Scalability Analysis of Sparse Matrix Computations on Many-core Processors

Weifeng Liu

Norwegian University of Science and Technology, Norway

NTNU

Science & Technology Facilities Council
Rutherford Appleton Laboratory

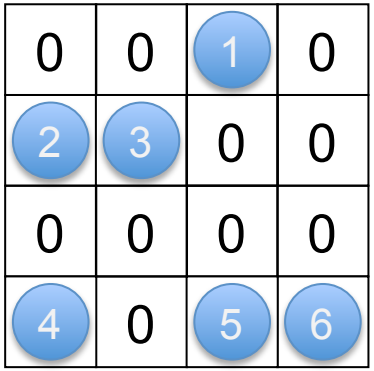
VirginiaTech
Invent the Future

TU/e Technische Universiteit Eindhoven University of Technology

September 6th – 8th, 2017, Toulouse, France

Sparse matrices

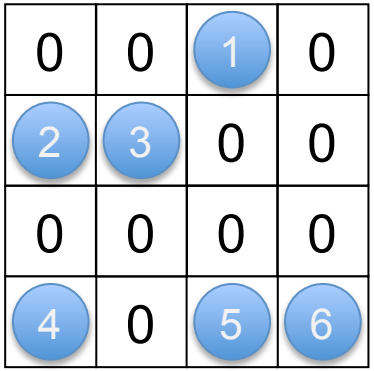
- If the majority of entries in a matrix are zeros, we can store the matrix using a sparse representation that only records nonzero entries.
- The Compressed Sparse Row (CSR) format is the most widely used.



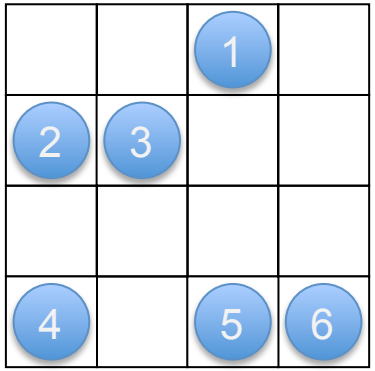
A
(4x4)
dense

Sparse matrices

- If the majority of entries in a matrix are zeros, we can store the matrix using a sparse representation that only records nonzero entries.
- The Compressed Sparse Row (CSR) format is the most widely used.



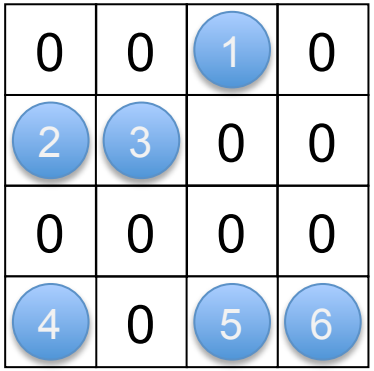
A
(4x4)
dense



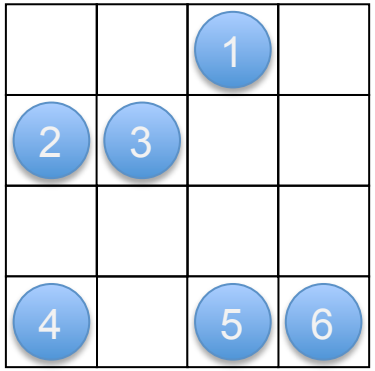
A
(4x4)
sparse, 6 nonzeros

Sparse matrices

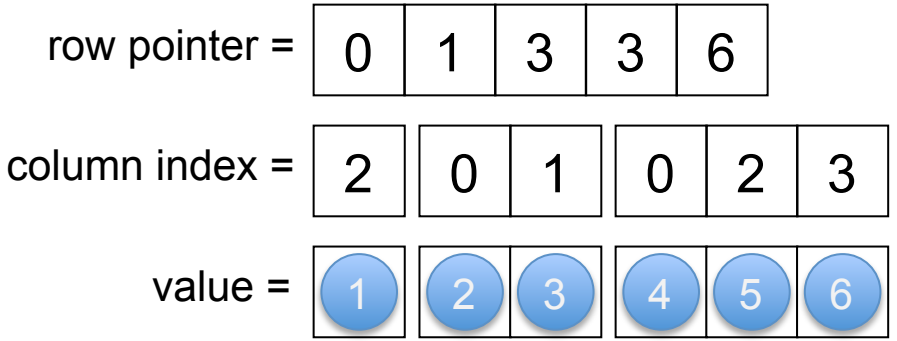
- If the majority of entries in a matrix are zeros, we can store the matrix using a sparse representation that only records nonzero entries.
- The Compressed Sparse Row (CSR) format is the most widely used.



A
(4x4)
dense

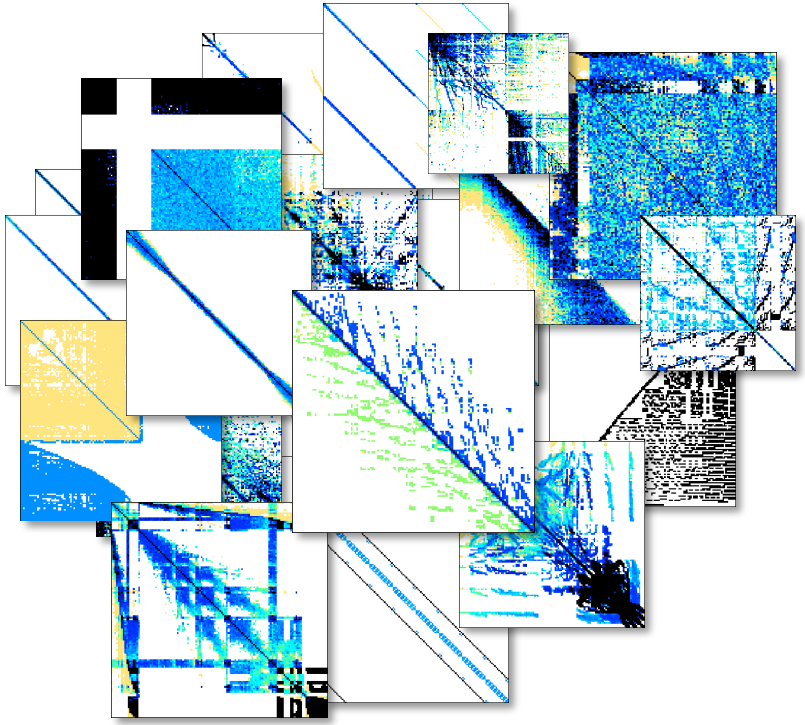


A
(4x4)
sparse, 6 nonzeros

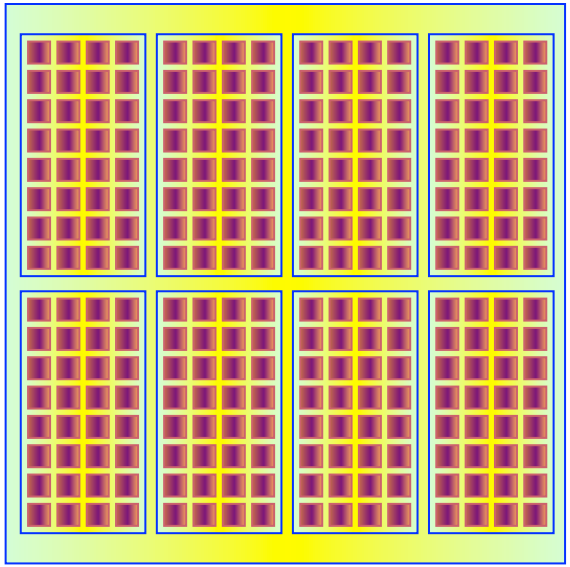


A (in CSR)
(4x4)
6 nonzeros

Sparse matrices on many-core processors

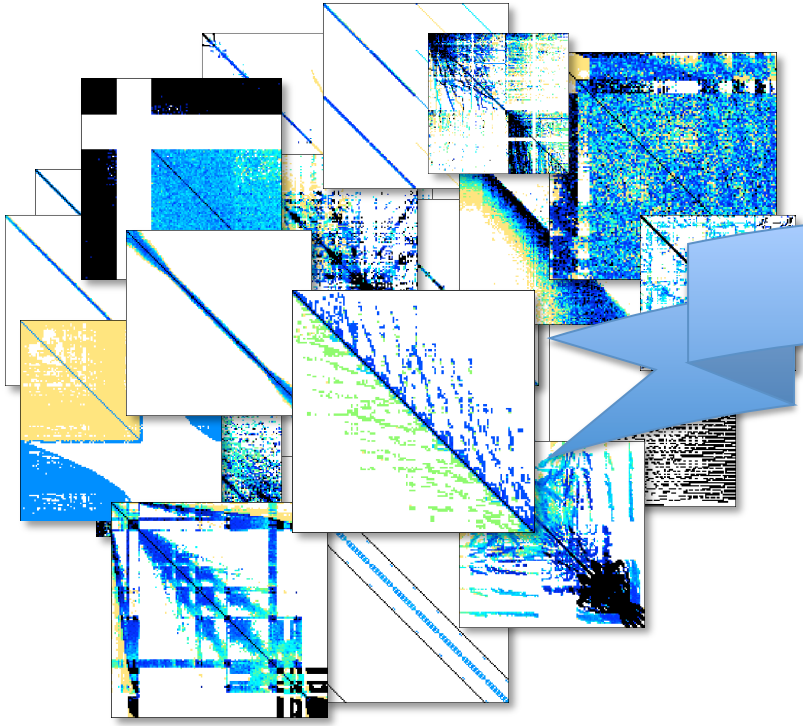


Various sparsity structures



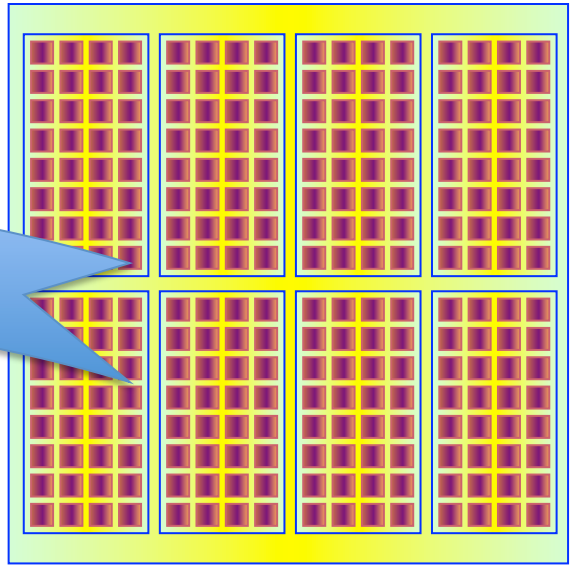
Many cores with a large amount of “small” SIMD units

Sparse matrices on many-core processors



Various sparsity structures

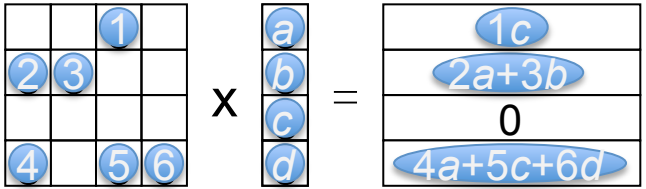
Scalable high performance?



Many cores with a large amount of “small” SIMD units

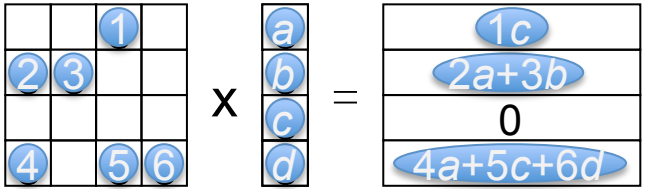
Four kernels used to study scalability

- Sparse matrix-vector Multiplication (SpMV)

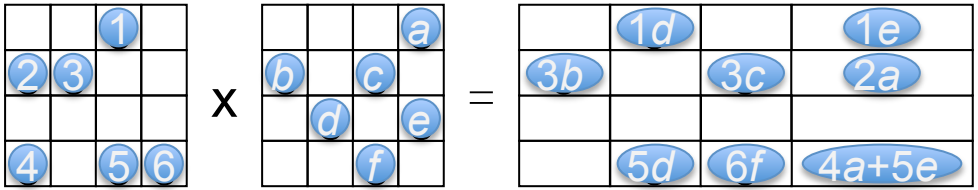


Four kernels used to study scalability

- Sparse matrix-vector Multiplication (SpMV)

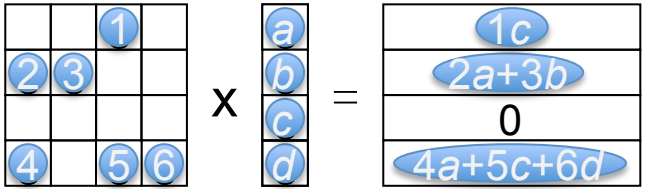


- Sparse matrix-matrix Multiplication (SpGEMM)

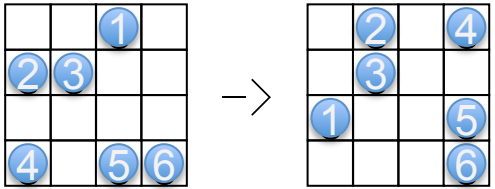


Four kernels used to study scalability

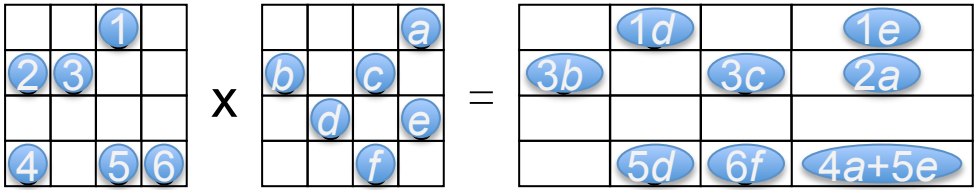
- Sparse matrix-vector Multiplication (SpMV)



- Sparse transposition (SpTRANS)

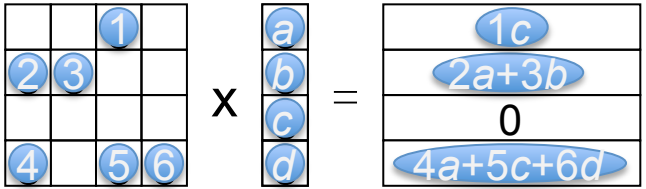


- Sparse matrix-matrix Multiplication (SpGEMM)

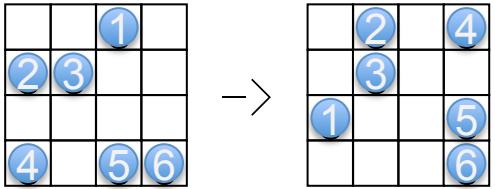


Four kernels used to study scalability

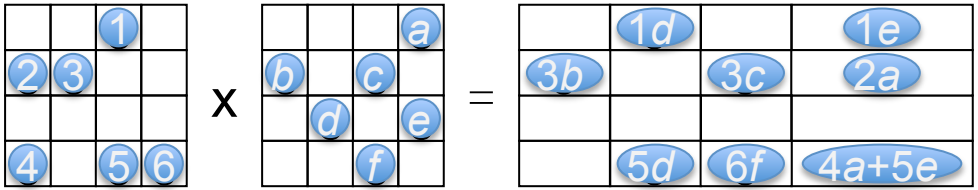
- Sparse matrix-vector Multiplication (SpMV)



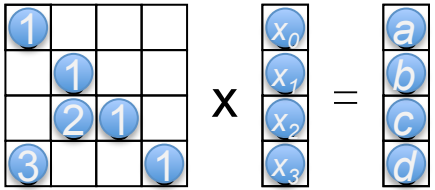
- Sparse transposition (SpTRANS)



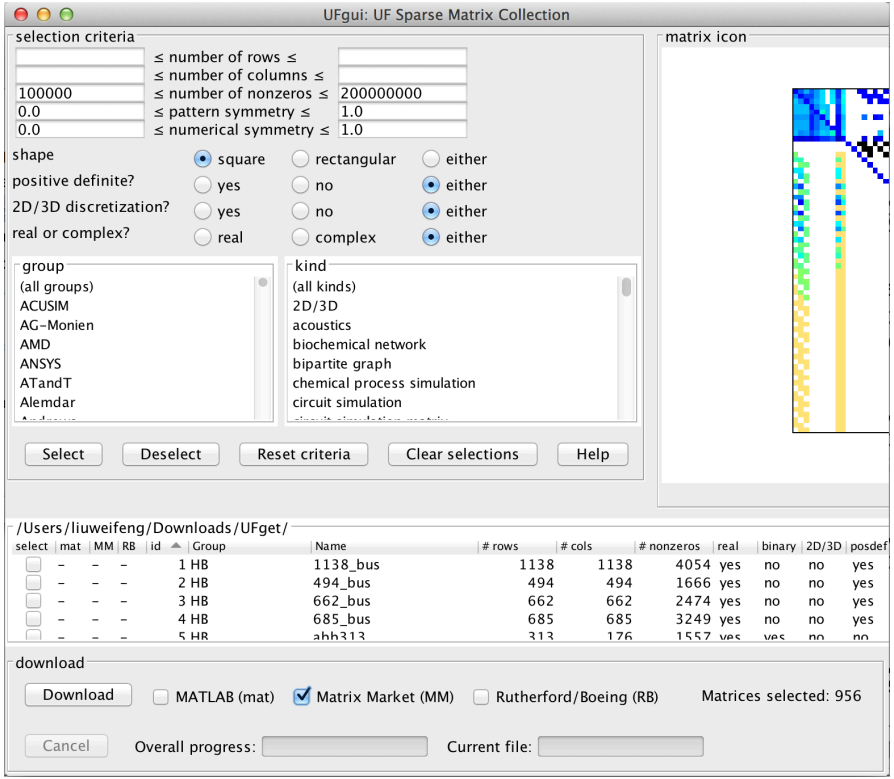
- Sparse matrix-matrix Multiplication (SpGEMM)



- Sparse triangular solve (SpTRSV)



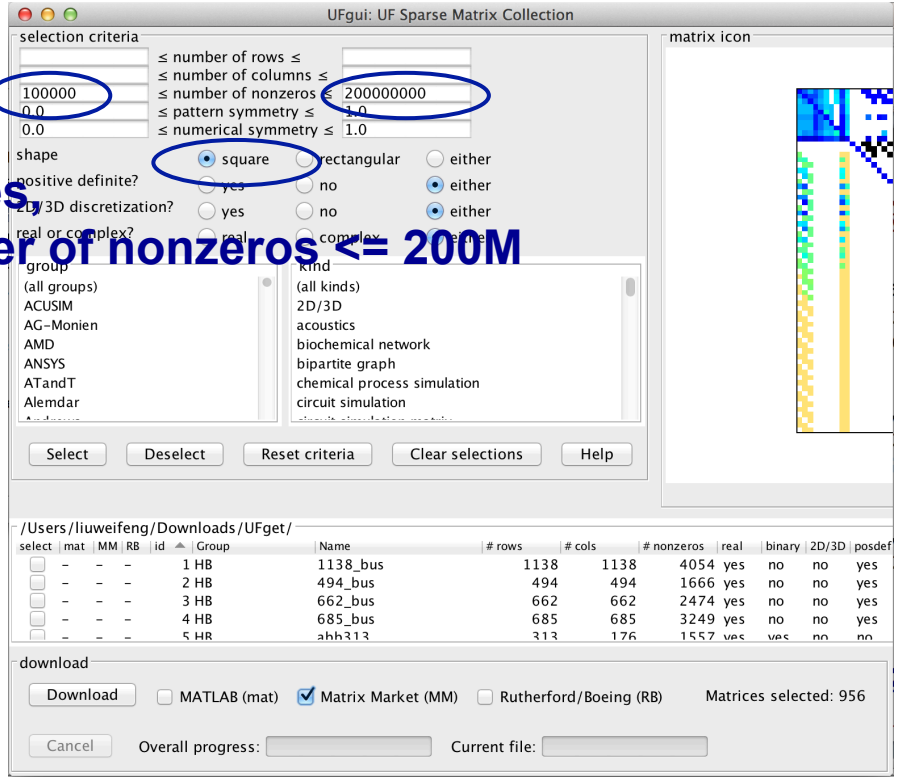
956 matrices used to study scalability



NTNU

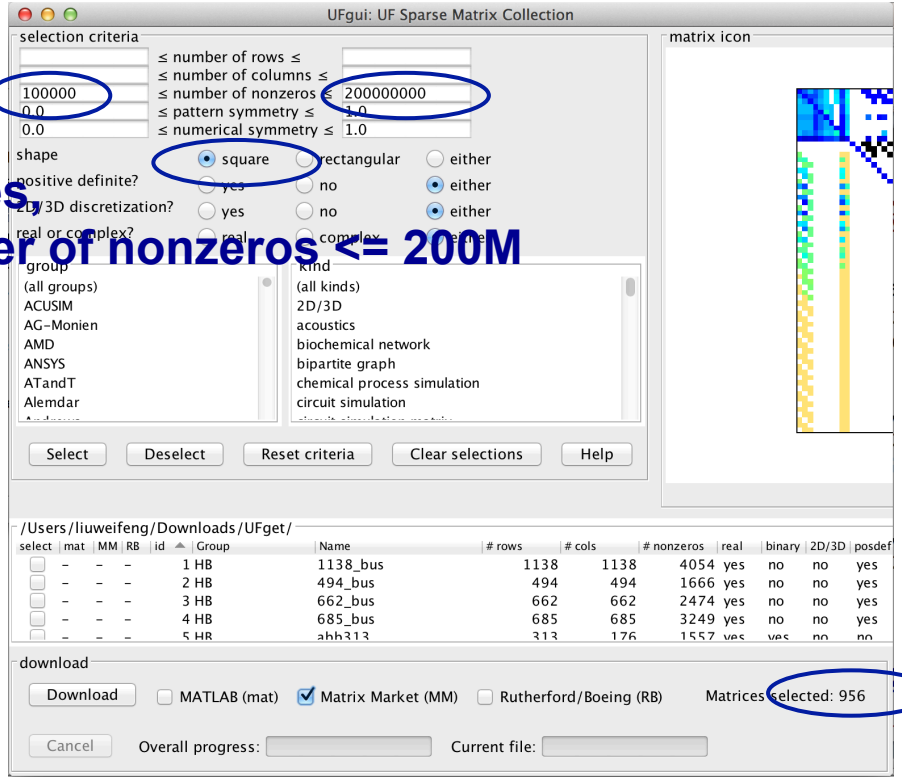
956 matrices used to study scalability

square matrices,
100K <= number of nonzeros <= 200M



956 matrices used to study scalability

square matrices,
100K <= number of nonzeros <= 200M



956 matrices from in total 2757 matrices

NTNU

Three GPUs used to study strong scalability

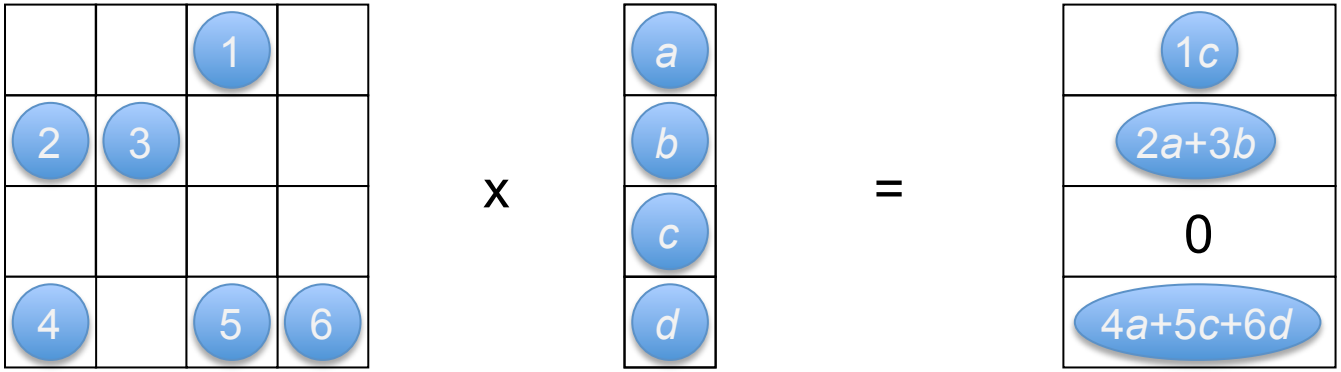
- **nVidia GeForce Titan X**, Pascal GP102, 3584 CUDA cores @ 1.4 GHz, 10.1 Tflops (SP), 3 MB L2 cache, 12GB GDDR5X, 480 GB/s B/W.
- **nVidia GeForce GTX 1080**, Pascal GP104, 2560 CUDA cores @ 1.6 GHz, 8.2 Tflops (SP), 2 MB L2 cache, 8GB GDDR5X, 320 GB/s B/W.
- **nVidia GeForce GTX 1060**, Pascal GP106, 1280 CUDA cores @ 1.6 GHz, 4.1 Tflops (SP), 1.5 MB L2 cache, 6GB GDDR5, 192 GB/s B/W.

Ratio of peak perf. 1 : 2 : 2.5 **Ratio of bandwidth 1.2 : 2 : 3**

Kernel 1. Sparse Matrix-Vector Multiplication (SpMV)

SpMV

- Multiply a sparse matrix *A* by a dense vector *x*, and obtain a resulting dense vector *y*.



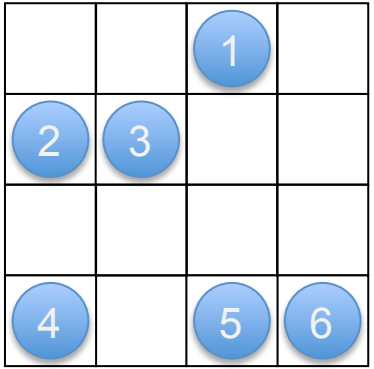
A
(4x4)
sparse, 6 nonzeros

x
(4x1)
dense

y
(4x1)
dense

Parallel SpMV (using the CSR format)

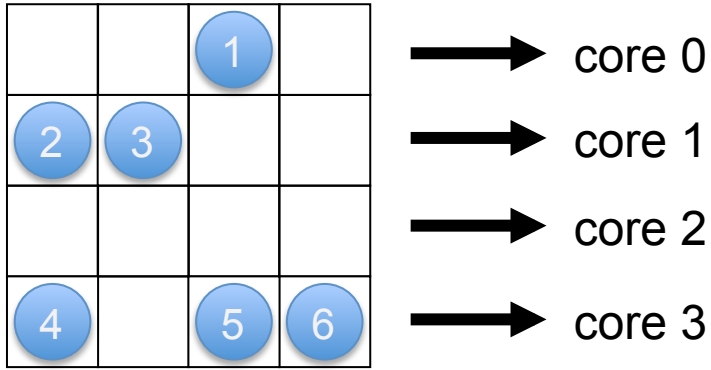
- Assign a row block to one core of a many-core processor.



Nathan Bell, Michael Garland. **Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors**. *SC09*.

Parallel SpMV (using the CSR format)

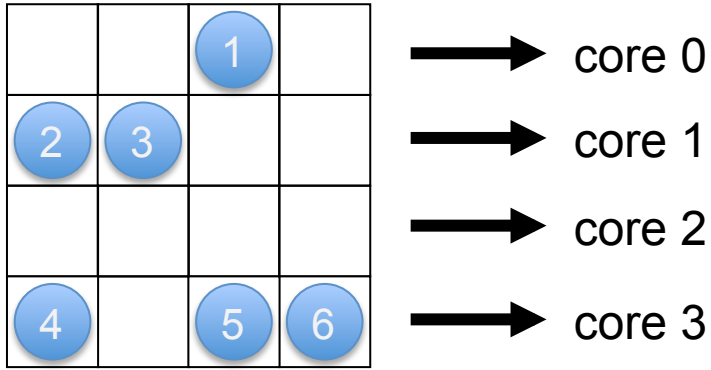
- Assign a row block to one core of a many-core processor.



Nathan Bell, Michael Garland. **Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors**. *SC09*.

Parallel SpMV (using the CSR format)

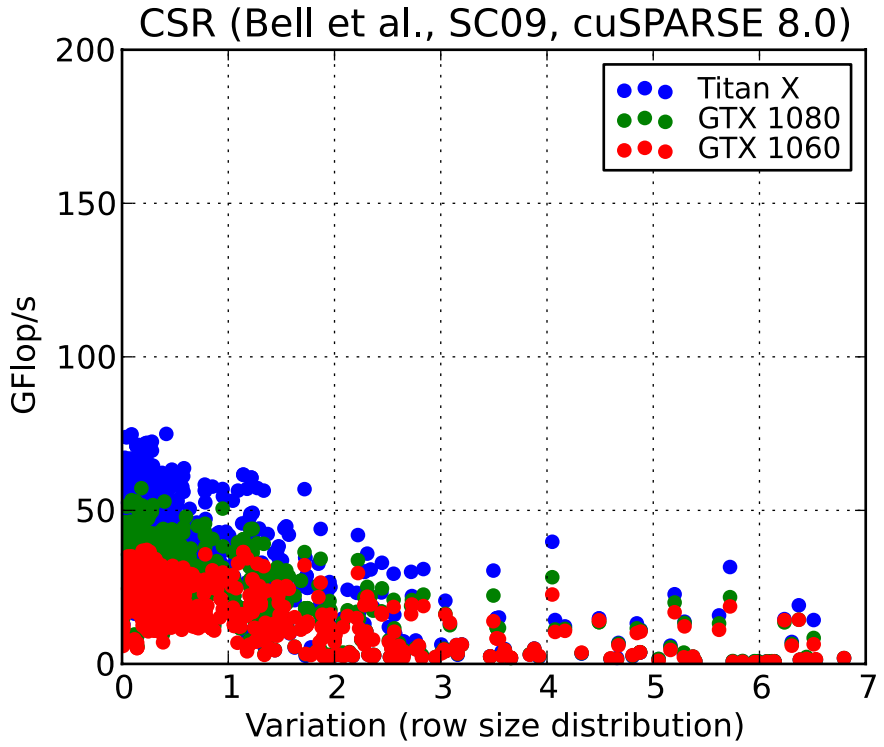
- Assign a row block to one core of a many-core processor.



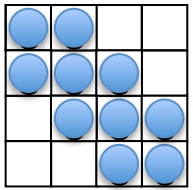
Probably imbalanced computation, degraded performance on many-core processors

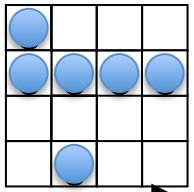
Nathan Bell, Michael Garland. **Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors**. *SC09*.

Scalability of SpMV using CSR



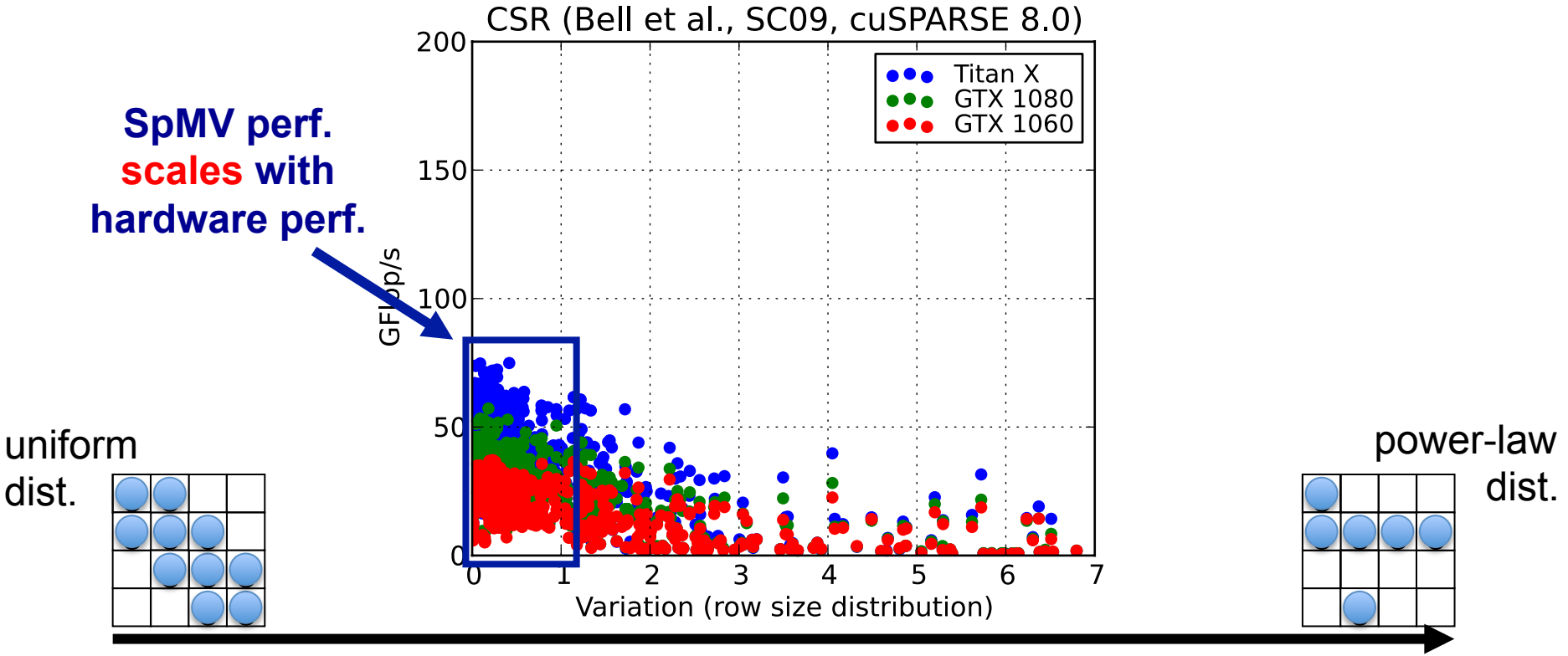
uniform dist.





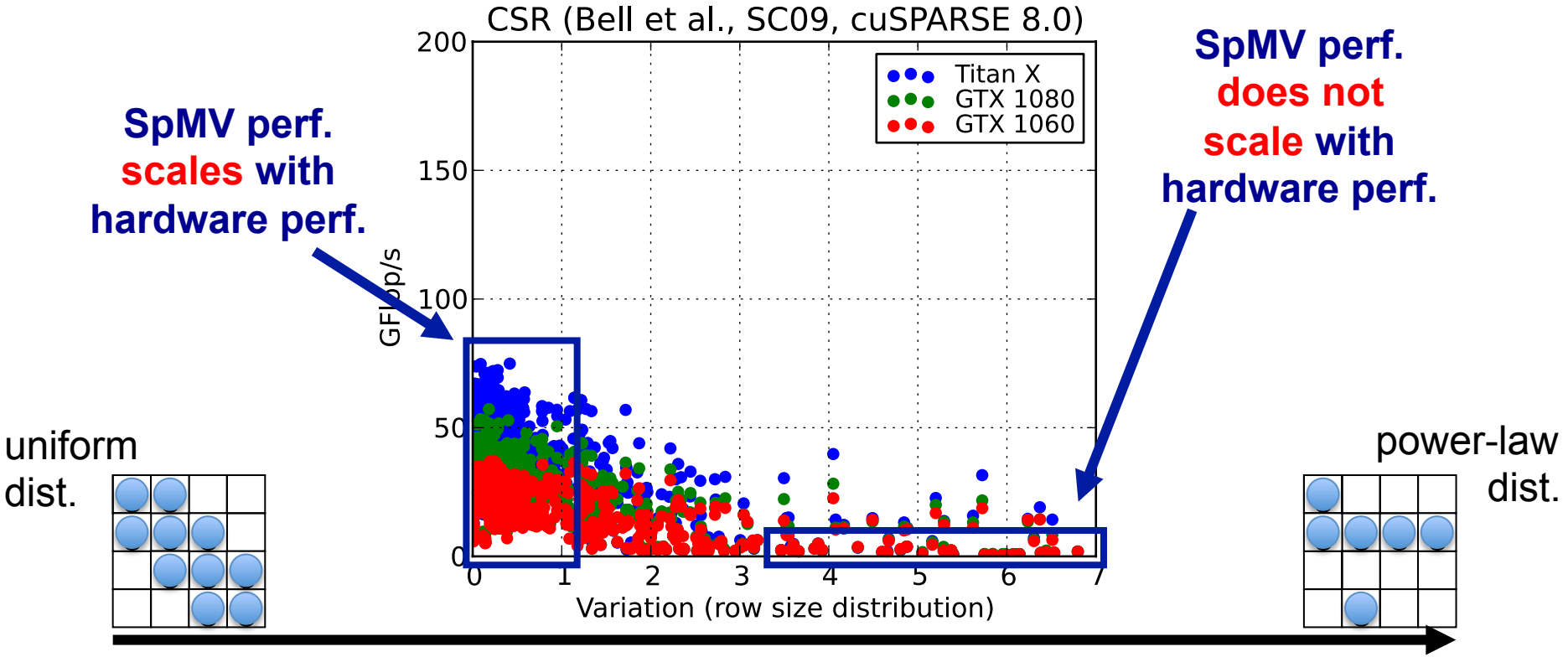
power-law dist.

Scalability of SpMV using CSR



NTNU

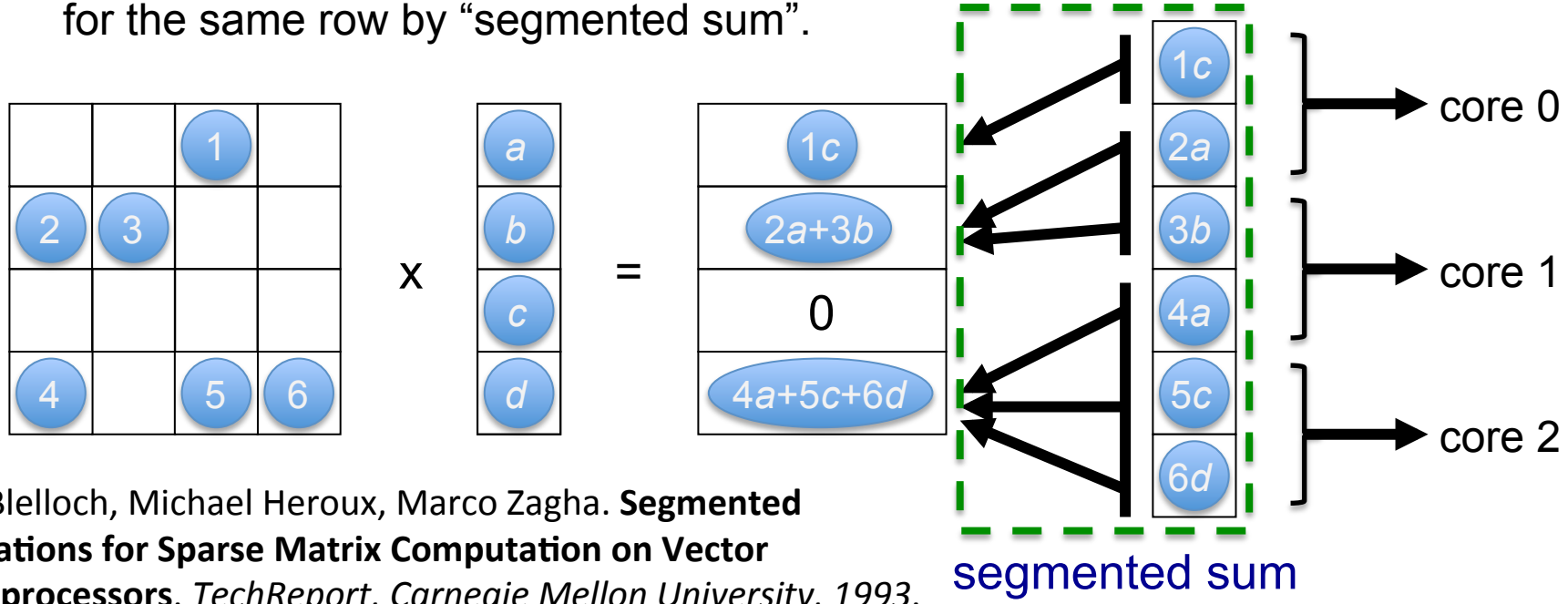
Scalability of SpMV using CSR



NTNU

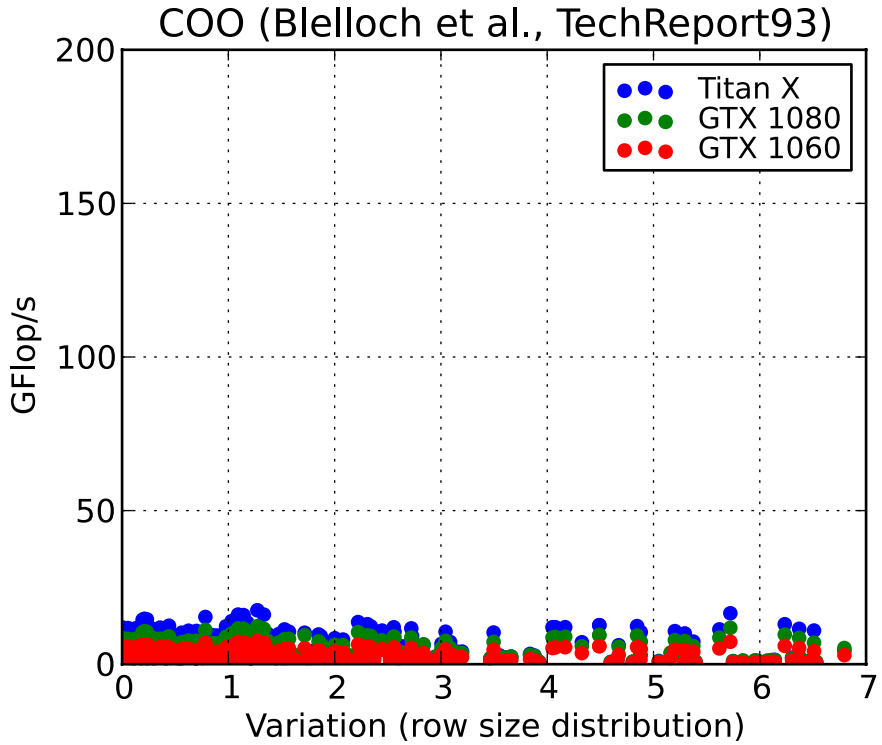
Parallel SpMV (using the COO format)

- Evenly assign nonzeros to cores, compute partial products, sum the products for the same row by “segmented sum”.

