Karlsruhe Institute of Technology

Steinbuch Centre for Computing

ParILUT – A New Parallel Threshold ILU

05/07/2018
Kolloquiumsvortrag in der Fakultät für Informatik

Hartwig Anzt
Steinbuch Centre for Computing (SCC)

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?*

$$\mathcal{S}(A) = \{(i,j) \in \mathbb{N}^2 : A_{ij} \neq 0\}$$

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 $Ax = b$:

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

- De-Facto standard for solving dense problems.

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 $Ax = b$:

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

- 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 $Ax = b$:

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

- 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 $\mathcal{S}$.

  $L \in \mathbb{R}^{n \times n}$ lower (unit-) triangular, sparse.

  $U \in \mathbb{R}^{n \times n}$ upper triangular, sparse.

  $L_{ij} = U_{ij} = 0 \; \forall (i,j) \notin \mathcal{S}.$

  $R = L - U, \; R_{ij} = 0 \; \forall (i,j) \in \mathcal{S}.$

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 $Ax = b$:

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

- 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 $Ax = b$:

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

- 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!

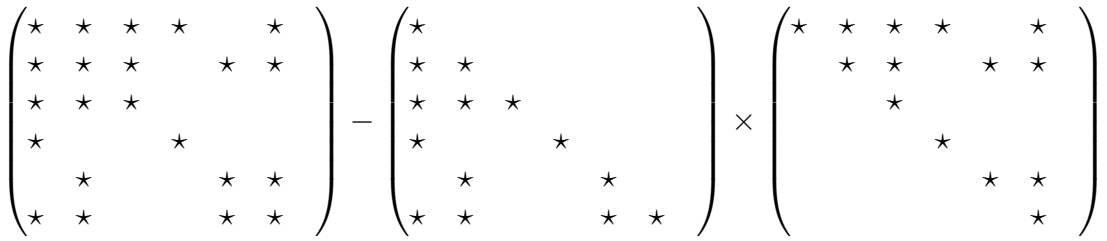
- Use a parallel iterative process to generate factors.
- The preconditioner should have a moderate number of nonzero elements,
  but we don't care too much about intermediate data.

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…

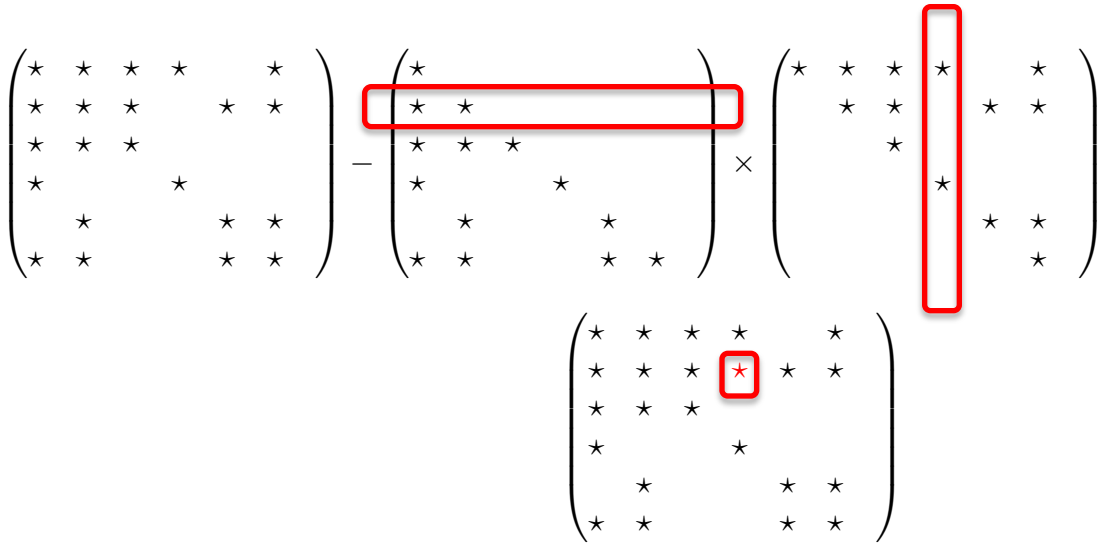


ILU residual $R = A - L \times U$

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.*

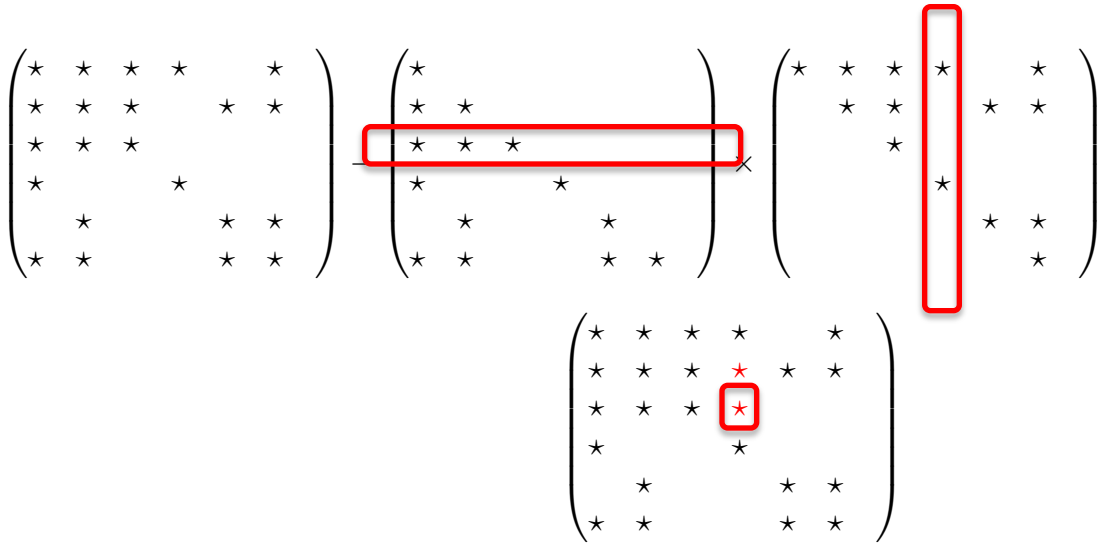
- This is an optimization problem…



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.*

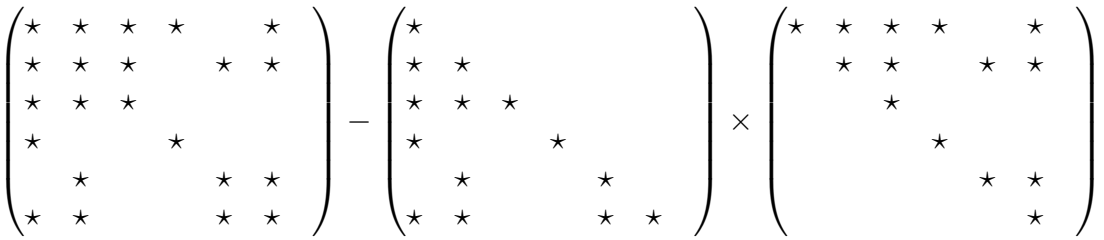
- This is an optimization problem…



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…

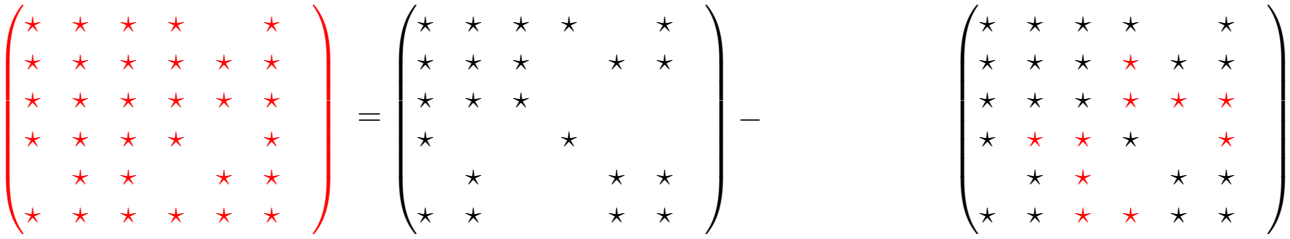


Considerations

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…

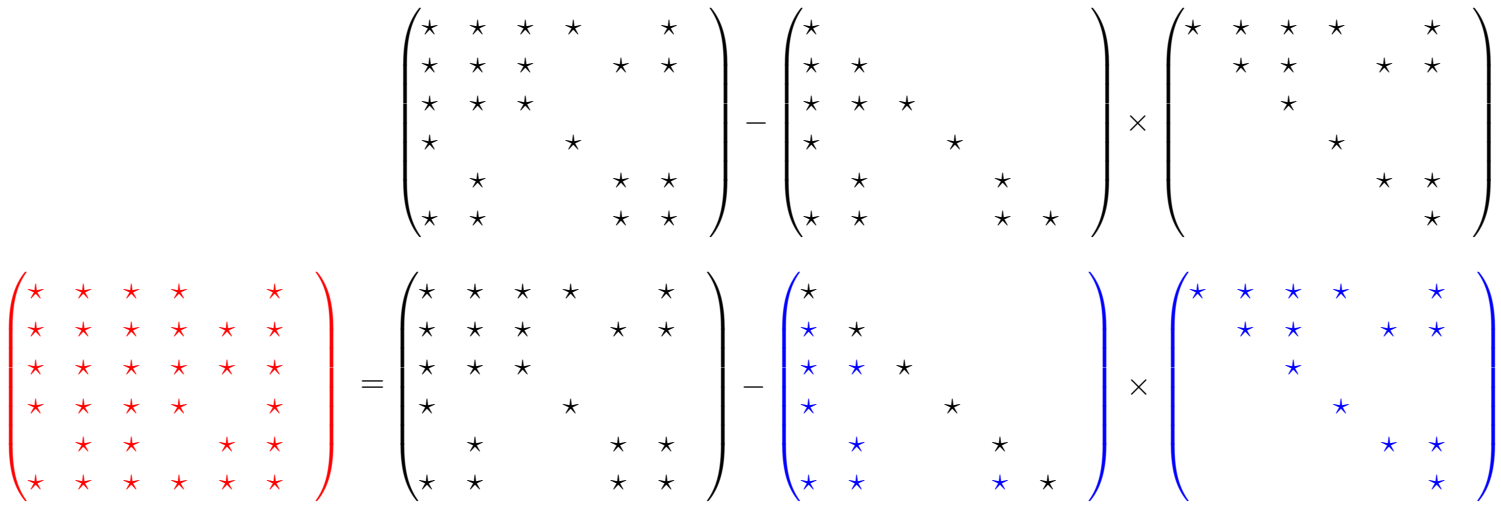
Residual:



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.



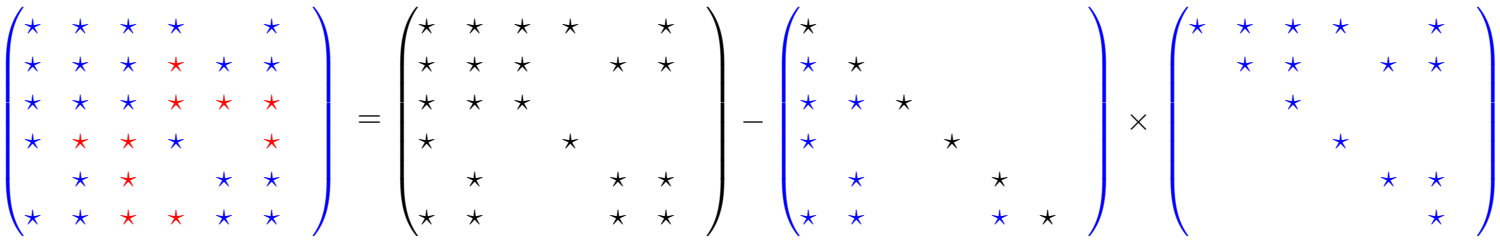
Sparsity pattern $\mathcal{S}$

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



Sparsity pattern $\mathcal{S}$

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]:

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

[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]:

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

- 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]:

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

Fixed-point sweep approximates incomplete factors.

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

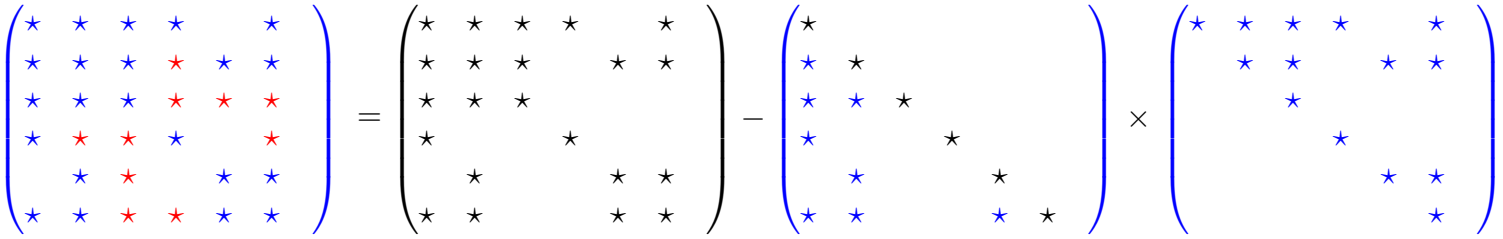
[1] Chow and Patel. **"Fine-grained Parallel Incomplete LU Factorization".** *In: SIAM J. on Sci. Comp. (2015).*

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.***

Compute ILU residual & check convergence.

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



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.

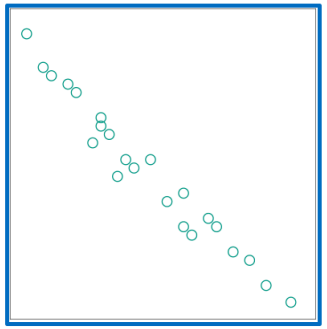
Fixed-point sweep approximates incomplete factors.

Considerations

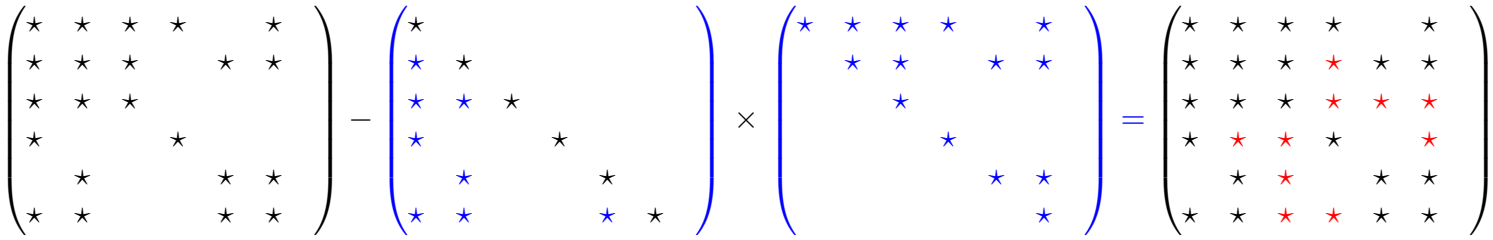
1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

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].



