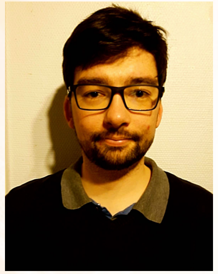
Algorithm Design in the Advent of Exascale Computing

4th International Symposium on Research and Education of Computational Science (RECS)
University of Tokyo, October 2nd, 2019

Hartwig Anzt, Terry Cojean, Goran Flegar, Thomas Grűtzmacher, Pratik Nayak, Tobias Ribizel
Steinbuch Centre for Computing (SCC)


Terry Cojean


Thomas Grützmacher


Pratik Nayak


Tobias Ribizel


Mike Tsai

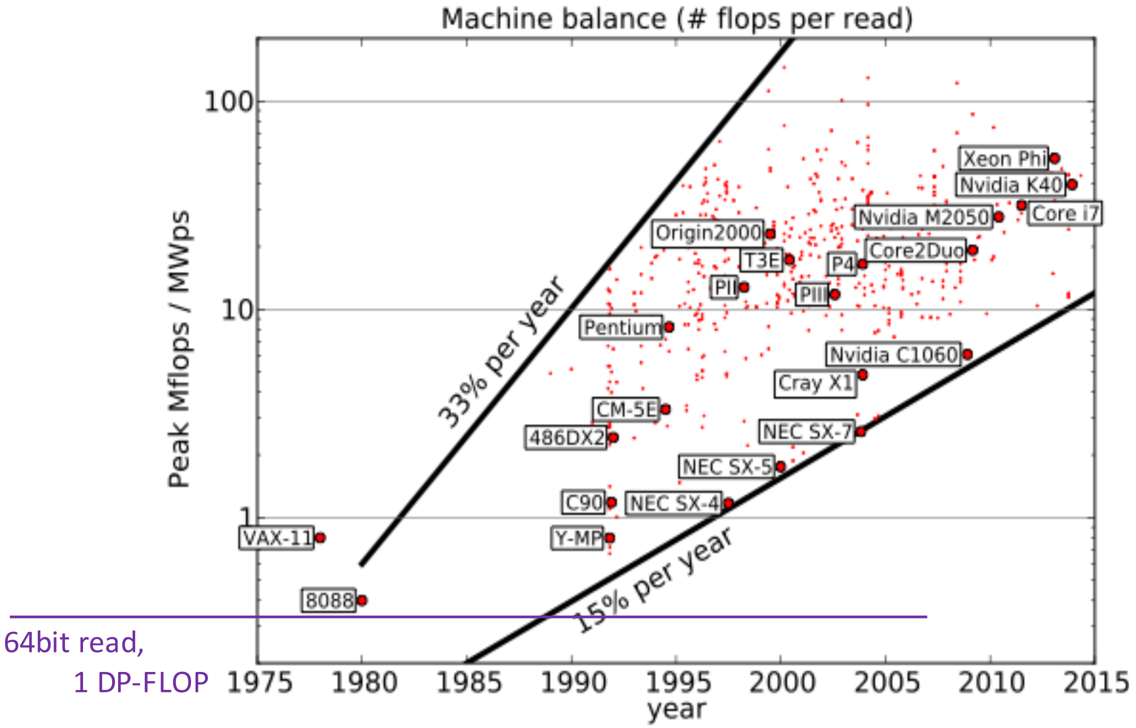
Where do we stand?

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

Copyright@ORNL

Where do we stand?

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence



John D. McCalpin (TACC)

1. Compute power (#FLOPs) grows much faster than bandwidth.

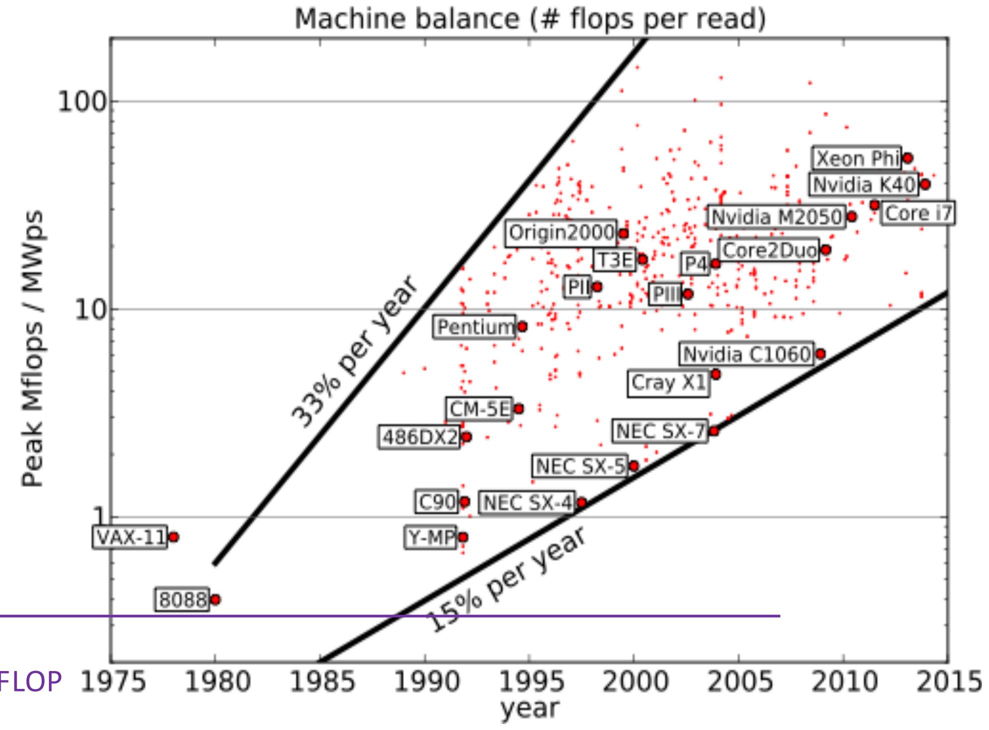
"Operations are free, mem access and comm is what counts."

Where do we stand?

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY | summit

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

Copyright@ORNL



John D. McCalpin (TACC)

1. Compute power (#FLOPs) grows much faster than bandwidth.
   "Operations are free, mem access and comm is what counts."
2. Manycore architectures need new algorithmic approaches.
   "Sync-Free fine-grained parallelism needed."

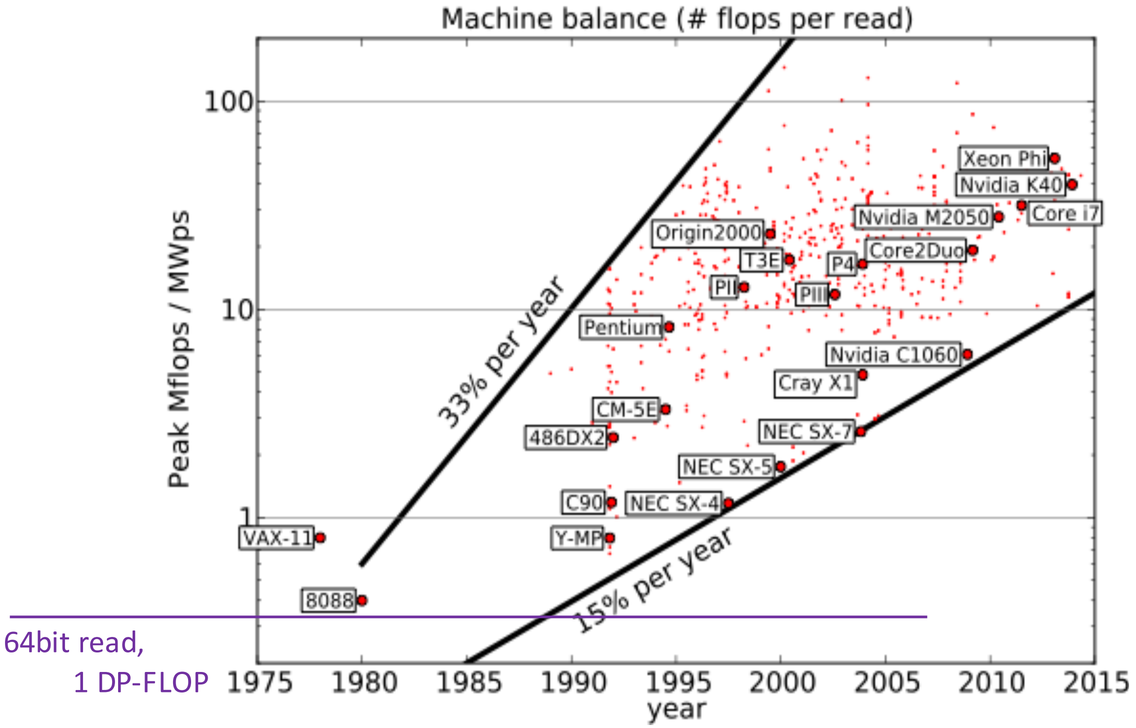
Where do we stand?

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

summit

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

Copyright@ORNL



John D. McCalpin (TACC)

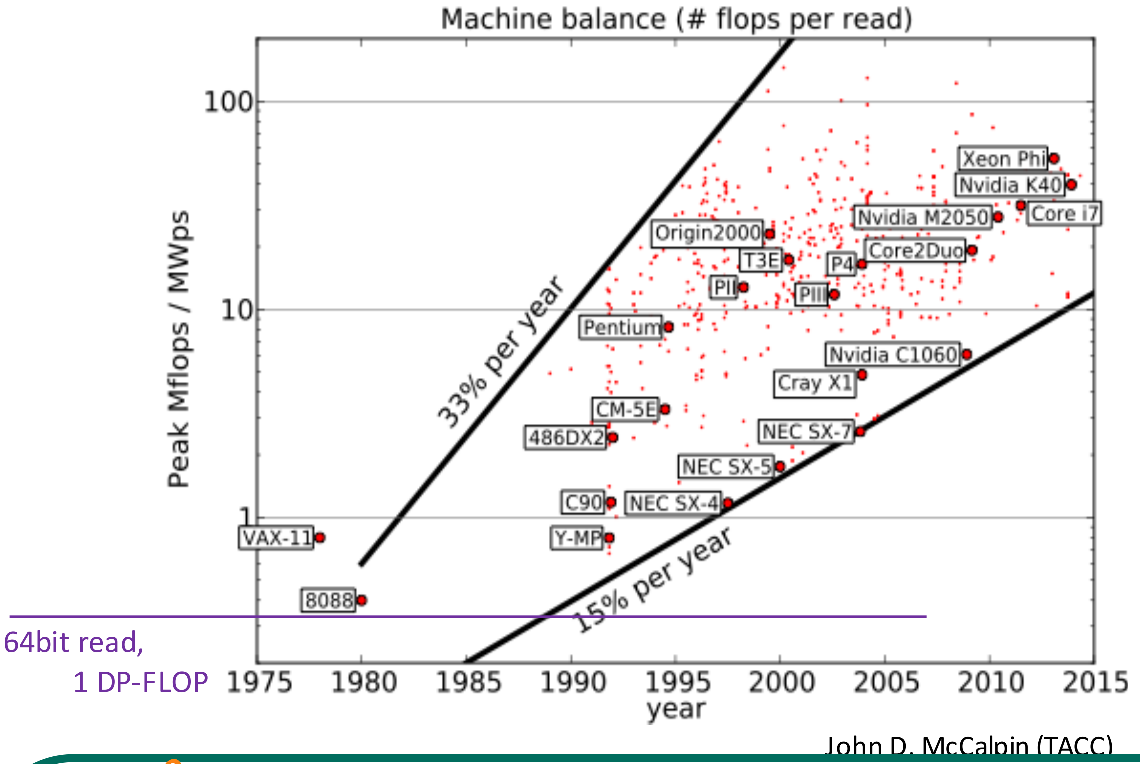
1. Compute power (#FLOPs) grows much faster than bandwidth.
   "Operations are free, mem access and comm is what counts."
2. Manycore architectures need new algorithmic approaches.
   "Sync-Free fine-grained parallelism needed."
3. Software lives longer than hardware.
   "We need a paradigm change to embrace software development."

Where do we stand?

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

summit

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence



Bandwidth Challenge

1. Compute power (#FLOPs) grows much faster than bandwidth.
   "Operations are free, mem access and comm is what counts."

Manycore Challenge

2. Manycore architectures need new algorithmic approaches.
   "Sync-Free fine-grained parallelism needed."

Software Challenge

3. Software lives longer than hardware.
   "We need a paradigm change to embrace software development."

The Communication Bottleneck

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

The Communication Bottleneck

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

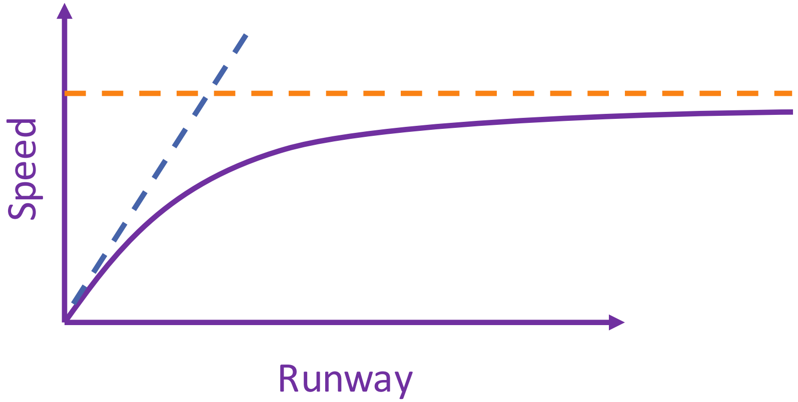
Roofline Model

Given certain hardware characteristics:

memory bandwidth, Acceleration
arithmetic power, Top Speed

the performance of any operation is

- either bound by the data access/communication (*memory bound*),
- or by the arithmetic operations (*compute bound*).



The Communication Bottleneck

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

summit

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

Roofline Model

Given certain hardware characteristics:

memory bandwidth, Acceleration

arithmetic power, Top Speed

the performance of any operation is

- either bound by the data access/communication (*memory bound*),
- or by the arithmetic operations (*compute bound*).

Matrix-Matrix Product (GEMM): $C = A \times B$ $\quad A, B, C \in \mathbb{R}^{n \times n}$

$3n^2$ Memory operations

$2n^3$ Arithmetic operations

We just need to increase the size, and at some point the operation becomes compute bound.

"we infinitely extend the acceleration runway"

The Communication Bottleneck

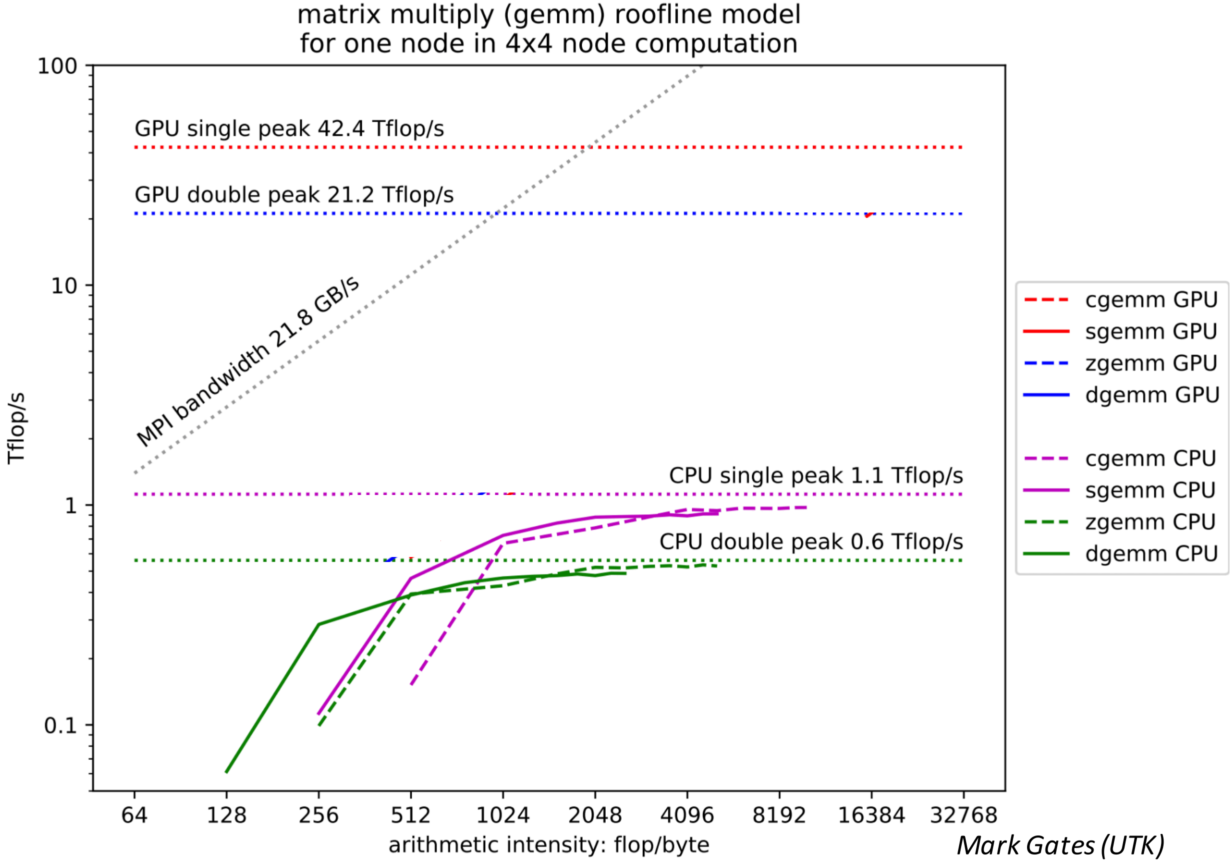
OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

summit

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

Dense Matrix Operations?

- The inter-node communication is the limiting resource;
- Each node has more computational power than what we can leverage;



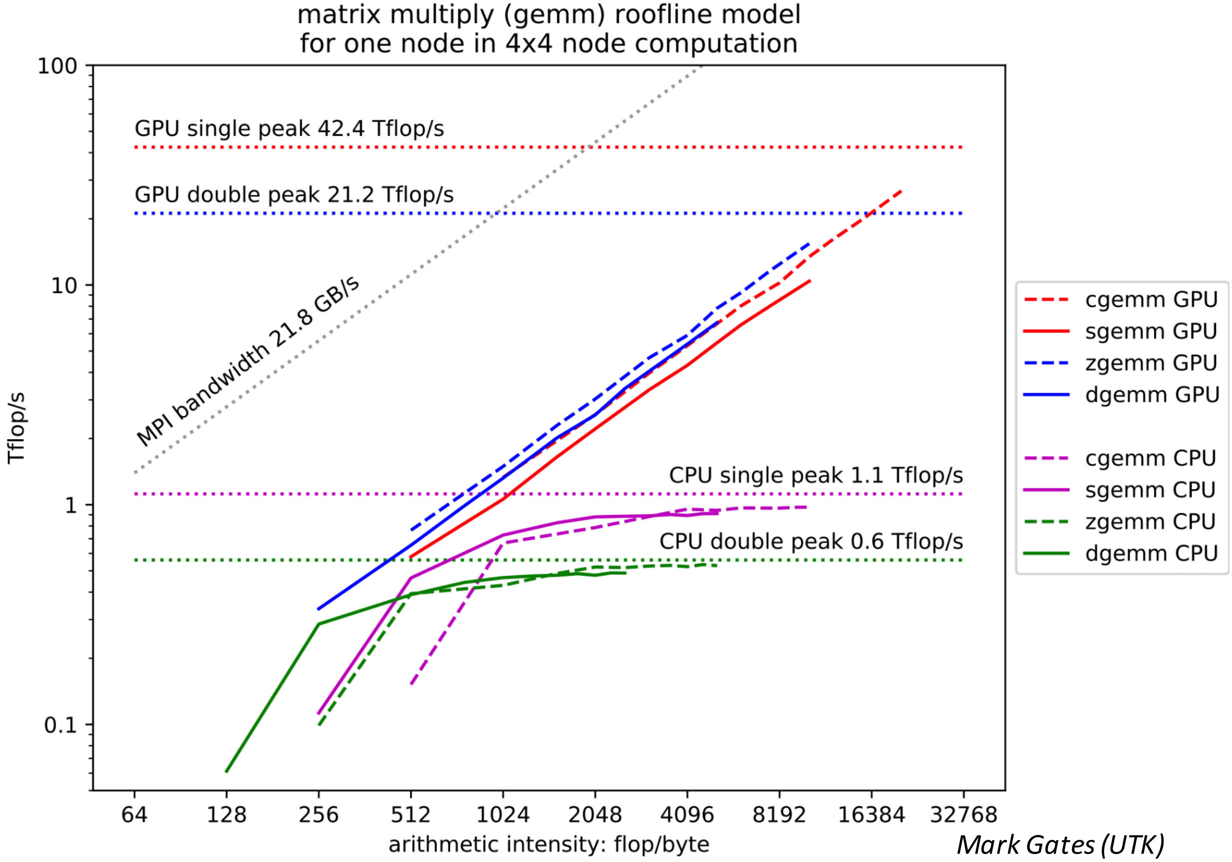
The Communication Bottleneck

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

Copyright@ORNL

Dense Matrix Operations?

- The inter-node communication is the limiting resource;
- Each node has more computational power than what we can leverage;



Mark Gates (UTK)

The Communication Bottleneck

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

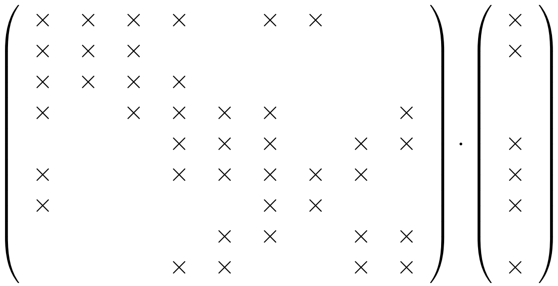
Copyright@ORNL

Dense Matrix Operations?

- The inter-node communication is the limiting resource;
- Each node has more computational power than what we can leverage;

Sparse / Graph Problems?

- ***S**parse **M**atrix **V**ector Product* (SpMV) is a central building block;



The Communication Bottleneck

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

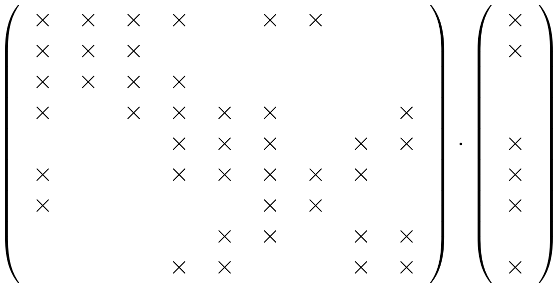
Copyright@ORNL

Dense Matrix Operations?

- The inter-node communication is the limiting resource;
- Each node has more computational power than what we can leverage;

Sparse / Graph Problems?

- ***S****parse* ***M****atrix* ***V****ector Product* (SpMV) is a central building block;
- For many of the problems in the SuiteSparse Matrix Collection[1], a Multi-node SpMV is slower than a Single-node SpMV;
- The inter-node communication is an order of magnitude slower than the local computations.



[1]SuiteSparse Matrix Collection: https://sparse.tamu.edu/

The Communication Bottleneck

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

summit

- Node: 2 IBM POWER9 + 6 NVIDIA V100 GPUs
- 4,608 nodes, 9,216 IBM Power9 CPUs
- 27,648 V100 GPUs (8 TFLOPs / GPU)
- Peak performance of **200 Pflop/s** for modeling & simulation
- Peak performance of **3.3 Eflop/s** (10^18) for 16 bit floating point used in data analytics and artificial intelligence

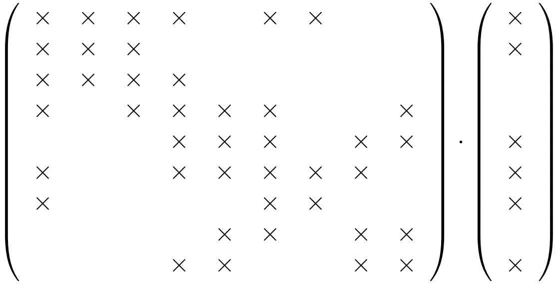
@ORNL

Dense Matrix Operations?

- The inter-node comm… …n is the limiting resource;
- Each node has m… …putational power than what we can leverage;

Sparse / Gr… …blems?

- *Spa… …ix Vector Product* (SpMV) is a central building block;
- …ny of the problems in the SuiteSparse Matrix Collection[1], …Multi-node SpMV is slower than a Single-node SpMV;
- The inter-node communication is an order of magnitude slower than the local computations.



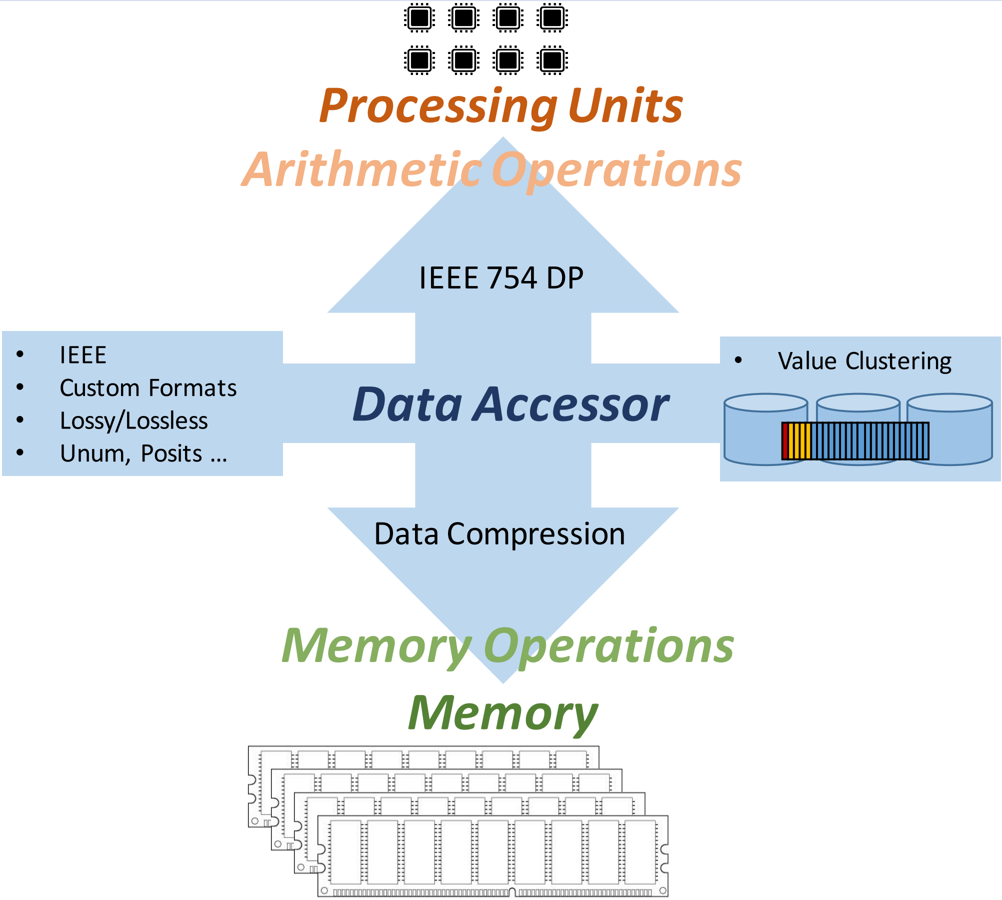
Radically decouple storage format from arithmetic format.

[1]SuiteSparse Matrix Collection: https://sparse.tamu.edu/

The Communication Bottleneck

Radically decouple storage format from arithmetic format.

- The arithmetic operations should use high precision formats natively supported by hardware.
- **Data access** should be as cheap as possible, **reduced precision**.
- Consider a wide range of memory formats:
  - IEEE standard precision formats
  - Customized formats (configuring mantissa/exponent)
  - Lossy compression
  - …



Copyright@ORNL

[1]SuiteSparse Matrix Collection: https://sparse.tamu.edu/

Spotlight Example: Use reduced precision for "approximate Operators"

- Solve sparse linear system $Ax = b$
- Preconditioners for iterative solvers.
  - Idea: Approximate inverse of system matrix to make the system "easier to solve": $P^{-1} \approx A^{-1}$

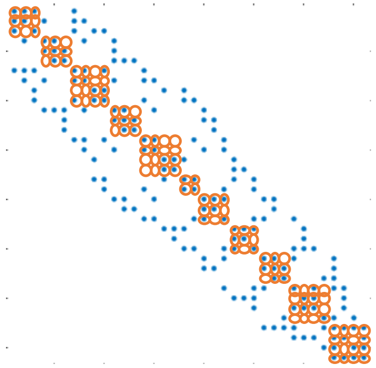
    $\tilde{A} = P^{-1}A$, $\tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.

Spotlight Example: Use reduced precision for "approximate Operators"

- **Solve sparse linear system** $Ax = b$
- **Preconditioners for iterative solvers.**
  - Idea: Approximate inverse of system matrix to make the system "easier to solve": $P^{-1} \approx A^{-1}$
    $\tilde{A} = P^{-1}A\,, \quad \tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.
- **Why should we store the preconditioner matrix P^{-1} in full (high) precision?**
  - **We have to ensure regularity!** (Reducing precision can turn matrix singular)

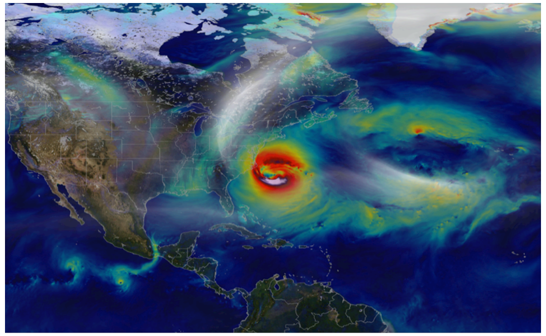
Spotlight Example: **Use reduced precision for "approximate Operators"**

- **Solve sparse linear system** $Ax = b$
- **Preconditioners for iterative solvers.**
  - Idea: Approximate inverse of system matrix to make the system "easier to solve": $P^{-1} \approx A^{-1}$
    $\tilde{A} = P^{-1}A, \quad \tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.
- **Why should we store the preconditioner matrix** P^{-1} **in full (high) precision?**
  - **We have to ensure regularity!** (Reducing precision can turn matrix singular)



- **Jacobi method** based on **diagonal scaling** $P = diag(A)$
- **Block-Jacobi** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Large set of small diagonal blocks.
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically **improve convergence**.
    - *Larger* blocks make block-Jacobi **more expensive**.

Extreme case: one block of matrix size.