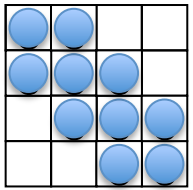


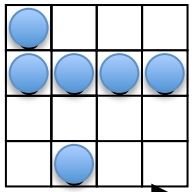
Guy Blelloch, Michael Heroux, Marco Zagha. **Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors**. *TechReport, Carnegie Mellon University, 1993*.

Scalability of SpMV using COO



uniform dist.

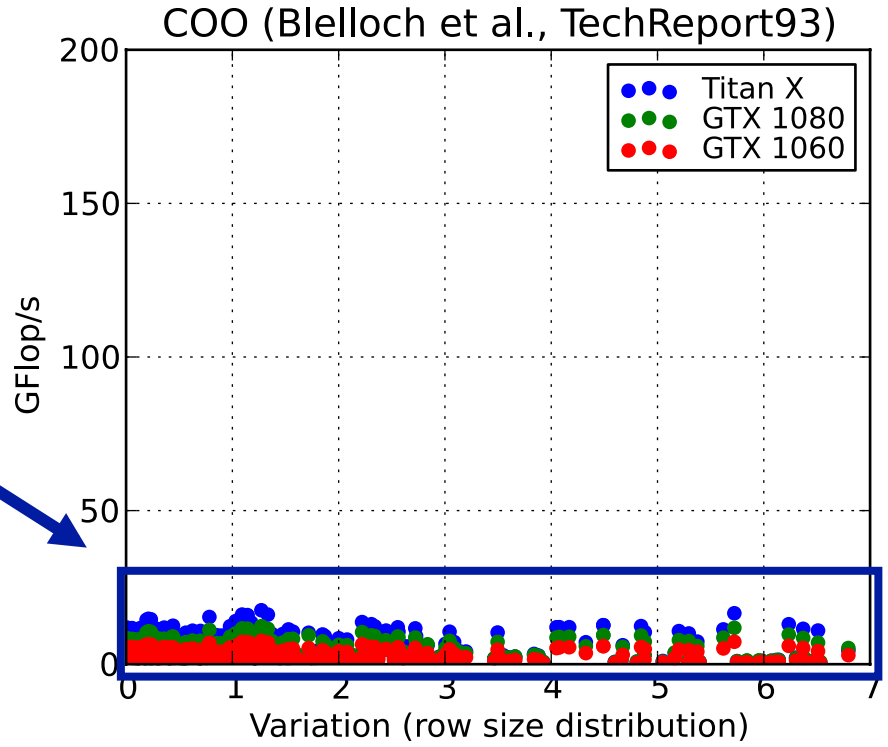




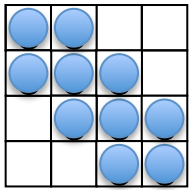
power-law dist.

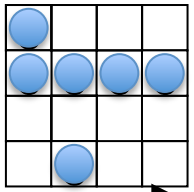
Scalability of SpMV using COO

SpMV perf. always scales with hardware perf.



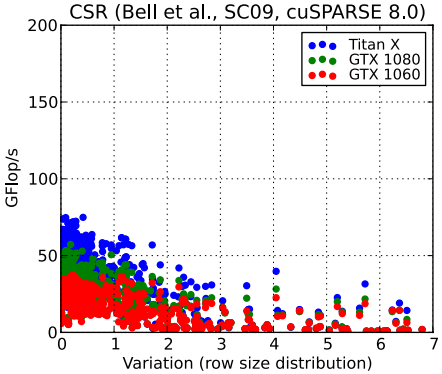
uniform dist.



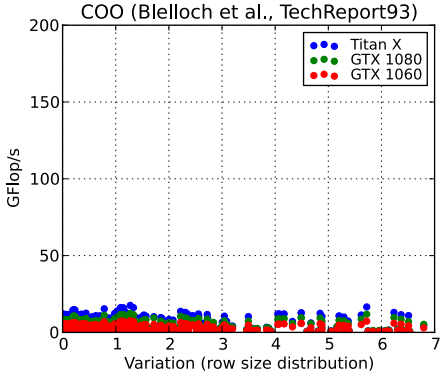


power-law dist.

Contradiction btw. scalability and high perf.

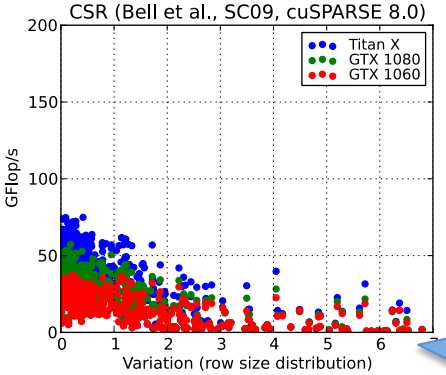


For regular matrices (uniform dist.), high performance method cannot scale.

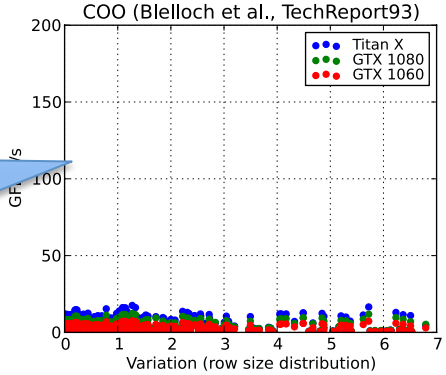


For both regular and irregular matrices (power-law dist.), scalable method gives low performance.

Contradiction btw. scalability and high perf.



Does there exist the best of both worlds (scalability + high performance)?

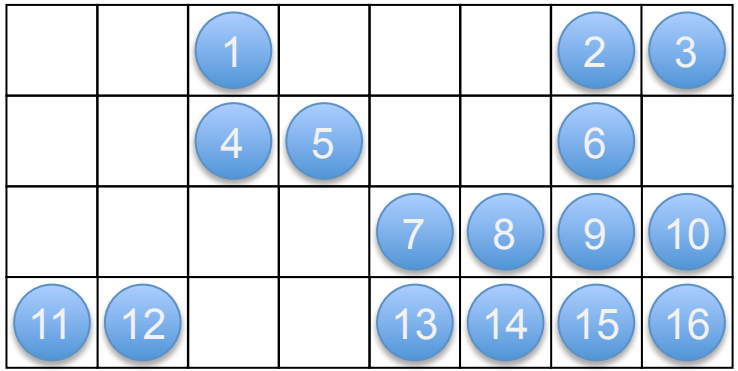


For regular matrices (uniform dist.), high performance method cannot scale.

For both regular and irregular matrices (power-law dist.), scalable method gives low performance.

yaSpMV (and the BCCOO/BCCOO+ formats)

- Store a matrix with 2D blocks, use bit-flag for row offset of reduced size, and use “segmented sum” for load balancing, auto-tune parameters.



A (4x8)

bit-flag = T F T T F

column index = 1 3 0 2 3

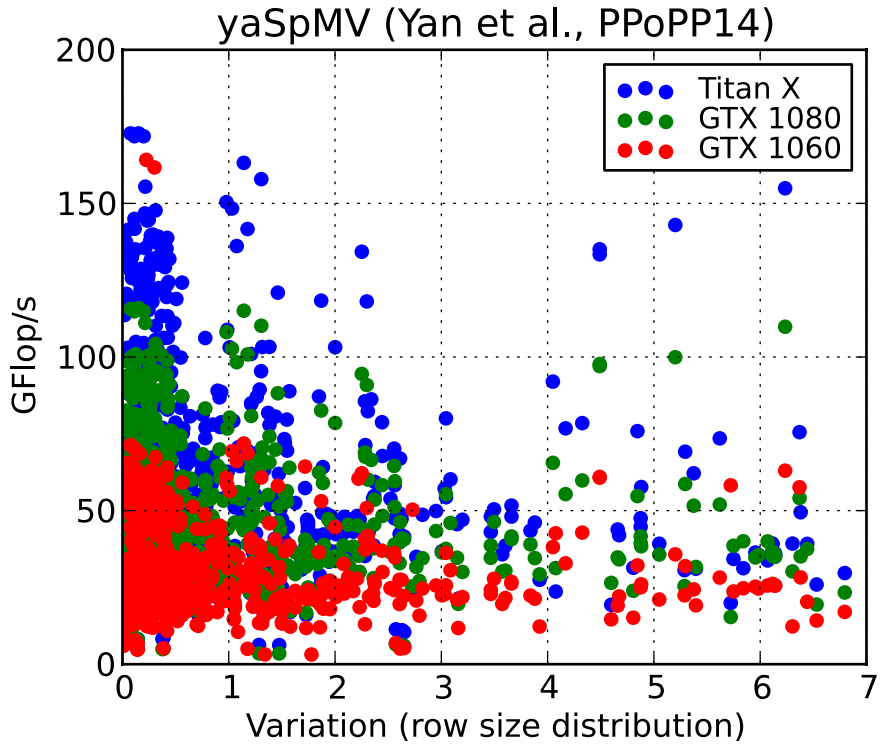
value =

1		2	3			7	8	9	10
4	5	6		11	12	13	14	15	16

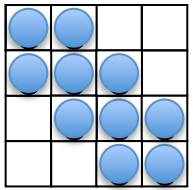
A (in the BCCOO format)

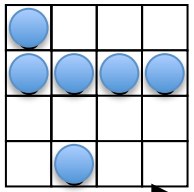
Shengen Yan, Chao Li, Yunquan Zhang, Huiyang Zhou. **yaSpMV: Yet Another SpMV Framework on GPUs**. *PPoPP14*.

Scalability of yaSpMV



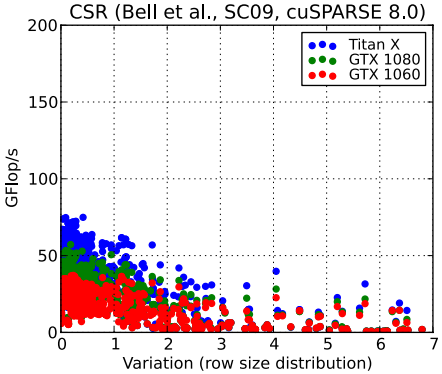
uniform dist.



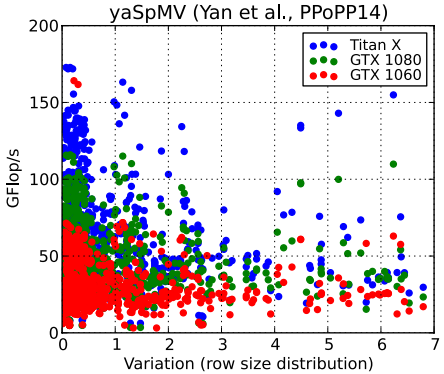


power-law dist.

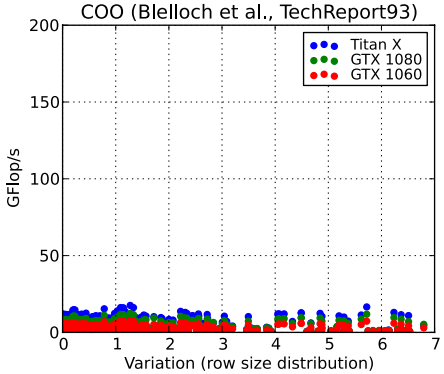
Contradiction btw. scalability and high perf.



For regular matrices (uniform dist.), high performance method cannot scale.

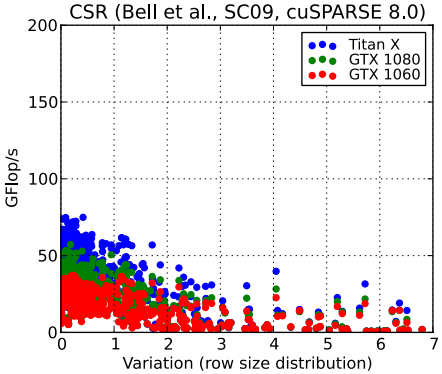


For all matrices, we now have both scalability and high performance.

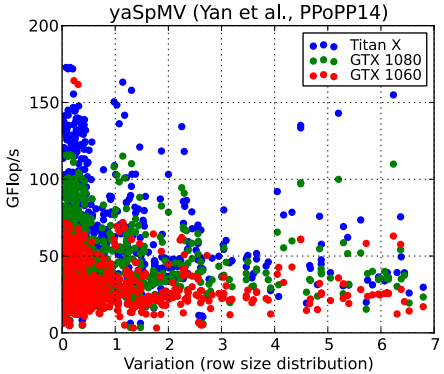


For both regular and irregular matrices (power-law dist.), scalable method gives low performance.

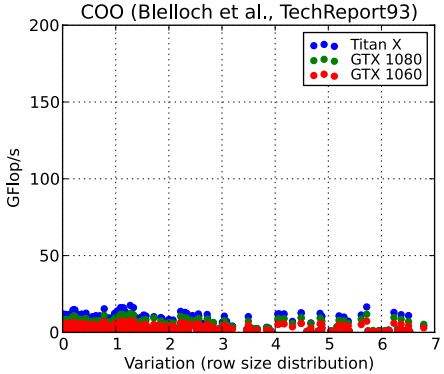
Contradiction btw. scalability and high perf.



For regular matrices (uniform dist.), high performance method cannot scale.



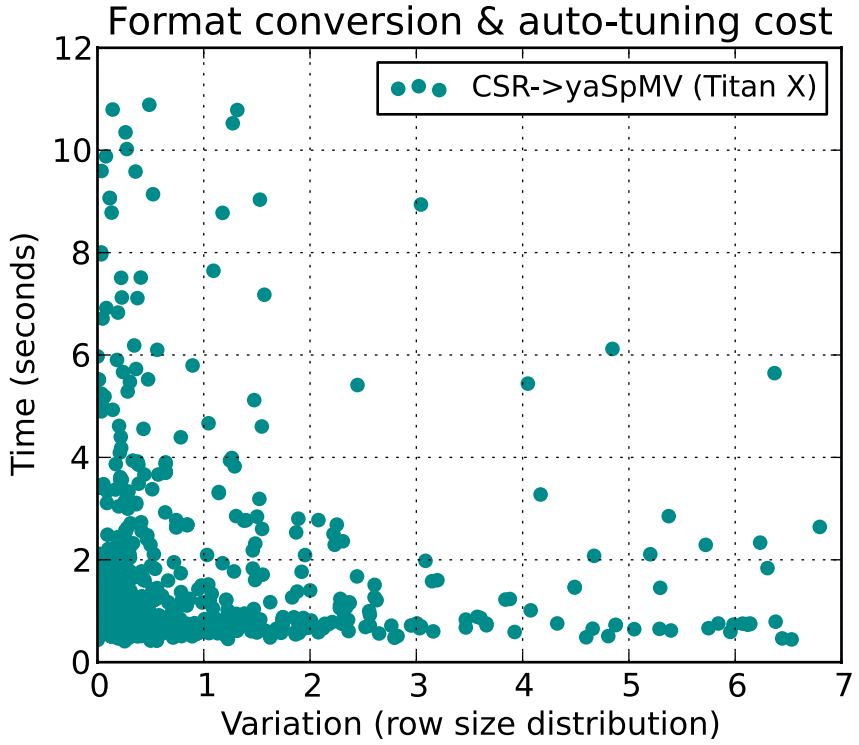
For all matrices, we now have both scalability and high performance.



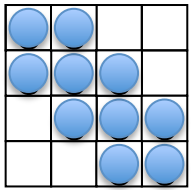
For both regular and irregular matrices (power-law dist.), scalable method gives low performance.

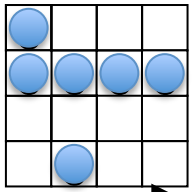
But...

Data conv. and autotuning cost of yaSpMV



uniform dist.



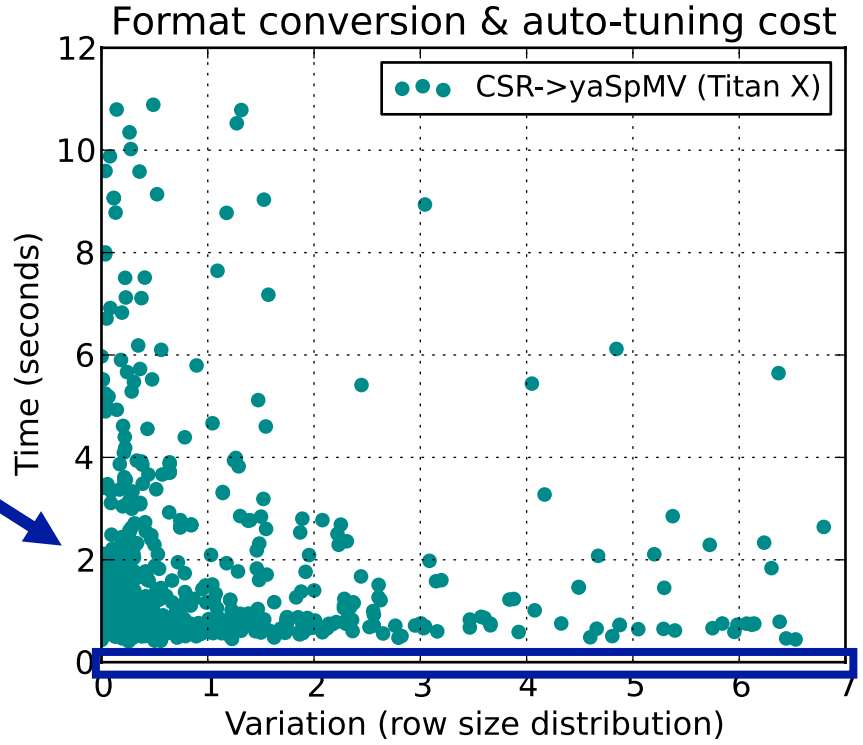


power-law dist.

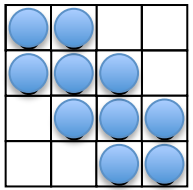
NTNU

Data conv. and autotuning cost of yaSpMV

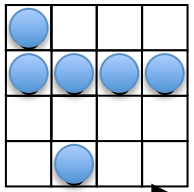
A single SpMV needs about 10^{-4} seconds.



uniform dist.

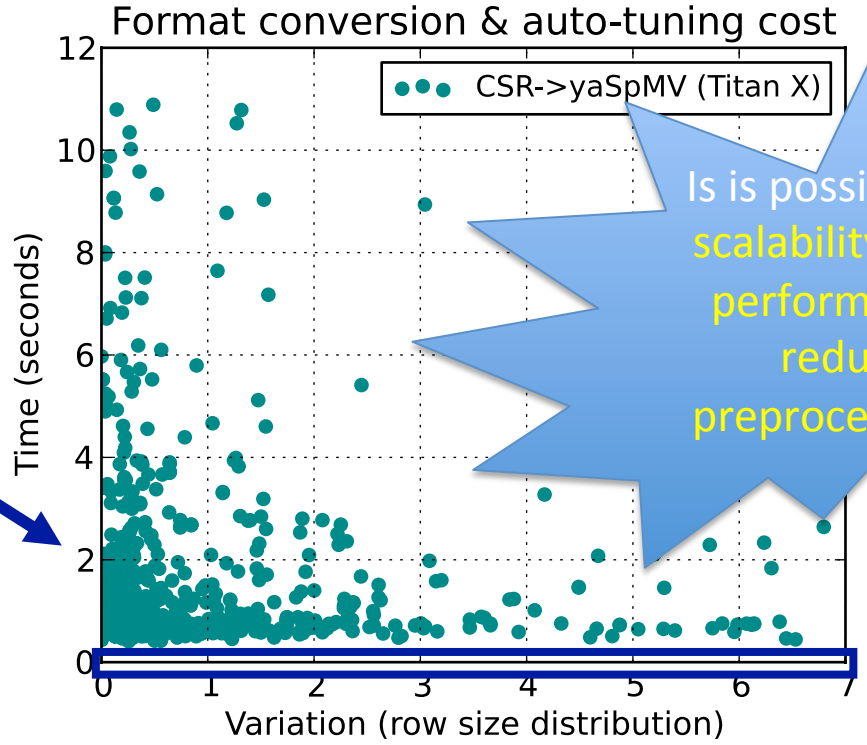


power-law dist.



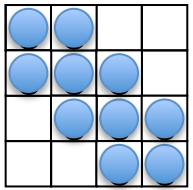
Data conv. and autotuning cost of yaSpMV

A single SpMV needs about 10^{-4} seconds.

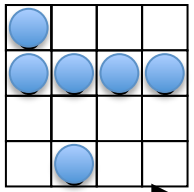


Is is possible to keep scalability and good performance, and reduce the preprocessing cost?

uniform dist.



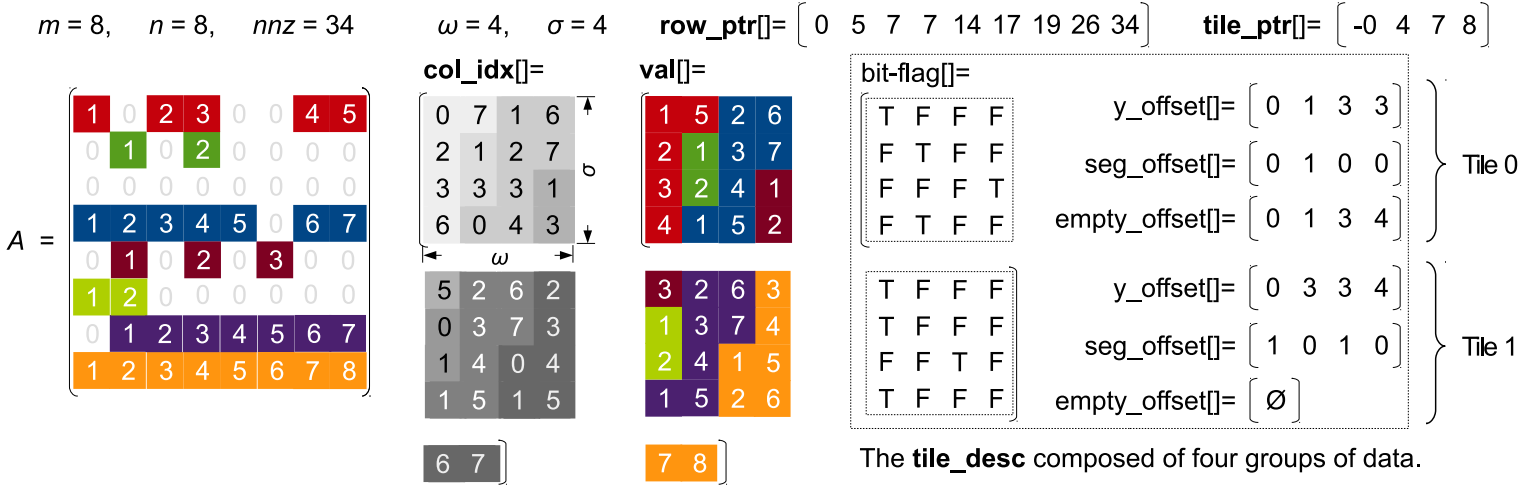
power-law dist.



NTNU

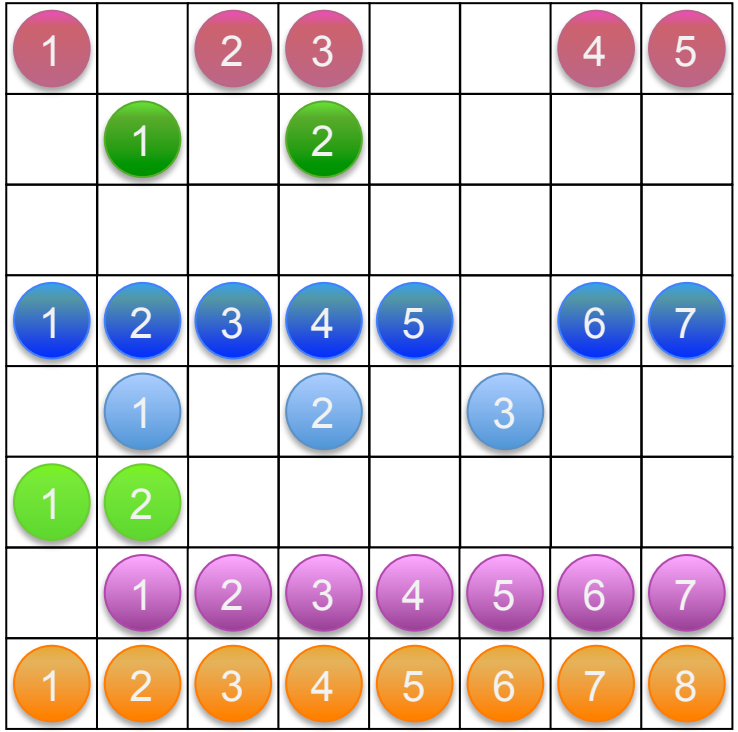
The CSR5 format (our work)

- Organize nonzeros in Tiles of identical size. The design objectives include load balancing, SIMD-friendly, low preprocessing cost and reduced storage space.



Weifeng Liu, Brian Vinter. **CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication**. *ICS15*.

CSR5: tiling

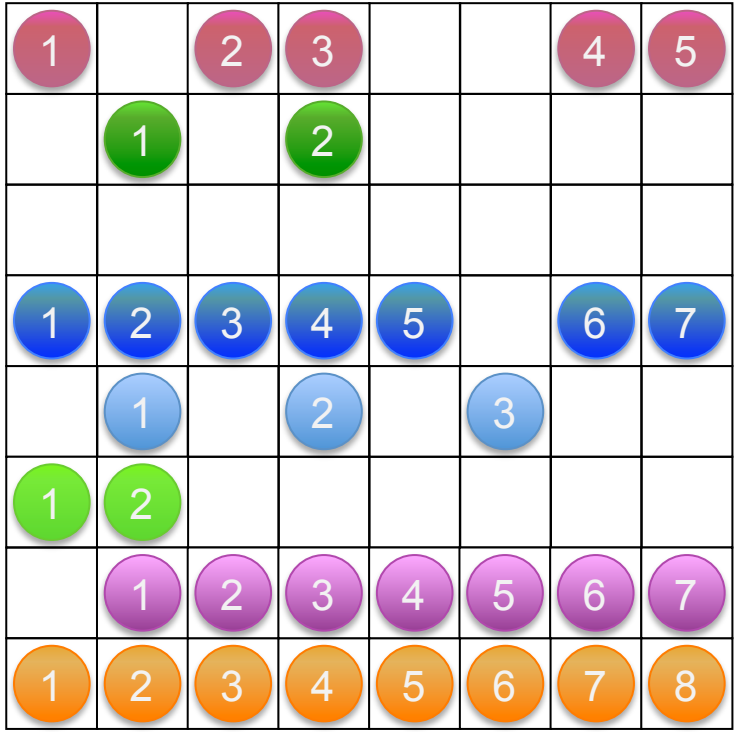


A (8x8), 34 nonzeros

core 0

core 1

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7
2	
3	
6	

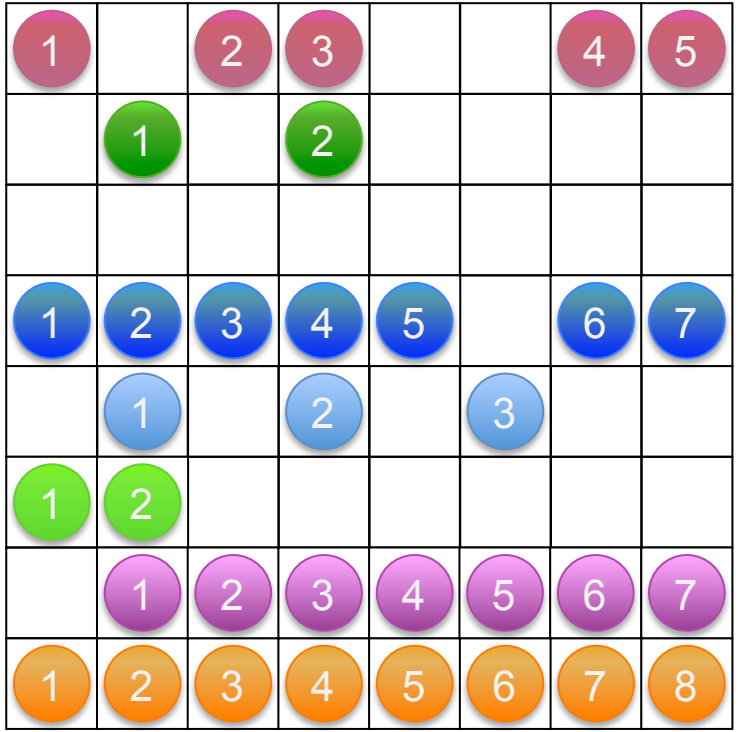
value =

1	5
2	
3	
4	

bit-flag =

T	F
F	
F	
F	

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7
2	1
3	3
6	

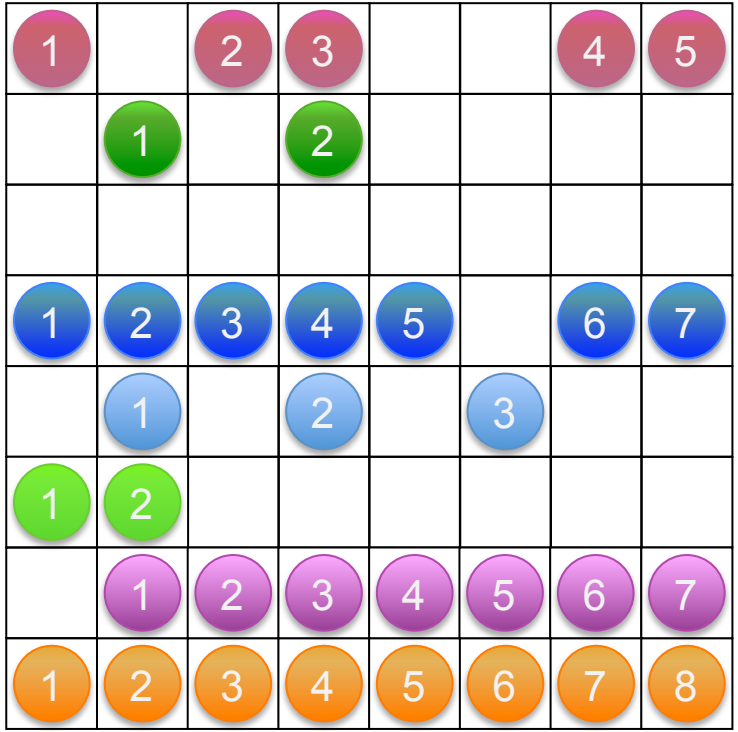
value =

1	5
2	1
3	2
4	

bit-flag =

T	F
F	T
F	F
F	

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	
6	0	4	

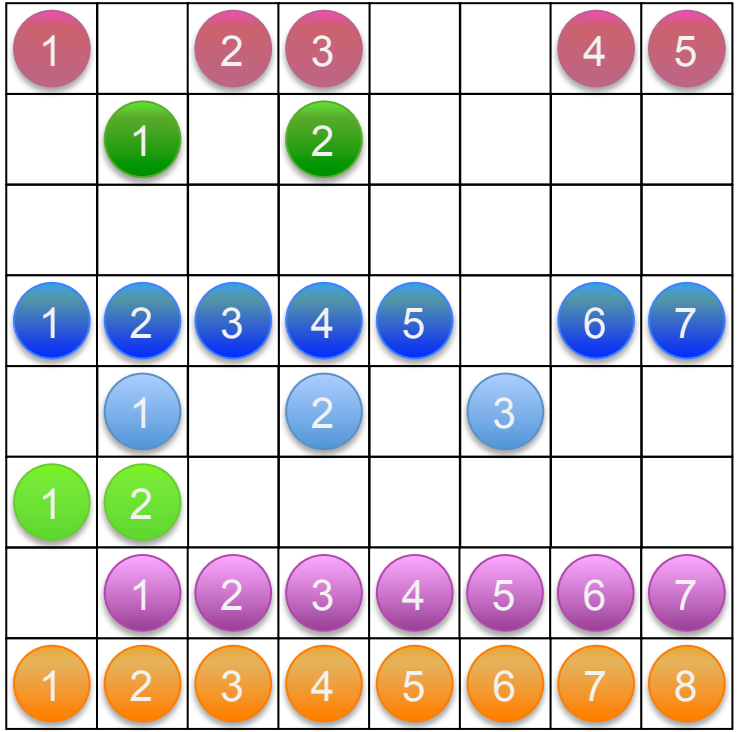
value =

1	5	2	6
2	1	3	7
3	2	4	
4	1	5	

bit-flag =

T	F	F	F
F	T	F	F
F	F	F	
F	T	F	

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3

value =

1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

bit-flag =

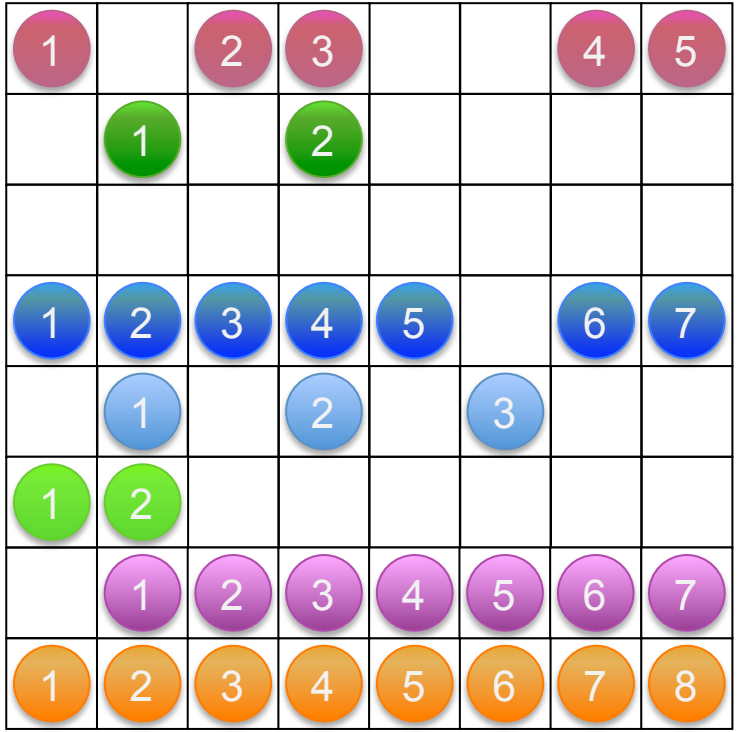
T	F	F	F
F	T	F	F
F	F	F	T
F	T	F	F

5	

3	

T	

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3

value =

1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

bit-flag =

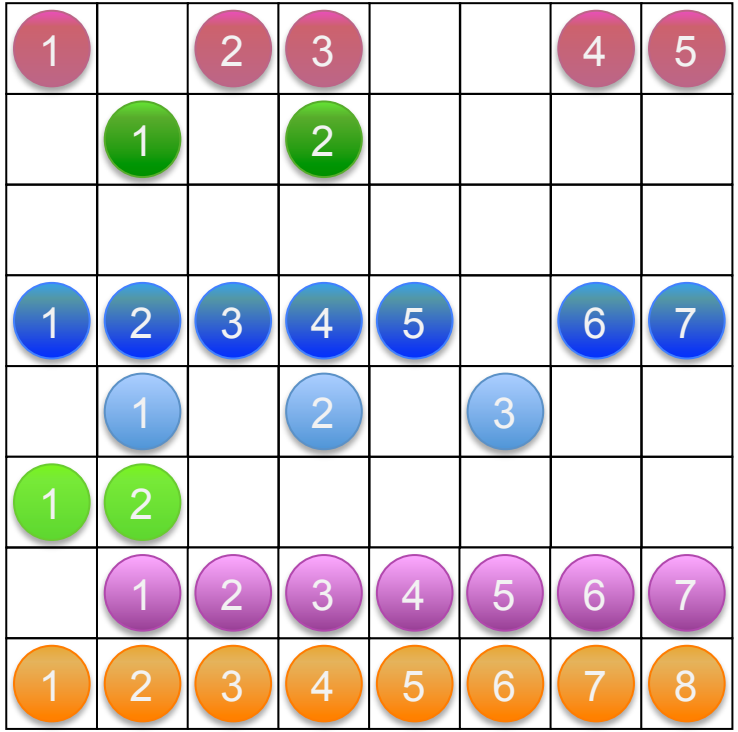
T	F	F	F
F	T	F	F
F	F	F	T
F	T	F	F

5
0
1

3
1
2

T
T
F

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3
5	2	6	
0	3	7	
1	4		
1	5		

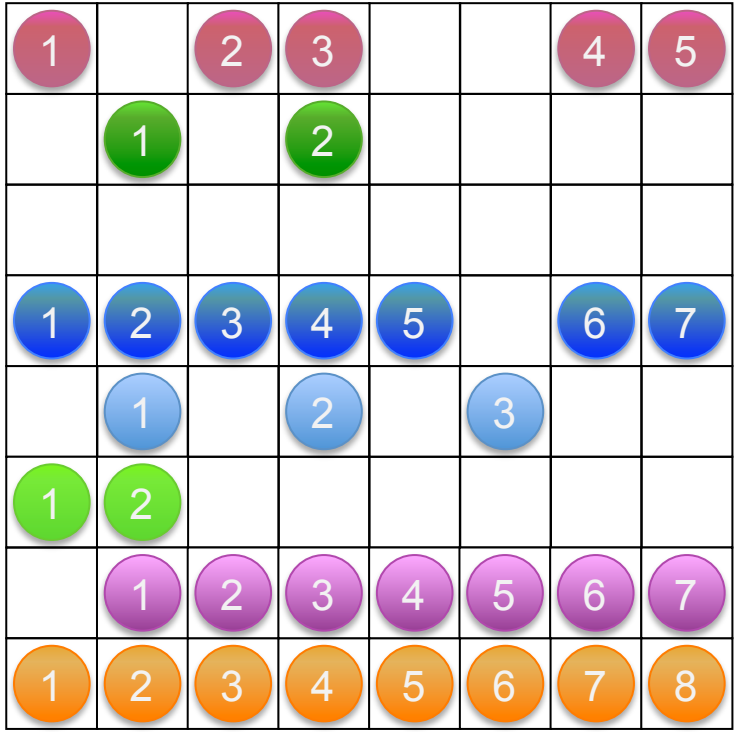
value =

1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2
3	2	6	
1	3	7	
2	4		
1	5		

bit-flag =

T	F	F	F
F	T	F	F
F	F	F	T
F	T	F	F
T	F	F	
T	F	F	
F	F		
T	F		

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3
5	2	6	2
0	3	7	3
1	4	0	4
1	5	1	5
6	7		

