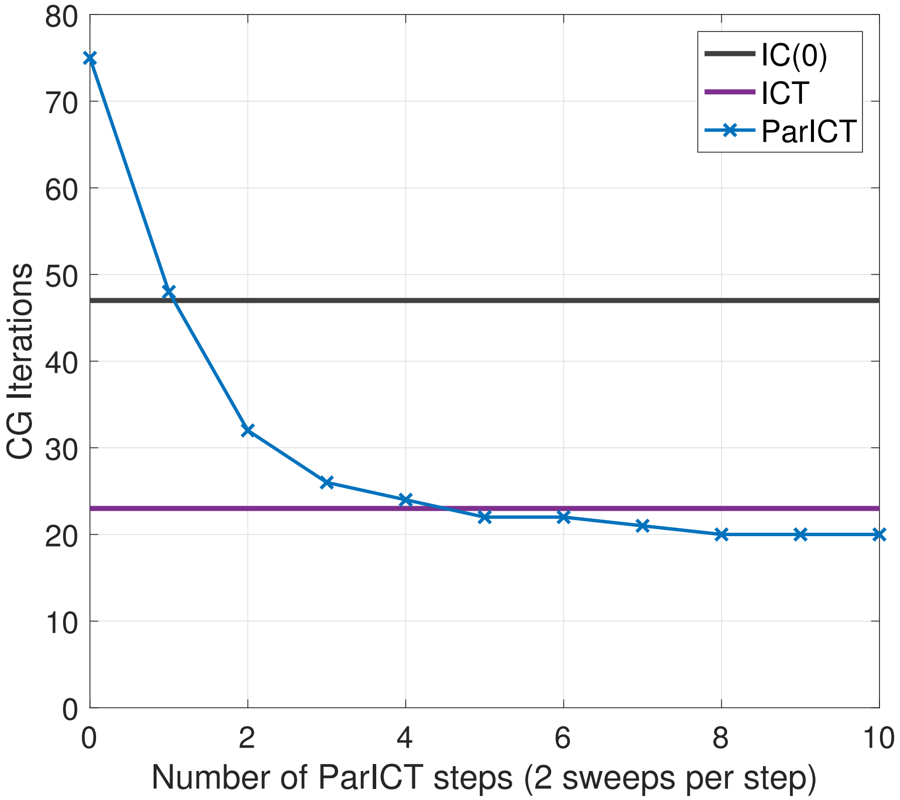
ParILUT

Interleaving fixed-point sweeps approximating values with pattern-changing symbolic routines.



ParILUT quality

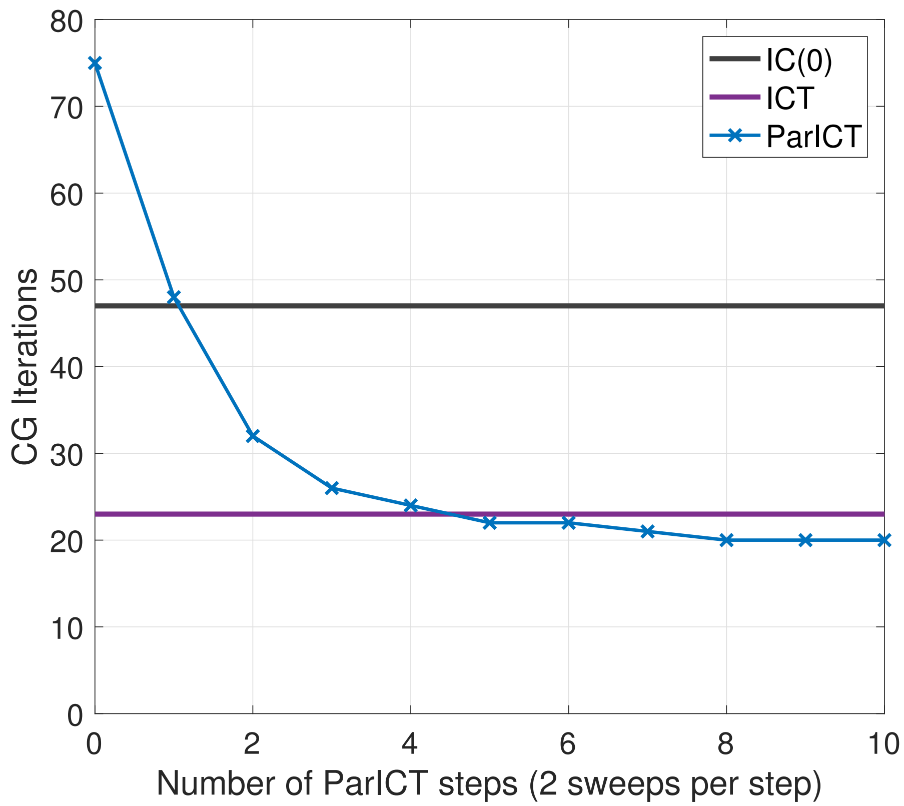
Anisotropic fluid flow problem
n: 741, nz: 4,951

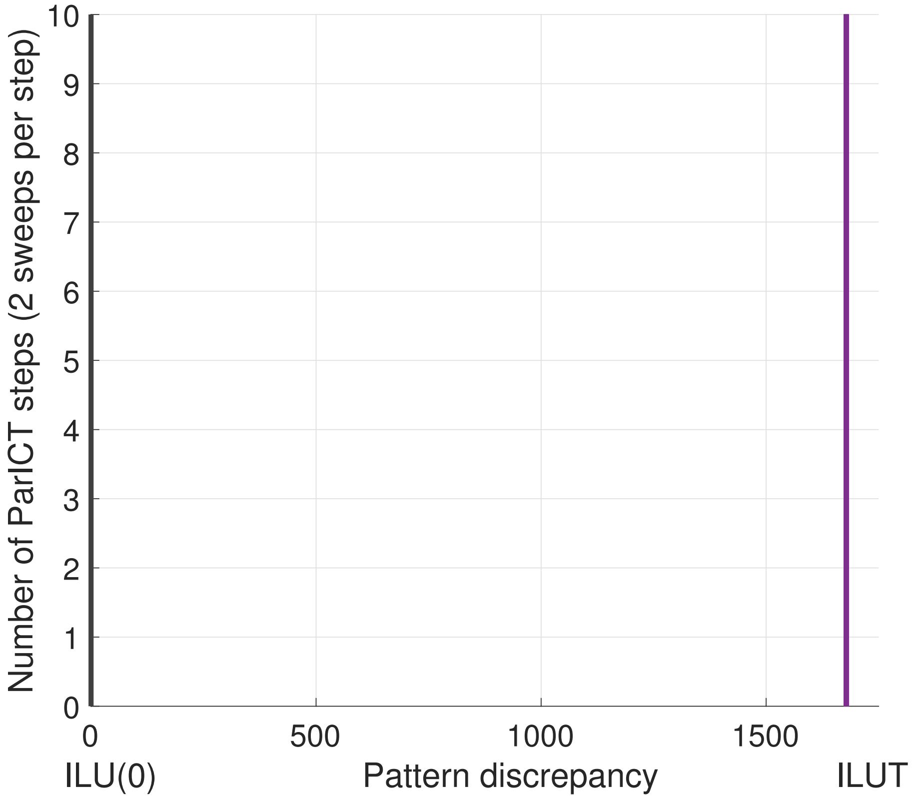


- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than ILUT?

ParILUT quality

Anisotropic fluid flow problem
n: 741, nz: 4,951

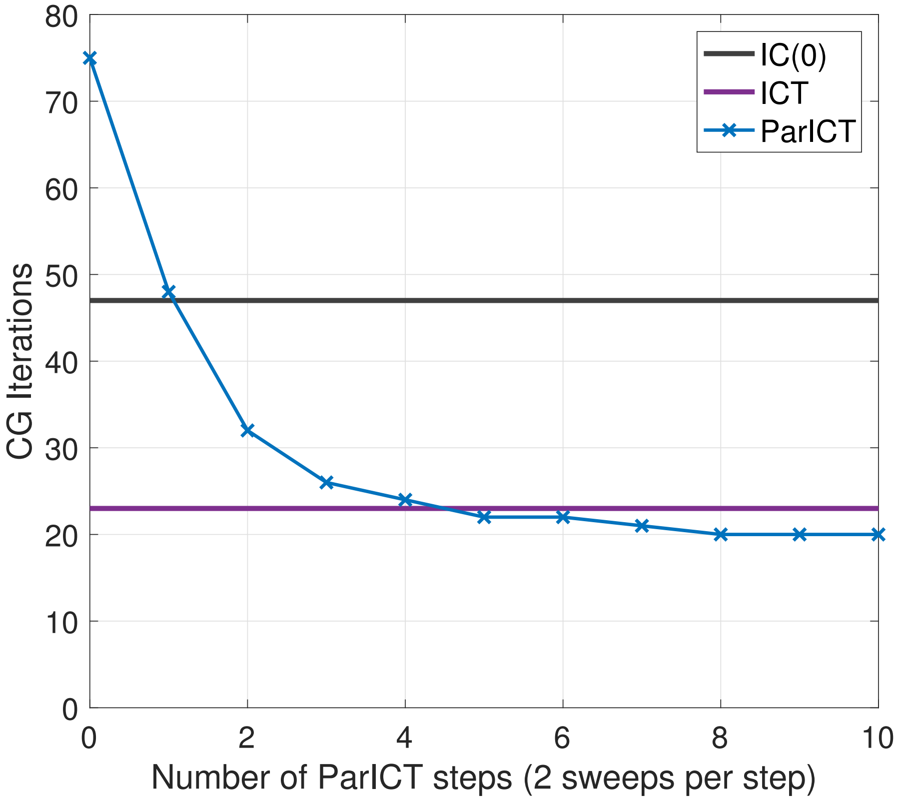


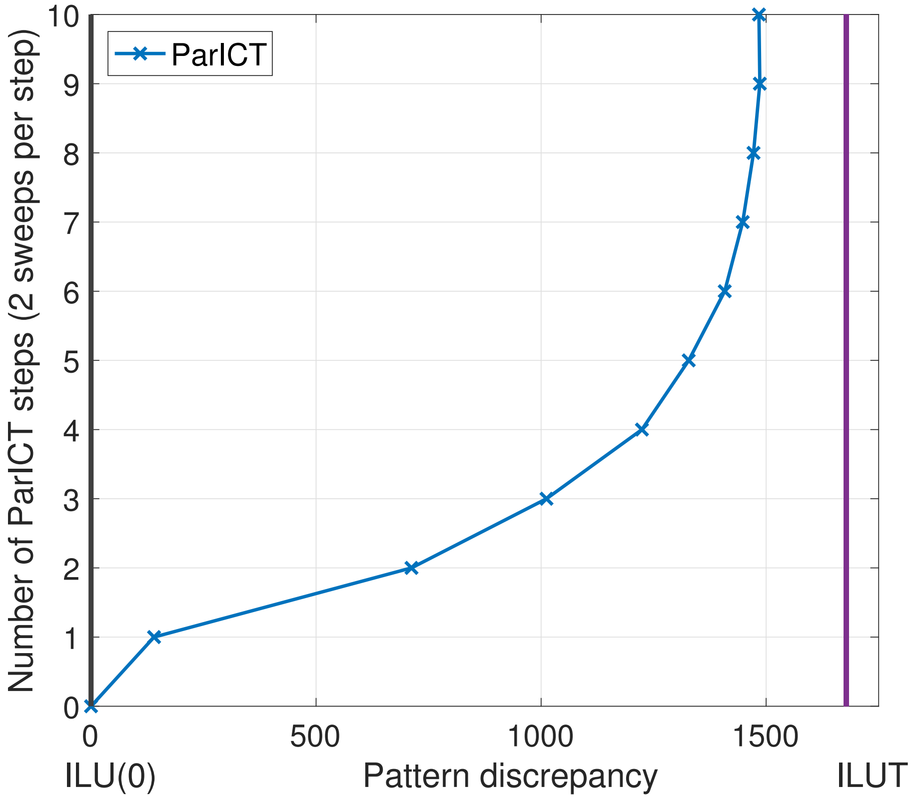


- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than ILUT?