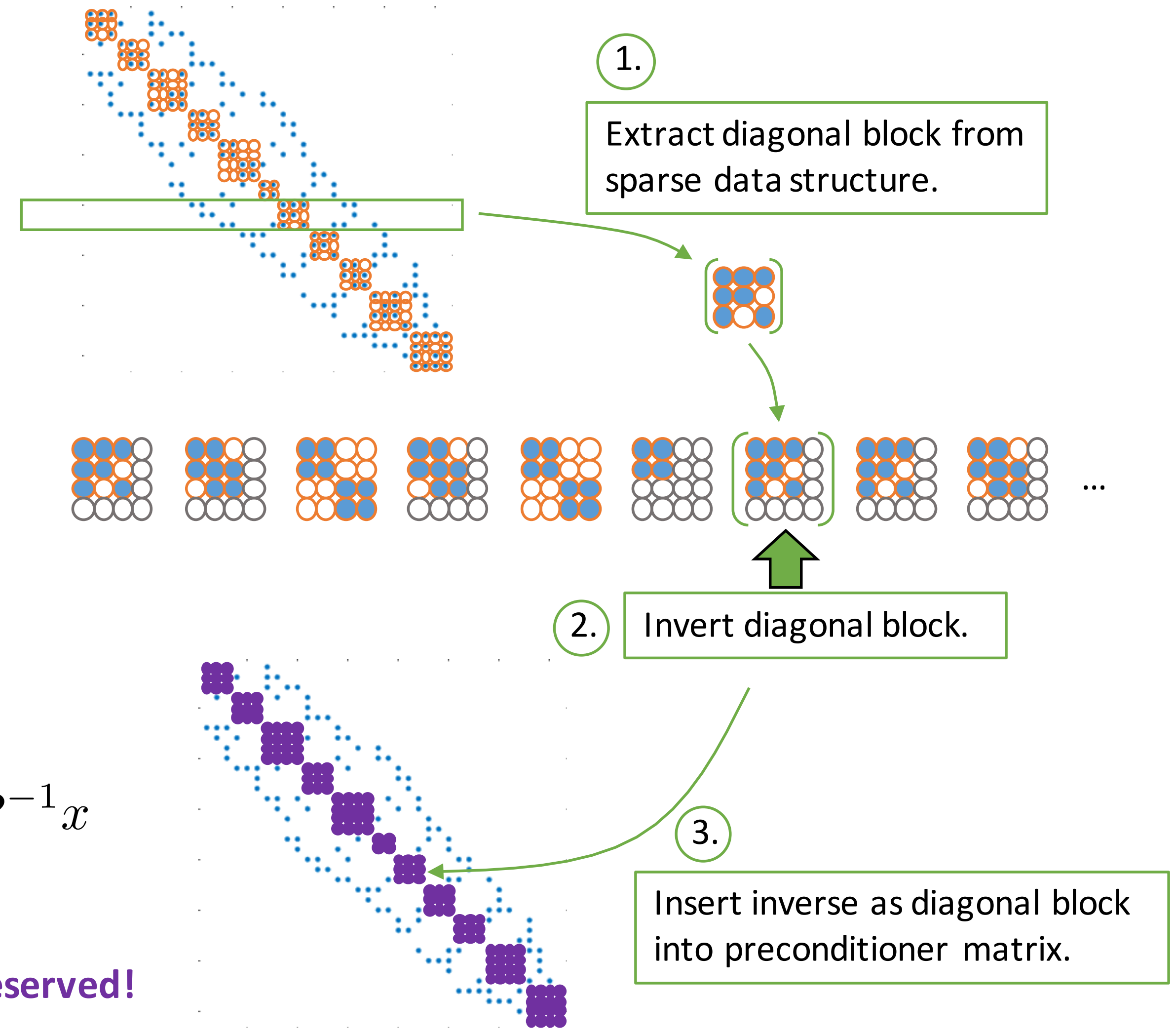


https://science.nasa.gov/earth-science/focus-areas/earth-weather

Spotlight Example: Block-Jacobi Preconditioning

Preconditioner Setup:

- Identify the diagonal blocks $P = diag_B(A)$
- Form the block-Inverse $P^{-1} \approx A^{-1}$



Preconditioner Application:

- Apply the preconditioner in every solver iteration via:

$$y := P^{-1}x$$

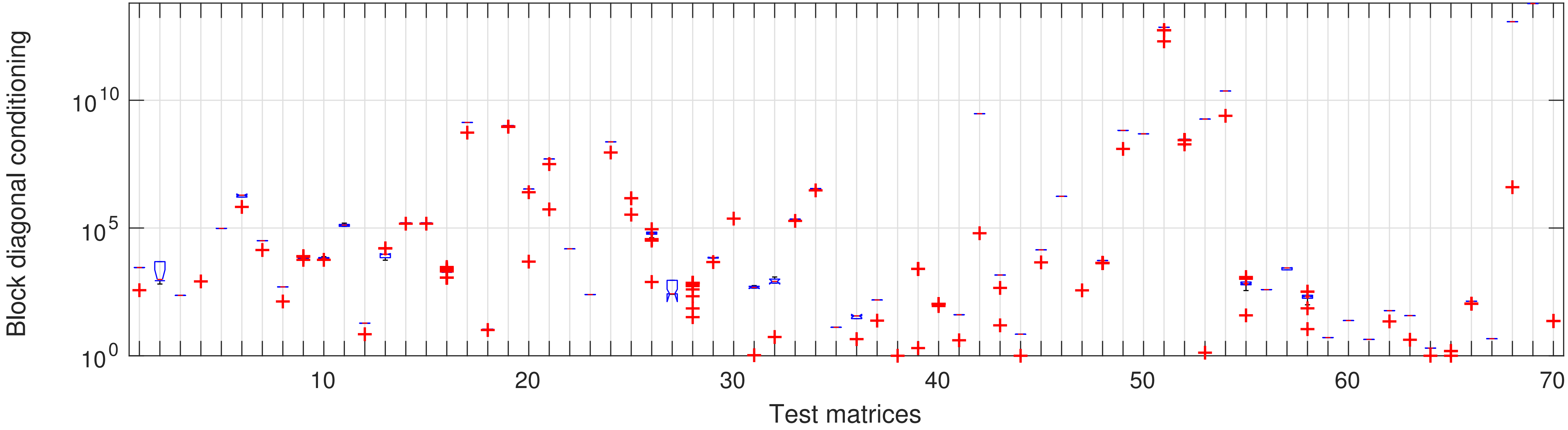
We can store diagonal blocks in lower precision, if regularity is preserved!

Adaptive Precision Block-Jacobi Preconditioning

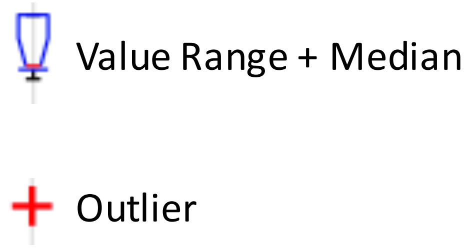
- 70 matrices from the SuiteSparse Matrix Collection
- Use block-size 24 with Super-Variable agglomeration (24 is upper bound for size of blocks)
- Report conditioning of all arising diagonal blocks

Value Range + Median

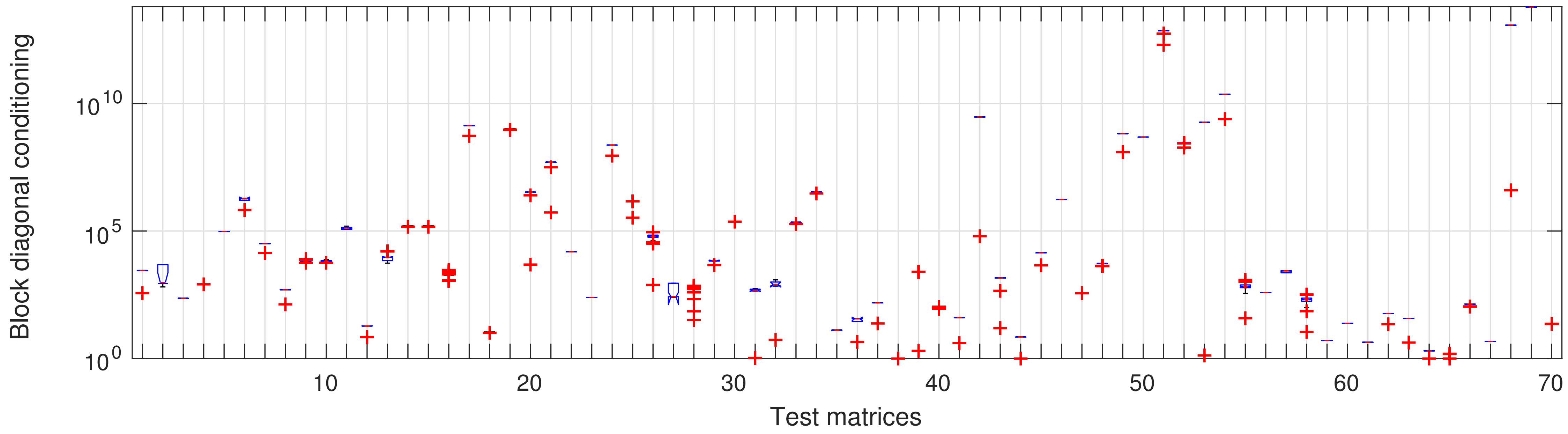
Outlier



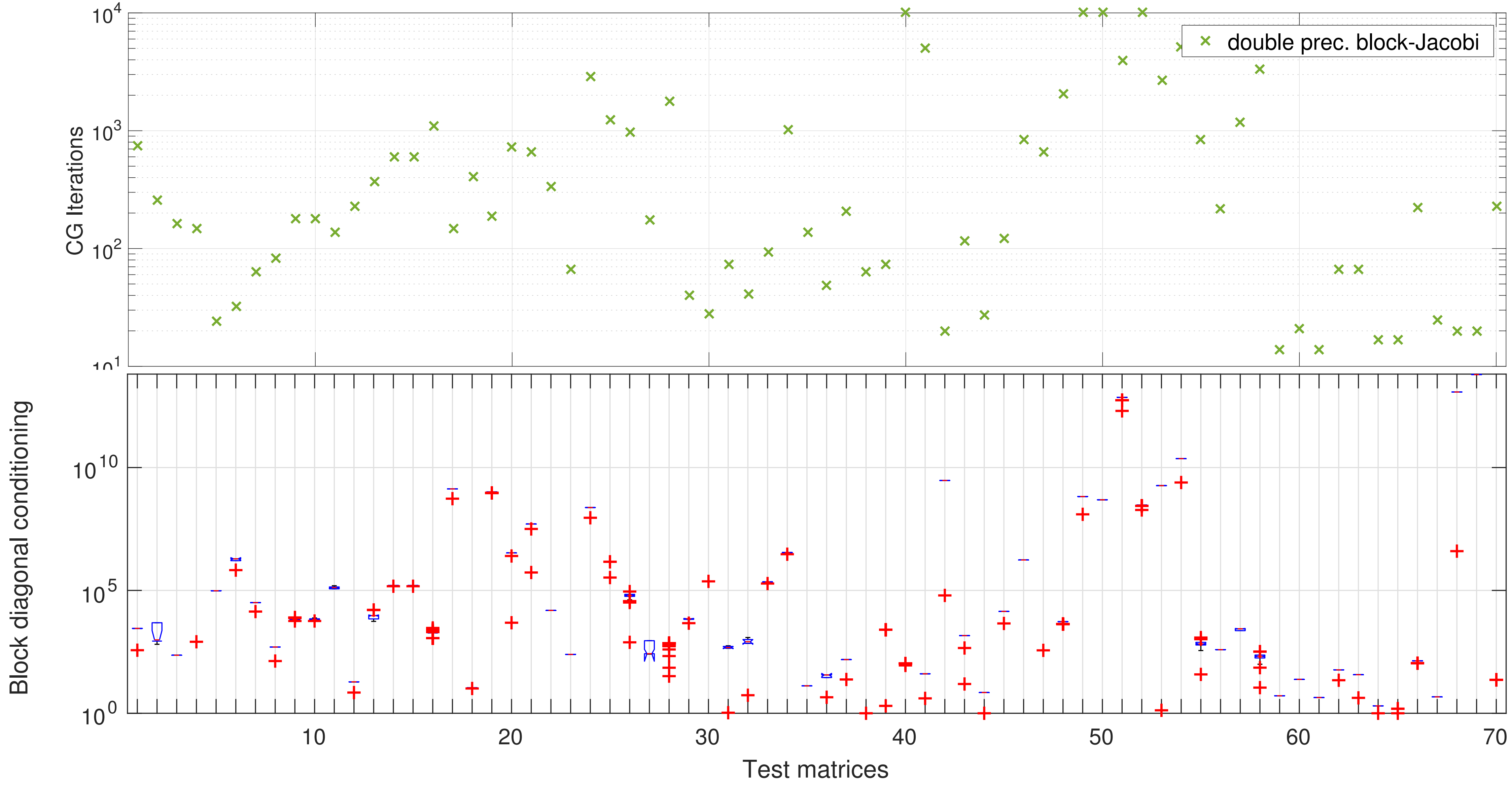
Adaptive Precision Block-Jacobi Preconditioning



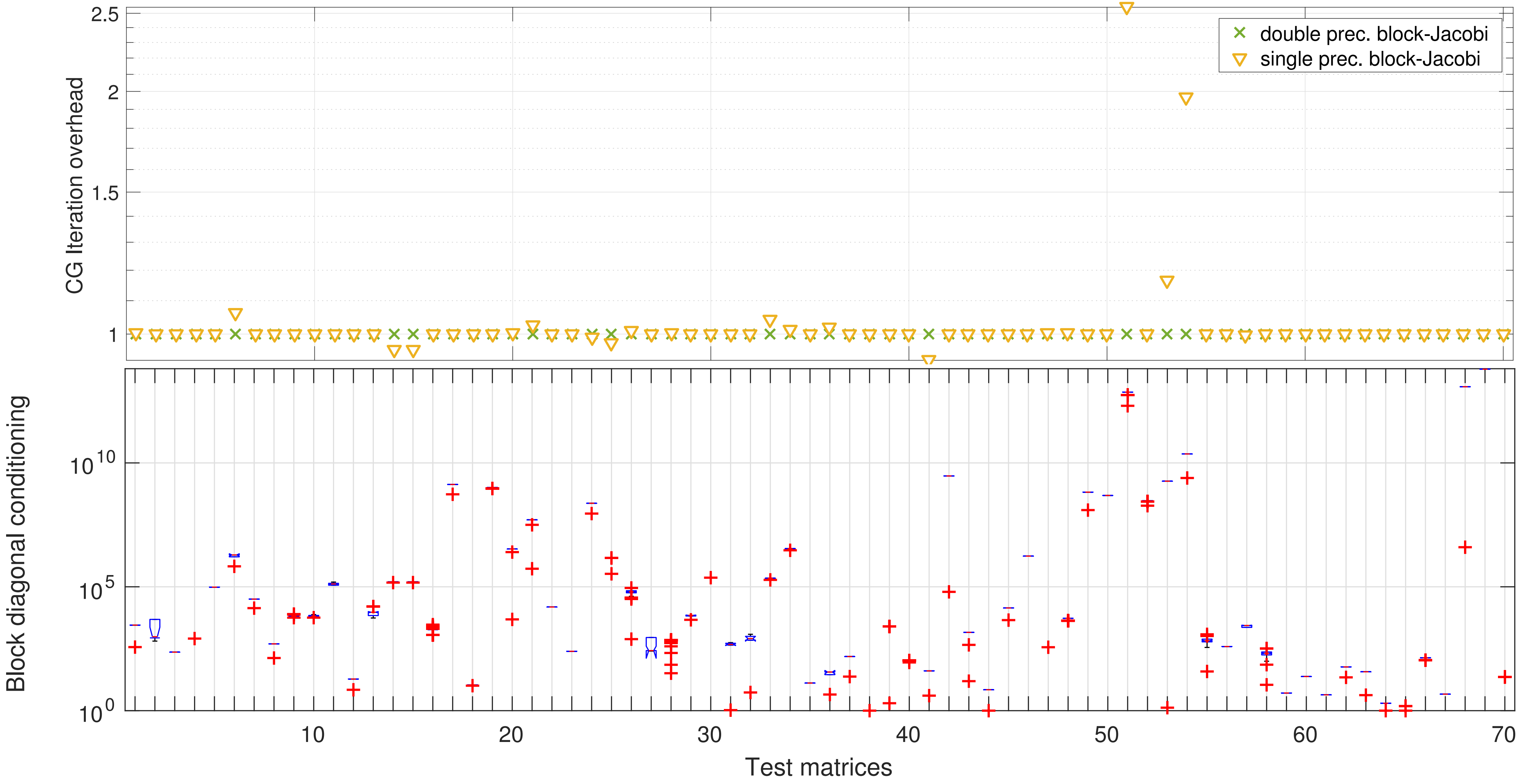
- 70 matrices from the SuiteSparse Matrix Collection
- Use block-size 24 with Super-Variable agglomeration (24 is upper bound for size of blocks)
- Report conditioning of all arising diagonal blocks
- Analyze the impact of storing block-Jacobi in lower precision a top-level Conjugate Gradient solver (CG)



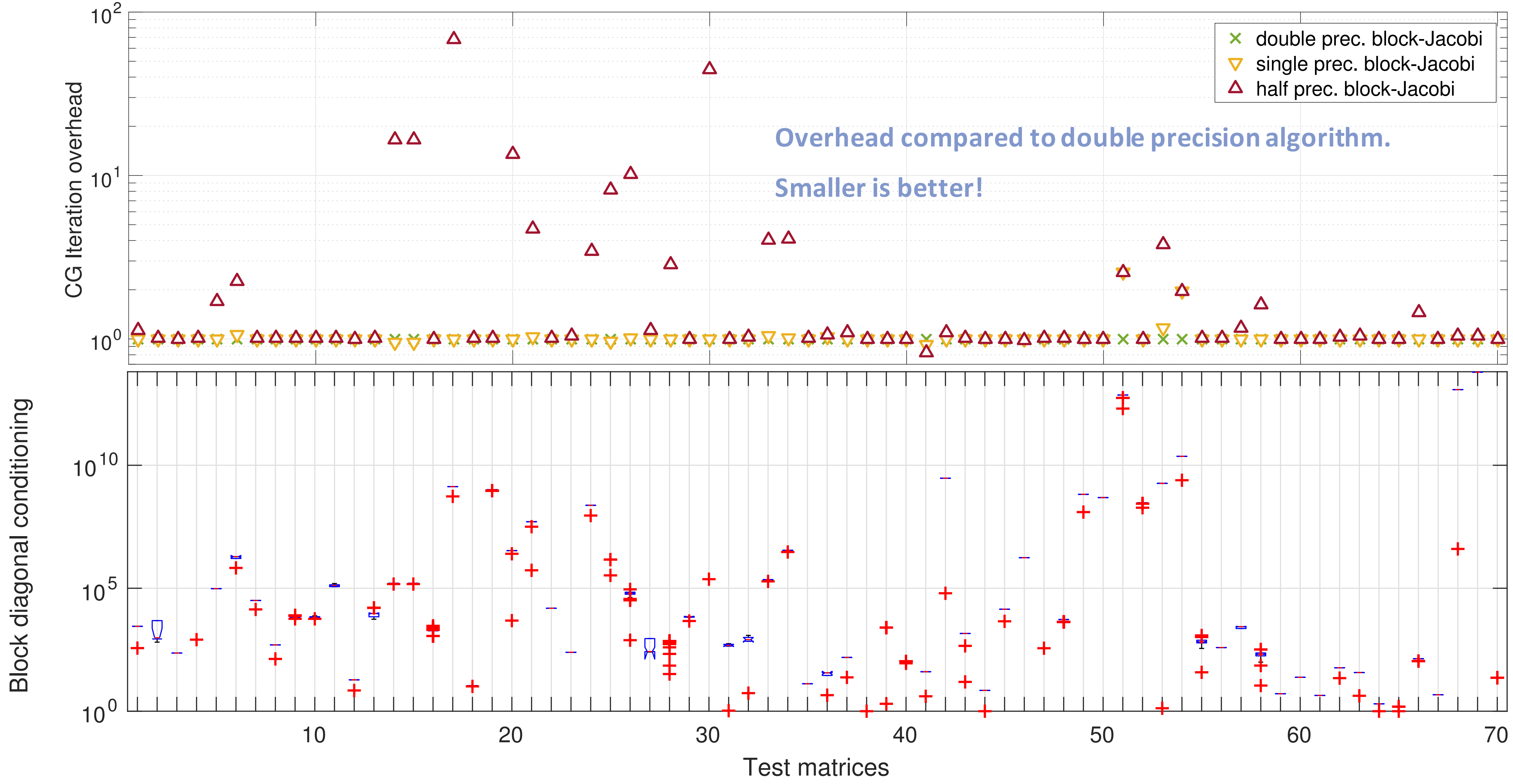
Adaptive Precision Block-Jacobi Preconditioning



Adaptive Precision Block-Jacobi Preconditioning



Adaptive Precision Block-Jacobi Preconditioning

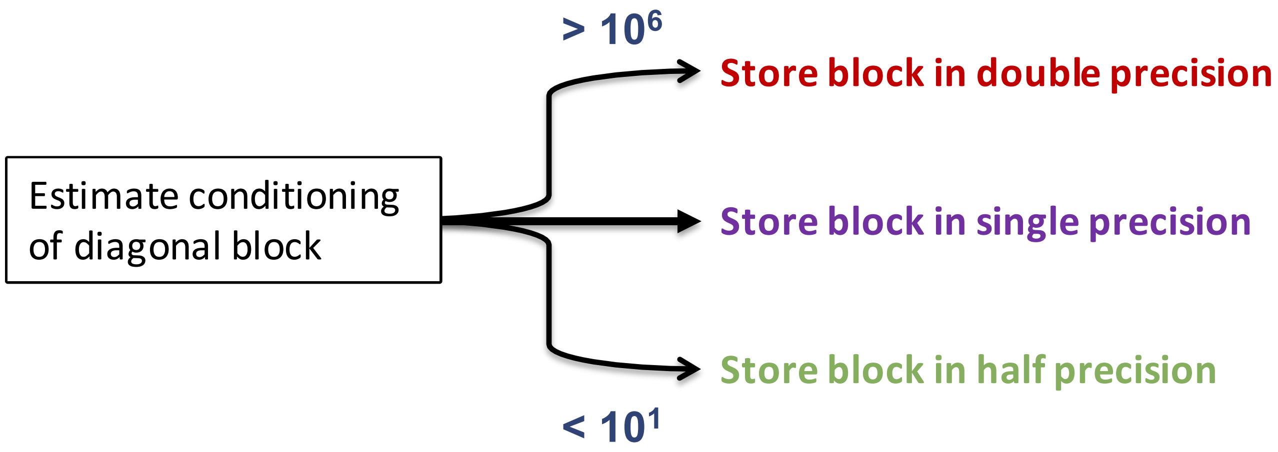


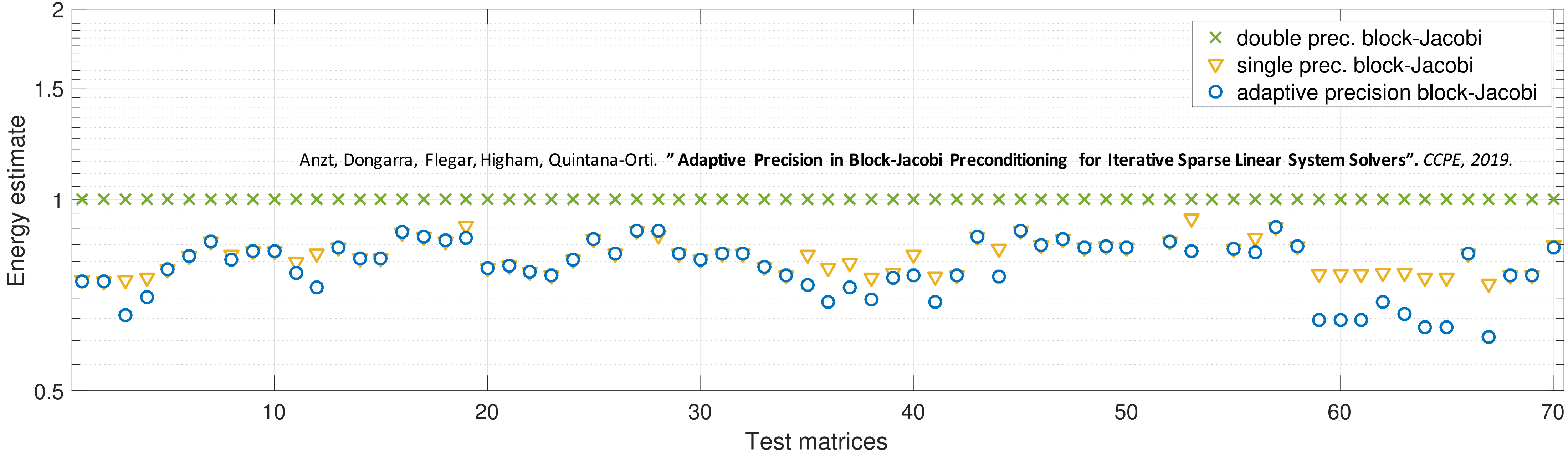
Adaptive Precision Block-Jacobi Preconditioning

Multi-Precision Idea:

- **All computations use double precision!**

- Store distinct blocks in different formats
- Use **single precision as standard** storage format
- Where **necessary:** switch to **double**
- For well-conditioned blocks use **half precision**

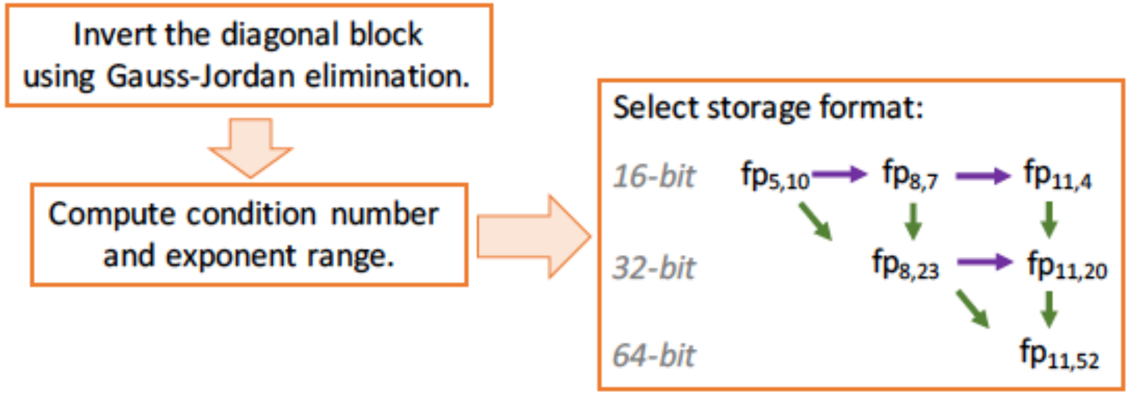




Adaptive Precision Block-Jacobi Preconditioning

Multi-Precision Idea:

- **All computations use double precision!**
- **Depart from the rigid IEEE precision formats!**
- Preserve either 1 or 2 digits accuracy of the inverted diagonal blocks.

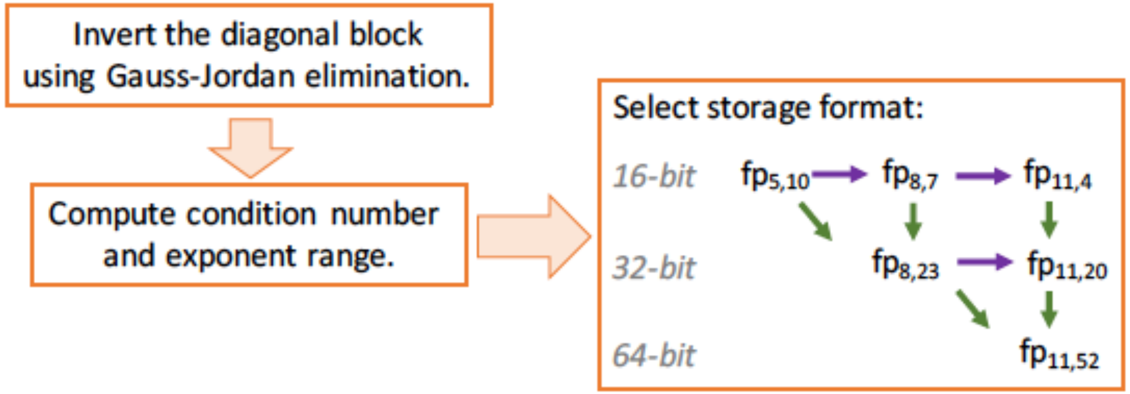


Flegar, Anzt, Quintana-Orti. **"Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors".** *TOMS, submitted.*

Adaptive Precision Block-Jacobi Preconditioning

Multi-Precision Idea:

- **All computations use double precision!**
- **Depart from the rigid IEEE precision formats!**
- Preserve either 1 or 2 digits accuracy of the inverted diagonal blocks.

