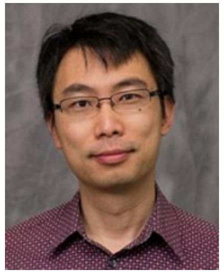
33rd IEEE International Parallel and Distributed Computing Symposium
May 21st, 2019 | Rio de Janeiro

Karlsruher Institut für Technologie

# ParILUT - A Parallel Threshold ILU for GPUs

Hartwig Anzt, Tobias Ribizel, Goran Flegar, Edmond Chow, Jack Dongarra




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 does no longer improve for the nonzero count.*

# 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…

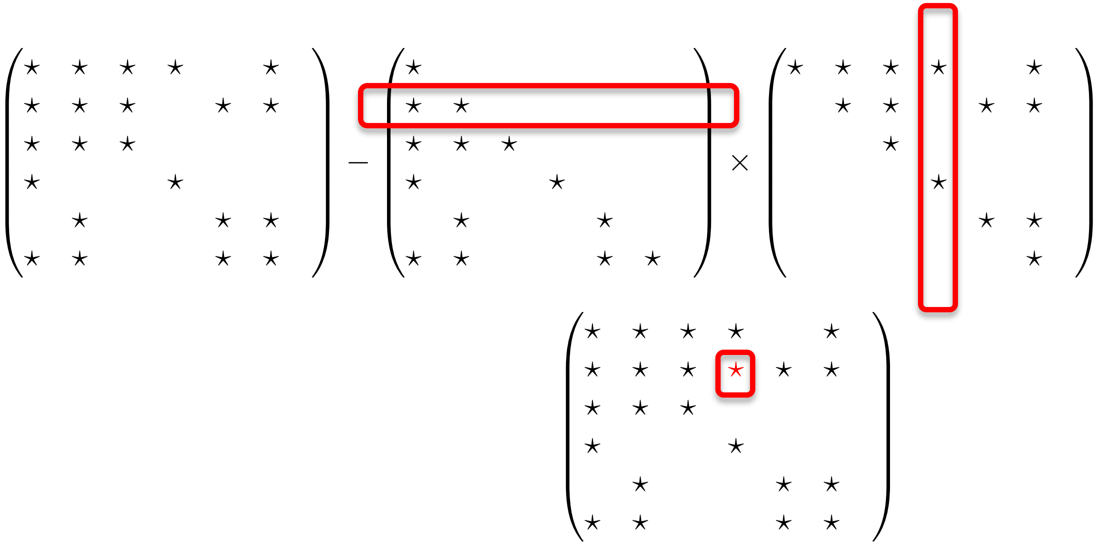
$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\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

**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.*

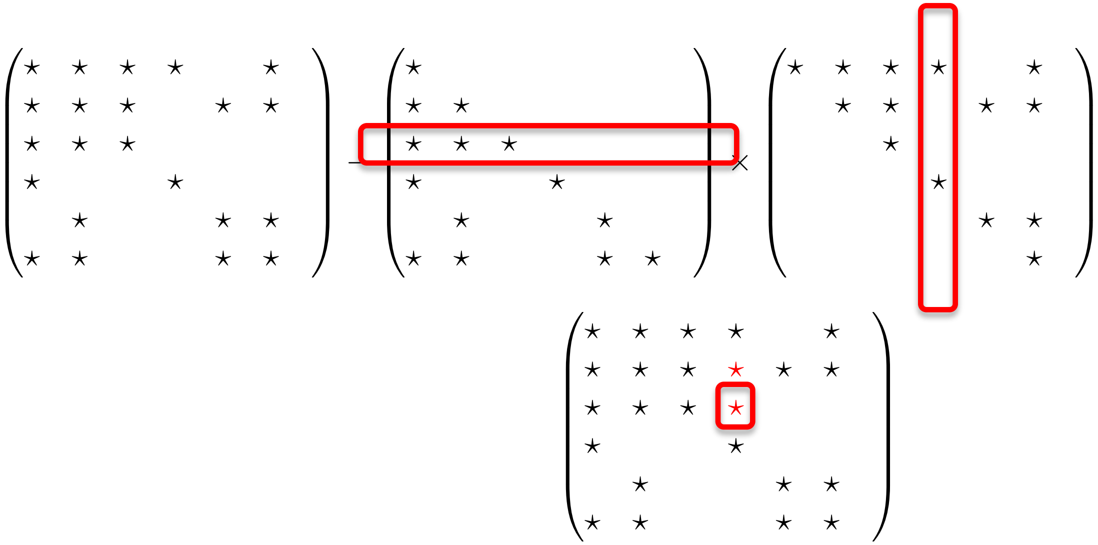
- This is an optimization problem…



# 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…

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

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

# 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…

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

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

# 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.

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

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

# 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

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

# 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.***

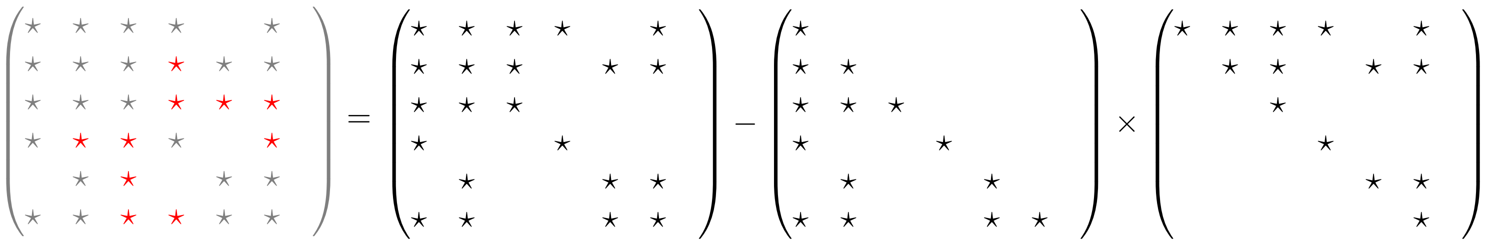
Fixed-point sweep approximates incomplete factors.

# 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.***

Compute ILU residual & check convergence.

- Maybe use the ILU residual as quality metric.



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.*

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

Compute ILU residual & check convergence.

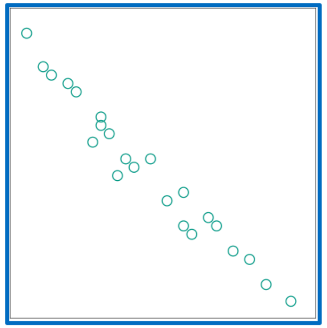
Fixed-point sweep approximates incomplete factors.

# Considerations

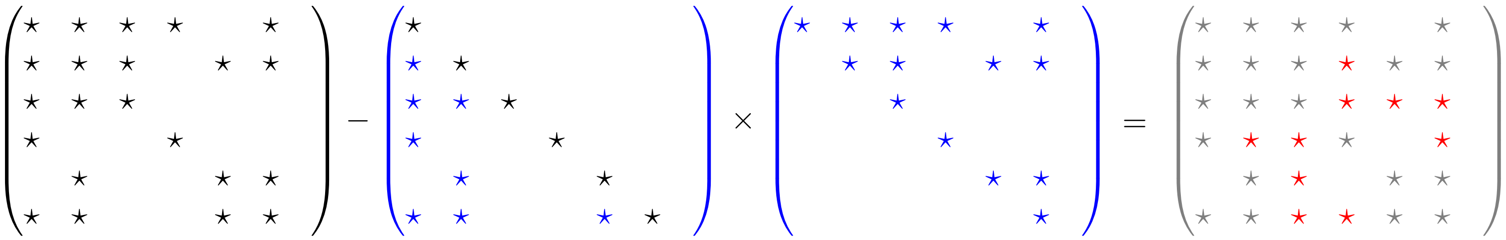
1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

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].



Fixed-point sweep approximates incomplete factors.

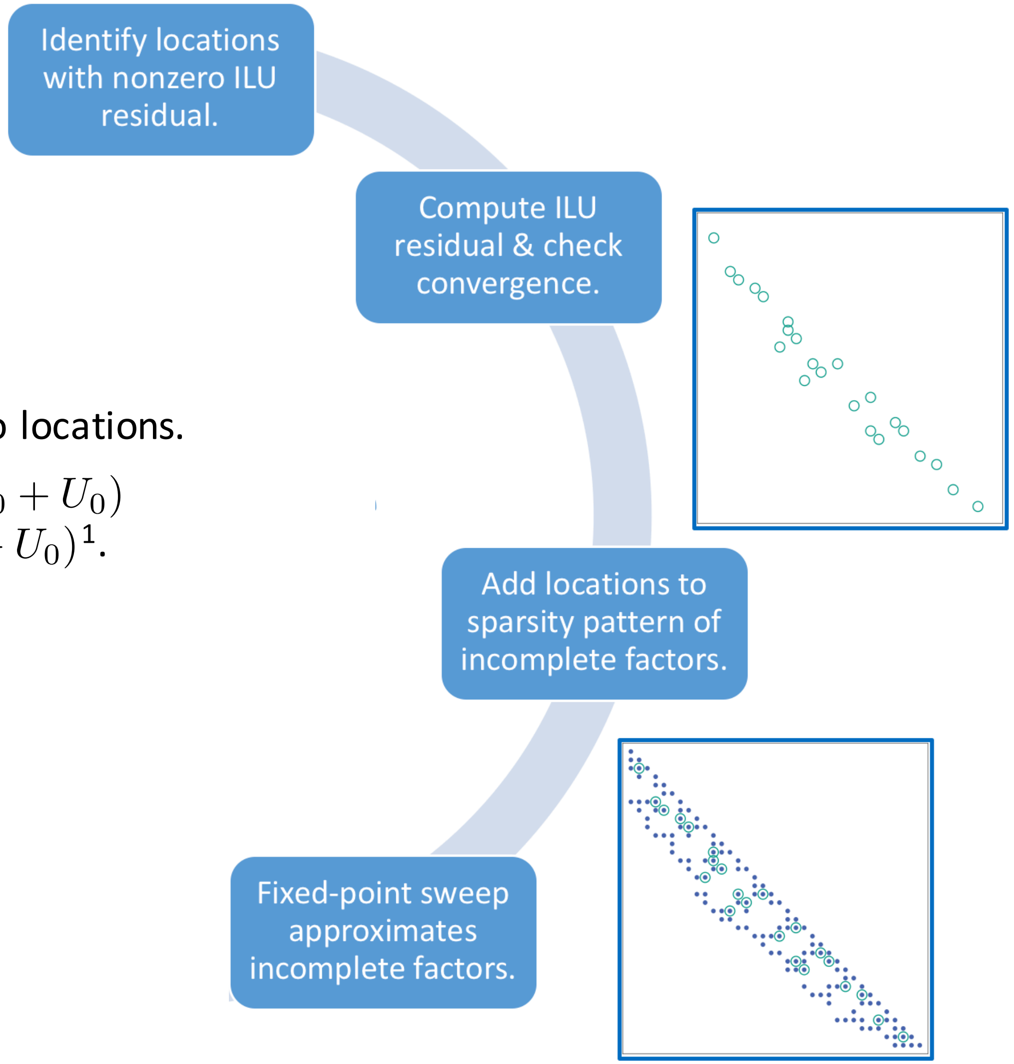
[1] Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition"**. *(2003)*.