ParILUT quality

Anisotropic fluid flow problem
n: 741, nz: 4,951



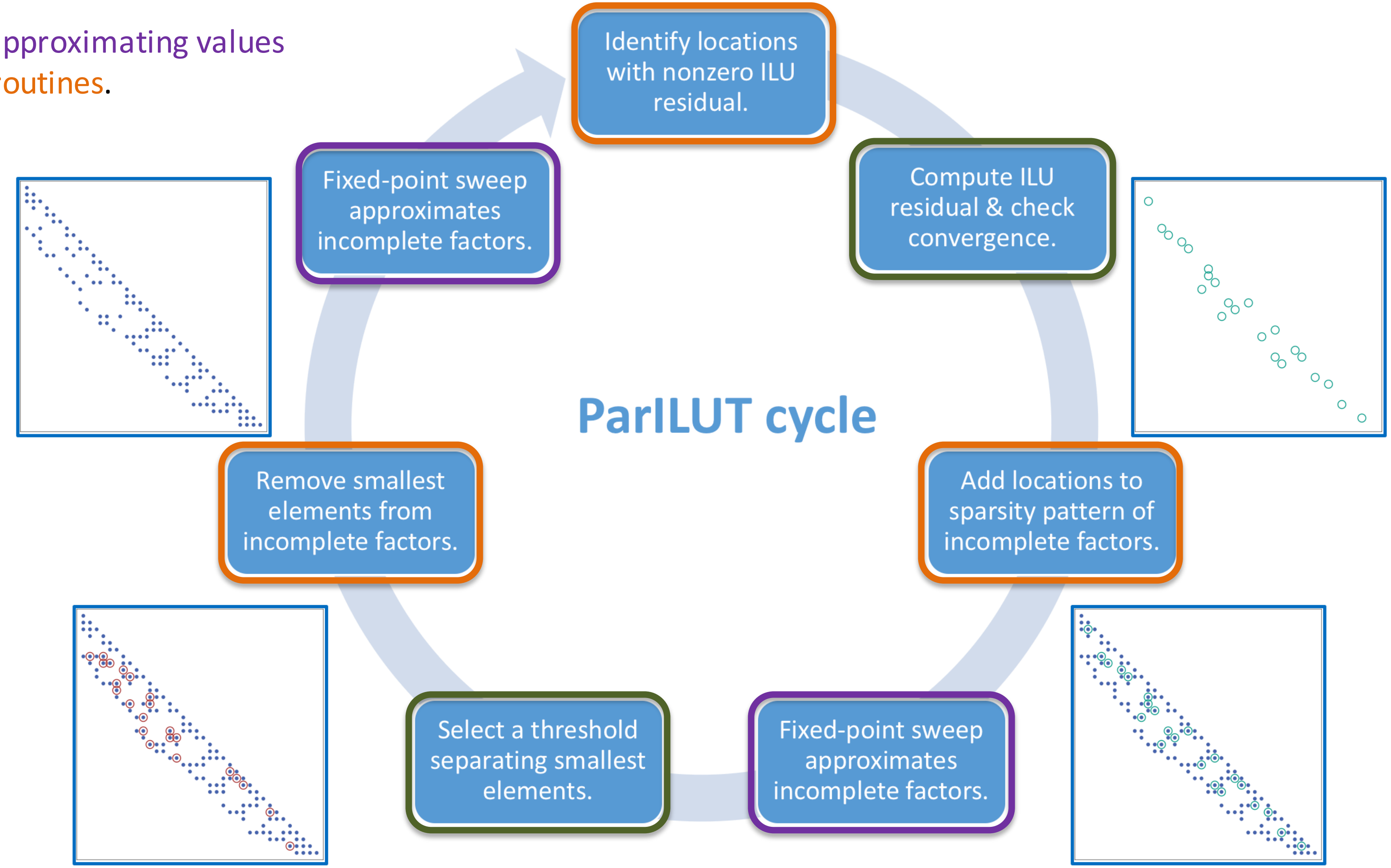


- Top-level solver iterations as quality metric.
- Few sweeps give a "better" preconditioner than ILU(0).
- Better than ILUT?

- Pattern stagnates after few sweeps.
- Pattern "more like" ILUT than ILU(0).

ParILUT

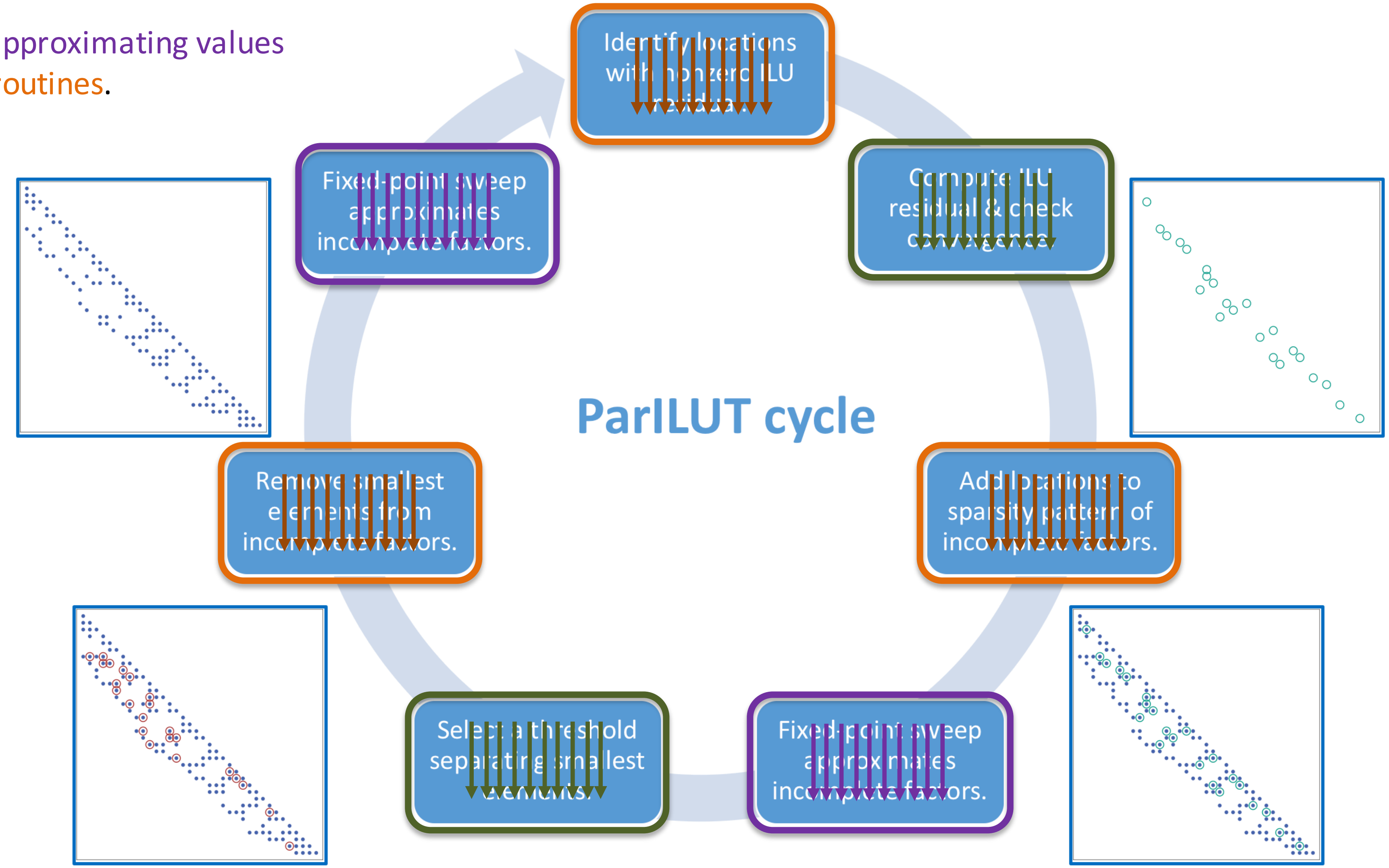
Interleaving fixed-point sweeps approximating values with pattern-changing symbolic routines.



ParILUT – a parallel threshold ILU

Parallelism inside the building blocks.

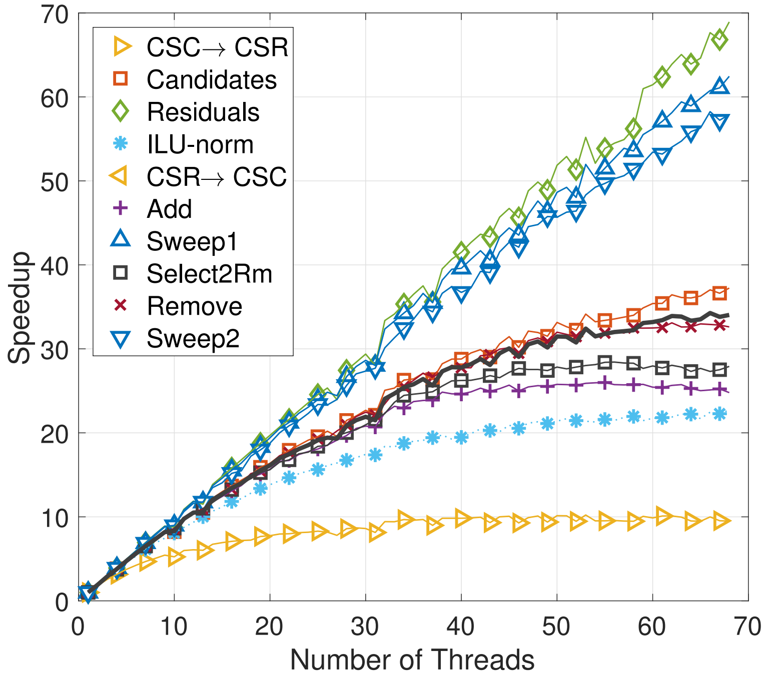
Interleaving fixed-point sweeps approximating values with pattern-changing symbolic routines.