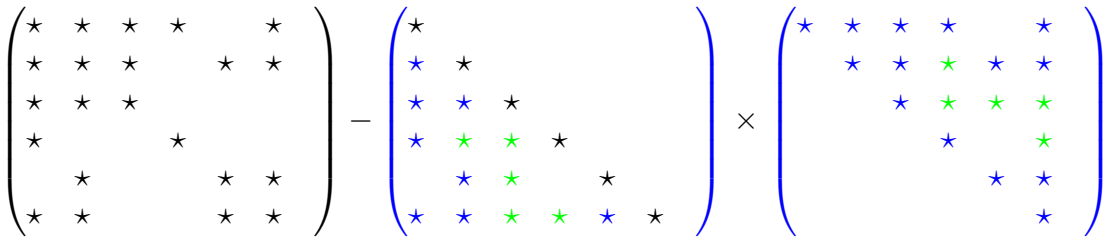
Fixed-point sweep approximates incomplete factors.

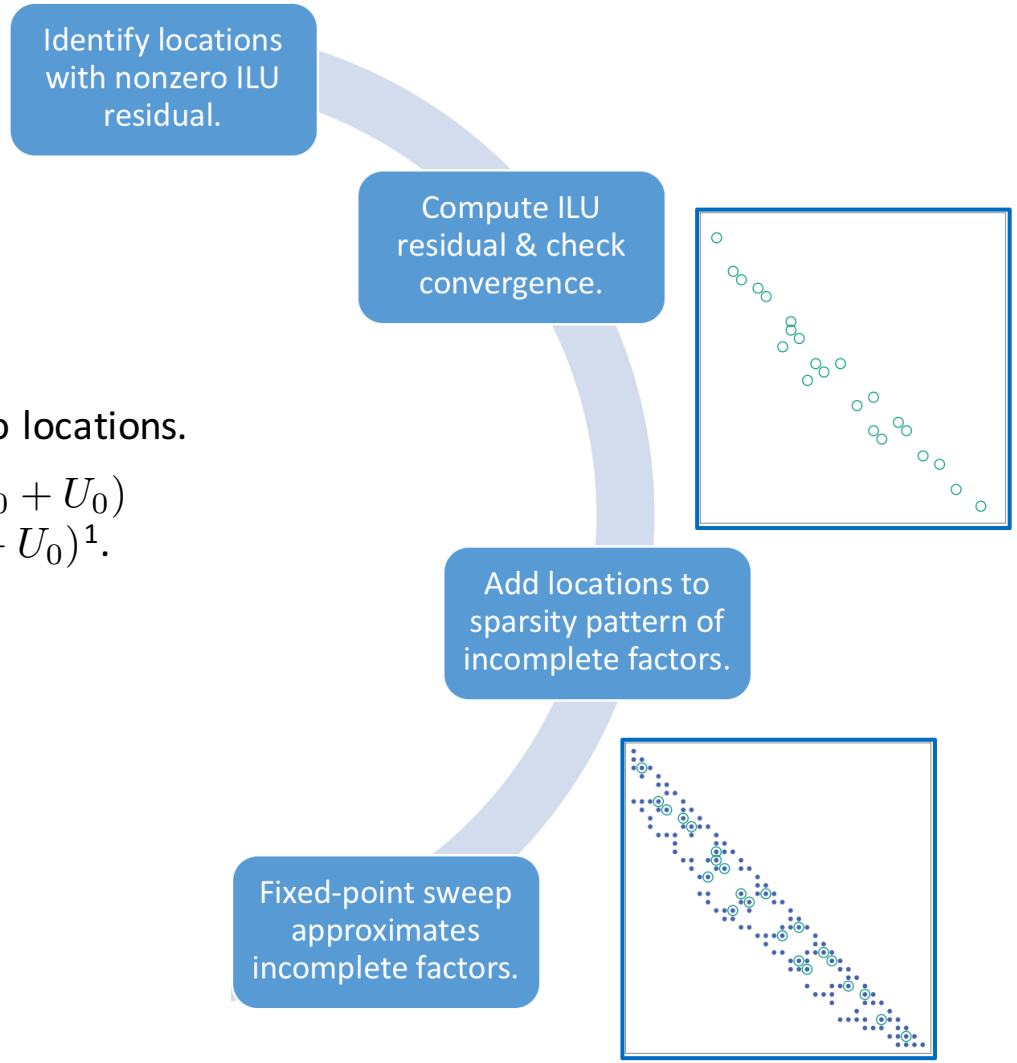
[1]Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition".** *(2003).*

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…

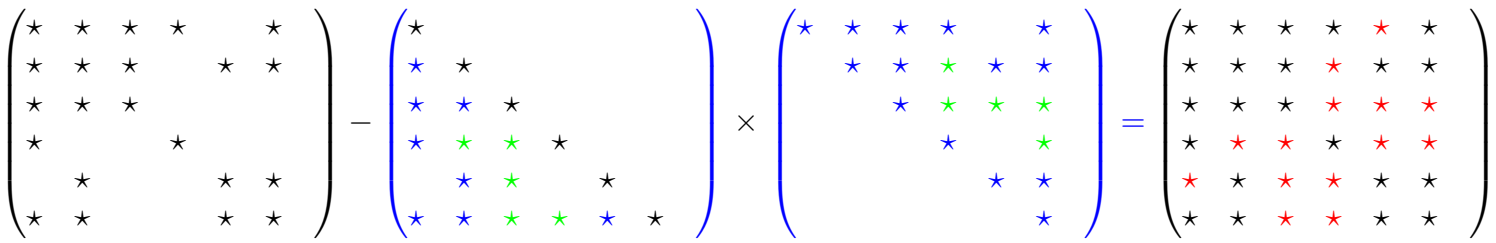




Considerations

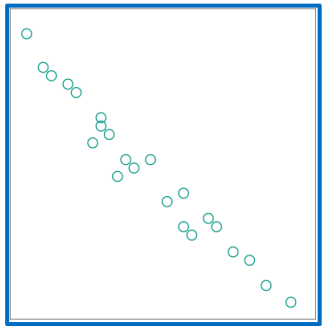
1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that* $A \approx L \cdot U$ *is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

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



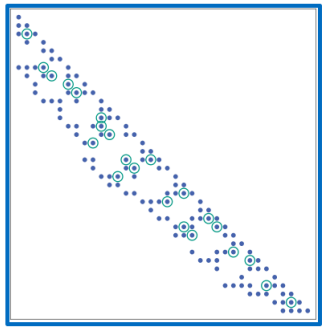
Identify locations with nonzero ILU residual.

Compute ILU residual & check convergence.



Add locations to sparsity pattern of incomplete factors.

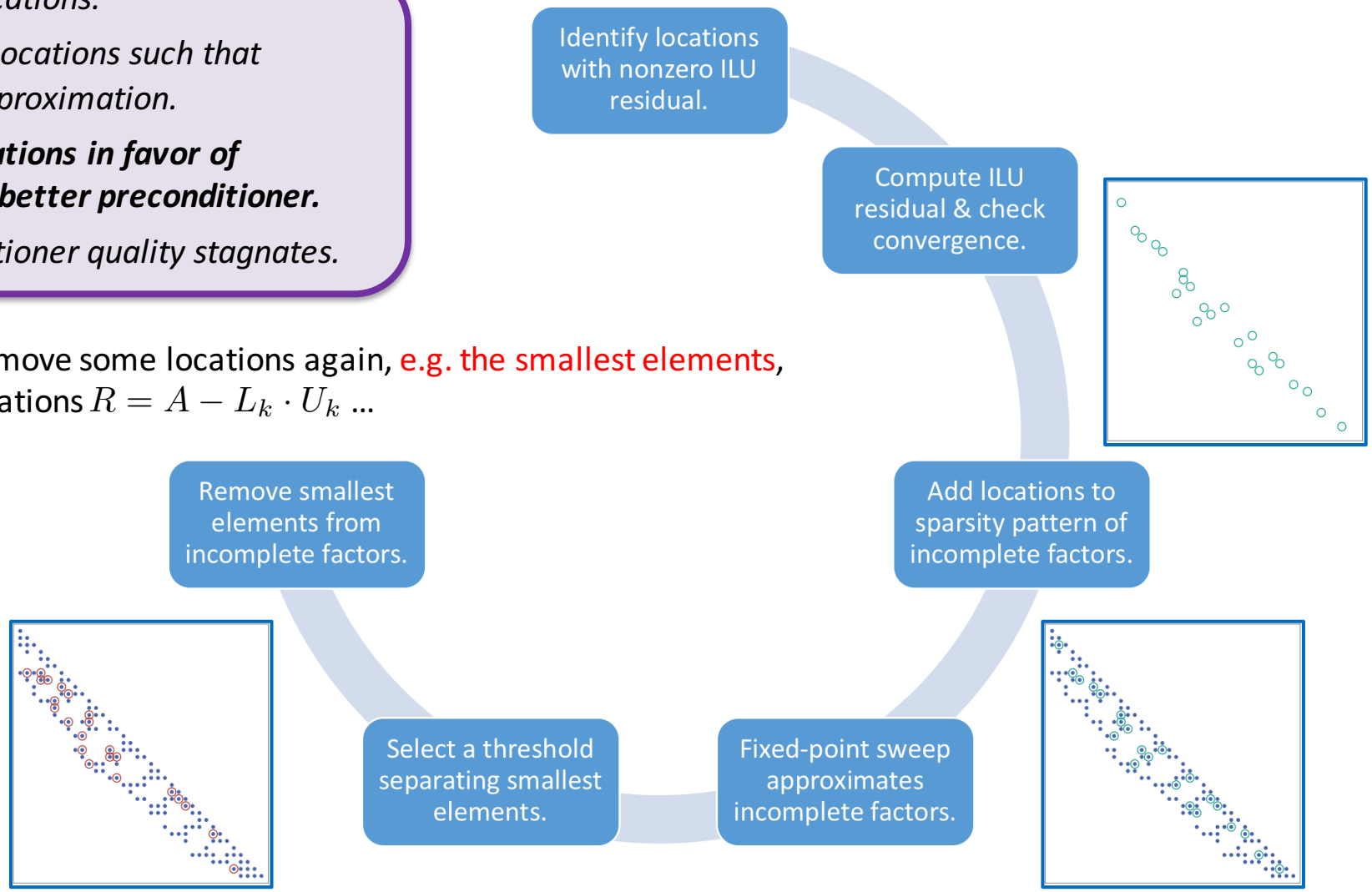
Fixed-point sweep approximates incomplete factors.



Considerations

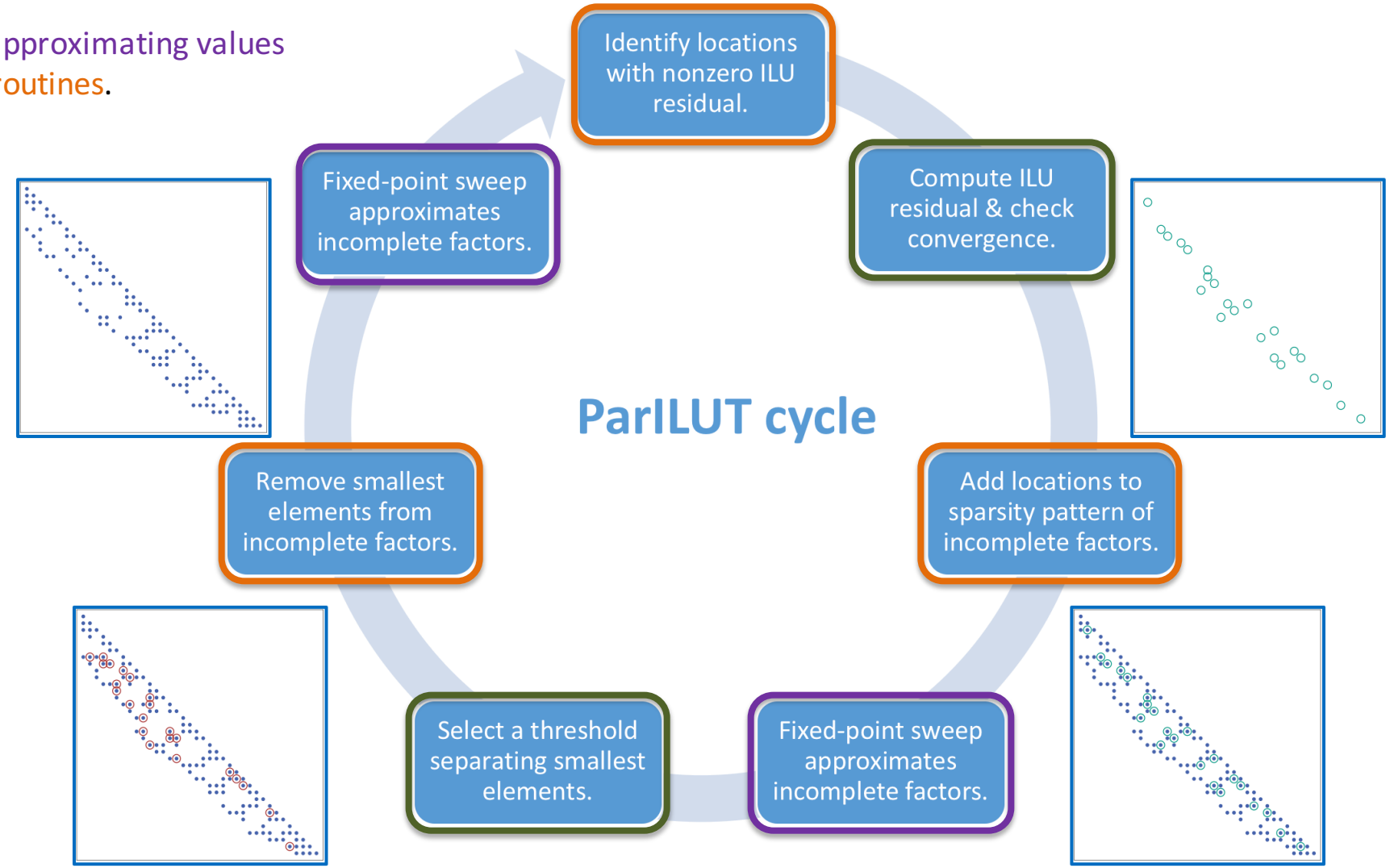
1. *Select a set of nonzero locations.*
2. *Compute values in those locations such that $A \approx L \cdot U$ is a "good" approximation.*
3. ***Maybe change some locations in favor of locations that result in a better preconditioner.***
4. *Repeat until the preconditioner quality stagnates.*