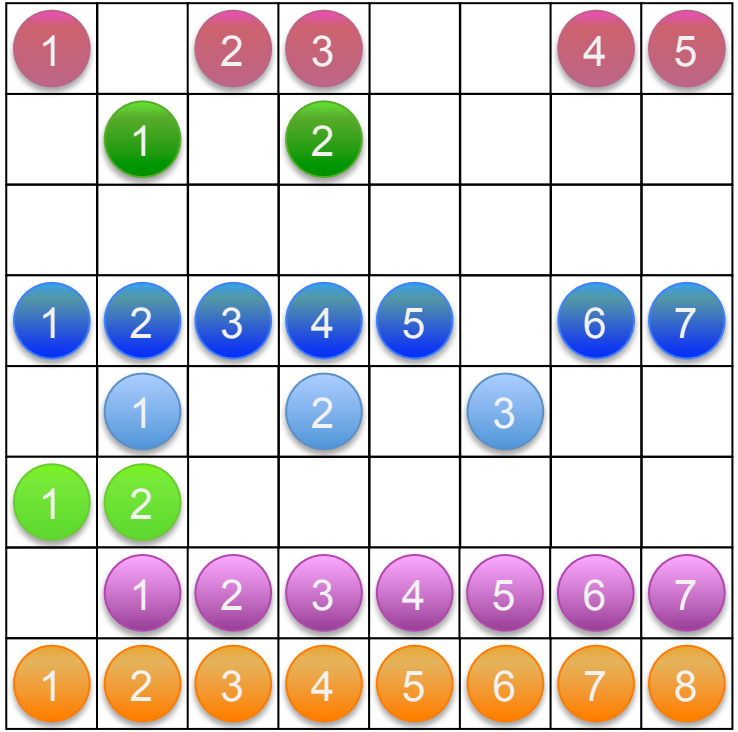
value =

1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2
3	2	6	3
1	3	7	4
2	4	1	5
1	5	2	6
7	8		

bit-flag =

T	F	F	F
F	T	F	F
F	F	F	T
F	T	F	F
T	F	F	F
T	F	F	F
F	F	T	F
T	F	F	F

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3

value =

bit-flag =

T	F	F	F
F	T	F	F
F	F	F	T
F	T	F	F

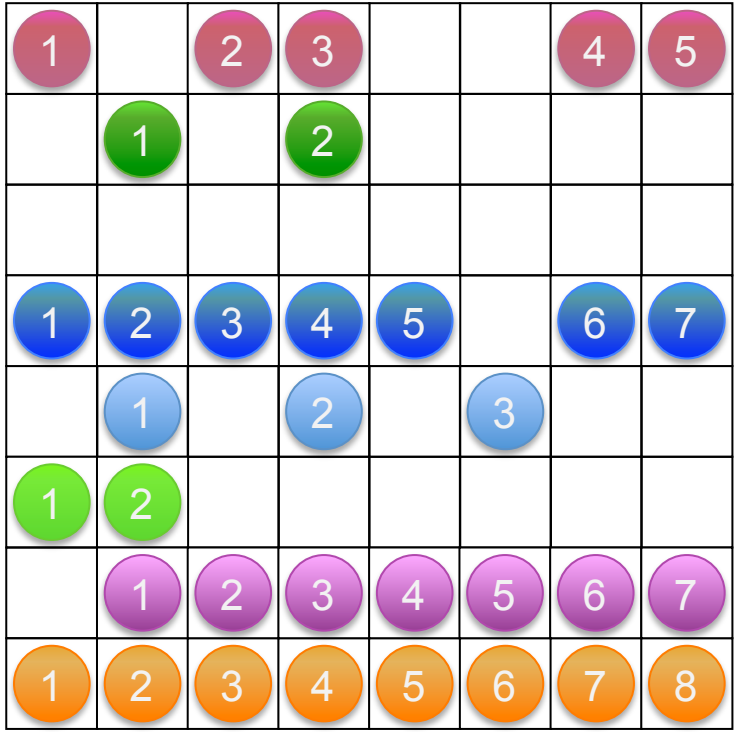
5	2	6	2
0	3	7	3
1	4	0	4
1	5	1	5

T	F	F	F
T	F	F	F
F	F	T	F
T	F	F	F

6	7

7	8

CSR5: tiling



A (8x8), 34 nonzeros

column index =

0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3

value =

bit-flag =

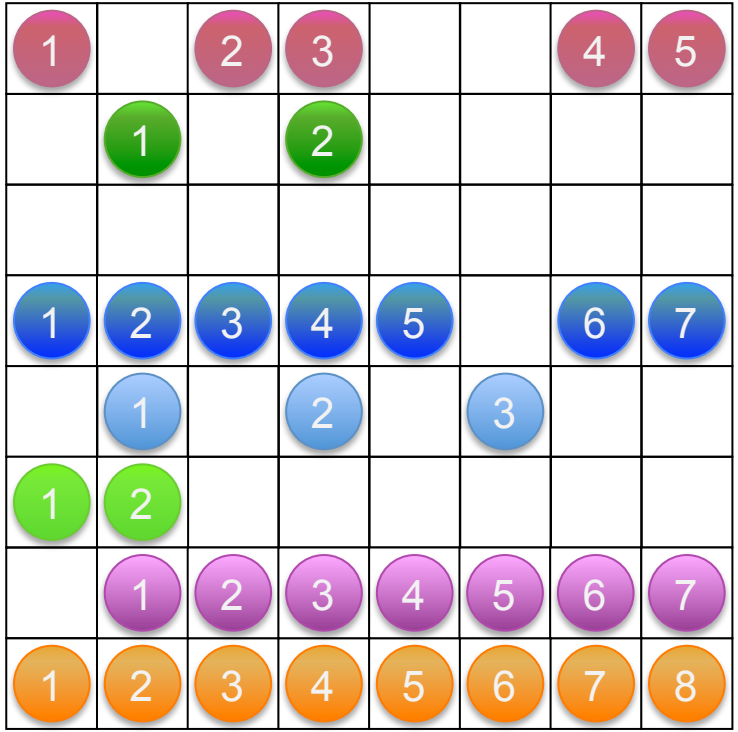
T	F	F	F
F	T	F	F
F	F	F	T
F	T	F	F

5	2	6	2
0	3	7	3
1	4	0	4
1	5		
6	7		

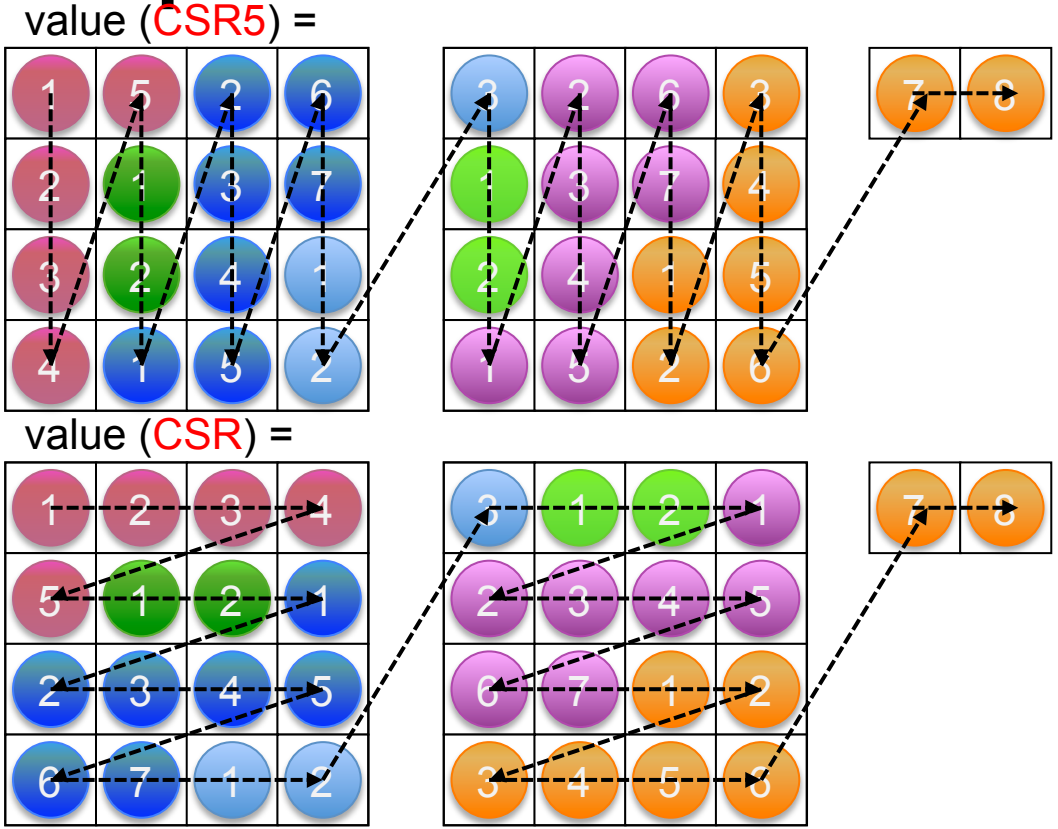
T	F	F	F
T	F	F	F
F	F	T	F

Tile, but not nonzero, is the basic work unit for load balancing.

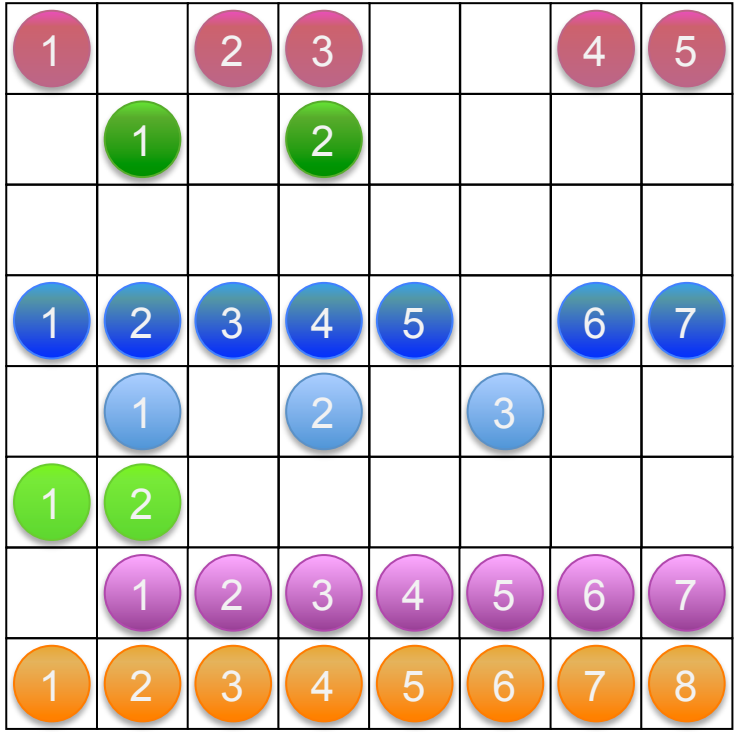
CSR5: tile-wise transposition



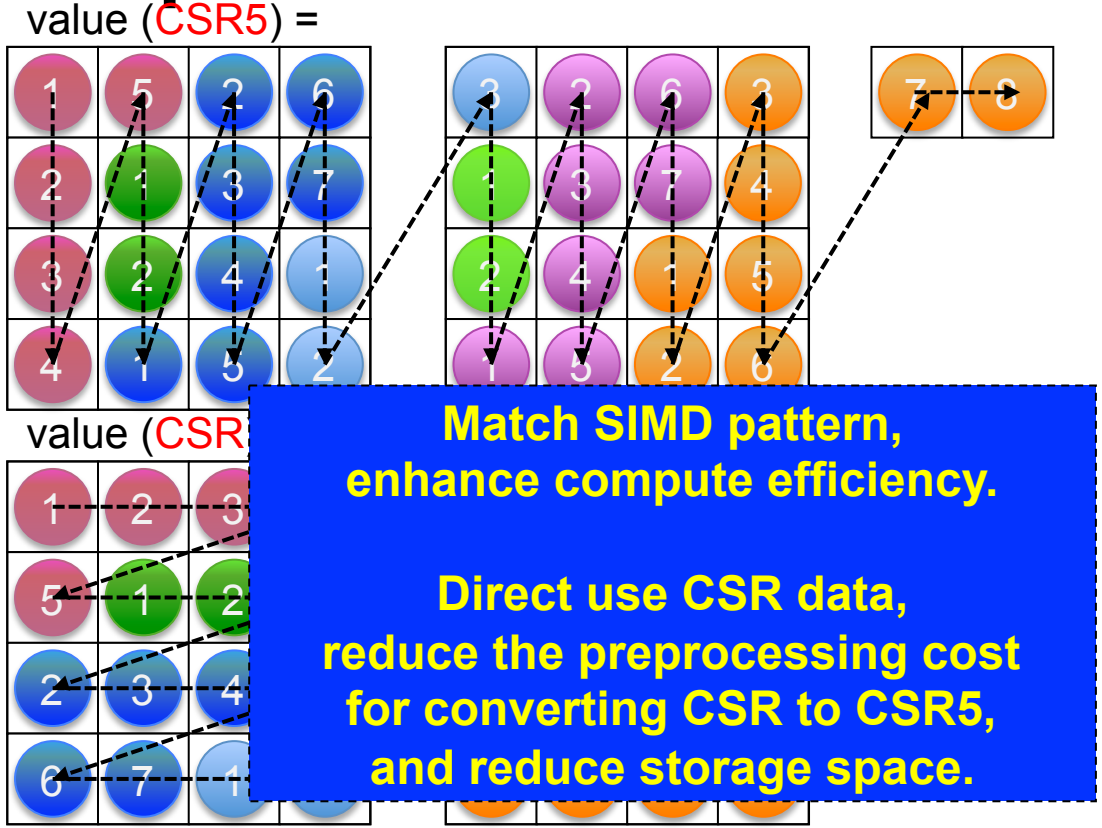
A (8x8), 34 nonzeros



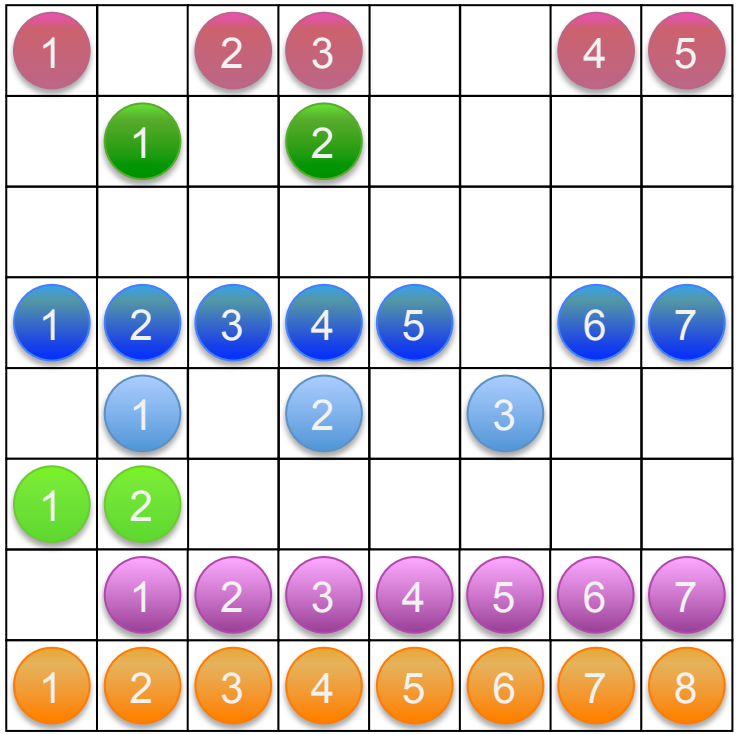
CSR5: tile-wise transposition



A (8x8), 34 nonzeros



CSR5: tile pointer array



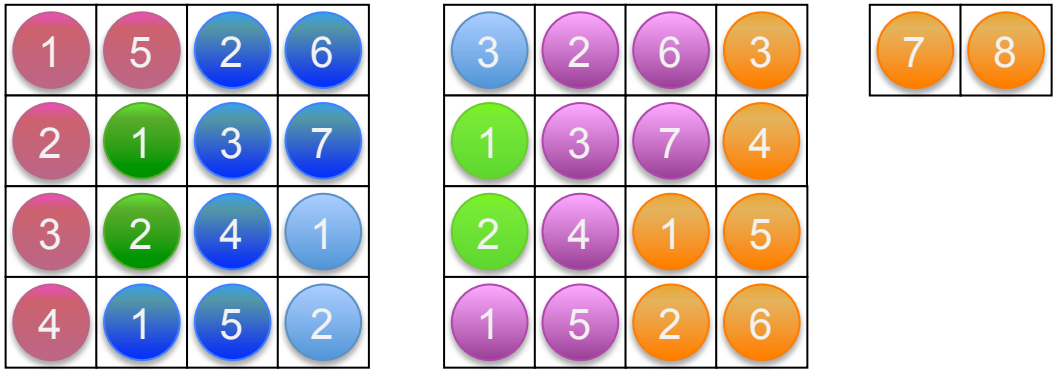
A (8x8), 34 nonzeros

tile pointer =

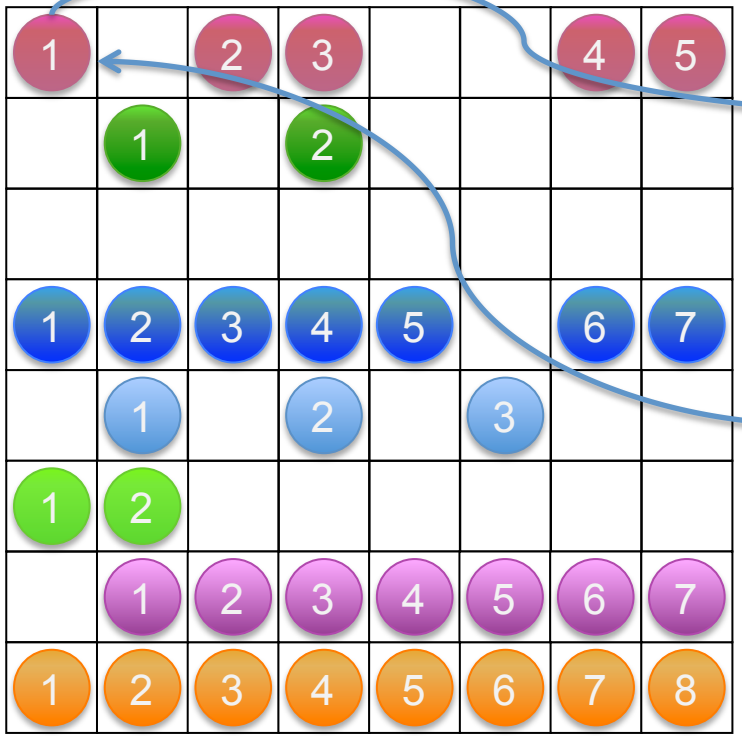
0	4	7	8

***Tile pointer* records the row indices of the first entries in tiles. If empty rows exist in a tile, the its pointer is negative.**

value =



CSR5: tile pointer array

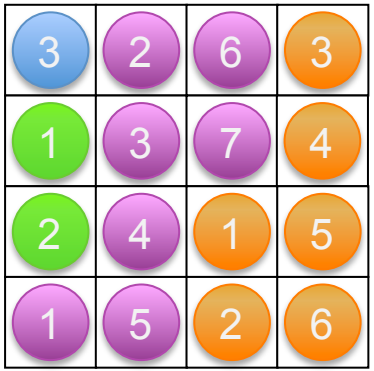


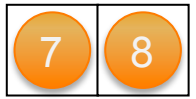
A (8x8), 34 nonzeros

tile pointer =

0	4	7	8

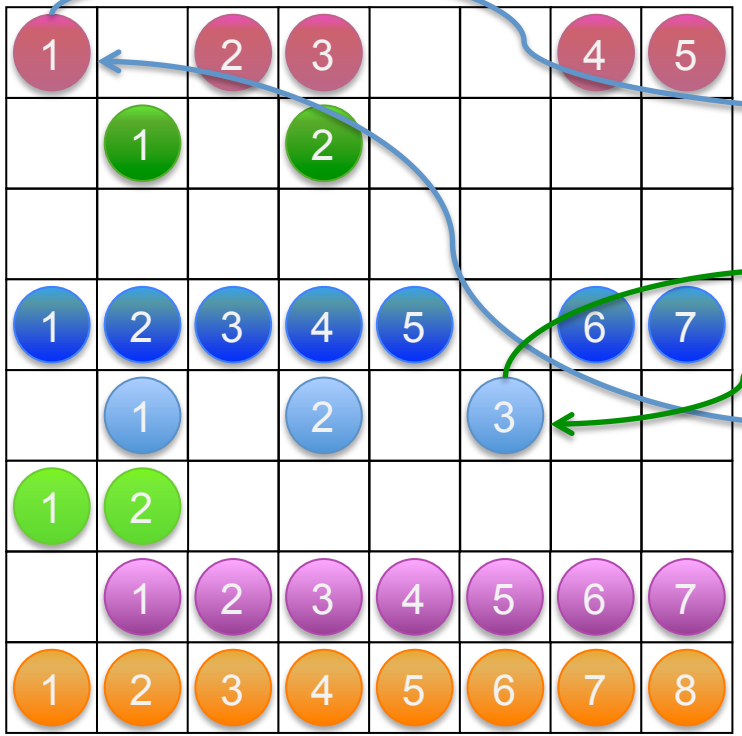
value =





***Tile pointer* records the row indices of the first entries in tiles. If empty rows exist in a tile, the its pointer is negative.**

CSR5: tile pointer array

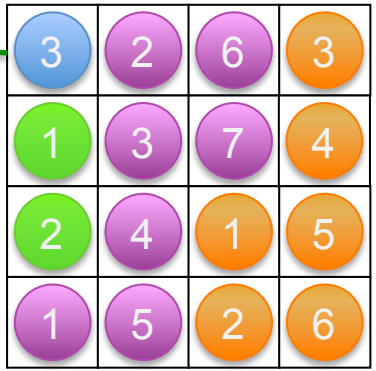


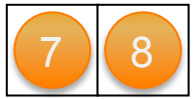
A (8x8), 34 nonzeros

tile pointer =

0	4	7	8

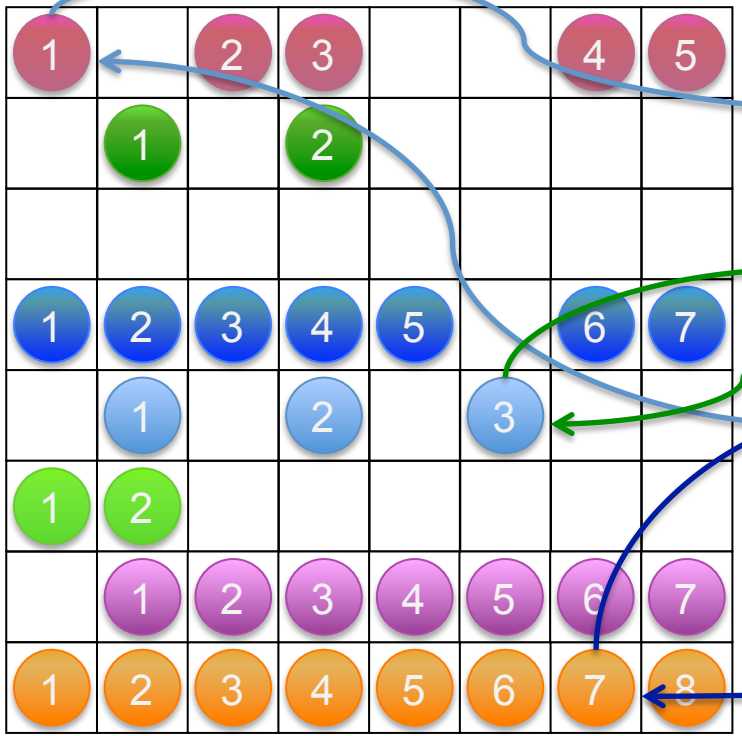
value =





***Tile pointer* records the row indices of the first entries in tiles. If empty rows exist in a tile, the its pointer is negative.**

CSR5: tile pointer array

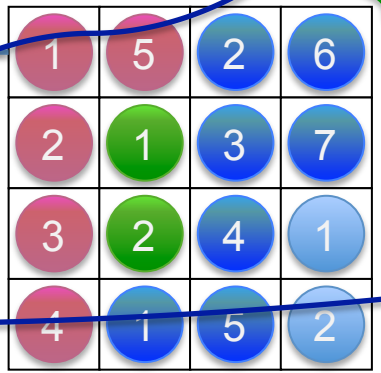


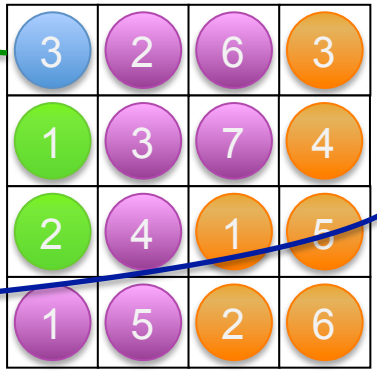
A (8x8), 34 nonzeros

tile pointer =

0	4	7	8

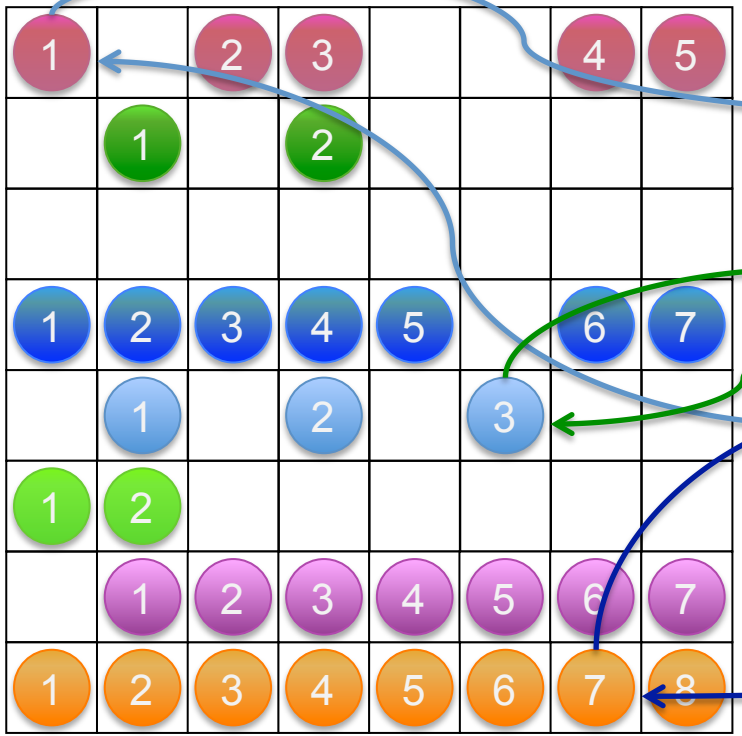
value =





***Tile pointer* records the row indices of the first entries in tiles. If empty rows exist in a tile, the its pointer is negative.**

CSR5: tile pointer array

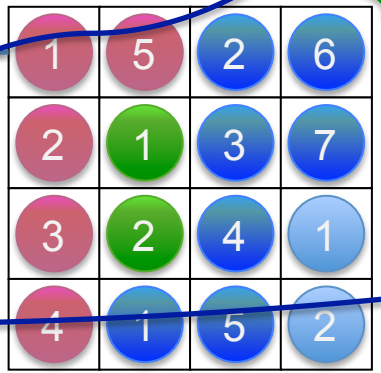


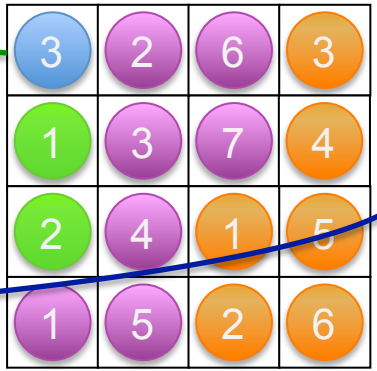
A (8x8), 34 nonzeros

tile pointer =

~~8~~	4	7	8
-0			

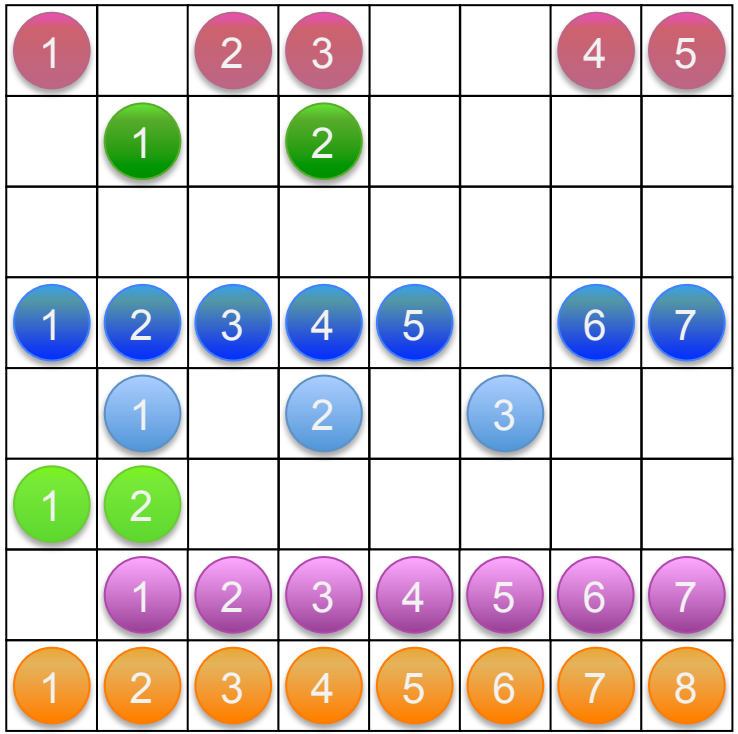
value =



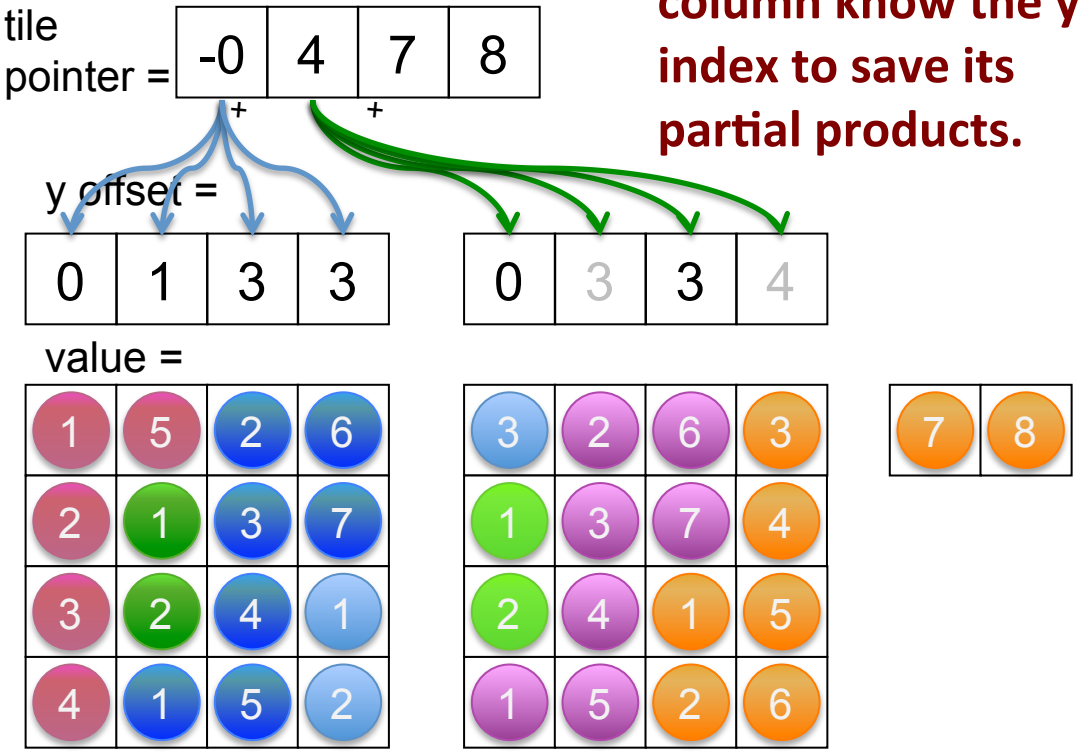


***Tile pointer* records the row indices of the first entries in tiles. If empty rows exist in a tile, the its pointer is negative.**

CSR5: tile descriptor info

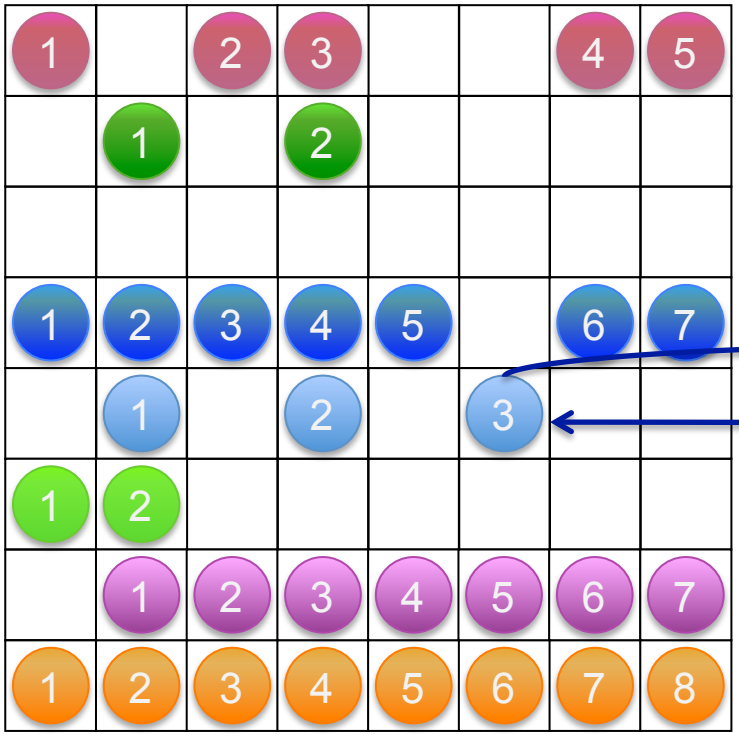


A (8x8), 34 nonzeros

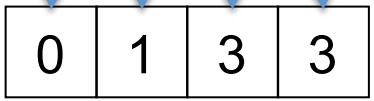


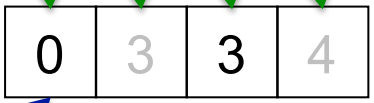
y offset **lets each column know the y index to save its partial products.**

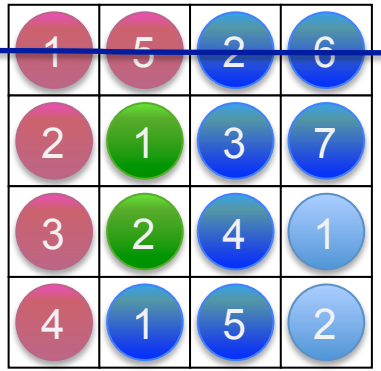
CSR5: tile descriptor info

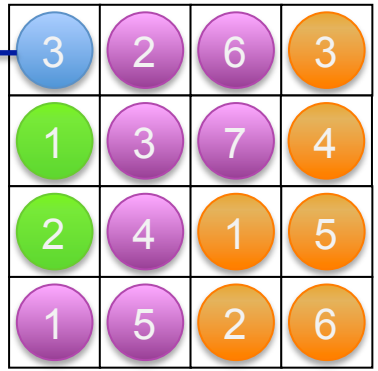


A (8x8), 34 nonzeros



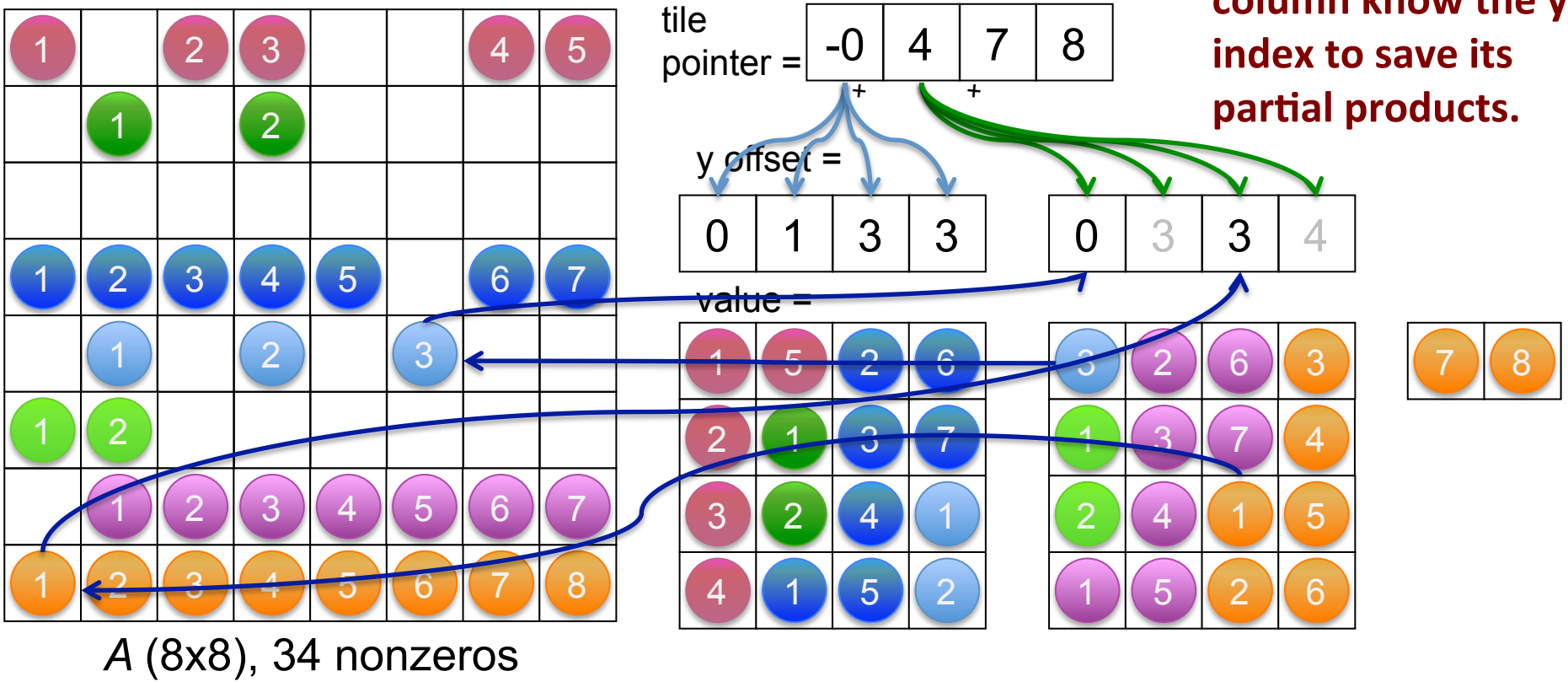






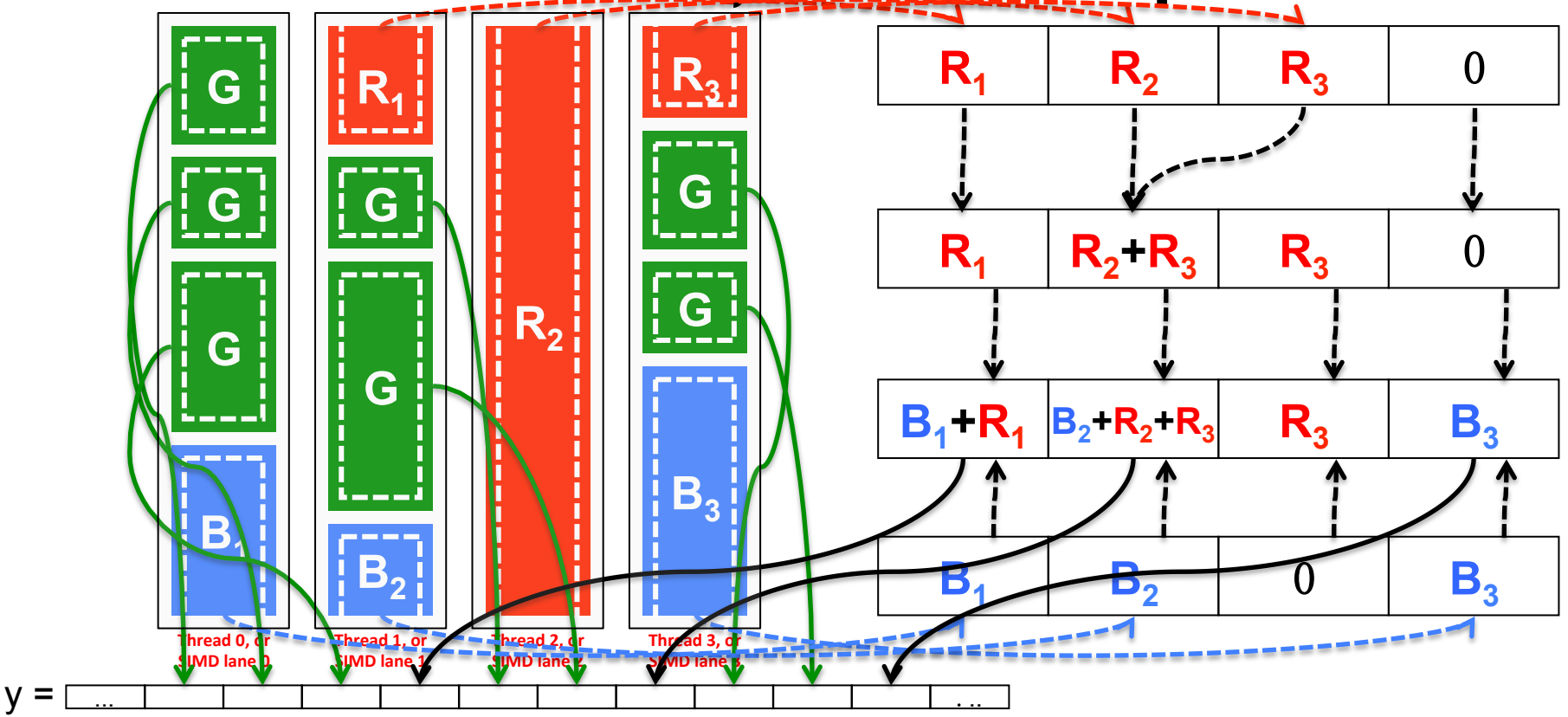
y offset **lets each column know the y index to save its partial products.**