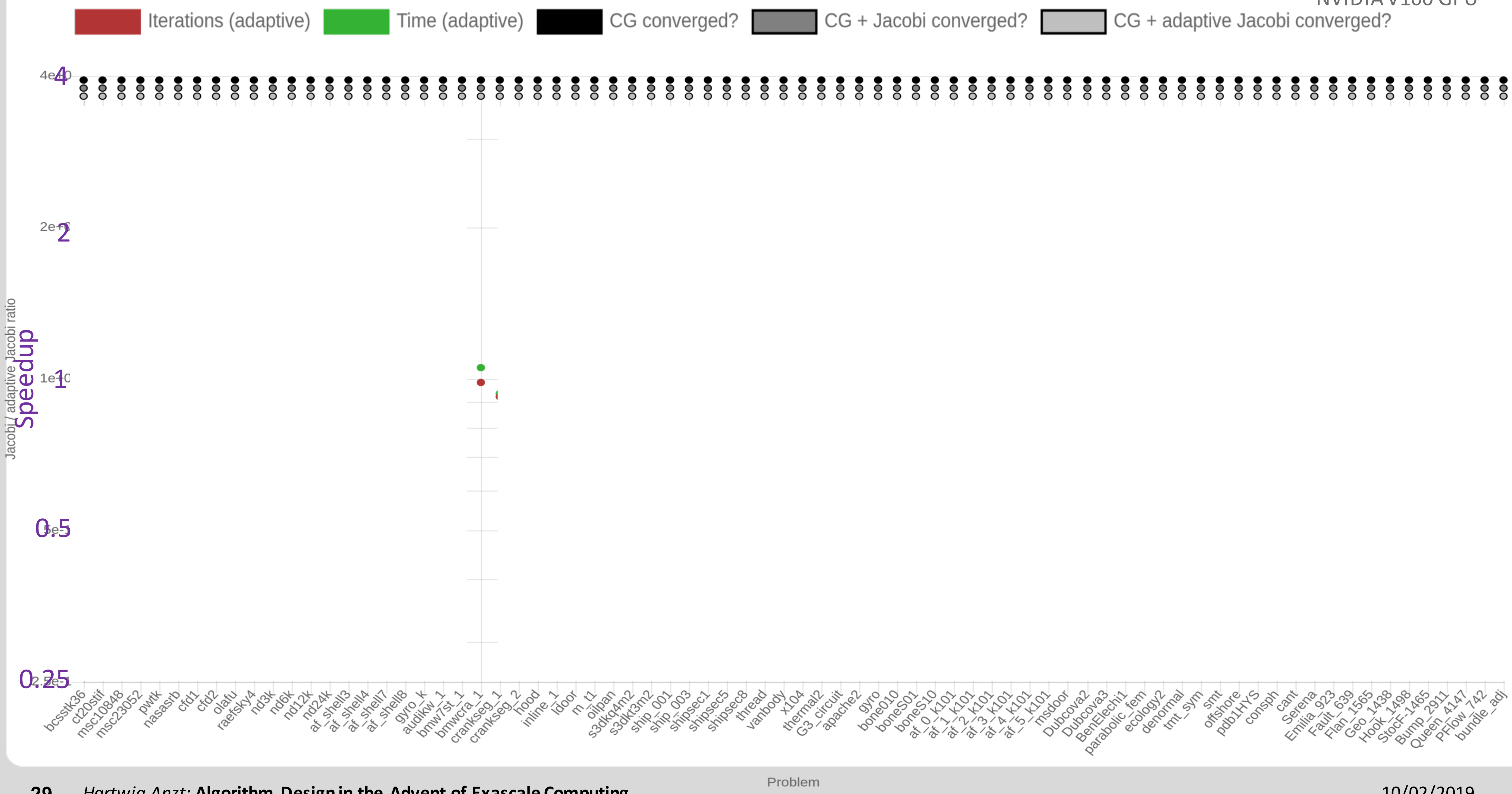


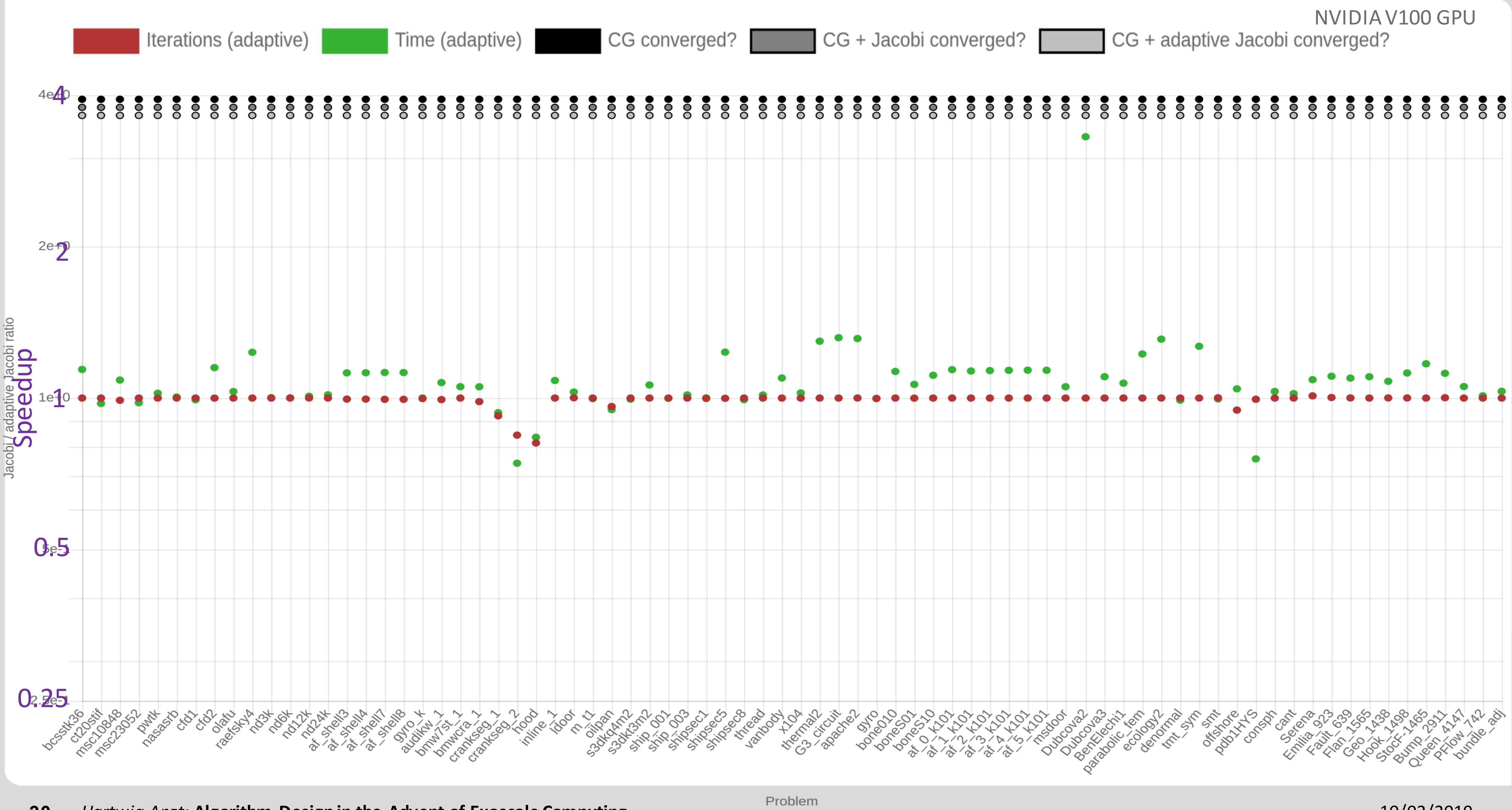
- ✓ Regularity preserved;
- ✓ No flexible Krylov solver needed (Preconditioner constant operator);
- ✓ Can handle non-spd problems (inversion features pivoting);
- ✓ Preconditioner for any iterative preconditionable solver;

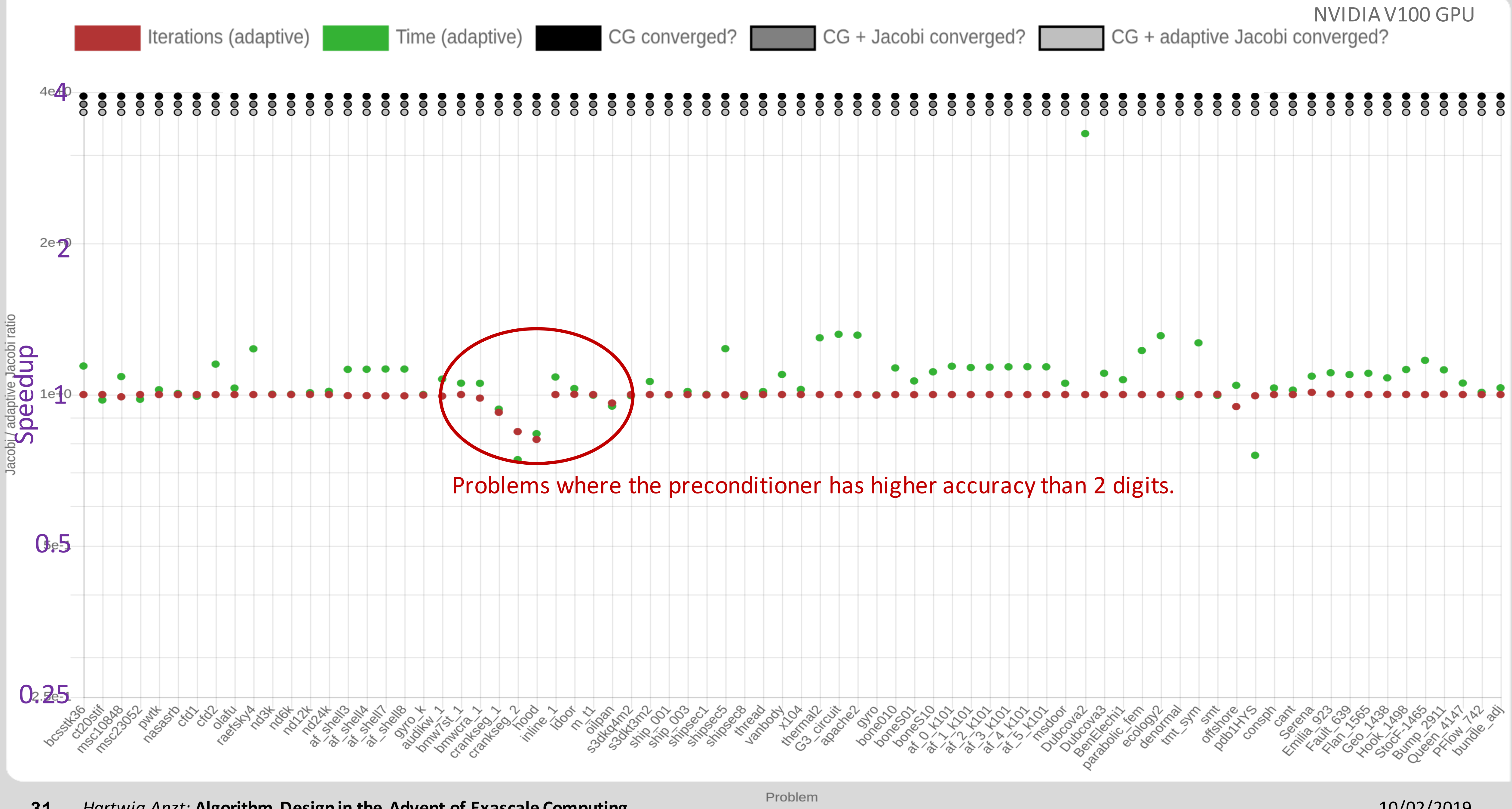
- **Overhead** of the **precision detection** (condition number calculation);
- **Overhead** from storing **precision information** (need to additionally store/retrieve flag);
- Speedups / preconditioner quality **problem-dependent**;

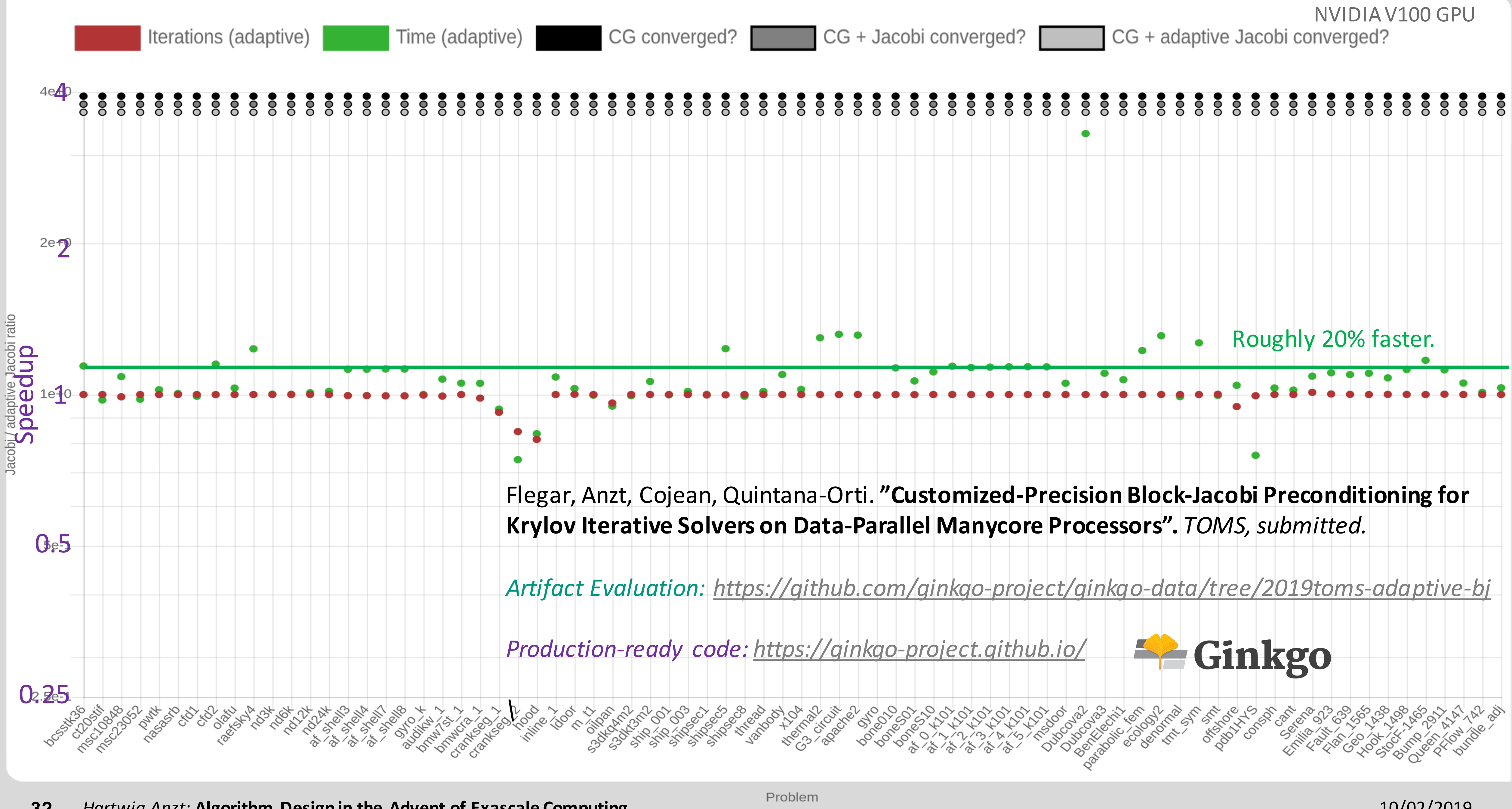

NVIDIA V100 GPU
Iterations (adaptive)
Time (adaptive)
CG converged?
CG + Jacobi converged?
CG + adaptive Jacobi converged?
Speedup
Jacobi / adaptive Jacobi ratio
4
2
1
0.5
0.25
bcsstk36 ct20stif msc10848 msc23052 pwtk nasasrb cfd1 cfd2 olafu raefsky4 nd3k nd6k nd12k nd24k af_shell3 af_shell4 af_shell7 af_shell8 gyro_k audikw_1 bmw7st_1 bmwcra_1 crankseg_1 crankseg_2 hood inline_1 ldoor m_t1 oilpan s3dkq4m2 s3dkt3m2 ship_001 ship_003 shipsec1 shipsec5 shipsec8 thread vanbody x104 thermal2 G3_circuit apache2 gyro bone010 boneS01 boneS10 af_0_k101 af_1_k101 af_2_k101 af_3_k101 af_4_k101 af_5_k101 msdoor Dubcova2 Dubcova3 BenElechi1 parabolic_fem ecology2 denormal tmt_sym smt offshore pdb1HYS consph cant Serena Emilia_923 Fault_639 Flan_1565 Geo_1438 Hook_1498 StocF-1465 Bump_2911 Queen_4147 PFlow_742 bundle_adj
Problem

NVIDIA V100 GPU




NVIDIA V100 GPU
Iterations (adaptive)
Time (adaptive)
CG converged?
CG + Jacobi converged?
CG + adaptive Jacobi converged?
Speedup
Jacobi / adaptive Jacobi ratio
4
2
1
0.5
0.25
Problem
bcsstk36
ct20stif
msc10848
msc23052
pwtk
nasasrb
cfd1
cfd2
olafu
raefsky4
nd3k
nd6k
nd12k
nd24k
af_shell3
af_shell4
af_shell7
af_shell8
gyro_k
audikw_1
bmw7st_1
bmwcra_1
crankseg_1
crankseg_2
hood
inline_1
ldoor
m_t1
oilpan
s3dkq4m2
s3dkt3m2
ship_001
ship_003
shipsec1
shipsec5
shipsec8
thread
vanbody
x104
thermal2
G3_circuit
apache2
gyro
bone010
boneS01
boneS10
af_0_k101
af_1_k101
af_2_k101
af_3_k101
af_4_k101
af_5_k101
msdoor
Dubcova2
Dubcova3
BenElechi1
parabolic_fem
ecology2
denormal
tmt_sym
smt
offshore
pdb1HYS
consph
cant
Serena
Emilia_923
Fault_639
Flan_1565
Geo_1438
Hook_1498
StocF-1465
Bump_2911
Queen_4147
PFlow_742
bundle_adj


NVIDIA V100 GPU
Iterations (adaptive)
Time (adaptive)
CG converged?
CG + Jacobi converged?
CG + adaptive Jacobi converged?
Speedup
Jacobi / adaptive Jacobi ratio
4
2
1
0.5
0.25
Problems where the preconditioner has higher accuracy than 2 digits.
Problem
bcsstk36
ct20stif
msc10848
msc23052
pwtk
nasasrb
cfd1
cfd2
olafu
raefsky4
nd3k
nd6k
nd12k
nd24k
af_shell3
af_shell4
af_shell7
af_shell8
gyro_k
audikw_1
bmw7st_1
bmwcra_1
crankseg_1
crankseg_2
hood
inline_1
ldoor
m_t1
oilpan
s3dkq4m2
s3dkt3m2
ship_001
ship_003
shipsec1
shipsec5
shipsec8
thread
vanbody
x104
thermal2
G3_circuit
apache2
gyro
bone010
boneS01
boneS10
af_0_k101
af_1_k101
af_2_k101
af_3_k101
af_4_k101
af_5_k101
msdoor
Dubcova2
Dubcova3
BenElechi1
parabolic_fem
ecology2
denormal
tmt_sym
smt
offshore
pdb1HYS
consph
cant
Serena
Emilia_923
Fault_639
Flan_1565
Geo_1438
Hook_1498
StocF-1465
Bump_2911
Queen_4147
PFlow_742
bundle_adj



Flegar, Anzt, Cojean, Quintana-Orti. "**Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors**". *TOMS, submitted.*

Artifact Evaluation: *https://github.com/ginkgo-project/ginkgo-data/tree/2019toms-adaptive-bj*

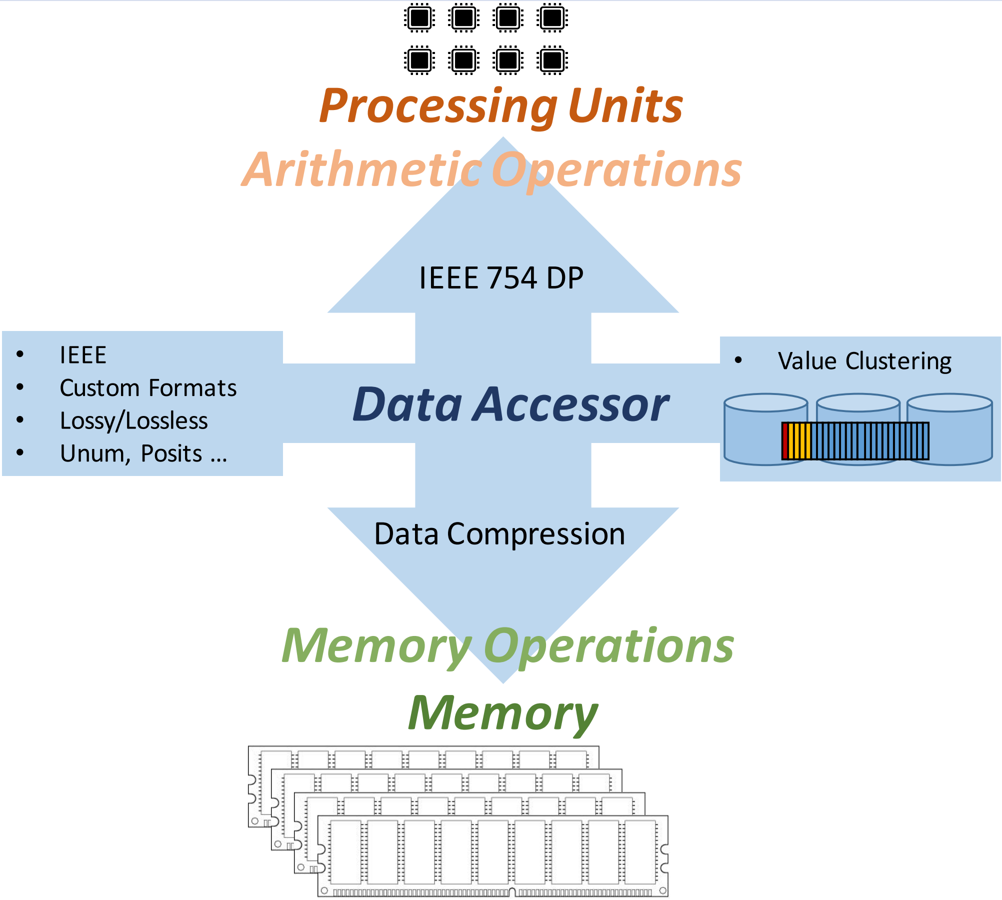
Production-ready code: *https://ginkgo-project.github.io/*

Ginkgo

The Communication Bottleneck

Radically decouple storage format from arithmetic format.

- The arithmetic operations should use high precision formats natively supported by hardware.
- **Data access** should be as cheap as possible, **reduced precision**.
- Consider a wide range of memory formats:
  - IEEE standard precision formats
  - Customized formats (configuring mantissa/exponent)
  - Lossy compression
  - …



Copyright@ORNL

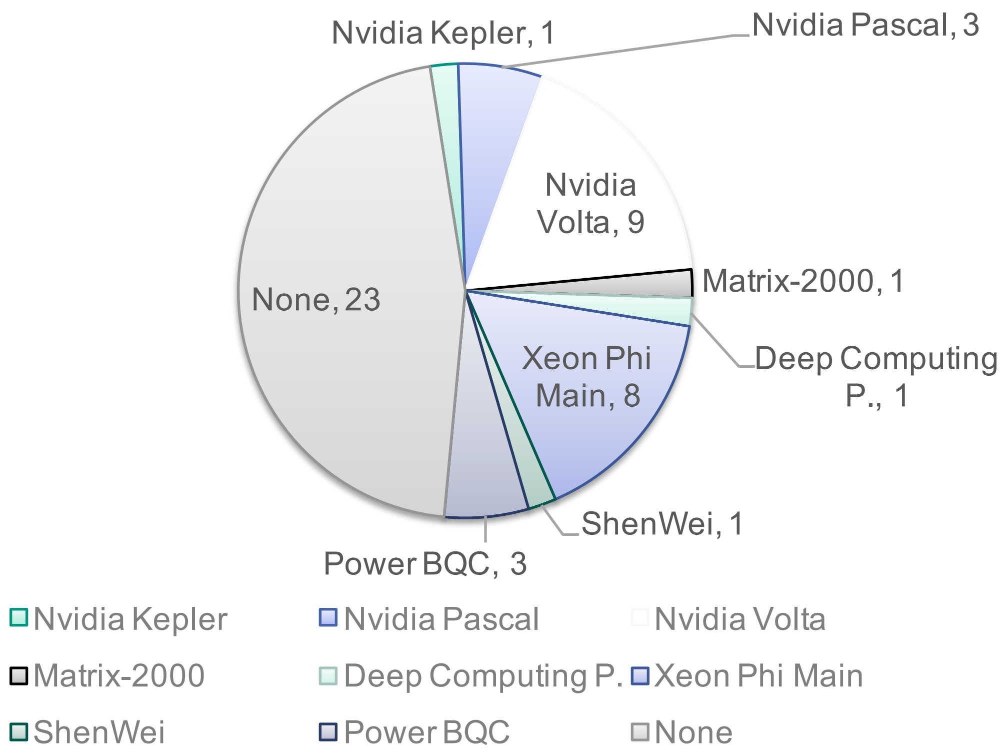
[1]SuiteSparse Matrix Collection: https://sparse.tamu.edu/

How to deal with the Manycore Parallelism?

- Increasing adoption of manycore accelerators
  -- partly motivated by the Machine Learning excitement;
- Integration of low-precision tensor units;
- The GPU streaming model is dominating;

- Algorithms need fine-grained parallelism
  -- thousands of SIMT threads!
- Global synchronizations are killing performance;
- Runtime scheduling of thread blocks virtually impossible;
- Memory access pattern central (coalesced data access);
- Asynchronous algorithms needed;
- Reformulation as fixed-point iteration;

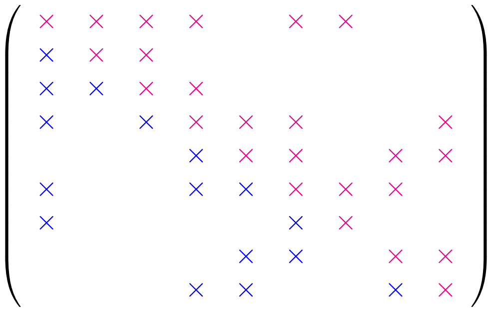
Accelerator share in the TOP50 systems [Jun 2019]



Jack Dongarra (UTK)

Spotlight Example: Incomplete Sparse Factorizations

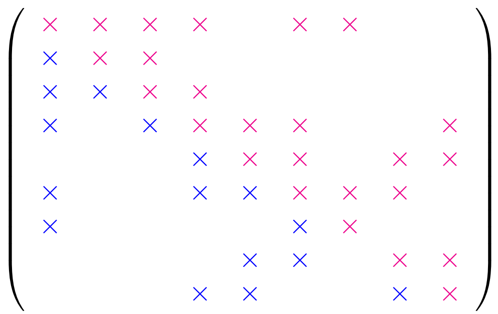
We are looking for a factorization-based preconditioner such that $A \approx L \cdot U$.
is a good approximation with moderate nonzero count (e.g. $nnz(L+U) = nnz(A)$).



Spotlight Example: Incomplete Sparse Factorizations

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?*

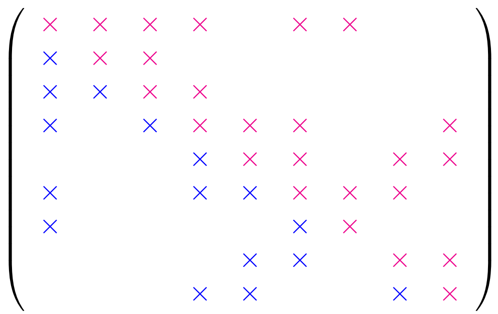


$$\mathcal{S}(A) = \{(i,j) \in \mathbb{N}^2 : A_{ij} \neq 0\}$$

Spotlight Example: Incomplete Sparse Factorizations

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?*



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

Exact LU Factorization

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

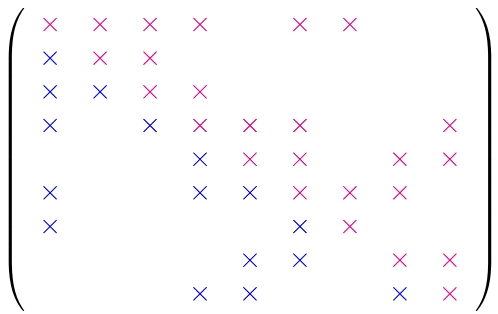
$$Ly = b \Rightarrow y \qquad \Rightarrow \qquad Ux = y \Rightarrow x$$

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

Spotlight Example: Incomplete Sparse Factorizations

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…*

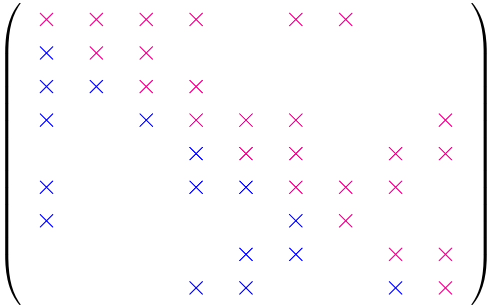
Incomplete LU Factorization (ILU)

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

Spotlight Example: Incomplete Sparse Factorizations

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)

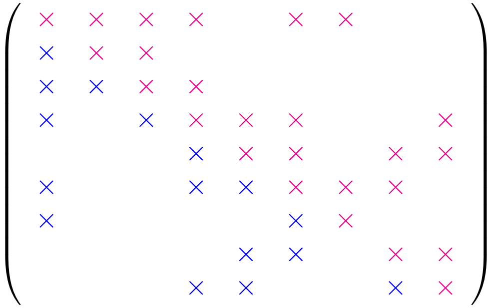
- Focused on restricting fill-in to a specific sparsity pattern $\mathcal{S}$.

- 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?*

Spotlight Example: Incomplete Sparse Factorizations

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…*

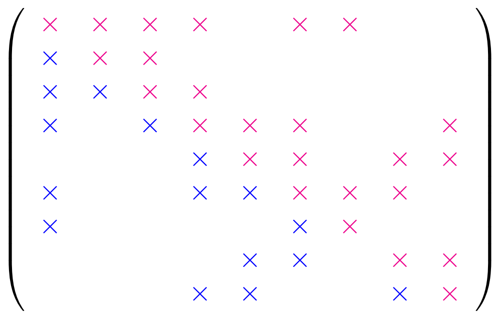
Incomplete LU Factorization (ILU)

- 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.

Spotlight Example: Incomplete Sparse Factorizations

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?*



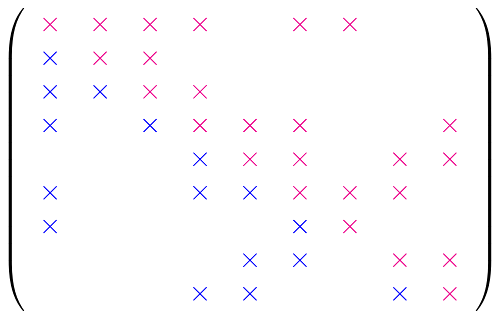
Rethink the overall strategy!

- 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.*

Spotlight Example: Incomplete Sparse Factorizations

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…

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

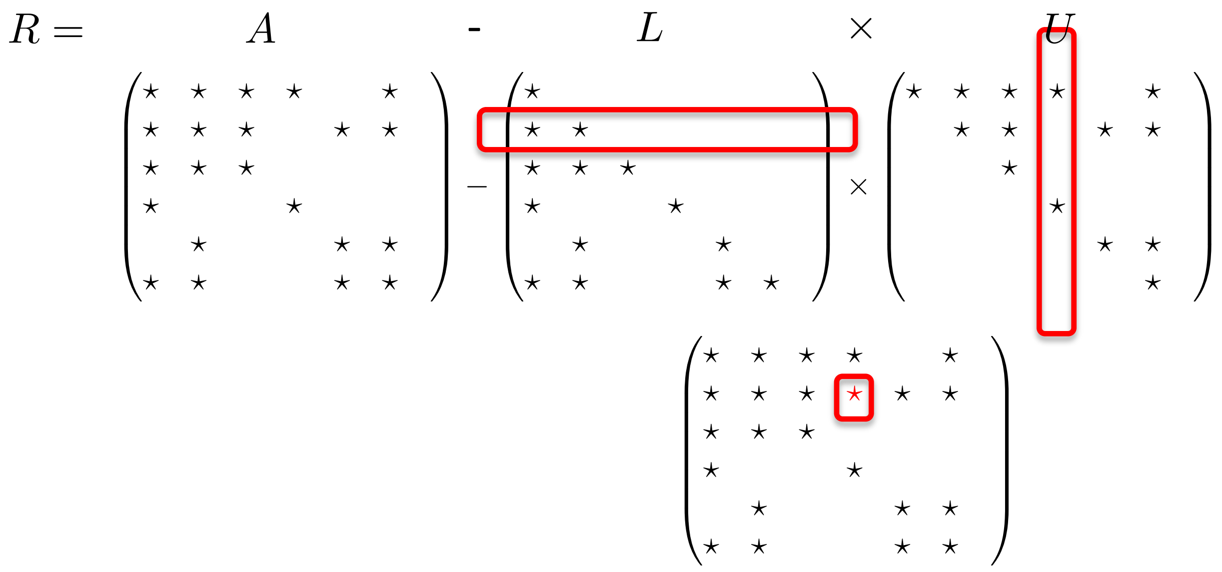
$$\begin{pmatrix} \star & \star & \star & \star & & \star \\ \star & \star & \star & & \star & \star \\ \star & \star & \star & & & \\ \star & & & \star & & \\ & \star & & & \star & \star \\ \star & \star & & & \star & \star \end{pmatrix} - \begin{pmatrix} \star & & & & & \\ \star & \star & & & & \\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

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…

ILU residual

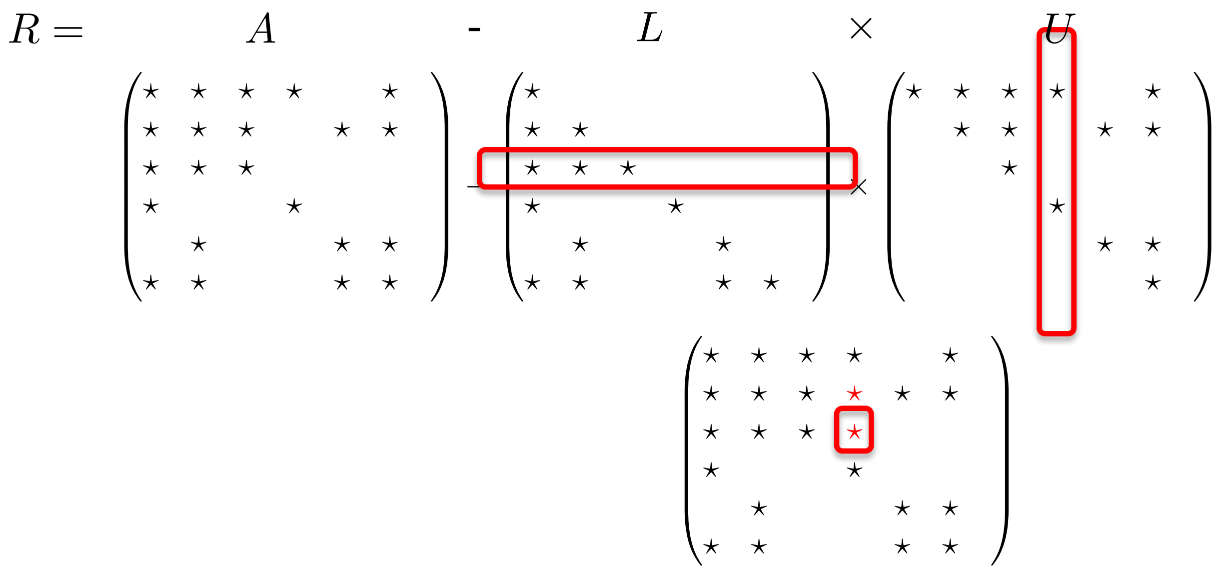


Considerations

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…

ILU 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…

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

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

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

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…

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

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

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.