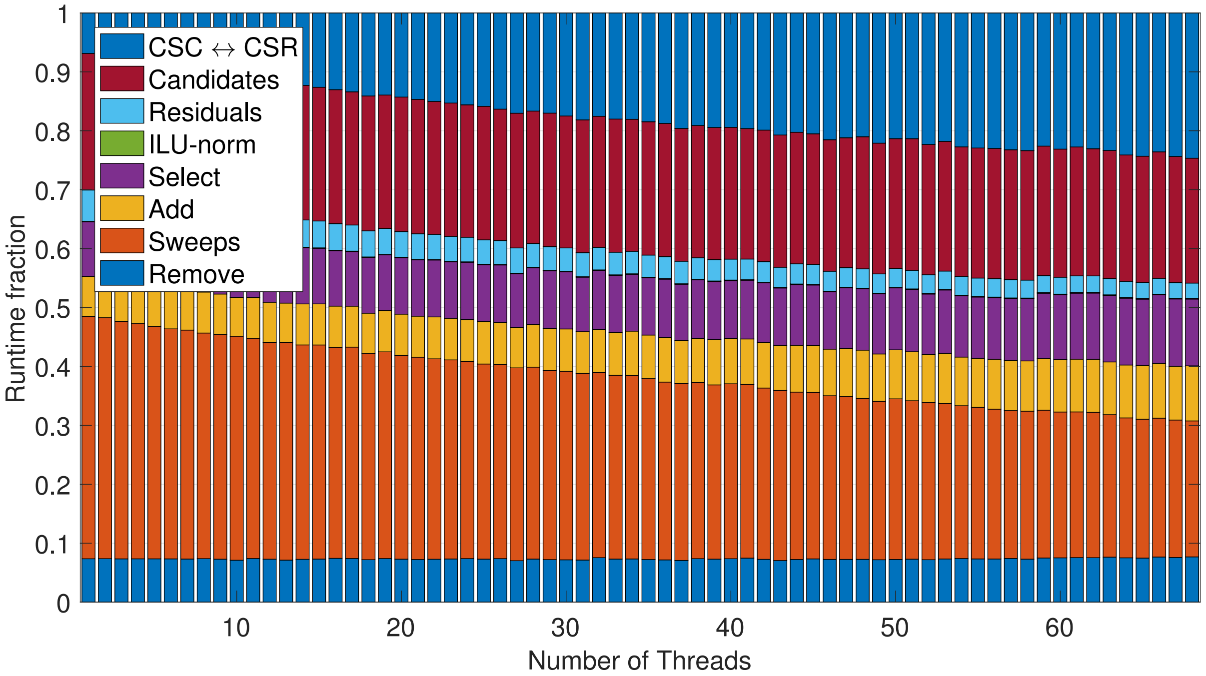


Scalability

Intel Xeon Phi 7250 "Knights Landing"
68 cores @1.40 GHz,
16GB MCDRAM @490 GB/s

`thermal2` matrix from SuiteSparse, RCM ordering, 8 el/row.



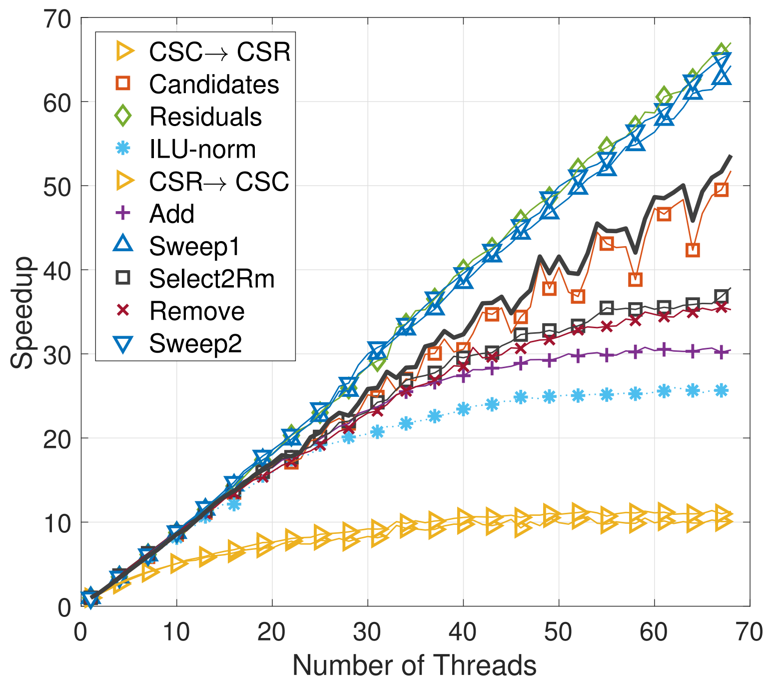


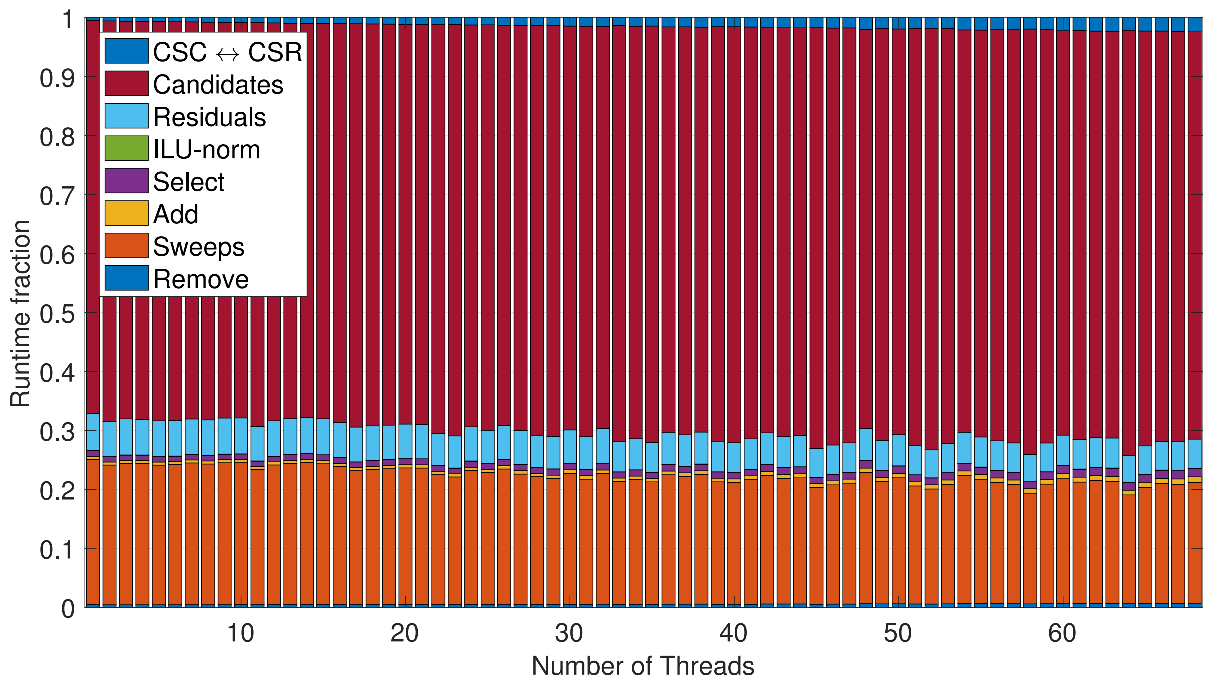
- Building blocks scale with 15% - 100% parallel efficiency.
- Transposition and sort are the bottlenecks.
- Overall speedup ~35x when using 68 KNL cores.

Scalability

Intel Xeon Phi 7250 "Knights Landing"
68 cores @1.40 GHz,
16GB MCDRAM @490 GB/s

`topopt120` matrix from topology optimization, 67 el/row.





- Building blocks scale with 15% - 100% parallel efficiency.
- Dominated by candidate search.
- Overall speedup ~52x when using 68 KNL cores.

Performance

Intel Xeon Phi 7250 "Knights Landing"
68 cores @1.40 GHz,
16GB MCDRAM @490 GB/s

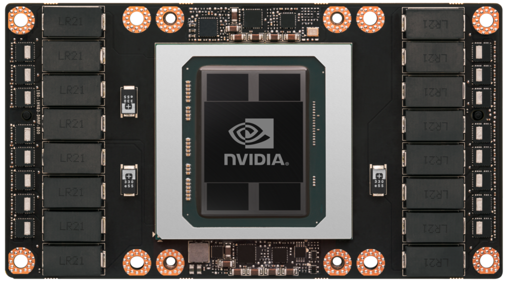
Runtime of 5 ParILUT / ParICT steps and speedup over SuperLU ILUT*.

Matrix	Origin	Rows	Nonzeros	Ratio	SuperLU	ParILUT		ParICT	
ani7	2D Anisotropic Diffusion	203,841	1,407,811	6.91	10.48 s	0.45 s	23.34	0.30 s	35.16
apache2	Suite Sparse Matrix Collect.	715,176	4,817,870	6.74	62.27 s	1.24 s	50.22	0.65 s	95.37
cage11	Suite Sparse Matrix Collect.	39,082	559,722	14.32	60.89 s	0.54 s	112.56	--	
jacobianMat9	Fun3D Fluid Flow Problem	90,708	5,047,042	55.64	153.84 s	7.26 s	21.19	--	
thermal2	Thermal Problem (Suite Sp.)	1,228,045	8,580,313	6.99	91.83 s	1.23 s	74.66	0.68 s	134.25
tmt_sym	Suite Sparse Matrix Collect.	726,713	5,080,961	6.97	53.42 s	0.70 s	76.21	0.41 s	131.25
topopt120	Geometry Optimization	132,300	8,802,544	66.53	44.22 s	14.40 s	3.07	8.24 s	5.37
torso2	Suite Sparse Matrix Collect.	115,967	1,033,473	8.91	10.78 s	0.27 s	39.92	--	
venkat01	Suite Sparse Matrix Collect.	62,424	1,717,792	27.52	8.53 s	0.74 s	11.54	--	

*We thank Sherry Li and Meiyue Shao for technical help in generating the performance numbers.