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



ParILUT

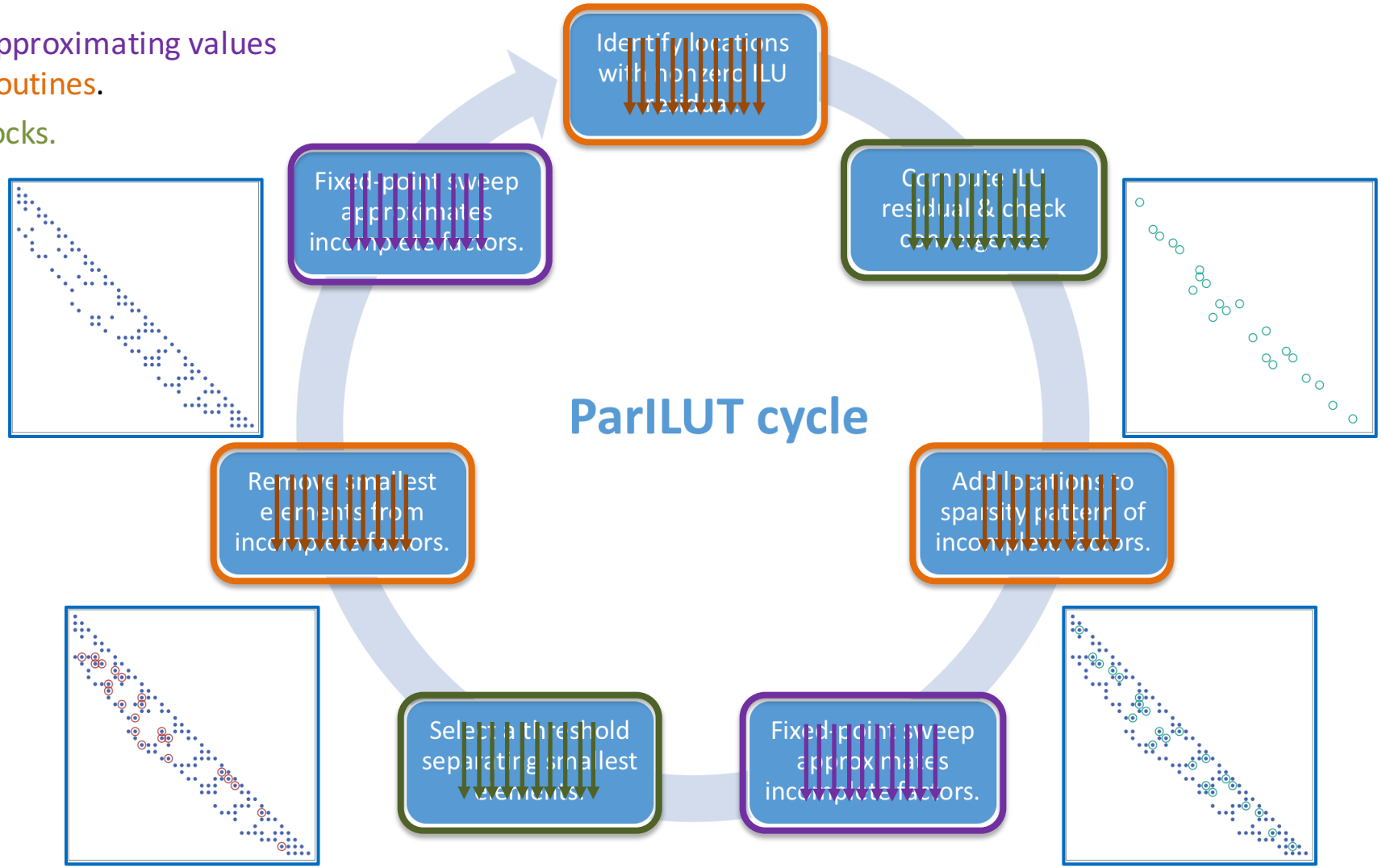
Interleaving fixed-point sweeps approximating values with pattern-changing symbolic routines.



ParILUT : Parallelism inside the blocks

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

Parallelism inside the building blocks.



Parallelism inside the blocks: Fixed-point sweeps

Fixed-point sweep approximates incomplete factors.

Compute ILU residual & check convergence.

Fixed-point sweeps approximate values in ILU factors and residual[1]:

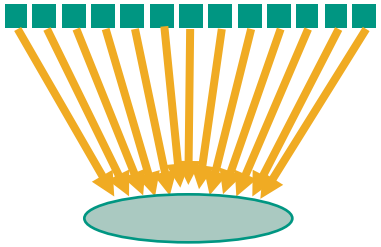
- Inherently parallel operation.
- Elements can be updated asynchronously.

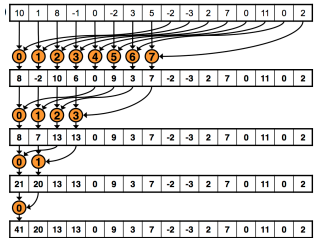
- *We can expect 100% parallel efficiency if number of cores < number of elements*

- Residual norm is a global reduction.

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

bilinear fixed-point iteration can be parallelized by elements





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

Parallelism inside the blocks: Candidate search

Identify locations that are symbolically nonzero:

- Combination of sparse matrix product and sparse matrix sums.
- Building blocks available in SparseBLAS.
- Blocks can be combined into one kernel for higher (memory) efficiency.
- Kernel can be parallelized by rows.

- *Cost heavily dependent on sparsity pattern.*
- *Kernel performance bound by memory bandwidth.*

Identify locations with nonzero ILU residual.

$$\mathcal{S}^* = (\mathcal{S}(A) \cup \mathcal{S}(L \cdot U)) \setminus \mathcal{S}(L + U)$$

sparse matrix product

sparse matrix sum

sparse matrix sum

Parallelism inside the blocks: Selecting thresholds

A threshold separating the smallest elements is needed
for removing insignificant locations and keeping sparsity.

Select a threshold separating smallest elements.

- Standard approach: sort / selection algorithms
  - High computational cost
  - Memory-intensive
  - Hard to parallelize

- Thresholds do not need to be exact:
  - Inaccurate thresholds result in a few additional / less elements.
  - We can use sampling to get reasonable approximations.
  - Multiple sampling-based selection runs allow to generate thresholds of reasonable quality in parallel.

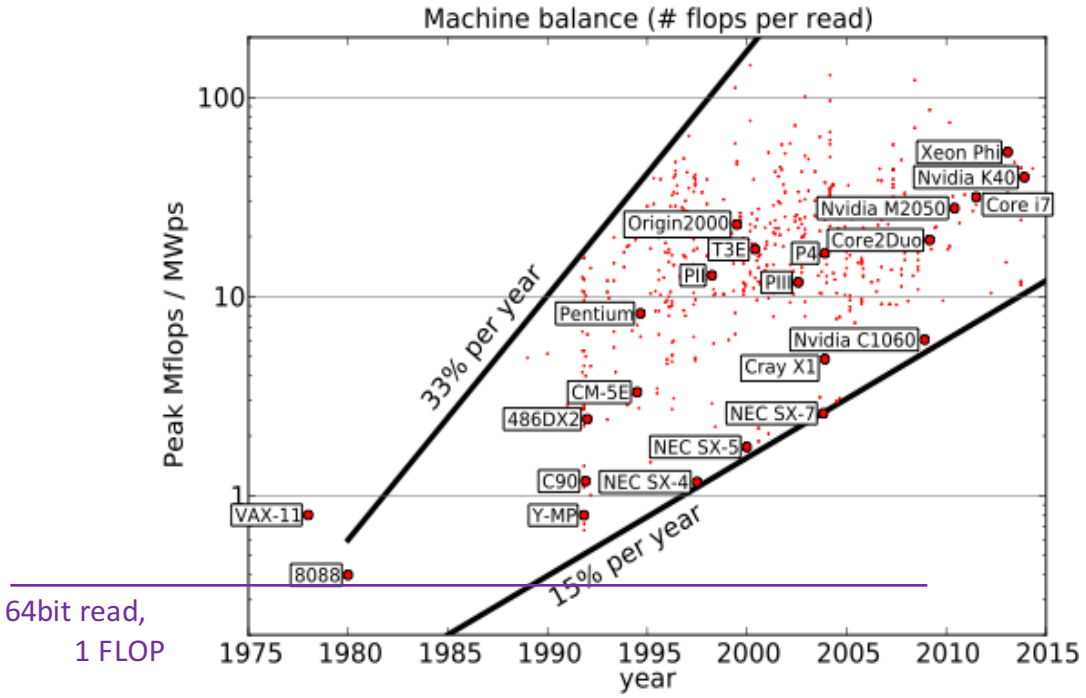
- Is this appropriate for many-core architectures with 5K threads executing simultaneously?

Threshold selection on parallel architectures

Selection algorithms traditionally based on re-arranging elements in memory.

- SelectionSort ($\mathcal{O}(n^2)$ comparisons, $\mathcal{O}(n)$ element swaps)
- QuickSelect (average $\mathcal{O}(n)$ comparisons, $\mathcal{O}(n)$ element swaps)
- Floyd-Rivest algorithm ($n + min(k, n-k) + O(\sqrt{n})$)
- IntroSelect (worst case $\mathcal{O}(n)$ comparisons, $\mathcal{O}(n)$ element swaps)

Select a threshold separating smallest elements.



John D. McCaplin (TACC)

- Compute power (#FLOPs) grows much faster than memory bandwidth.
- Data-rearranging selection algorithms become inefficient.

"Operations are free, memory access is what counts."

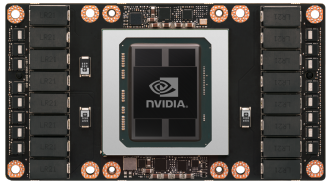
Threshold selection on parallel architectures: StreamSelect

Rethink the overall strategy!

- Primary goal: reduce the memory traffic.
- Account for high core counts, each having a set of registers.
- Assume "nice" distribution of values (~uniform).
- Accept some inaccuracy in the generated threshold (approximation).

1. *Find the largest and smallest elements to get the data range.*
2. *Generate a **fine grid of thresholds**, **distribute** them to the cores.*
3. ***Stream all data one single time.***
4. ***Each core** handles a set of thresholds and **counts** how many **elements** are larger/smaller.*
5. ***Select the threshold** with the element count closest to the target value.*

80 Multiprocessors, each with 64 FP32 cores
[oversubscribe with threads to hide latency]



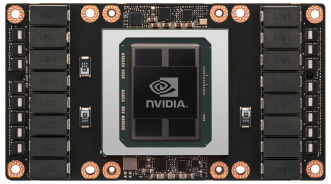
NVIDIA V100 "Volta"
7.8 TFLOP/s DP
16GB RAM @900 GB/s

Threshold selection on parallel architectures: StreamSelect

Rethink the overall strategy!

- Primary goal: reduce the memory traffic.
- Account for high core counts, each having a set of registers.
- Assume "nice" distribution of values (~uniform).
- Accept some inaccuracy in the generated threshold (approximation).

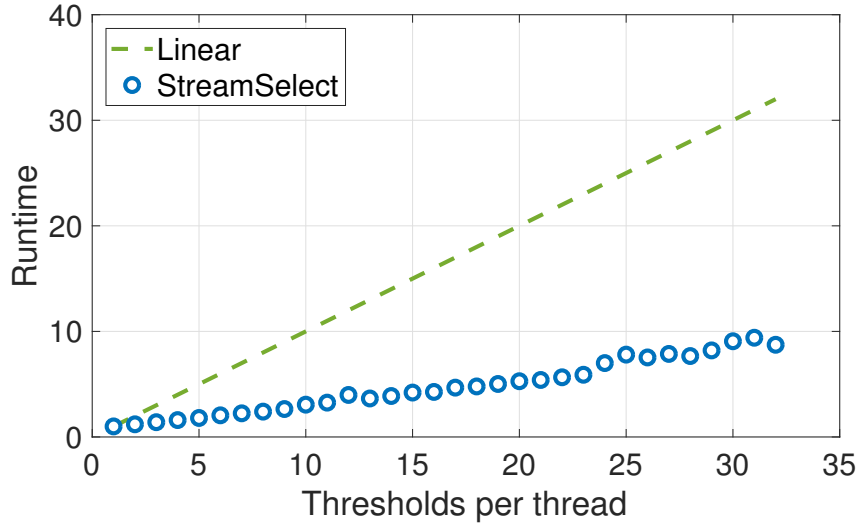
1. *Find the largest and smallest elements to get the data range.*
2. *Generate a **fine grid of thresholds**, **distribute** them to the cores.*
3. ***Stream all data one single time.***
4. ***Each core** handles a set of thresholds and **counts** how many **elements** are larger/smaller.*
5. ***Select the threshold** with the element count closest to the target value.*



NVIDIA V100 "Volta"
7.8 TFLOP/s DP
16GB RAM @900 GB/s

80 Multiprocessors, each with 64 FP32 cores
[oversubscribe with threads to hide latency]

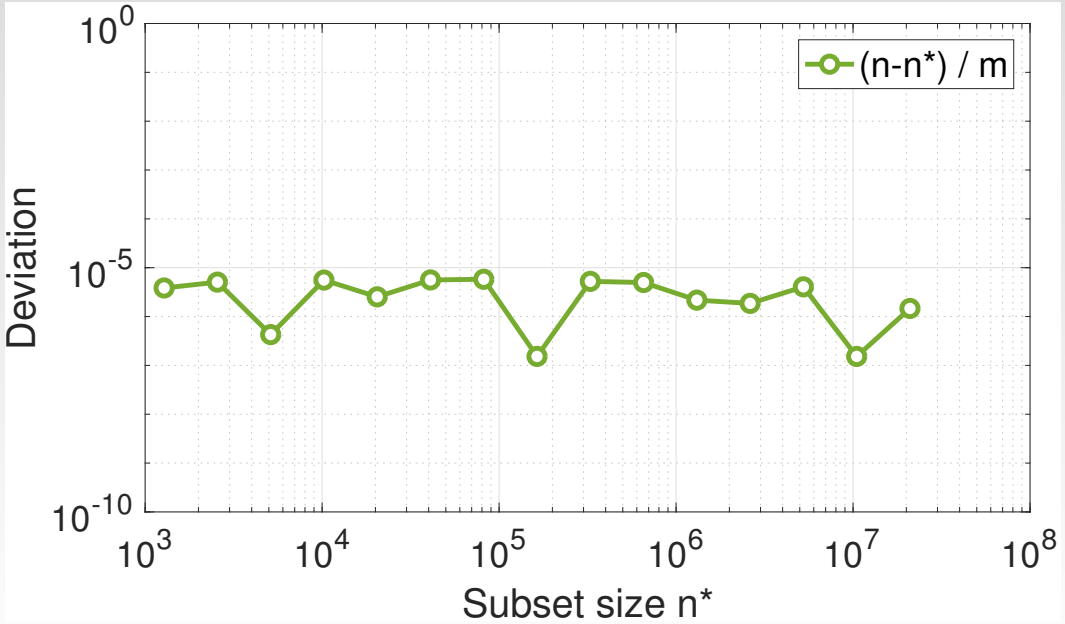
Run 5120 threads (bind to cores) each a few thresholds:



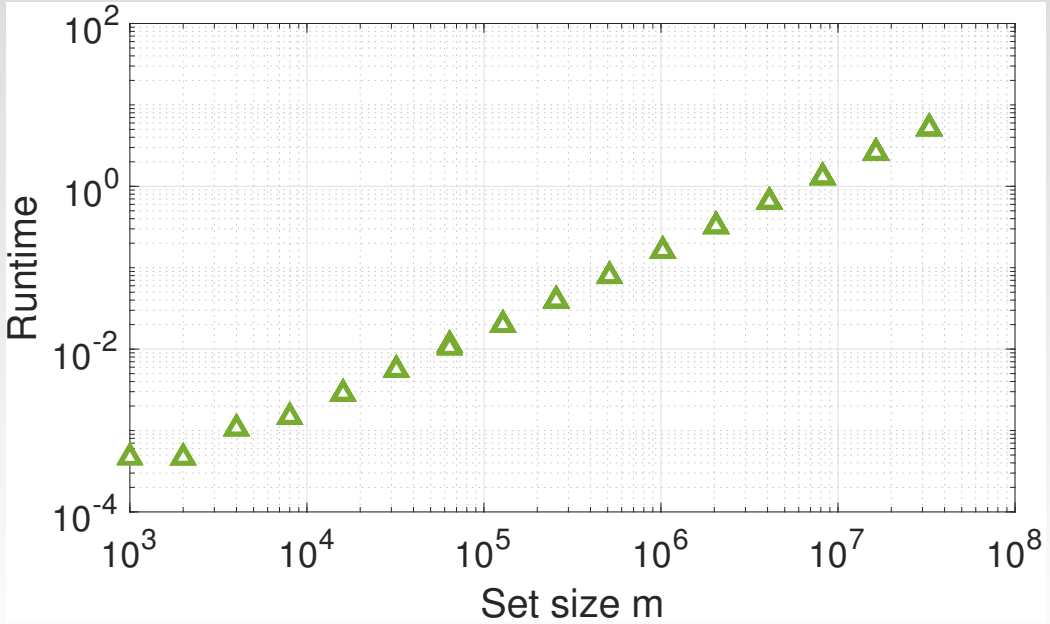
32 thresholds gives mesh granularity of 6.1035e-06

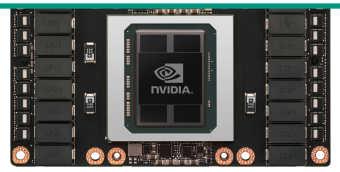
Threshold selection on parallel architectures: StreamSelect

- Set size m, subset size n*
- For uniform distribution: quality ~ mesh granularity.

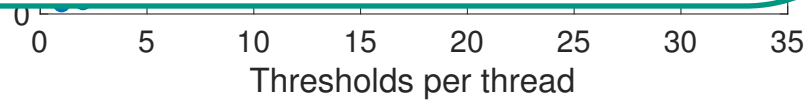


- Runtime increases linear with set size.
- Runtime independent of subset size.





NVIDIA V100 "Volta"
7.8 TFLOP/s DP
16GB RAM @900 GB/s

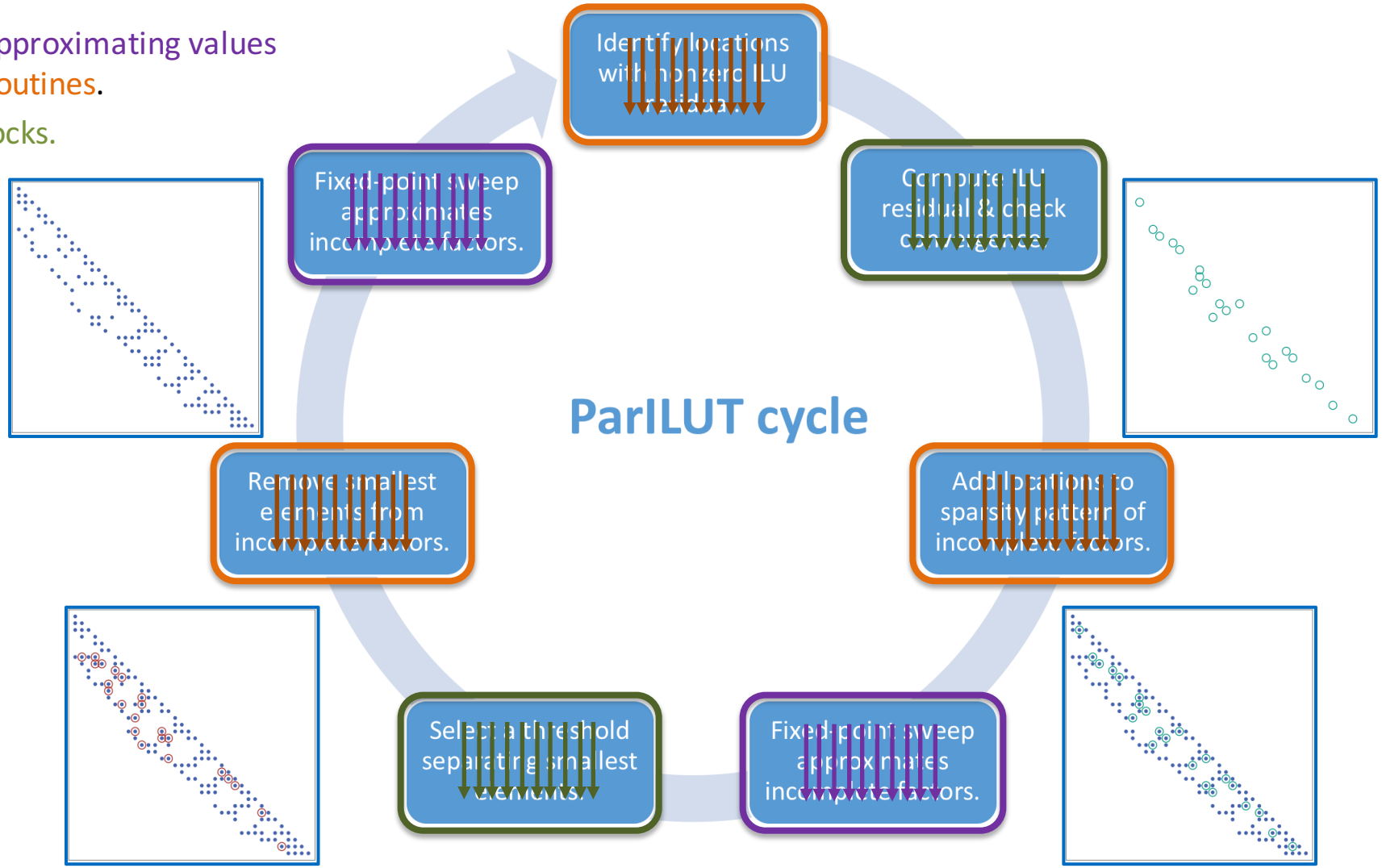


32 thresholds gives mesh granularity of 6.1035e-06

Is this a future-oriented algorithm?

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

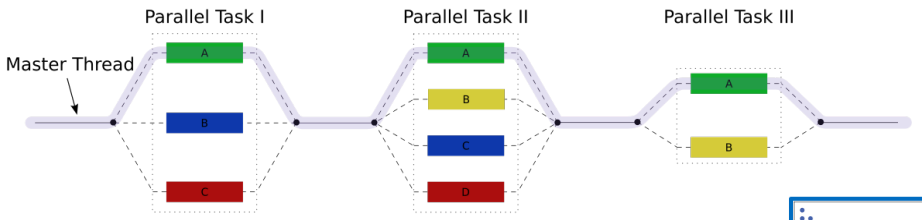
Parallelism inside the building blocks.



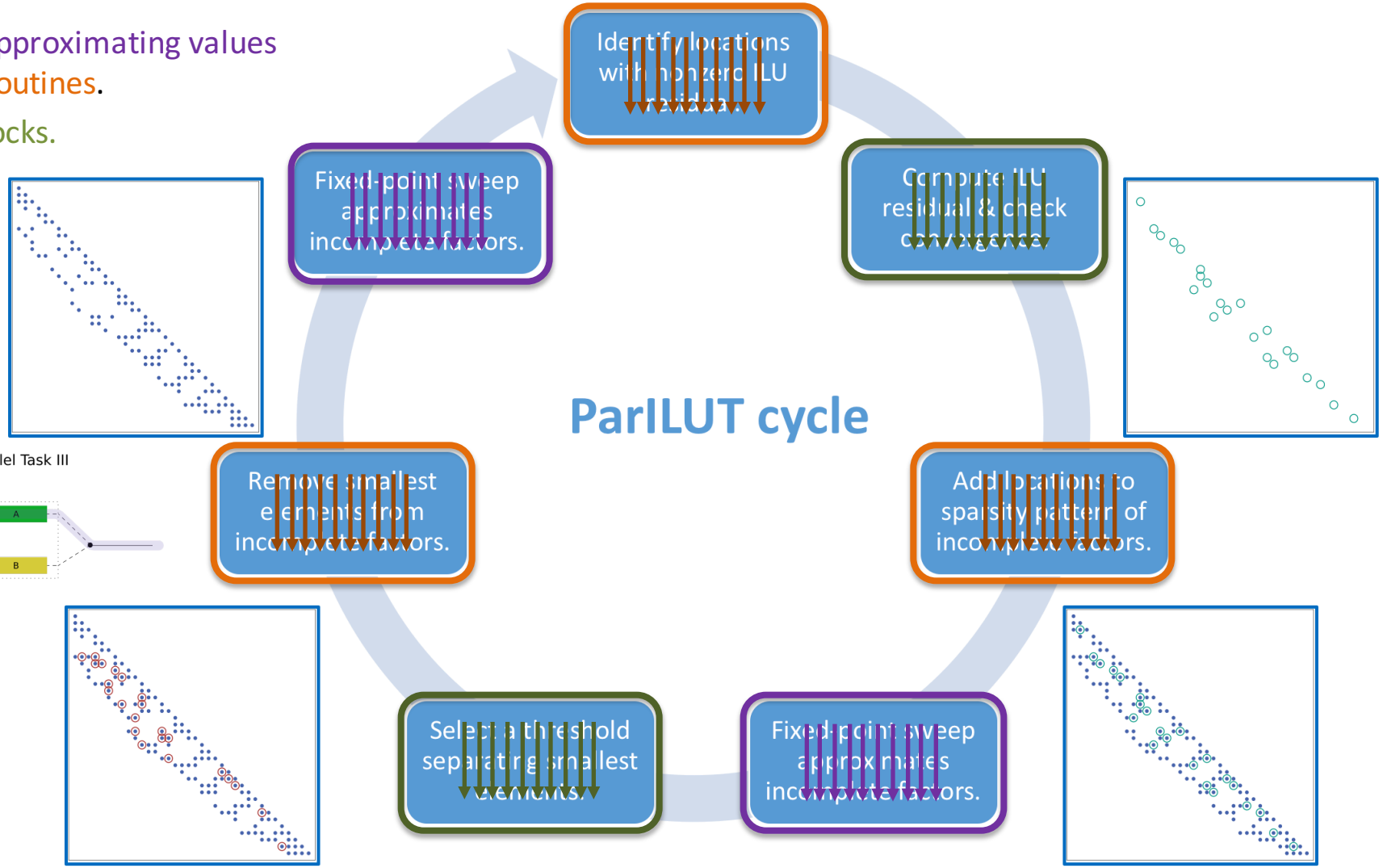
Is this a future-oriented algorithm?

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

Parallelism inside the building blocks.



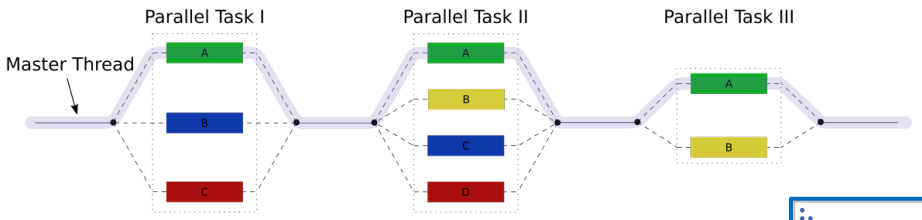
Bulk-Synchronous Algorithm!