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

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

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

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

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]:

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

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

ParILU Algorithm

- *Fixed-Point based algorithm for computing ILU;*
- *Fine-grained parallelism and asynchronous execution;*
- *Faster than Level-Scheduling*
- *Outperforms NVIDIA's cuSPARSE ILU*

Matrix (UFMC)	NVIDIA cuSPARSE	ParILU	Speedup
APA	61. ms	8.8 ms	**6.9**
ECO	107.ms	6.7 ms	**16.0**
G3	110. ms	12.1 ms	**9.1**
OFF	219. ms	25.1 ms	**8.7**
PAR	131. ms	6.1 ms	**21.6**
THM	454. ms	15.7 ms	**28.9**
L2D	112. ms	7.4 ms	**15.2**
L3D	94. ms	47.5 ms	**2.0**

Chow, Anzt, Dongarra, ISC 2015

[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]:

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

- 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.***

- Maybe use the ILU residual norm as quality metric.

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

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

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.

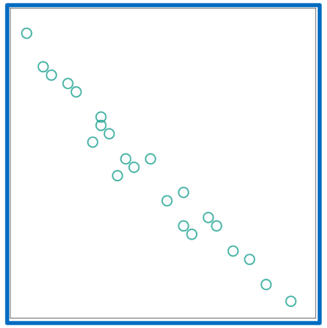
Fixed-point sweep approximates incomplete factors.

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].

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

Fixed-point sweep approximates incomplete factors.

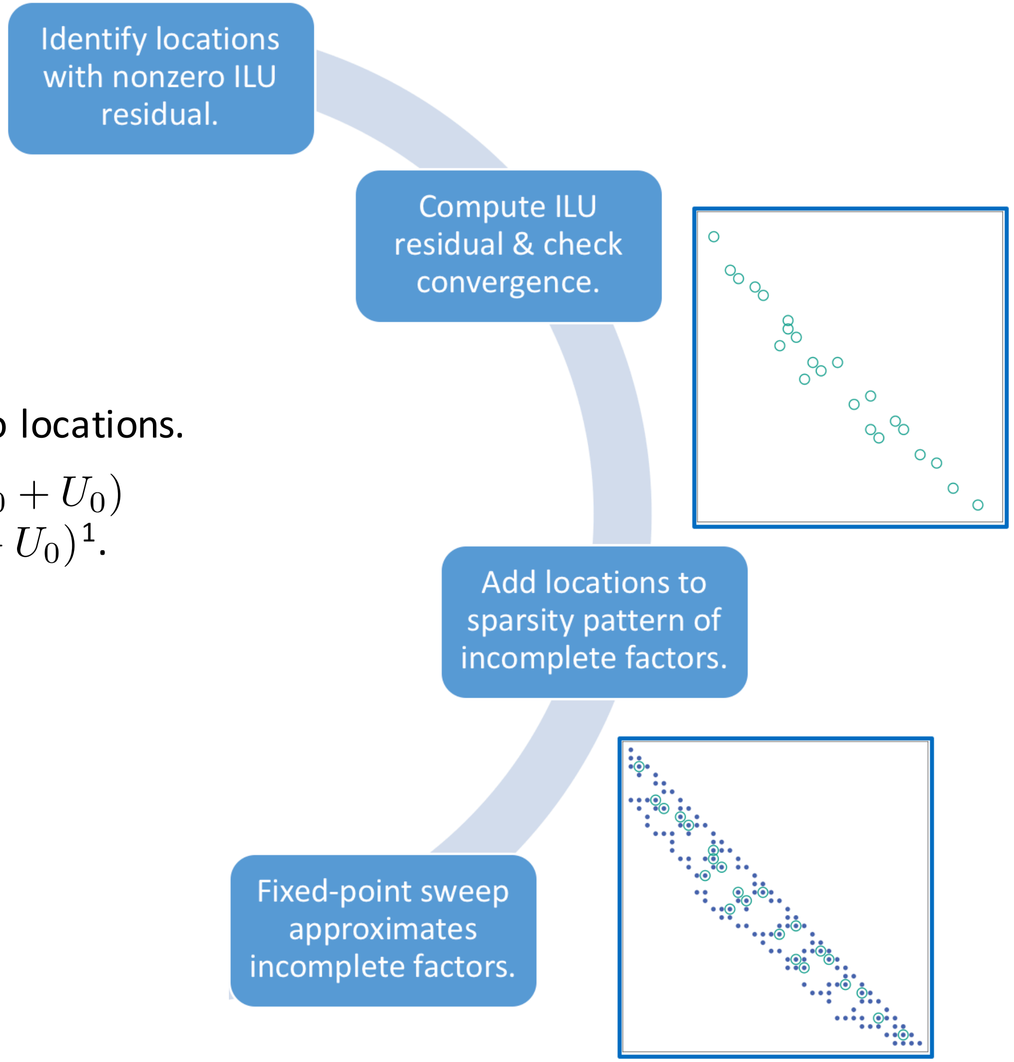
[1] Saad. **"Iterative Methods for Sparse Linear Systems, 2nd Edition"**. *(2003).*

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.*

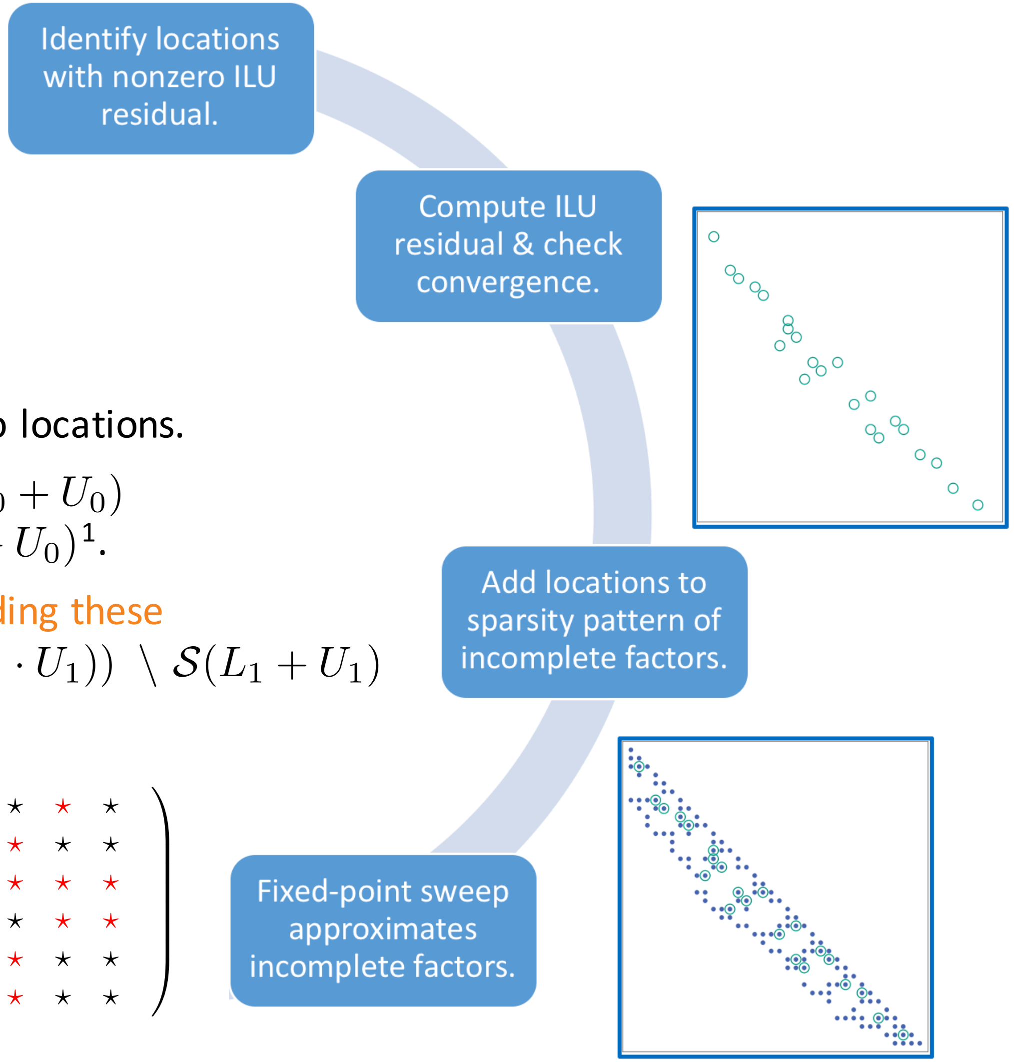
- The sparsity pattern of A might be a good initial start for nonzero locations.
- Then, the approximation will be exact for all locations $\mathcal{S}_0 = \mathcal{S}(L_0 + U_0)$ and nonzero in locations $\mathcal{S}_1 = (\mathcal{S}(A) \, \cup \, \mathcal{S}(L_0 \cdot U_0)) \setminus \mathcal{S}(L_0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

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

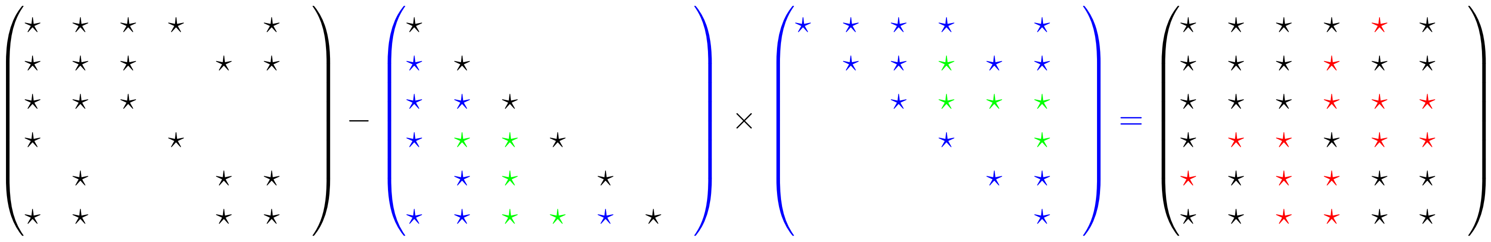


Considerations

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.*



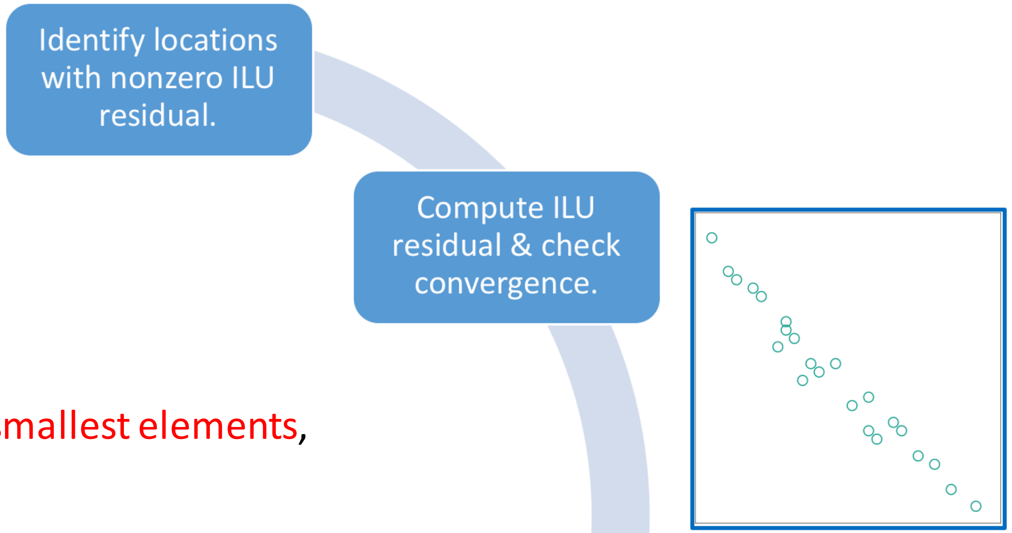
- The sparsity pattern of A might be a good initial start for nonzero locations.
- Then, the approximation will be exact for all locations $\mathcal{S}_0 = \mathcal{S}(L_0 + U_0)$ and nonzero in locations $\mathcal{S}_1 = (\mathcal{S}(A) \cup \mathcal{S}(L_0 \cdot U_0)) \setminus \mathcal{S}(L_0 + U_0)^1$.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

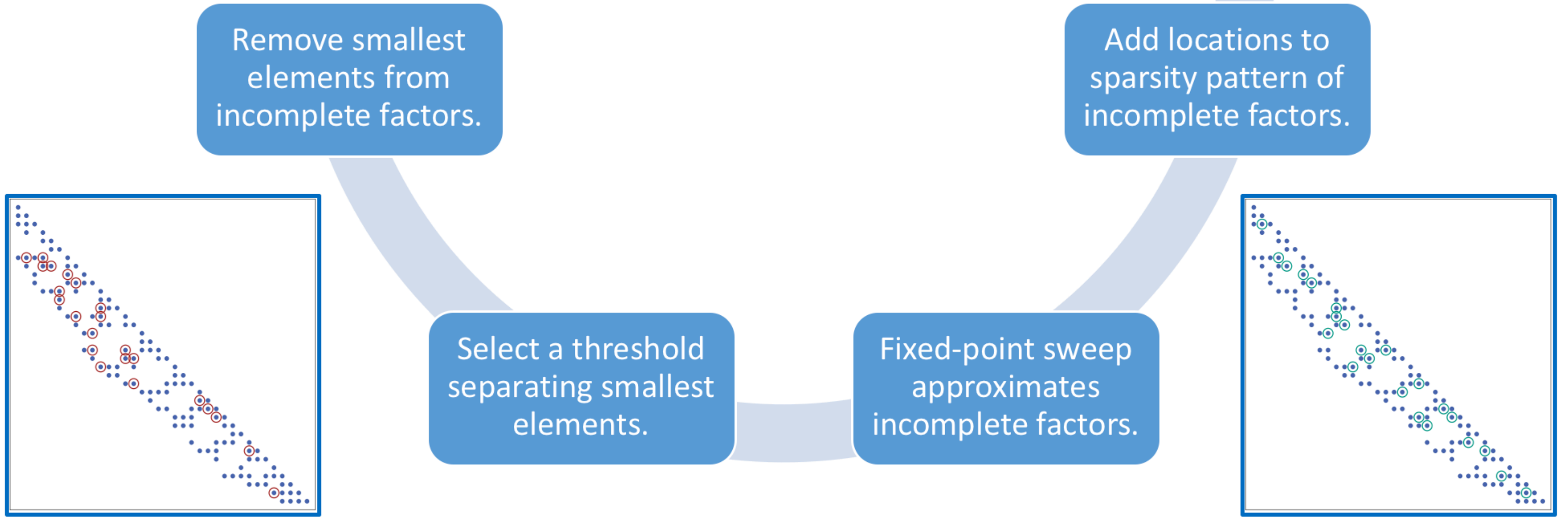


Considerations

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.*

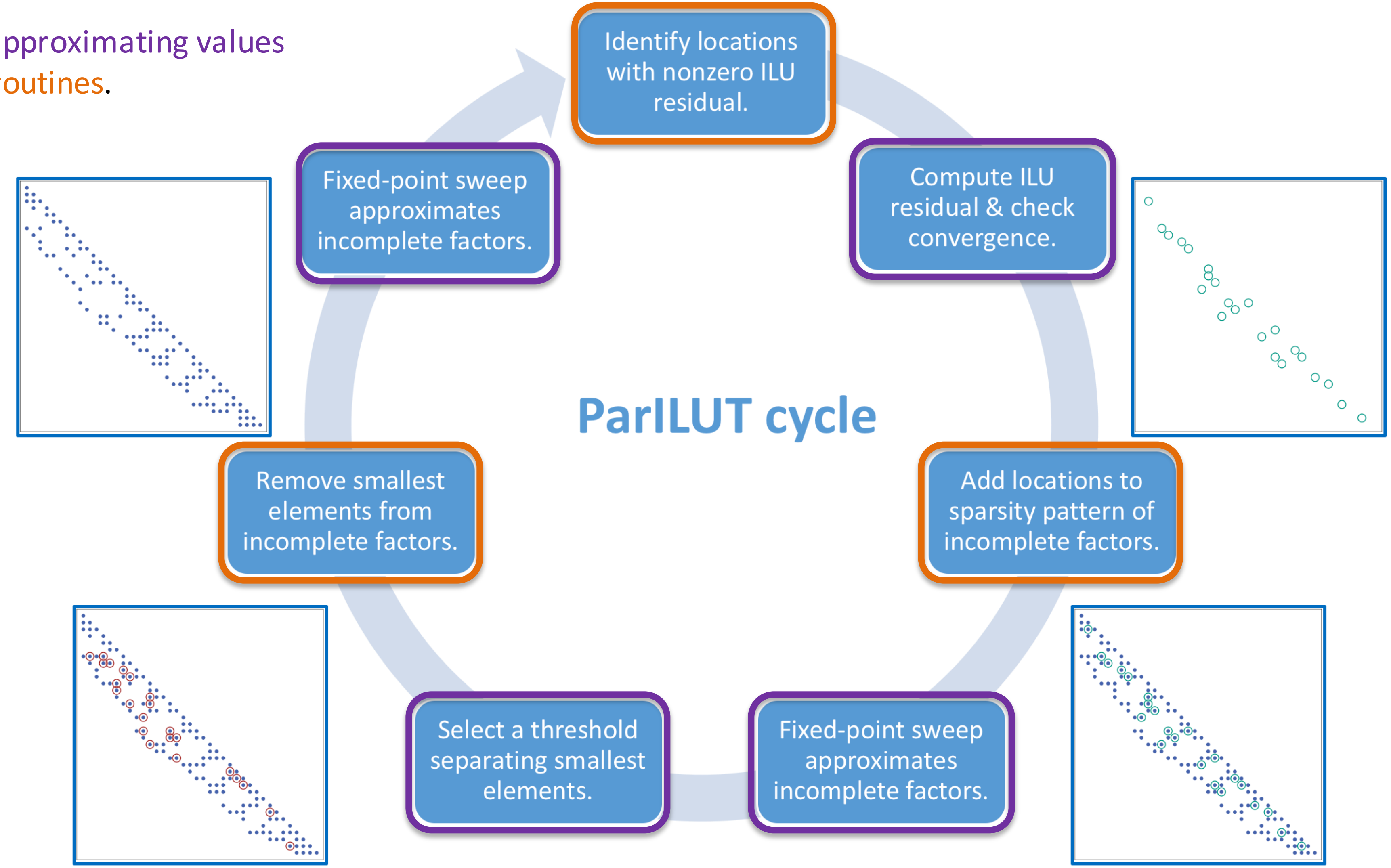
- At some point we should remove some locations again, e.g. the smallest elements, and start over looking at locations $R = A - L_k \cdot U_k$ …





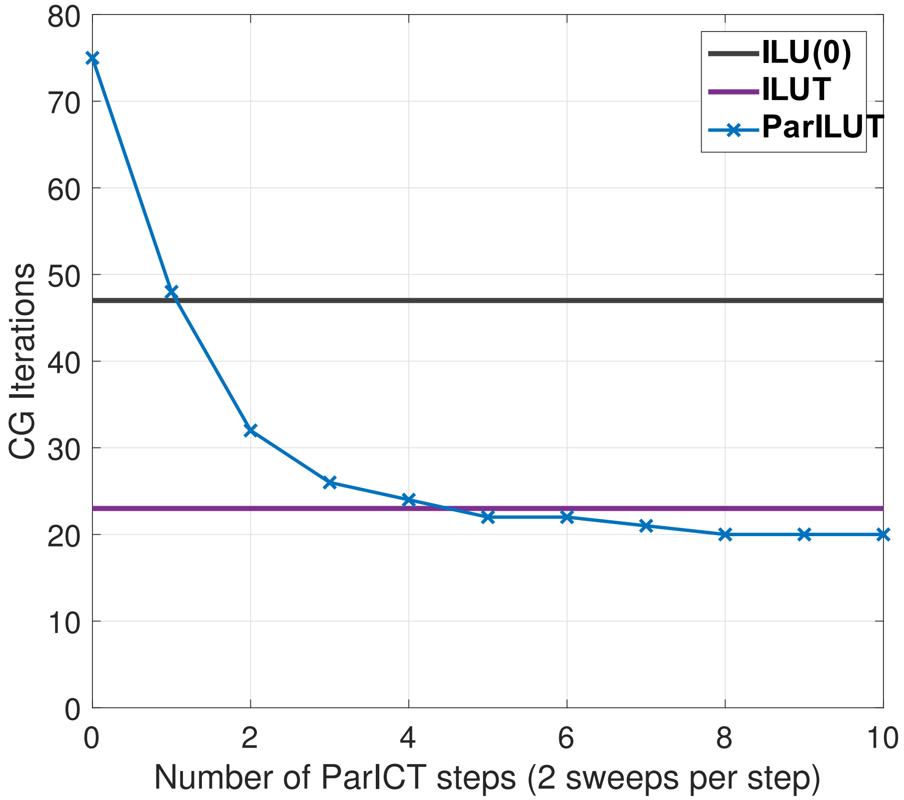
ParILUT Algorithm

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



ParILUT Quality

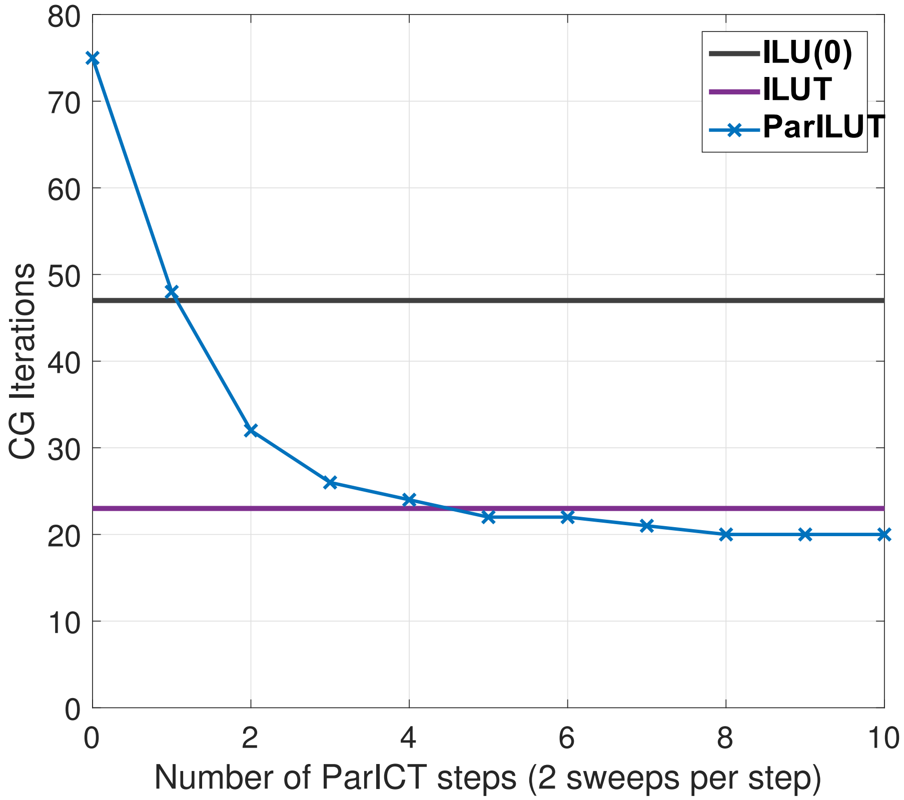
Anisotropic diffusion problem
n: 741, nz: 4,951

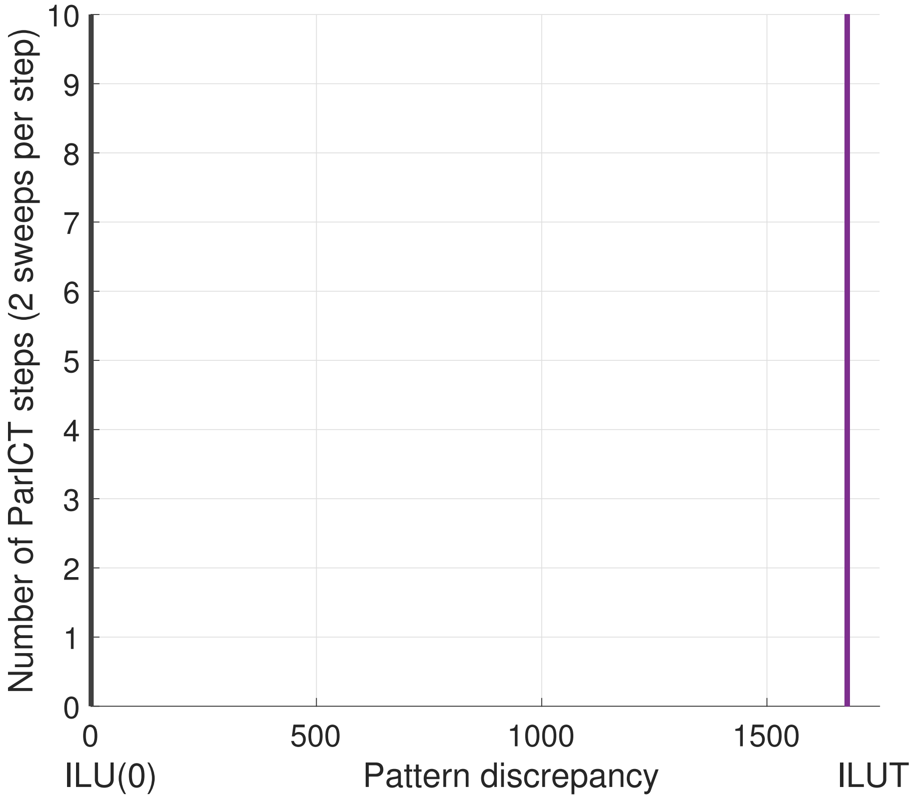


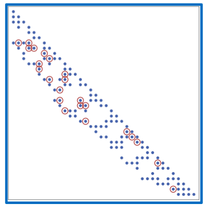
- Top-level solver iterations as quality metric.
- Few sweeps give a "better" preconditioner than ILU(0).
- Better than conventional ILUT?

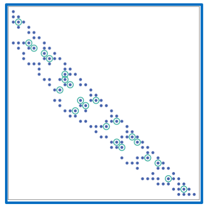
ParILUT Quality

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





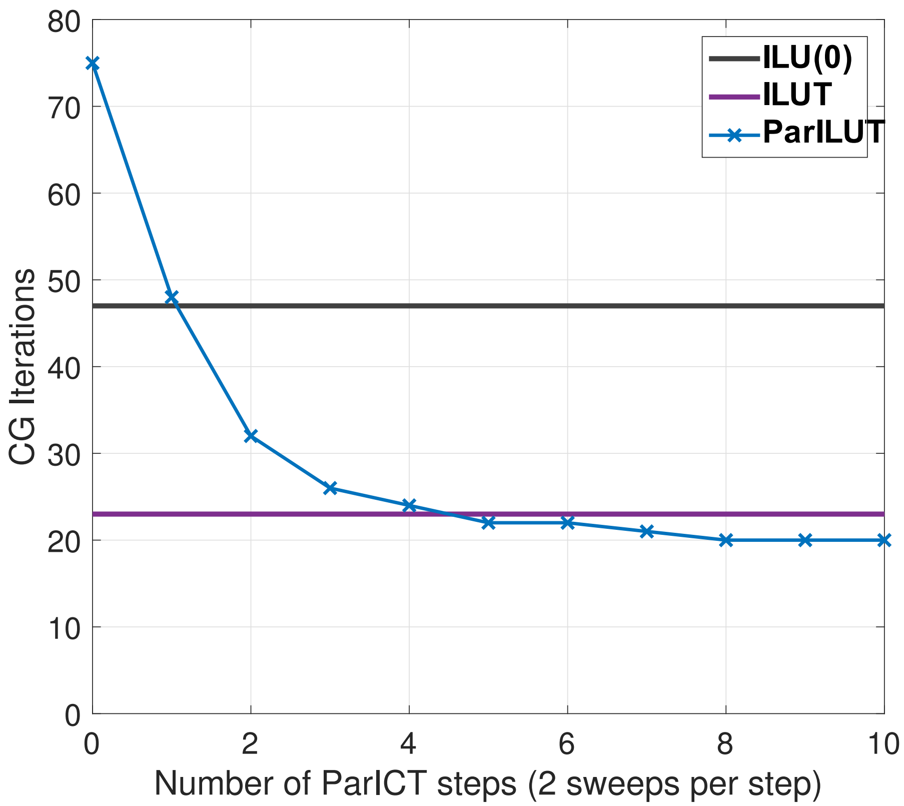


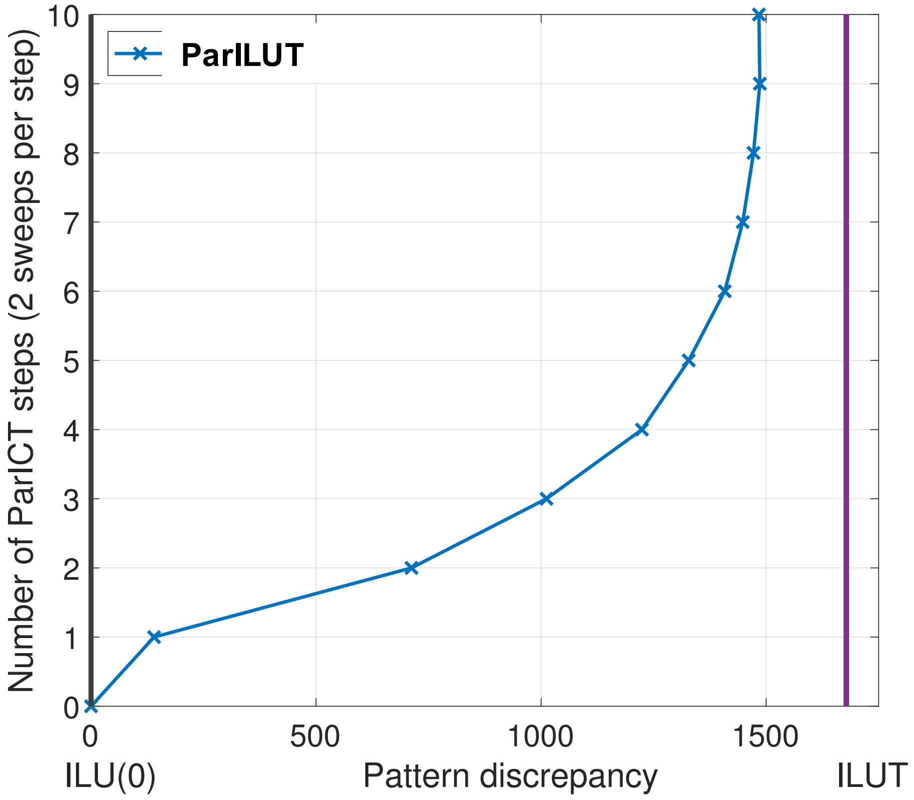


- Top-level solver iterations as quality metric.
- Few sweeps give a "better" preconditioner than ILU(0).
- Better than conventional ILUT?