How about GPUs?

- Fine-grained parallelism
- High bandwidth for coalescent reads
- No deep cache hierarchy
- We need to oversubscribe cores for hiding latency

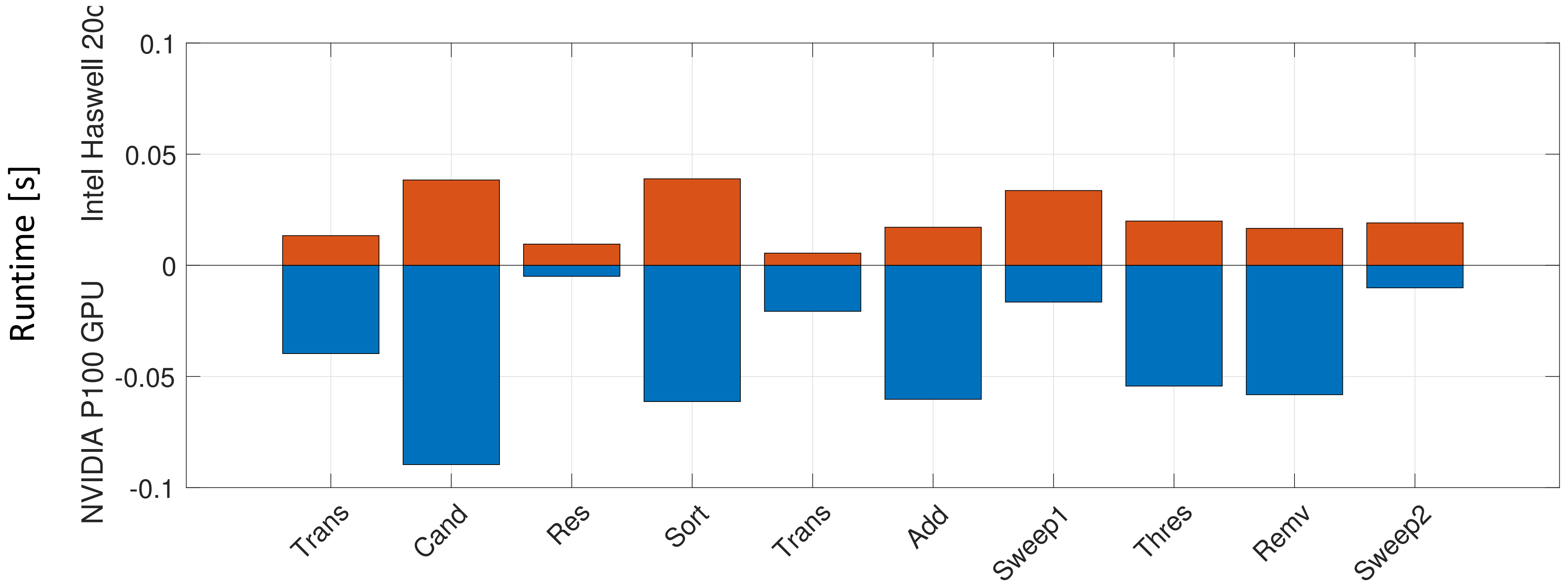


NVIDIA P100 “Pascal”
4.7 TFLOP/s DP
16GB RAM @732 GB/s

How about GPUs?

- Fine-grained parallelism
- High bandwidth for coalescent reads
- No deep cache hierarchy
- We need to oversubscribe cores for hiding latency

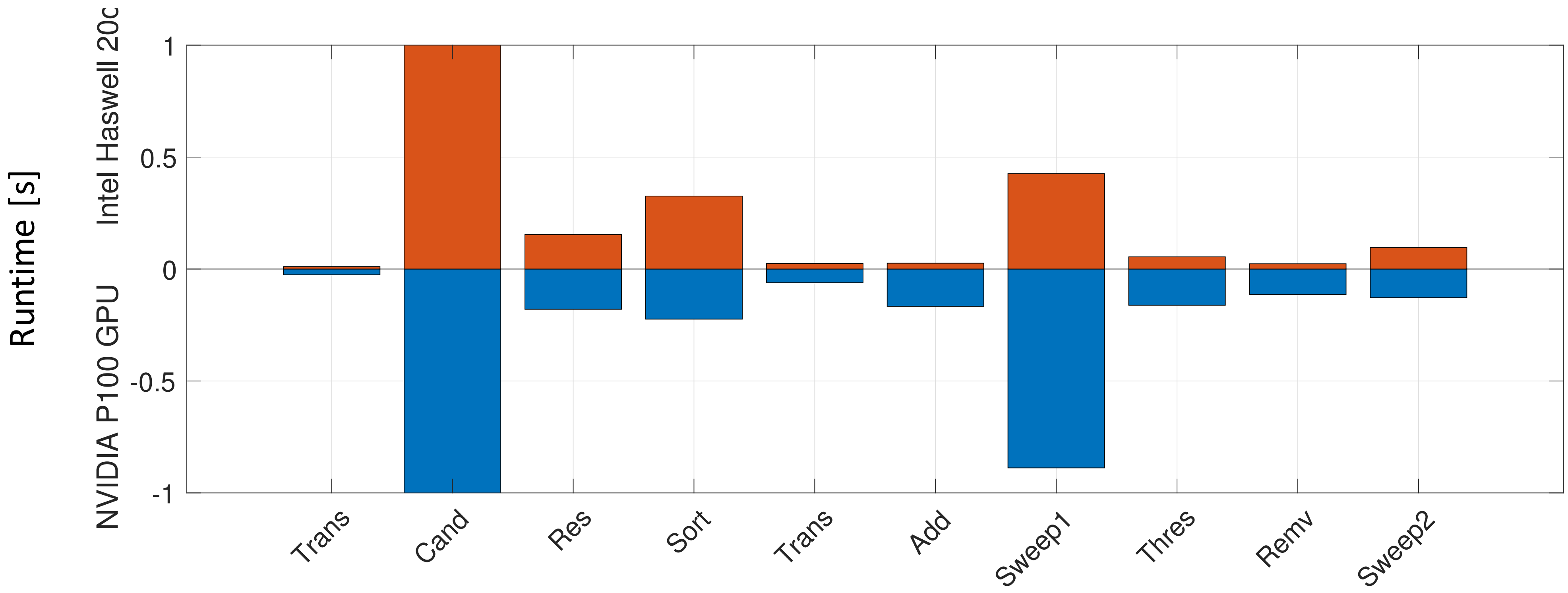
`thermal2` matrix from SuiteSparse, RCM ordering, 8 el/row.



How about GPUs?

- Fine-grained parallelism
- High bandwidth for coalescent reads
- No deep cache hierarchy
- We need to oversubscribe cores for hiding latency

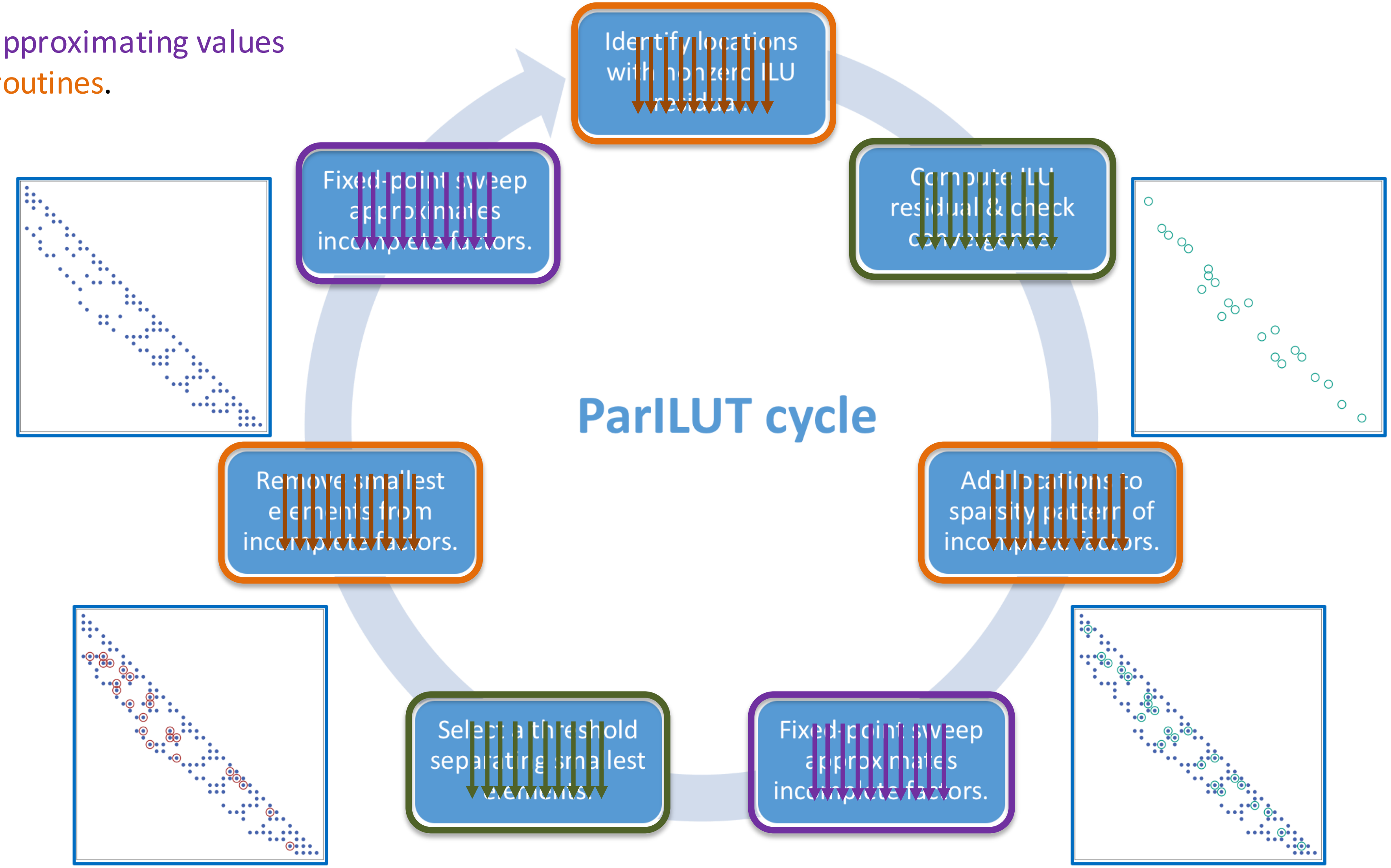
`topopt120` matrix from topology optimization, 67 el/row.



