

Scalability Analysis of Sparse Matrix Computations on Many-core Processors

Weifeng Liu

Norwegian University of Science and Technology, Norway



NTNU



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.

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

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.

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

A
(4x4)
dense

		1	
2	3		
4		5	6

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.

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

A
(4x4)
dense

		1	
2	3		
4		5	6

A
(4x4)
sparse, 6 nonzeros

row pointer =

0	1	3	3	6
---	---	---	---	---

column index =

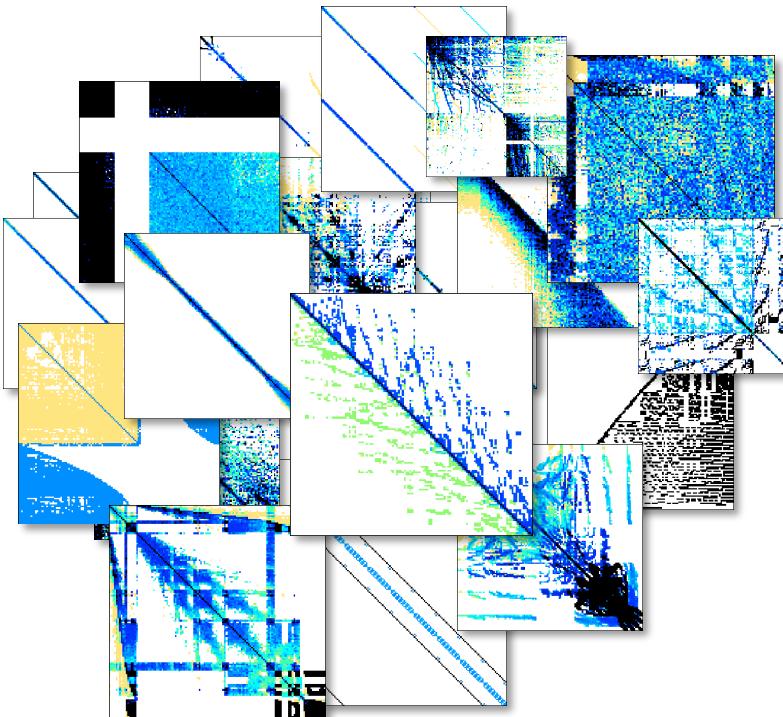
2	0	1	0	2	3
---	---	---	---	---	---

value =

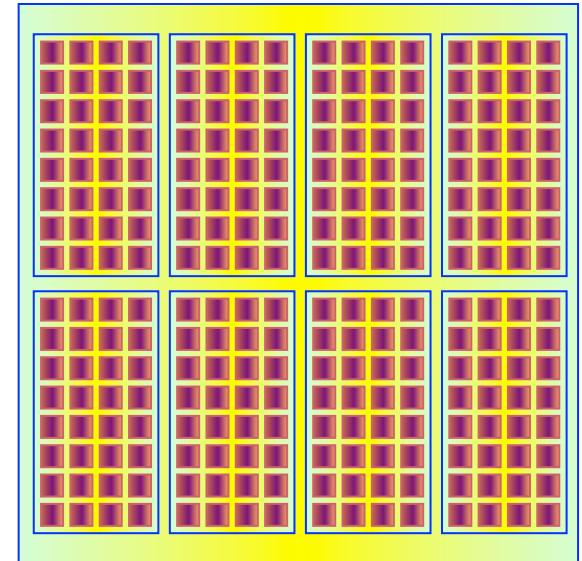
1	2	3	4	5	6
---	---	---	---	---	---

A (in CSR)
(4x4)
6 nonzeros

Sparse matrices on many-core processors

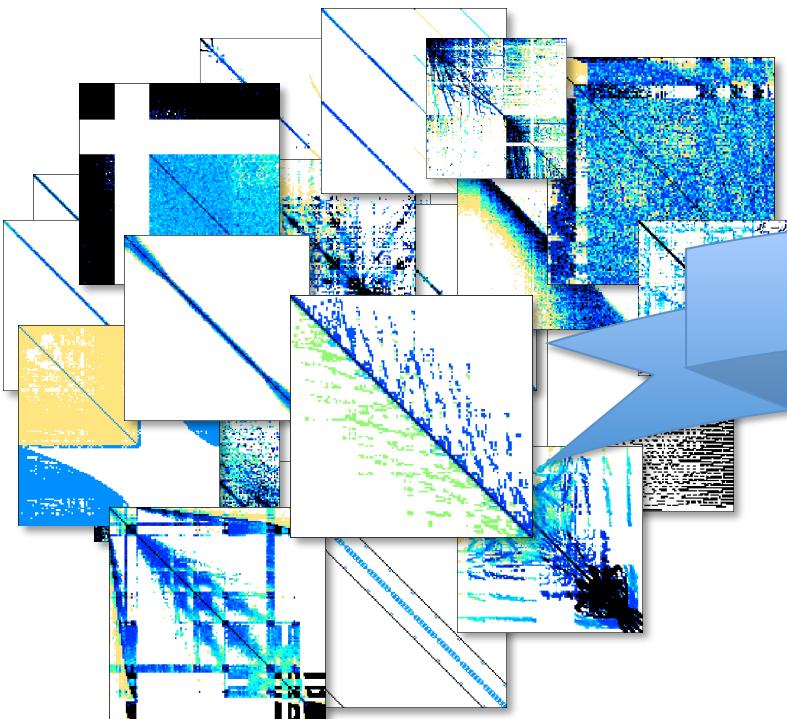


Various sparsity structures



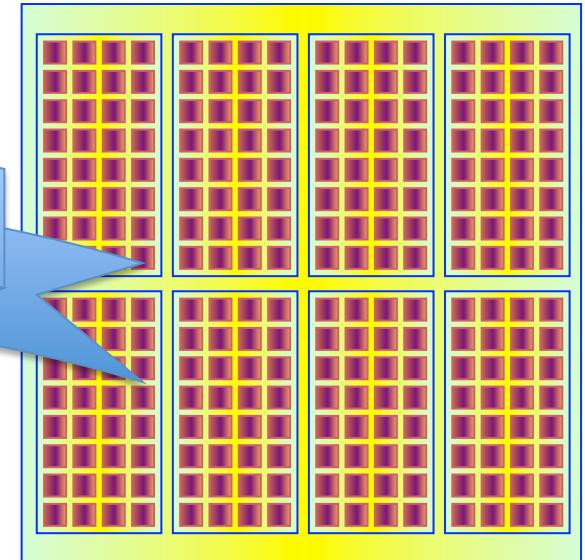
Many cores with a large amount of “small” SIMD units