# 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…

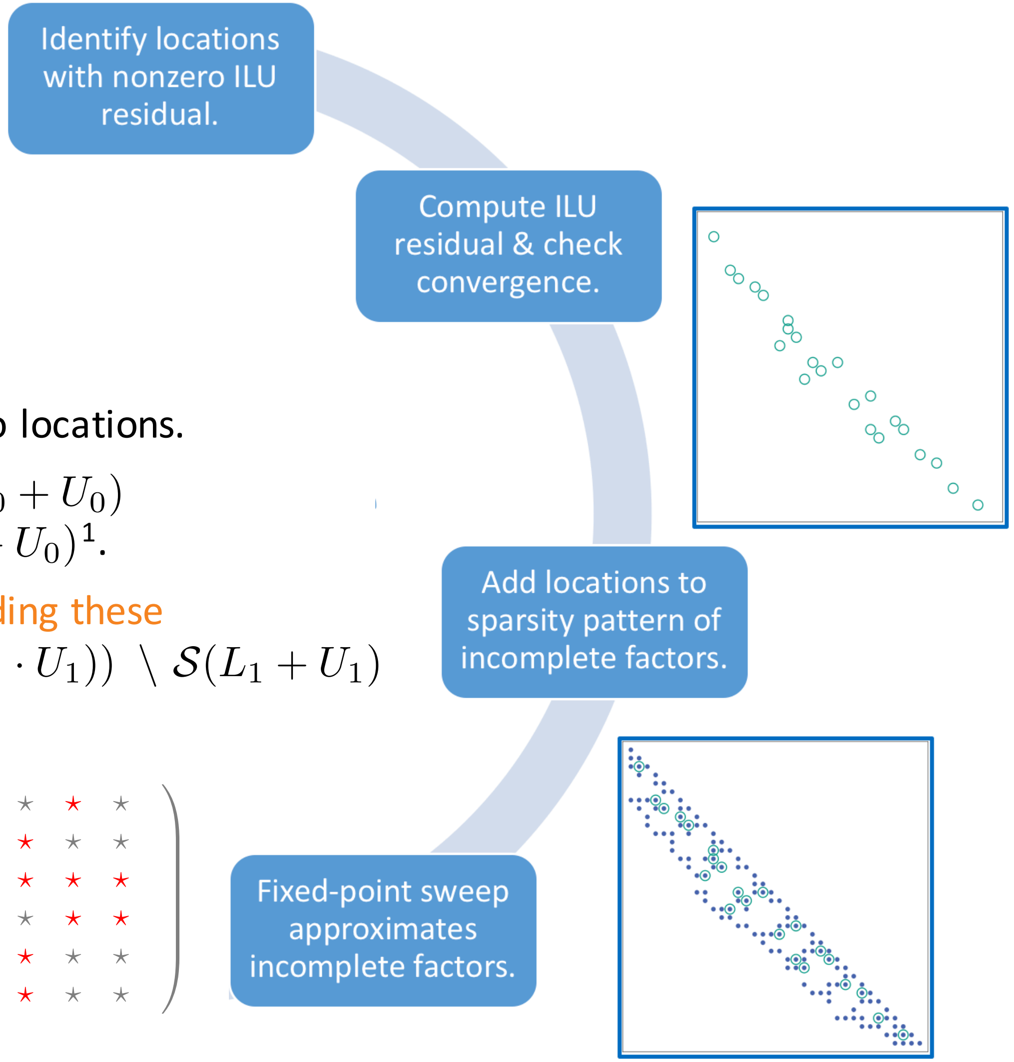
$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\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



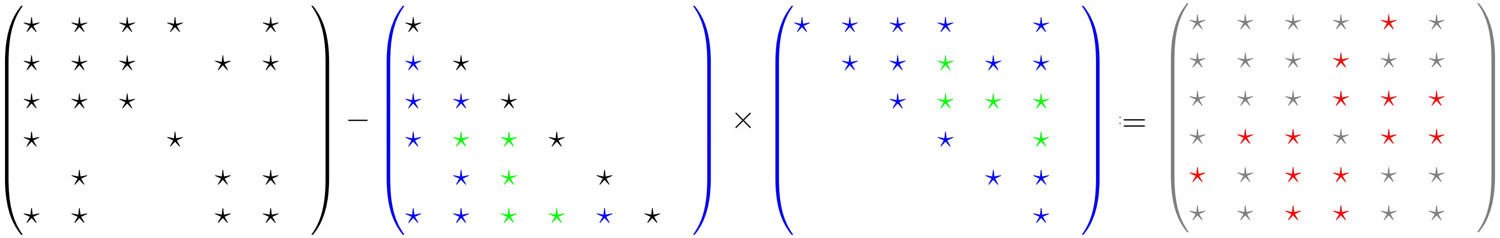

[1]Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition".** *(2003).*

# 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 $\mathcal{S}_2 = (\mathcal{S}(A) \cup \mathcal{S}(L_1 \cdot U_1)) \setminus \mathcal{S}(L_1 + U_1)$

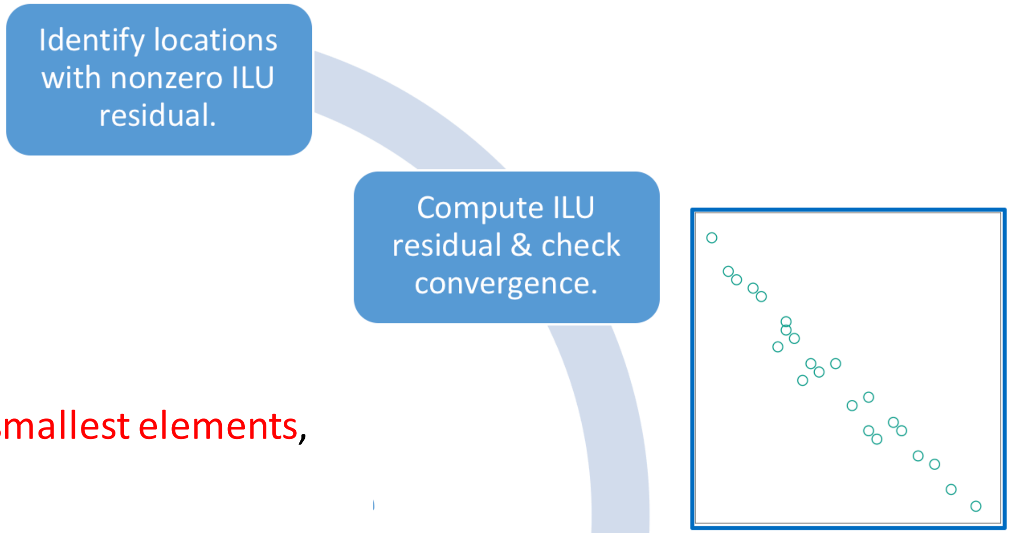


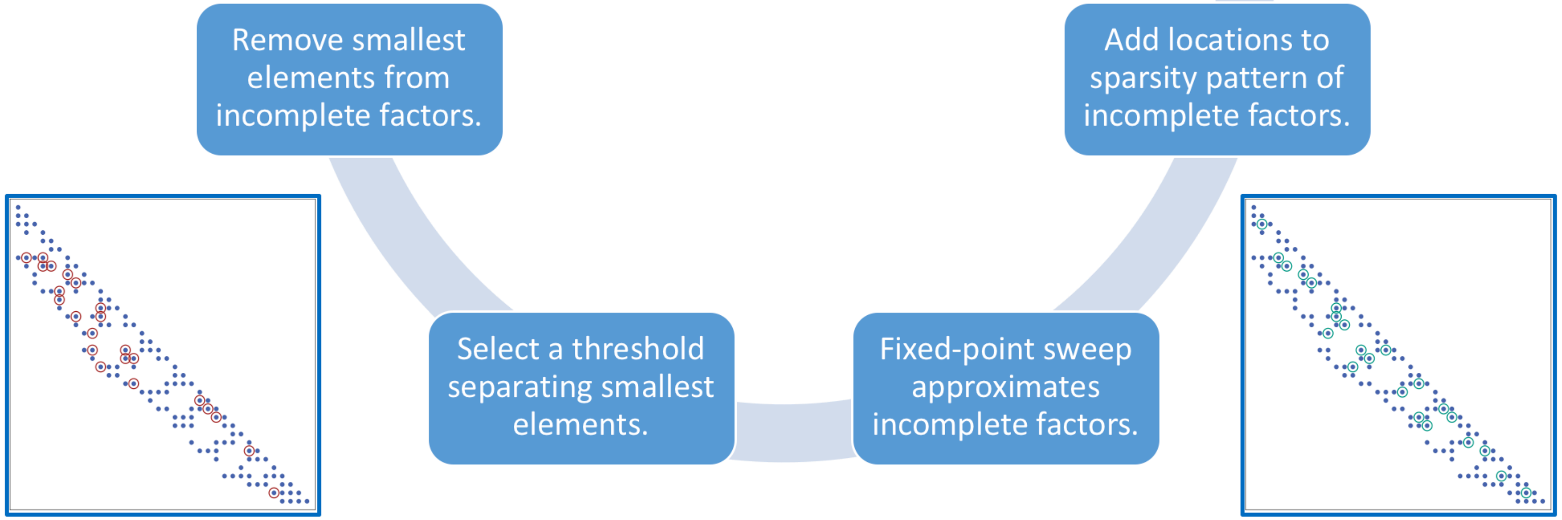
[1]Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition".** *(2003).*