ParILUT Quality

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





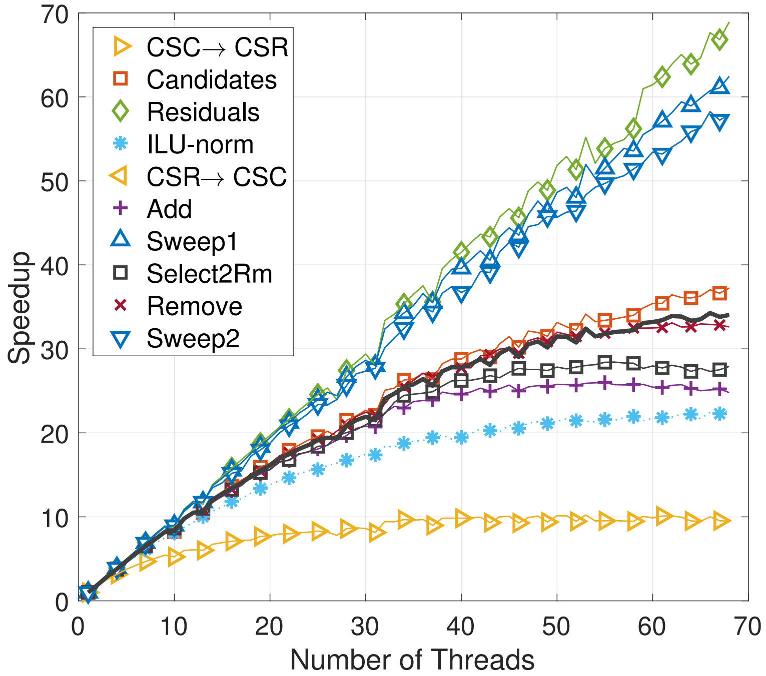
- Top-level solver iterations as quality metric.
- Few sweeps give a “better” preconditioner than ILU(0).
- Better than conventional ILUT?

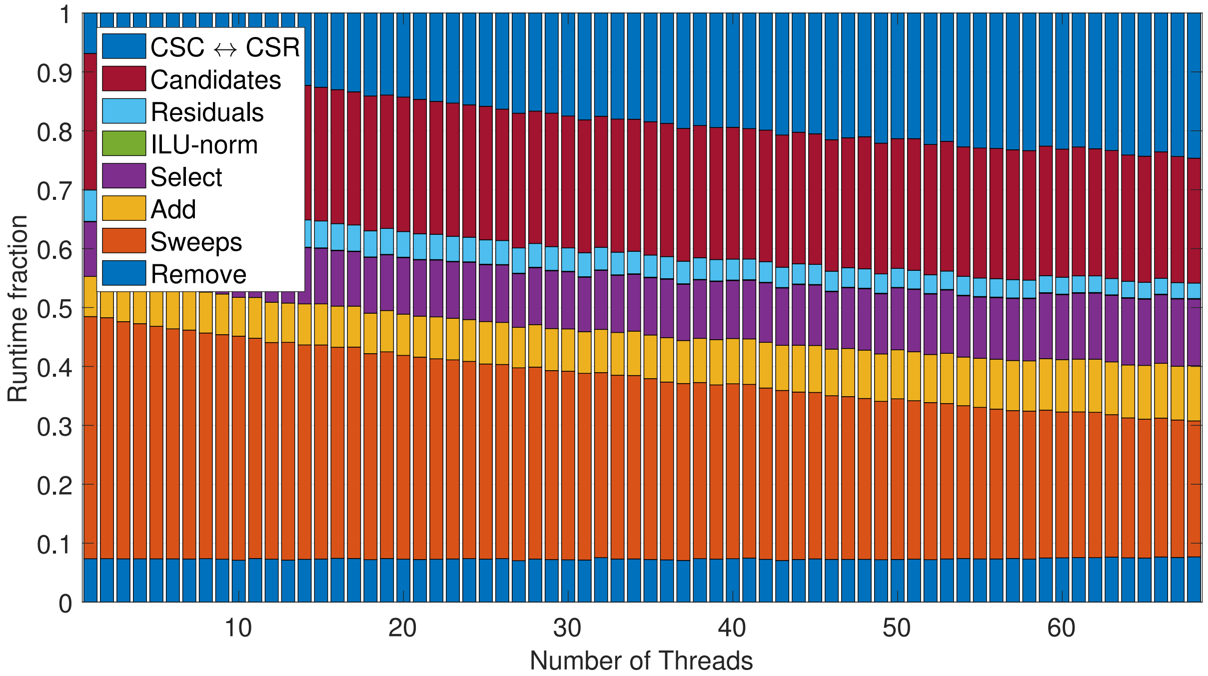
- Pattern converges after few sweeps.
- Pattern “more like” ILUT than ILU(0).

ParILUT Scalability

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

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



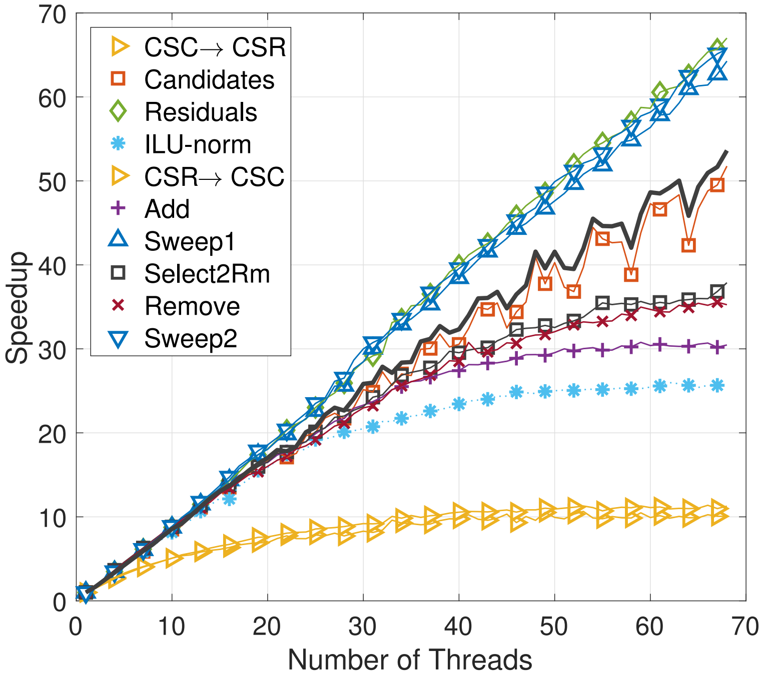


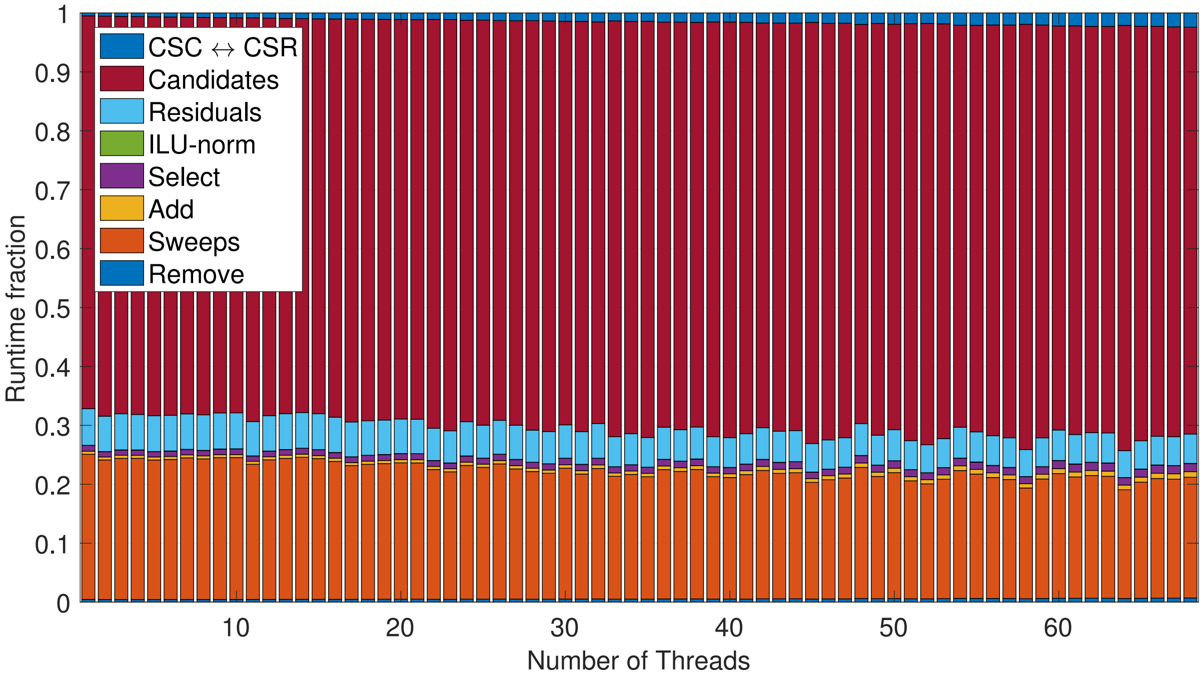
- Building blocks scale with 15% - 100% parallel efficiency.
- Transposition and sort are the bottlenecks.
- Overall speedup ~35x when using 68 KNL cores.

ParILUT 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.

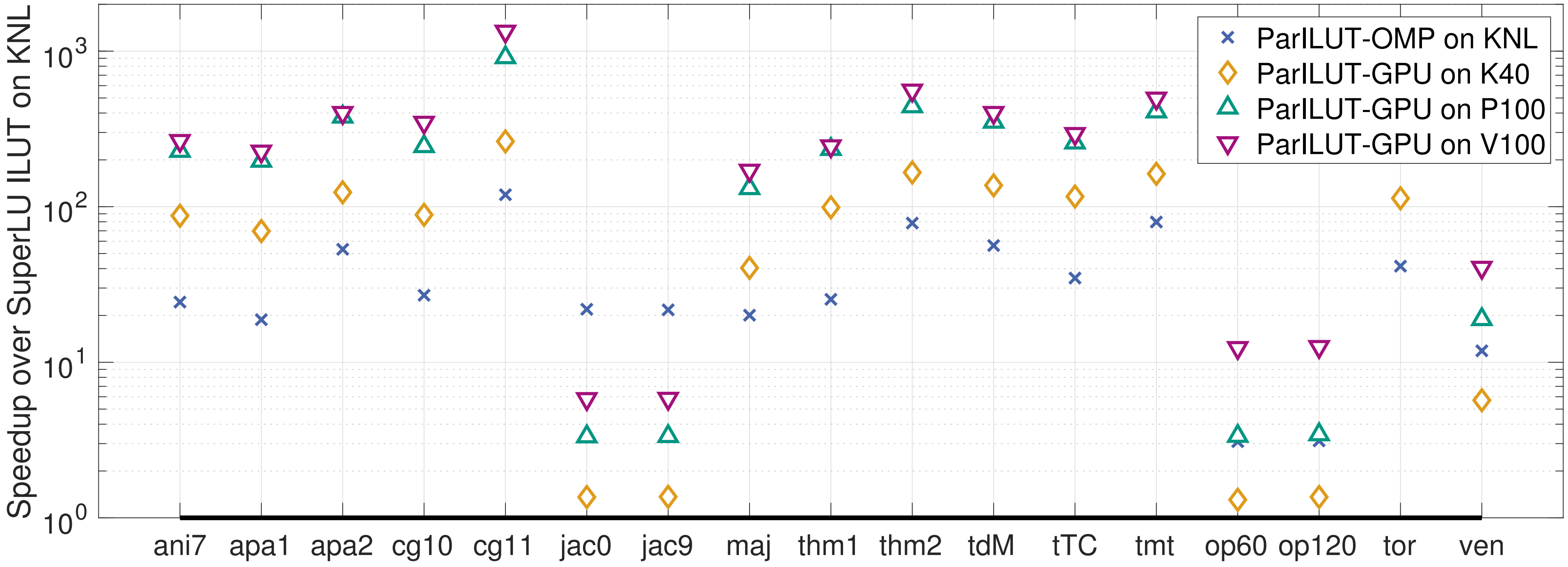




- Building blocks scale with 15% - 100% parallel efficiency.
- Dominated by candidate search.
- Overall speedup ~52x when using 68 KNL cores.

ParILUT Performance across Manycore architectures

We compare against ILUT in SuperLU from LBNL – and thank *Sherry Li* for help and support in doing this comparison.
The SuperLU ILUT is a sequential implementation – **ParILUT is the first parallel ILUT algorithm.**



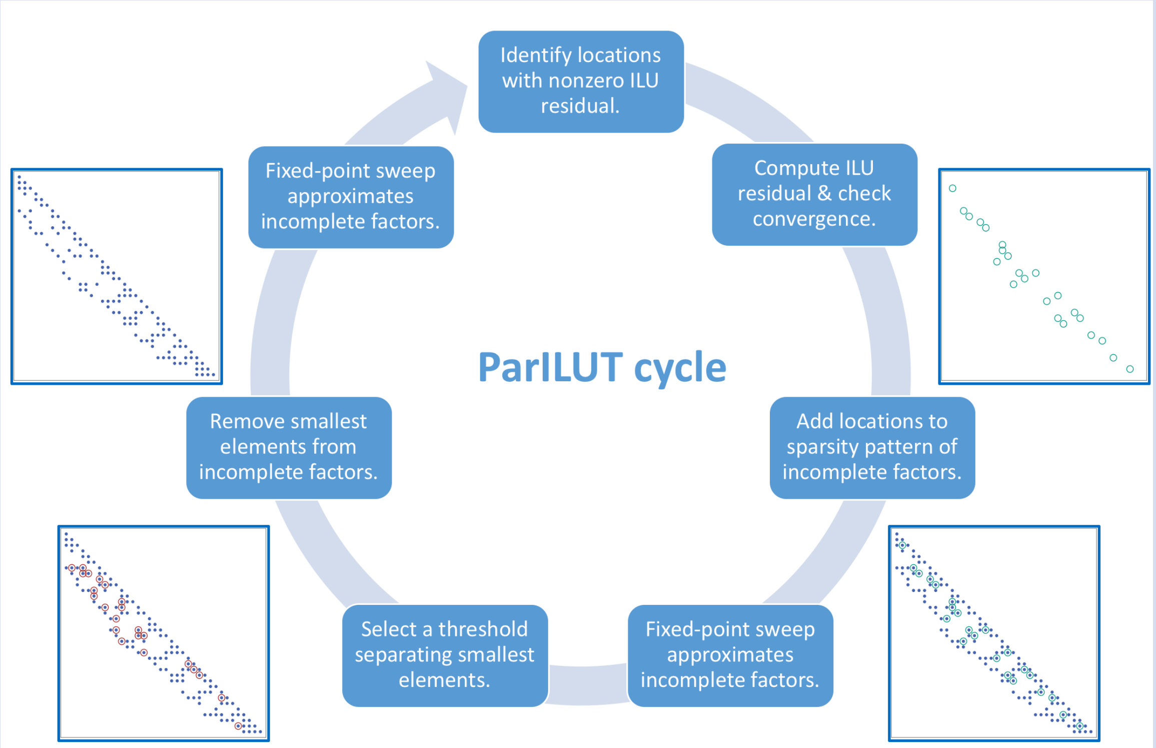
Bibliography: [1]Chow et al. **"Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs"**. *In ISC 2015.*
[2]Anzt et al. **"ParILUT – A new parallel threshold ILU".** *In: SIAM Journal on Scientific Comp. (2018).*
[3]Ribizel et al. **"Approximate and Exact Selection on GPUs".** *In AsHES workshop, 2019.*
[4]Anzt et al. **"ParILUT – A parallel threshold ILU for GPUs".** *In IPDPS conference, 2019.*

The Manycore Challenge

Reformulate algorithms as element-parallel fixed-point Iterations

- Algorithms need fine-grained parallelism

  -- thousands of SIMT threads!
- Global synchronizations are killing performance;
- Runtime scheduling of thread blocks virtually impossible;
- Memory access pattern central (coalesced data access);
- Asynchronous algorithms needed;
- Reformulation as fixed-point iteration;



Copyright@ORNL

The Software Challenge

- **Software is an central component in Exascale Computing!**
  - We should focus more on sustainable software than on hardware development.
  - Software often lives longer than a HPC system.

- **Close collaboration with hardware developers and Universities is key to prepare for future hardware!**

- **We still lack the acceptance of scientific software engineers!**
  - The standard perception is: we buy new hardware, your core runs faster….
  - We need the **academic acceptance** of **scientific software engineers**!

- **We are running an inefficient, publication-driven system ignoring the importance of production code!**

Creating a Sustainable HPC Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

\+ **New ideas** presented;
\+ **Performance evaluations** presented;
\- Performance evaluations are typically "**selective**";
\- Users / Application Scientists need to **re-implement code**;
\- Difficult if **few details** are provided;
\- **Not integrated into community packages**;

Creating a Sustainable HPC Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

+ **New ideas** presented;
+ **Performance evaluations** presented;
- Performance evaluations are typically "**selective**";
- Users / Application Scientists need to **re-implement code**;
- Difficult if **few details** are provided;
- **Not integrated into community packages**;

Established community software packages

- are the powertrain behind many scientific simulation codes;
- often fall short in providing production-ready implementations of novel algorithms;
- often accept merge requests that lack comprehensive documentation and rigorous performance assessment;

TRILINOS
PETSc
Ginkgo
hypre
MAGMA
GROMACS FAST. FLEXIBLE. FREE.
LAMMPS

Creating a Sustainable HPC Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

+ **New ideas** presented;
+ **Performance evaluations** presented;
- Performance evaluations are typically "**selective**";
- Users / Application Scientists need to **re-implement code**;
- Difficult if **few details** are provided;
- **Not integrated into community packages**;

In a **perfect world**, new algorithms, implementations & performance results are

- fully reproducible;
- publicly accessible;
- ready to be used by the community / domain scientists;
- integrated into community packages;

Established community software packages

- are the powertrain behind many scientific simulation codes;
- often fall short in providing production-ready implementations of novel algorithms;
- often accept merge requests that lack comprehensive documentation and rigorous performance assessment;

TRILINOS

PETSc

Ginkgo

hypre

MAGMA

GROMACS
FAST. FLEXIBLE. FREE.

LAMMPS

Creating a Sustainable HPC Landscape

The Typical Publication in HPC Conferences / Journals

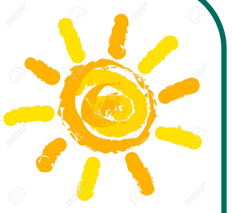
- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

+ **New ideas** presented;
+ **Performance evaluations** presented;
- Performance evaluations are typically "**selective**";
- Users / Application Scientists need to **re-implement code**;
- Difficult if **few details** are provided;
- **Not integrated into community packages**;

In a **perfect world**, new algorithms, implementations & performance results are

- fully reproducible;
- publicly accessible;
- ready to be used by the community / domain scientists;
- integrated into community packages;



Established community software packages

- are the powertrain behind many scientific simulation codes;
- often fall short in providing production-ready implementations of novel algorithms;
- often accept merge requests that lack comprehensive documentation and rigorous performance assessment;

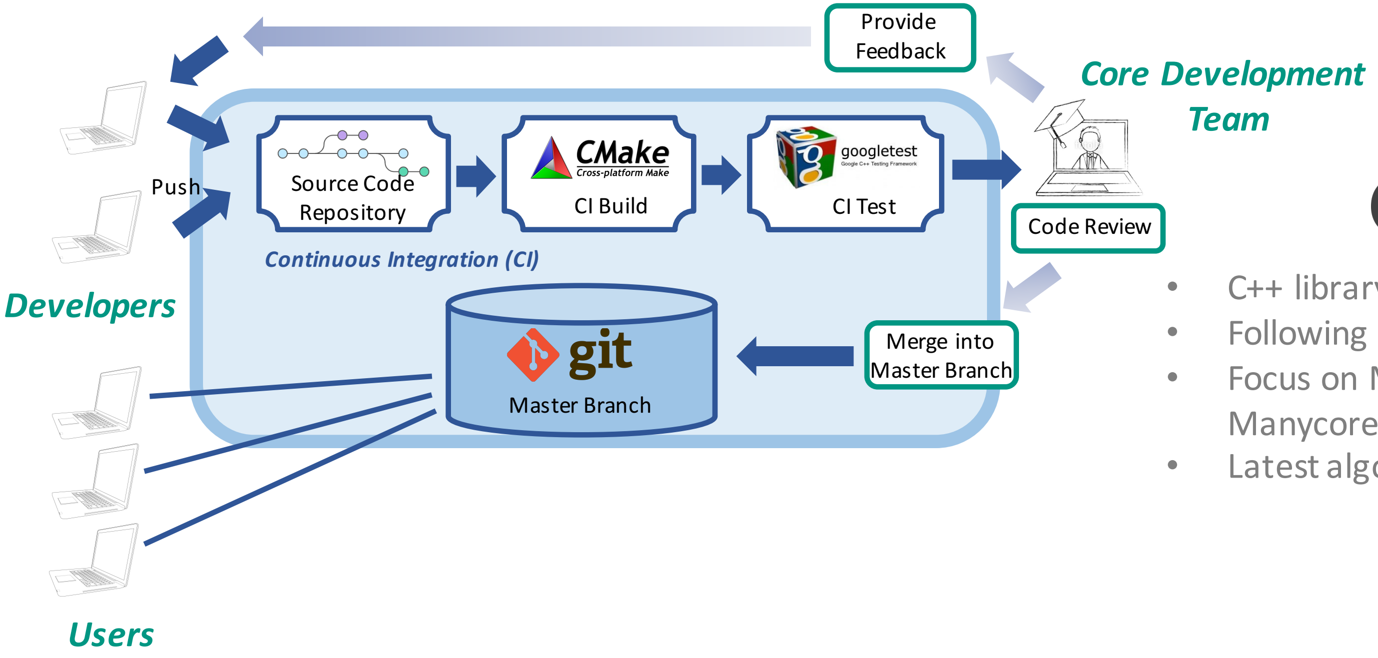
Why are we not changing the system?

- $\text{effort(Prototype Code)} \ll \text{effort(Production Code)}$;
- Little academic reward for sustainable software development;
- Promotion and appointability based on scientific papers;

Status Quo Extremely inefficient and unsatisfying!

My Efforts towards a Sustainable HPC Landscape

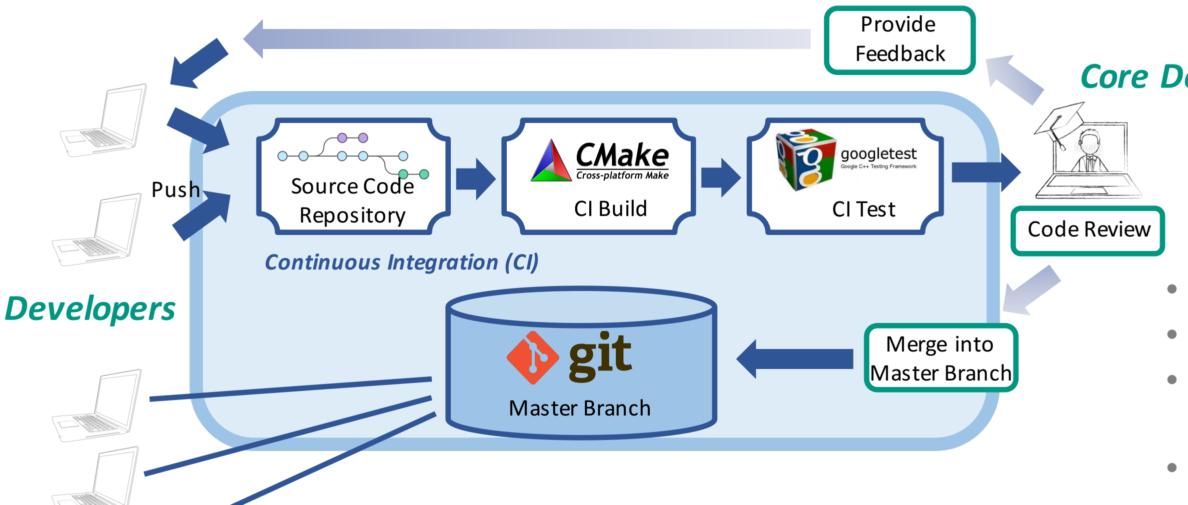
1. Sustainable Software Development in the HYIG FiNE



- C++ library for Sparse Linear Algebra
- Following the SOLID design principle
- Focus on Multi- and Manycore Technology
- Latest algorithm developments

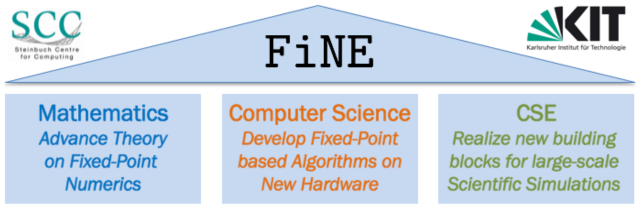
My Efforts towards a Sustainable HPC Landscape

1. Sustainable Software Development in the HYIG FiNE



Ginkgo

- C++ library for Sparse Linear Algebra
- Following the SOLID design principle
- Focus on Multi- and Manycore Technology
- Latest algorithm developments



2. Community Engagement: Changing the Culture of Academic Software Development

- Promote **Sustainable Algorithm and Software Development** (PASC 2019)
  www.bit.ly/ContinuousBenchmarking
- Address the **Challenges** of **Academic Software** Development (BSSw Blog Article)
  www.bit.ly/AcademicResearchSoftware
- Argue for accepting **Software Patches as Full Conference Contributions** (PDSEC 2019)
  www.bit.ly/AreWeDoingTheRightThing
- Welcome Software Patches as Conference Contributions at
  Workshop on Scalable Data Analytics in Scientific Computing (SDASC 2020) in conjunction with ISC'20 in Frankfurt

Core Concept: Separate Algorithm from Kernels

Library core contains architecture-agnostic algorithm implementation;

Core

Library Infrastructure
Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

Architecture-specific kernels execute the algorithm on target architecture;

Kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

Core Concept: Separate Algorithm from Kernels

Library core contains architecture-agnostic algorithm implementation;

Architecture-specific kernels execute the algorithm on target architecture;

Kernels

Core

Library Infrastructure
Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

Reference

Reference kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Core Concept: Separate Algorithm from Kernels

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure
Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

Kernels

Reference

Reference kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

NVIDIA-GPU kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

Core Concept: Separate Algorithm from Kernels

Ginkgo

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure
Algorithm Implementations
- Iterative Solvers
- Preconditioners
- …

Kernels

Reference

Reference kernels
- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

NVIDIA-GPU kernels
- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels
- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

googletest
Google C++ Testing Framework

Core Concept: Separate Algorithm from Kernels

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure
Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

Kernels

Reference

Reference kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

NVIDIA-GPU kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

HIP

AMD-GPU kernels

- Accessor
- SpMV

Multi-GPU

NVIDIA-GPU kernels

- Accessor
- SpMV
- Solver kernels
- Precond kernels
- …

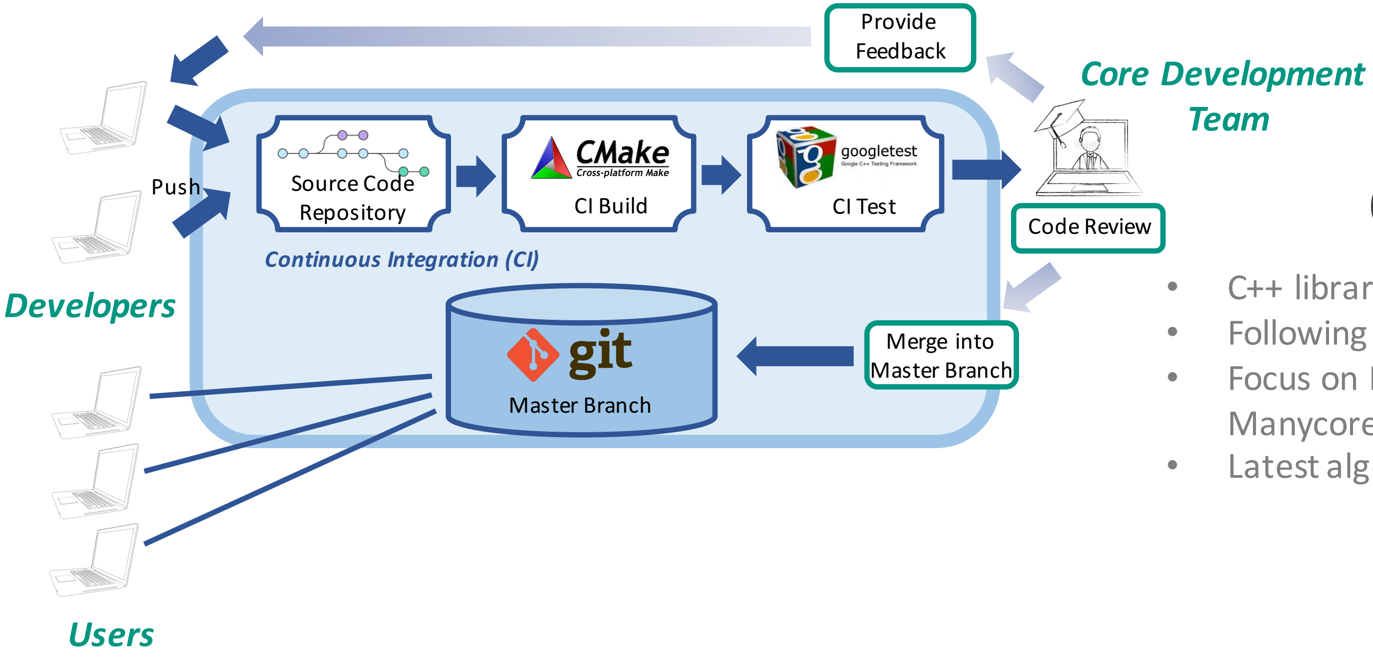
Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

googletest
Google C++ Testing Framework

My Efforts towards a Sustainable HPC Landscape

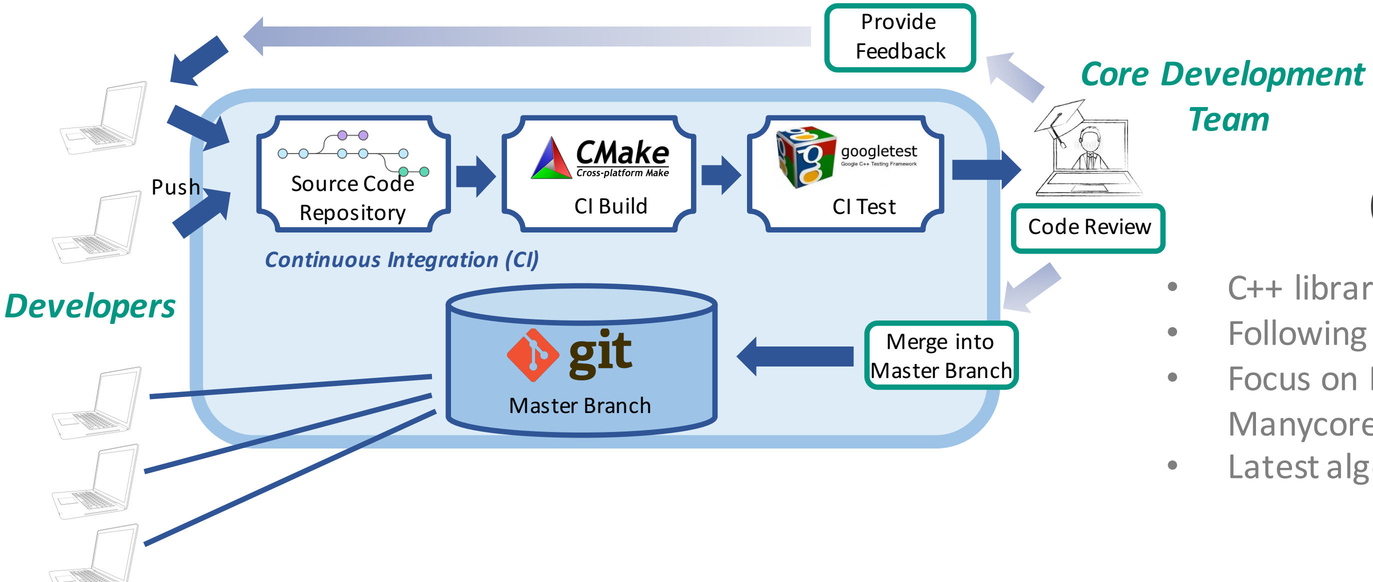
1. Sustainable Software Development in the HYIG FiNE