CSR5: tile descriptor info



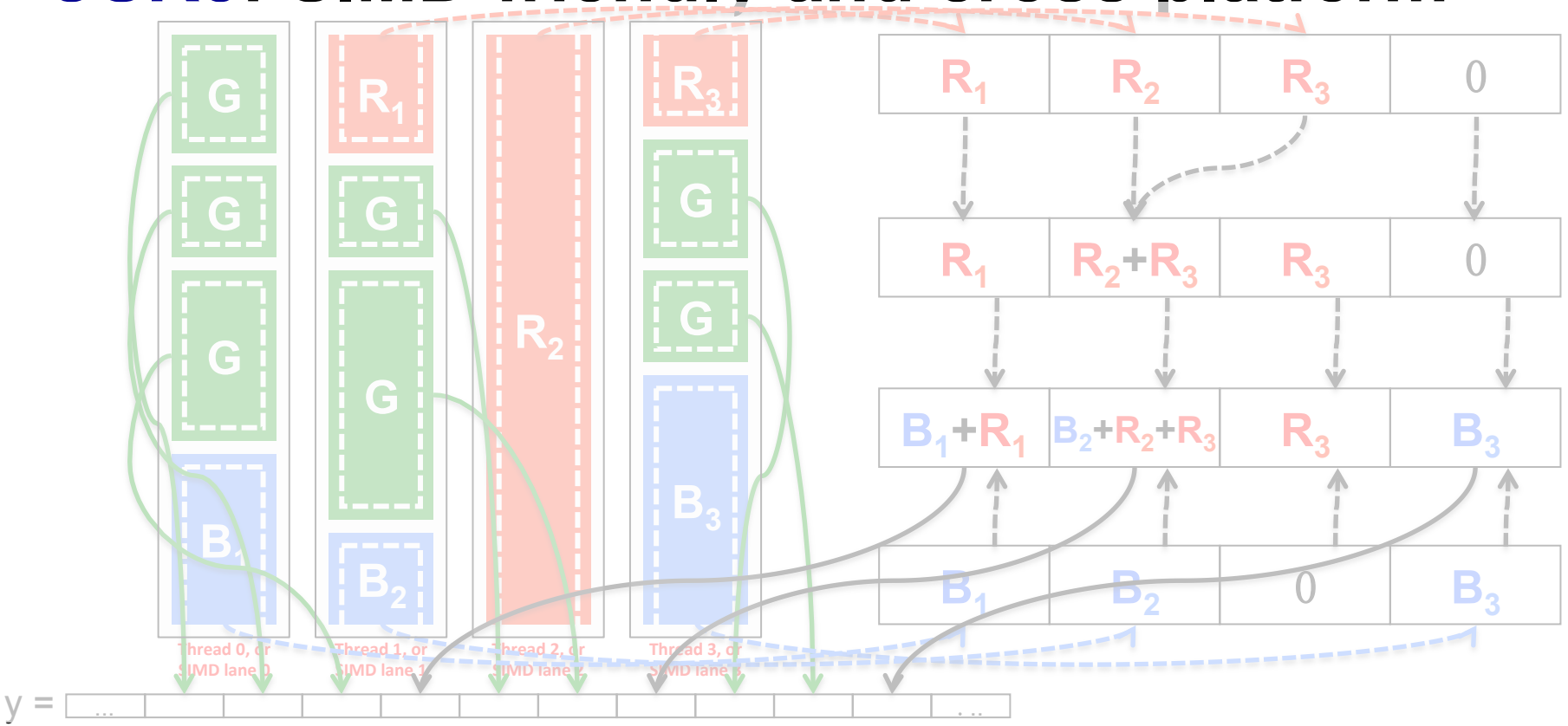
y offset lets each column know the y index to save its partial products.

CSR5: SIMD-friendly and cross platform



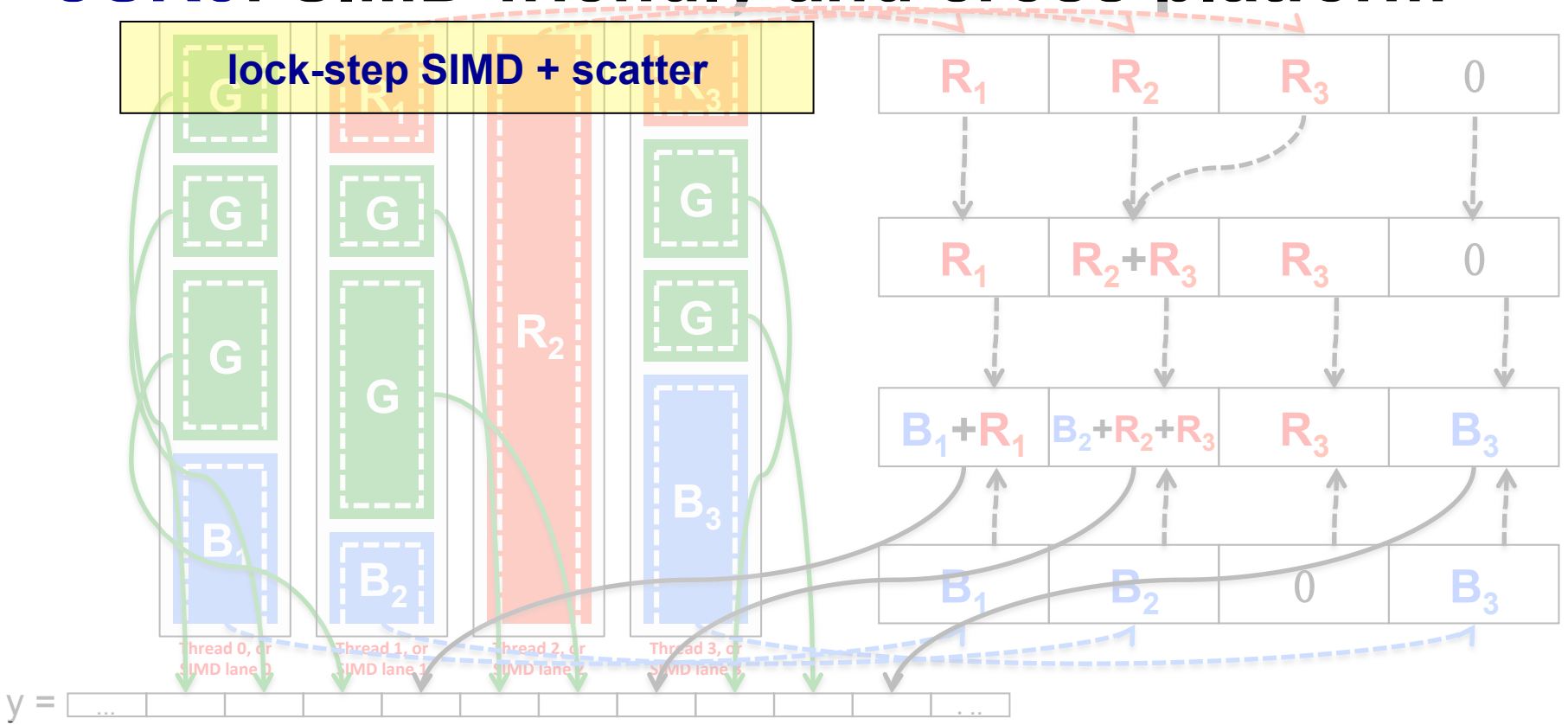
NTNU

CSR5: SIMD-friendly and cross platform

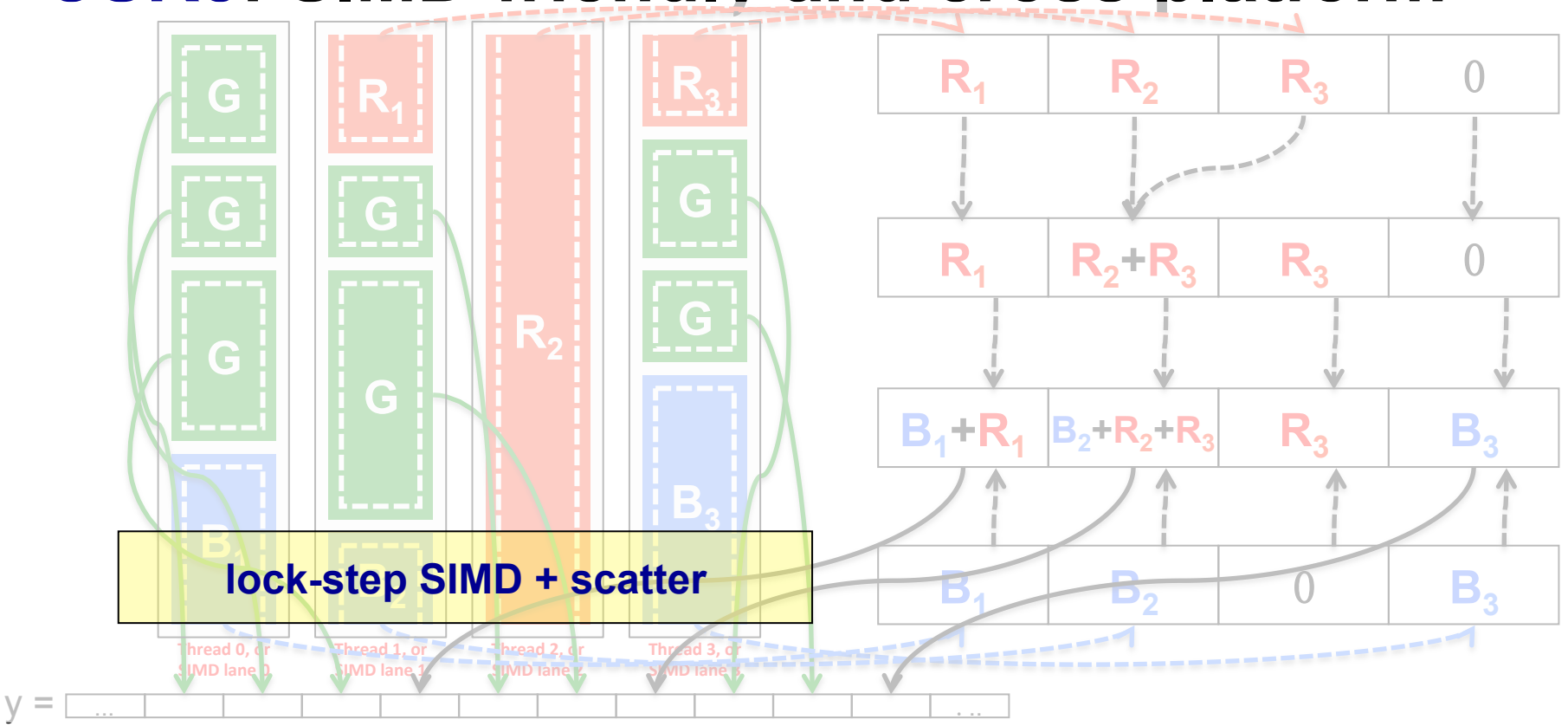


NTNU

CSR5: SIMD-friendly and cross platform

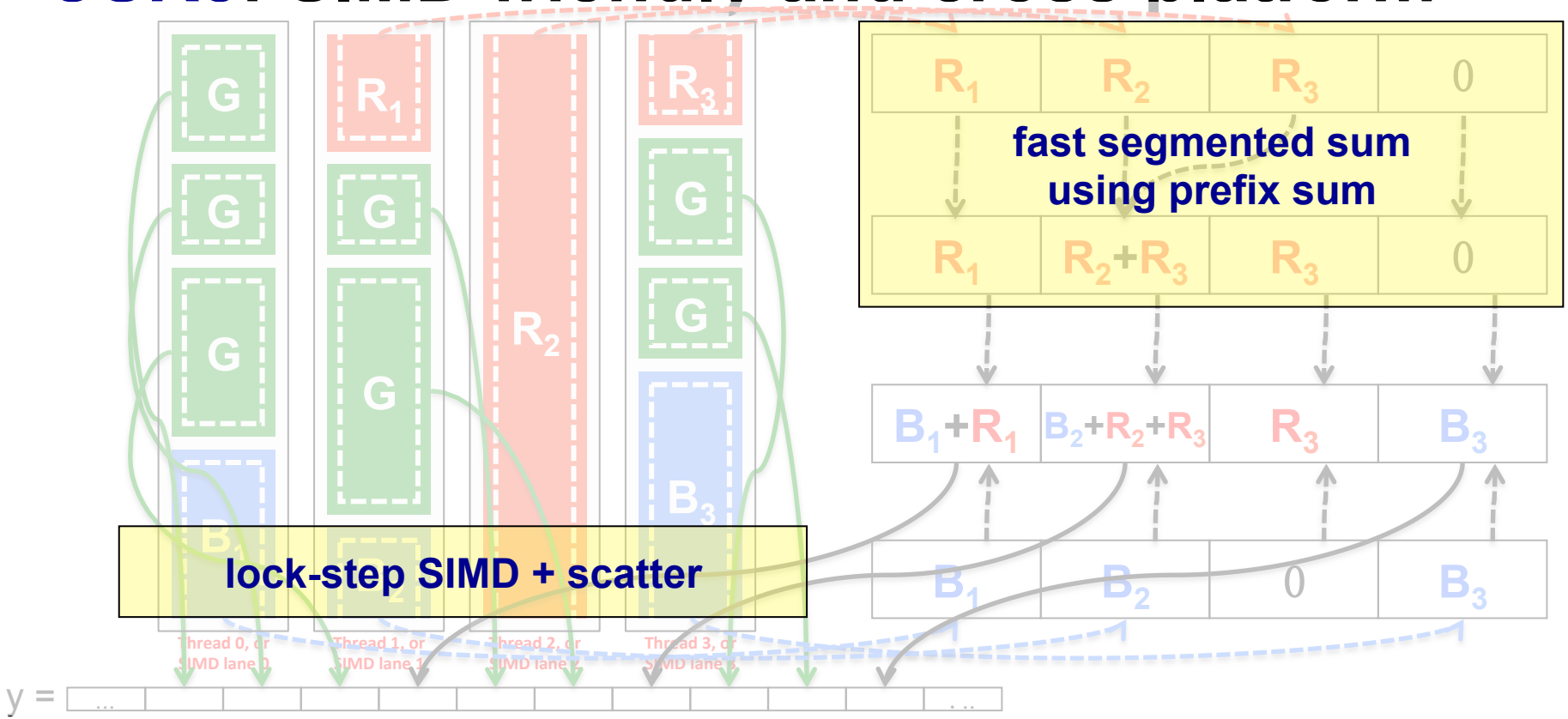


CSR5: SIMD-friendly and cross platform



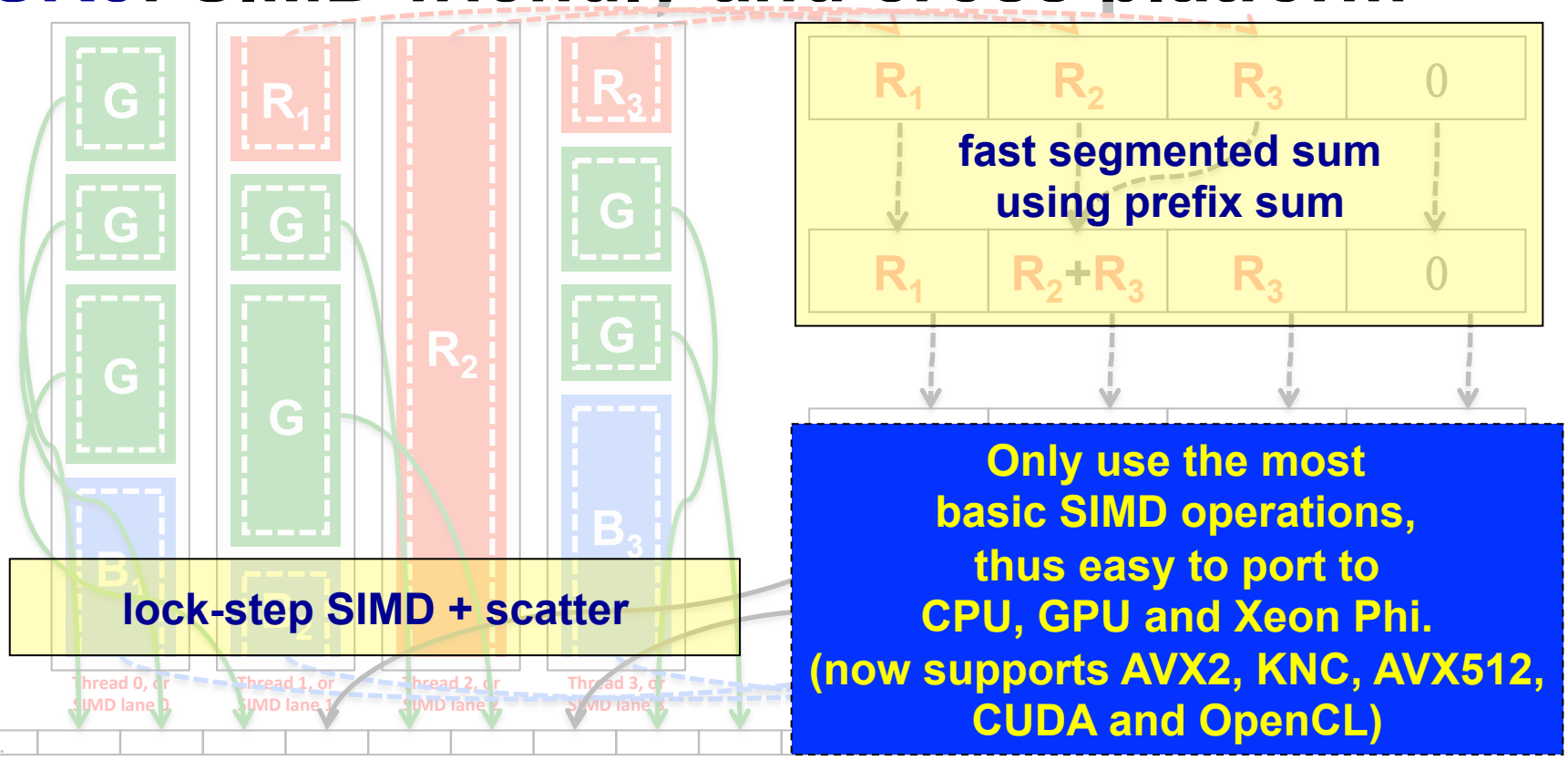
NTNU

CSR5: SIMD-friendly and cross platform

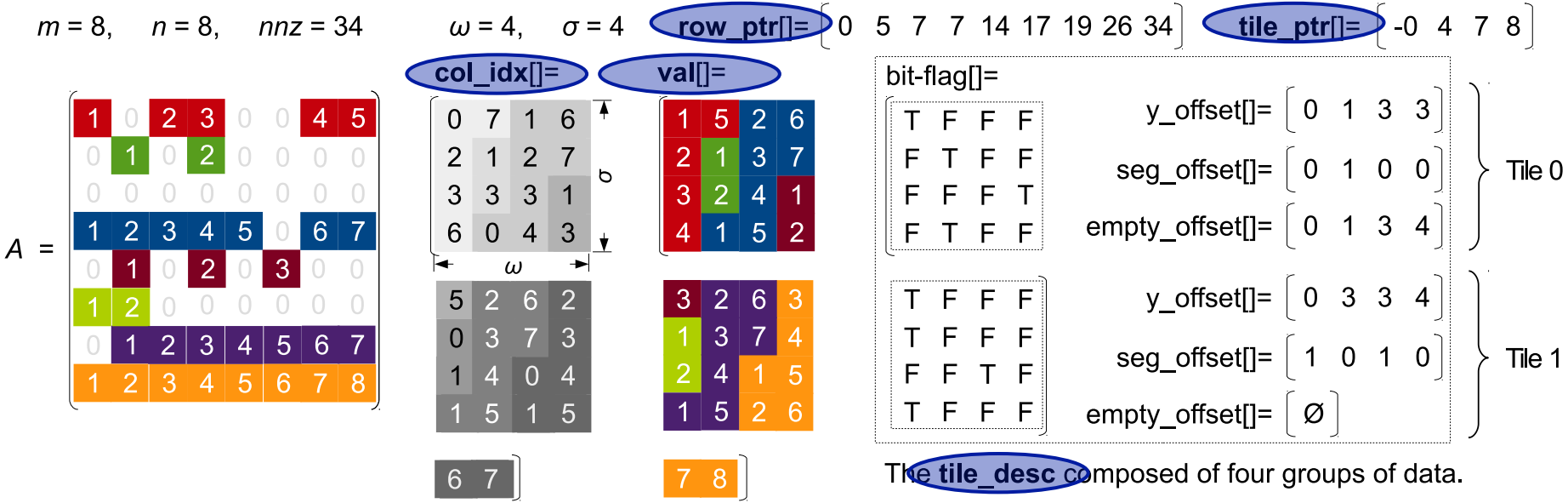


NTNU

CSR5: SIMD-friendly and cross platform

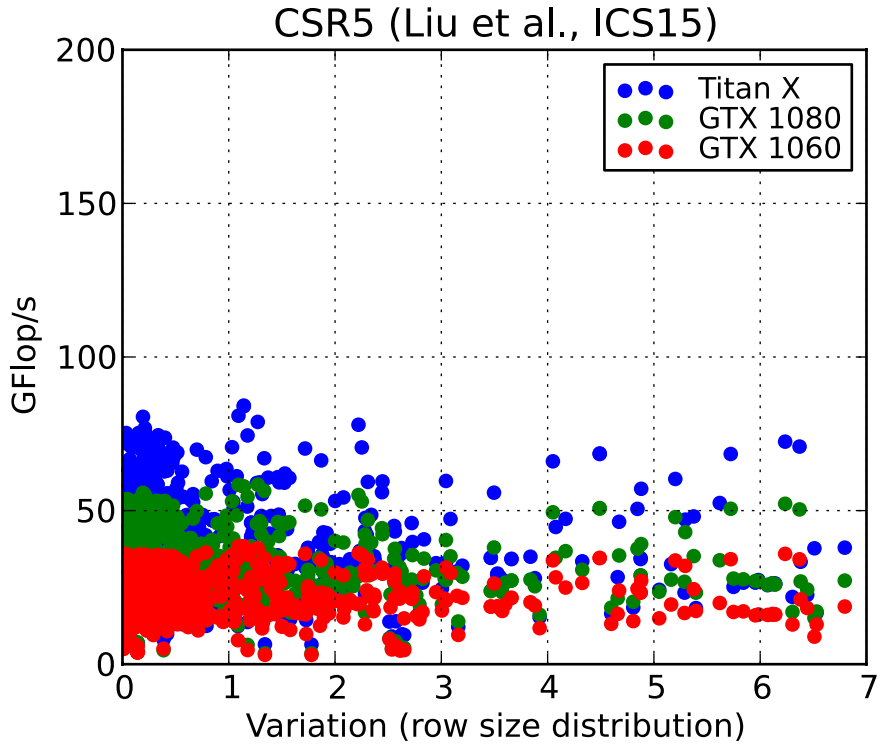


CSR5: naming

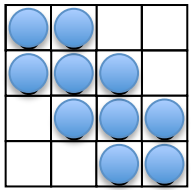


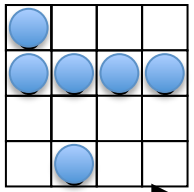
Compared three groups of info for CSR, CSR5 uses CSR data and stores five groups of info. So it is called CSR5.

Scalability of SpMV using CSR5



uniform dist.

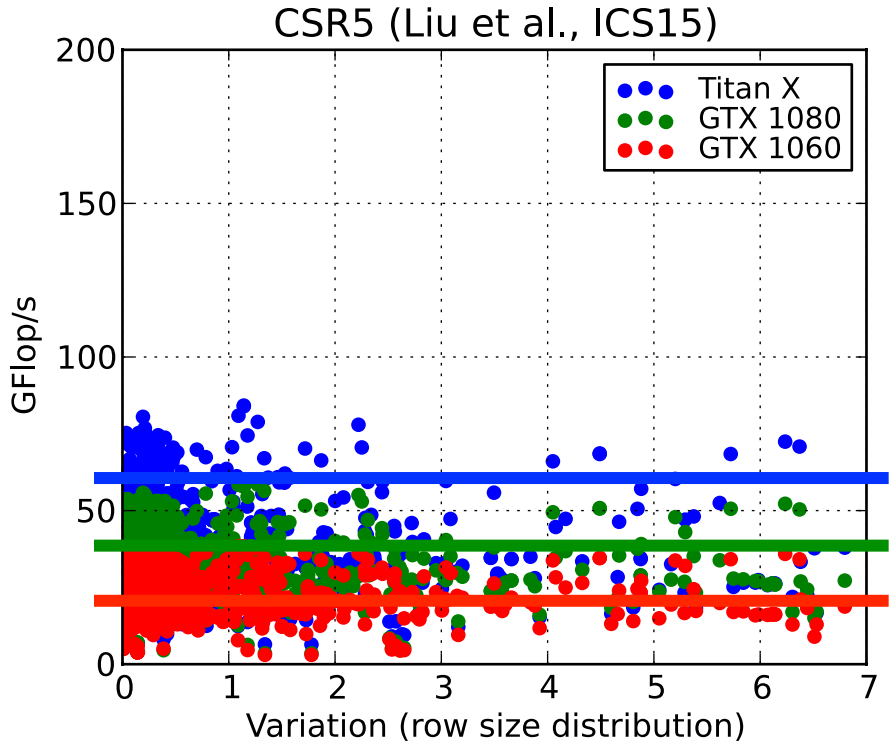




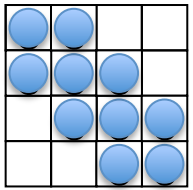
power-law dist.

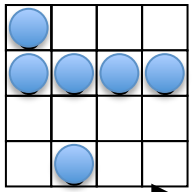
NTNU

Scalability of SpMV using CSR5



uniform dist.

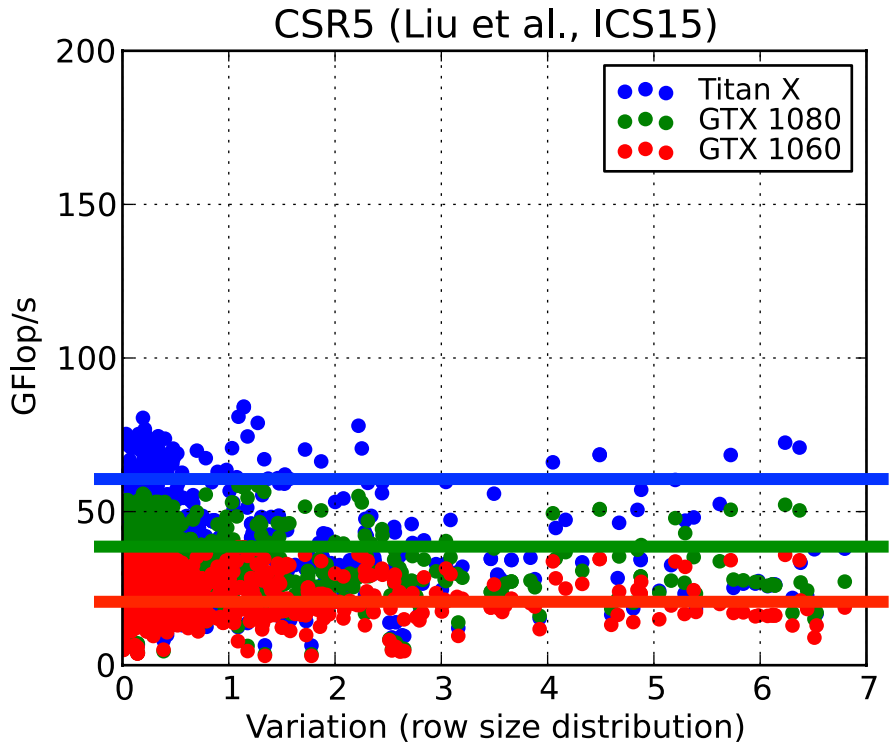




power-law dist.

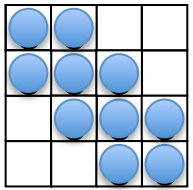
NTNU

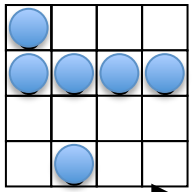
Scalability of SpMV using CSR5



SpMV perf. scales with hardware perf., and is insensitive to sparsity structures.

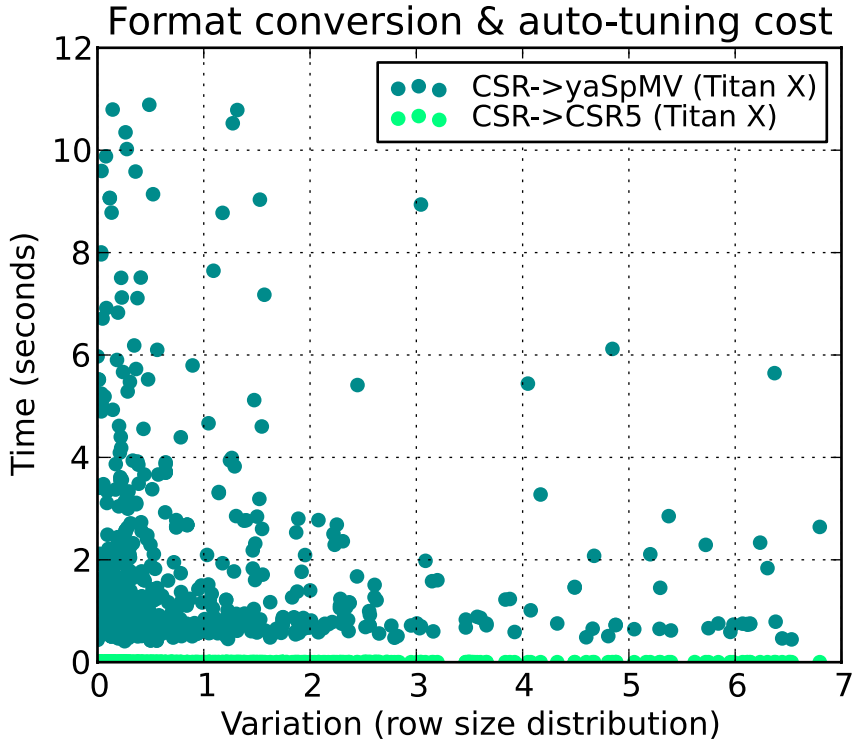
uniform dist.



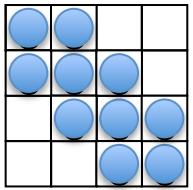


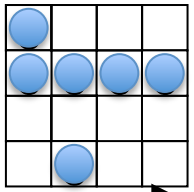
power-law dist.

Data conv. and autotuning cost of CSR5



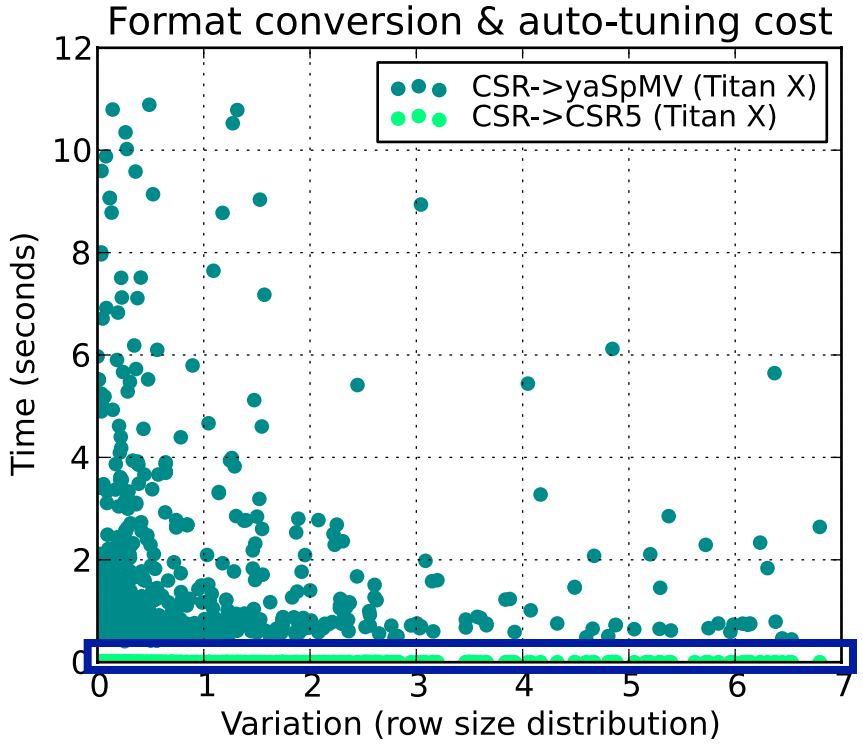
uniform dist.





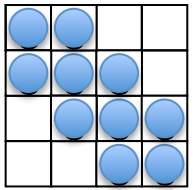
power-law dist.

Data conv. and autotuning cost of CSR5

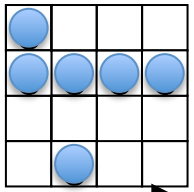


Preprocessing cost for CSR5 is just the cost for a few SpMV.

uniform dist.

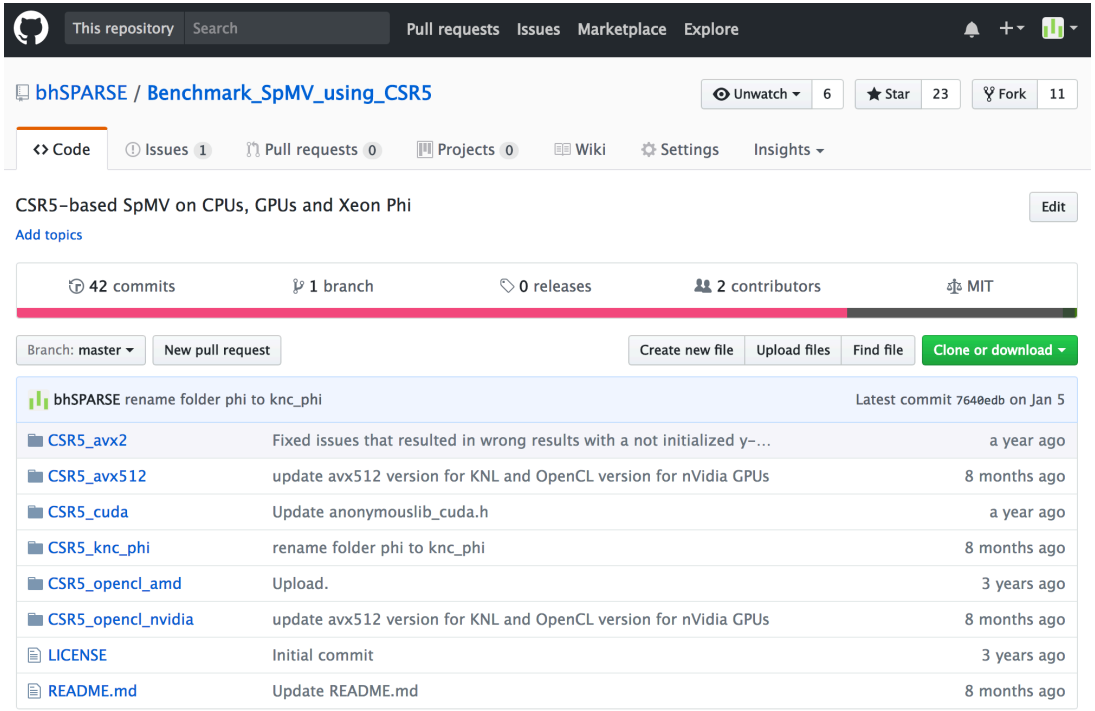


power-law dist.