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)

$$\begin{array}{|c|c|c|}\hline & & 1 \\ \hline 2 & 3 & \\ \hline & & \\ \hline 4 & 5 & 6 \\ \hline\end{array} \times \begin{array}{|c|}\hline a \\ \hline b \\ \hline c \\ \hline d \\ \hline\end{array} = \begin{array}{|c|}\hline 1c \\ \hline 2a+3b \\ \hline 0 \\ \hline 4a+5c+6d \\ \hline\end{array}$$

Four kernels used to study scalability

- Sparse matrix-vector Multiplication (SpMV)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline 2 & 3 & \\ \hline & & \\ \hline 4 & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|} \hline a \\ \hline b \\ \hline c \\ \hline d \\ \hline \end{array} = \begin{array}{|c|} \hline 1c \\ \hline 2a+3b \\ \hline 0 \\ \hline 4a+5c+6d \\ \hline \end{array}$$

- Sparse matrix-matrix Multiplication (SpGEMM)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline 2 & 3 & \\ \hline & & \\ \hline 4 & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline & & a \\ \hline b & c & \\ \hline d & e & \\ \hline & f & \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline & 1d & & 1e \\ \hline 3b & & 3c & 2a \\ \hline & & & \\ \hline & 5d & 6f & 4a+5e \\ \hline \end{array}$$

Four kernels used to study scalability

- Sparse matrix-vector Multiplication (SpMV)
- Sparse transposition (SpTRANS)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline 2 & 3 & \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|} \hline a \\ \hline b \\ \hline c \\ \hline d \\ \hline \end{array} = \begin{array}{|c|} \hline 1c \\ \hline 2a+3b \\ \hline 0 \\ \hline 4a+5c+6d \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline 2 & 3 & \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|} \hline 2 & & 4 \\ \hline 3 & & \\ \hline 1 & & 5 \\ \hline 4 & & 6 \\ \hline \end{array}$$

- Sparse matrix-matrix Multiplication (SpGEMM)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline 2 & 3 & \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline & & a \\ \hline b & c & \\ \hline d & e & \\ \hline f & & \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline & 1d & & 1e \\ \hline 3b & & 3c & 2a \\ \hline & & & \\ \hline & 5d & 6f & 4a+5e \\ \hline \end{array}$$

Four kernels used to study scalability

- Sparse matrix-vector Multiplication (SpMV)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline & 2 & 3 \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|} \hline a \\ \hline b \\ \hline c \\ \hline d \\ \hline \end{array} = \begin{array}{|c|} \hline 1c \\ \hline 2a+3b \\ \hline 0 \\ \hline 4a+5c+6d \\ \hline \end{array}$$

- Sparse transposition (SpTRANS)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline & 2 & 3 \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|} \hline 2 & & 4 \\ \hline 3 & & 5 \\ \hline 1 & & 6 \\ \hline \end{array}$$

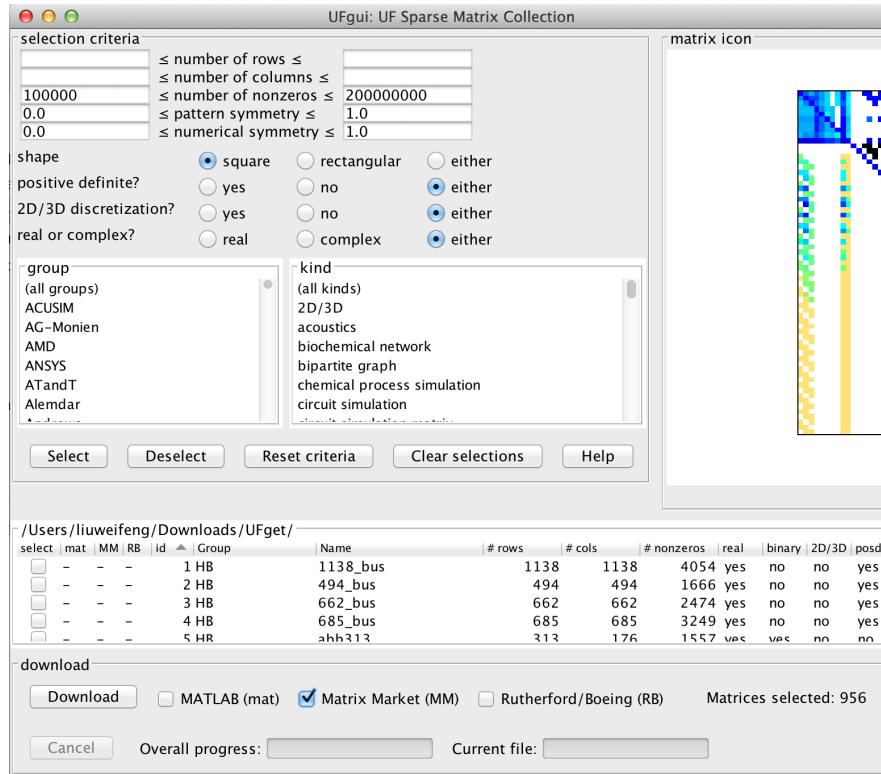
- Sparse matrix-matrix Multiplication (SpGEMM)

$$\begin{array}{|c|c|c|} \hline & & 1 \\ \hline & 2 & 3 \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline & & a \\ \hline & b & c \\ \hline & d & e \\ \hline & f \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & 1d & & 1e \\ \hline & 3b & 3c & 2a \\ \hline & 5d & 6f & 4a+5e \\ \hline \end{array}$$

- Sparse triangular solve (SpTRSV)

$$\begin{array}{|c|c|c|} \hline 1 & & \\ \hline & 1 & \\ \hline & 2 & 1 \\ \hline 3 & & 1 \\ \hline \end{array} \times \begin{array}{|c|} \hline x_0 \\ \hline x_1 \\ \hline x_2 \\ \hline x_3 \\ \hline \end{array} = \begin{array}{|c|} \hline a \\ \hline b \\ \hline c \\ \hline d \\ \hline \end{array}$$