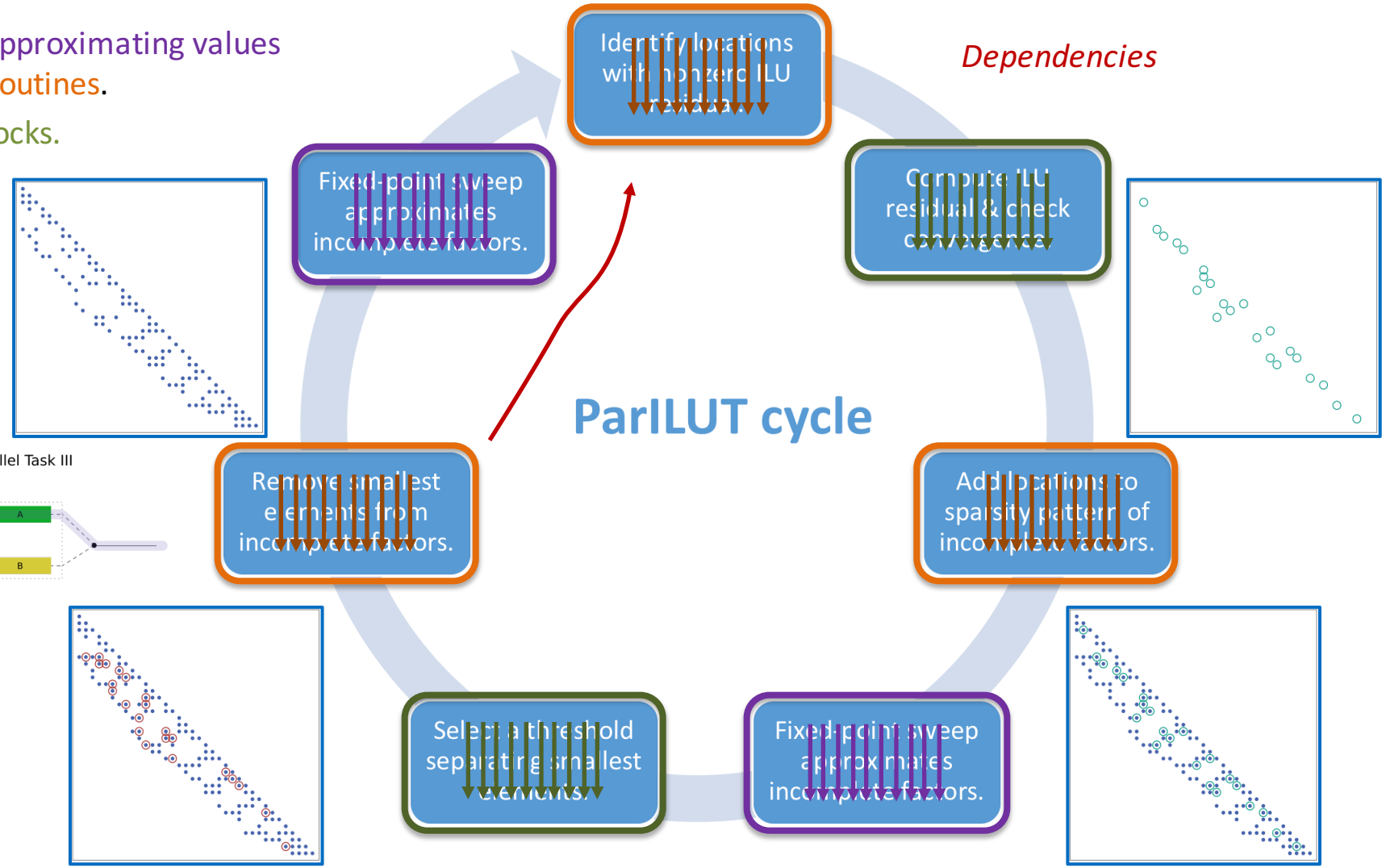
Is this a future-oriented algorithm?

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

Parallelism inside the building blocks.



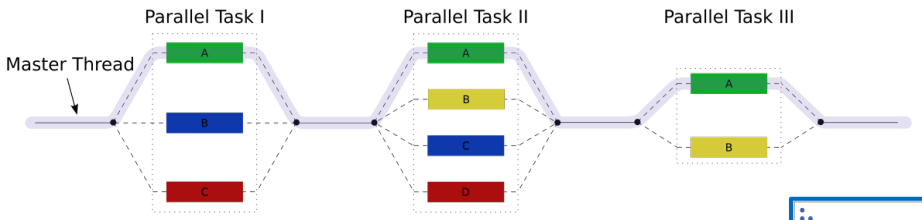
Bulk-Synchronous Algorithm!



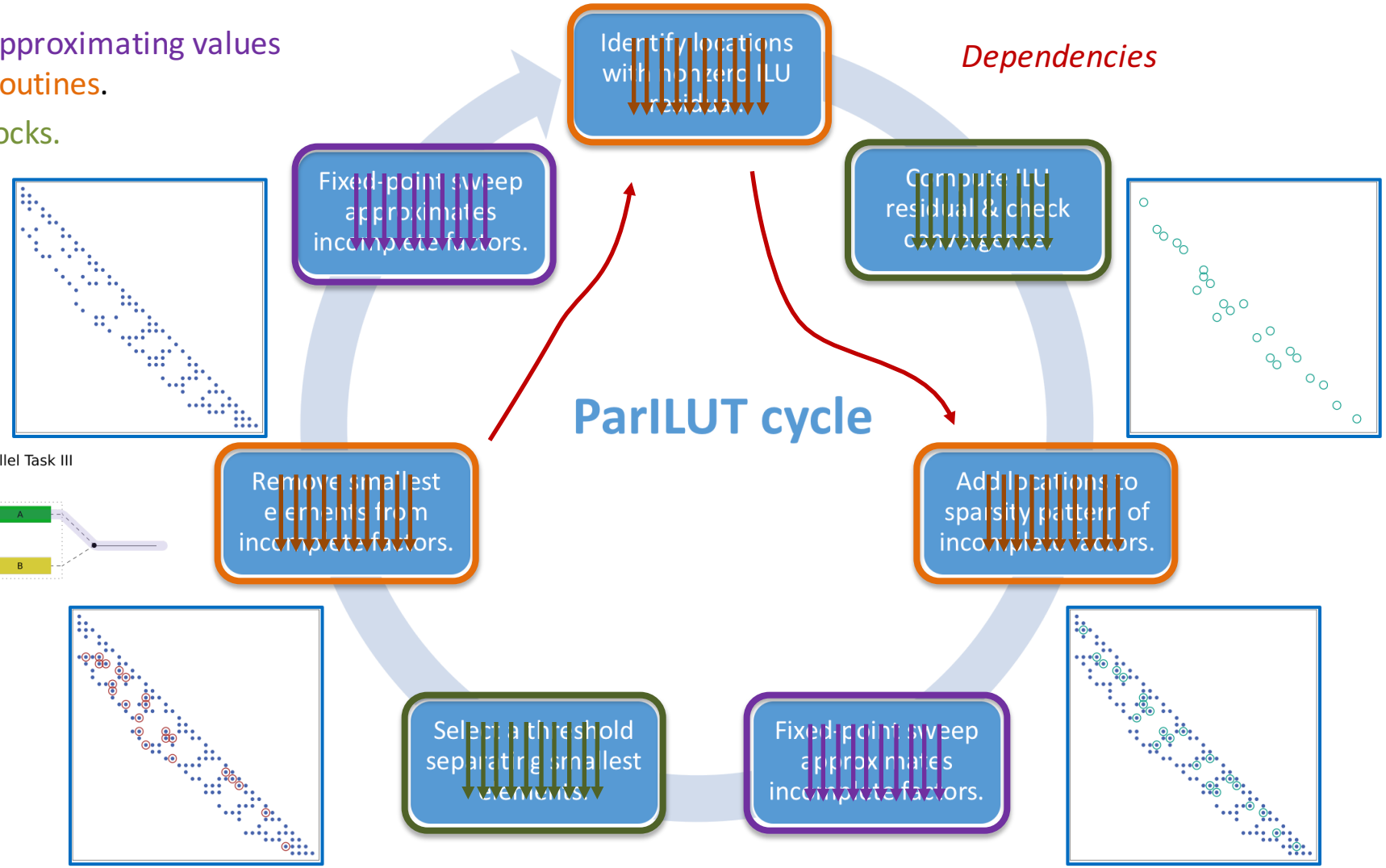
Is this a future-oriented algorithm?

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

Parallelism inside the building blocks.



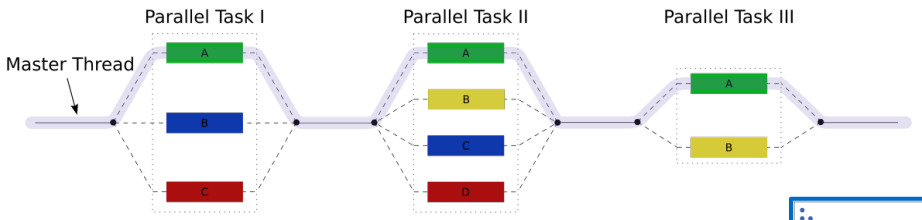
Bulk-Synchronous Algorithm!



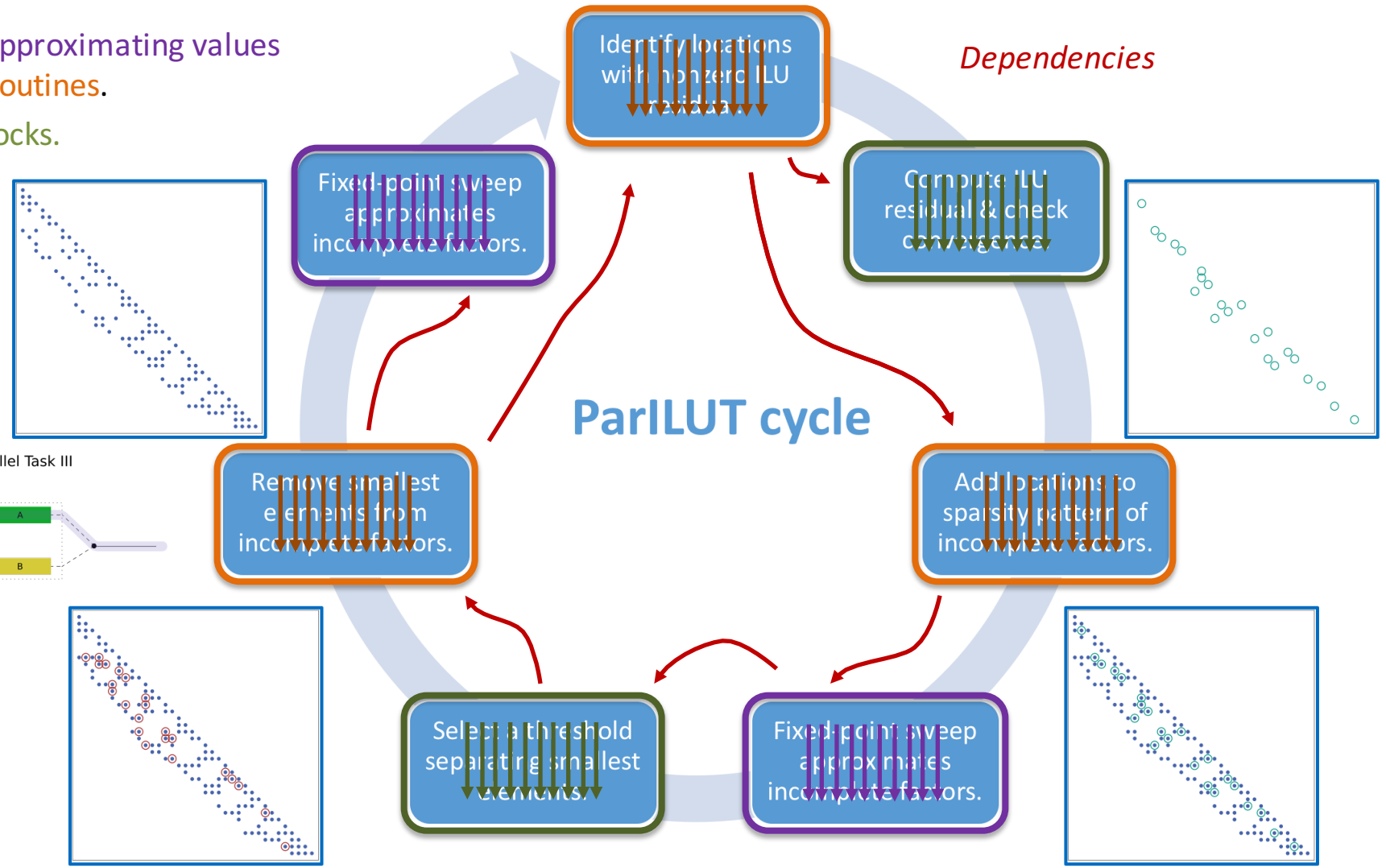
Is this a future-oriented algorithm?

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

Parallelism inside the building blocks.



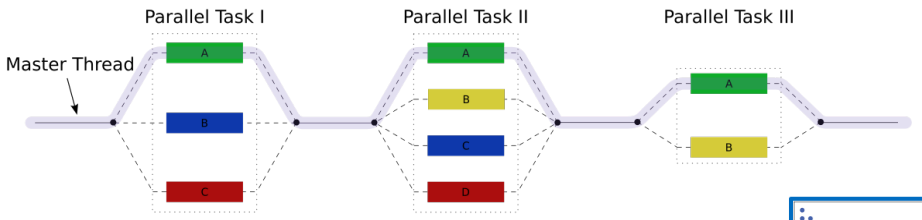
Bulk-Synchronous Algorithm!



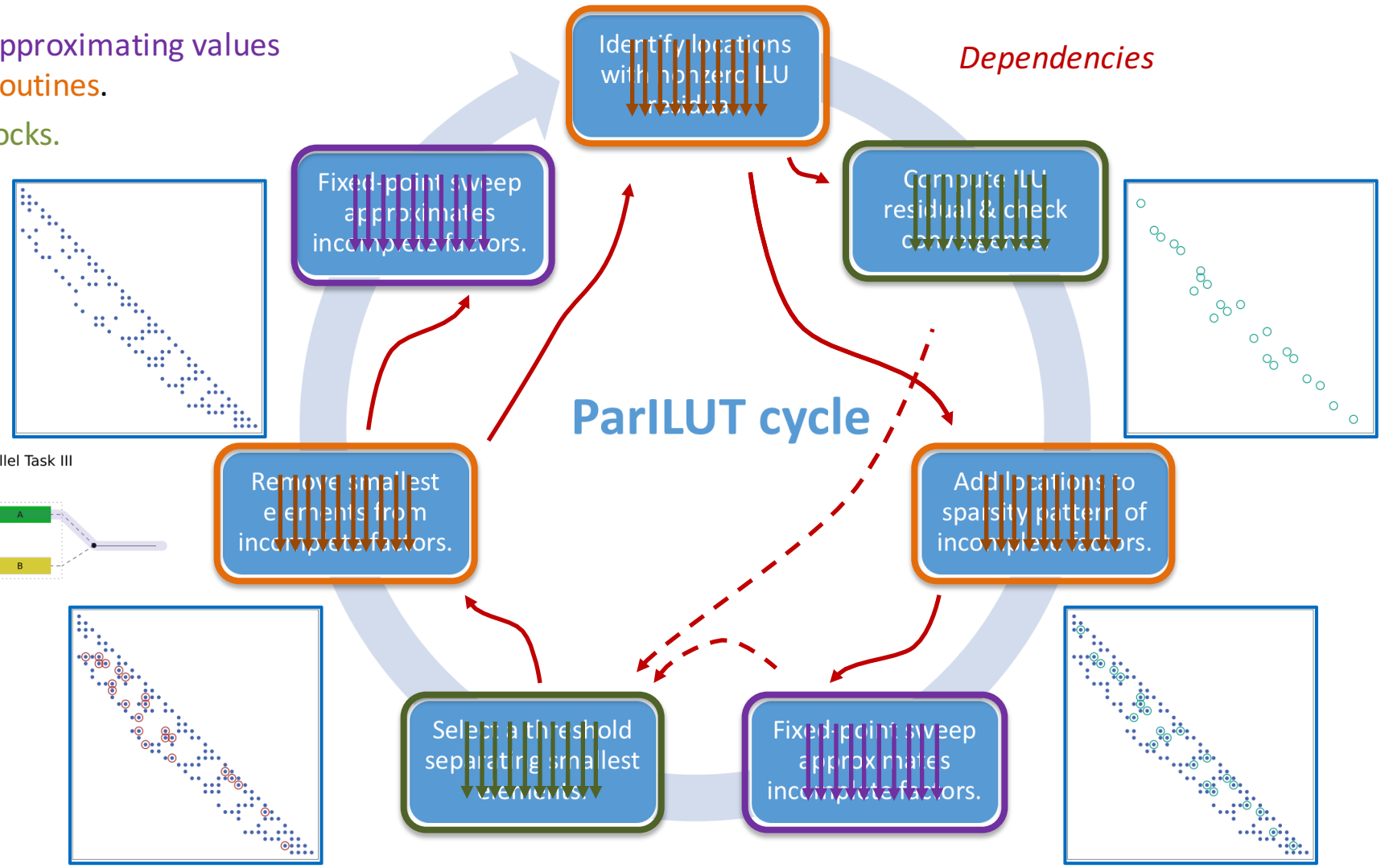
Is this a future-oriented algorithm?