# 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 $R = A - L_k \cdot U_k$ …

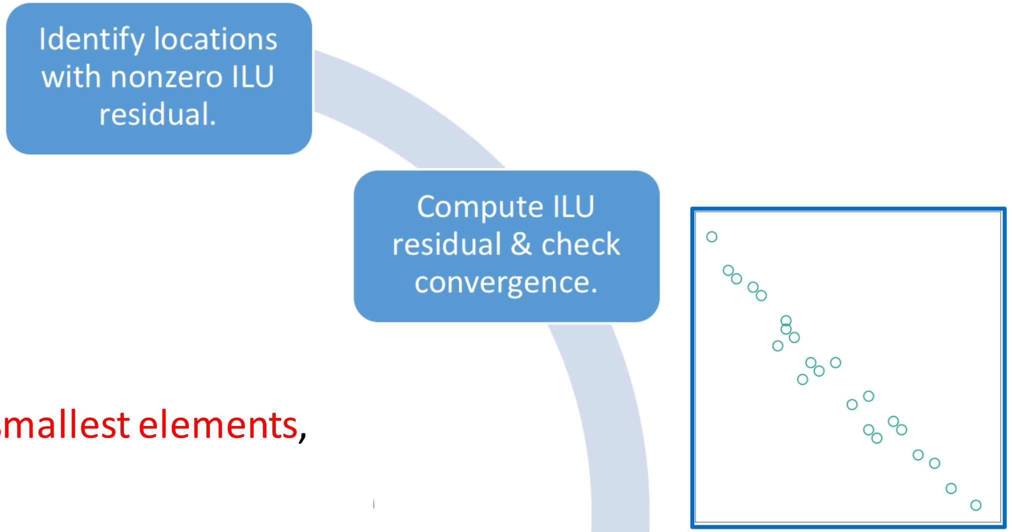


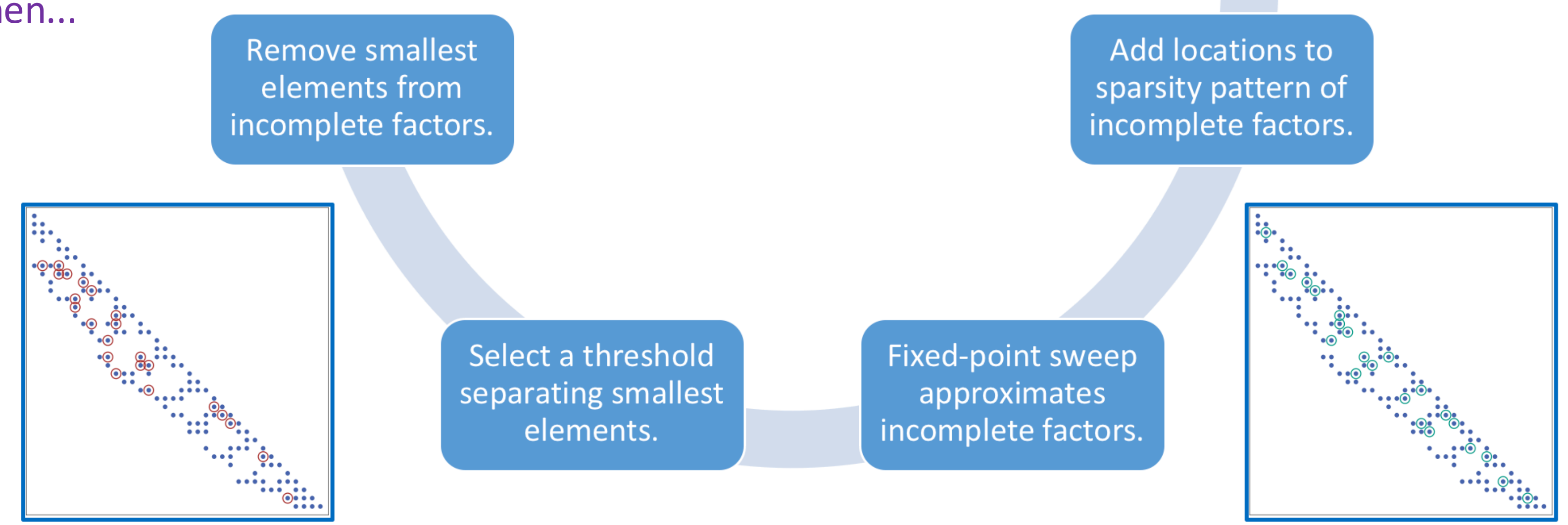



# Considerations

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.*

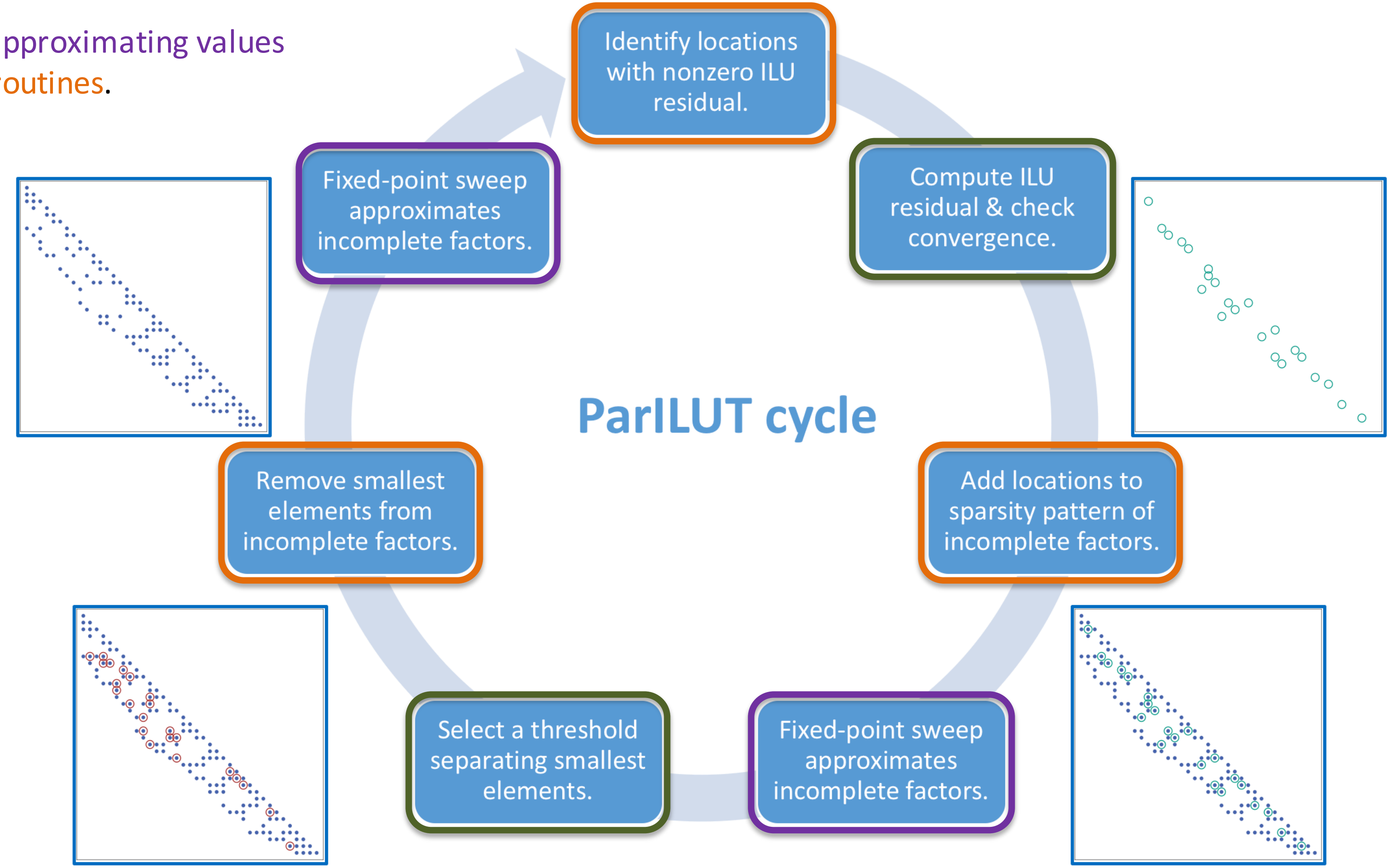
- At some point we should remove some locations again, e.g. the smallest elements, and start over looking at locations $R = A - L_k \cdot U_k$ ...
- We need another sweep, then...



# ParILUT

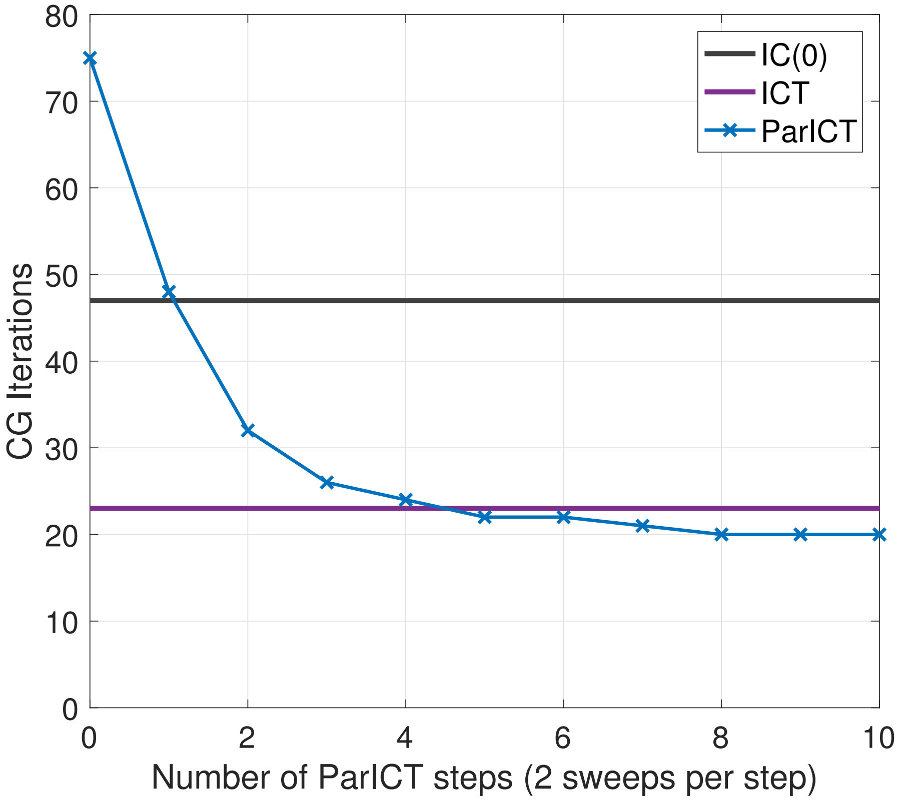
Interleaving fixed-point sweeps approximating values with pattern-changing symbolic routines.