956 matrices used to study scalability



956 matrices used to study scalability

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

UFgui: UF Sparse Matrix Collection

selection criteria

- 100000
- 0.0
- 0.0
- ≤ number of rows ≤
- ≤ number of columns ≤
- ≤ number of nonzeros ≤ 200000000
- ≤ pattern symmetry ≤ 1.0
- ≤ numerical symmetry ≤ 1.0

shape square rectangular either

positive definite? yes no either either

2D/3D discretization? yes no either either

real or complex? real complex either

group
(all groups)
ACUSIM
AG-Monien
AMD
ANSYS
ATandT
Alemdar

kind
(all kinds)
2D/3D
acoustics
biochemical network
bipartite graph
chemical process simulation
circuit simulation

Select Deselect Reset criteria Clear selections Help

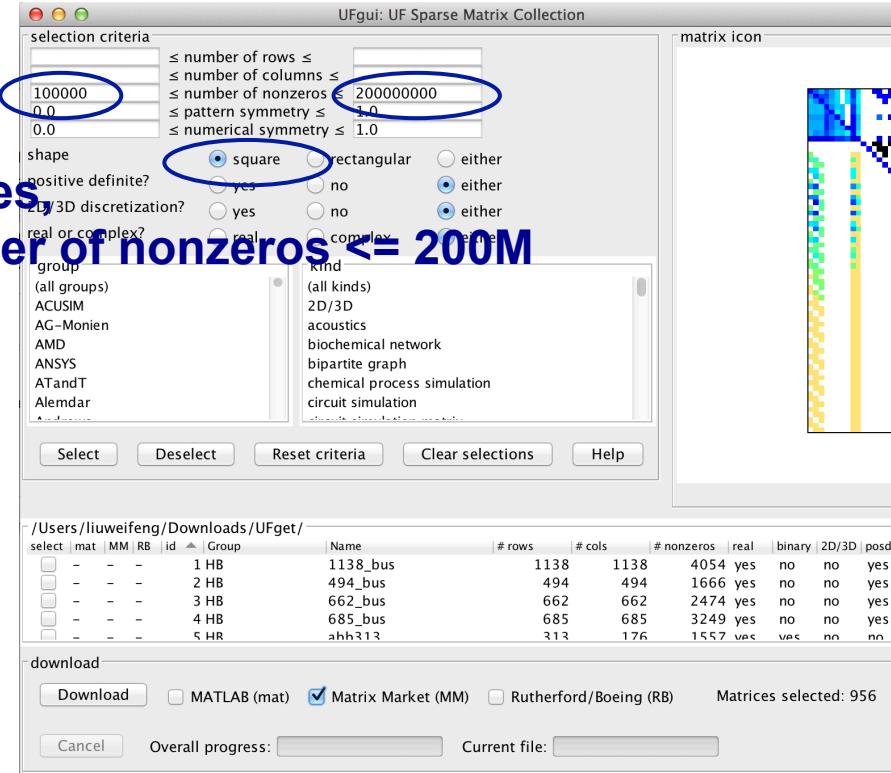
/Users/liuwefeng/Downloads/UFget/

select	mat	MM	RB	id	Group	Name	# rows	# cols	# nonzeros	real	binary	2D/3D	posdef
<input type="checkbox"/>	-	-	-	1	HB	1138_bus	1138	1138	4054	yes	no	no	yes
<input type="checkbox"/>	-	-	-	2	HB	494_bus	494	494	1666	yes	no	no	yes
<input type="checkbox"/>	-	-	-	3	HB	662_bus	662	662	2474	yes	no	no	yes
<input type="checkbox"/>	-	-	-	4	HB	685_bus	685	685	3249	yes	no	no	yes
<input type="checkbox"/>	-	-	-	5	HR	ahb313	313	176	1557	yes	yes	no	no

download

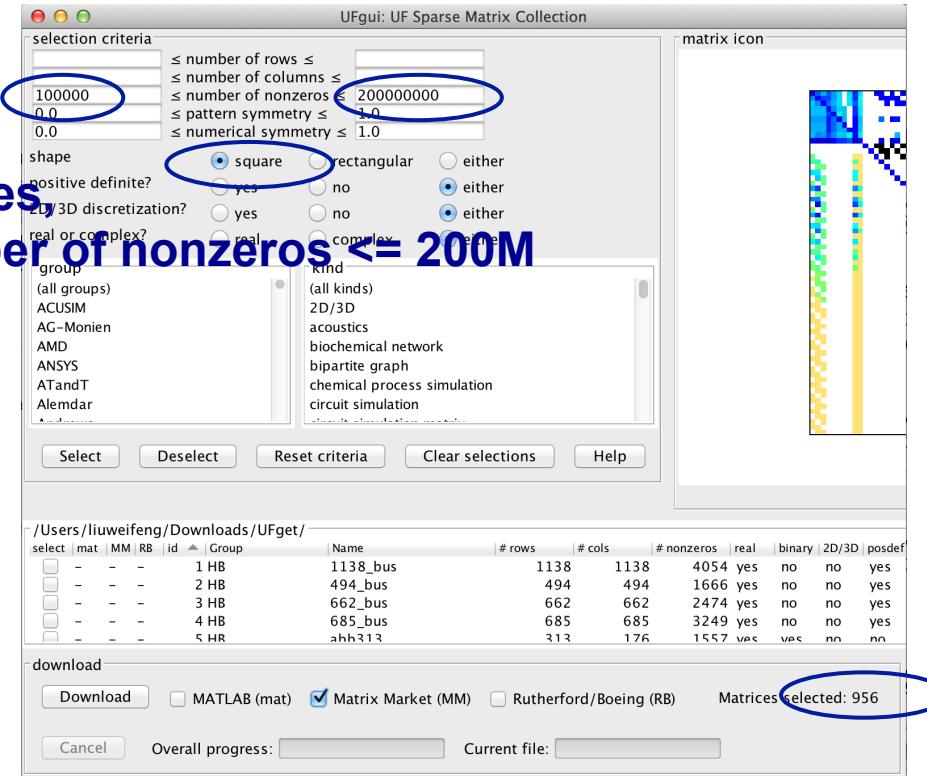
Download MATLAB (mat) Matrix Market (MM) Rutherford/Boeing (RB) Matrices selected: 956

Cancel Overall progress: Current file:



956 matrices used to study scalability

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



956 matrices
from in total
2757 matrices

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 .