- C++ library for Sparse Linear Algebra
- Following the SOLID design principle
- Focus on Multi- and Manycore Technology
- Latest algorithm developments

My Efforts towards a Sustainable HPC Landscape

1. Sustainable Software Development in the HYIG FiNE



Ginkgo

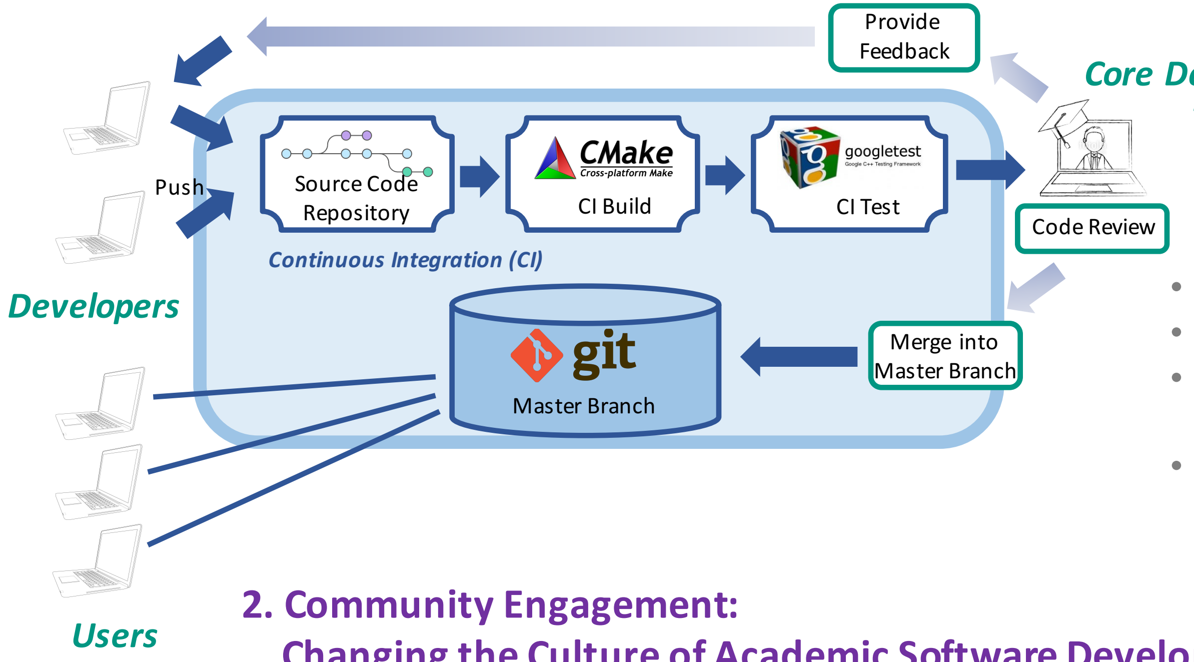
- C++ library for Sparse Linear Algebra
- Following the SOLID design principle
- Focus on Multi- and Manycore Technology
- Latest algorithm developments

2. Community Engagement: Changing the Culture of Academic Software Development

- Promote **Sustainable Algorithm and Software Development** (PASC 2019)
  www.bit.ly/ContinuousBenchmarking
- Address the **Challenges** of **Academic Software** Development (BSSw Blog Article)
  www.bit.ly/AcademicResearchSoftware
- Argue for accepting **Software Patches as Full Conference Contributions** (PDSEC 2019)
  www.bit.ly/AreWeDoingTheRightThing

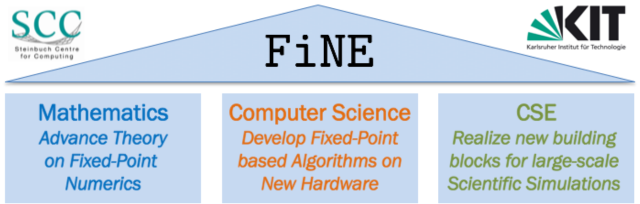
My Efforts towards a Sustainable HPC Landscape

1. Sustainable Software Development in the HYIG FiNE



Ginkgo

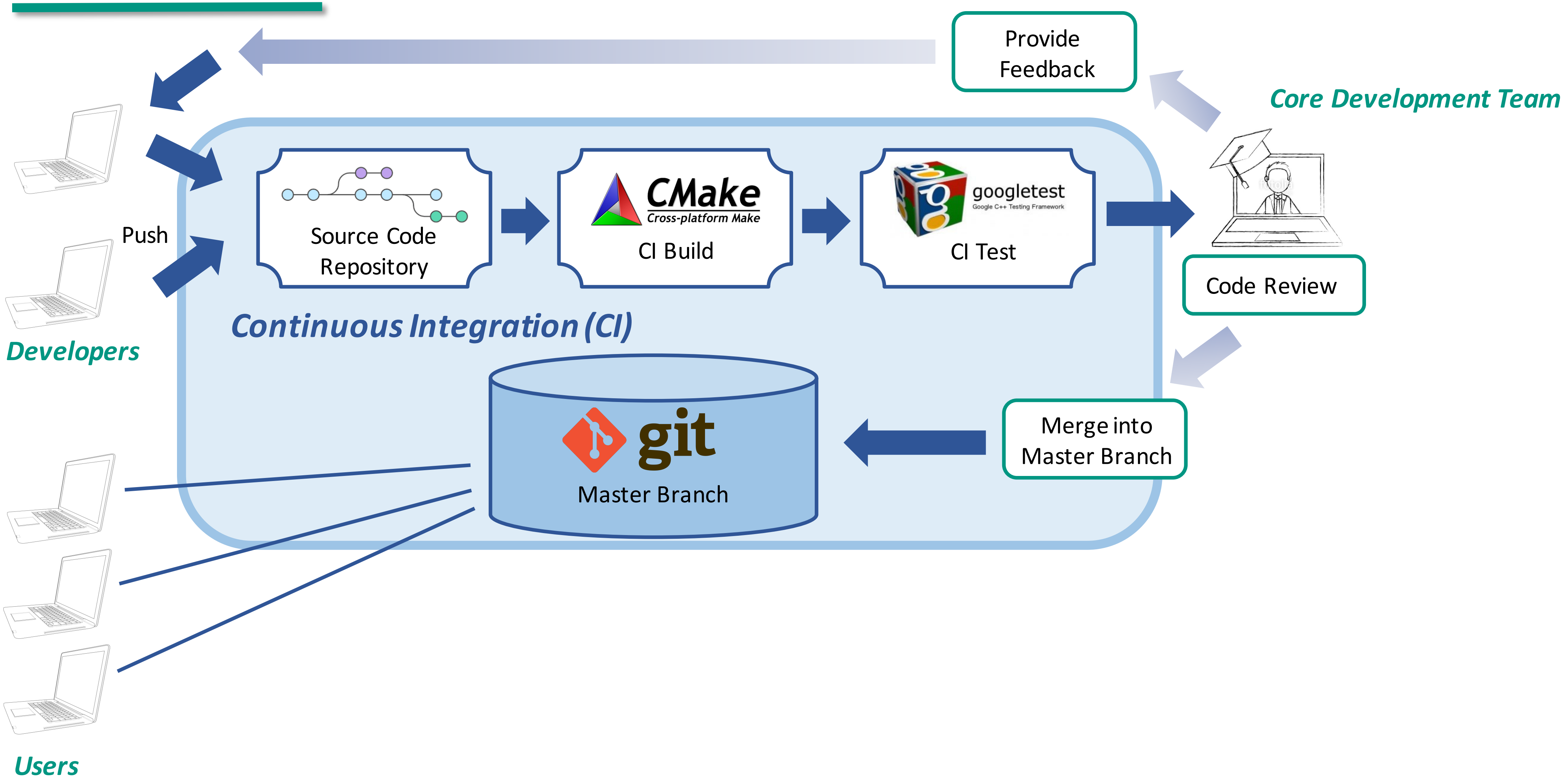
- C++ library for Sparse Linear Algebra
- Following the SOLID design principle
- Focus on Multi- and Manycore Technology
- Latest algorithm developments



2. Community Engagement: Changing the Culture of Academic Software Development

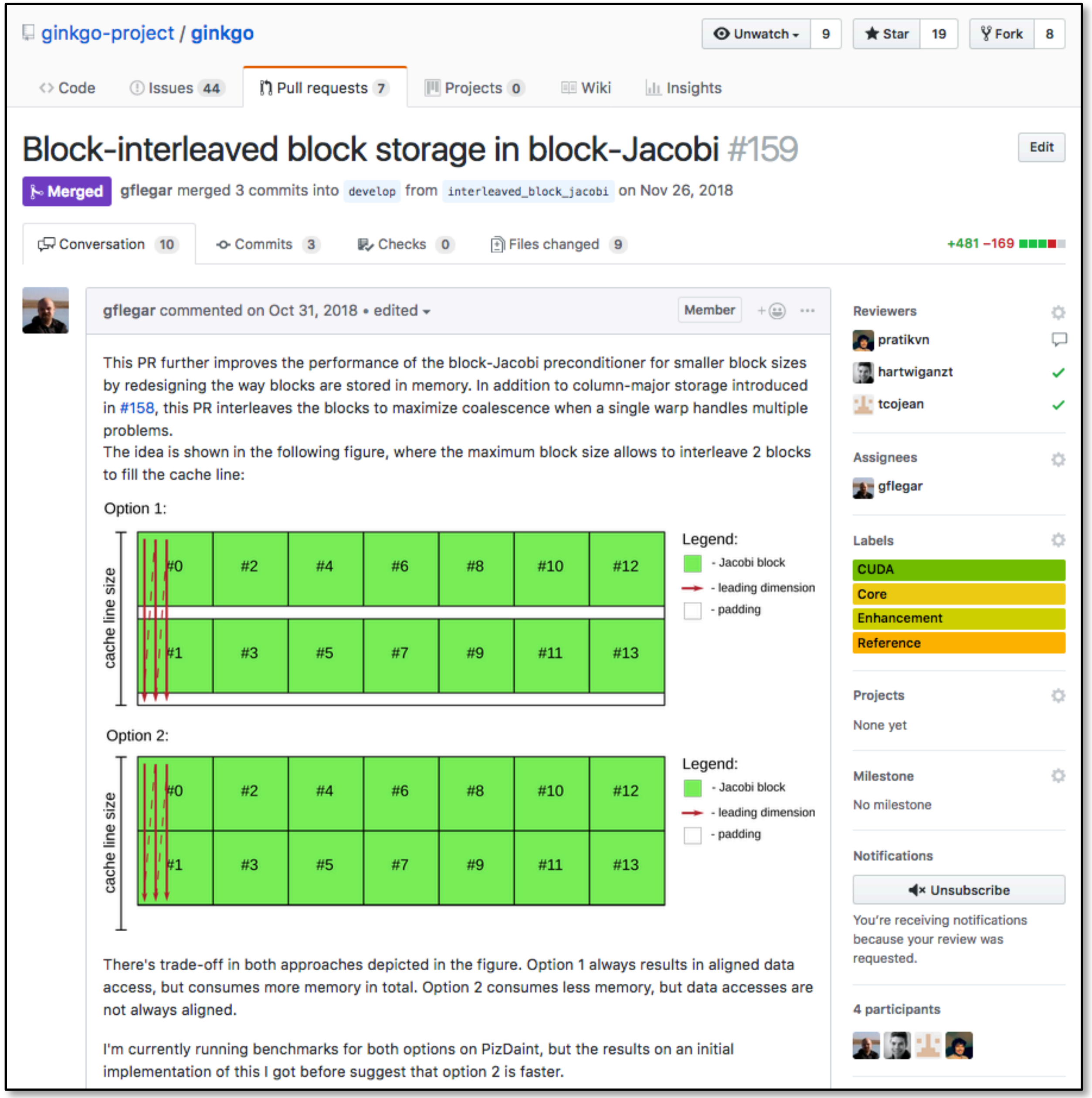
- Promote **Sustainable Algorithm and Software Development** (PASC 2019)
  www.bit.ly/ContinuousBenchmarking
- Address the **Challenges** of **Academic Software** Development (BSSw Blog Article)
  www.bit.ly/AcademicResearchSoftware
- Argue for accepting **Software Patches as Full Conference Contributions** (PDSEC 2019)
  www.bit.ly/AreWeDoingTheRightThing

A Healthy Software Development Cycle



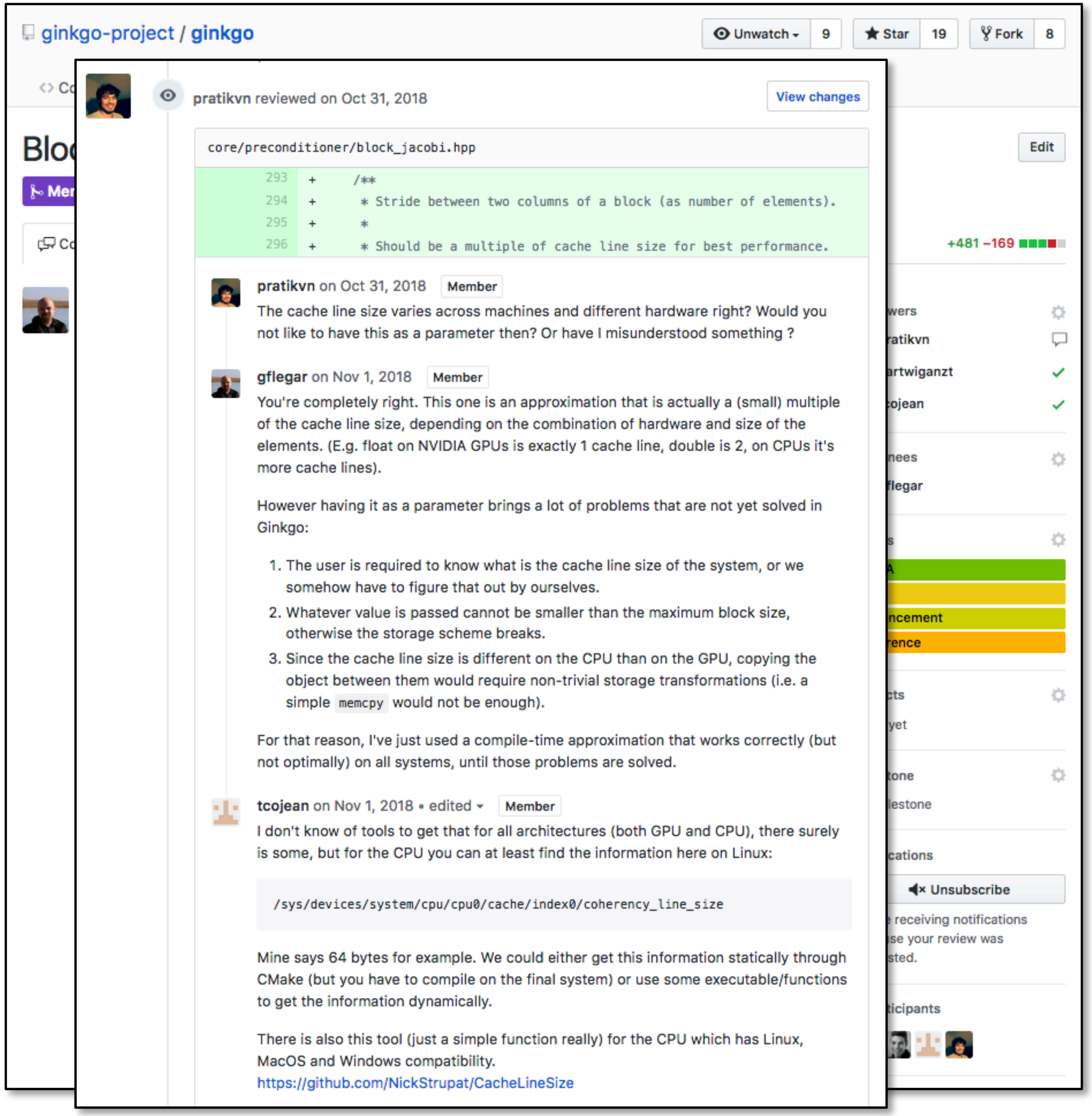
Software Patches

- Software patches usually submitted as *merge-/ push- request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.



Software Patches

- Software patches usually submitted as *merge-/ push- request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.

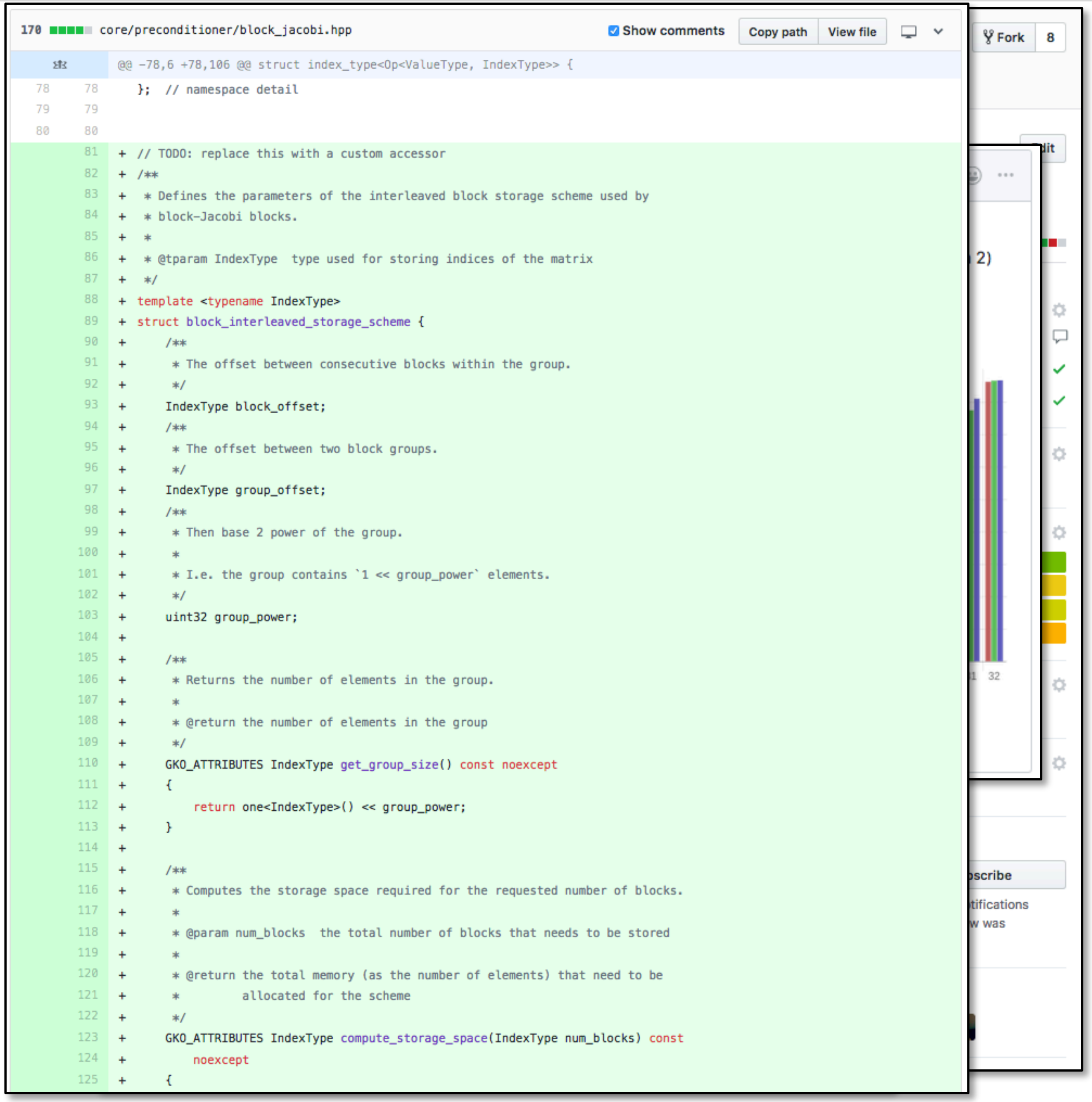


Software Patches

- Software patches usually submitted as *merge-/ push- request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.
- The submitter can attach a performance analysis to the software patch.

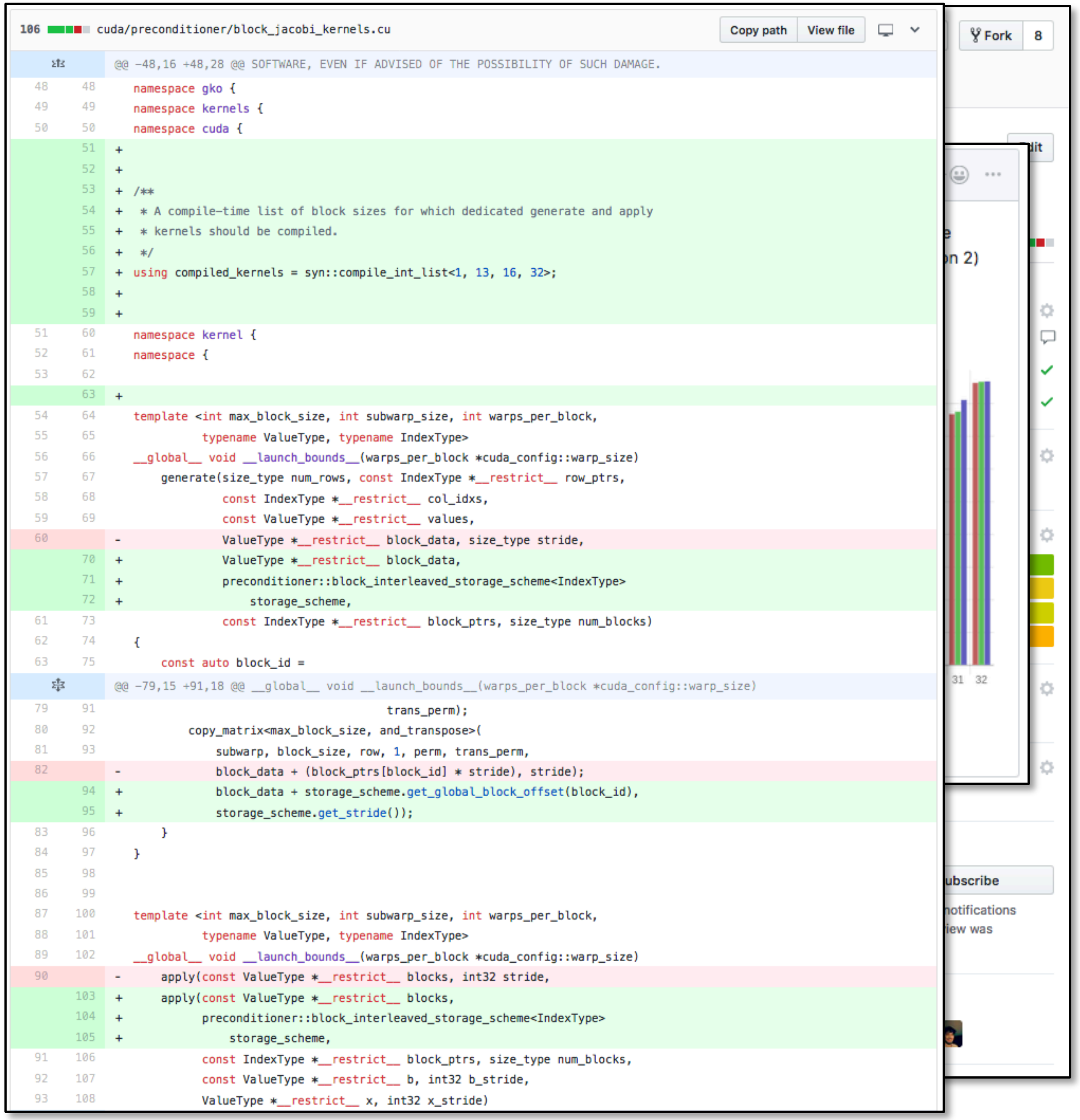
Software Patches

- Software patches usually submitted as *merge-/ push- request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.
- The submitter can attach a performance analysis to the software patch.
- Software patches can either add new functionality…



Software Patches

- Software patches usually submitted as *merge-/ push- request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.
- The submitter can attach a performance analysis to the software patch.
- Software patches can either add new functionality…
  … or change / enhance existing code.



Software Patches as Conference Contribution

- ✓ **Full reproducibility** and **traceability** is ensured;
- ✓ Not only reviewers but the complete **community can track the software patch**;
- ✓ The versioning systems helps to **identify the main contributors** of a software contribution, **ensuring full recognition**;
- ✓ The versioning systems also **links to the right person in case of technical questions**;
- ✓ **Novel algorithms** and hardware-optimized implementations are **quickly integrated into community packages**;
- ✓ The **code quality is increased** as the community can comment on the patches;
- ✓ Software patches as conference contributions naturally imply an **extremely high level of code documentation**;
- ✓ Presenting patches at a conference **makes** the whole **community aware of a new feature**;
- ✓ **Domain scientists** can directly **interact with software developers**;

Software Patches as Conference Contribution

Envisioned Workflow:

1. The algorithm/implementation **developer submits a software patch to a community package** with
   - detailed description of the functionality and code documentation;
   - comprehensive performance assessment;
   - mark the patch for a conference contribution;
2. The **core development team and the community**
   - **comments** on the algorithm, the implementation, and the performance;
   - **reviews** and ultimately merges the patch;
3. The **developer submits the patch as a conference contribution**
   - linking to all documentation, performance results, and comments;
   - acknowledging significant comments from community;

Software Patches as Conference Contribution

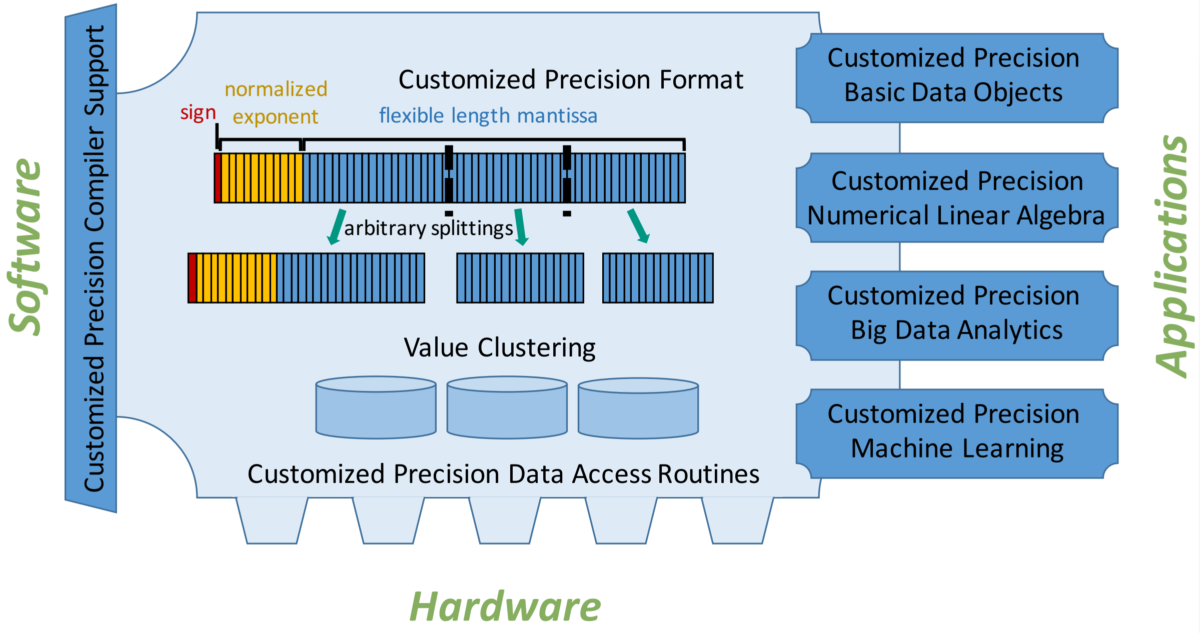
Envisioned Workflow:

1. The algorithm/implementation **developer submits a software patch to a community package** with
   - detailed description of the functionality and code documentation;
   - comprehensive performance assessment;
   - mark the patch for a conference contribution;
2. The **core development team and the community**
   - **comments** on the algorithm, the implementation, and the performance;
   - **reviews** and ultimately merges the patch;
3. The **developer submits the patch as a conference contribution**
   - linking to all documentation, performance results, and comments;
   - acknowledging significant comments from community;
4. The **conference committee / external reviewers** do a "light" review of functionality, documentation, performance.
5. If accepted, the conference contribution is presented along with a **user tutorial or application examples**;
6. The submission is as a **regular paper** included in the conference proceedings
   - potentially featuring a shorter general introduction;
   - including the algorithm description and performance assessment;
     potentially including code segments, digital artifacts, or a link to the merge request;
   - listing all (significant) code reviewers / commenters;

Summary and next steps

- **Decouple arithmetic precision from memory precision.**
- Using **customized precisions** for memory operations.
- Speedup of up to 1.3x for adaptive precision block-Jacobi preconditioning.
- Creating a **Modular Precision Ecosystem** inside Ginkgo.

https://github.com/ginkgo-project/ginkgo



EXASCALE COMPUTING PROJECT

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration and the Helmholtz Impuls und Vernetzungsfond VH-NG-1241.