NTNU

Source code of CSR5 on Github



https://github.com/bhSPARSE/Benchmark_SpMV_using_CSR5

CSR5 in MAGMA

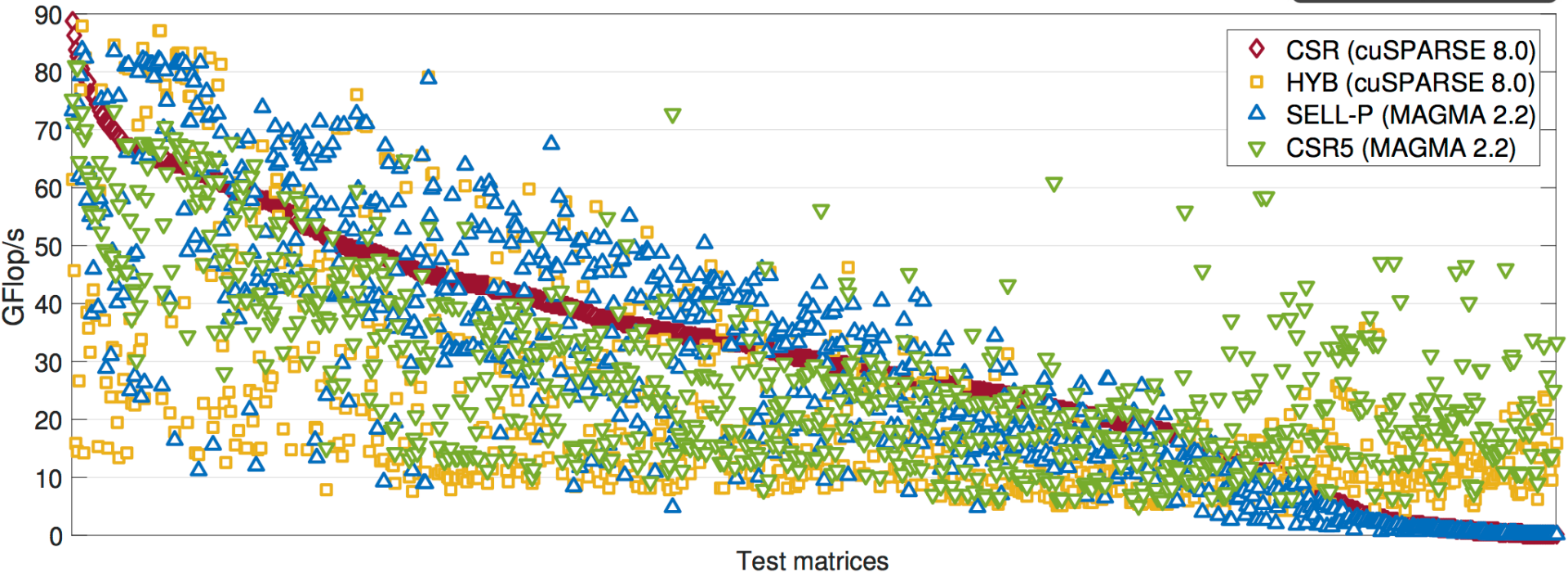
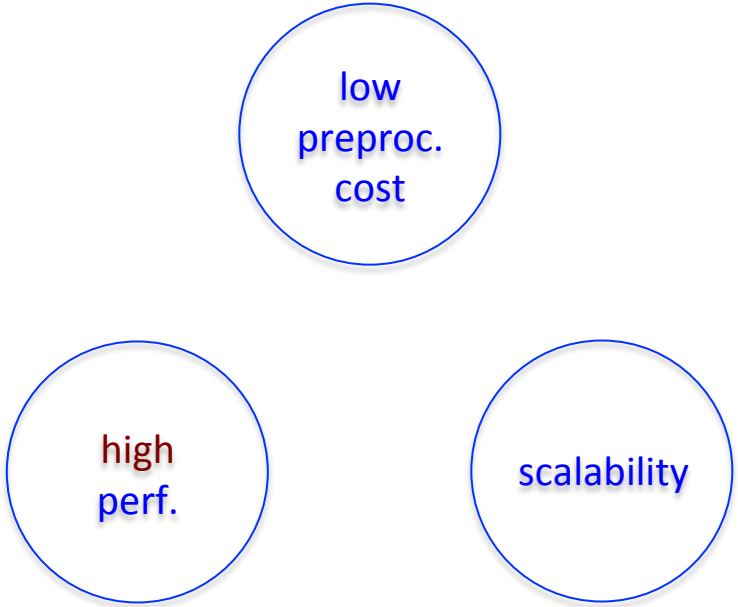
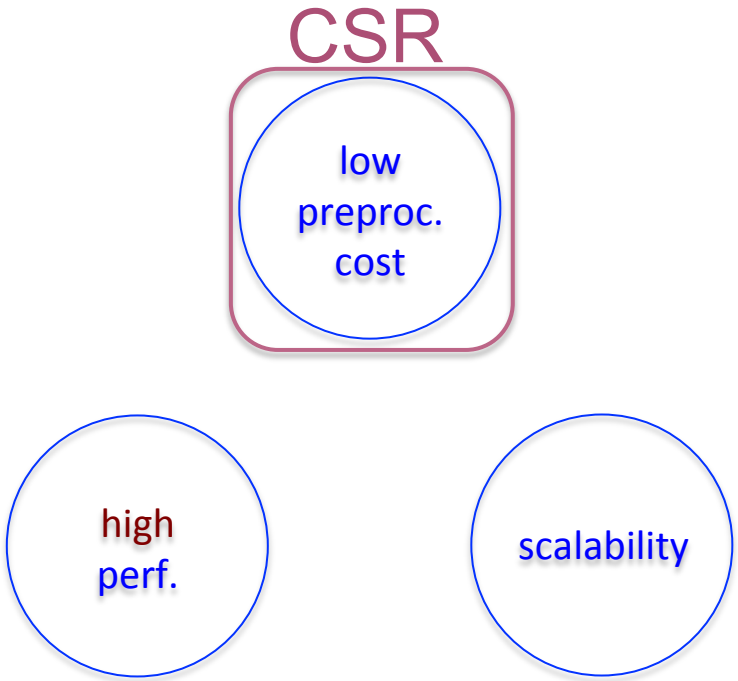


Figure from Dr. Hartwig Anzt at ICL, UTK; Device: NVIDIA Tesla P100; Precision: double.

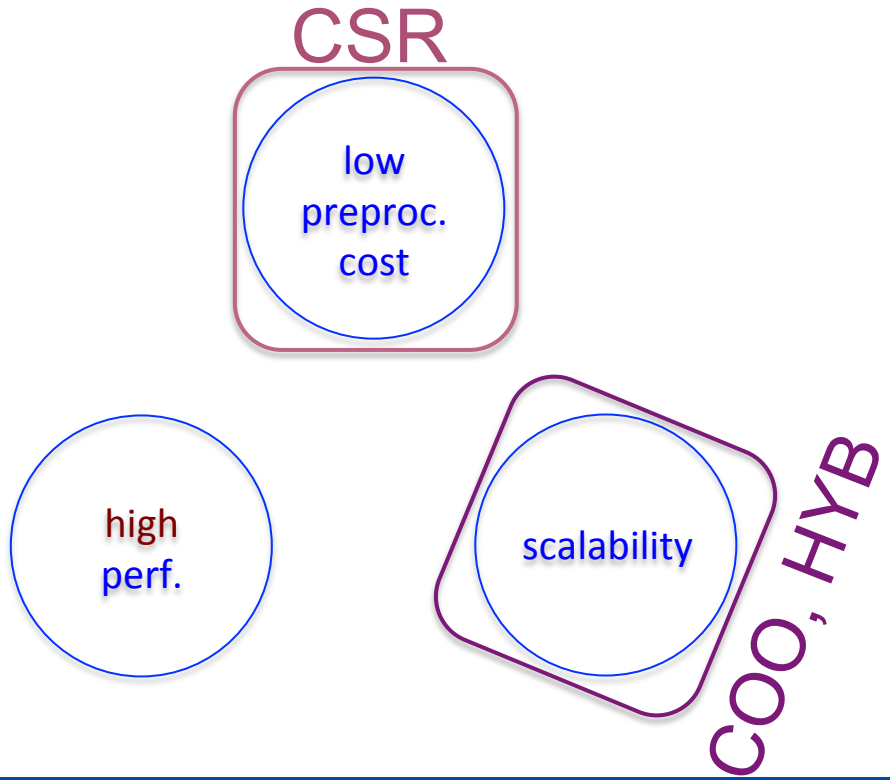
Scalability, high perf. and low preproc. cost (choose two of the three?)



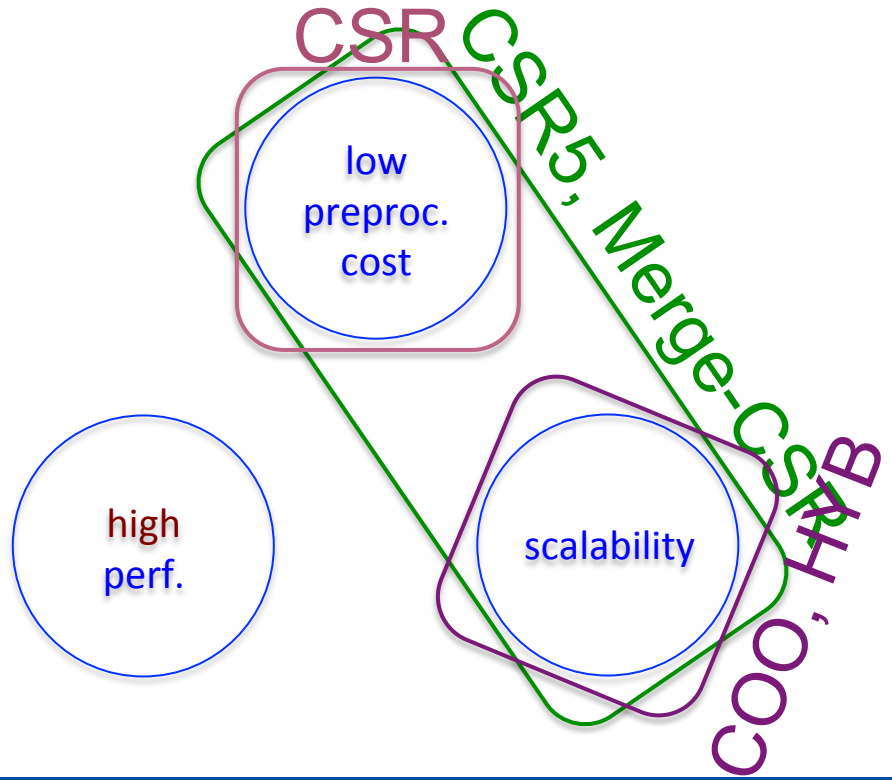
Scalability, high perf. and low preproc. cost (choose two of the three?)



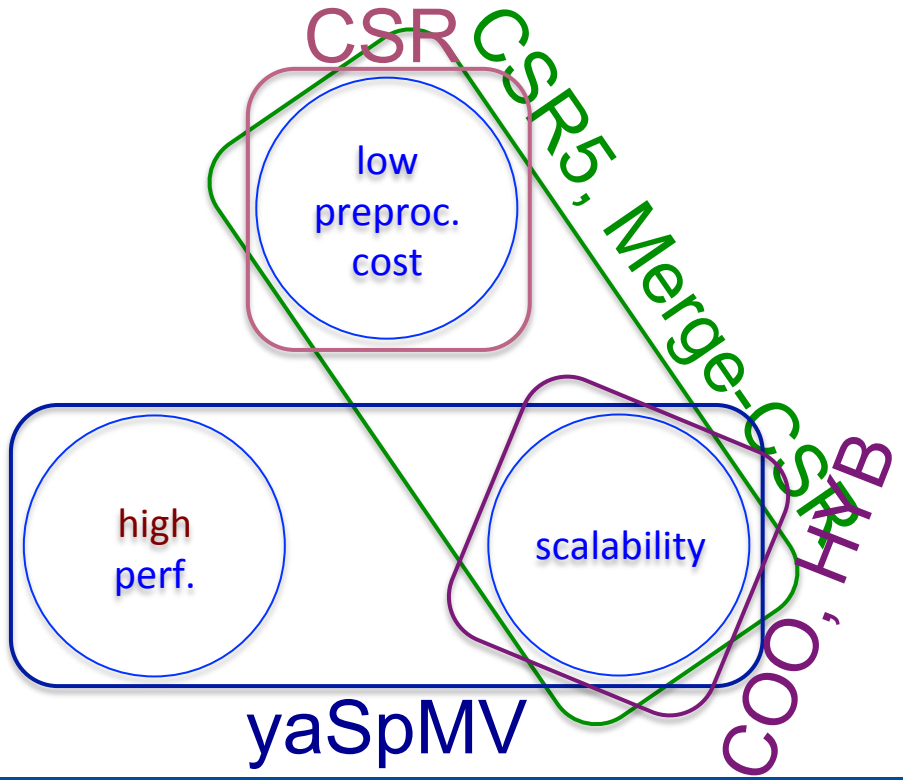
Scalability, high perf. and low preproc. cost (choose two of the three?)



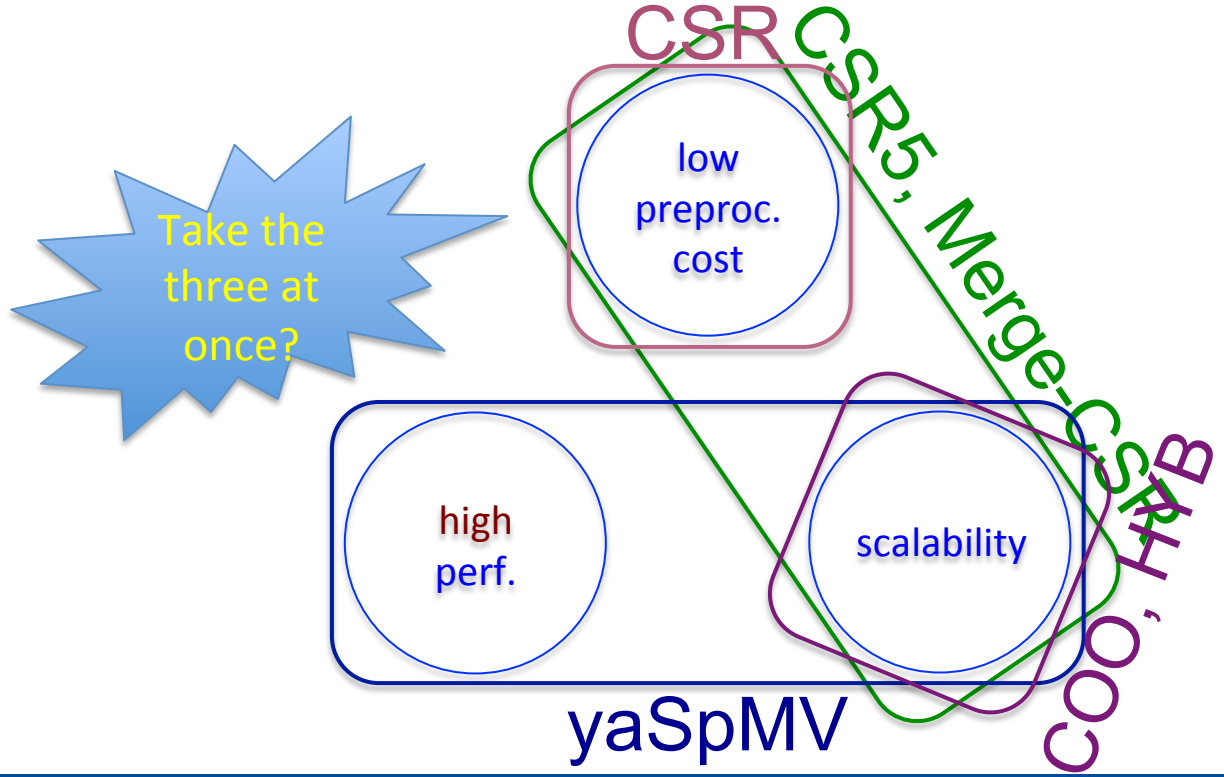
Scalability, high perf. and low preproc. cost (choose two of the three?)



Scalability, high perf. and low preproc. cost (choose two of the three?)



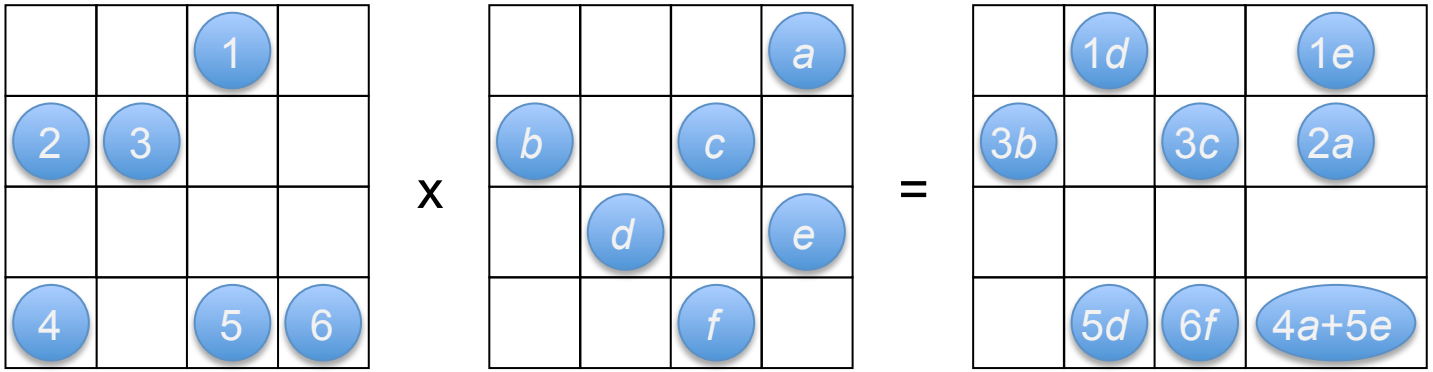
Scalability, high perf. and low preproc. cost (choose two of the three?)



Kernel 2. Sparse Matrix-Matrix Multiplication (SpGEMM)

SpGEMM

- Multiply a sparse matrix *A* by a sparse matrix *B*, and obtain a resulting sparse matrix *C*.



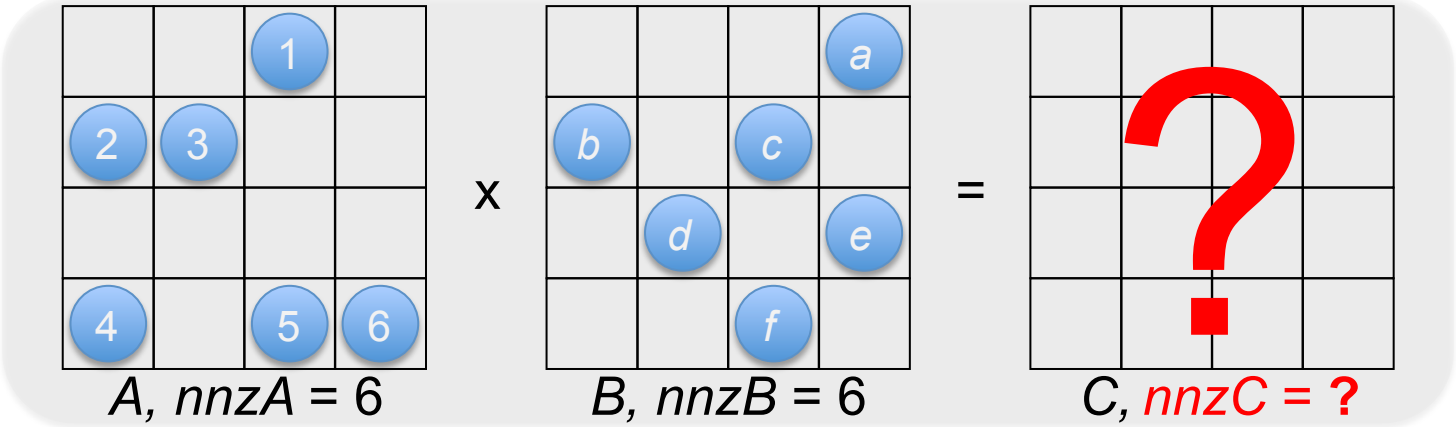
A (4x4) sparse, 6 nonzeros

B (4x4) sparse, 6 nonzeros

C (4x4) sparse, 8 nonzeros

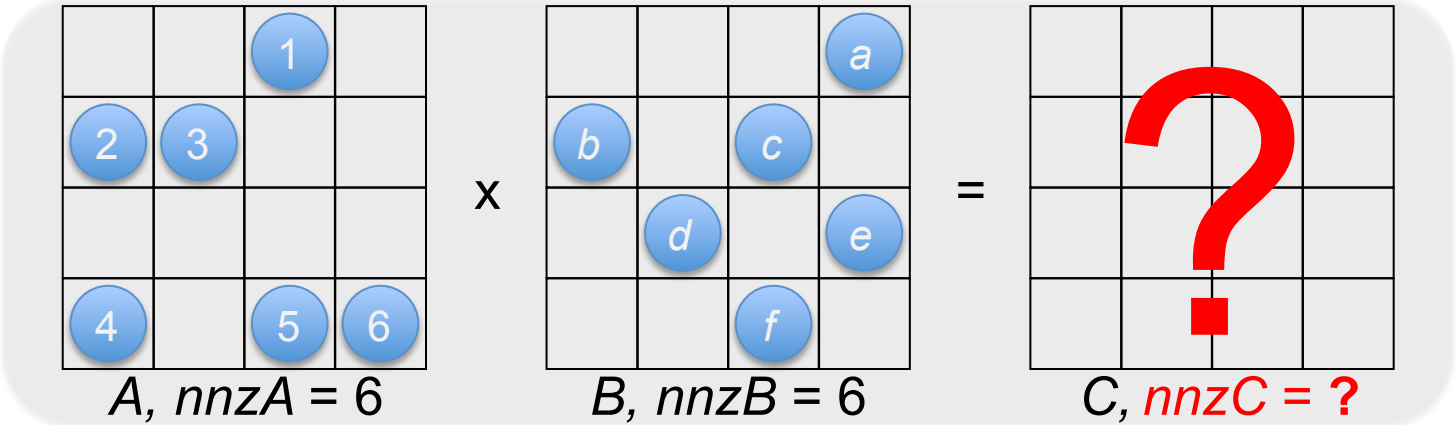
SpGEMM challenge 1 - unknown *nnzC*

- The number of nonzeros in C is unknown in advance. Thus it is not possible to pre-allocate C precisely.



SpGEMM challenge 1 - unknown *nnzC*

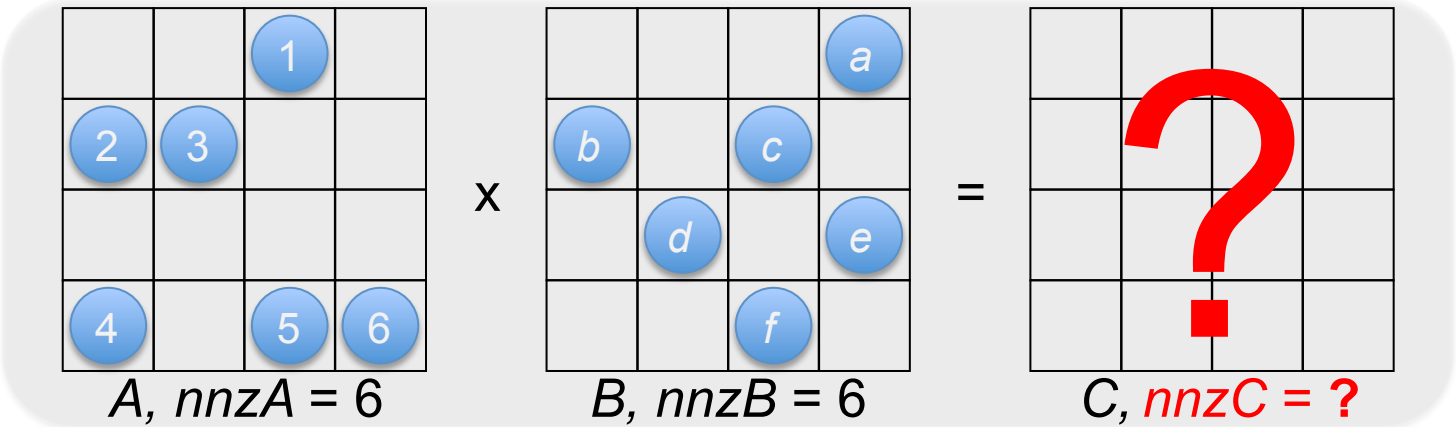
- The number of nonzeros in C is unknown in advance. Thus it is not possible to pre-allocate C precisely.



- A 2-phase method: symbolic SpGEMM first and numeric SpGEMM later?

SpGEMM challenge 1 - unknown *nnzC*

- The number of nonzeros in C is unknown in advance. Thus it is not possible to pre-allocate C precisely.

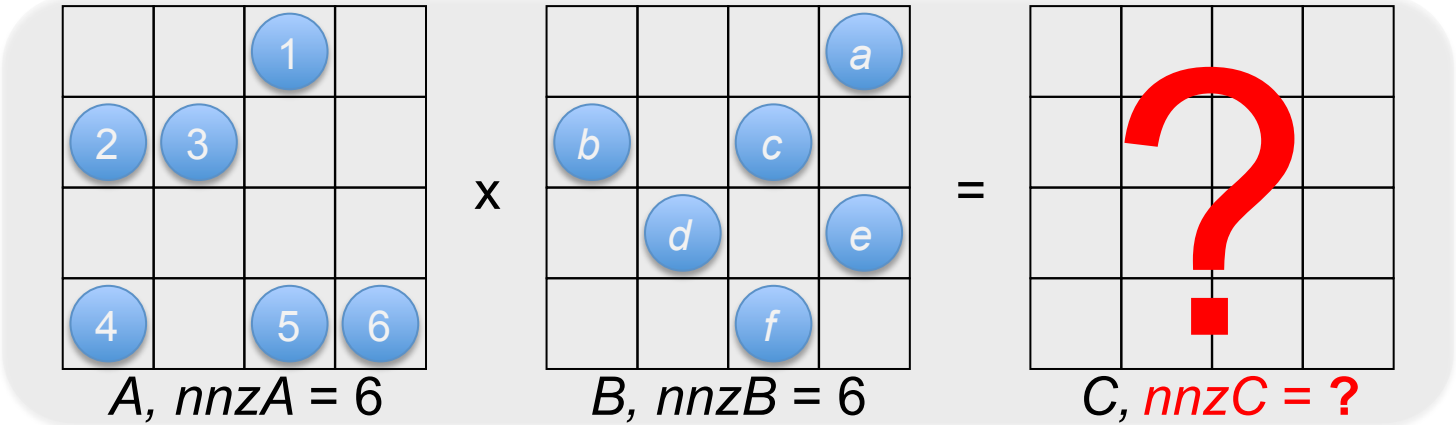


- A 2-phase method: symbolic SpGEMM first and numeric SpGEMM later?

NTNU

SpGEMM challenge 1 - unknown *nnzC*

- The number of nonzeros in C is unknown in advance. Thus it is not possible to pre-allocate C precisely.

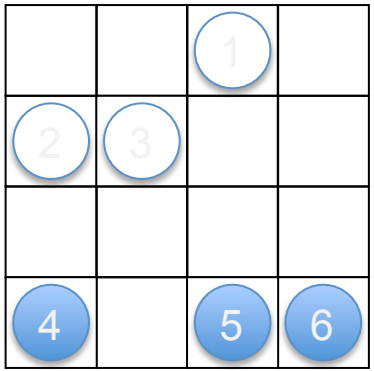


- A 2-phase method: symbolic SpGEMM first and numeric SpGEMM later?

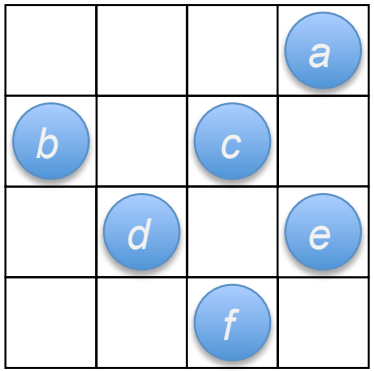
NTNU

SpGEMM challenge 2 - parallel insertion

- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.



A

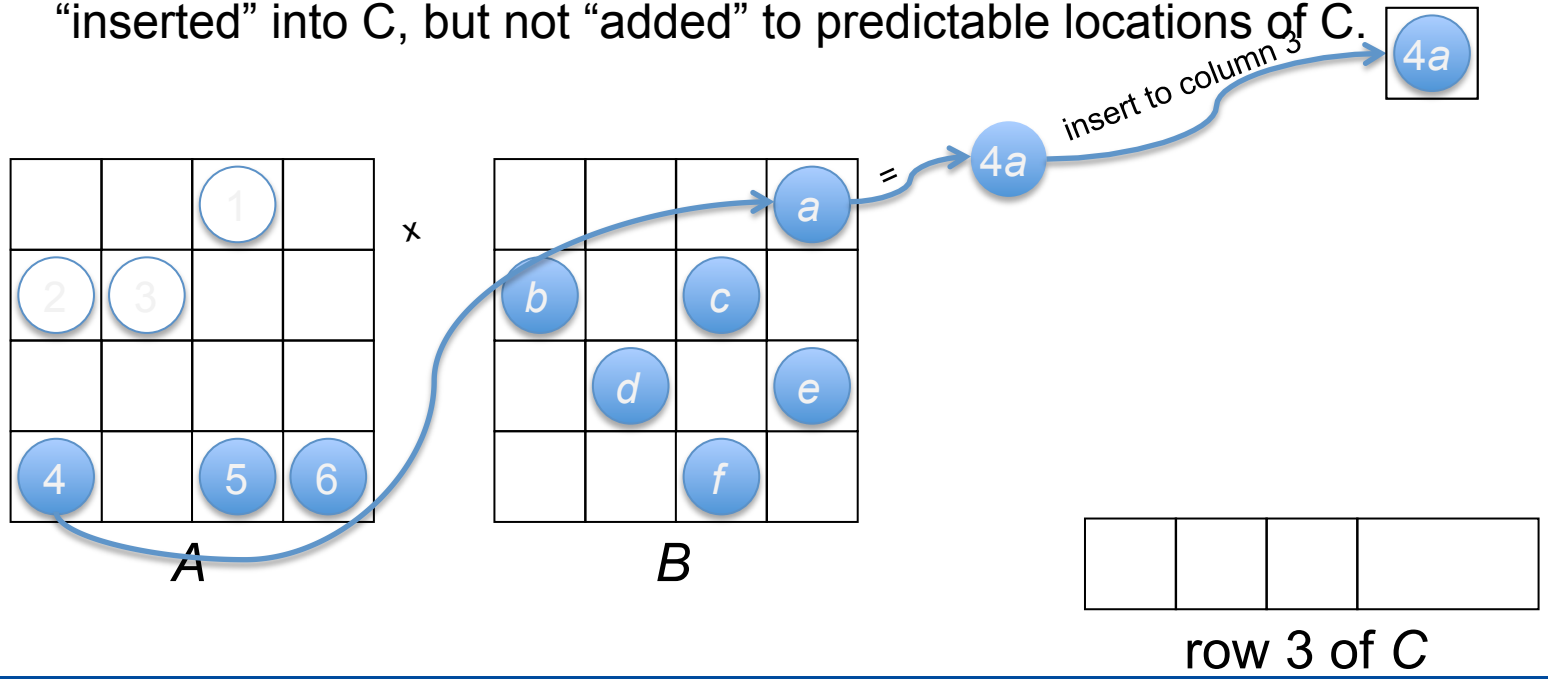


B

row 3 of *C*

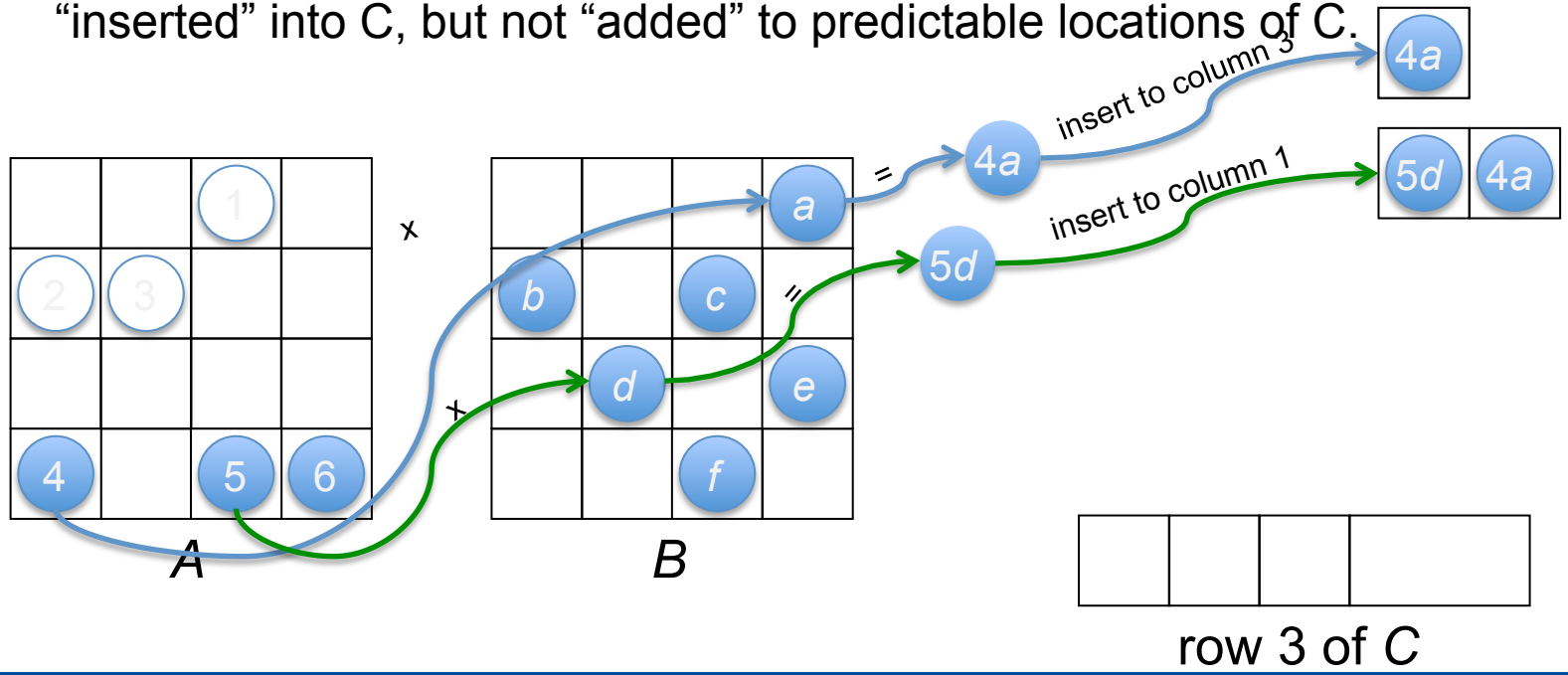
SpGEMM challenge 2 - parallel insertion

- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.



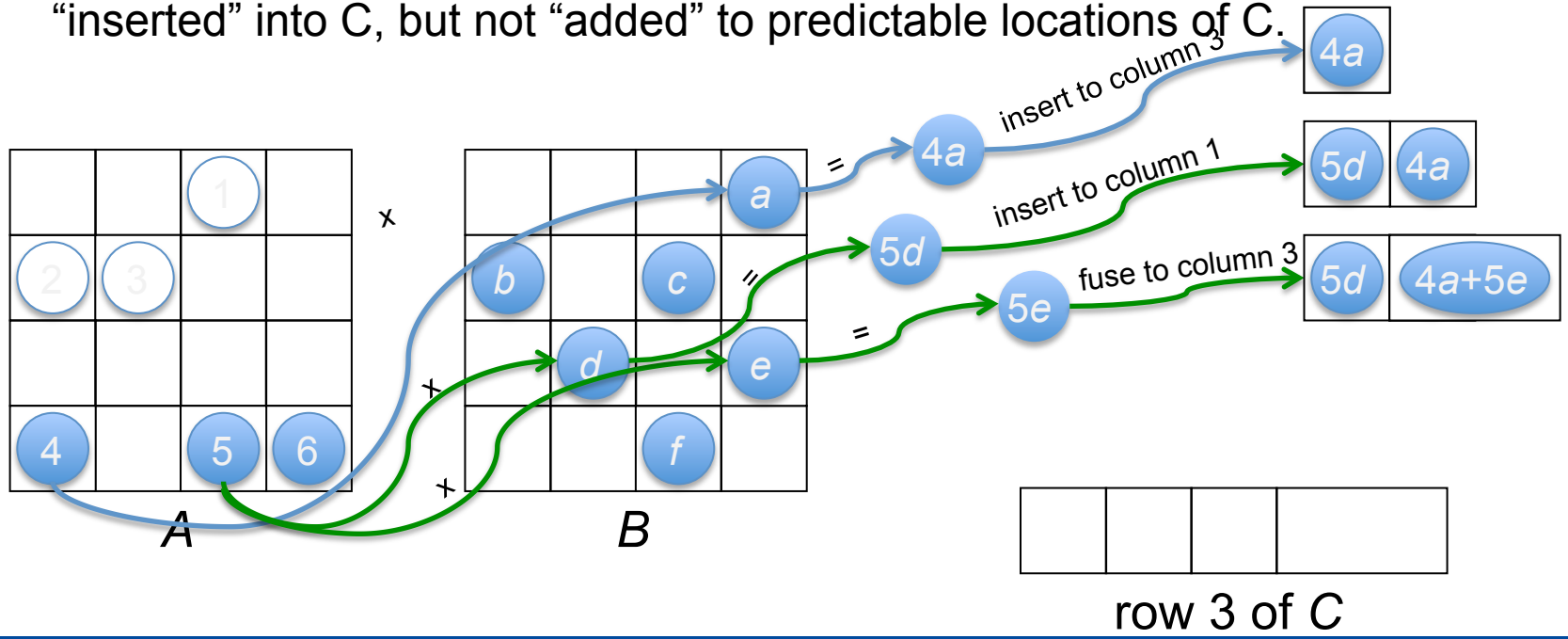
SpGEMM challenge 2 - parallel insertion

- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.



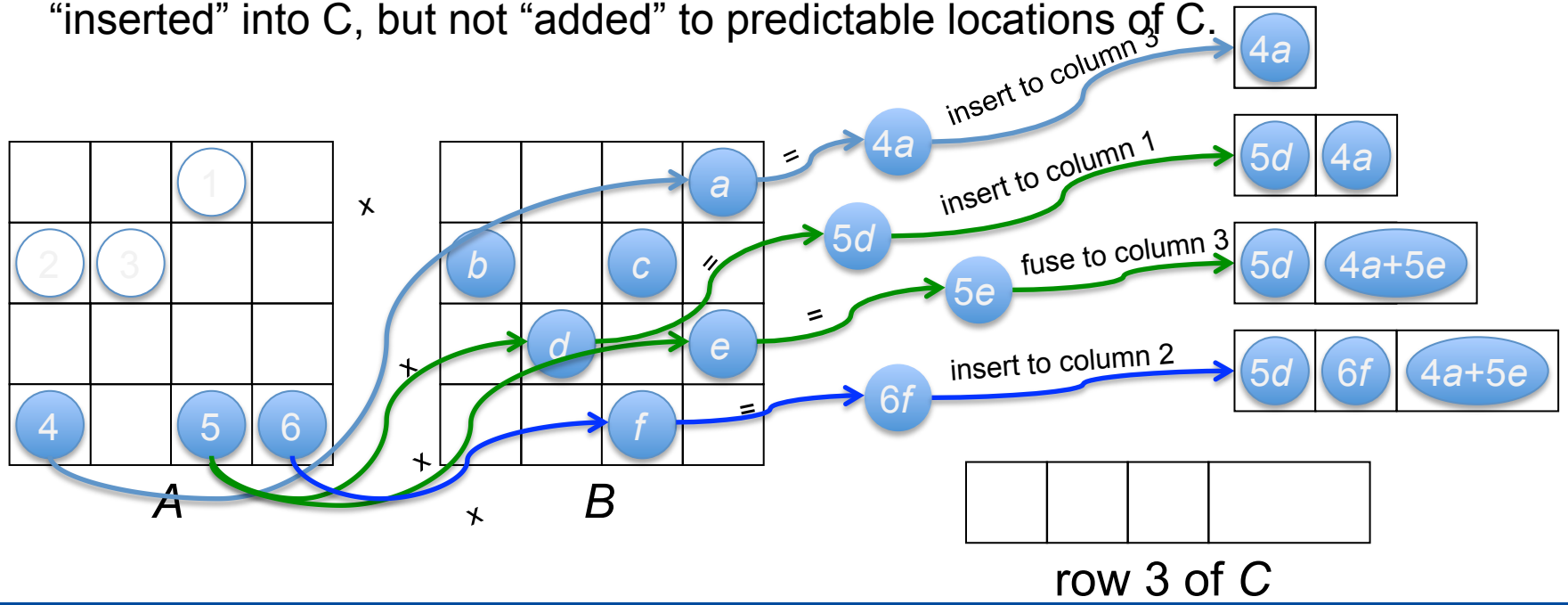
SpGEMM challenge 2 - parallel insertion

- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.