[1]Anzt et al. **"ParILUT – A new parallel threshold ILU"**. *In: SIAM J. on Sci. Comp. (2018).*

# ParILUT quality

Anisotropic diffusion problem
n: 741, nz: 4,951

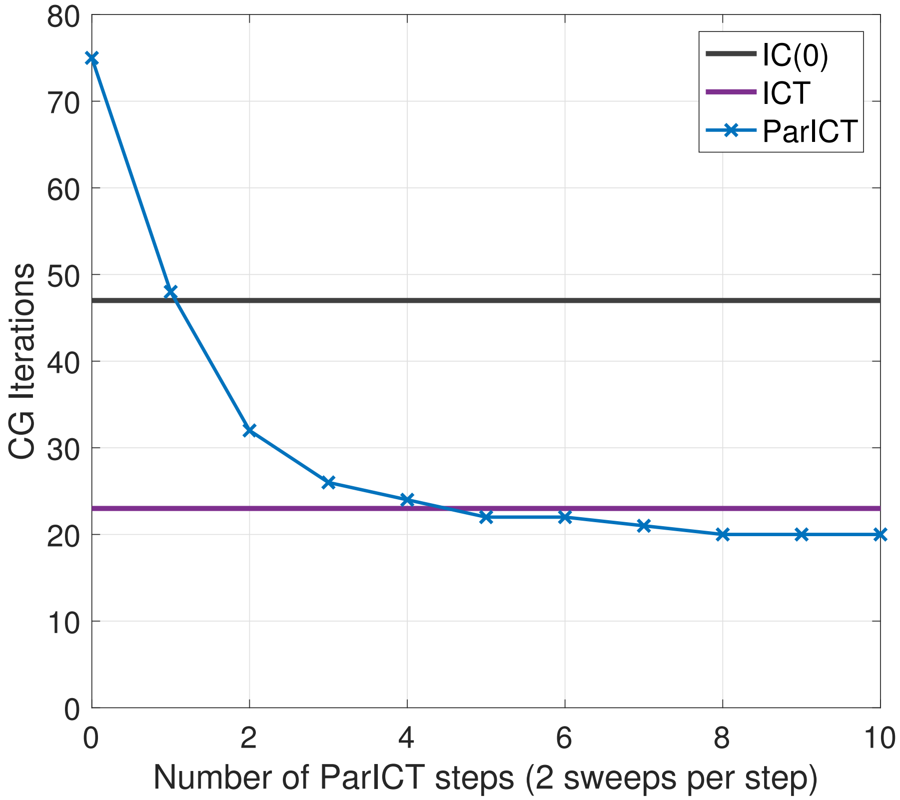


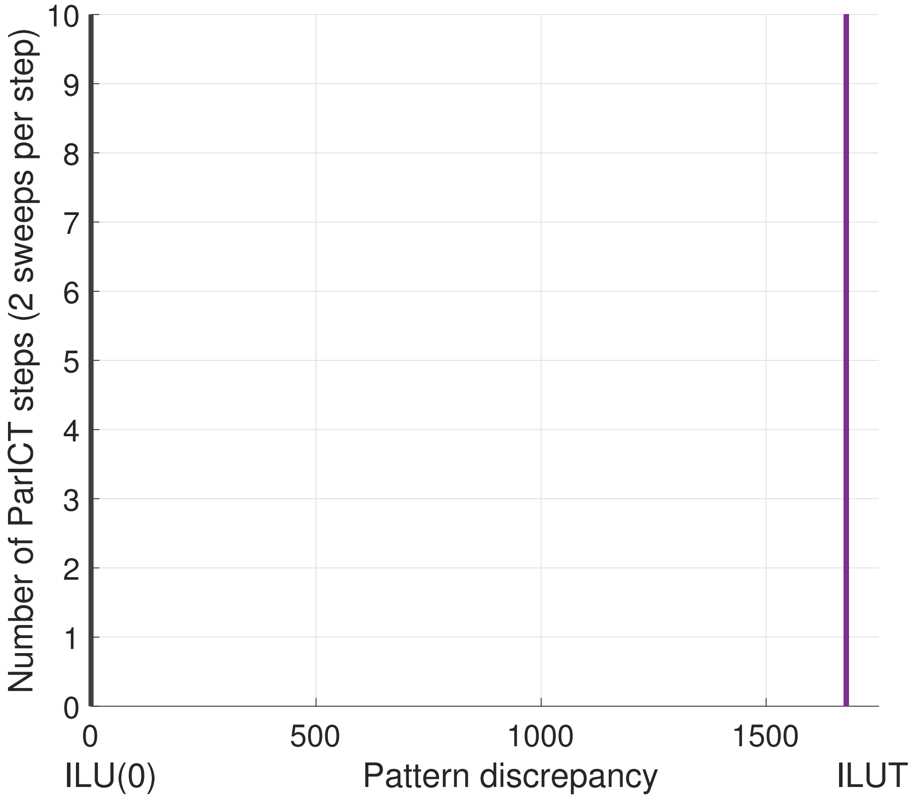

- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than conventional ILUT?

[1]Anzt et al. **“ParILUT – A new parallel threshold ILU”.** *In: SIAM J. on Sci. Comp. (2018).*

# ParILUT quality

Anisotropic diffusion problem
n: 741, nz: 4,951



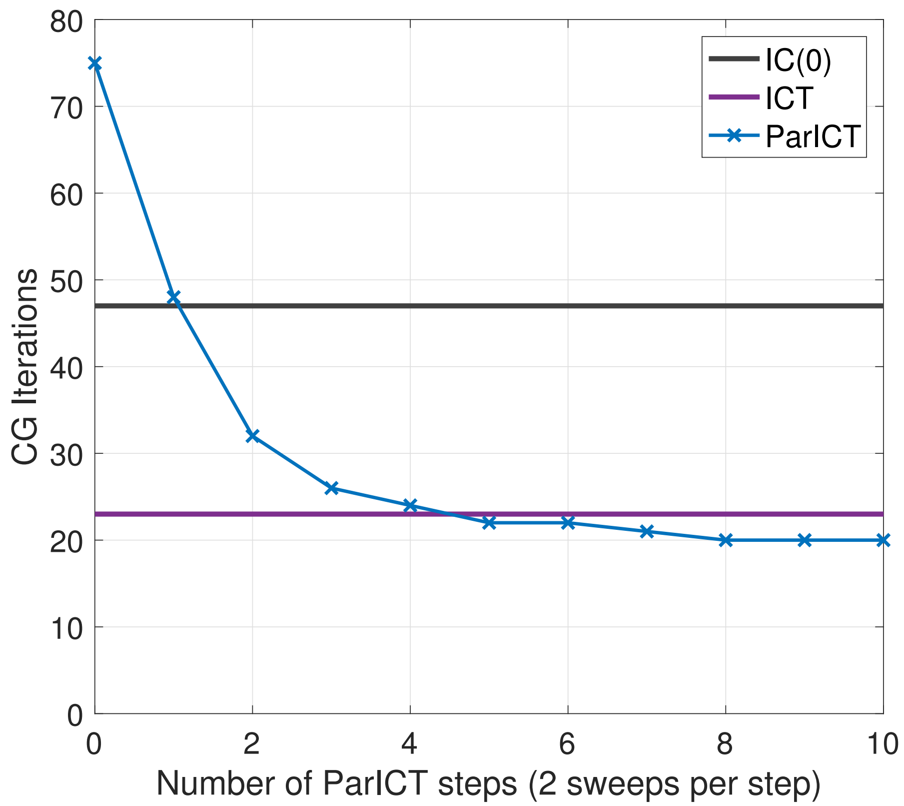



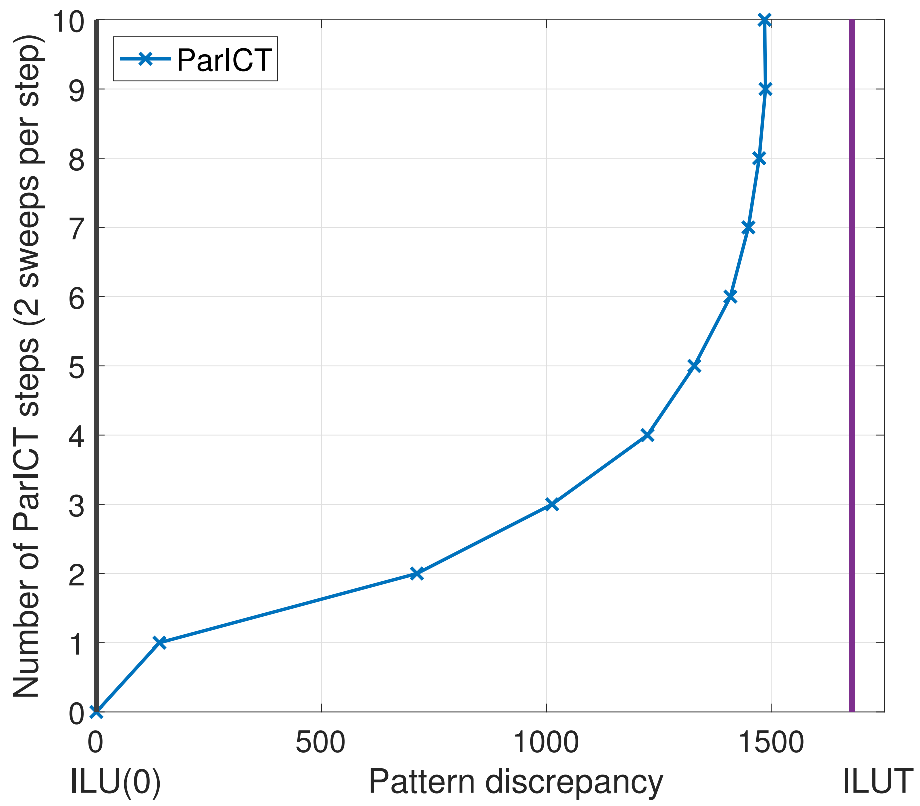

- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than conventional ILUT?