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]:

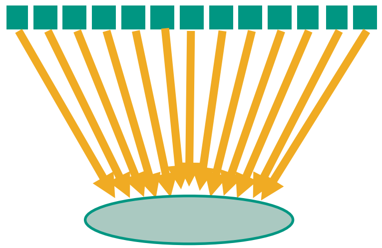
- Inherently parallel operation.
- Elements can be updated asynchronously.

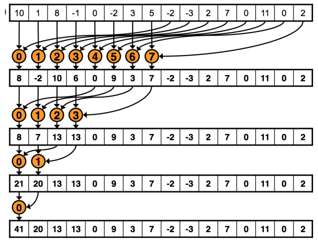
- *We can expect 100% parallel efficiency if number of cores < number of elements*

- 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

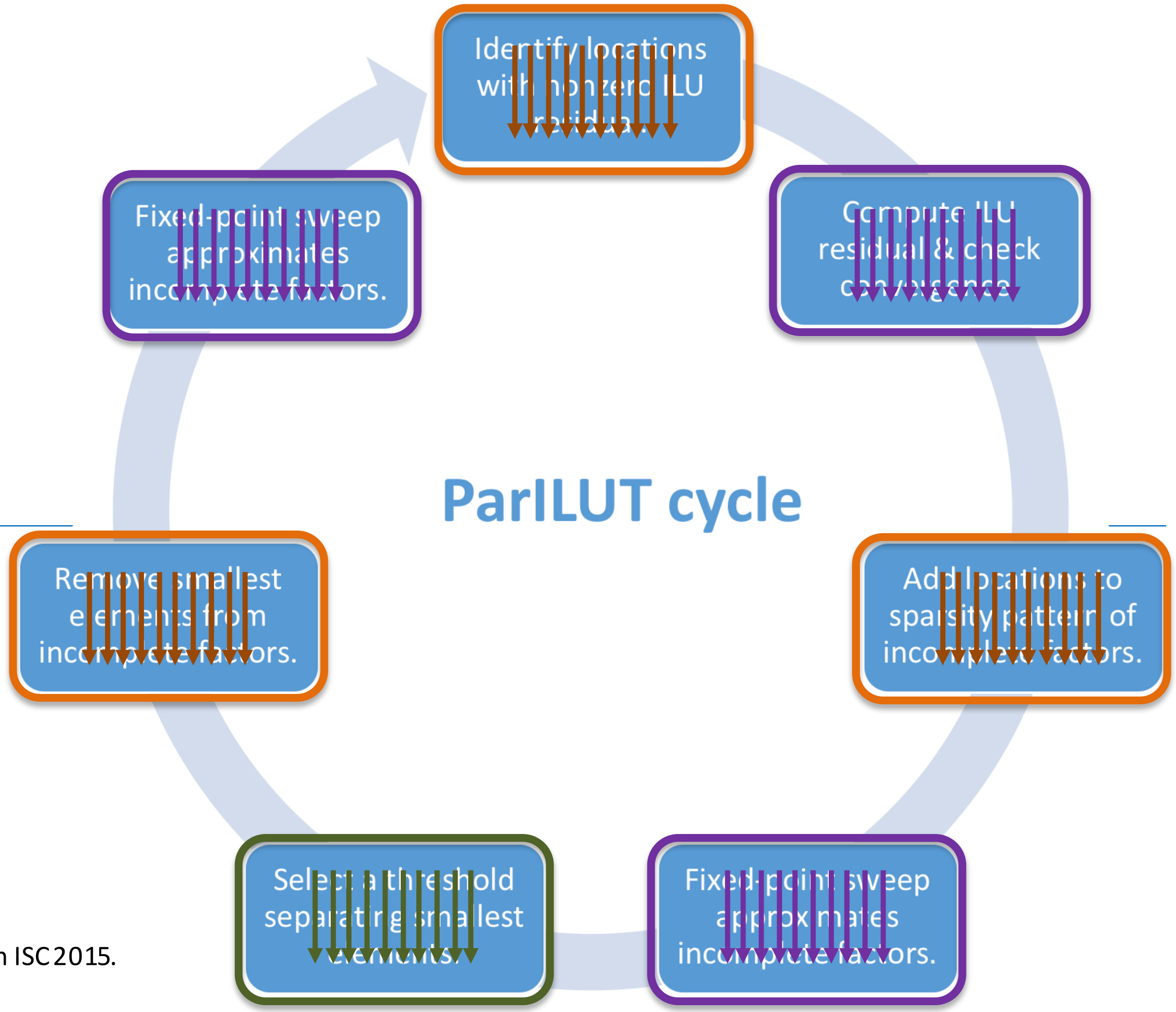




[1]Chow et al. **"Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs"**. In ISC 2015.

ParILUT : Parallelism inside the blocks

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



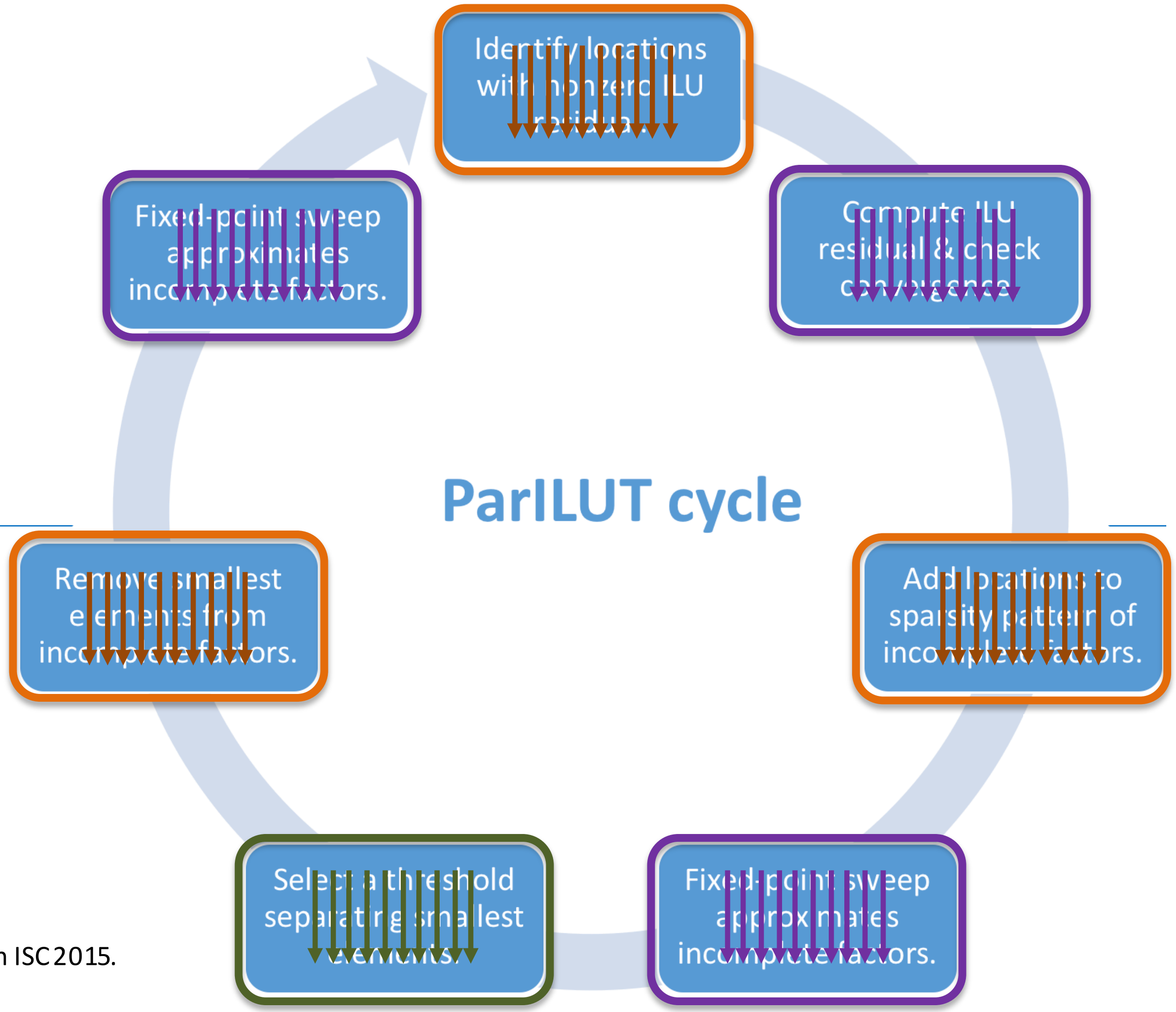
Parallelism inside the building blocks:

- 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).*

ParILUT : Parallelism inside the blocks

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).*

Parallel 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

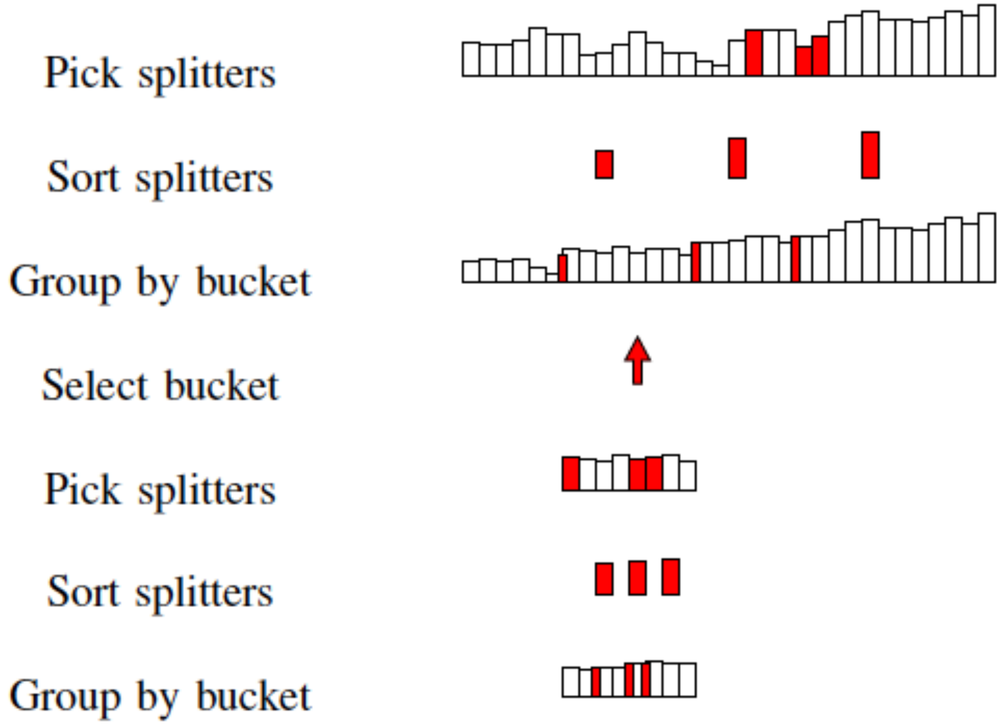
$$x_{i_0} \le x_{i_1} \le x_{i_2} \le x_{i_3} \le \cdots \le x_{i_k} \le \ldots x_{i_{n-1}}$$

$$\uparrow k$$

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Tobias Ribizel

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

Parallel 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

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

$$\uparrow$$
$$k$$

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

We do not necessarily need to sort the complete sequence!

Tobias Ribizel

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

Parallel 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

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

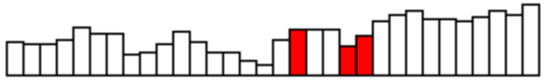
$$\uparrow k$$

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm

Pick splitters



Tobias Ribizel

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

Parallel 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

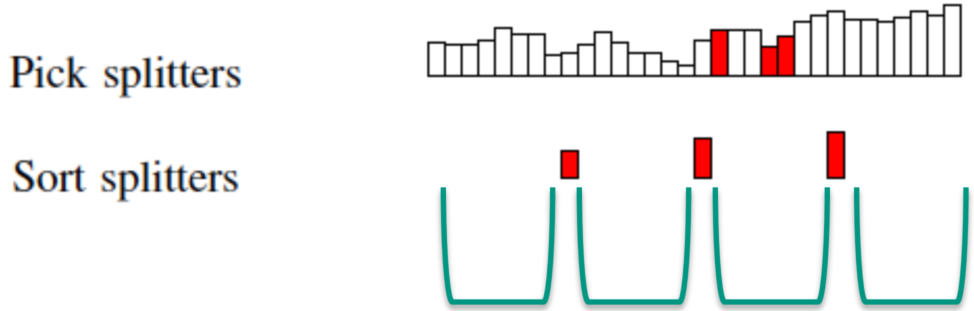
$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \dots \leq x_{i_k} \leq \dots x_{i_{n-1}}$$

$$\uparrow k$$

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Splitters separate buckets

Tobias Ribizel

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

Parallel 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

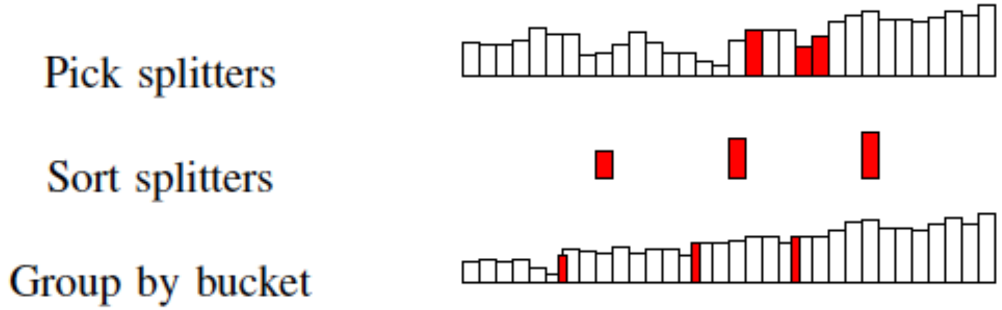
$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \dots \leq x_{i_k} \leq \dots x_{i_{n-1}}$$

$$\uparrow k$$

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Tobias Ribizel

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

Parallel 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

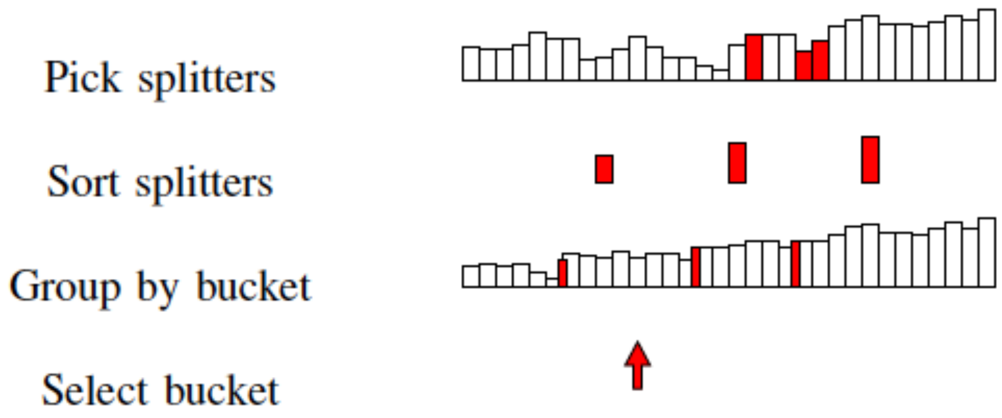
$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \cdots \leq x_{i_k} \leq \ldots x_{i_{n-1}}$$

$$\uparrow k$$

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Tobias Ribizel

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

Parallel 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

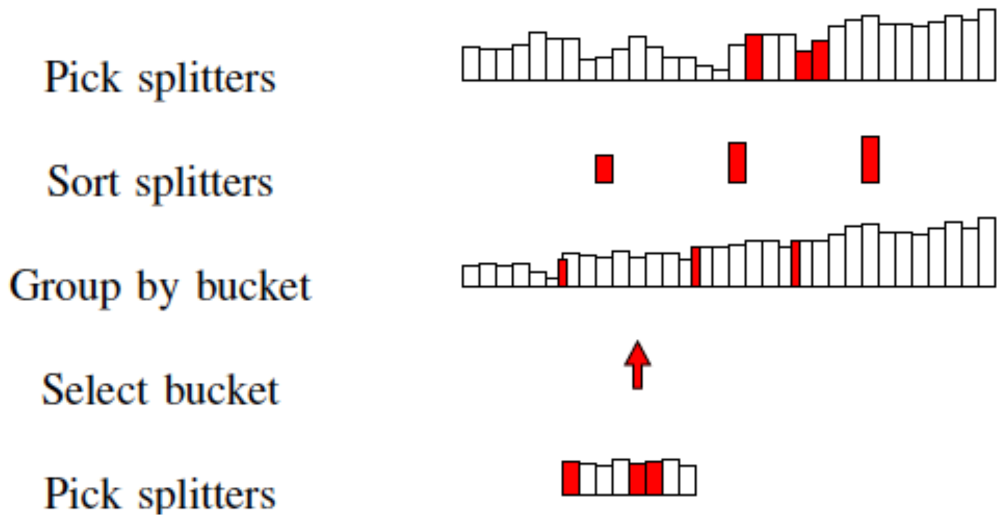
$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \dots \leq x_{i_k} \leq \dots x_{i_{n-1}}$$

$\uparrow$ k

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Tobias Ribizel

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

Parallel 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

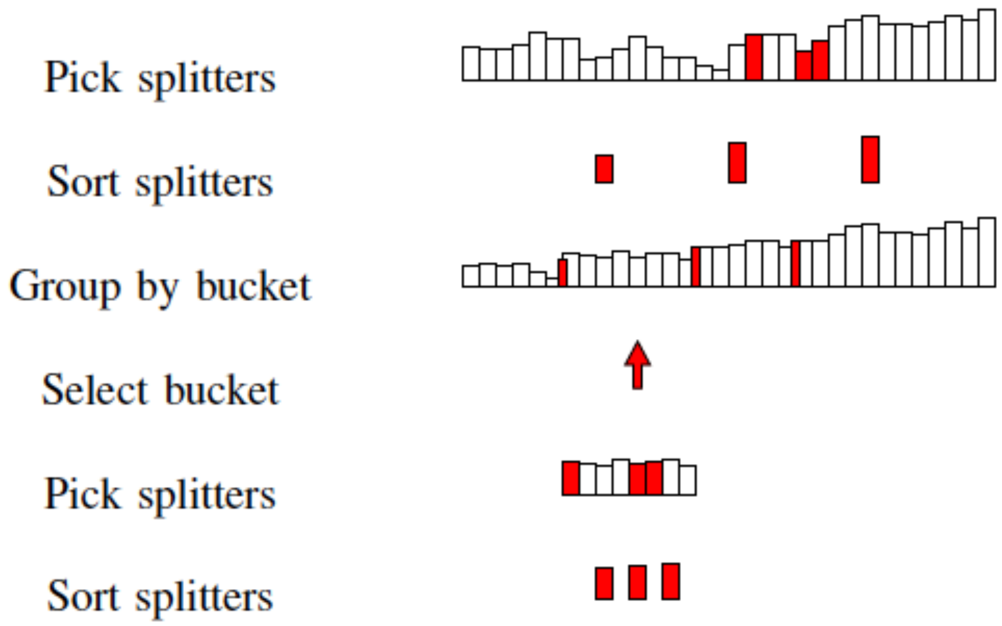
$$x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq x_{i_3} \leq \cdots \leq x_{i_k} \leq \dots x_{i_{n-1}}$$

↑ k

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Tobias Ribizel

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

Parallel 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

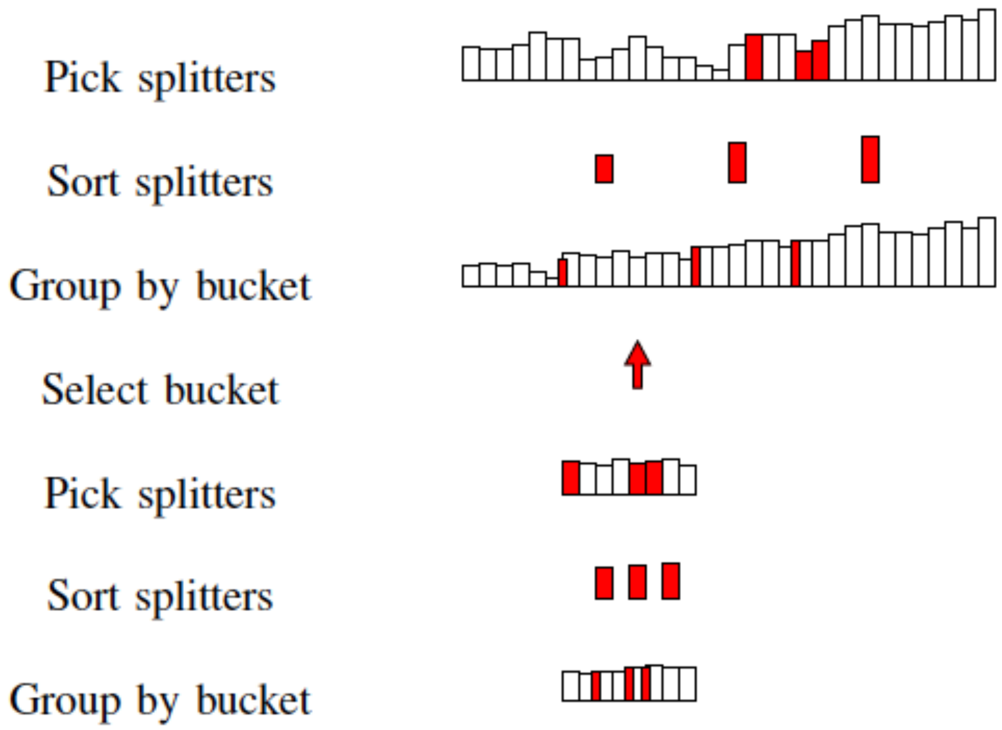
$$x_{i_0} \le x_{i_1} \le x_{i_2} \le x_{i_3} \le \dots \le x_{i_k} \le \dots x_{i_{n-1}}$$

$$\uparrow k$$

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

We do not necessarily need to sort the complete sequence!

SampleSelect Algorithm



Tobias Ribizel

Approximate and Exact Selection on GPUs

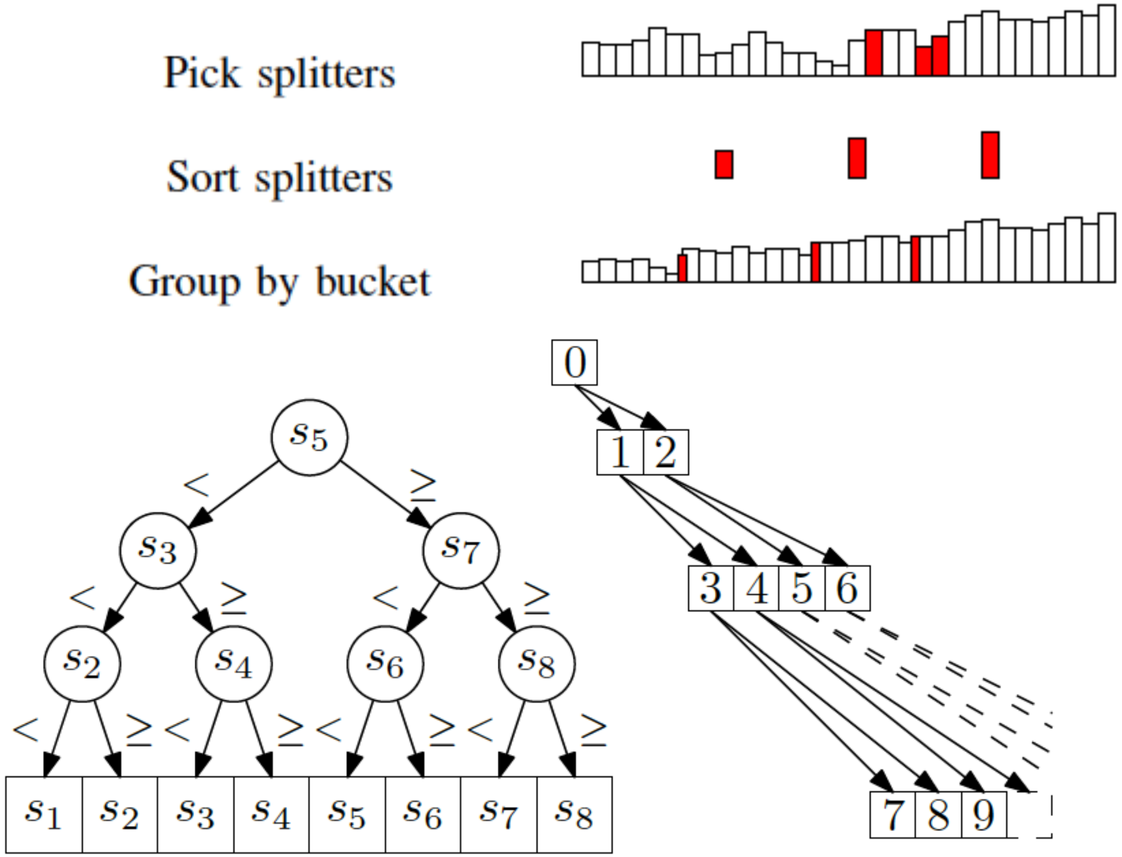
Tobias Ribizel*, Hartwig Anzt*†
*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany
†Innovative Computing Lab (ICL), University of Tennessee, Knoxville, USA
tobias.ribizel@student.kit.edu, hartwig.anzt@kit.edu

http://bit.ly/SampleSelectGPU

Parallel Threshold Selection on GPUs

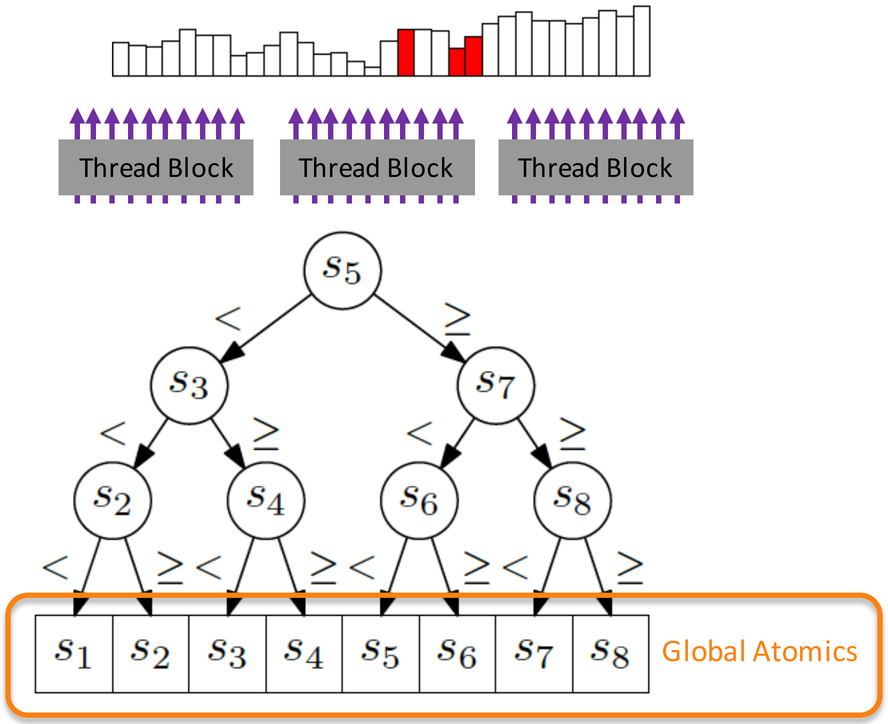
- We only copy elements of the bucket we are interested in;
- In case of identical splitter elements, they are placed in an *equality bucket*;
- If target rank is in an *equality bucket*, the algorithm can terminate early by returning lower bound;
- For sorting the splitters, small input datasets, and the lowest recursion level a *bitonic sort* in shared memory is used;
- Use a *binary search tree* to sort elements into the buckets;

SampleSelect Algorithm

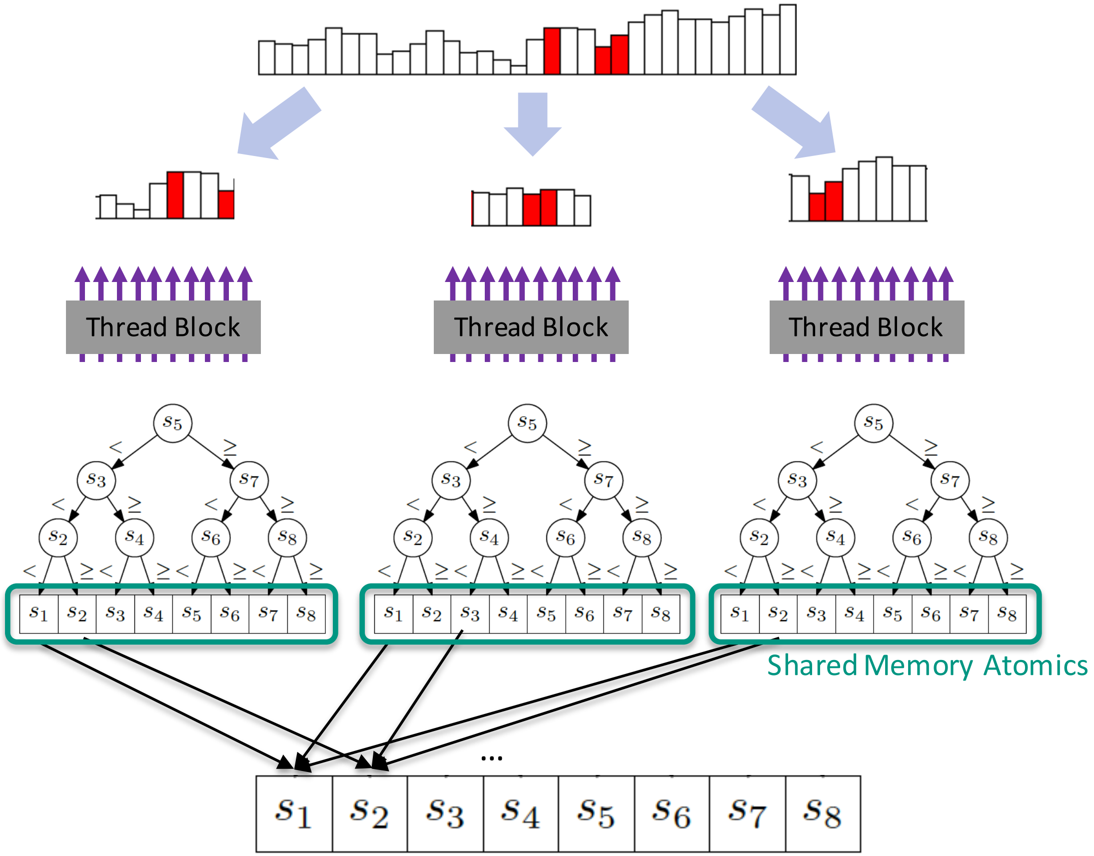


Parallel Threshold Selection on GPUs

Global Memory Atomics



Shared Memory Atomics



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

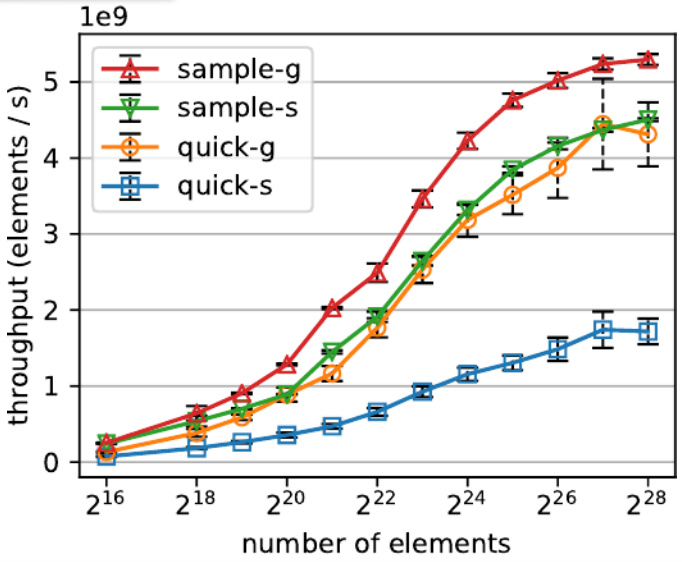
- Split data set into chunks, assign to thread blocks;
- Each thread block runs bucket count on its data;
- Use a global reduction to get global bucket counts;