		1	
2	3		
4		5	6

A
(4x4)
sparse, 6 nonzeros

a
b
c
d

x
(4x1)
dense

1c
2a+3b
0
4a+5c+6d

y
(4x1)
dense

Parallel SpMV (using the CSR format)

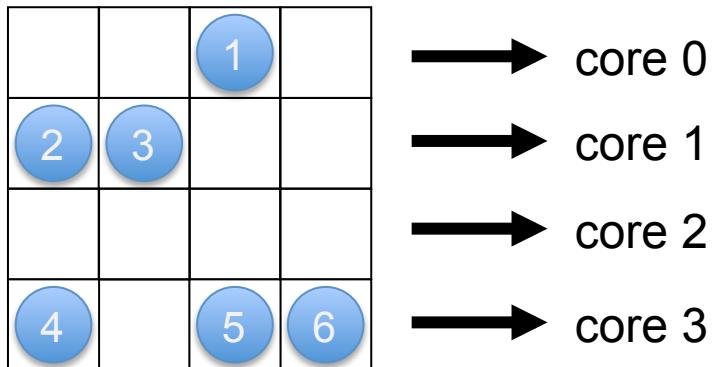
- Assign a row block to one core of a many-core processor.

		1	
2	3		
4		5	6

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.

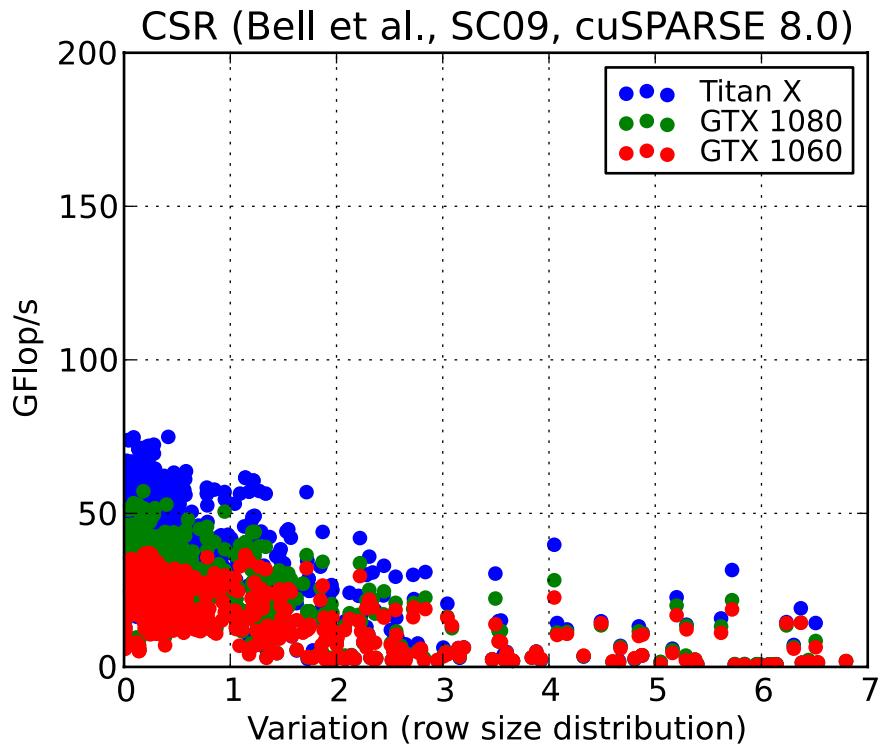
		1	
2	3		
4		5	6

→ core 0
→ core 1
→ core 2
→ core 3

Probably imbalanced computation, degraded performance on many-core processors

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

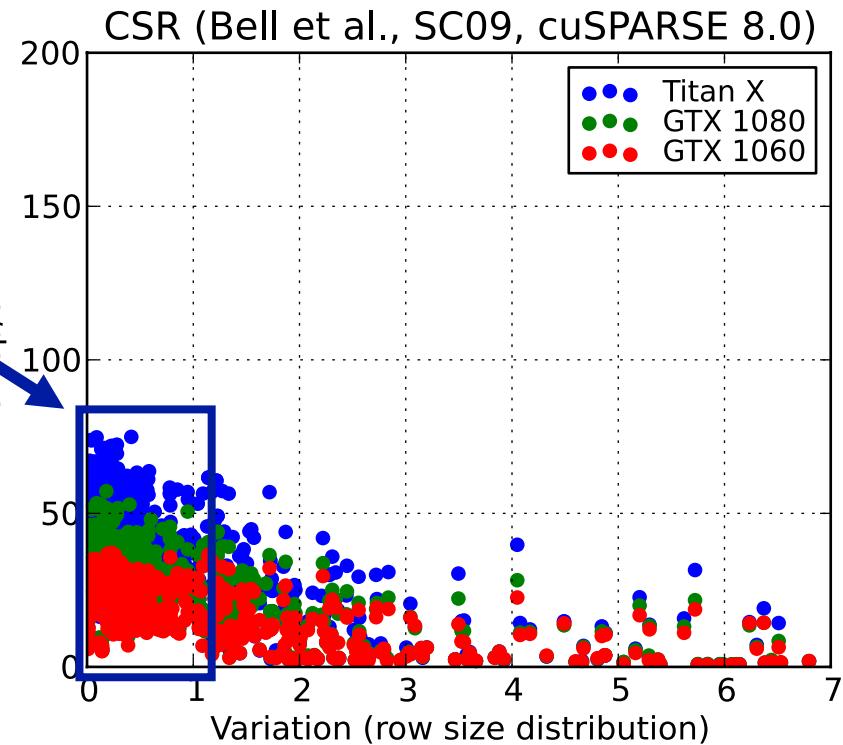
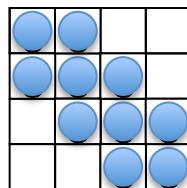
Scalability of SpMV using CSR



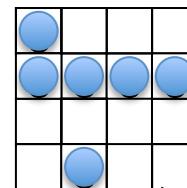
Scalability of SpMV using CSR

SpMV perf.
scales with
hardware perf.