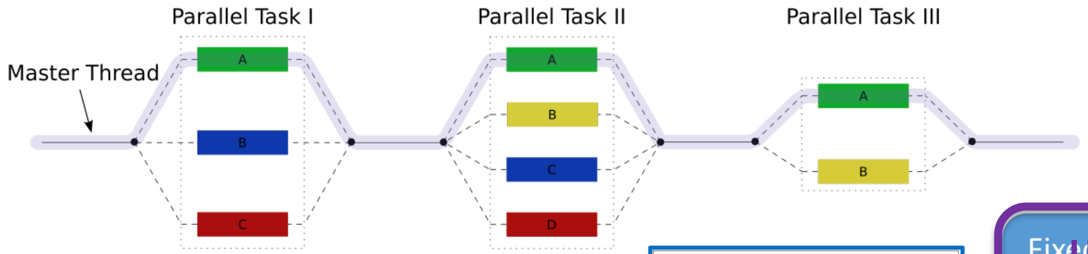
Is this a future-oriented algorithm?

Parallelism inside the building blocks.

Interleaving fixed-point sweeps approximating values with pattern-changing symbolic routines.

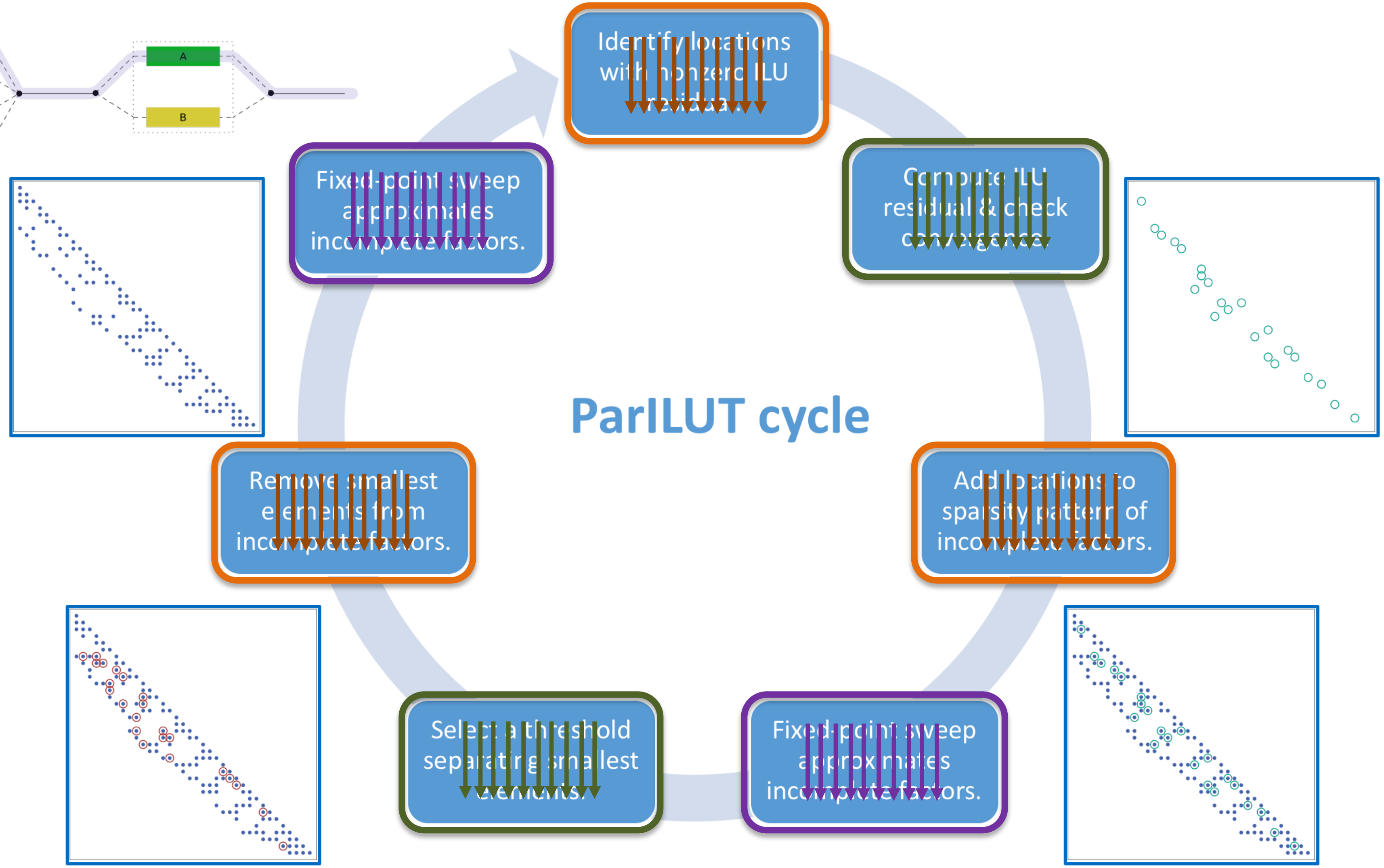


Is this a future-oriented algorithm?

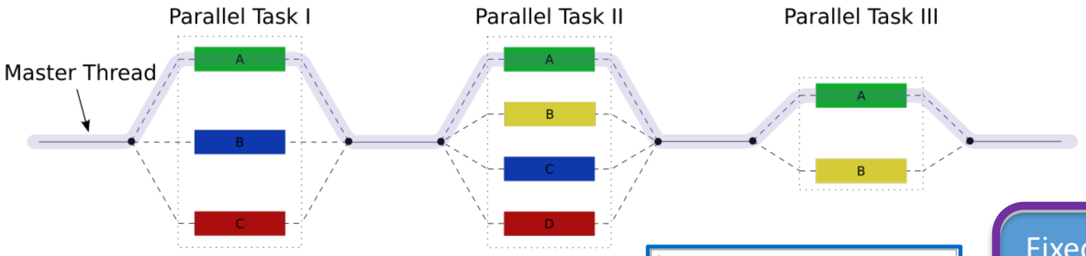


Bulk-Synchronous Algorithm!

…see John Shalf on Thursday…

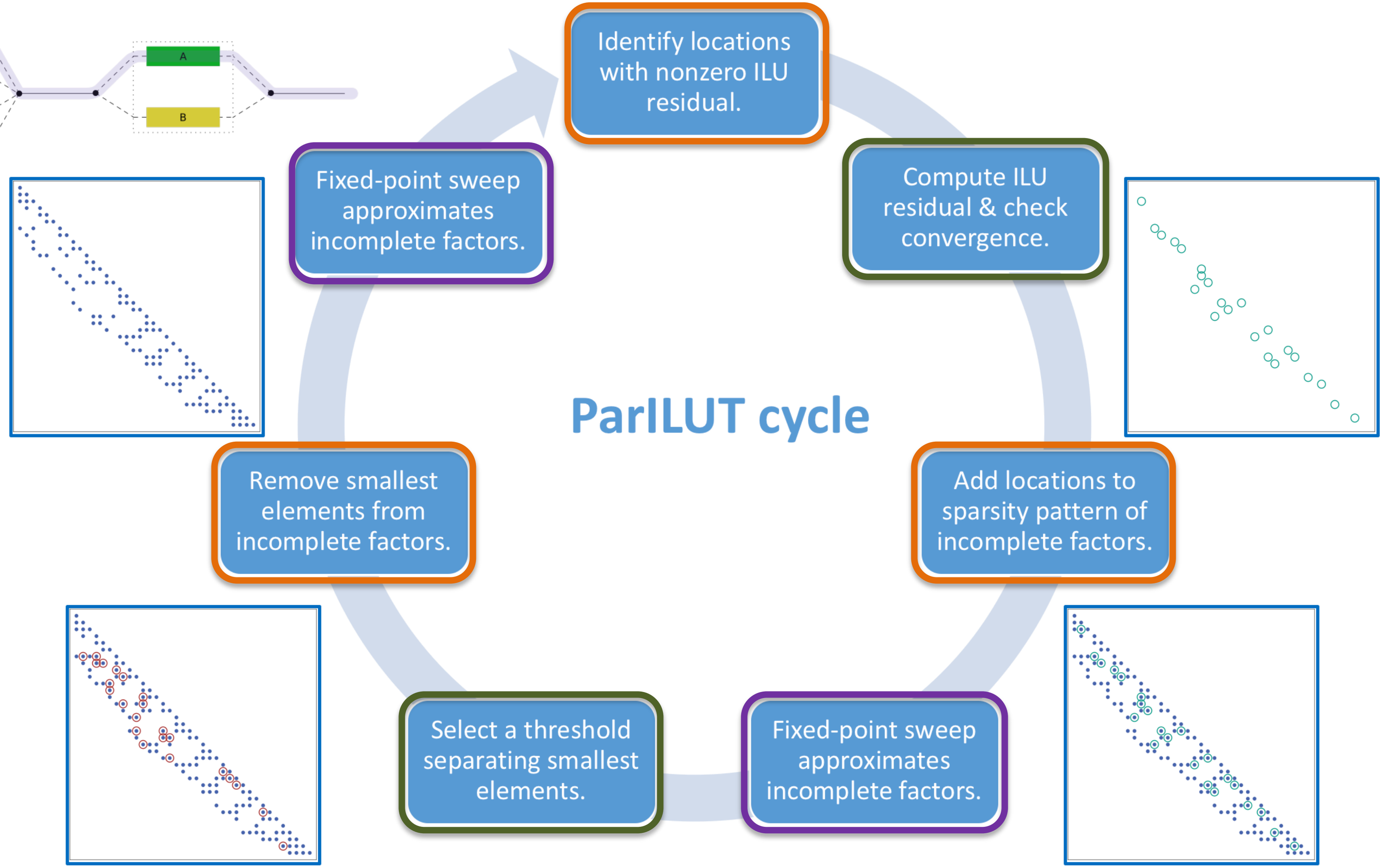


Is this a future-oriented algorithm?