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

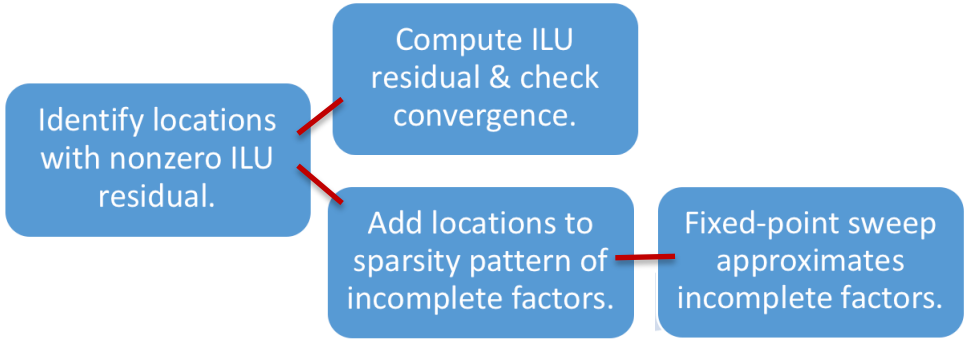
Parallelism inside the building blocks.



Bulk-Synchronous Algorithm!

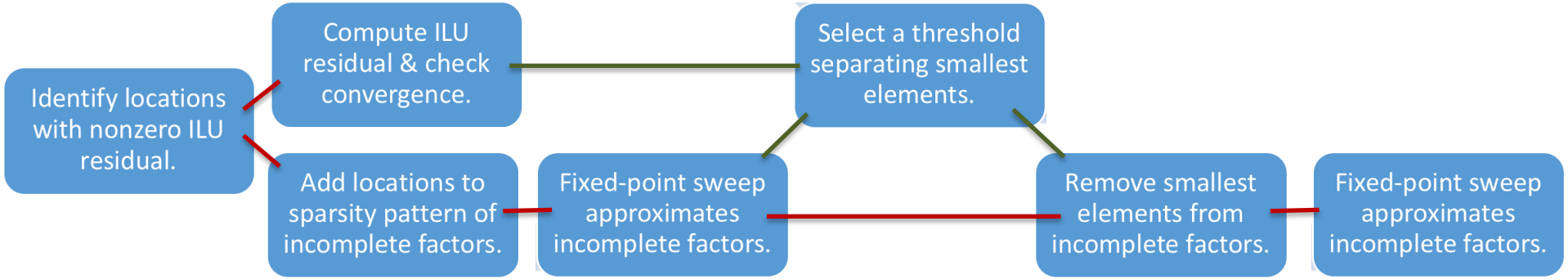


Is this a future-oriented algorithm?



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

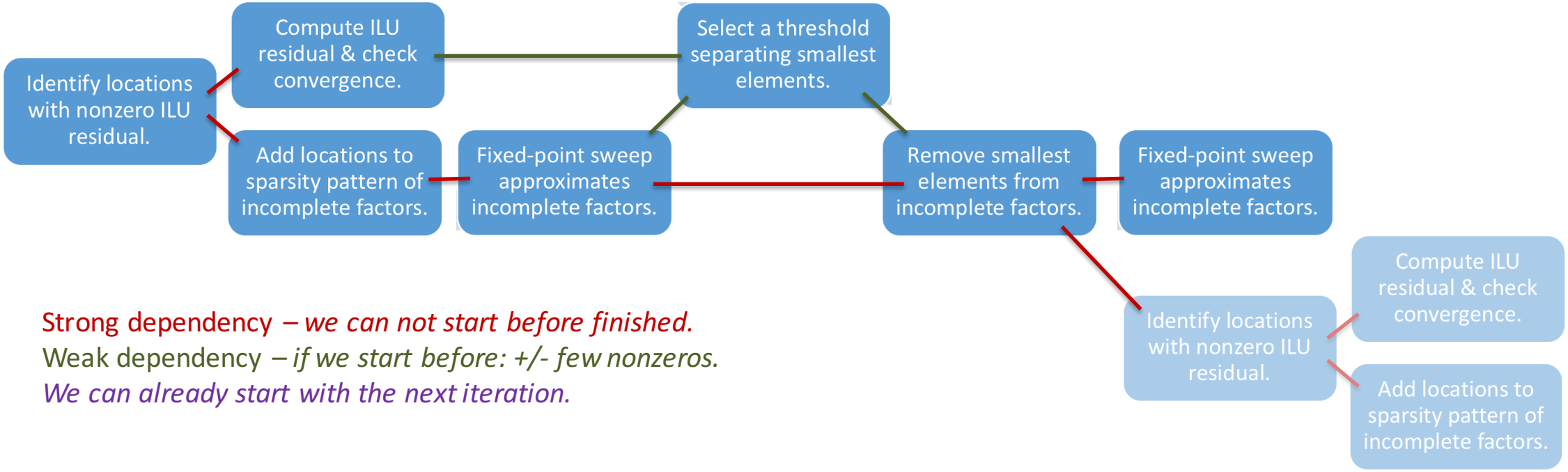
Is this a future-oriented algorithm?



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

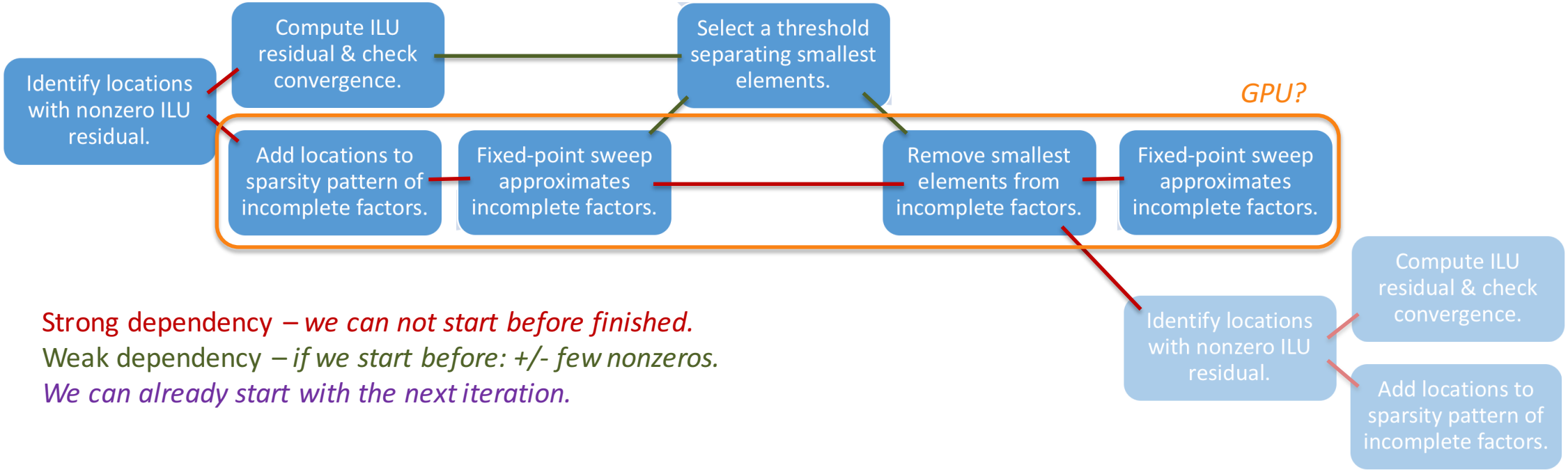
Weak dependency – *if we start before: +/- few nonzeros.*

Is this a future-oriented algorithm?



Strong dependency – *we can not start before finished.*
Weak dependency – *if we start before: +/- few nonzeros.*
We can already start with the next iteration.

Is this a future-oriented algorithm?



Strong dependency – *we can not start before finished.*
Weak dependency – *if we start before: +/- few nonzeros.*
We can already start with the next iteration.

Excellent candidate for hybrid hardware?
Asynchronous execution?

ParILUT – A New Parallel Threshold ILU

Karlsruhe Institute of Technology

Slides available:

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!

Ginkgo

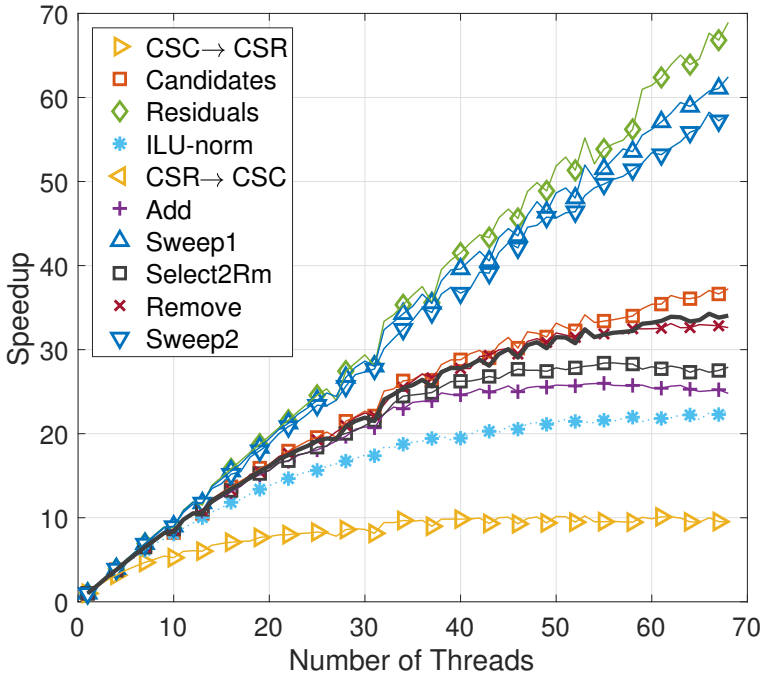
This research is in cooperation with Edmond Chow (GaTech) and Jack Dongarra (University of Tennessee).

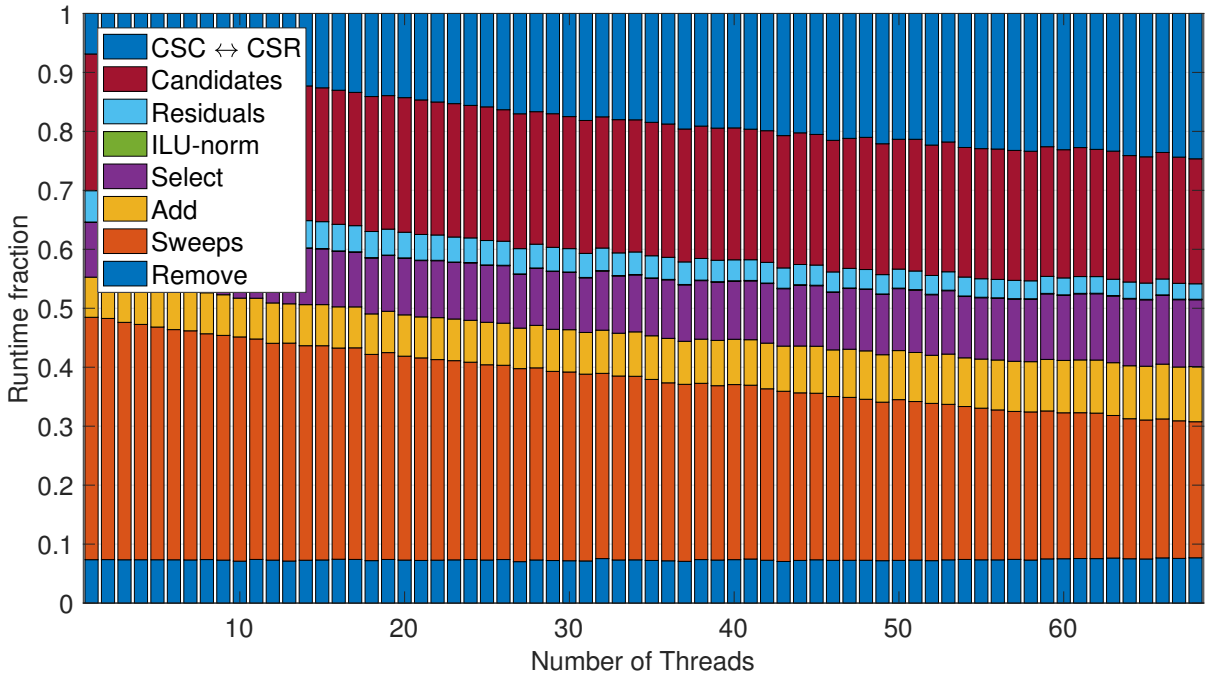
HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES
Helmholtz Impuls und Vernetzungsfond
VH-NG-1241

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.