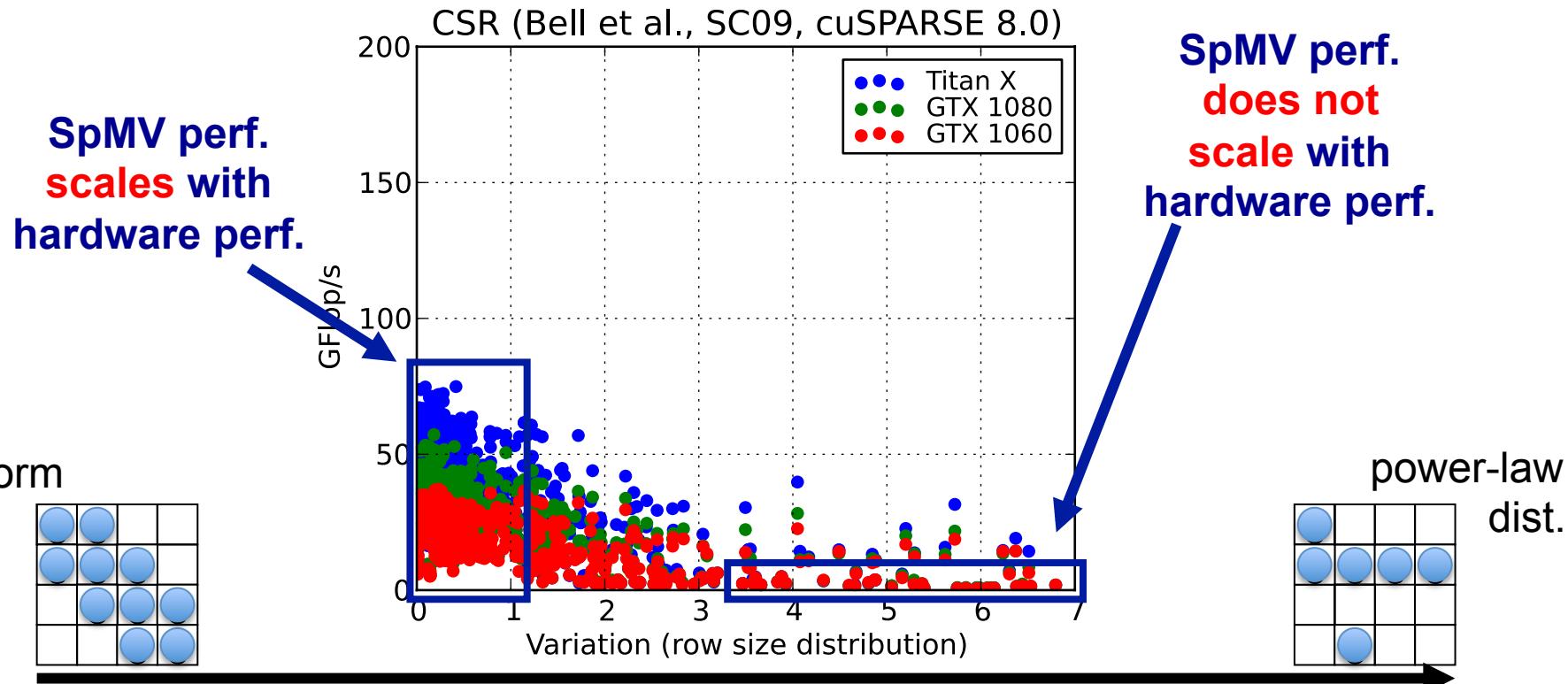
uniform
dist.



power-law
dist.



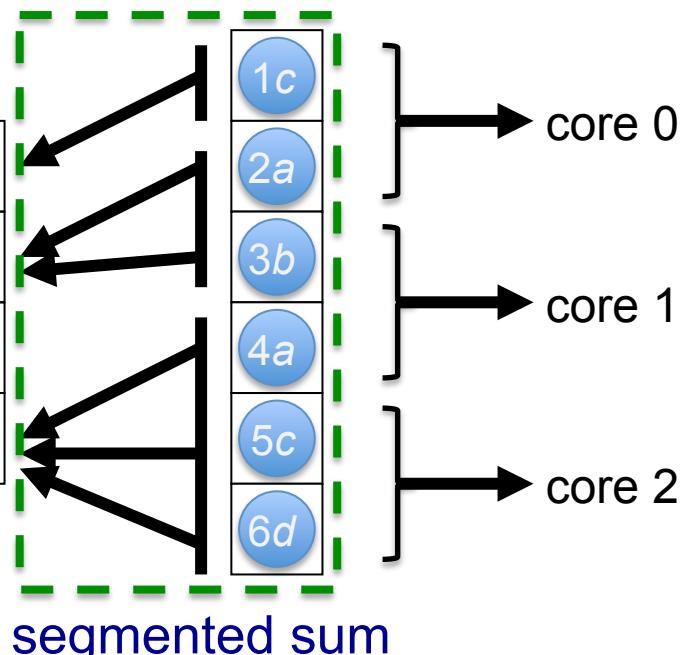
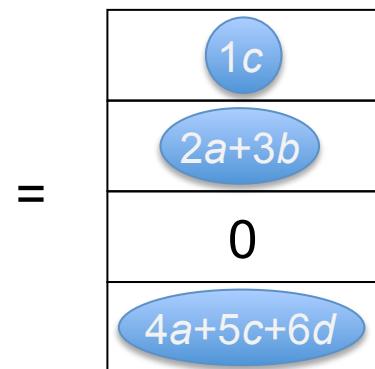
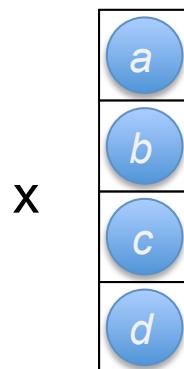
Scalability of SpMV using CSR



Parallel SpMV (using the COO format)