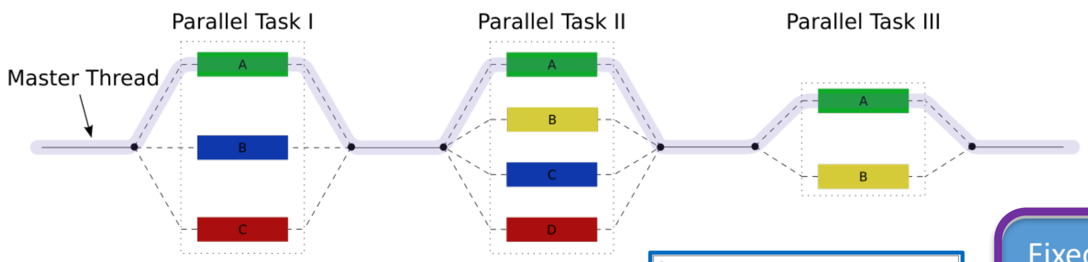


Bulk-Synchronous Algorithm!

Do we need that?

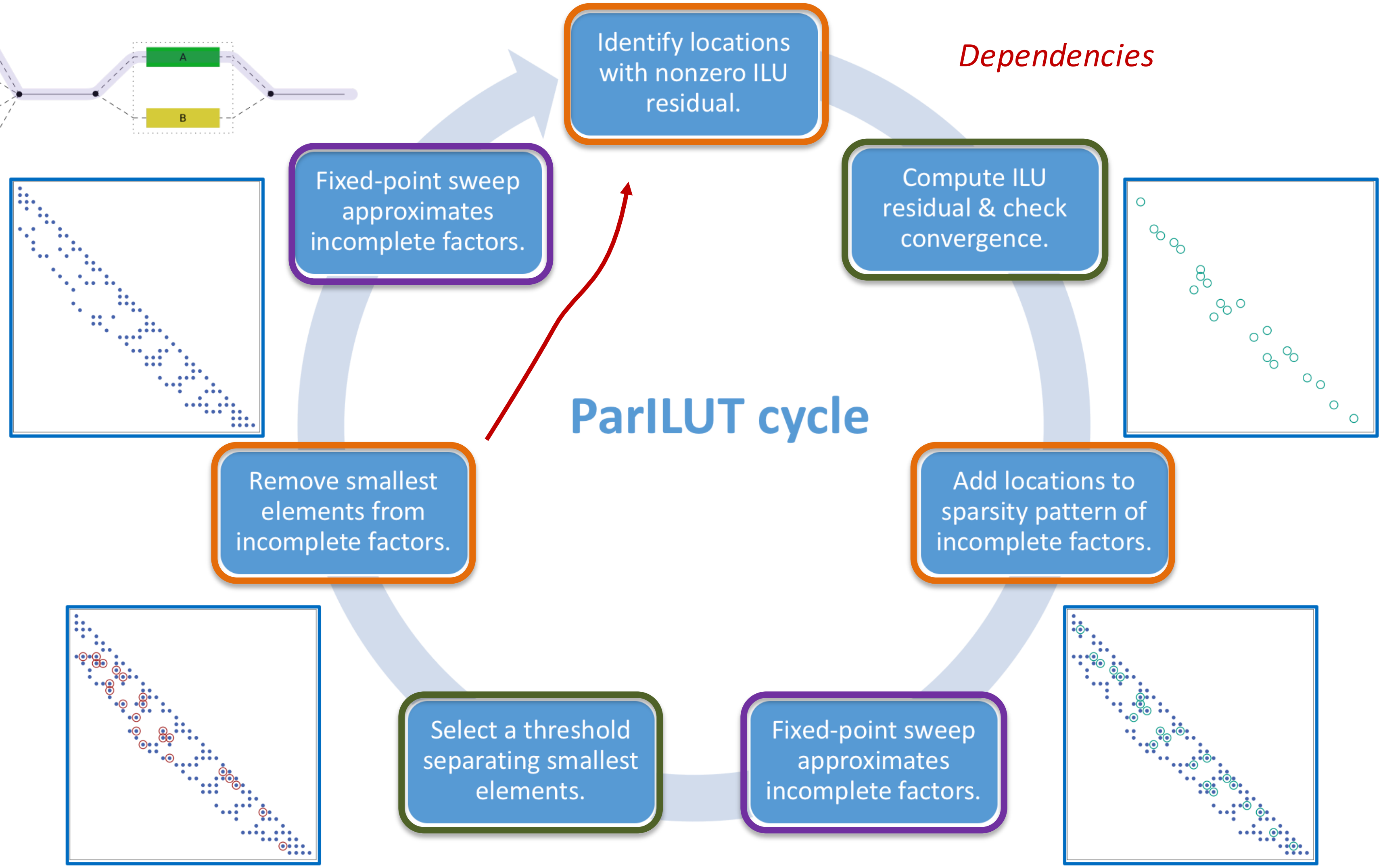


Is this a future-oriented algorithm?

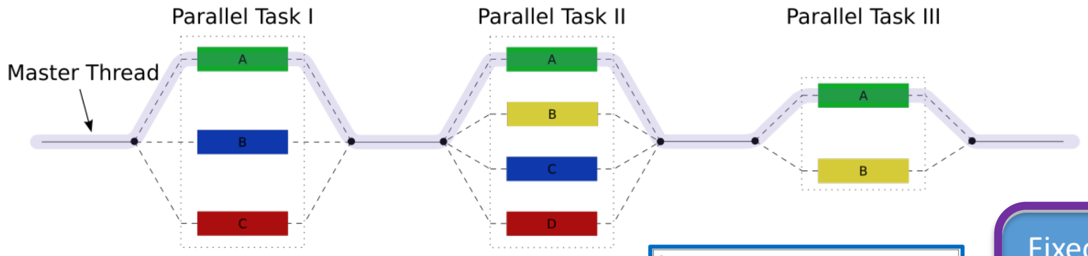


Bulk-Synchronous Algorithm!

Do we need that?

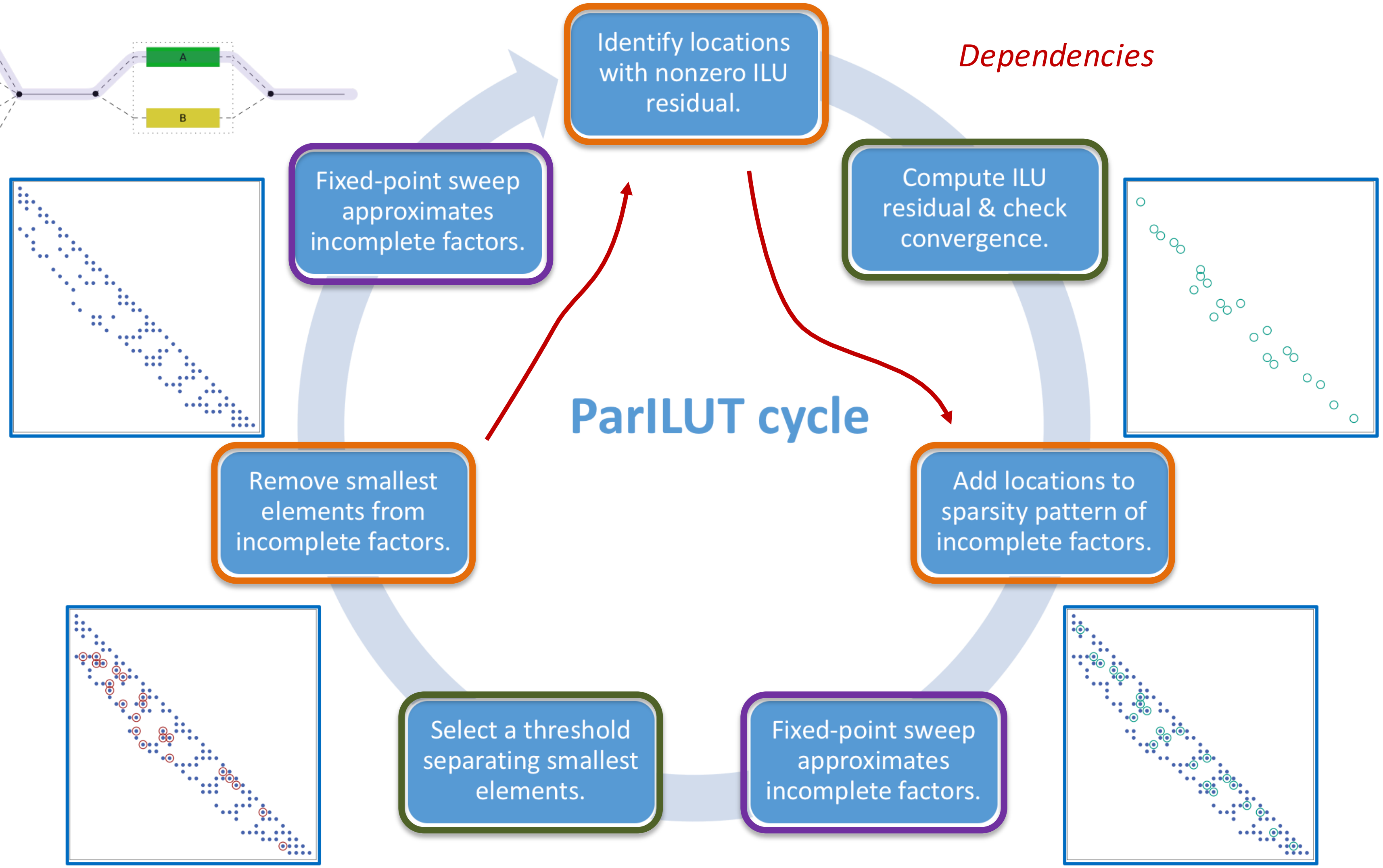


Is this a future-oriented algorithm?

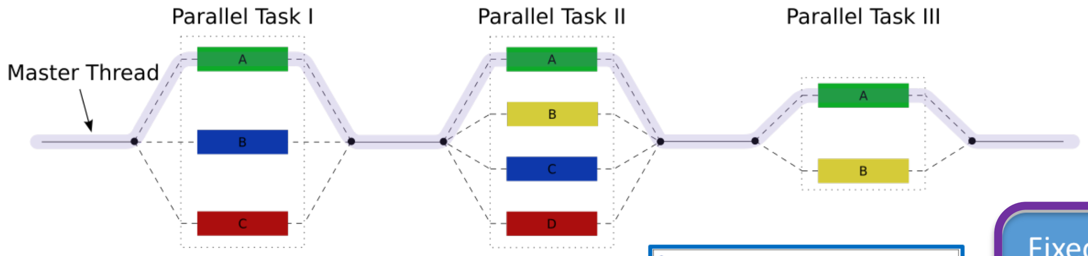


Bulk-Synchronous Algorithm!

Do we need that?