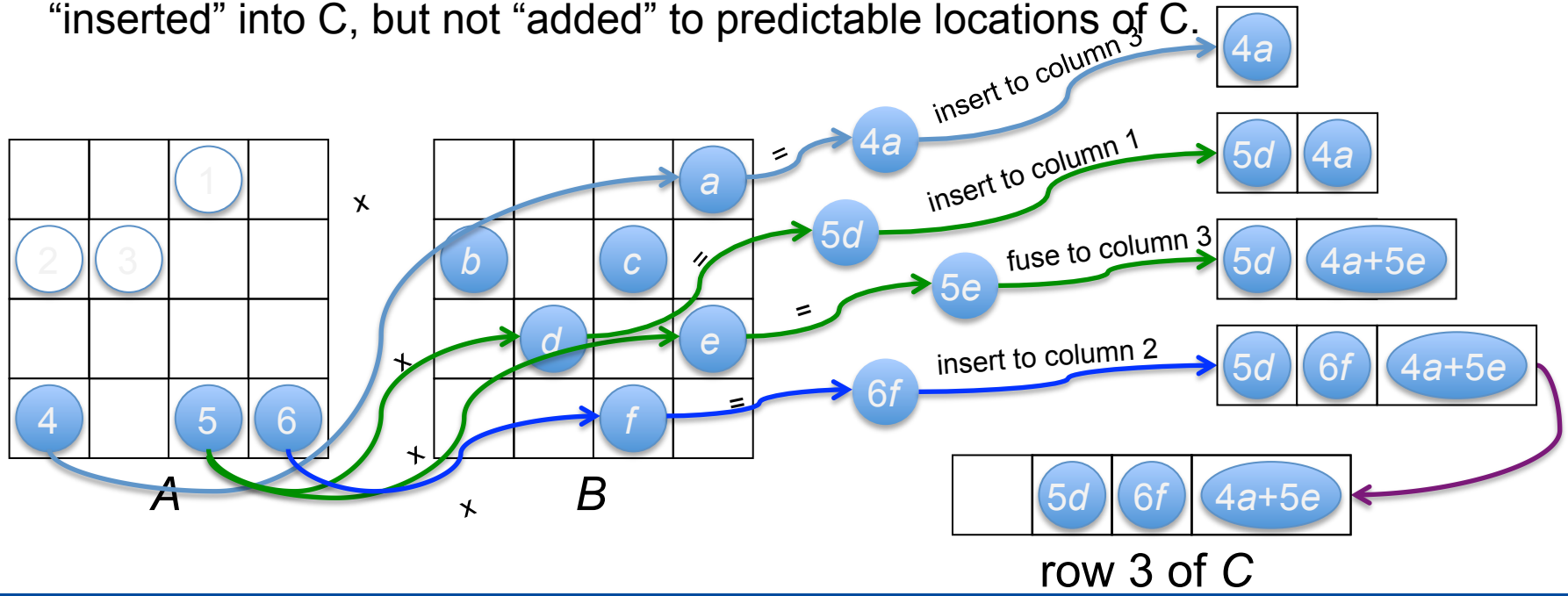
SpGEMM challenge 2 - parallel insertion

- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.



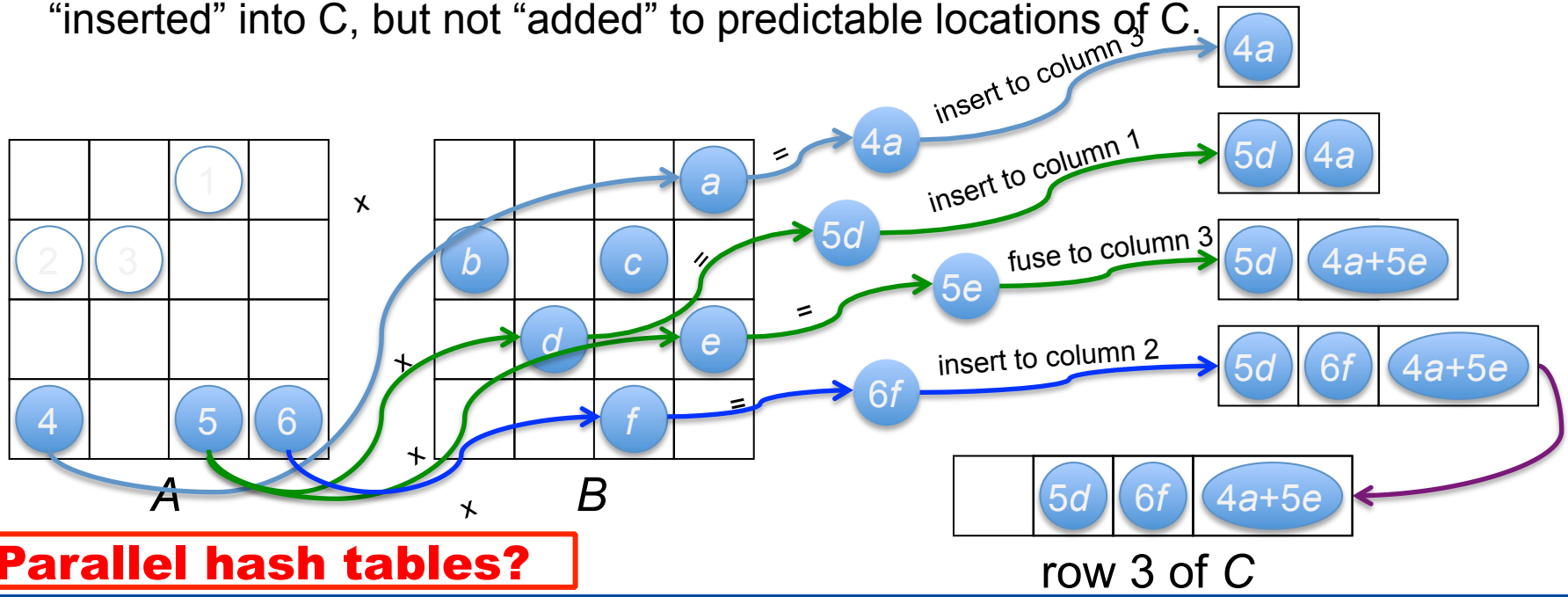
SpGEMM challenge 2 - parallel insertion

- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.



SpGEMM challenge 2 - parallel insertion

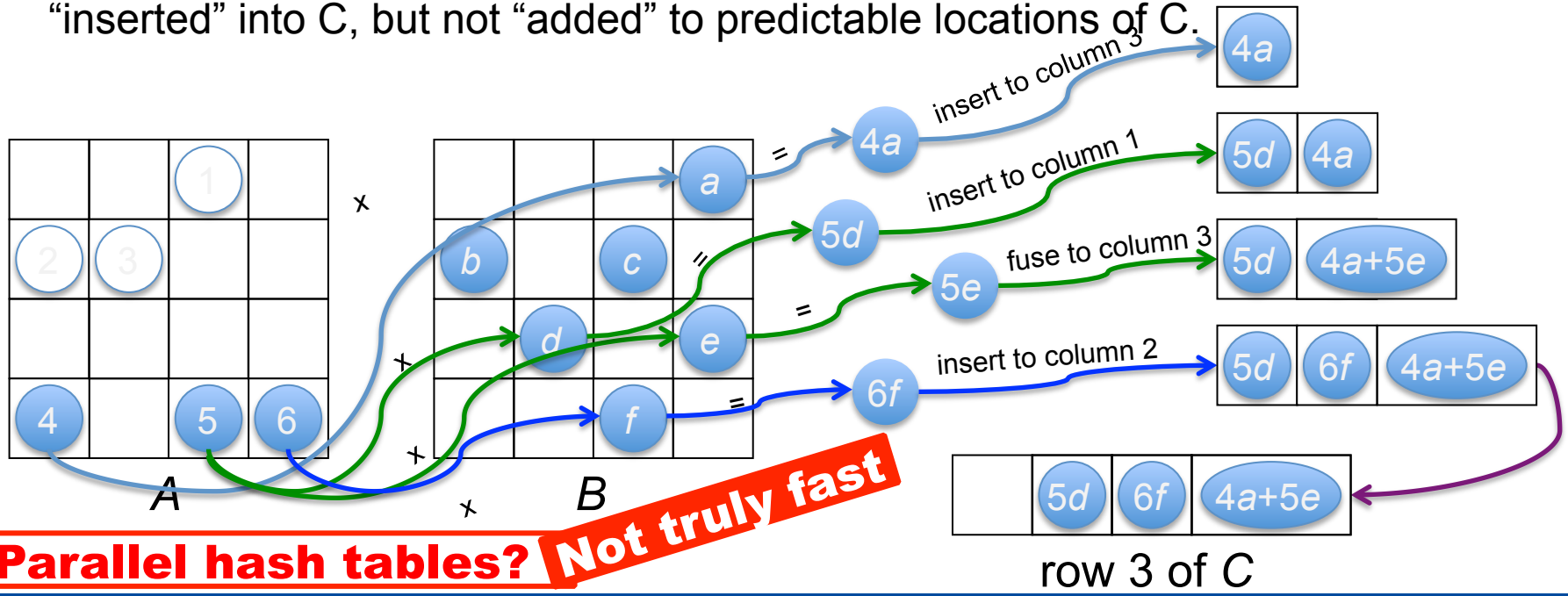
- Because of the compressed sparse format, the result nonzeros are "inserted" into C, but not "added" to predictable locations of C.



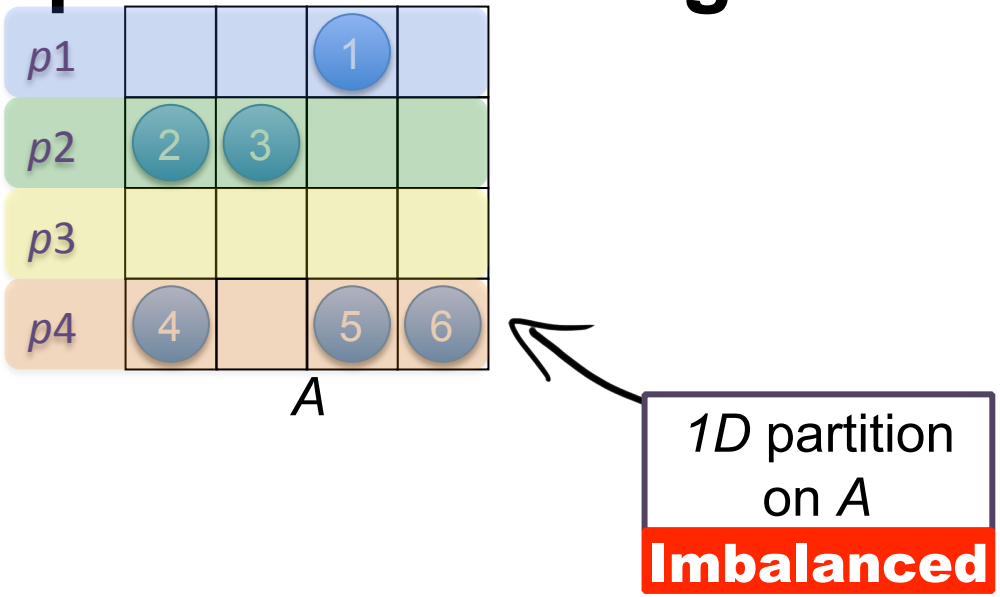
Parallel hash tables?

SpGEMM challenge 2 - parallel insertion

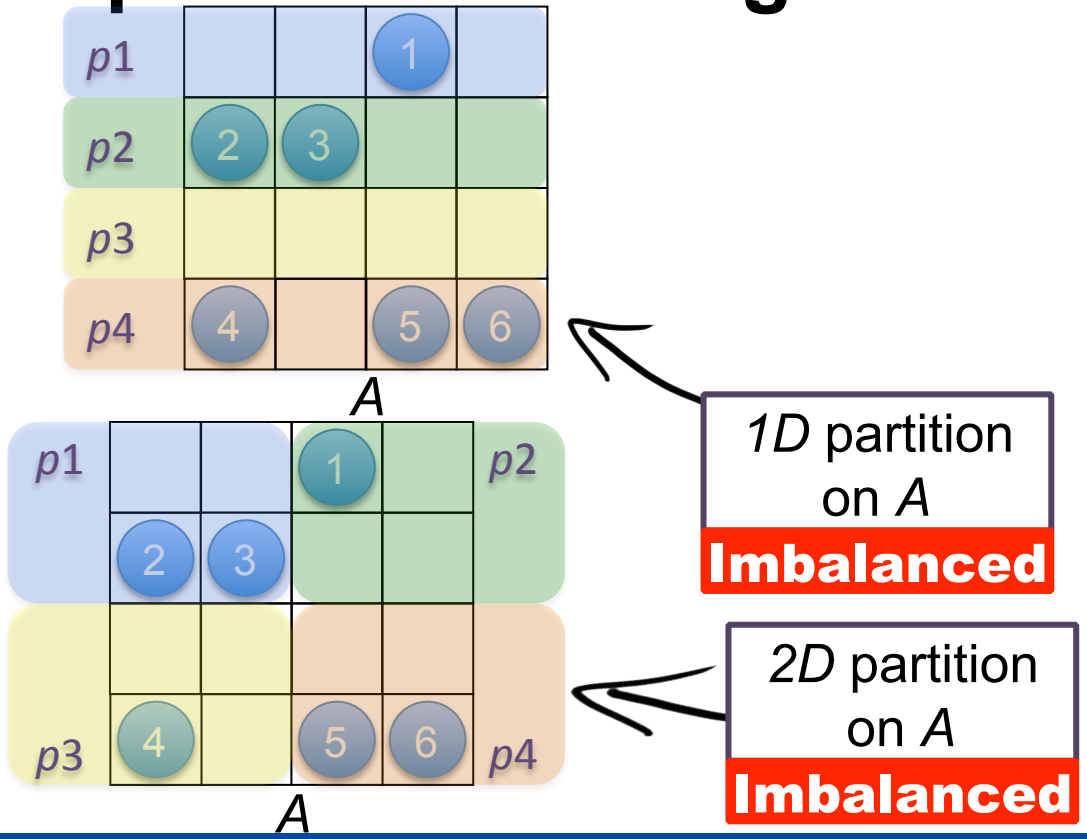
- Because of the compressed sparse format, the result nonzeros are "inserted" into C, but not "added" to predictable locations of C.



SpGEMM challenge 3 - load balancing



SpGEMM challenge 3 - load balancing



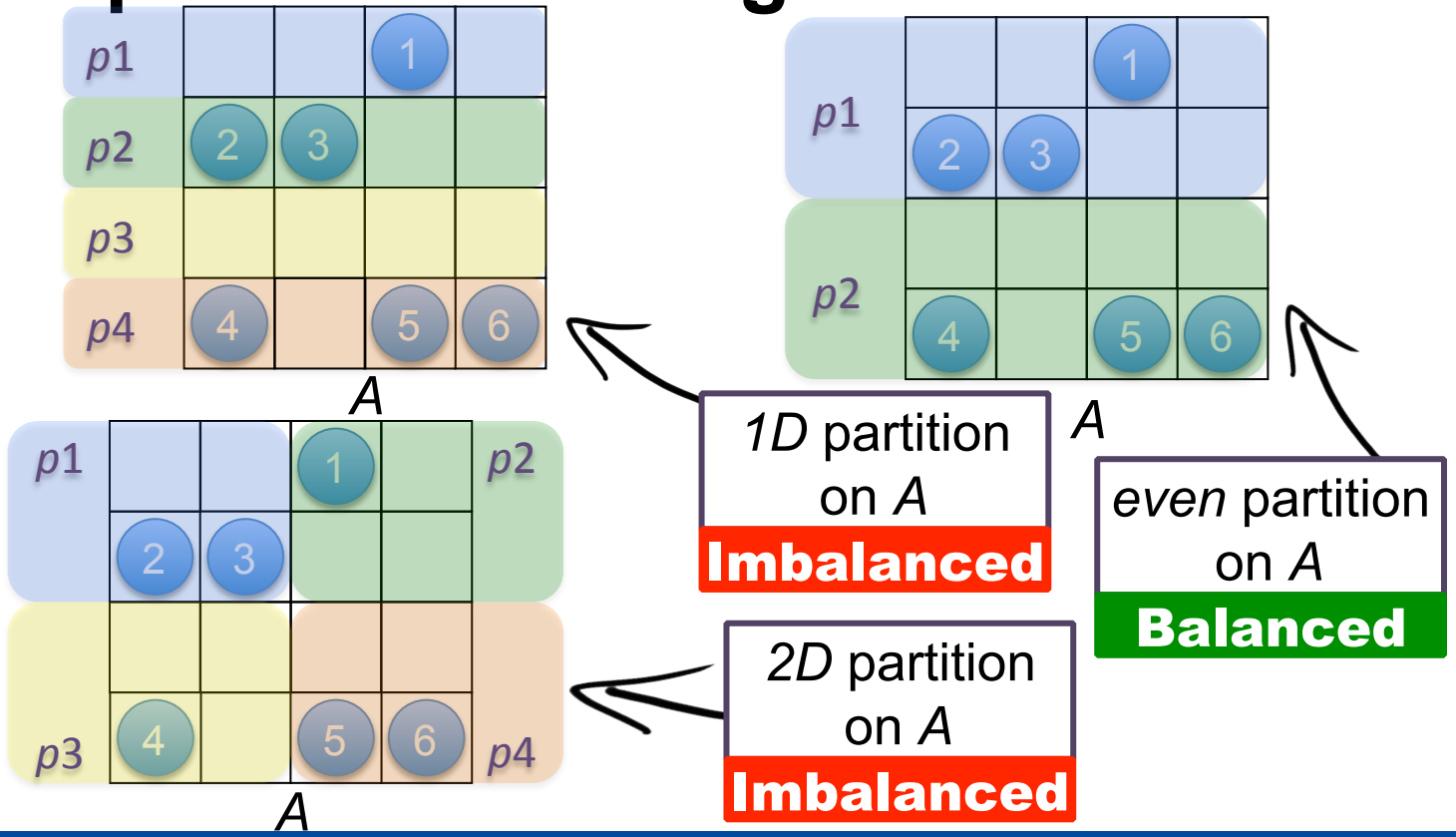
1D partition on *A*

Imbalanced

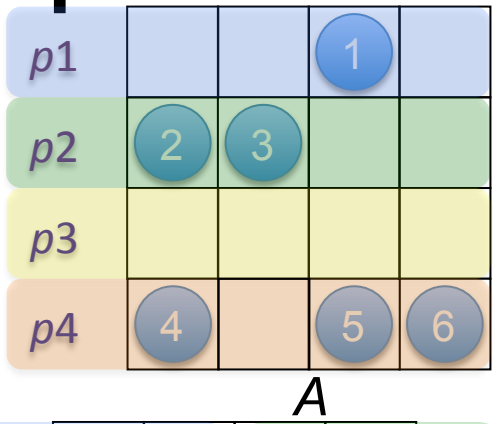
2D partition on *A*

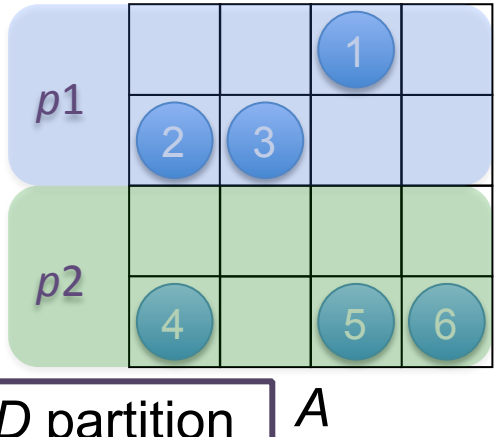
Imbalanced

SpGEMM challenge 3 - load balancing

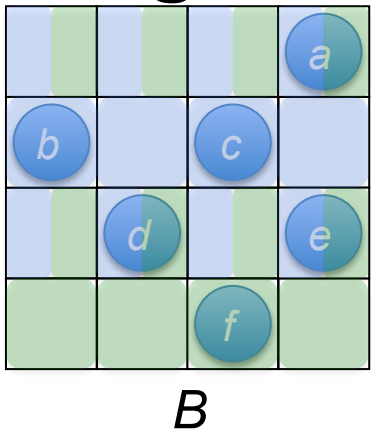


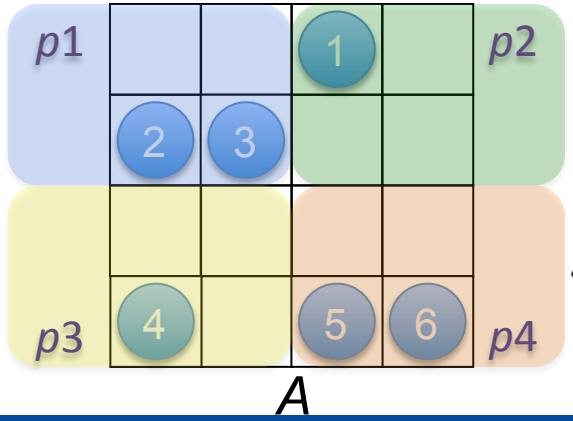
SpGEMM challenge 3 - load balancing





x





1D partition on *A*

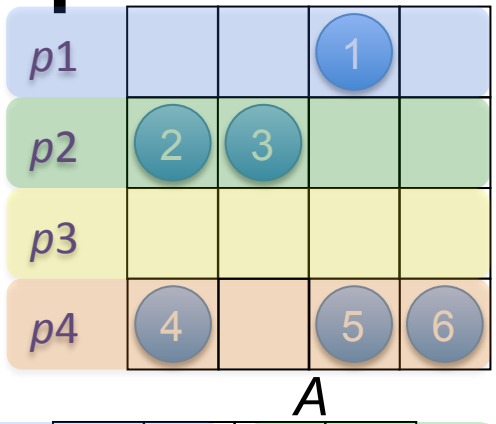
Imbalanced

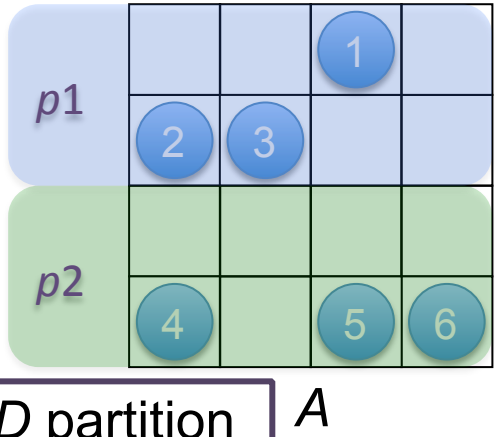
2D partition on *A*

Imbalanced

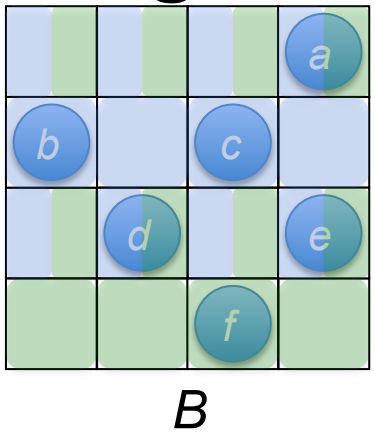
even partition on *A*	Accessing on *B*
Balanced	**Imbalanced**

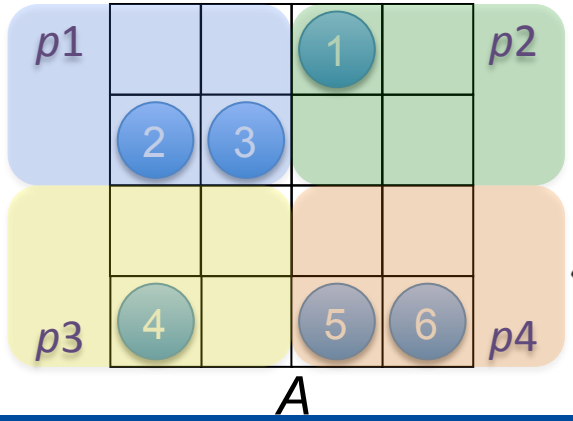
SpGEMM challenge 3 - load balancing





x





1D partition on *A*

Imbalanced

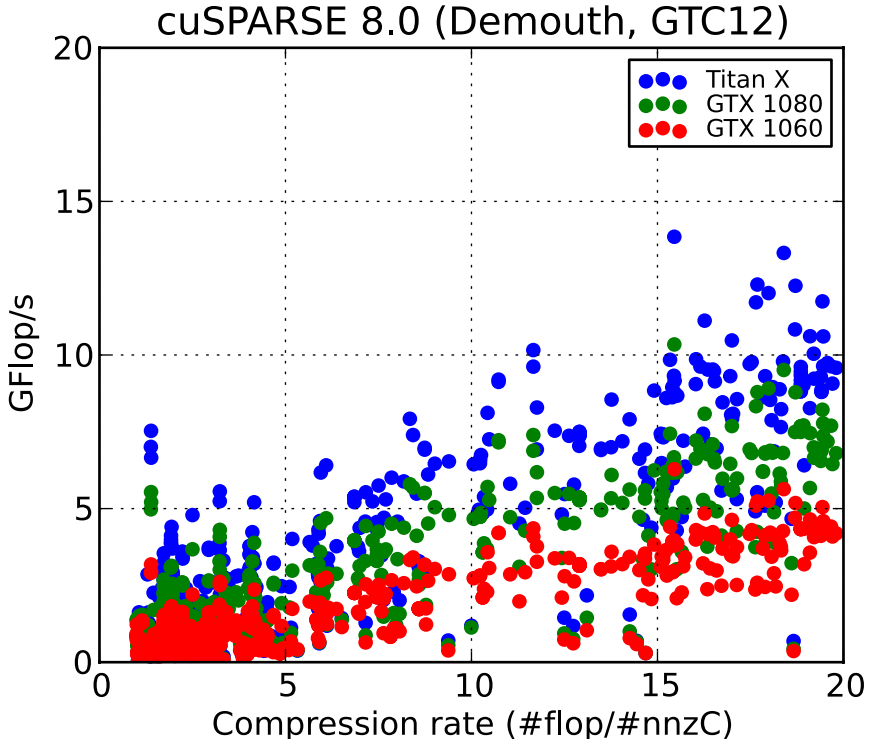
2D partition on *A*

Imbalanced

even partition on *A*	Accessing on *B*
Balanced	**Imbalanced**

Overall - imbalanced

Scalability of SpGEMM in cuSPARSE

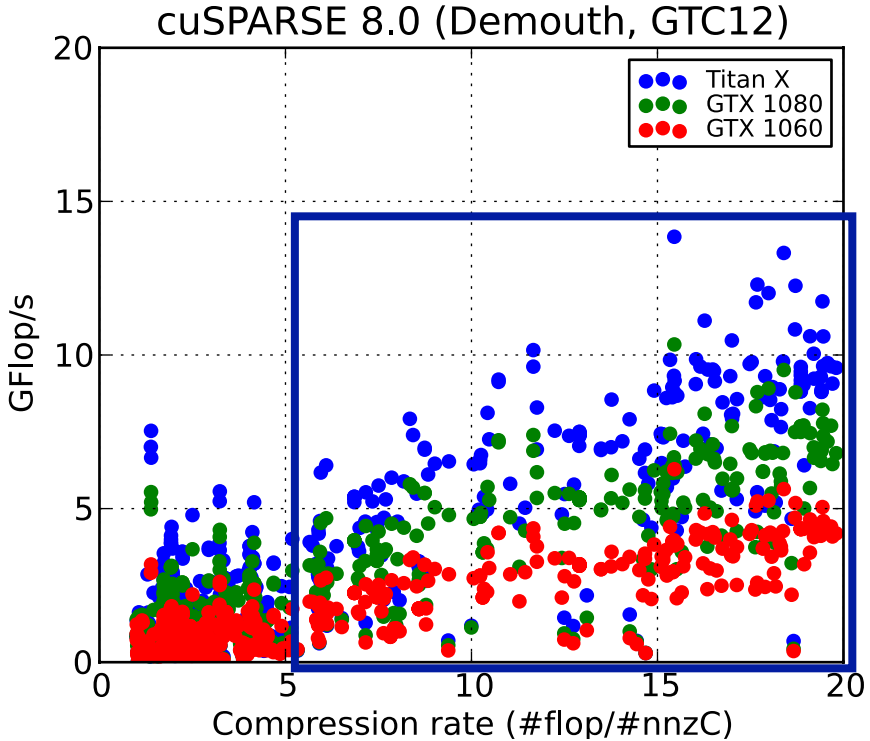


more "insertion"

more "fusion"

NTNU

Scalability of SpGEMM in cuSPARSE



SpGEMM perf. in general scales with hardware perf.

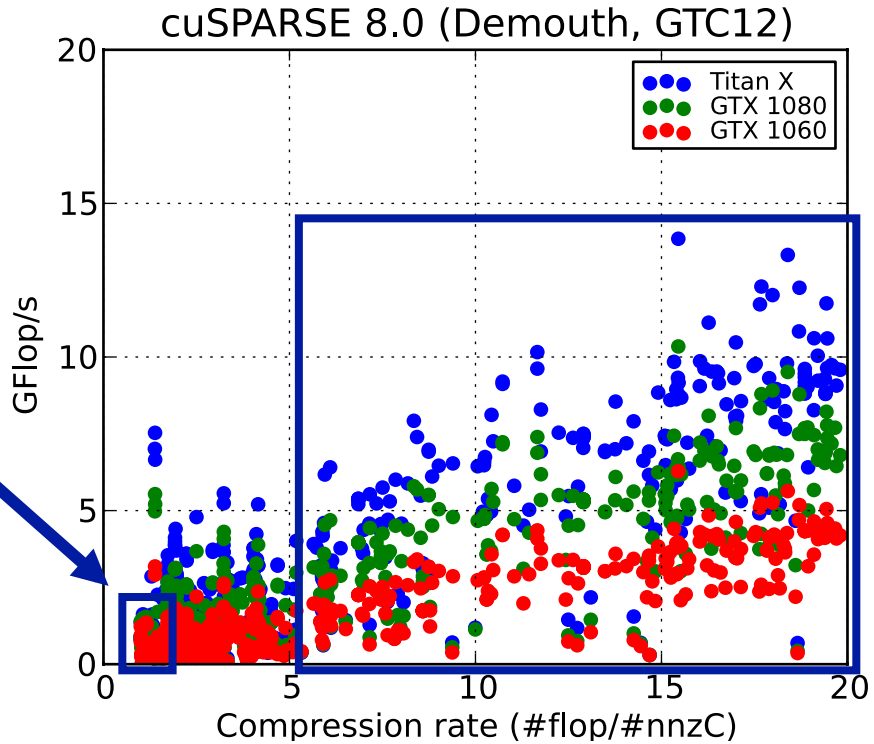
more "insertion"

more "fusion"

NTNU

Scalability of SpGEMM in cuSPARSE

SpGEMM perf. does not scale with hardware perf.



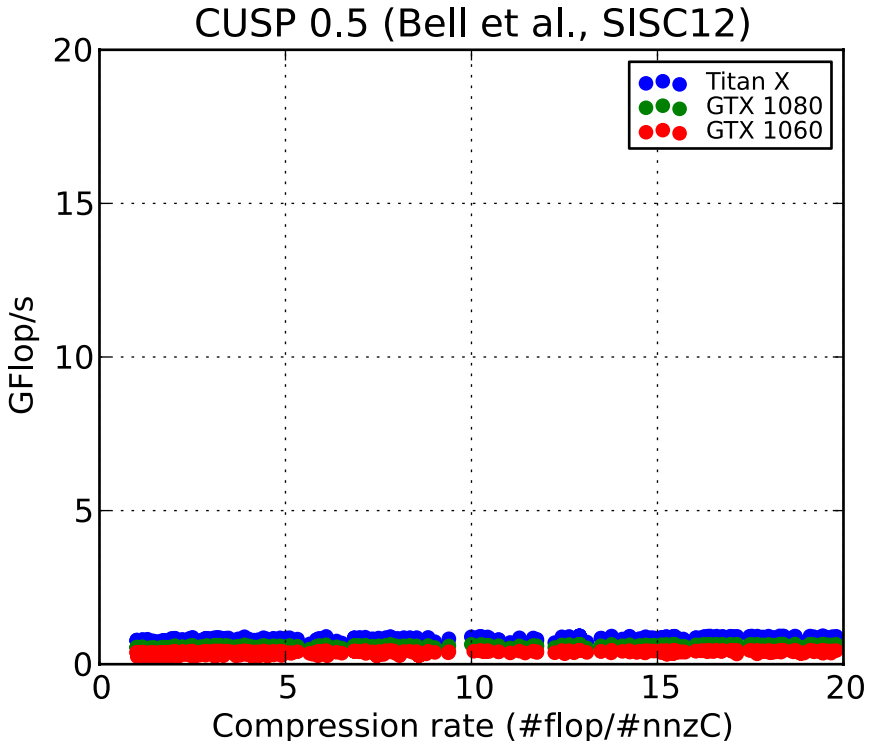
SpGEMM perf. in general scales with hardware perf.

more "insertion"

more "fusion"

NTNU

Scalability of SpGEMM in CUSP



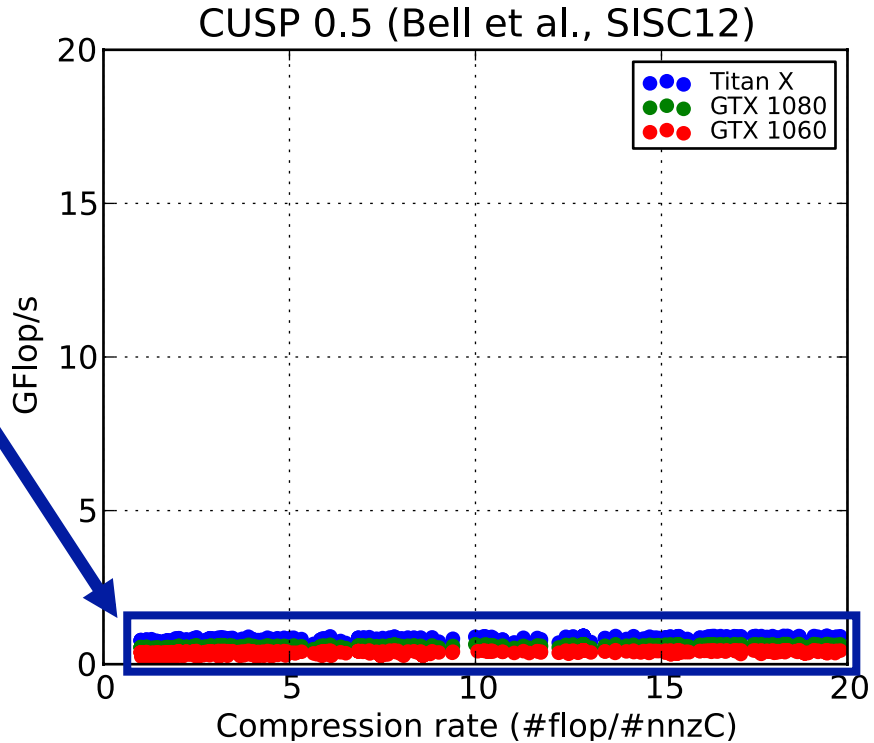
more "insertion"

more "fusion"

NTNU

Scalability of SpGEMM in CUSP

SpGEMM perf. always scales with hardware perf.

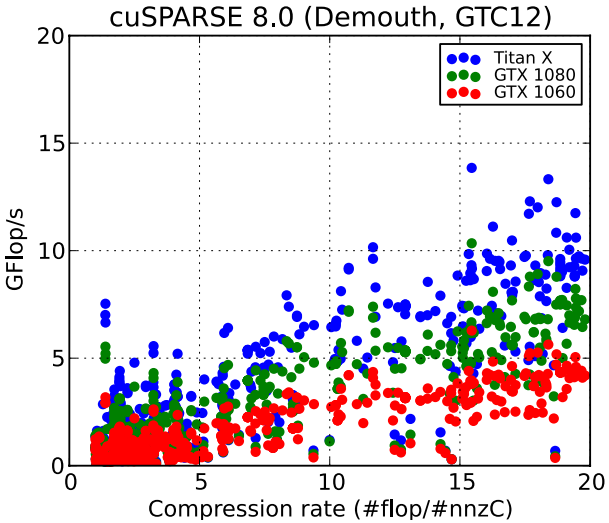


more "insertion"

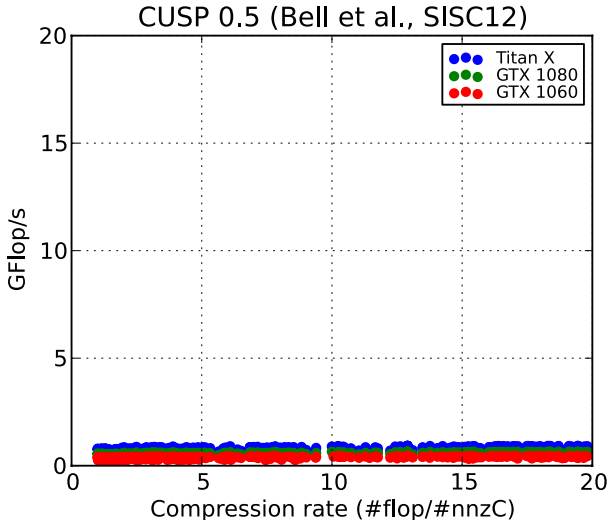
more "fusion"

NTNU

Contradiction btw. scalability and high perf.

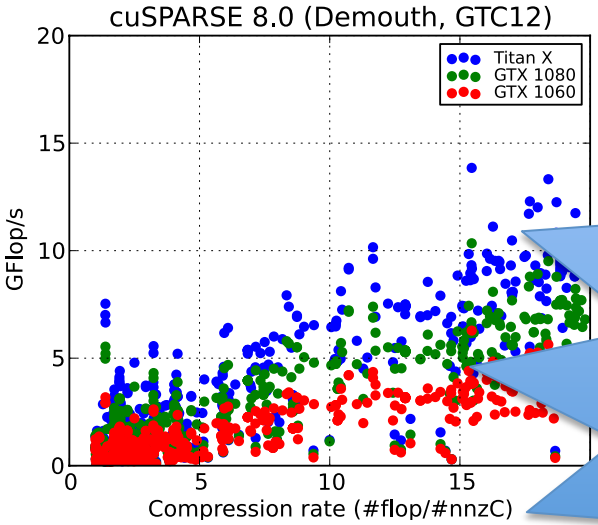


For regular matrices, high performance method cannot scale or scale imperfectly.