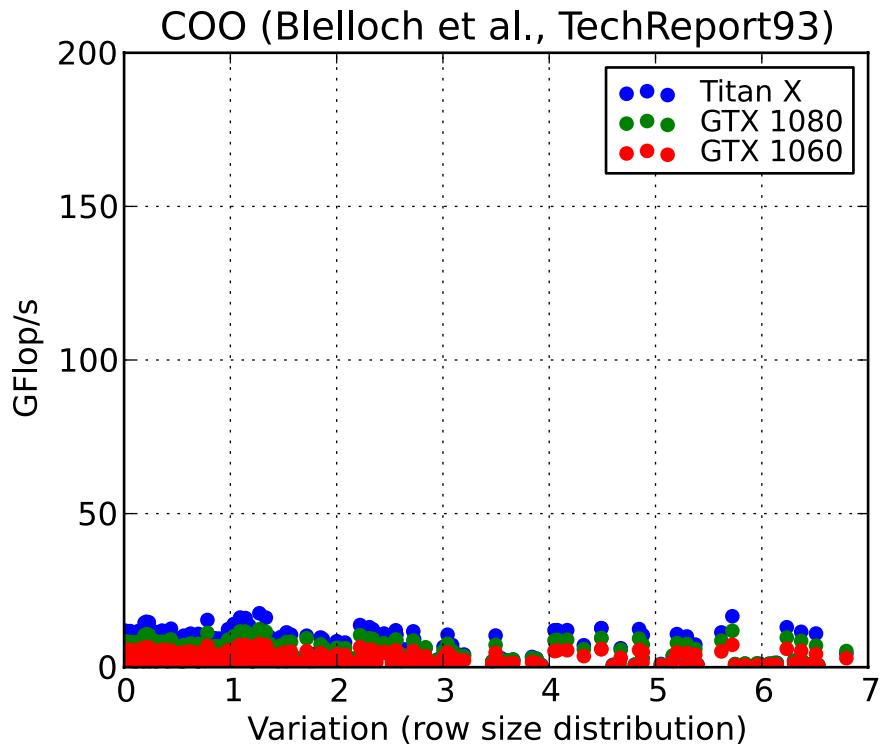
- Evenly assign nonzeros to cores, compute partial products, sum the products for the same row by “segmented sum”.

		1	
2	3		
4		5	6

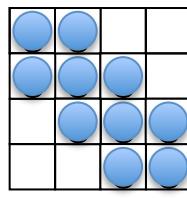


Guy Blelloch, Michael Heroux, Marco Zagha. **Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors.** TechReport, Carnegie Mellon University, 1993.

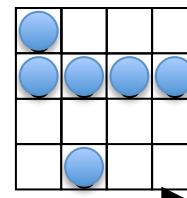
Scalability of SpMV using COO



uniform
dist.



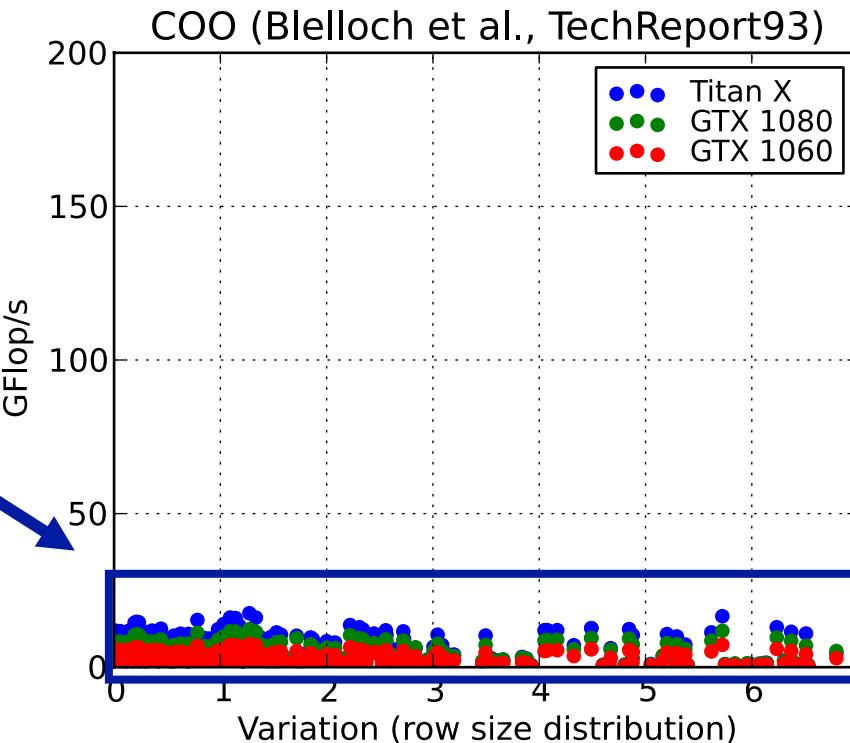
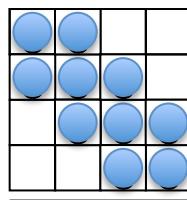
power-law
dist.



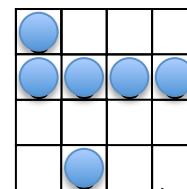
Scalability of SpMV using COO

SpMV perf.
always
scales with
hardware perf.

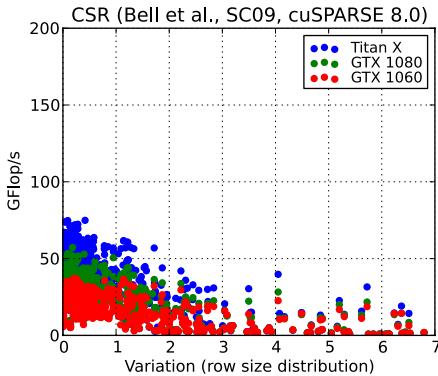
uniform
dist.



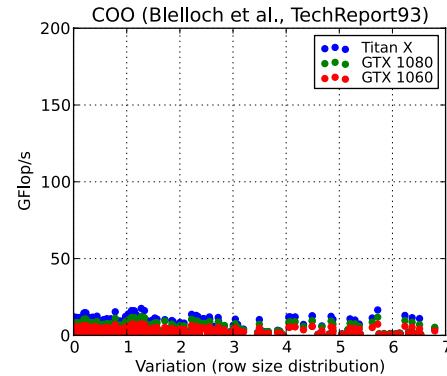
power-law
dist.



Contradiction btw. scalability and high perf.