Parallel Threshold Selection on GPUs

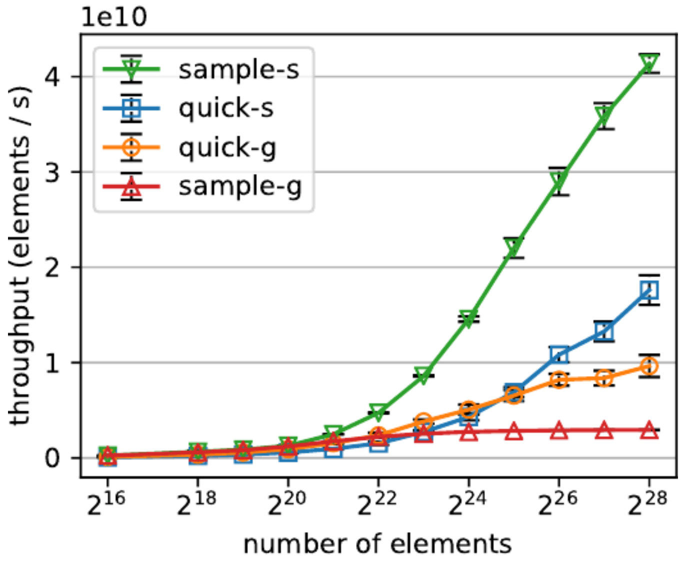
-g : global memory atomics
-s: shared memory atomics

single precision

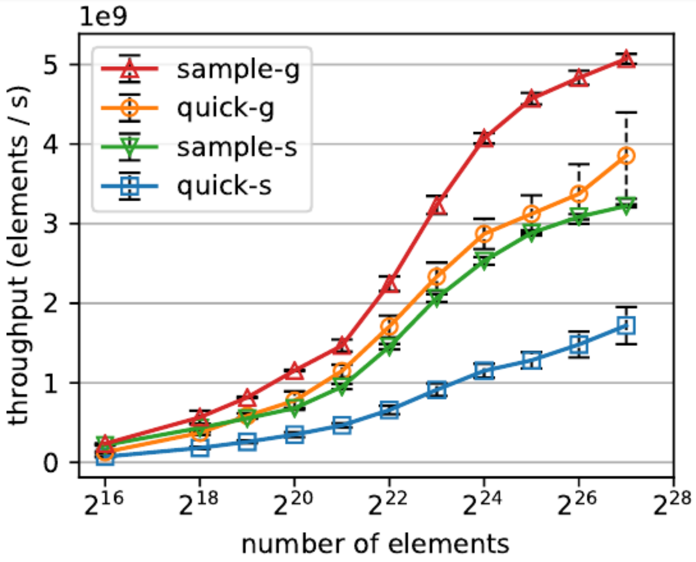
NVIDIA K40

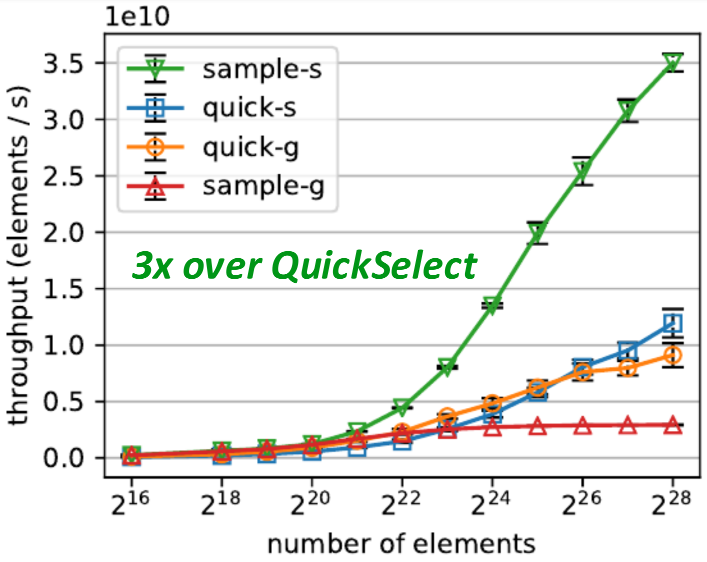


NVIDIA V100



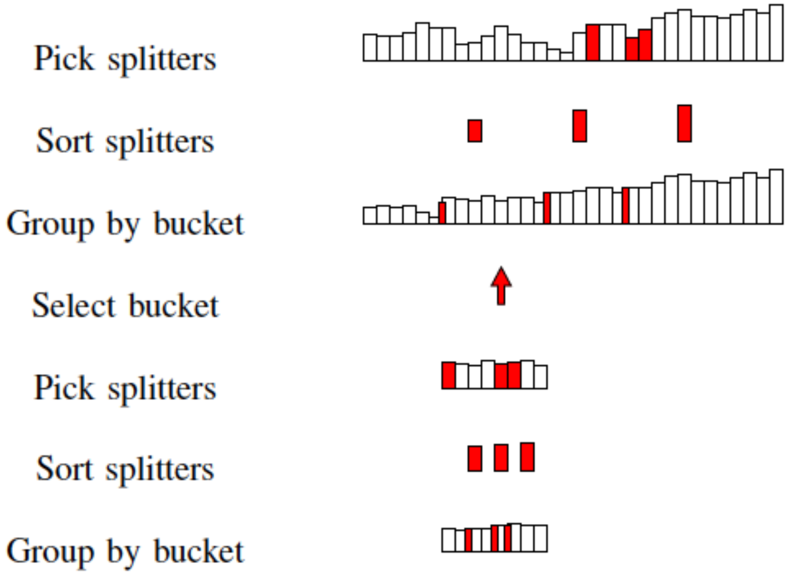
double precision





Approximate Threshold Selection

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

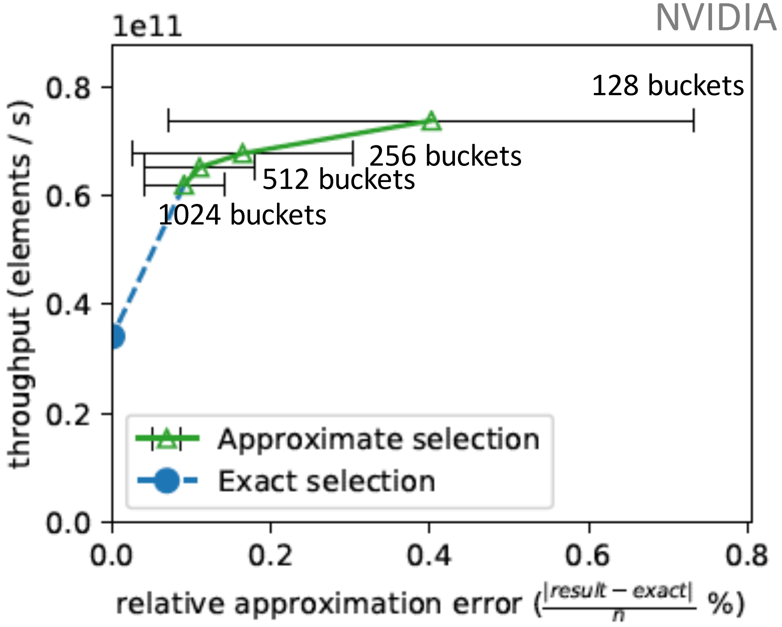
tobias.ribizel@student.kit.edu, hartwig.anzt@kit.edu

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

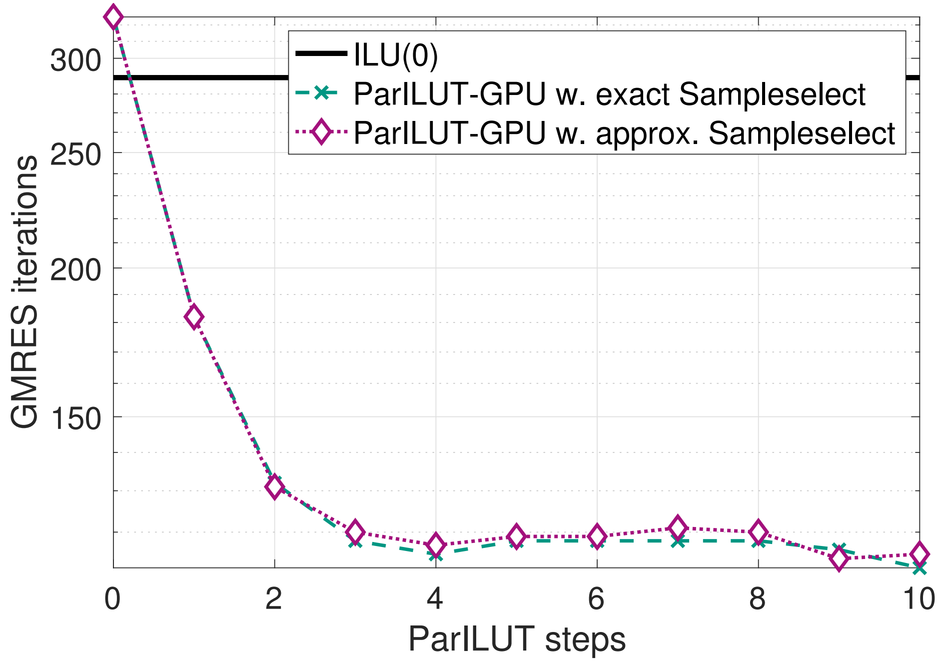


Approximate selection on 2^{28} uniformly distributed single precision values using 1 recursion level, only.

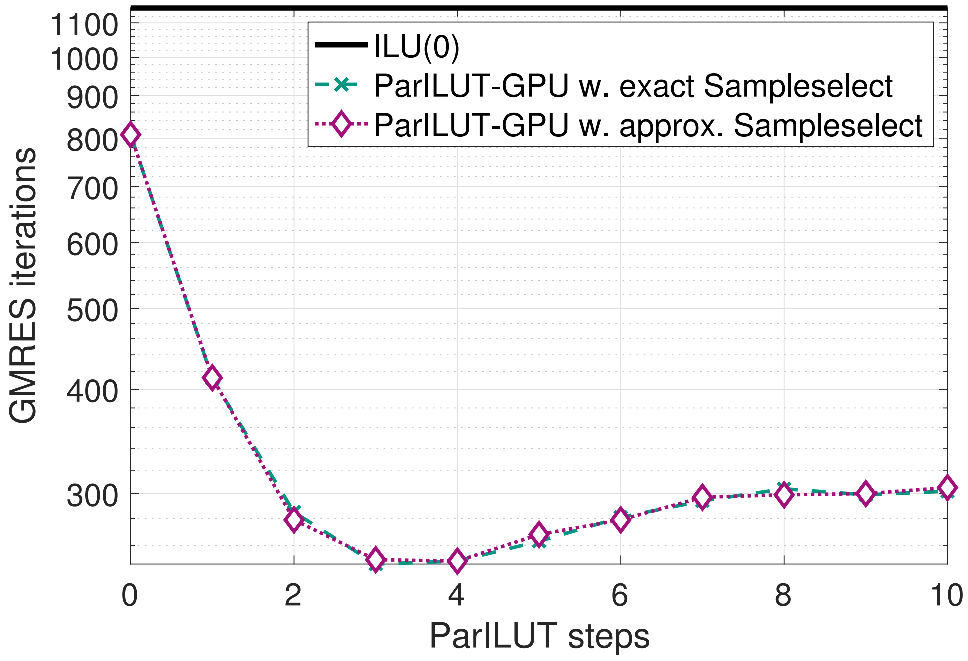
Approximate Threshold Selection

Impact of exact/approximate SampleSelect on ParILUT preconditioner quality

ANI5



ANI6



Parallelism inside the blocks: Candidate search

Identify locations with nonzero ILU residual.

Identify locations that are symbolically nonzero:

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

sparse matrix product

sparse matrix sum

sparse matrix sum

- 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.

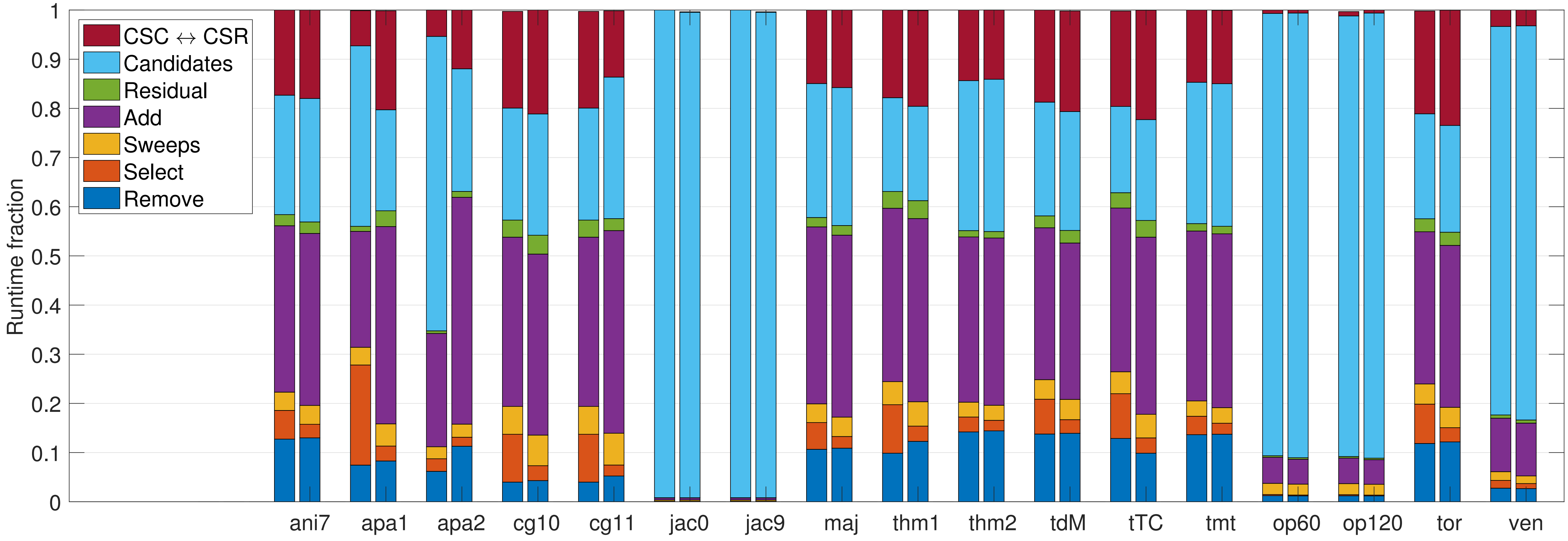
- *Cost heavily dependent on sparsity pattern.*
- *Kernel performance bound by memory bandwidth.*
- ***Design specialized Kernel*** [2]**.**

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

ParILUT Performance on GPUs

Impact of exact(1st bar) / approximate (2nd bar) SampleSelect on ParILUT runtime breakdown

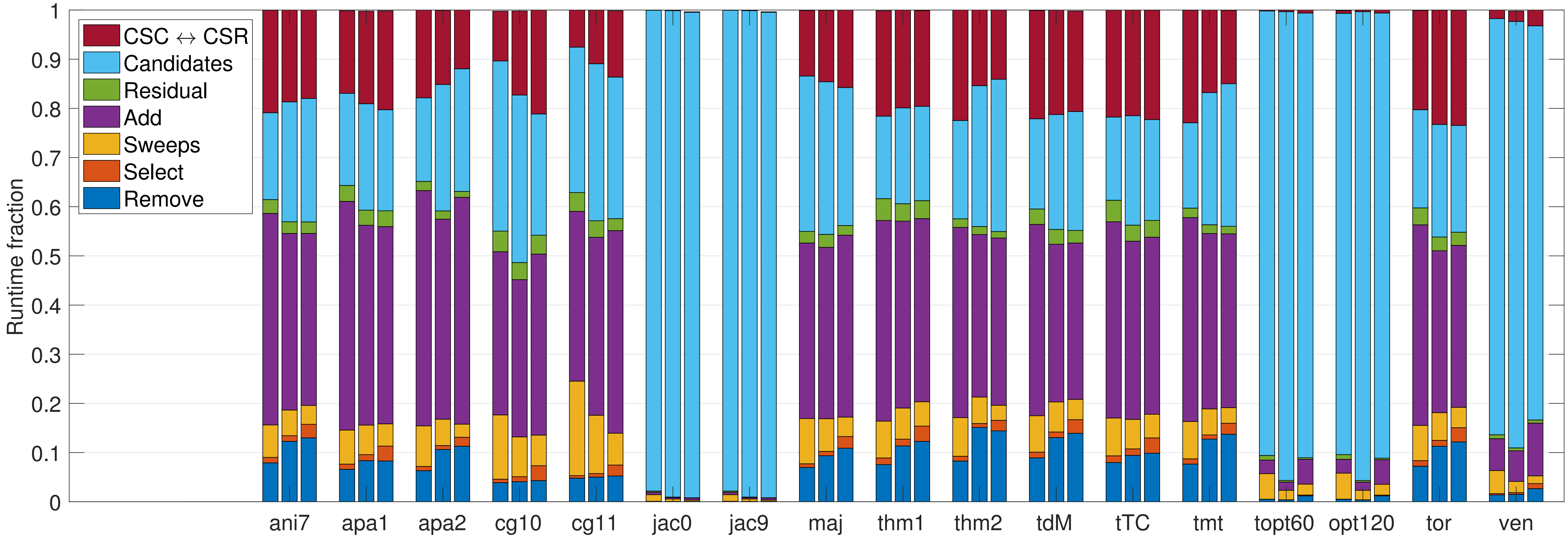
NVIDIA V100 GPU.
Matrices taken from Suite Sparse Matrix Collection.



ParILUT Performance on 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.



The Challenge: Iterative Solution of a Sparse Linear System

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.

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

The Challenge: Iterative Solution of a Sparse Linear System

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.

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

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

$$MAx = Mb \text{ (left preconditioned)}$$

The Challenge: Iterative Solution of a Sparse Linear System

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.

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

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

$$MAx = Mb \text{ (left preconditioned)}$$

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 Challenge: Iterative Solution of a Sparse Linear System

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.

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

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

$$MAx = Mb \text{ (left preconditioned)}$$

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…*

Explicitly forming MA is very expensive. The preconditioner is usually applied implicitly in the different iteration steps.

The Challenge: Iterative Solution of a Sparse Linear System

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.

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

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

$$MAx = Mb \text{ (left preconditioned)}$$

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…*

Explicitly forming MA is very expensive. The preconditioner is usually applied implicitly in the different iteration steps.

Instead of forming the preconditioner $M \approx A^{-1}$ explicitly, Incomplete Factorization Preconditioners (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:

$$\begin{aligned} z_{k+1} &= M r_{k+1} \\ M^{-1} z_{k+1} &= r_{k+1} \\ L \underbrace{U z_{k+1}}_{=:y} &= r_{k+1} \end{aligned}$$

$$\Rightarrow Ly = r_{k+1}, \quad U z_{k+1} = y$$

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	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	–	–

Performance

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

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

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	--	

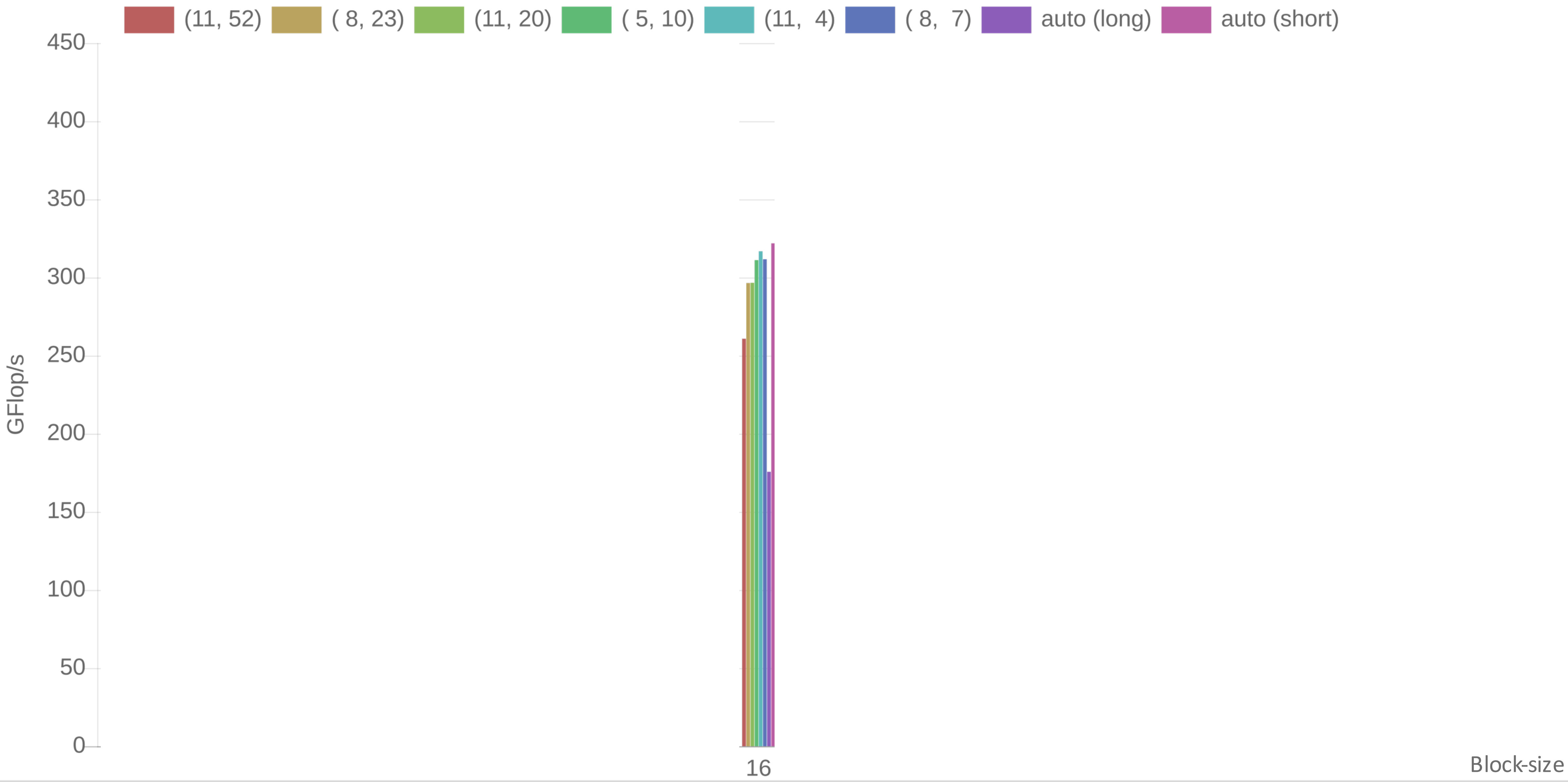
We thank Sherry Li and Meiyue Shao for technical help in generating the performance numbers.

Precision distribution in Adaptive Block-Jacobi

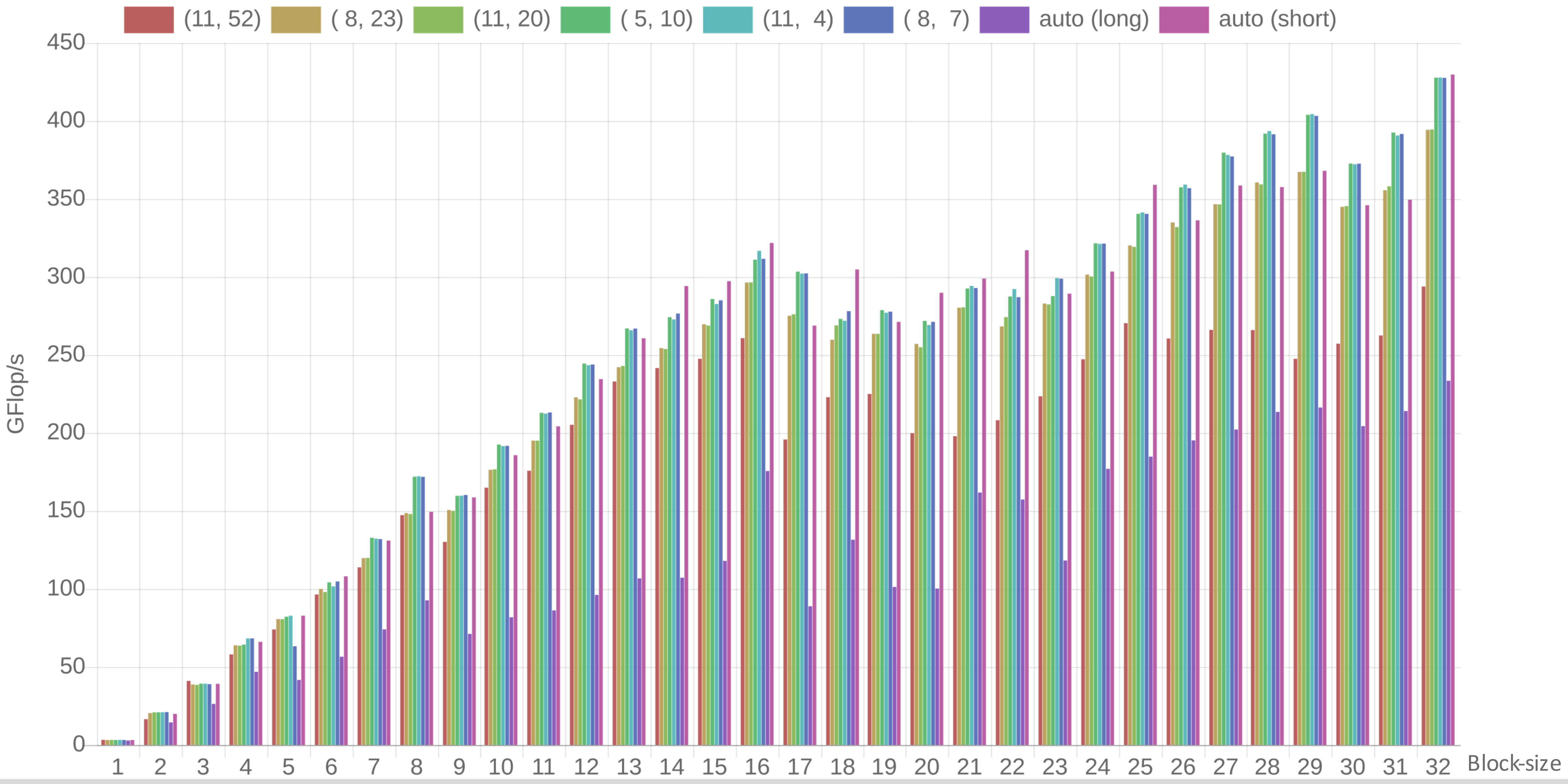
Precision distribution in Adaptive Block-Jacobi

NVIDIA V100 GPU

Adaptive Block-Jacobi Generation

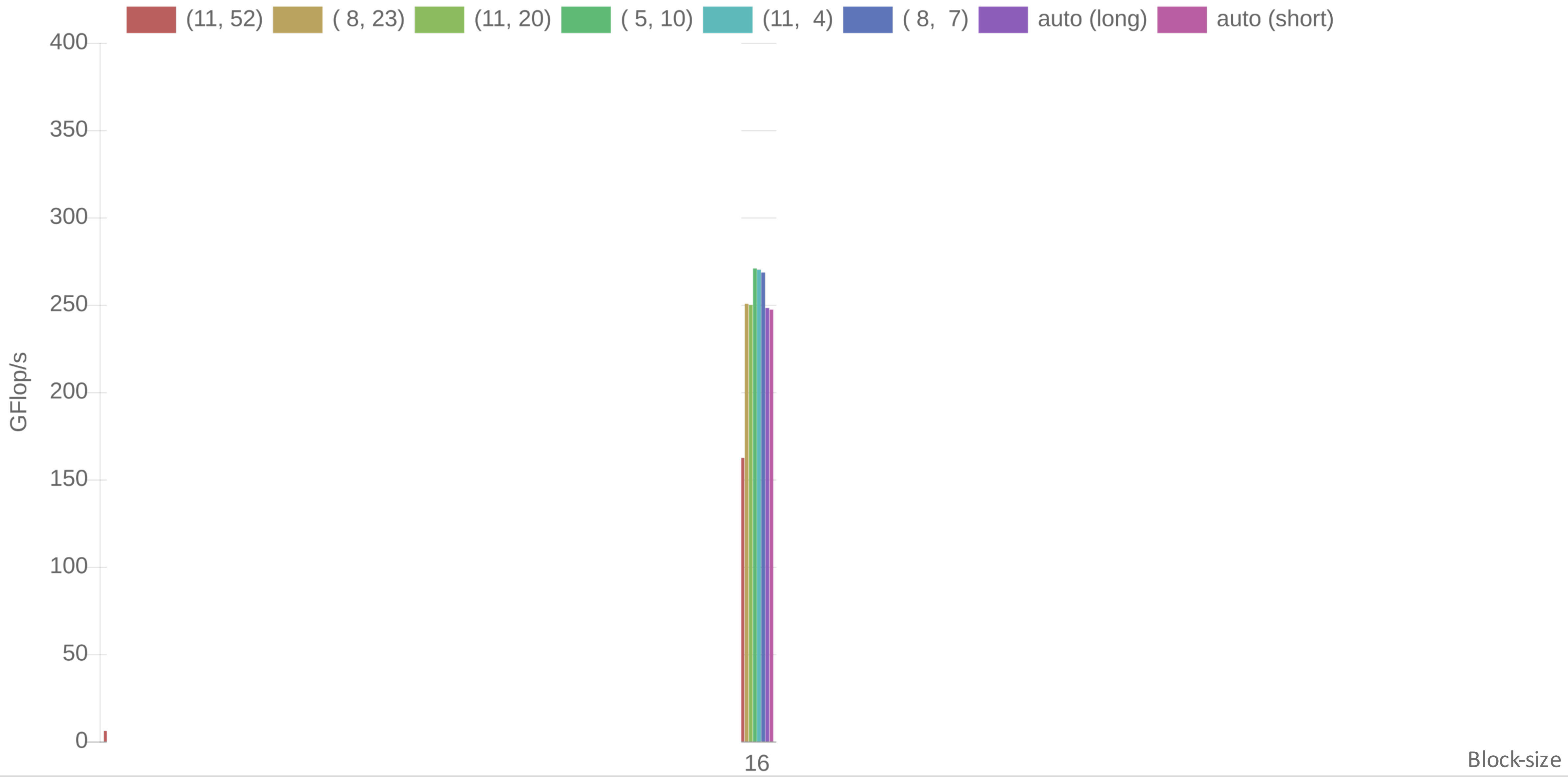


Adaptive Block-Jacobi Generation



Adaptive Block-Jacobi Application

NVIDIA V100 GPU



Adaptive Block-Jacobi Application