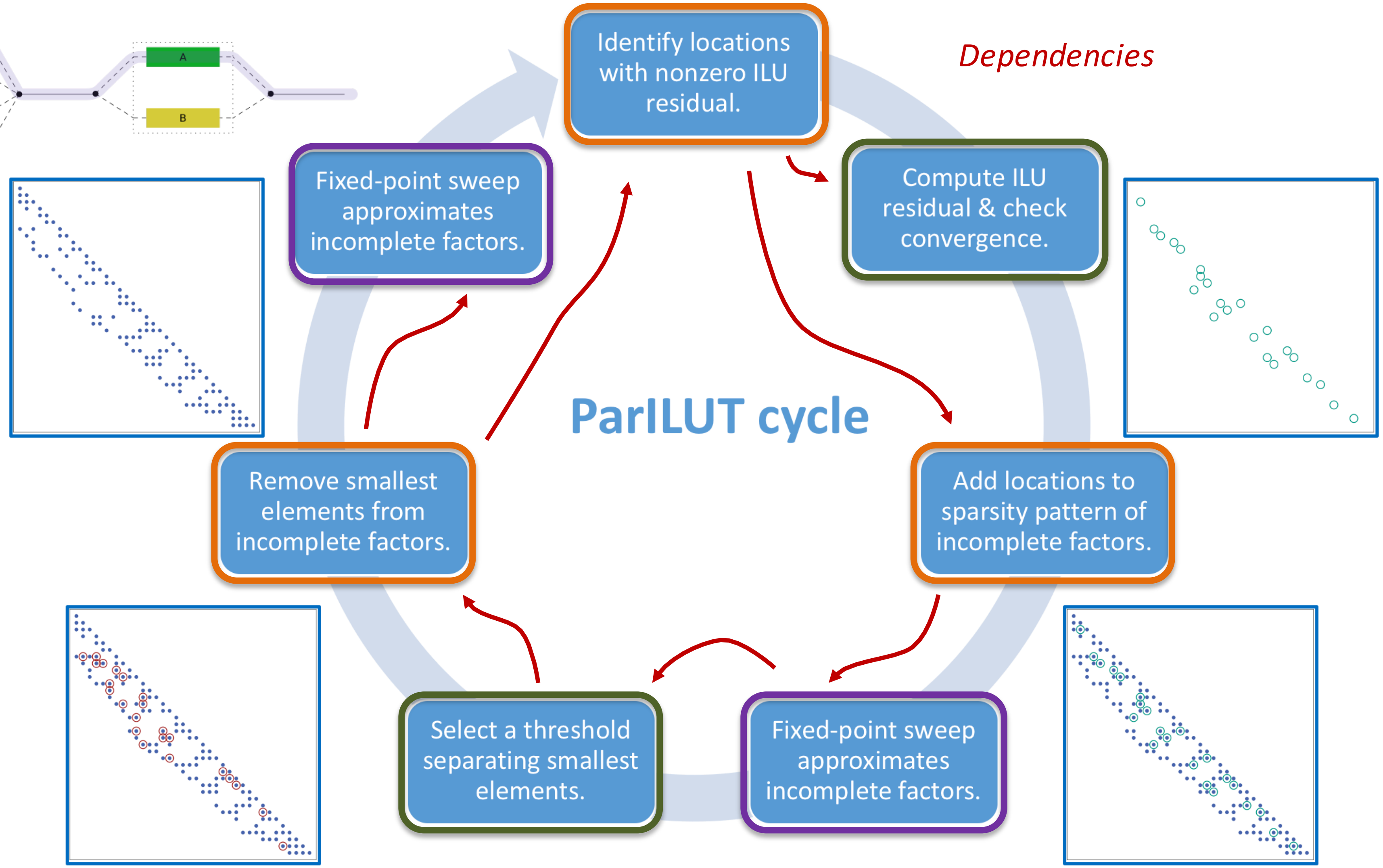


Is this a future-oriented algorithm?

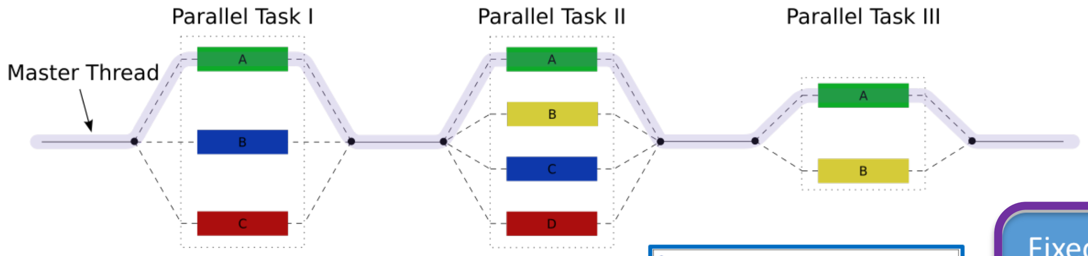


Bulk-Synchronous Algorithm!

Do we need that?

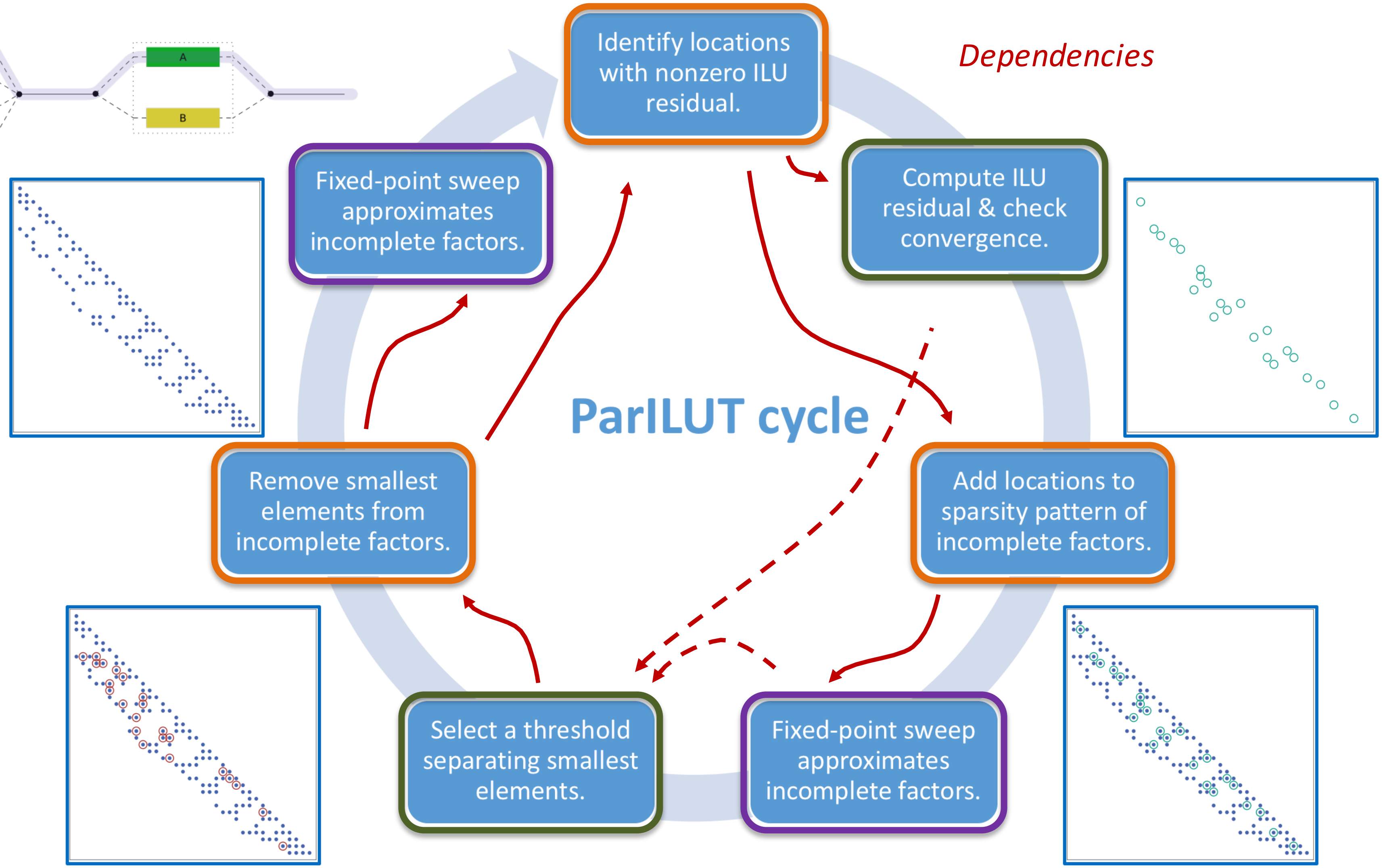


Is this a future-oriented algorithm?

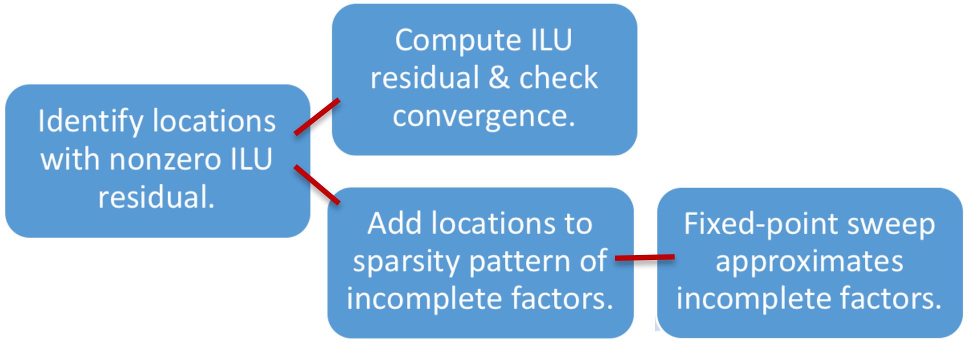


Bulk-Synchronous Algorithm!

Do we need that?



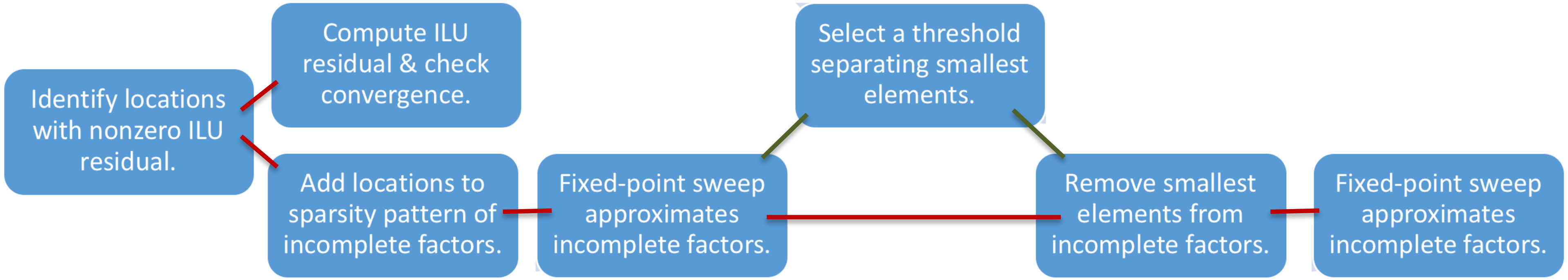
Is this a future-oriented algorithm?