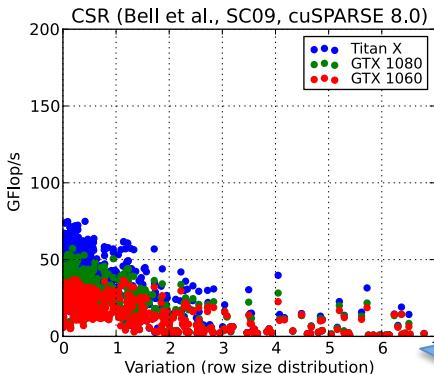


For regular matrices
(uniform dist.),
high performance
method cannot scale.



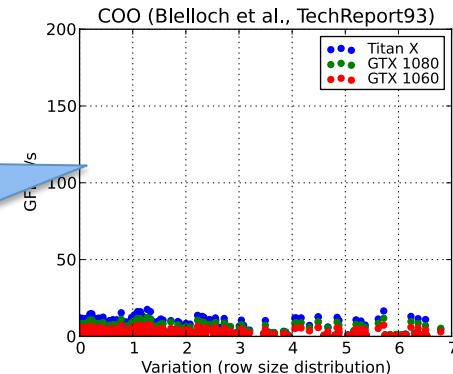
For both regular and
irregular matrices
(power-law dist.),
scalable method gives
low performance.

Contradiction btw. scalability and high perf.



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

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



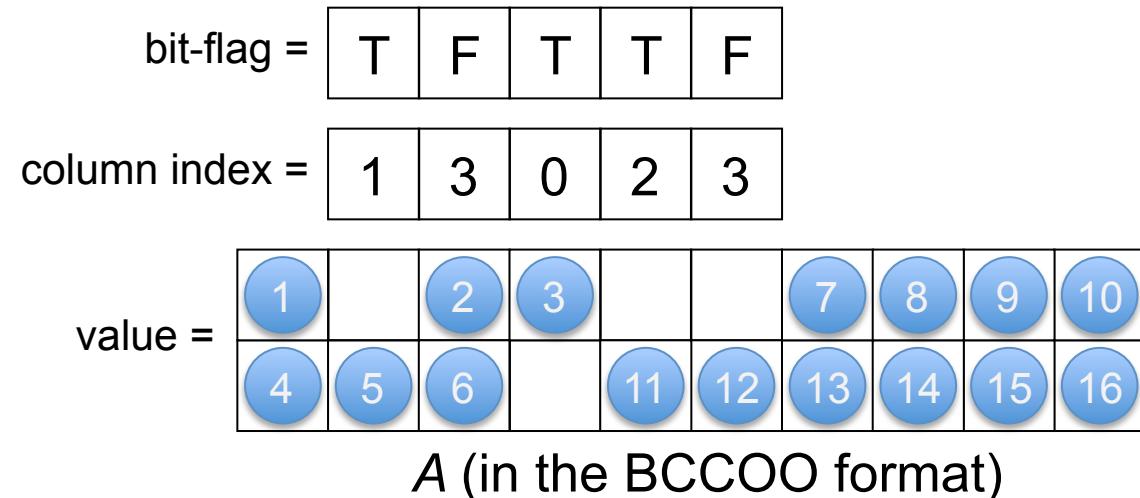
For both regular and
irregular matrices
(power-law dist.),
scalable method gives
low performance.

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.

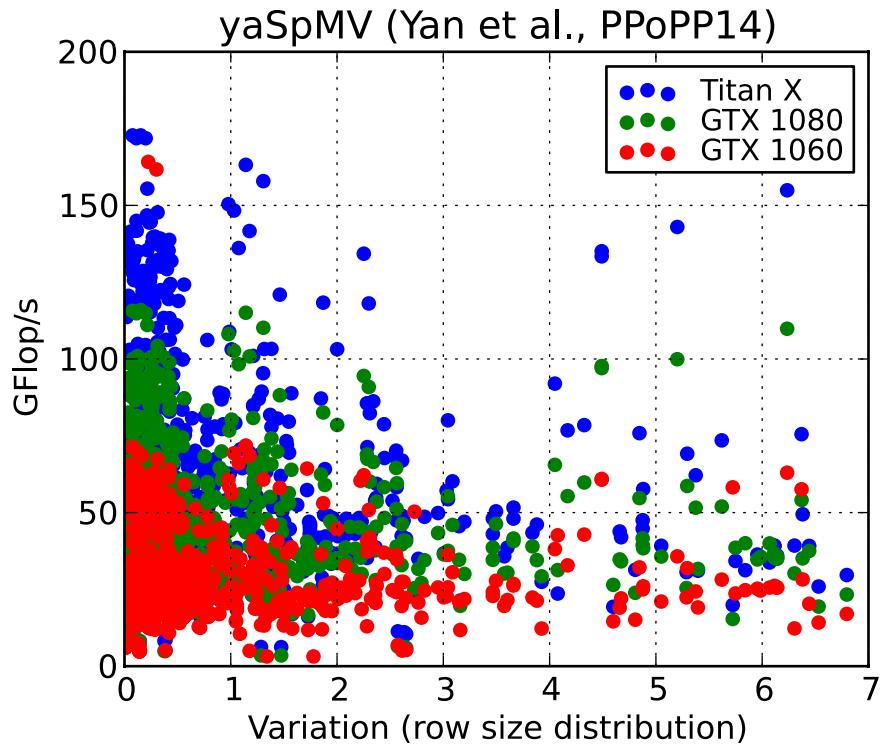
		1			2	3	
		4	5		6		
				7	8	9	10
11	12			13	14	15	16

A
(4x8)

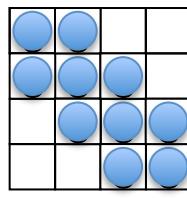


Shengen Yan, Chao Li, Yunquan Zhang, Huiyang Zhou. **yaSpMV: Yet Another SpMV Framework on GPUs.** *PPoPP14*.