[1]Anzt et al. **“ParILUT – A new parallel threshold ILU”.** *In: SIAM J. on Sci. Comp. (2018).*

# ParILUT quality

Anisotropic diffusion problem
n: 741, nz: 4,951



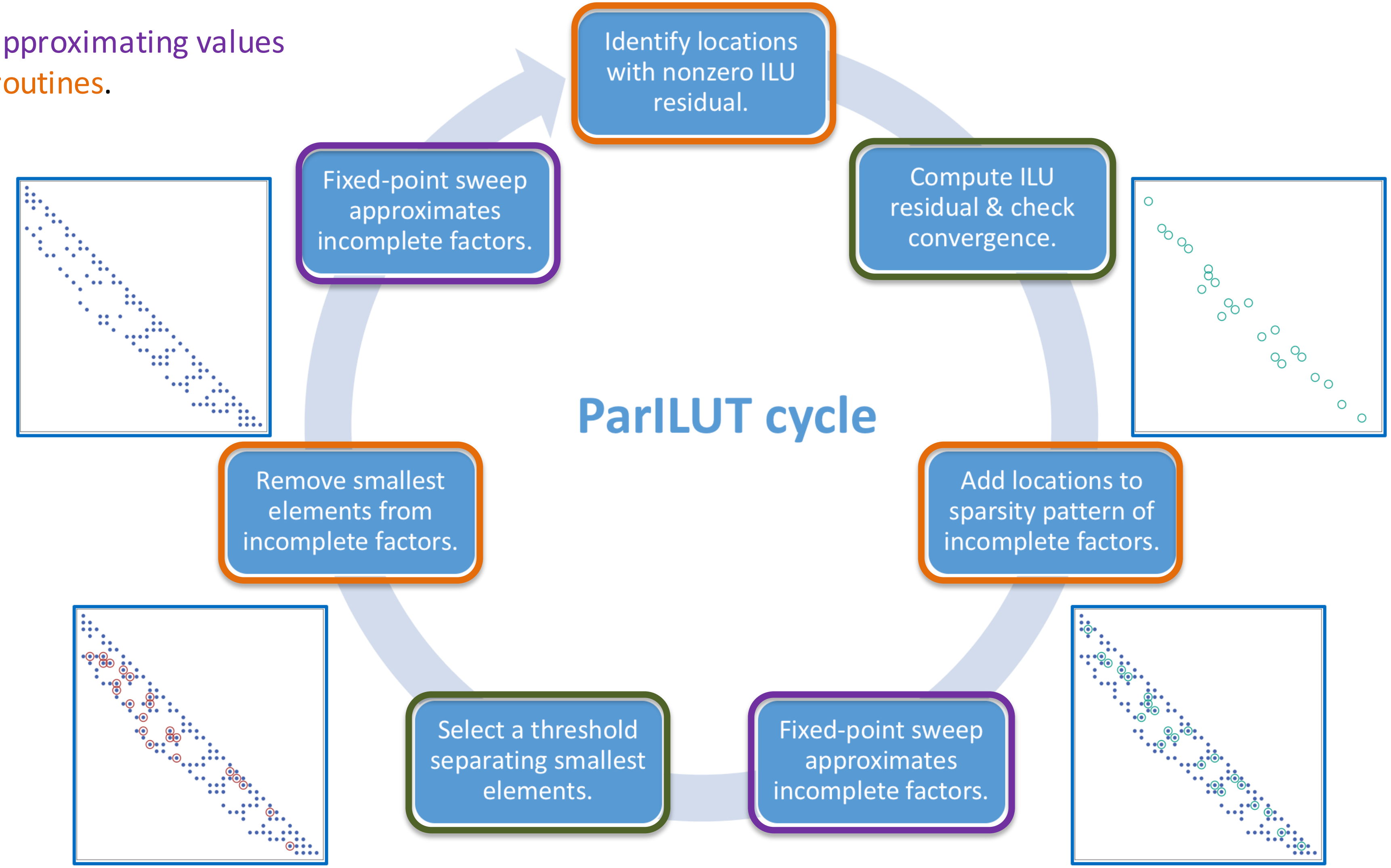

- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than ILUT?

- Pattern converges after few sweeps.
- Pattern “more like” ILUT than ILU(0).

[1]Anzt et al. **“ParILUT – A new parallel threshold ILU”.** *In: SIAM J. on Sci. Comp. (2018).*

# ParILUT – A Parallel Threshold ILU for GPUs

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



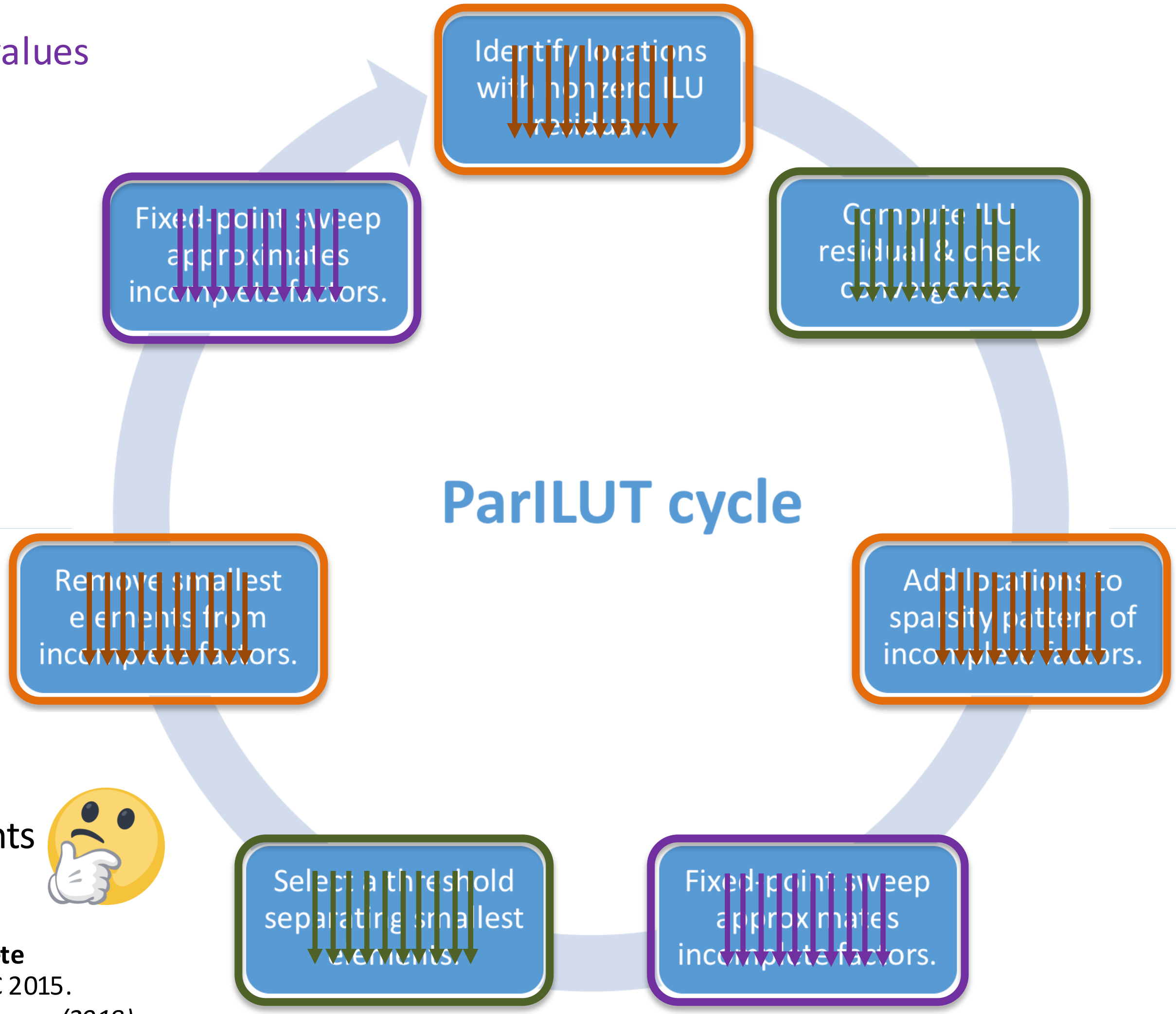

[1]Anzt et al. **"ParILUT – A new parallel threshold ILU".** *In: SIAM J. on Sci. Comp. (2018).*

# ParILUT – A Parallel Threshold ILU for GPUs

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

Parallelism inside the building blocks:

- Fixed-Point Sweeps[1] ✓
- Residuals[1] ✓
- Identify Fill-In Locations[2] ✓
- Add Locations[2] ✓
- Remove Locations[2] ✓
- Select Threshold Separating Smallest Elements




[1]Chow et al. **"Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs"**. In ISC 2015.

[2]Anzt et al. **"ParILUT – A new parallel threshold ILU"**. *In: SIAM J. on Sci. Comp. (2018).*

# Threshold Selection on GPUs

*This is equivalent to the Selection Problem!*

Given an unsorted sequence of real numbers $x_0, x_1, x_2, x_3, \dots x_{n-1}$, we want to find the element $x_{i_k}$ such that in the sorted sequence

$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \cdots \leq \underset{\uparrow \atop k}{x_{i_k}} \leq \dots x_{i_{n-1}}$$

the element $x_{i_k}$ is located in position $k$.

*We do not necessarily need to sort the complete sequence!*

## Approximate and Exact Selection on GPUs

Tobias Ribizel*, Hartwig Anzt*†
*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany
†Innovative Computing Lab (ICL), University of Tennessee, Knoxville, USA
*tobias.ribizel@student.kit.edu, hartwig.anzt@kit.edu*

http://bit.ly/SampleSelectGPU

# Threshold Selection on GPUs

*This is equivalent to the Selection Problem!*

Given an unsorted sequence of real numbers $x_0, x_1, x_2, x_3, \ldots x_{n-1}$, we want to find the element $x_{i_k}$ such that in the sorted sequence

$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \cdots \leq x_{i_k} \leq \ldots x_{i_{n-1}}$$

$k$

the element $x_{i_k}$ is located in position $k$.