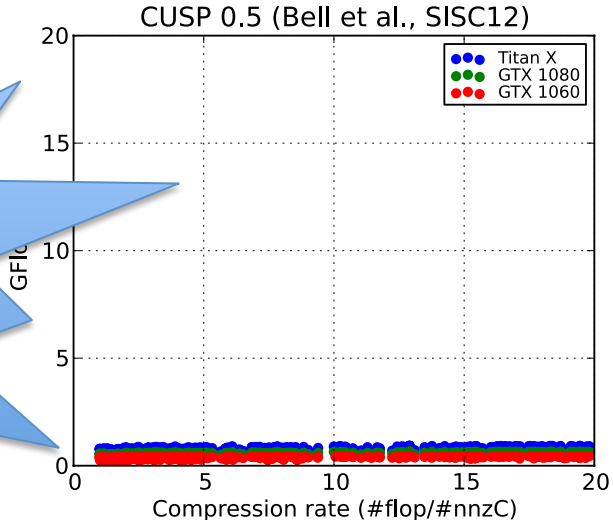


For both regular and irregular matrices, scalable method gives low performance.

Contradiction btw. scalability and high perf.



Does there exist the best of both worlds (scalability + high performance)?

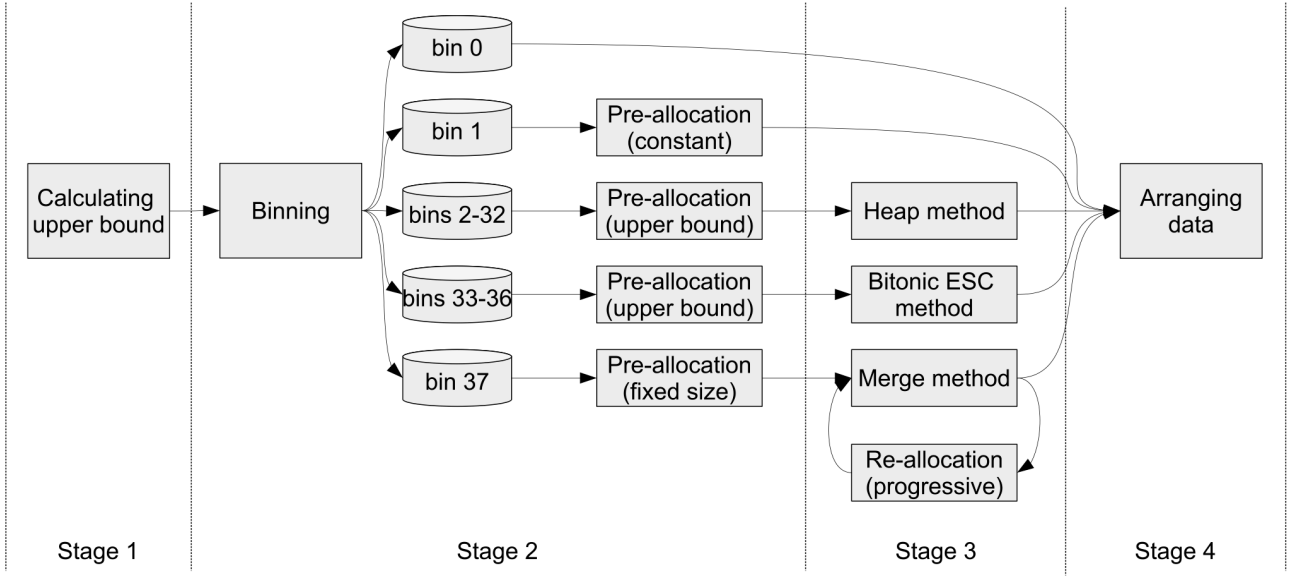


For regular matrices, high performance method cannot scale or scale imperfectly.

For both regular and irregular matrices, scalable method gives low performance.

NTNU

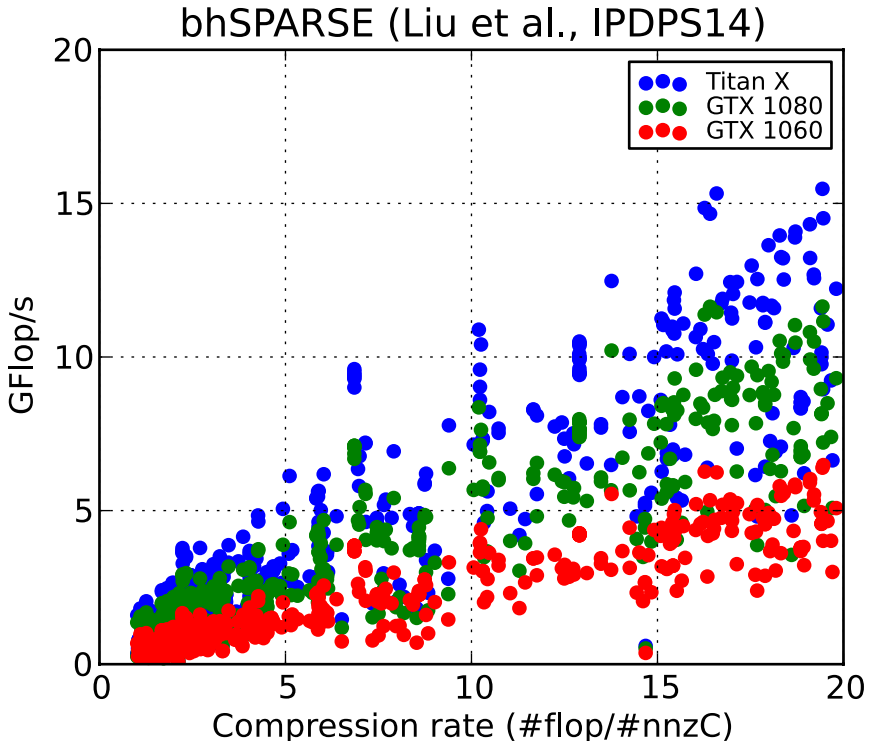
An SpGEMM framework (our work)



Weifeng Liu, Brian Vinter. **A Framework for General Sparse Matrix-Matrix Multiplication on GPUs and Heterogeneous Processors**. *JPDC. 2015.* (extended from *IPDPS14*)

Weifeng Liu, Brian Vinter. **An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data**. *IPDPS14*.

Scalability of SpGEMM in bhSPARSE

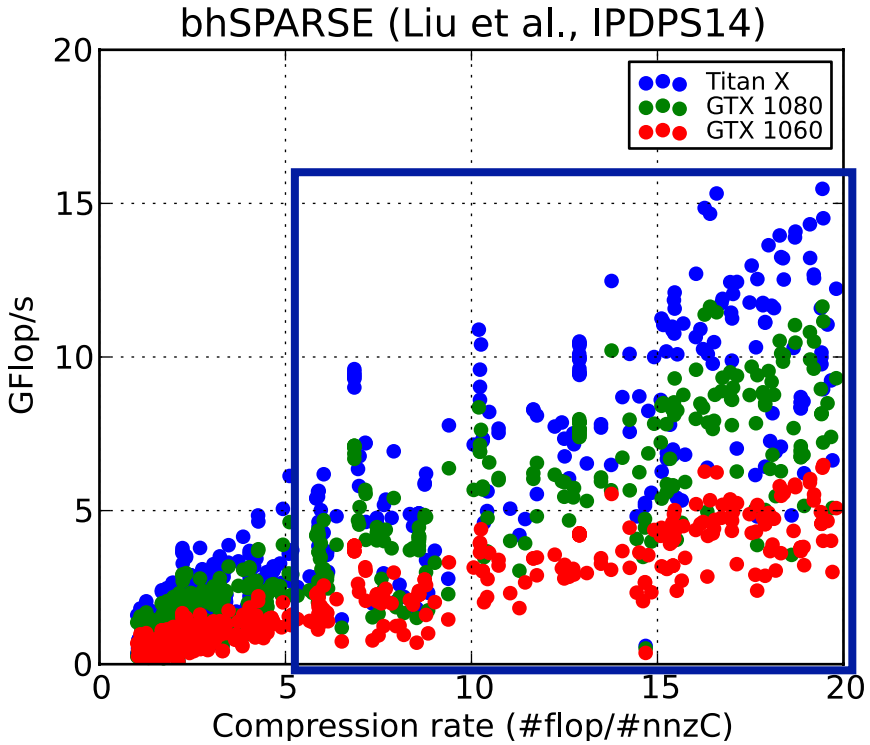


more "insertion"

more "fusion"

NTNU

Scalability of SpGEMM in bhSPARSE



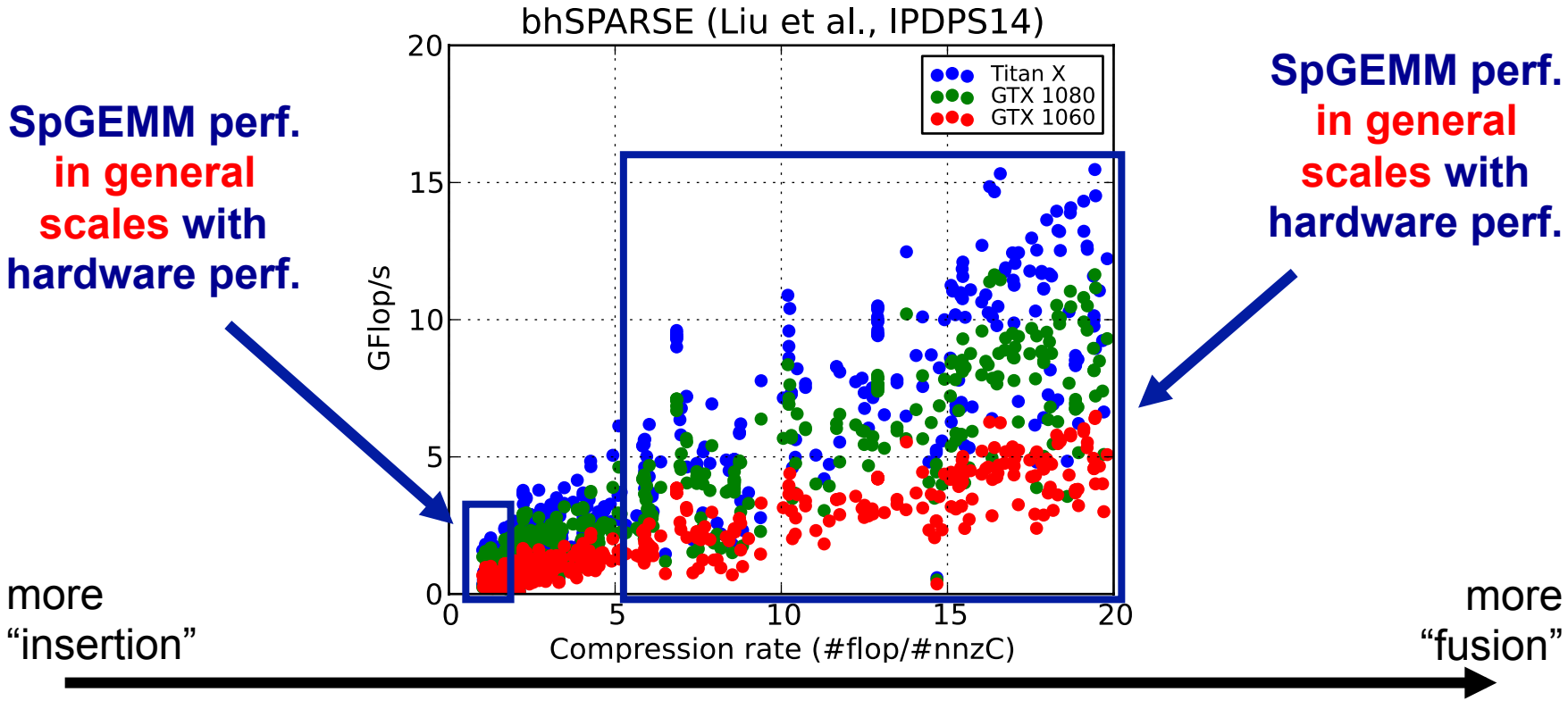
SpGEMM perf. in general scales with hardware perf.

more "insertion"

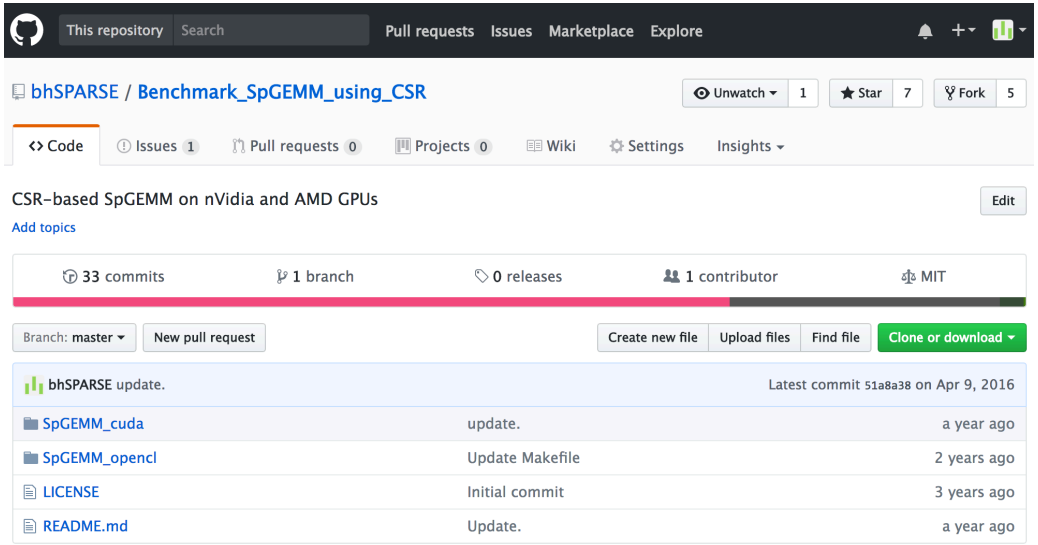
more "fusion"

NTNU

Scalability of SpGEMM in bhSPARSE



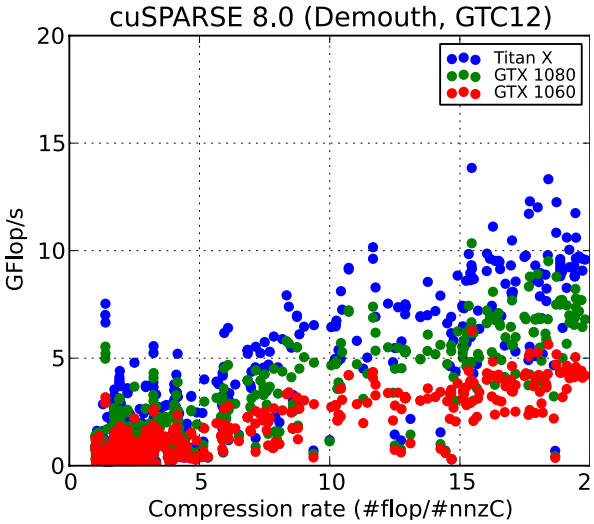
Source code of bhSPARSE on Github

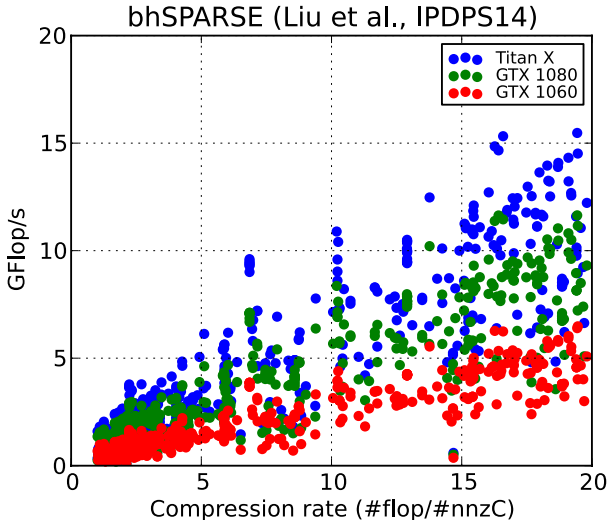


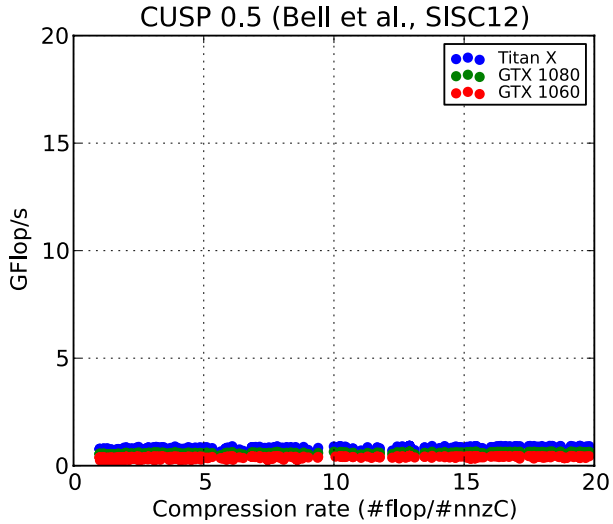
https://github.com/bhSPARSE/Benchmark_SpGEMM_using_CSR

Has been integated into AMD's clSPARSE library.
https://github.com/clMathLibraries/clSPARSE

Contradiction btw. scalability and high perf.





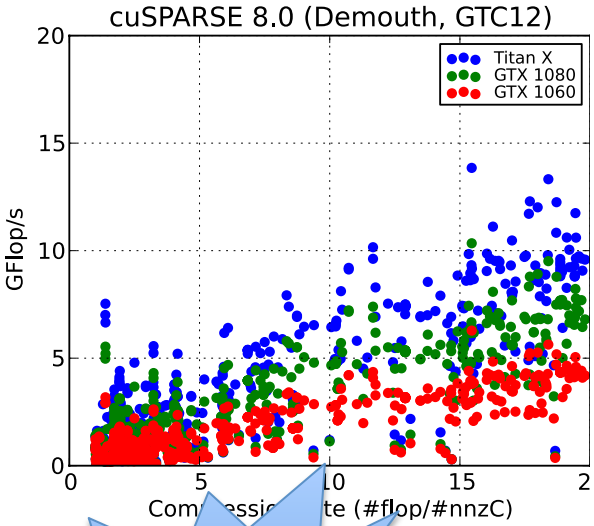


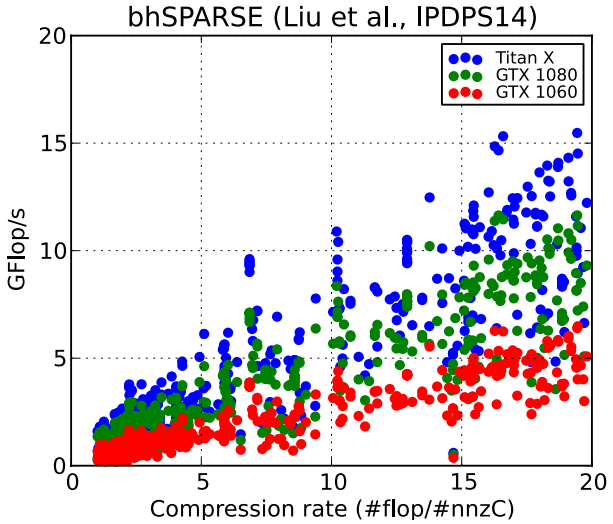
For regular matrices, high performance methods cannot scale or scale imperfectly.

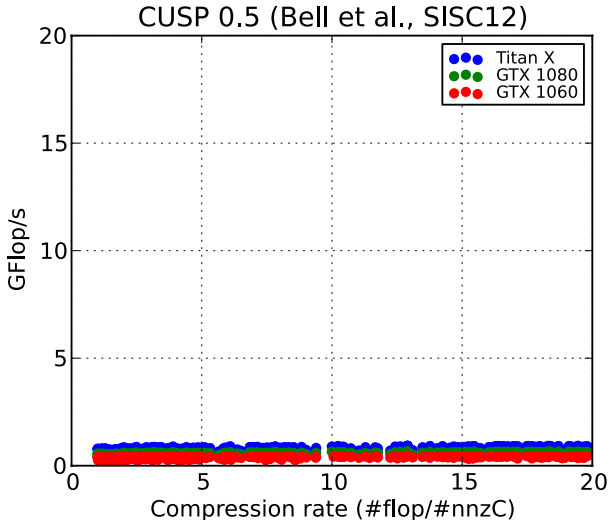
For both regular and irregular matrices, scalable method gives low performance.

NTNU

Contradiction btw. scalability and high perf.







Take the two at once?

For regular matrices, high performance methods cannot scale or scale imperfectly.

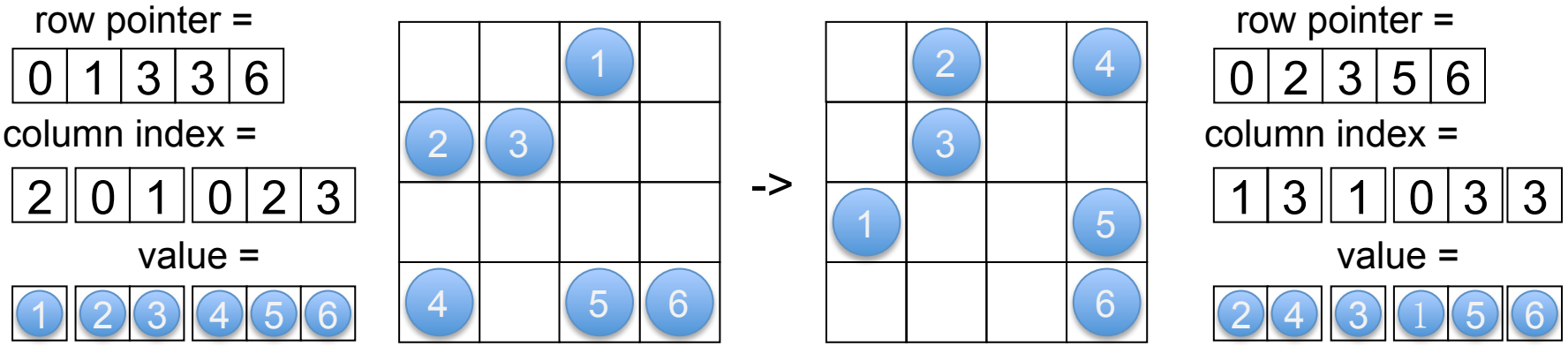
For both regular and irregular matrices, scalable method gives low performance.

NTNU

Kernel 3. Sparse Transposition (SpTRANS)

SpTRANS

- Transpose a sparse matrix *A* (in CSR) to a sparse matrix *B* (i.e., A^T, in CSR). This is equivalent to a conversion from CSR to CSC, or vice versa.

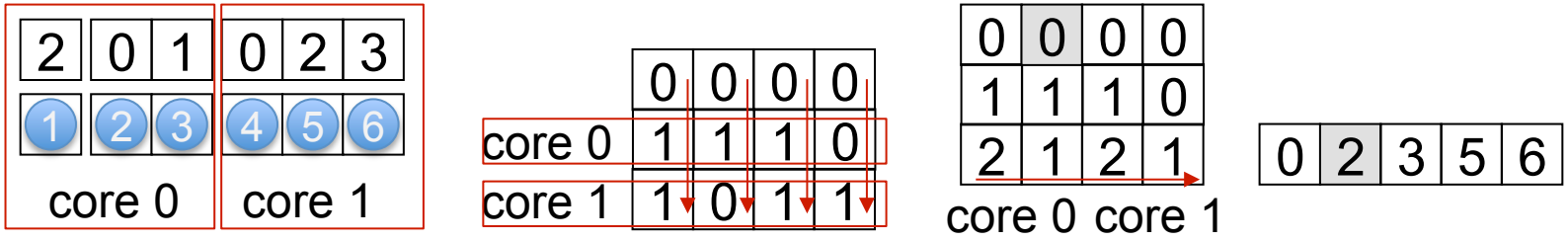


A (4x4) sparse, 6 nonzeros

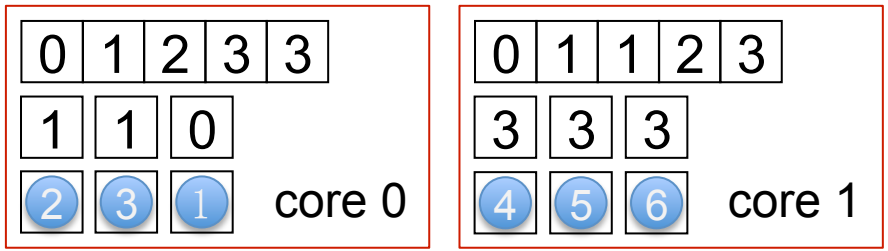
A^T (4x4) sparse, 6 nonzeros

ScanTrans and MergeTrans (our work)

- ScanTrans: evenly assign nonzeros, then vertical/horizontal scan for offsets

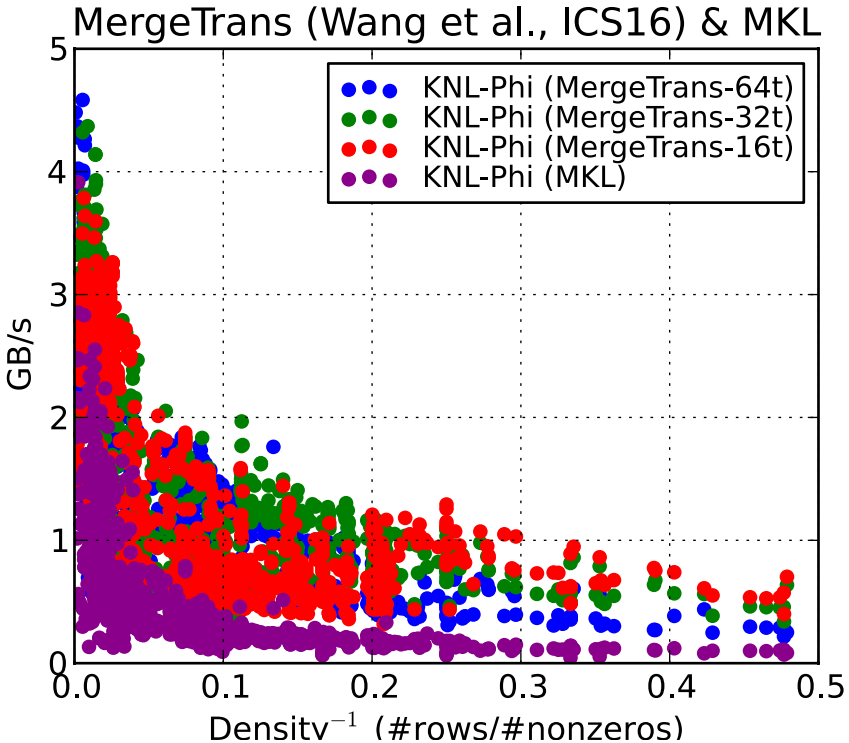


- MergeTrans: evenly assign nonzeros, transpose submatrices, then merge



Hao Wang, Weifeng Liu, Kaixi Hou, Wu-chun Feng. **Parallel Transposition of Sparse Data Structures**. *ICS16*.

Scalability of MergeTrans on KNL



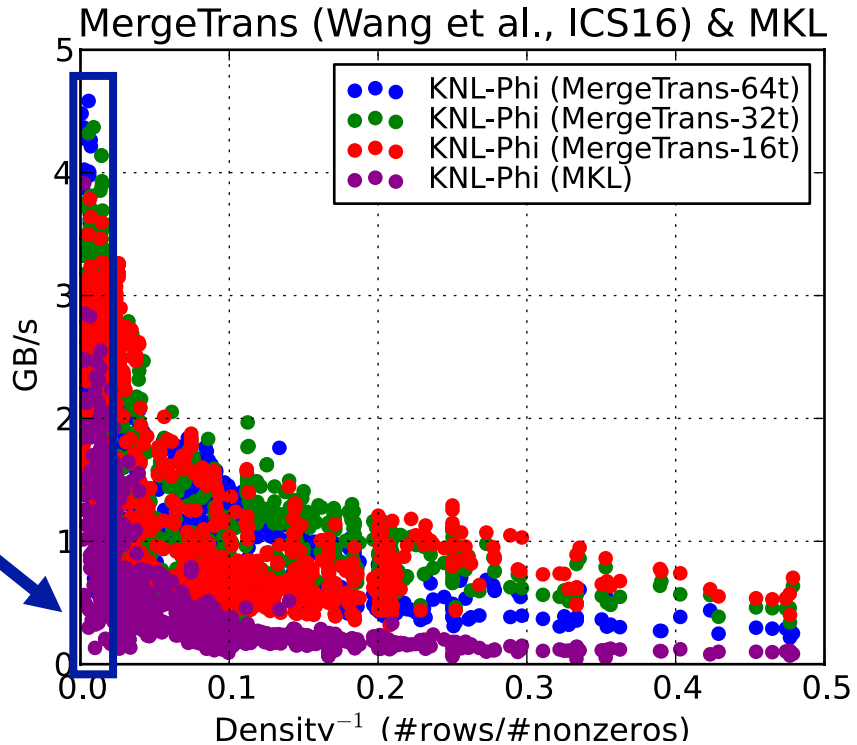
more "dense"

more "sparse"

NTNU

Scalability of MergeTrans on KNL

SpTRANS perf. in general scales with hardware perf. (64t is the best)

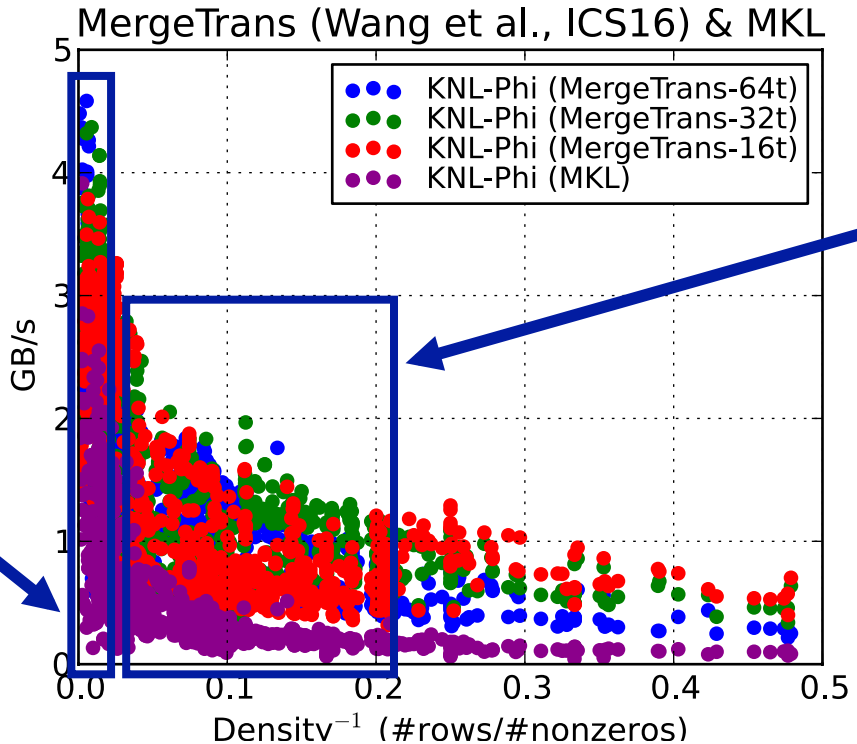


more "dense"

more "sparse"

NTNU

Scalability of MergeTrans on KNL



SpTRANS perf. in general scales with hardware perf. (64t is the best)

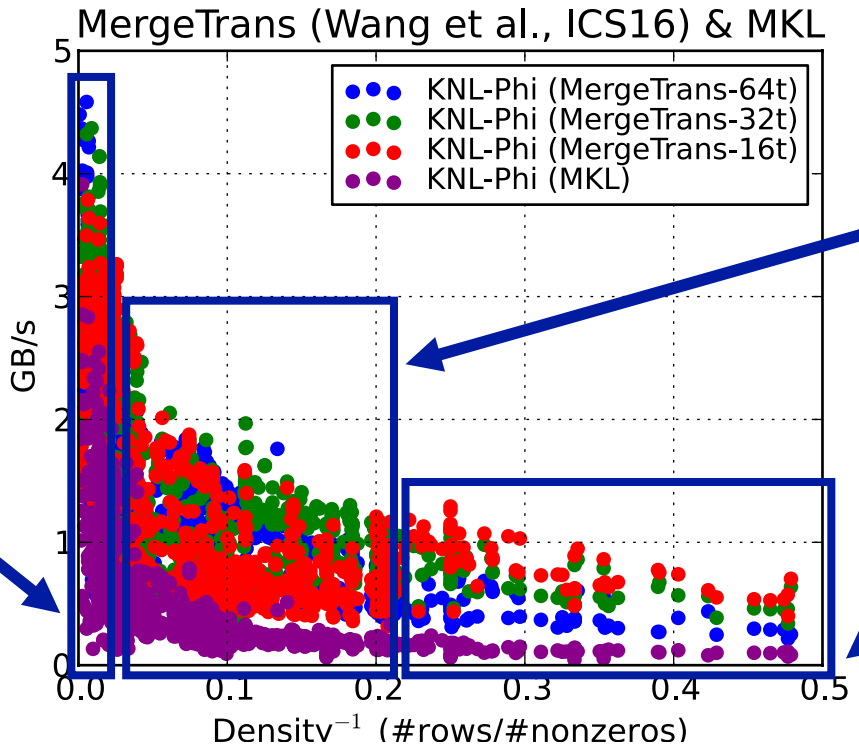
SpTRANS perf. does not scale with hardware perf. (32t is the best)

more "dense"

more "sparse"

NTNU

Scalability of MergeTrans on KNL



SpTRANS perf. in general scales with hardware perf. (64t is the best)

SpTRANS perf. does not scale with hardware perf. (32t is the best)

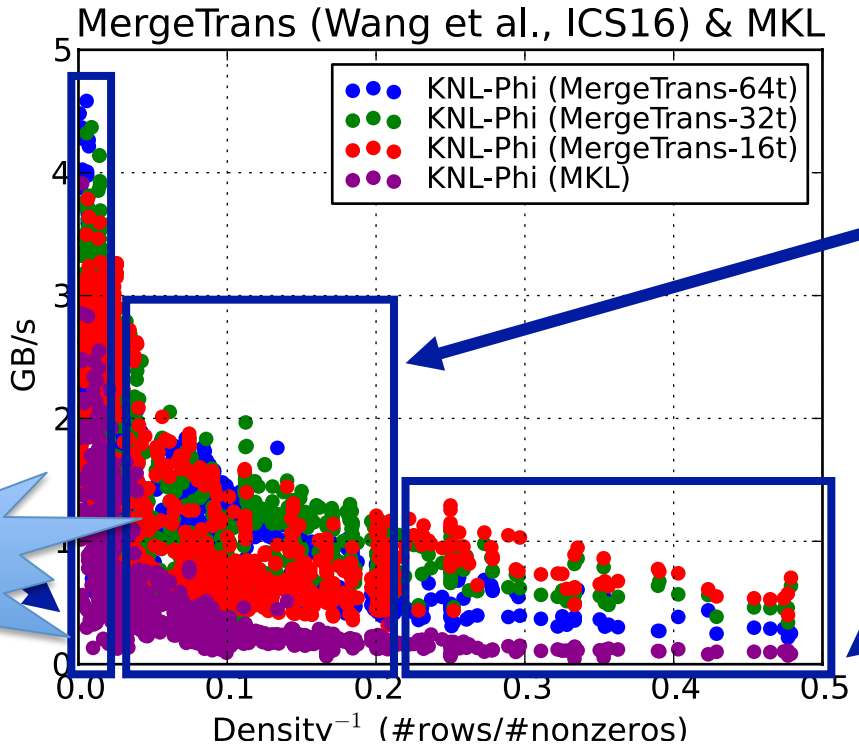
SpTRANS perf. inversely scales with hardware perf. (16t is the best)

more "dense"

more "sparse"

NTNU

Scalability of MergeTrans on KNL



SpTRANS perf. in general scales with hardware perf. (64t is the best)

SpTRANS perf. does not scale with hardware perf. (32t is the best)

SpTRANS perf. inversely scales with hardware perf. (16t is the best)

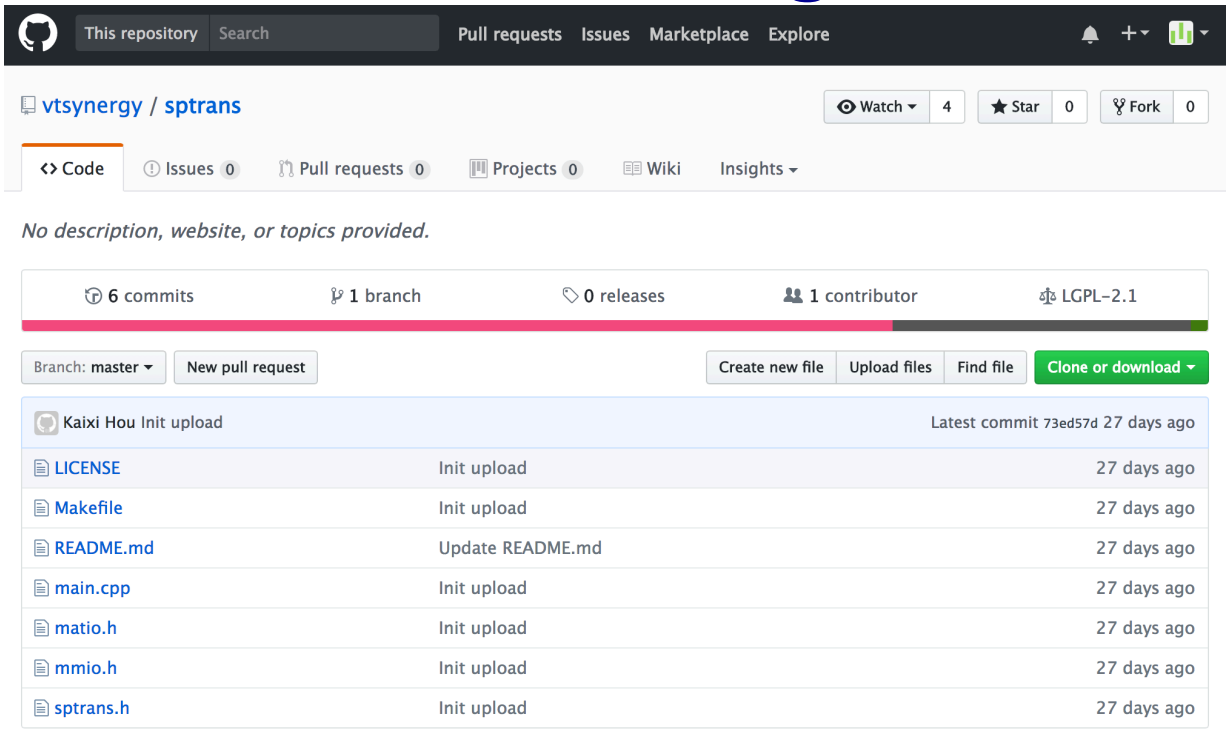
Scalable methods?

more "dense"

more "sparse"

NTNU

Source code of Scan/MergeTrans on Github



https://github.com/vtsynergy/sptrans

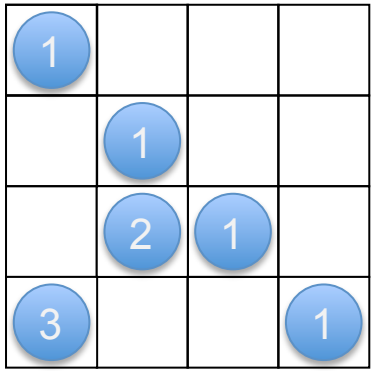
NTNU

Kernel 4. Sparse Triangular Solve

(SpTRSV)

SpTRSV

- Compute a dense solution vector *x* from a sparse linear system *Lx*=*b*, where *L* is a square lower triangular sparse matrix, and *b* is a dense vector.