Strong dependency – *we can not start before finished.*

Weak dependency – *if we start before: +/- few nonzeros.*

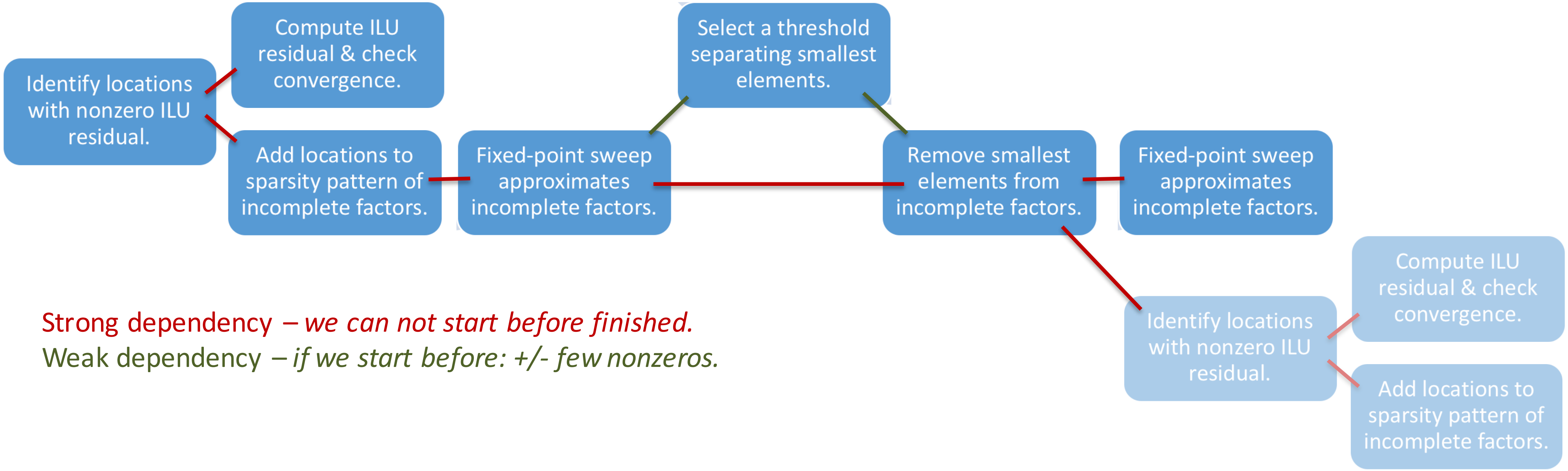
Is this a future-oriented algorithm?



Strong dependency – *we can not start before finished.*

Weak dependency – *if we start before: +/- few nonzeros.*

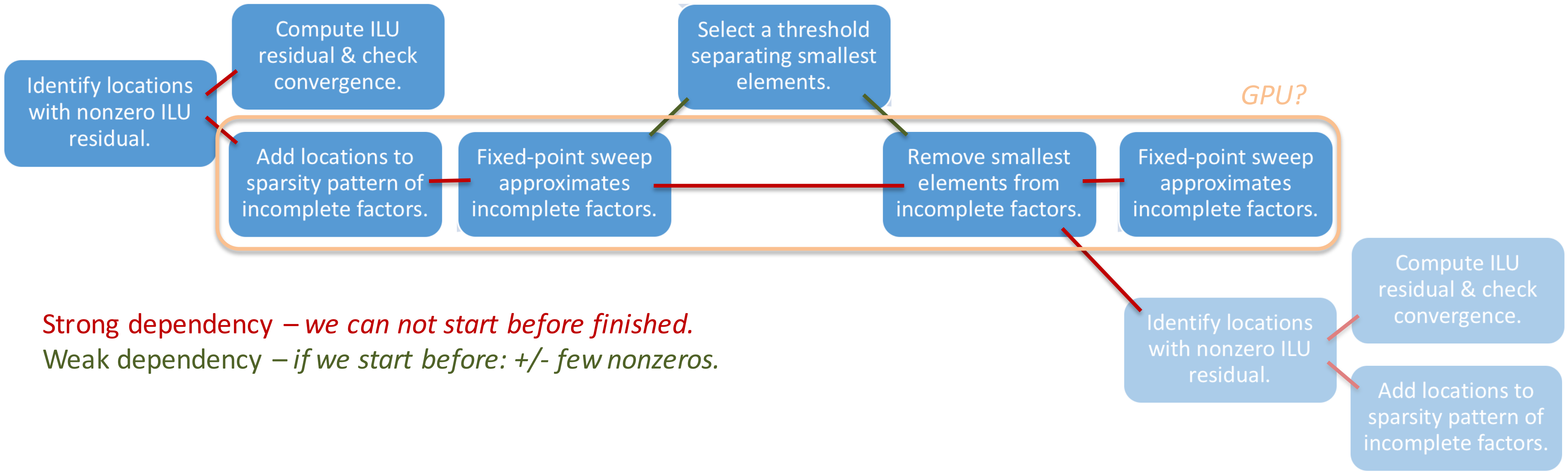
Is this a future-oriented algorithm?



Strong dependency – *we can not start before finished.*

Weak dependency – *if we start before: +/- few nonzeros.*

Is this a future-oriented algorithm?



Strong dependency – *we can not start before finished.*
Weak dependency – *if we start before: +/- few nonzeros.*

Excellent candidate for hybrid hardware?
Asynchronous execution?

Is this a future-oriented algorithm?

- **Hybrid ParILUT** version utilizing GPU and CPU, overlapping communication & computation.
- **Asynchronous** version relaxing dependencies.
- Use a **different sparsity-pattern generator**:
  - Randomized?
  - Machine learning techniques?
- **Increasing fill-in** towards "full" factorization.
- ParILUT routines available in MAGMA-sparse – they will be in Ginkgo.

Karlsruher Institut für Technologie

THE UNIVERSITY OF TENNESSEE KNOXVILLE

This research was sponsored by:

The Exascale Computing Project

A Collaborative effort of the U.S. Department of Energy Office of Science And the National Nuclear Security Administration

U.S. Department of Energy
ASCR Award Number DE-SC0016513

Helmholtz Impuls und Vernetzungsfond
VH-NG-1241

Test matrices

Matrix	Origin	SPD	Num. Rows	Nz	Nz/Row
ANI5	2D anisotropic diffusion	yes	12,561	86,227	6.86
ANI6	2D anisotropic diffusion	yes	50,721	349,603	6.89
ANI7	2D anisotropic diffusion	yes	203,841	1,407,811	6.91
APACHE1	Suite Sparse [10]	yes	80,800	542,184	6.71
APACHE2	Suite Sparse	yes	715,176	4,817,870	6.74
CAGE10	Suite Sparse	no	11,397	150,645	13.22
CAGE11	Suite Sparse	no	39,082	559,722	14.32
JACOBIANMAT0	Fun3D fluid flow [20]	no	90,708	5,047,017	55.64
JACOBIANMAT9	Fun3D fluid flow	no	90,708	5,047,042	55.64
MAJORBASIS	Suite Sparse	no	160,000	1,750,416	10.94
TOPOPT010	Geometry optimization [24]	yes	132,300	8,802,544	66.53
TOPOPT060	Geometry optimization	yes	132,300	7,824,817	59.14
TOPOPT120	Geometry optimization	yes	132,300	7,834,644	59.22
THERMAL1	Suite Sparse	yes	82,654	574,458	6.95
THERMAL2	Suite Sparse	yes	1,228,045	8,580,313	6.99
THERMOMECH_TC	Suite Sparse	yes	102,158	711,558	6.97
THERMOMECH_DM	Suite Sparse	yes	204,316	1,423,116	6.97
TMT_SYM	Suite Sparse	yes	726,713	5,080,961	6.99
TORSO2	Suite Sparse	no	115,967	1,033,473	8.91
VENKAT01	Suite Sparse	no	62,424	1,717,792	27.52

Convergence: GMRES iterations

Matrix	no prec.	ILU(0)	ILUT	ParILUT 0	1	2	3	4	5
ANI5	882	172	78	278	161	105	84	74	66
ANI6	1,751	391	127	547	315	211	168	143	131
ANI7	3,499	828	290	1,083	641	459	370	318	289
CAGE10	20	8	8	9	7	8	8	8	8
CAGE11	21	9	8	9	7	7	7	7	7
JACOBIANMAT0	315	40	34	63	36	33	33	33	33
JACOBIANMAT9	539	66	65	110	60	55	54	53	53
MAJORBASIS	95	15	9	26	12	11	11	11	11
TOPOPT010	2,399	565	303	835	492	375	348	340	339
TOPOPT060	2,852	666	397	963	584	445	417	412	410
TOPOPT120	2,765	668	396	959	584	445	416	408	408
TORSO2	46	10	7	18	8	6	7	7	7
VENKAT01	195	22	17	42	18	17	17	17	17

Convergence: CG iterations

Matrix	no prec.	IC(0)	ICT	ParICT 0	ParICT 1	ParICT 2	ParICT 3	ParICT 4	ParICT 5
ANI5	951	226	–	297	184	136	108	93	86
ANI6	1,926	621	–	595	374	275	219	181	172
ANI7	3,895	1,469	–	1,199	753	559	455	405	377
APACHE1	3,727	368	331	1,480	933	517	321	323	323
APACHE2	4,574	1,150	785	1,890	1,197	799	766	760	754
THERMAL1	1,640	453	412	626	447	409	389	385	383
THERMAL2	6,253	1,729	1,604	2,372	1,674	1,503	1,457	1,472	1,433
THERMOMECH_DM	21	8	8	8	7	7	7	7	7
THERMOMECH_TC	21	8	7	8	7	7	7	7	7
TMT_SYM	5,481	1,453	1,185	1,963	1,234	1,071	1,012	992	1,004
TOPOPT010	2,613	692	331	845	551	402	342	316	313
TOPOPT060	3,123	871	–	988	749	693	1,116	–	–
TOPOPT120	3,062	886	–	991	837	784	2,185	–	–