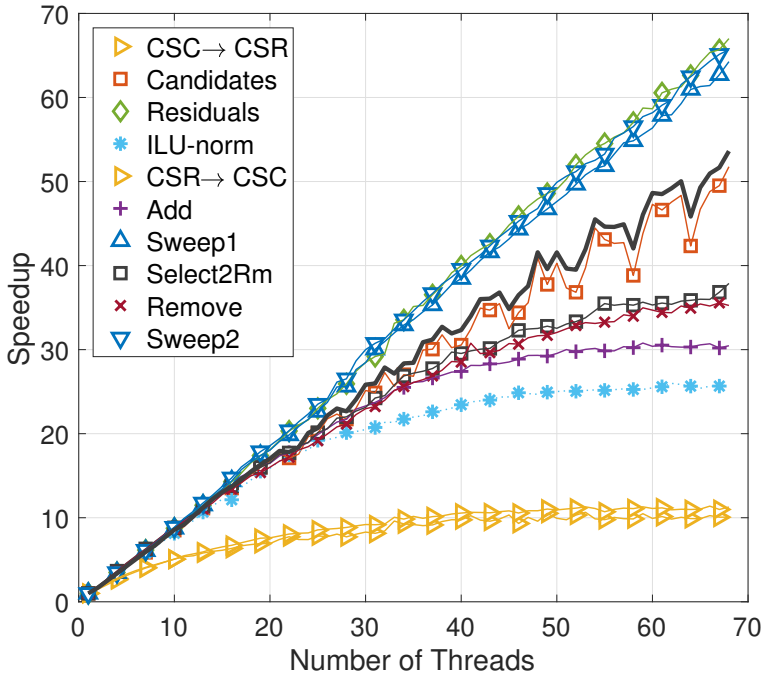


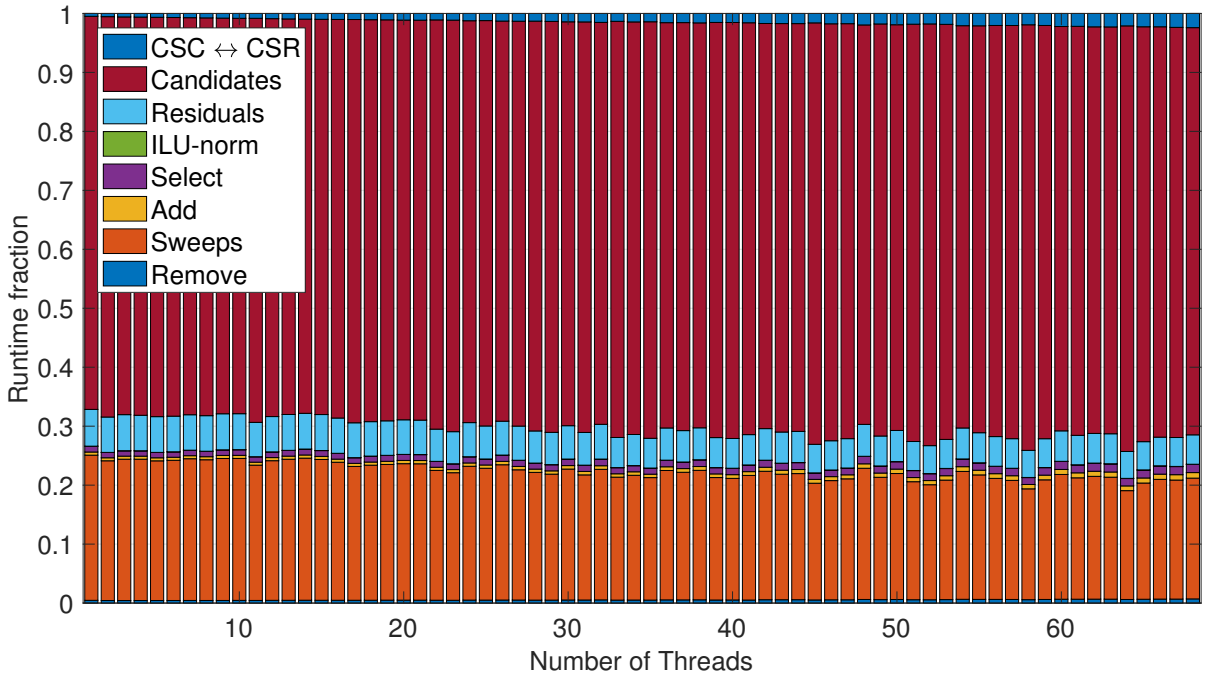
- Building blocks scale with 15% - 100% parallel efficiency.
- Transposition and sort are the bottlenecks.
- Overall speedup ~35x when using 68 KNL cores.

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

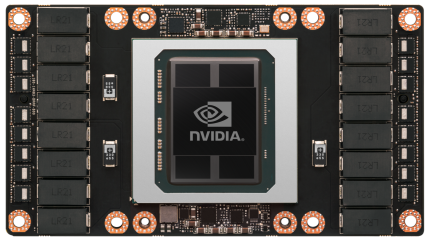
Runtime of 5 ParILUT / ParICT steps and speedup over SuperLU ILUT*.

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?

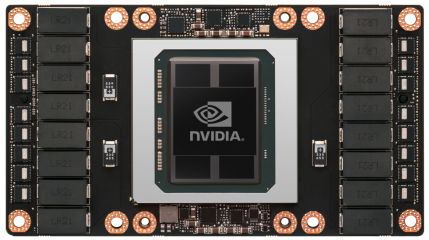
NVIDIA V100 “Volta”
7.8 TFLOP/s DP
16GB RAM @900 GB/s



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

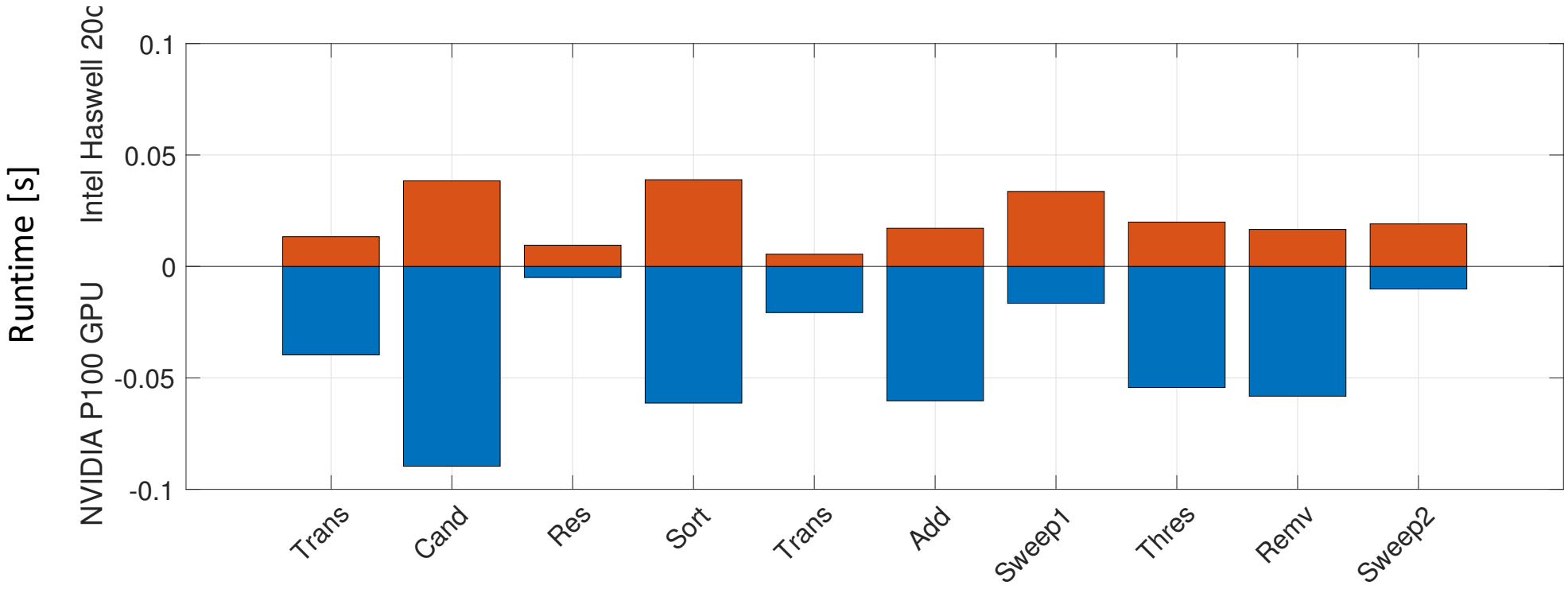
How about GPUs?

NVIDIA V100 "Volta"
7.8 TFLOP/s DP
16GB RAM @900 GB/s



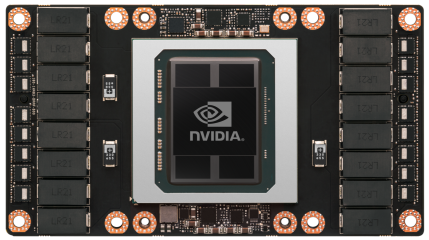
- Fine-grained parallelism
- High bandwidth for coalescent reads
- No deep cache hierarchy
- We need to oversubscribe cores for hiding latency

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



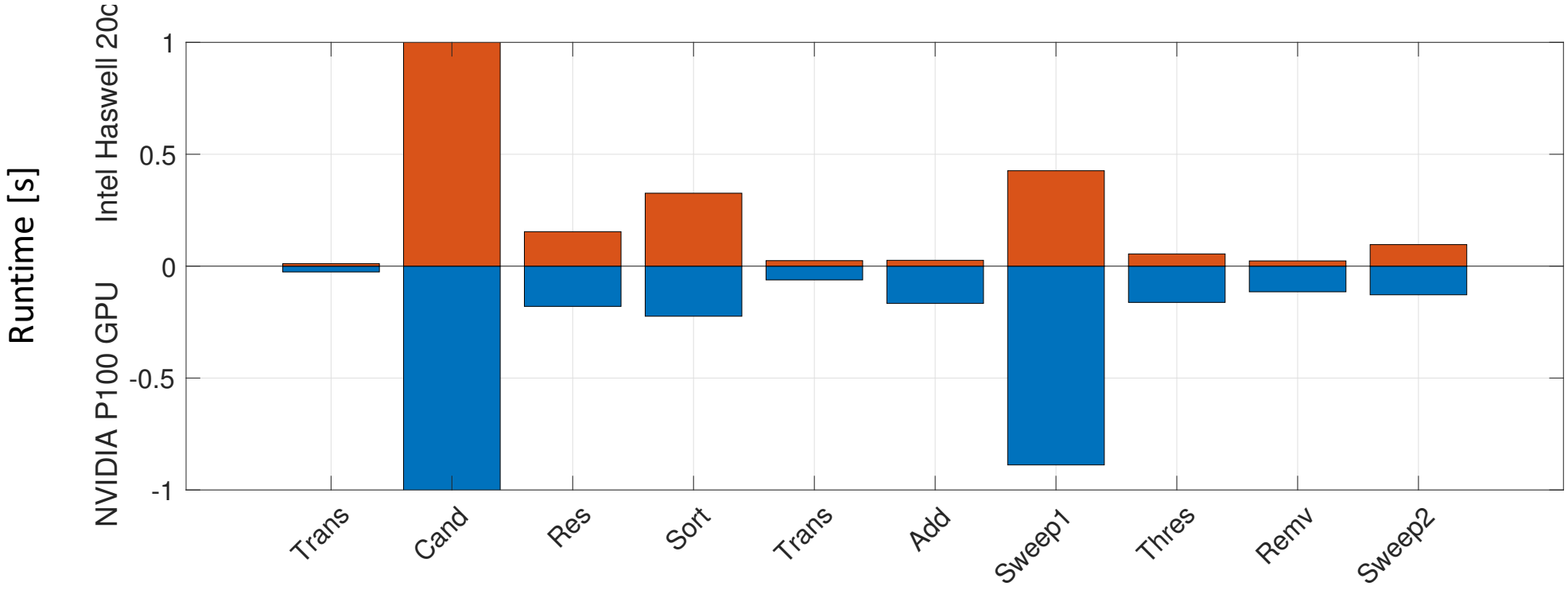
How about GPUs?

NVIDIA V100 "Volta"
7.8 TFLOP/s DP
16GB RAM @900 GB/s



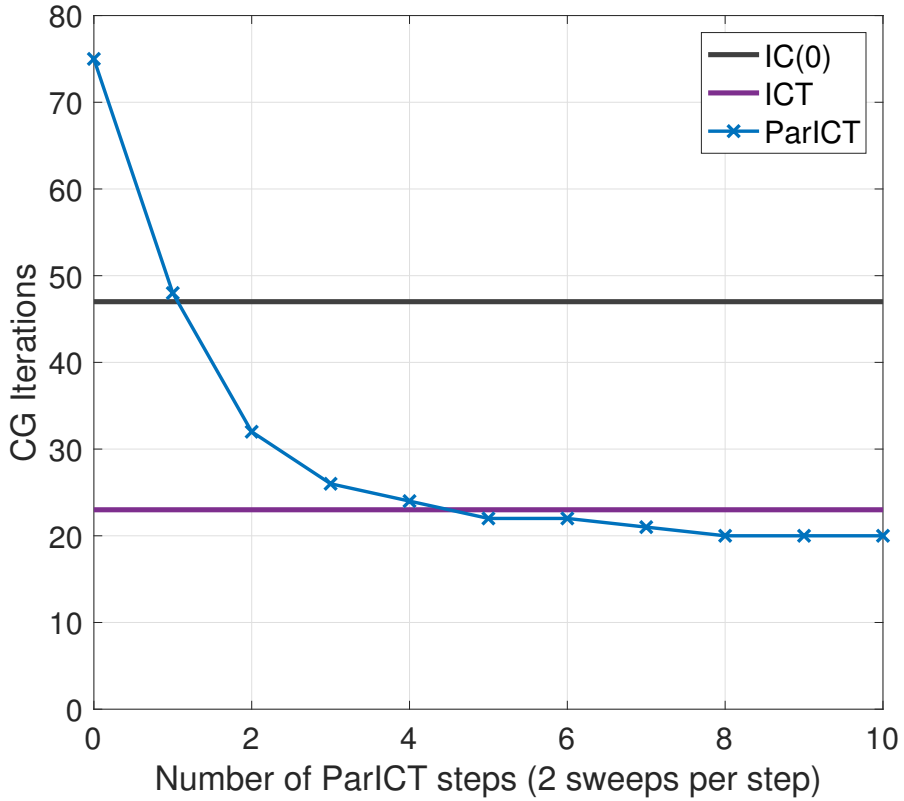
- Fine-grained parallelism
- High bandwidth for coalescent reads
- No deep cache hierarchy
- We need to oversubscribe cores for hiding latency