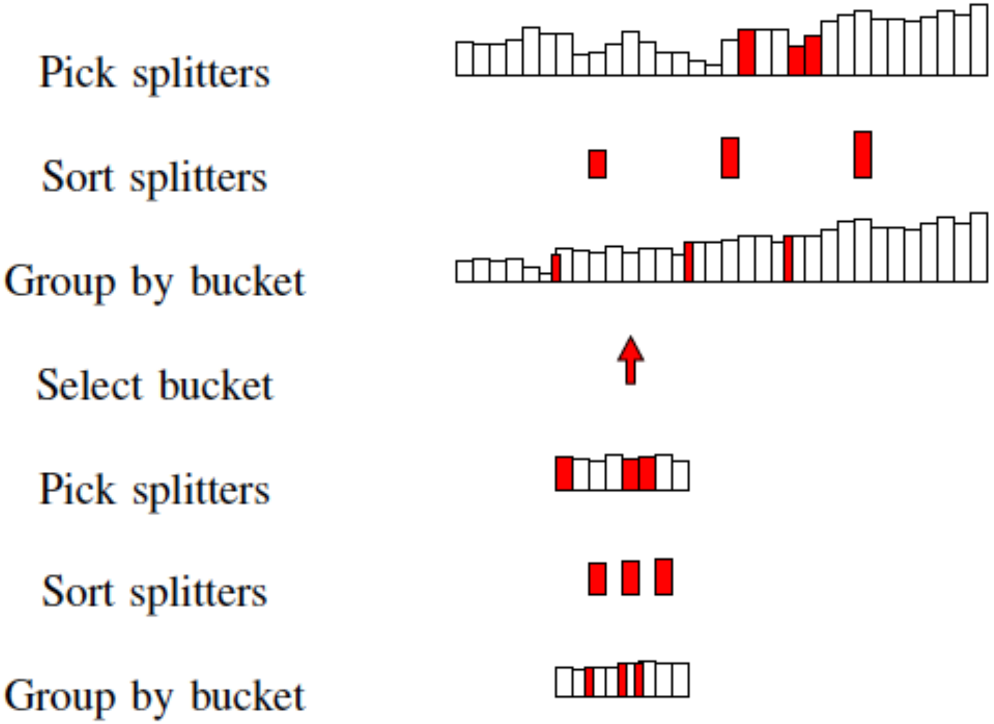
*We do not necessarily need to sort the complete sequence!*

Approximate and Exact Selection on GPUs

Tobias Ribizel*, Hartwig Anzt*†
*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany
†Innovative Computing Lab (ICL), University of Tennessee, Knoxville, USA
*tobias.ribizel@student.kit.edu, hartwig.anzt@kit.edu*

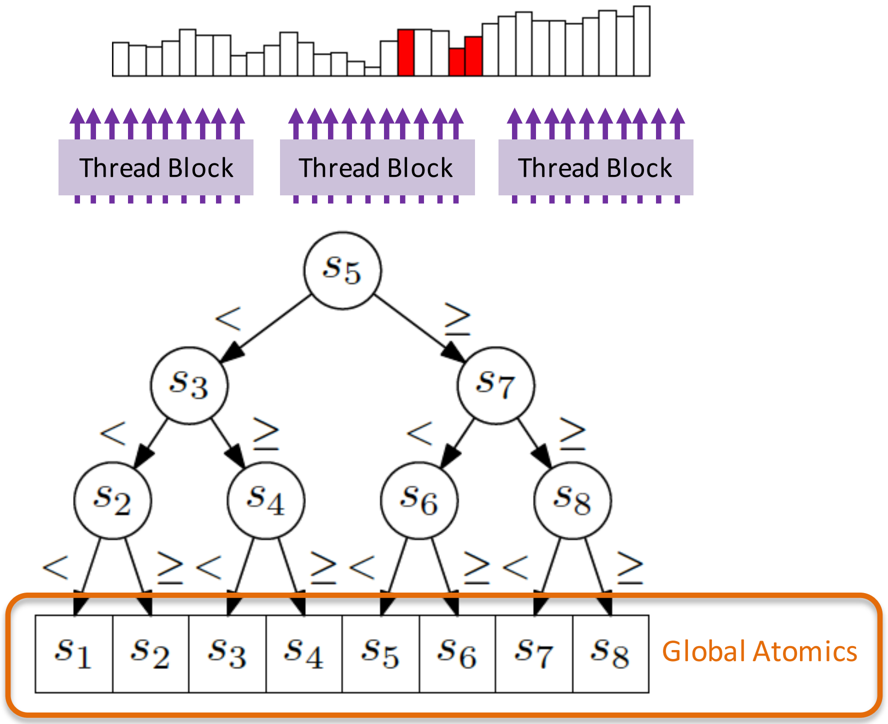
http://bit.ly/SampleSelectGPU

**SampleSelect Algorithm**



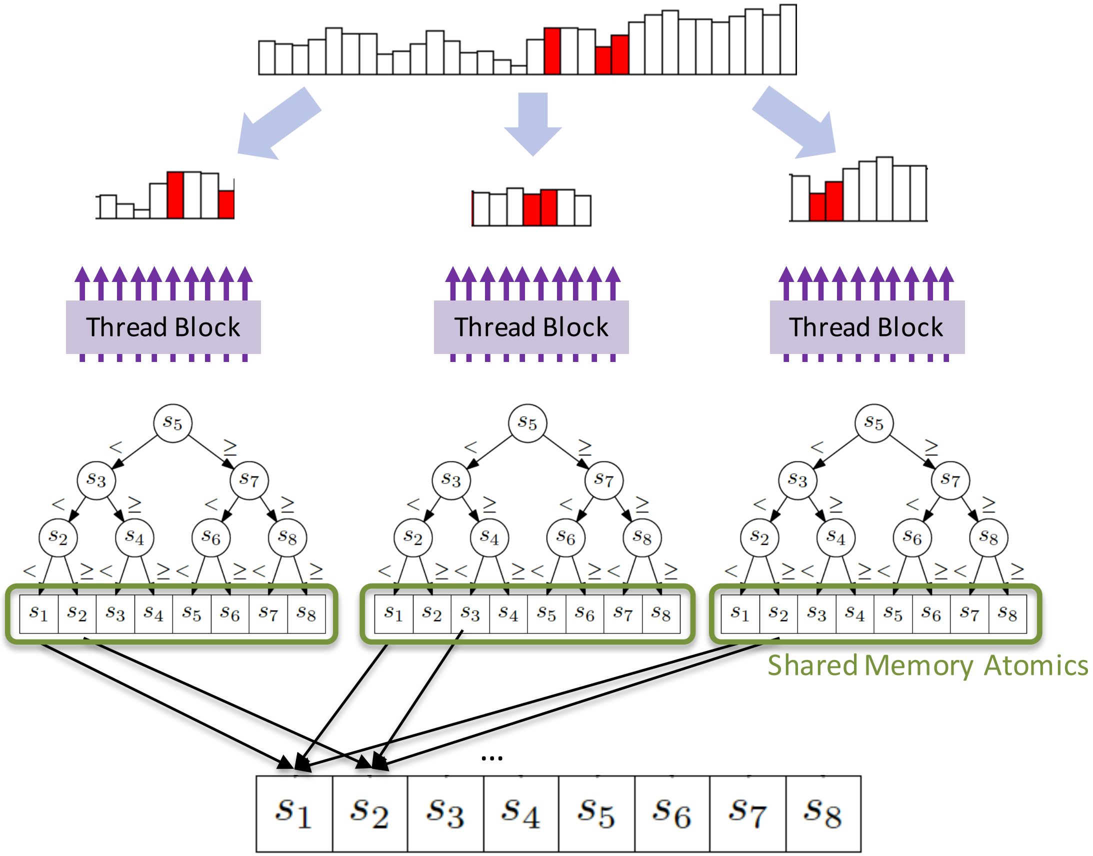
# Parallelization & Communication of SampleSelect on GPUs

## Global Memory Atomics




- Run SampleSelect using all resources on complete data set;
- Use global atomics to generate bucket counts;

## Shared Memory Atomics



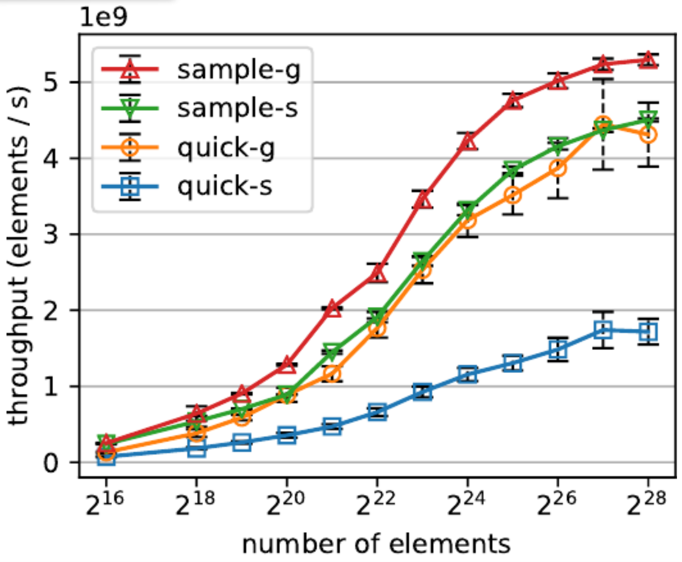

- Split data set into chunks, assign to thread blocks;
- Each thread block runs bucket count on its data;
- Use a global reduction to get global bucket counts;

# Global vs. Local Memory Atomics

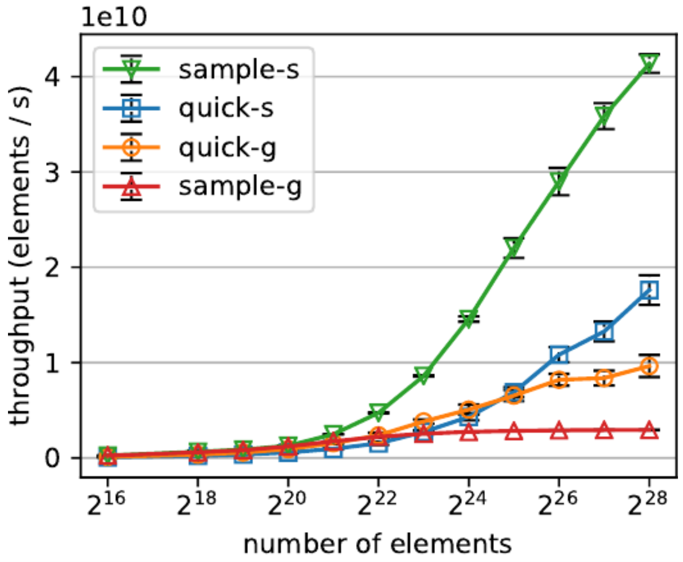
-g : global memory atomics
-s: shared memory atomics