L
(4x4)
nnzL = 6
known

x

x
(4x1)
dense
unknown

=

b
(4x1)
dense
known

system:

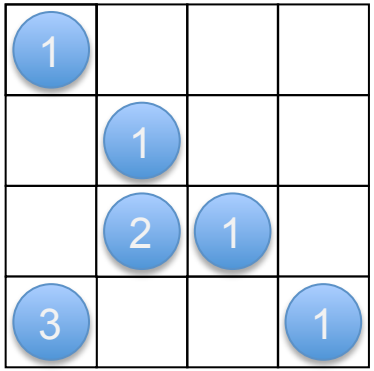
$1*x_0 = a$
$1*x_1 = b$
$2*x_1+1*x_2 = c$
$3*x_0+1*x_3 = d$

solution:

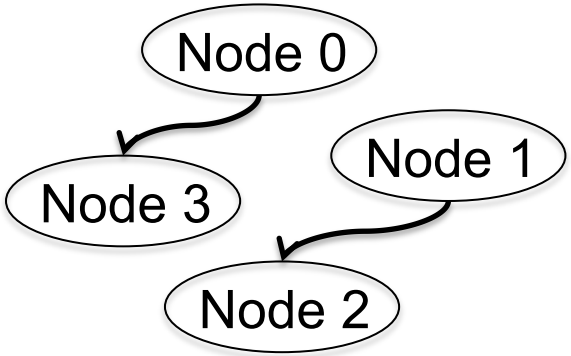
$x_0 = a$
$x_1 = b$
$x_2 = c - 2b$
$x_3 = d - 3a$

Level-set method for SpTRSV

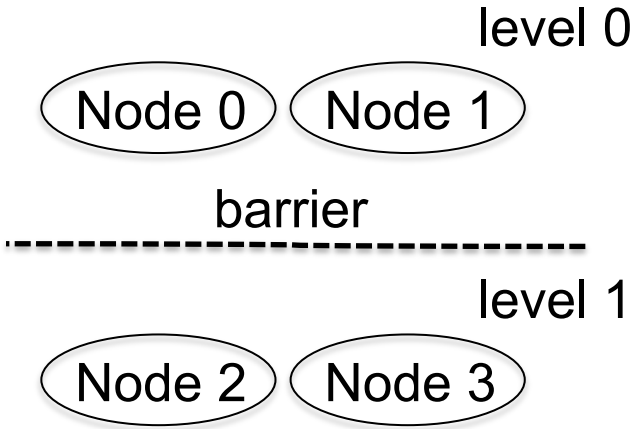
- The level-set method has two phases: (1) group nodes (x_0-x_{n-1}) that can be consumed in parallel, and (2) solve nodes group by group with barriers between.



matrix form

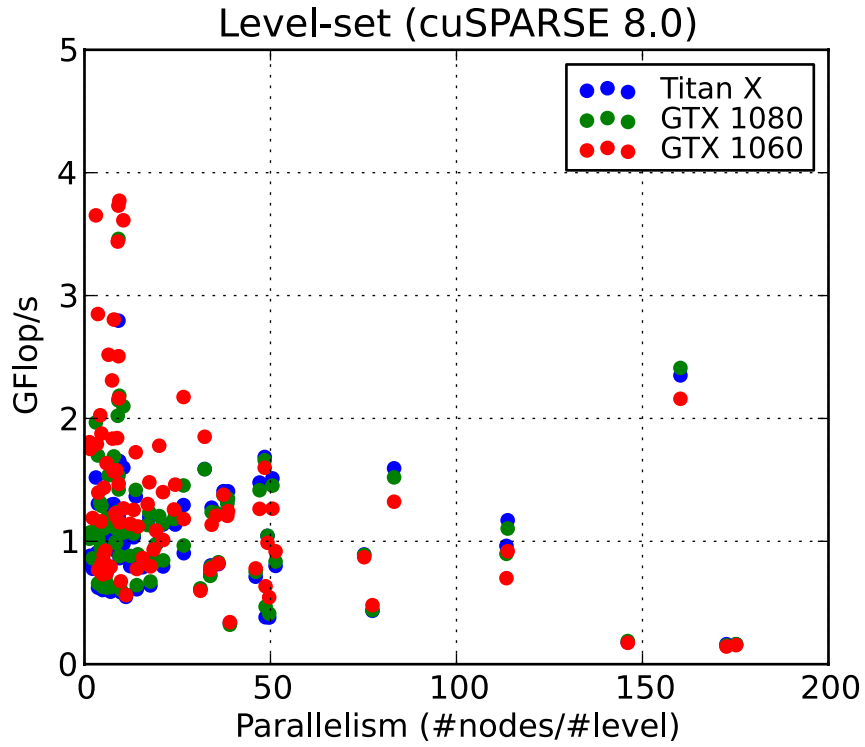


graph form



level-set scheduling

Scalability of level-set method for SpTRSV



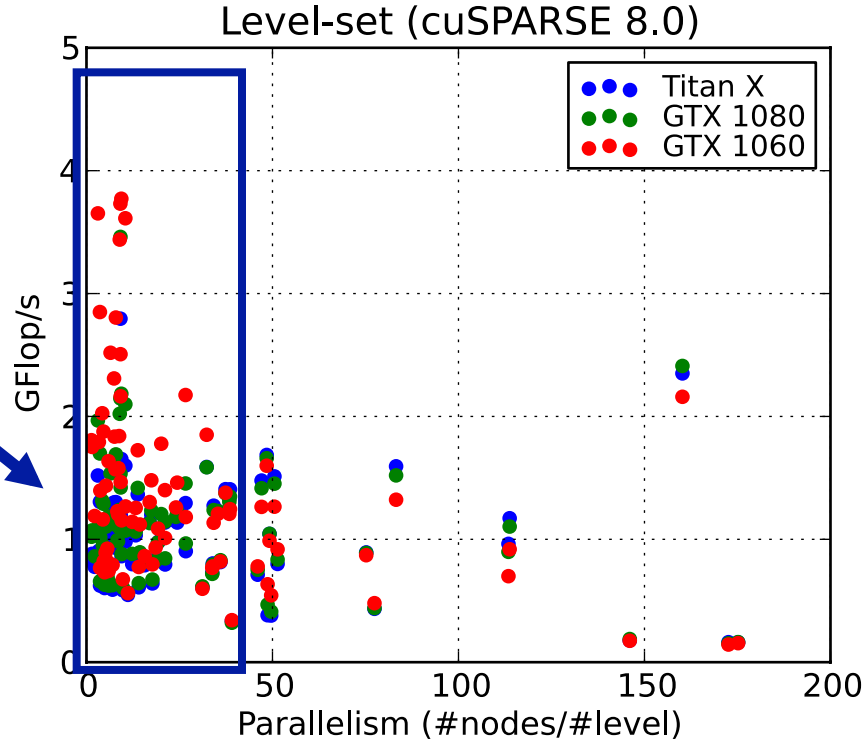
more
“levels”
less
“nodes per level”

more
“nodes per level”
less
“levels”

NTNU

Scalability of level-set method for SpTRSV

SpTRSV perf. inversely scales with hardware perf.



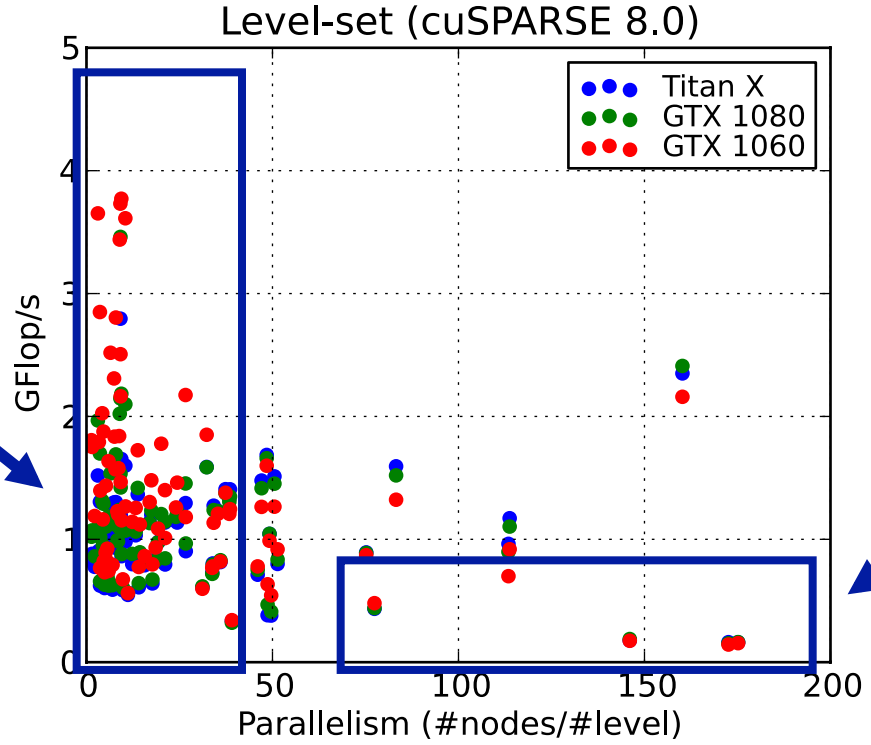
more "levels" less "nodes per level"

more "nodes per level" less "levels"

NTNU

Scalability of level-set method for SpTRSV

SpTRSV perf. inversely scales with hardware perf.



SpTRSV perf. does not scale with hardware perf.

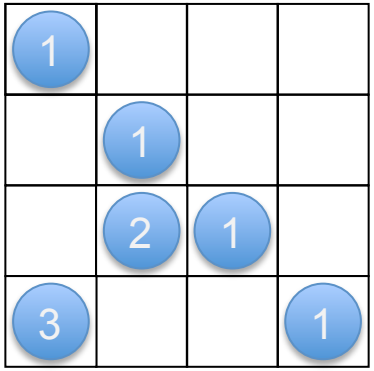
more "levels" less "nodes per level"

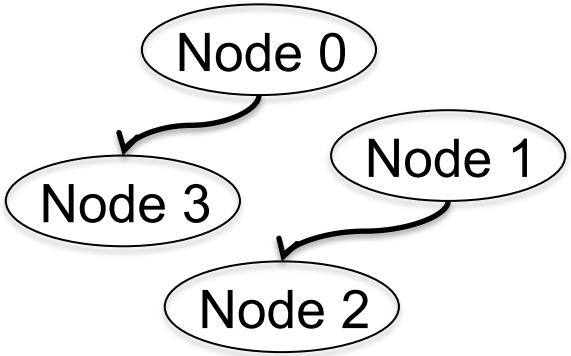
more "nodes per level" less "levels"

NTNU

Sync-free method for SpTRSV (our work)

- All cores (one for each column as a node) are activiated: some of them compute solution, the others busy-wait for spin-locks unlocked to start.





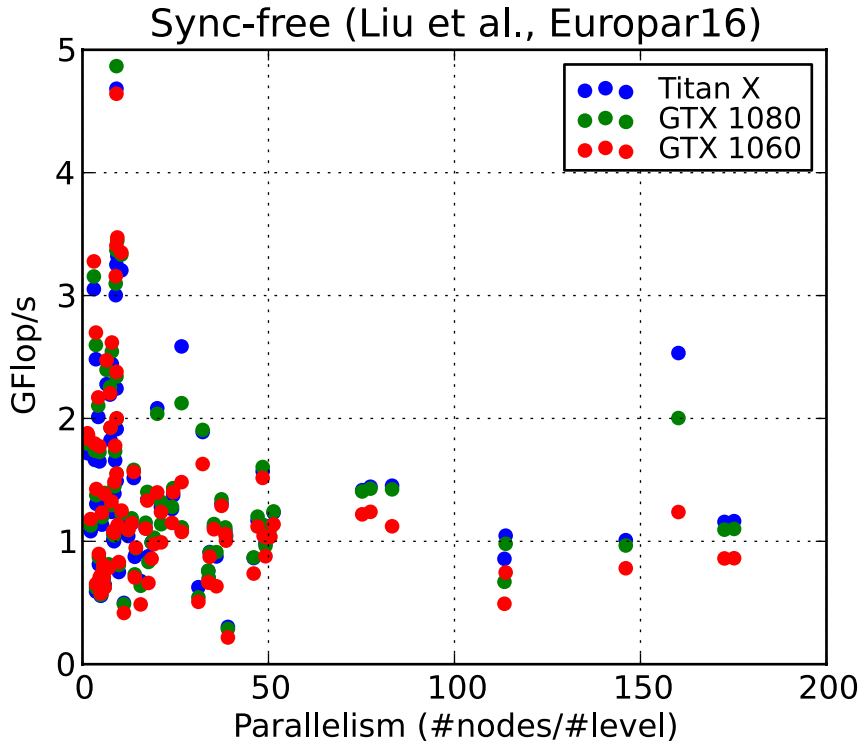
CSR row_ptr:	0	1	2	4	6
CSC column_ptr:	0	2	4	5	6

In-degree:	1	1	2	2
Out-degree:	2	2	1	1

Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, Brian Vinter. **Fast Synchronization-Free Algorithms for Parallel Sparse Triangular Solves with Multiple Right-Hand Sides**. *CCPE. 2017.* (extended from *Europar16*)

Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, Brian Vinter. **A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves**. *Europar16*.

Scalability of sync-free method for SpTRSV



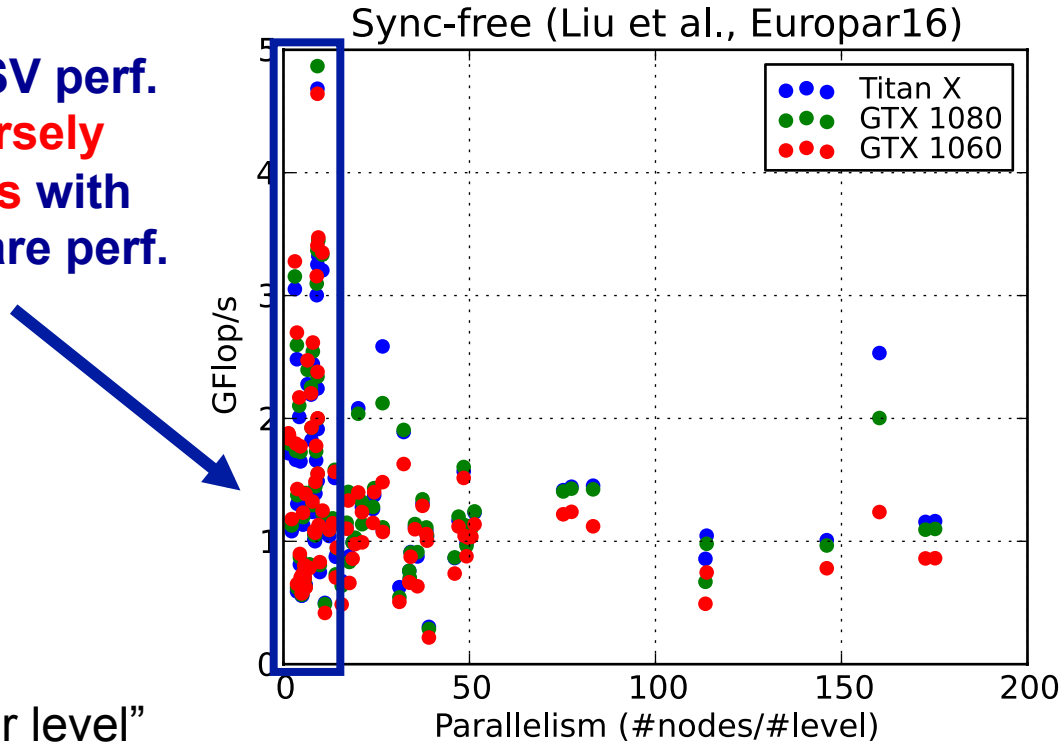
more
“levels”
less
“nodes per level”

more
“nodes per level”
less
“levels”

NTNU

Scalability of sync-free method for SpTRSV

SpTRSV perf. inversely scales with hardware perf.

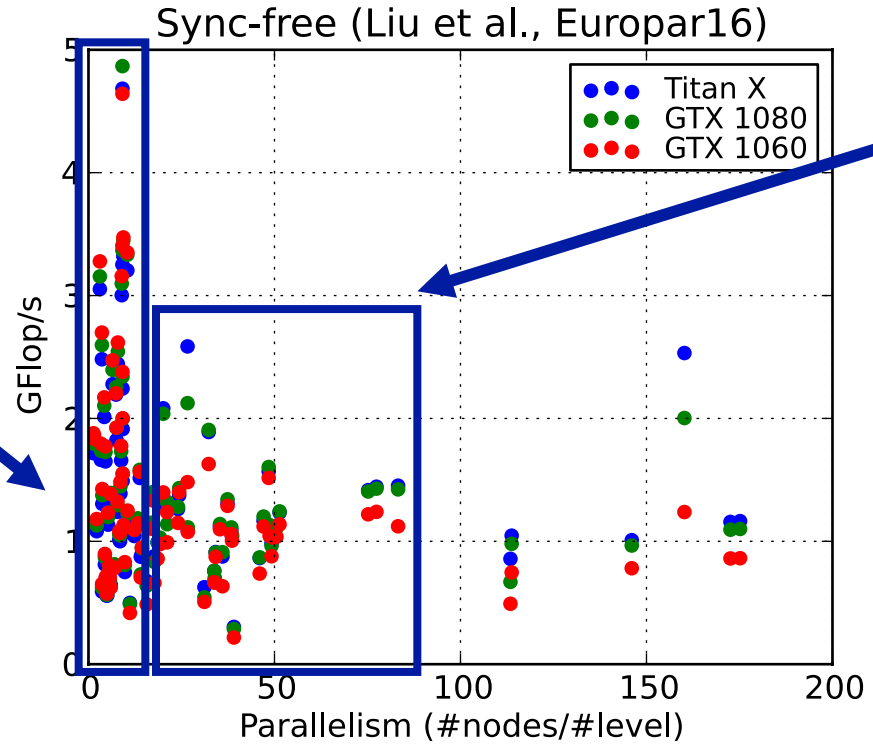


more "levels" less "nodes per level"

more "nodes per level" less "levels"

NTNU

Scalability of sync-free method for SpTRSV



SpTRSV perf. inversely scales with hardware perf.

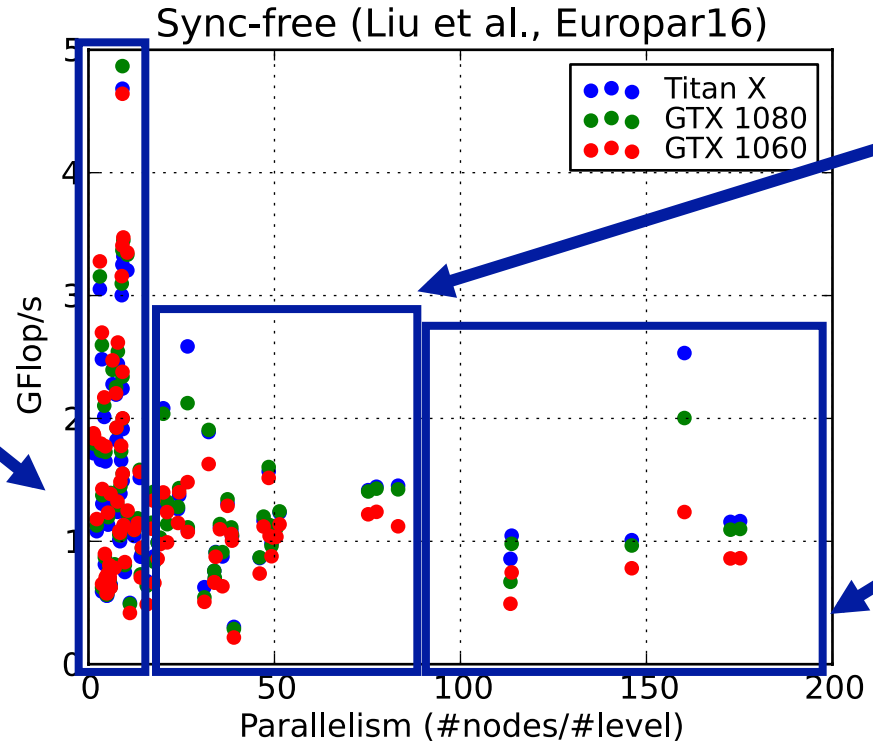
SpTRSV perf. does not scale with hardware perf.

more "levels" less "nodes per level"

more "nodes per level" less "levels"

NTNU

Scalability of sync-free method for SpTRSV



SpTRSV perf. inversely scales with hardware perf.

SpTRSV perf. does not scale with hardware perf.

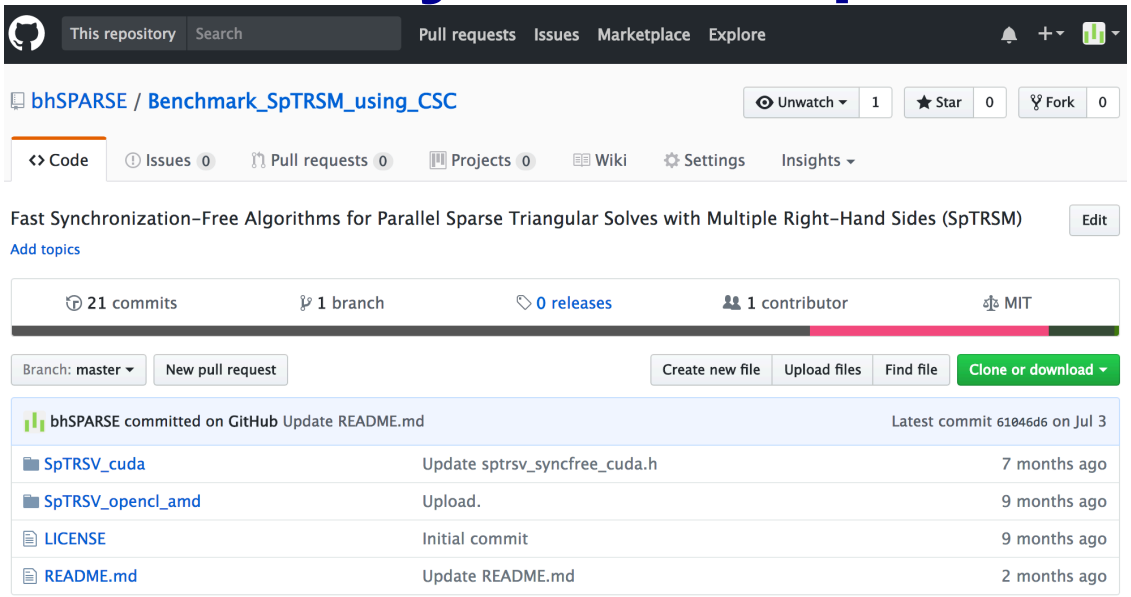
SpTRSV perf. scales with hardware perf.

more "levels" less "nodes per level"

more "nodes per level" less "levels"

NTNU

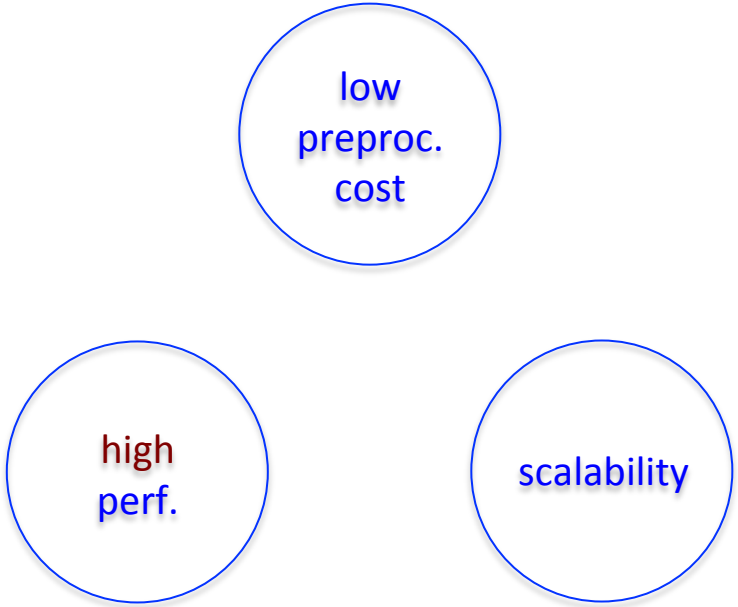
Source code of sync-free SpTRSV on Github



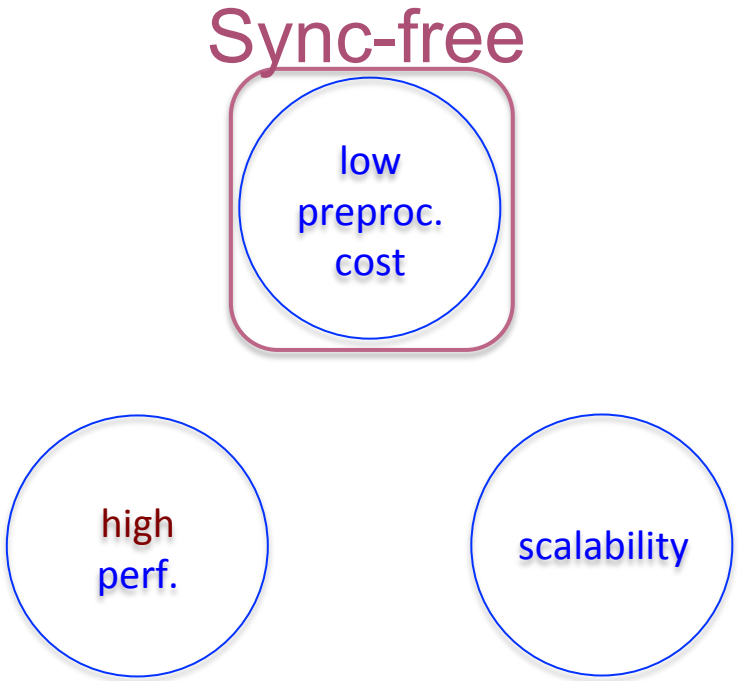
https://github.com/bhSPARSE/Benchmark_SpTRSM_using_CSC

Has been integated into the MAGMA library.
https://bitbucket.org/icl/magma

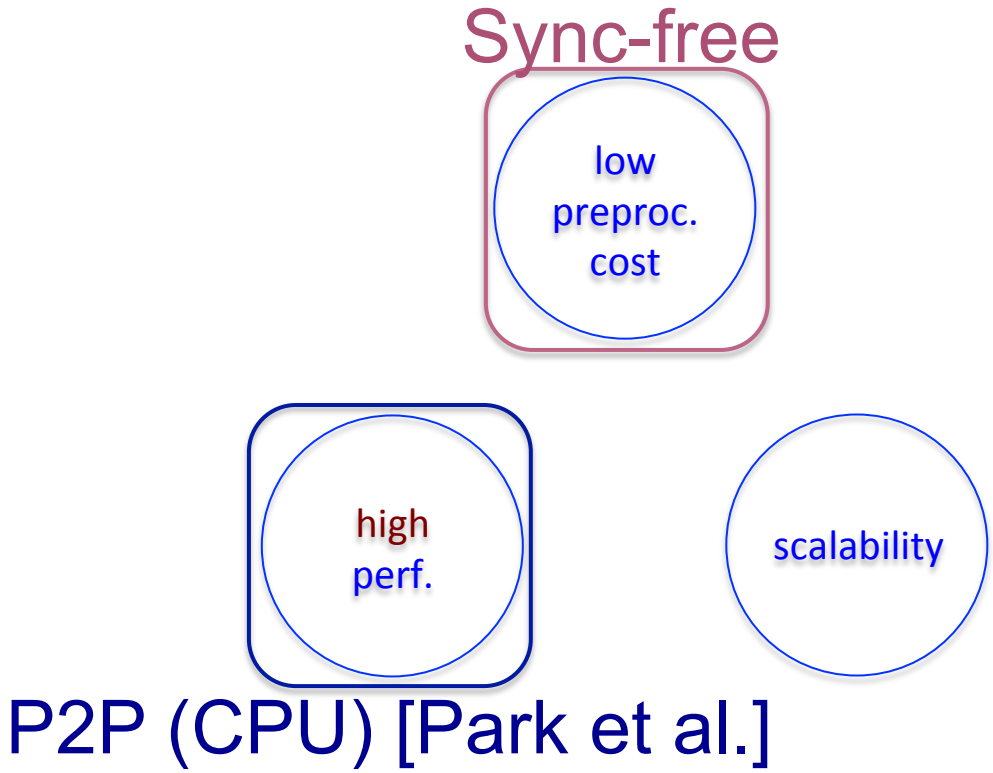
Scalability, high perf. and low preproc. cost (choose one of the three?)



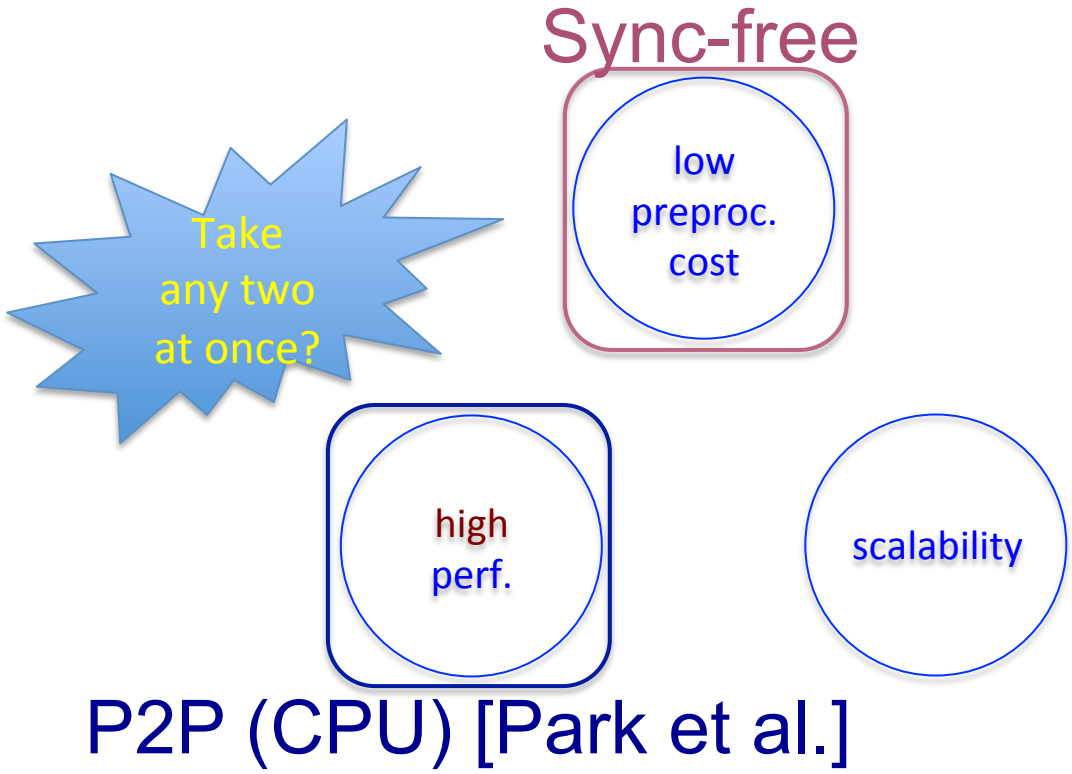
Scalability, high perf. and low preproc. cost (choose one of the three?)



Scalability, high perf. and low preproc. cost (choose one of the three?)



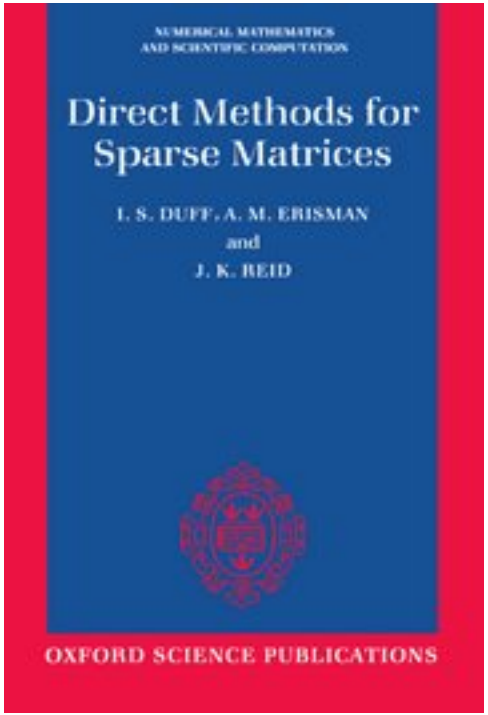
Scalability, high perf. and low preproc. cost (choose one of the three?)



Observations (Summary)

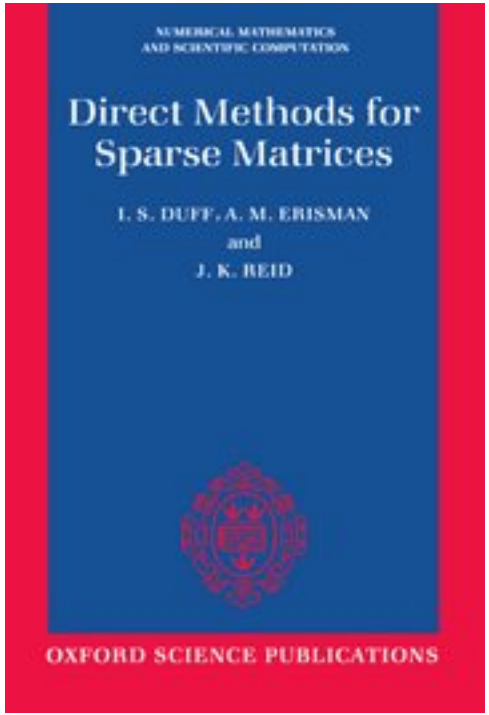
- The emergement of many-core processors magnifies the scalability problem brought by sparsity stuctures, and brings new research opportunities.
- There is no “enough good” SpMV implementation (low preprocesing cost + scalability + high performance); as for other operations, scalability and high performance cannot exist together.
- A lot of work to do for sparse BLAS.

Direct Methods for Sparse Matrices

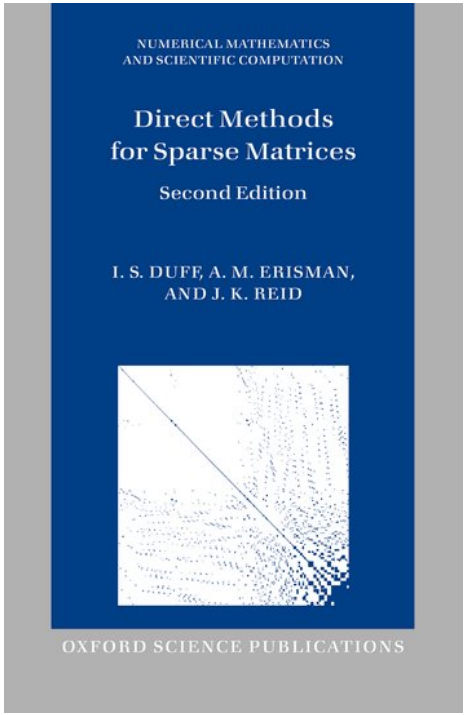


1st Edition, 1986

Direct Methods for Sparse Matrices

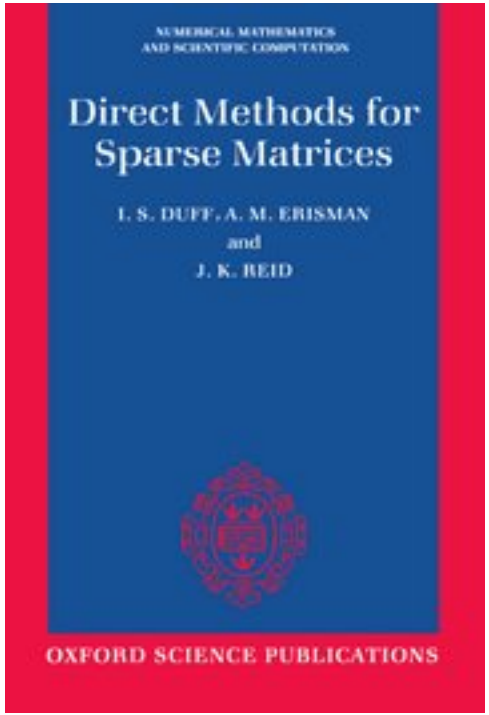


1st Edition, 1986

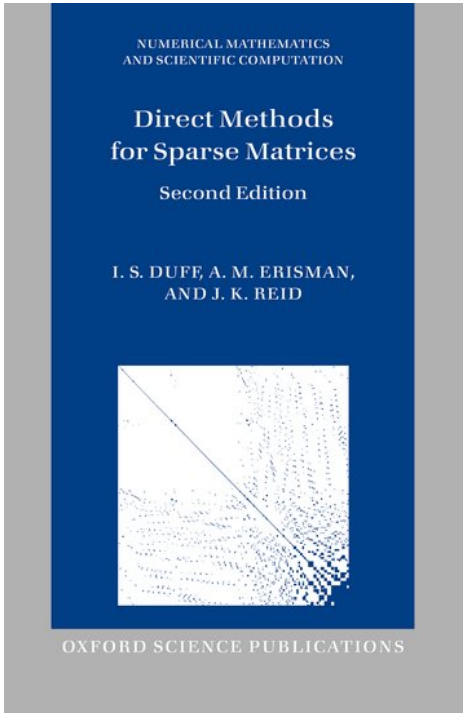


2nd Edition, 2017

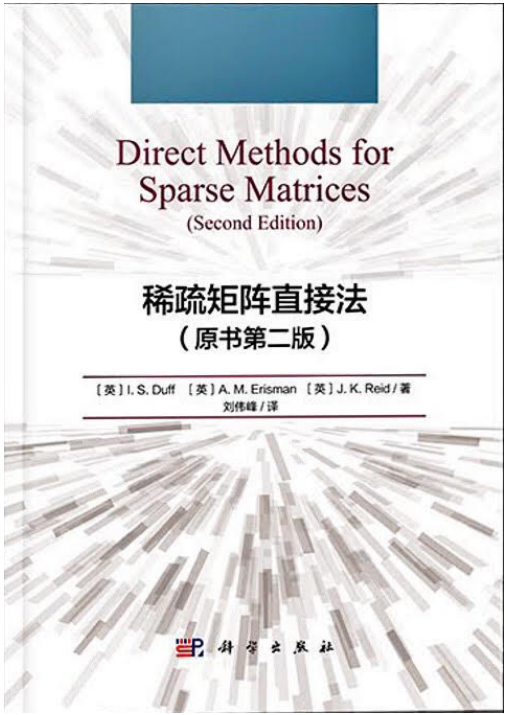
Direct Methods for Sparse Matrices



1st Edition, 1986



2nd Edition, 2017



Chinese Edition, 2018

NTNU

H y B h y l a i n !

0	4	6	10	13	15	16	17	18	19

T k u !

0	4	8	9

A y Q s n s ?

0	2	4	7	11	12	13

References

(1) Weifeng Liu, Brian Vinter. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. 29th ACM International Conference on Supercomputing (ICS '15), 2015, pp. 339-350.

(2) Weifeng Liu, Brian Vinter. A Framework for General Sparse Matrix-Matrix Multiplication on GPUs and Heterogeneous Processors. Journal of Parallel and Distributed Computing (JPDC), 2015, Volume 85, pp. 47-61.

(3) Weifeng Liu, Brian Vinter. An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data. 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS '14), 2014, pp. 370-381.

(4) Hao Wang, Weifeng Liu, Kaixi Hou, Wu-Chun Feng. Parallel Transposition of Sparse Data Structures. 30th ACM International Conference on Supercomputing (ICS '16), 2016, pp. 33:1-33:13.

(5) Weifeng Liu, Ang Li, Jonathan D. Hogg, Iain S. Duff, Brian Vinter. Fast Synchronization-Free Algorithms for Parallel Sparse Triangular Solves with Multiple Right-Hand Sides. Concurrency and Computation: Practice and Experience (CCPE), 2017.

(6) Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, Brian Vinter. A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves. 22nd International European Conference on Parallel and Distributed Computing (Euro-Par '16), 2016, pp. 617-630.