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



ParILUT quality

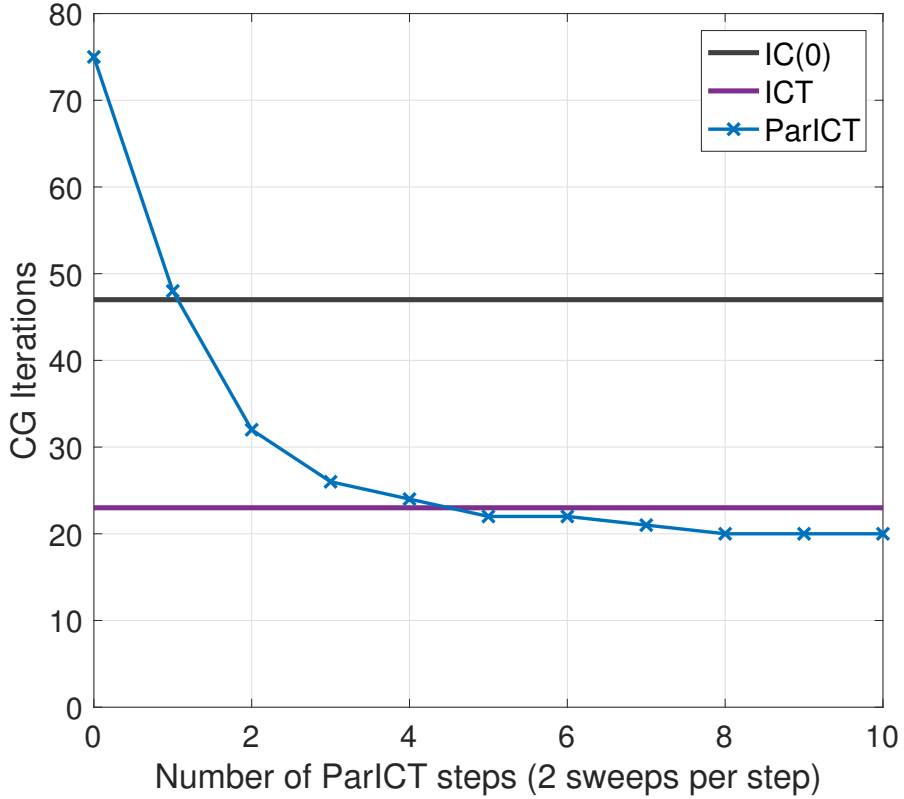
Anisotropic fluid flow problem
n: 741, nz: 4,951

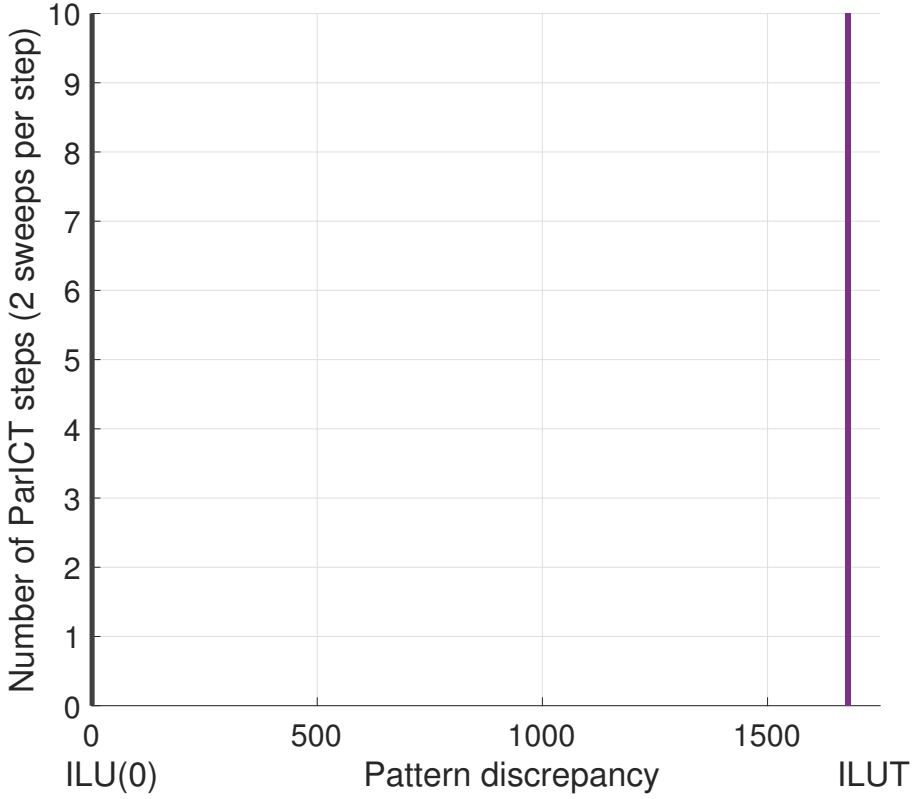


- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than ILUT?

ParILUT quality

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

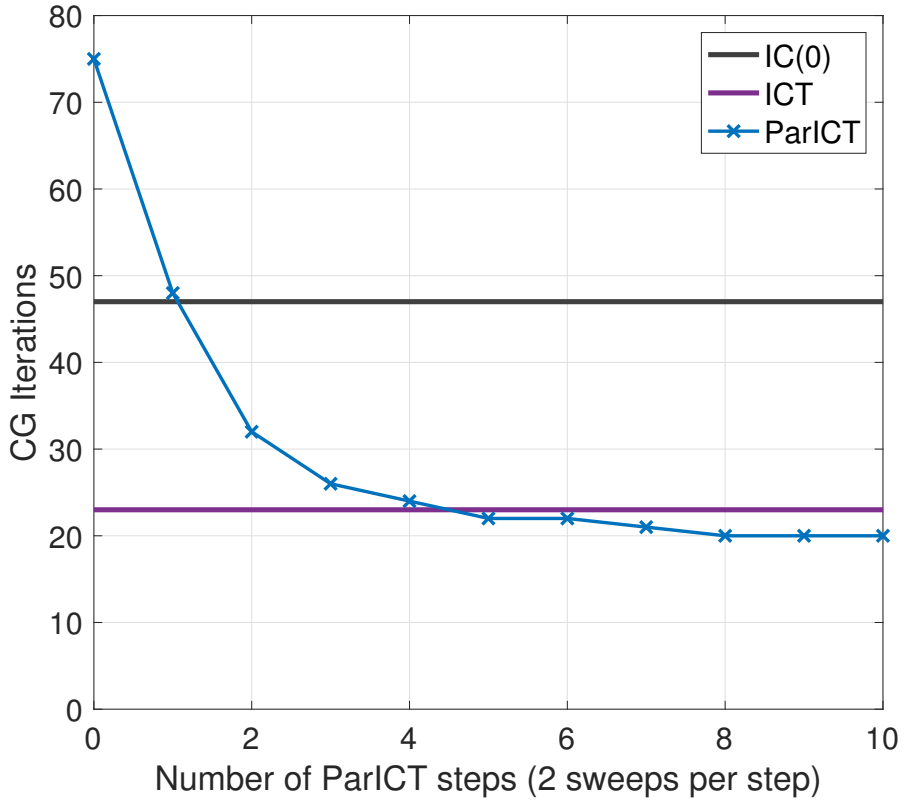


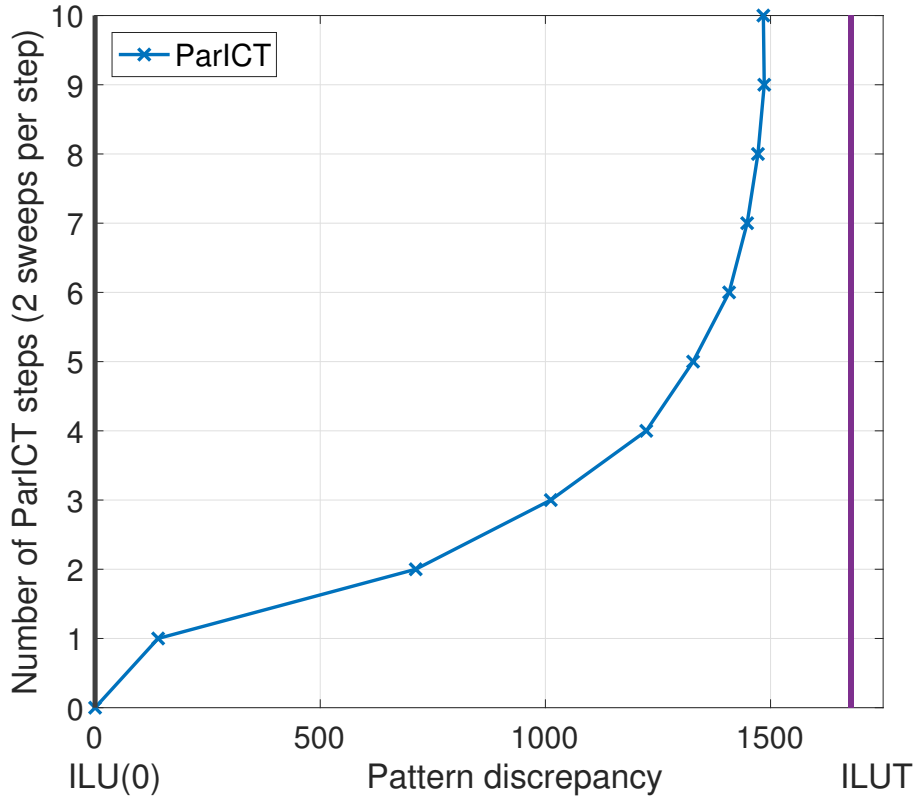


- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than ILUT?

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).

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	–	–

Preconditioning

We iteratively solve a linear system of the form $Ax = b$
Where $A \in \mathbb{R}^{n \times n}$ nonsingular and $b, x \in \mathbb{R}^n$.

The convergence rate typically depends on the conditioning of the linear system, which is the ratio between the largest and smallest eigenvalue.

$$\text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{\frac{1}{\lambda_{min}}}{\frac{1}{\lambda_{max}}} = \text{cond}_2(A^{-1})$$

With $M \approx A^{-1}$ we can transform the linear system into a system with a lower condition number:

$$MAx = Mb \text{ (left preconditioned)}$$
$$AMy = b,\ x = My \quad \text{(right preconditioned)}$$

If we now apply the iterative solver to the preconditioned system, $MAx = Mb$ we usually get faster convergence.

Preconditioning

We iteratively solve a linear system of the form $Ax = b$
Where $A \in \mathbb{R}^{n \times n}$ nonsingular and $b, x \in \mathbb{R}^n$.

The convergence rate typically depends on the conditioning of the linear system, which is the ratio between the largest and smallest eigenvalue.

$$\text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{\frac{1}{\lambda_{min}}}{\frac{1}{\lambda_{max}}} = \text{cond}_2(A^{-1})$$

With $M \approx A^{-1}$ we can transform the linear system into a system with a lower condition number:

$$MAx = Mb \text{ (left preconditioned)}$$
$$AMy = b,\ x = My \quad \text{(right preconditioned)}$$

If we now apply the iterative solver to the preconditioned system, $MAx = Mb$ we usually get faster convergence.

Assume $M = A^{-1}$, then: $x = MAx = Mb$.
Solution right available, but computing $M = A^{-1}$ is expensive…

Preconditioning

We iteratively solve a linear system of the form $Ax = b$
Where $A \in \mathbb{R}^{n \times n}$ nonsingular and $b, x \in \mathbb{R}^n$.

The convergence rate typically depends on the conditioning of the linear system, which is the ratio between the largest and smallest eigenvalue.

$$\text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{\frac{1}{\lambda_{min}}}{\frac{1}{\lambda_{max}}} = \text{cond}_2(A^{-1})$$

With $M \approx A^{-1}$ we can transform the linear system into a system with a lower condition number:

$$MAx = Mb \text{ (left preconditioned)}$$
$$AMy = b,\ x = My \quad \text{(right preconditioned)}$$

If we now apply the iterative solver to the preconditioned system, $MAx = Mb$ we usually get faster convergence.

Assume $M = A^{-1}$, then: $x = MAx = Mb$.
Solution right available, but computing $M = A^{-1}$ is expensive…

The preconditioned system MA is rarely formed explicitely, insted M is applied implicitely: $z_{k+1} = Mr_{k+1}$

Preconditioning

We iteratively solve a linear system of the form $Ax = b$
Where $A \in \mathbb{R}^{n \times n}$ nonsingular and $b, x \in \mathbb{R}^n$.

The convergence rate typically depends on the conditioning of the linear system, which is the ratio between the largest and smallest eigenvalue.

$$\text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{\frac{1}{\lambda_{min}}}{\frac{1}{\lambda_{max}}} = \text{cond}_2(A^{-1})$$

With $M \approx A^{-1}$ we can transform the linear system into a system with a lower condition number:

$$MAx = Mb \text{ (left preconditioned)}$$
$$AMy = b,\ x = My \quad \text{(right preconditioned)}$$

If we now apply the iterative solver to the preconditioned system, $MAx = Mb$ we usually get faster convergence.

Assume $M = A^{-1}$, then: $x = MAx = Mb$.
Solution right available, but computing $M = A^{-1}$ is expensive…

The preconditioned system MA is rarely formed explicitely, insted M is applied implicitely: $z_{k+1} = Mr_{k+1}$

Instead of forming the preconditioner $M \approx A^{-1}$ explicitly and applying as $z_{k+1} = Mr_{k+1}$,
Incomplete Factorization Preconditioner (ILU) try to find an approximate factorization $A \approx L \cdot U$.

Preconditioning

We iteratively solve a linear system of the form $Ax = b$
Where $A \in \mathbb{R}^{n \times n}$ nonsingular and $b, x \in \mathbb{R}^n$.

The convergence rate typically depends on the conditioning of the linear system, which is the ratio between the largest and smallest eigenvalue.

$$\text{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{\frac{1}{\lambda_{min}}}{\frac{1}{\lambda_{max}}} = \text{cond}_2(A^{-1})$$

With $M \approx A^{-1}$ we can transform the linear system into a system with a lower condition number:

$$MAx = Mb \text{ (left preconditioned)}$$
$$AMy = b,\ x = My \quad \text{(right preconditioned)}$$

If we now apply the iterative solver to the preconditioned system, $MAx = Mb$ we usually get faster convergence.

Assume $M = A^{-1}$, then: $x = MAx = Mb$.
Solution right available, but computing $M = A^{-1}$ is expensive…

The preconditioned system MA is rarely formed explicitely, insted M is applied implicitely: $z_{k+1} = Mr_{k+1}$

Instead of forming the preconditioner $M \approx A^{-1}$ explicitly and applying as $z_{k+1} = Mr_{k+1}$,
Incomplete Factorization Preconditioner (ILU) try to find an approximate factorization $A \approx L \cdot U$.

In the application phase, the preconditioner is only given implicitly, requiring two triangular solves:

$$z_{k+1} = Mr_{k+1}$$
$$M^{-1} z_{k+1} = r_{k+1}$$
$$L \underbrace{U z_{k+1}}_{=:y} = r_{k+1}$$
$$\Rightarrow Ly = r_{k+1}, \quad Uz_{k+1} = y$$