*single precision*

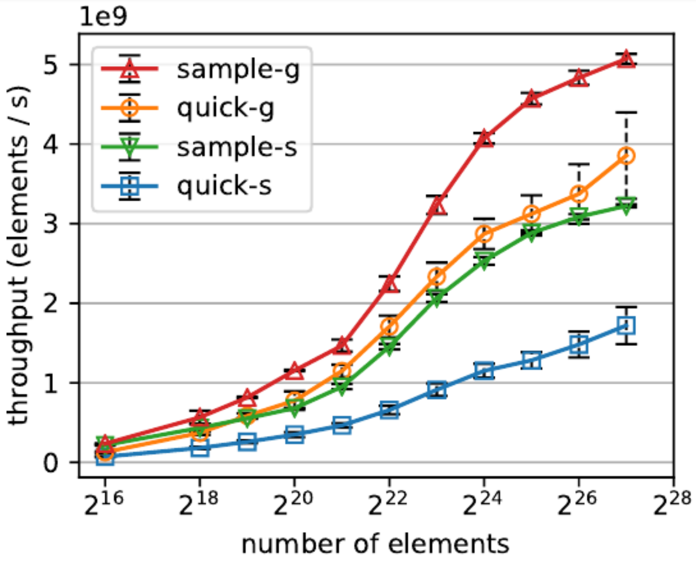
*NVIDIA K40*

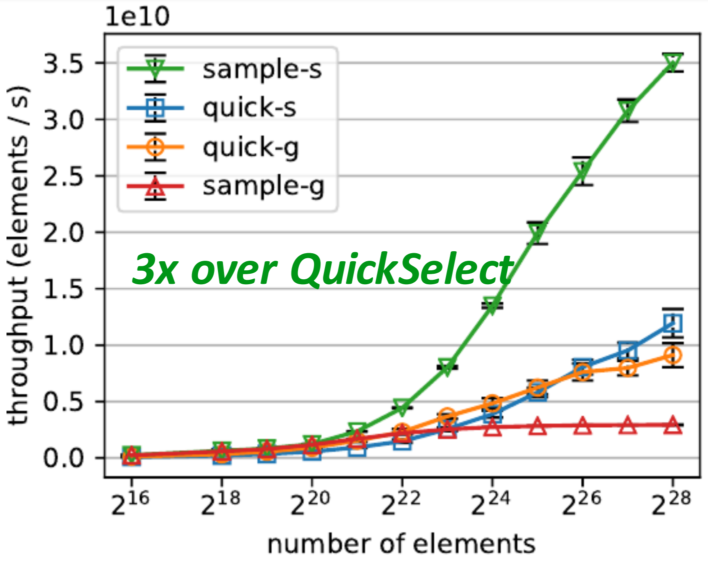



*NVIDIA V100*



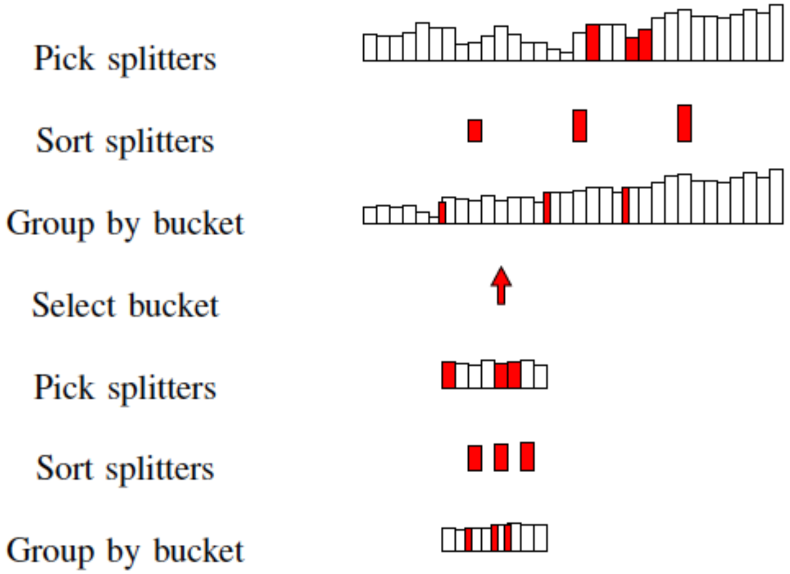

*double precision*



# Approximate and Exact Selection on GPUs[1]

## SampleSelect Algorithm




# Approximate and Exact Selection on GPUs

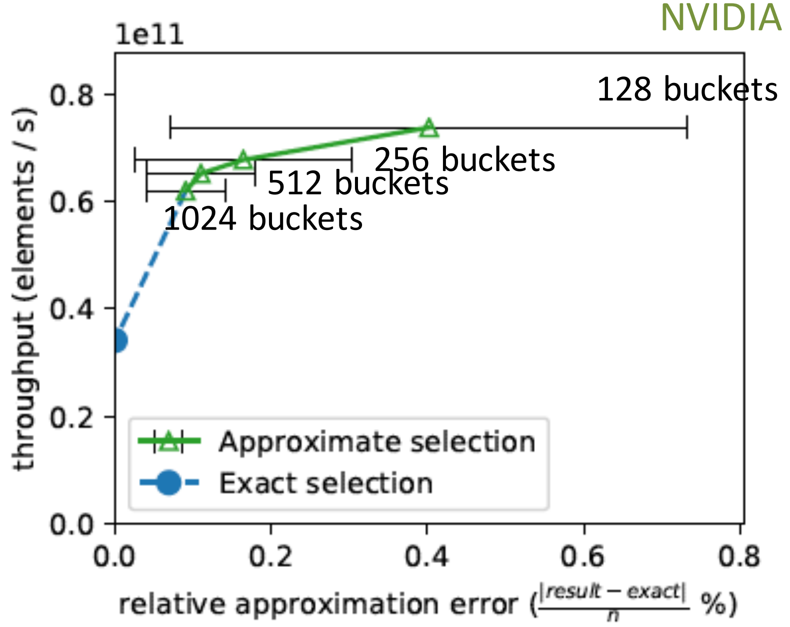

Tobias Ribizel*, Hartwig Anzt*†
*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany
†Innovative Computing Lab (ICL), University of Tennessee, Knoxville, USA
*tobias.ribizel@student.kit.edu, hartwig.anzt@kit.edu*


http://bit.ly/SampleSelectGPU

We do not descent to the lowest level of the recursion tree if we accept an approximate threshold.

- Accuracy depends on the ratio splitters vs. dataset size;
- Independent of value distribution (works on ranks, only);

NVIDIA V100 GPU



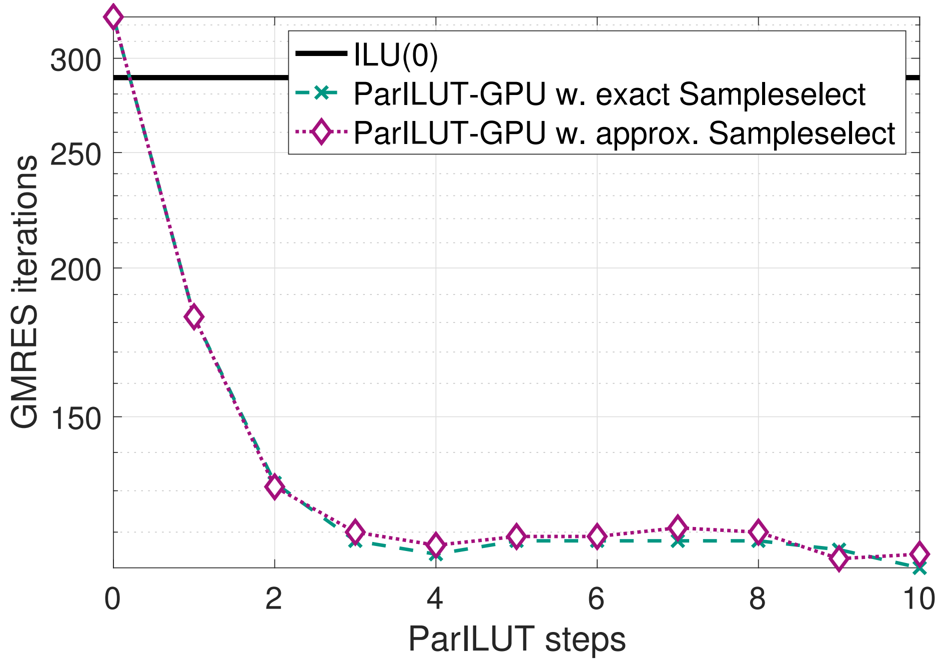

Approximate selection on $2^{28}$ uniformly distributed single precision values using 1 recursion level, only.

[1] Ribizel and Anzt. **"Approximate and Exact Selection on GPUs"**. *In: ASHES workshop 2019.*

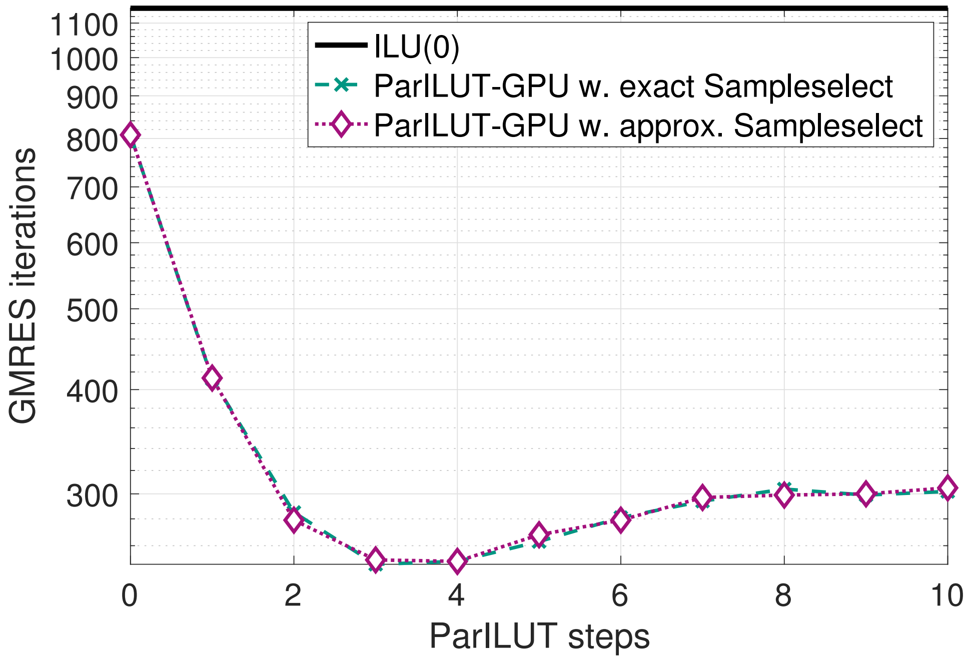
# ParILUT - A Parallel Threshold ILU for GPUs

*Impact of exact/approximate SampleSelect on ParILUT preconditioner quality*

ANI5



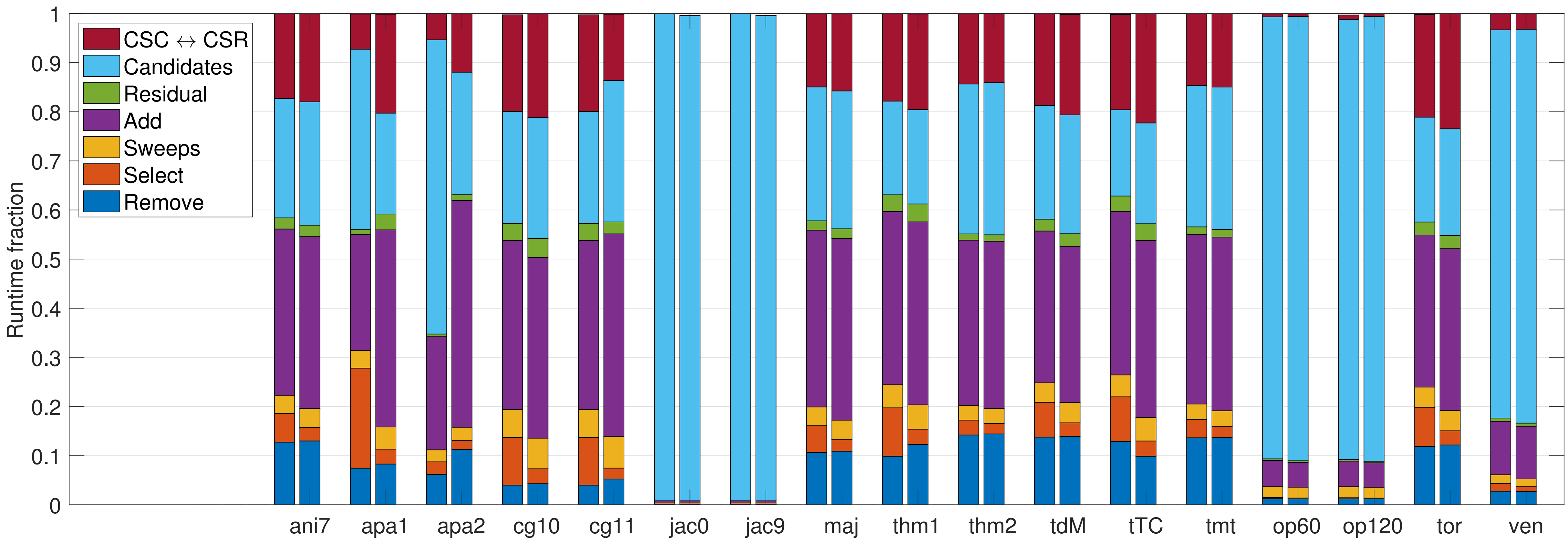

ANI6



# ParILUT - A Parallel Threshold ILU for GPUs

*Impact of exact/approximate SampleSelect on ParILUT runtime breakdown*

NVIDIA V100 GPU

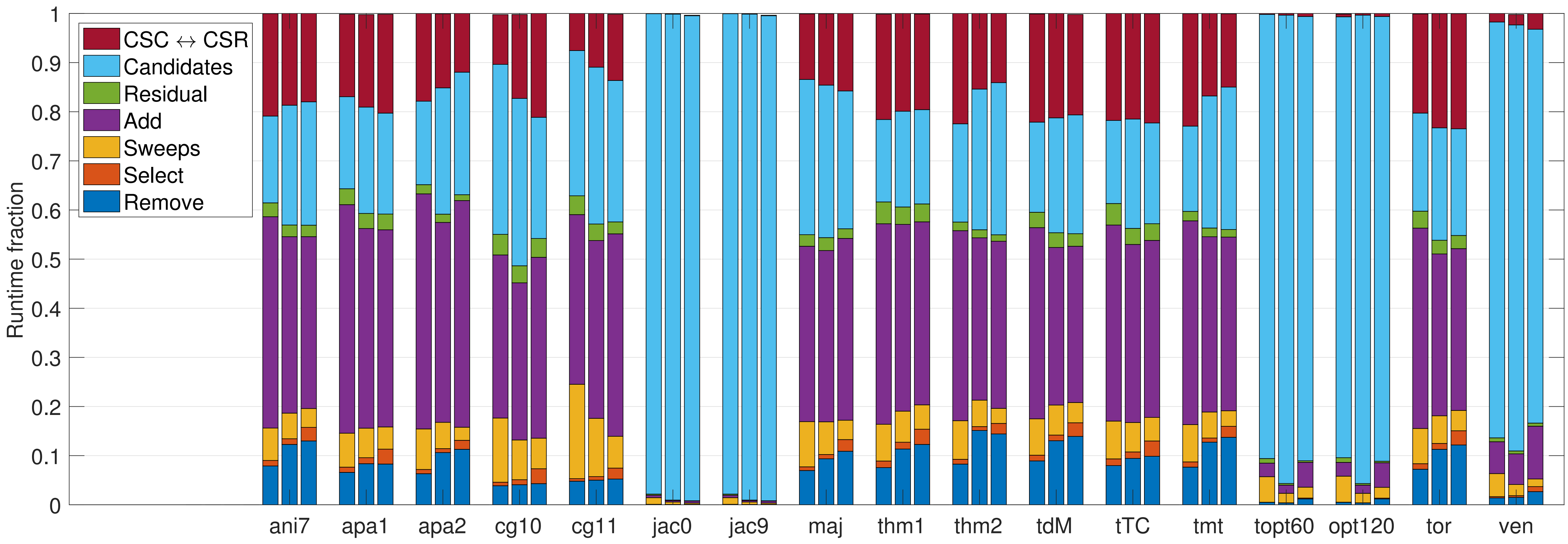



Matrices taken from Suite Sparse Matrix Collection.

# ParILUT - A Parallel Threshold ILU for GPUs

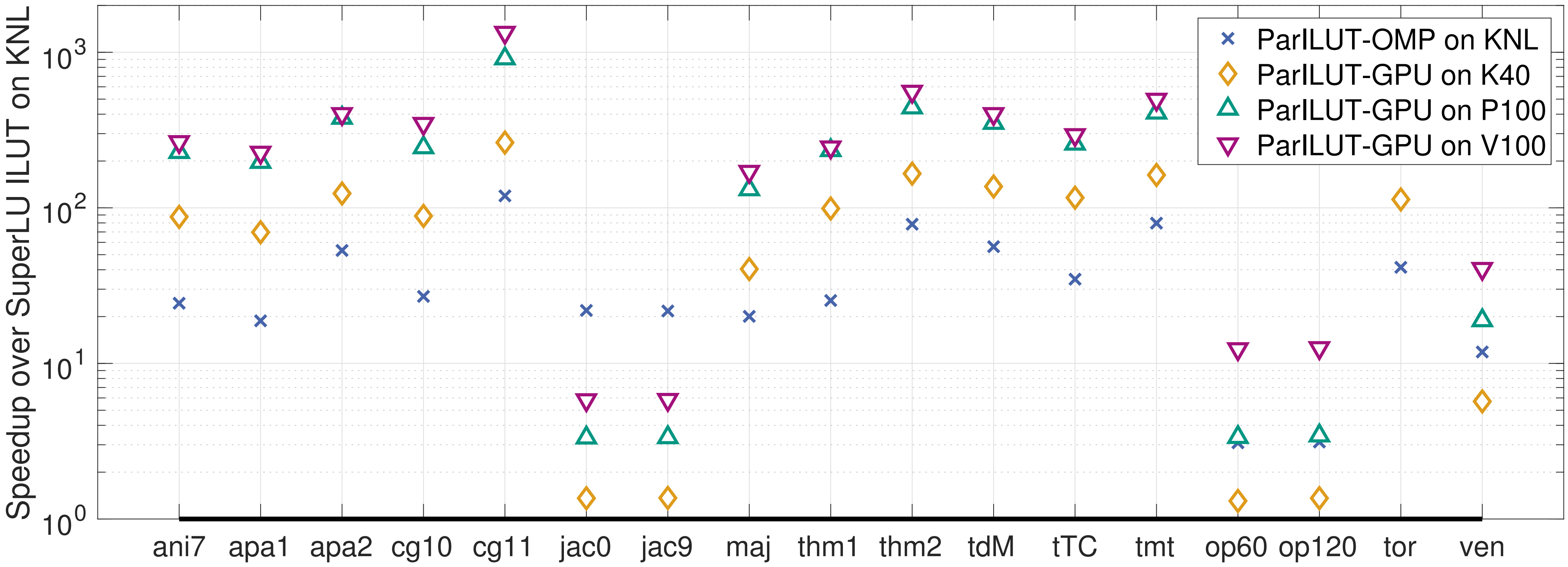
*ParILUT performance across different GPU generations:*

1st bar: NVIDIA K40
2nd bar: NVIDIA P100
3rd bar: NVIDIA V100




Matrices taken from Suite Sparse Matrix Collection.

# ParILUT Performance across architectures




Matrices taken from Suite Sparse Matrix Collection.

# Next Steps …

- **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.

http://bit.ly/ParILUTGPU

*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 | ParILUT 1 | ParILUT 2 | ParILUT 3 | ParILUT 4 | ParILUT 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 | 1 | 2 | 3 | 4 | 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 | – | – |