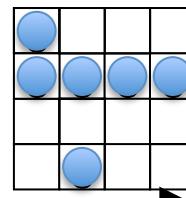
Scalability of yaSpMV



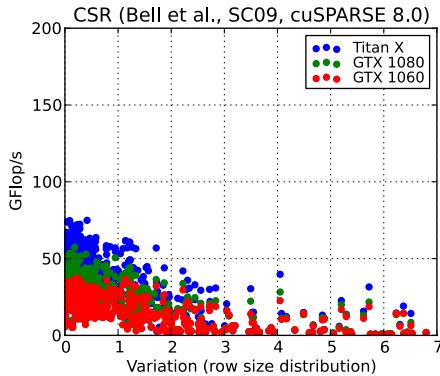
uniform
dist.



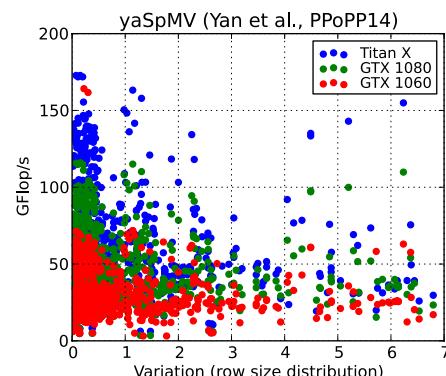
power-law
dist.



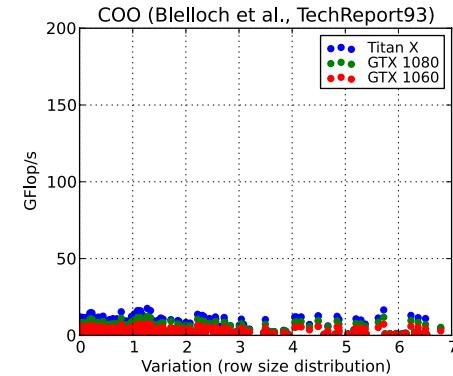
Contradiction btw. scalability and high perf.



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

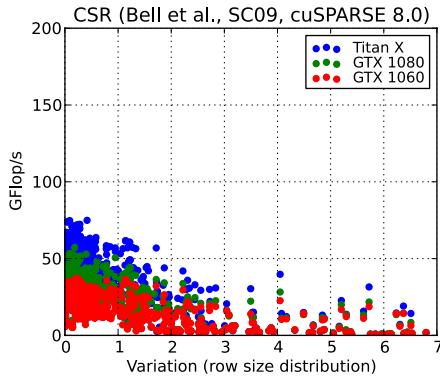


For all matrices,
we now have both
scalability and high
performance.

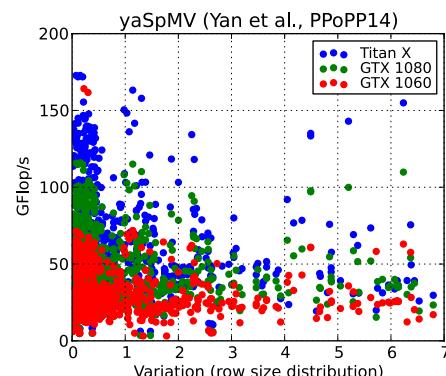


For both regular and
irregular matrices
(power-law dist.),
scalable method gives
low performance.

Contradiction btw. scalability and high perf.

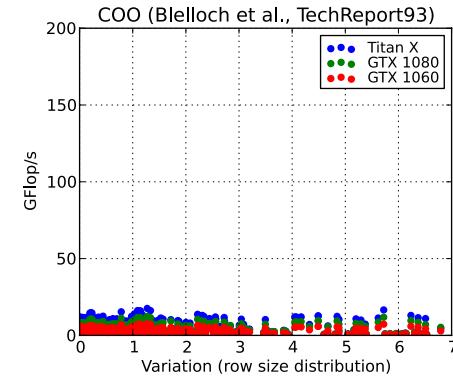


For regular matrices
(uniform dist.),
high performance
method cannot scale.



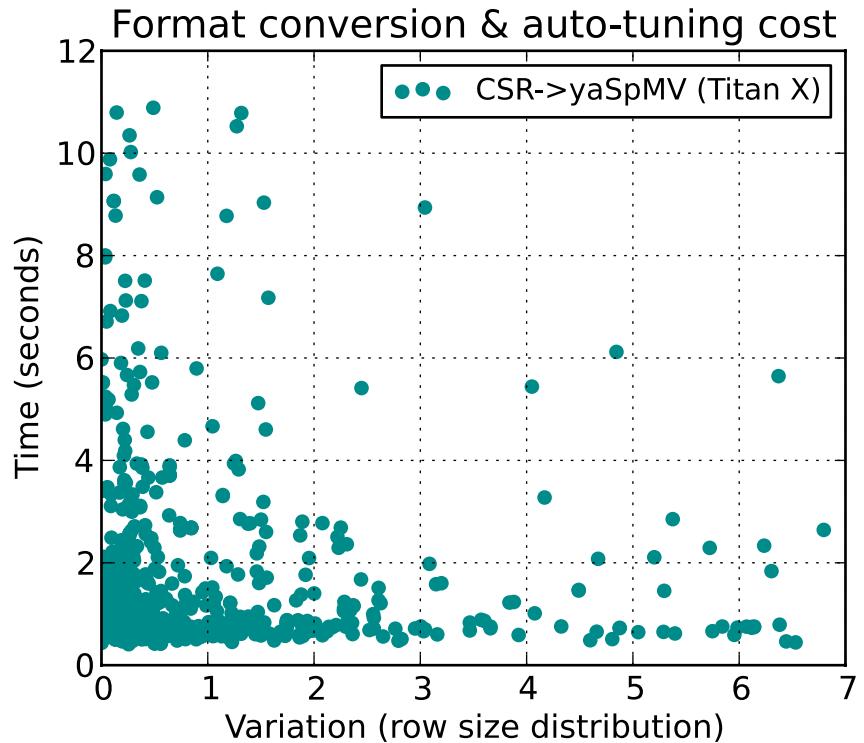
For all matrices,
we now have both
scalability and high
performance.

But...

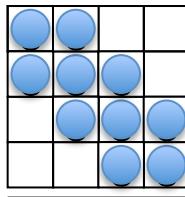


For both regular and
irregular matrices
(power-law dist.),
scalable method gives
low performance.

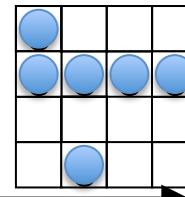
Data conv. and autotuning cost of yaSpMV



uniform
dist.



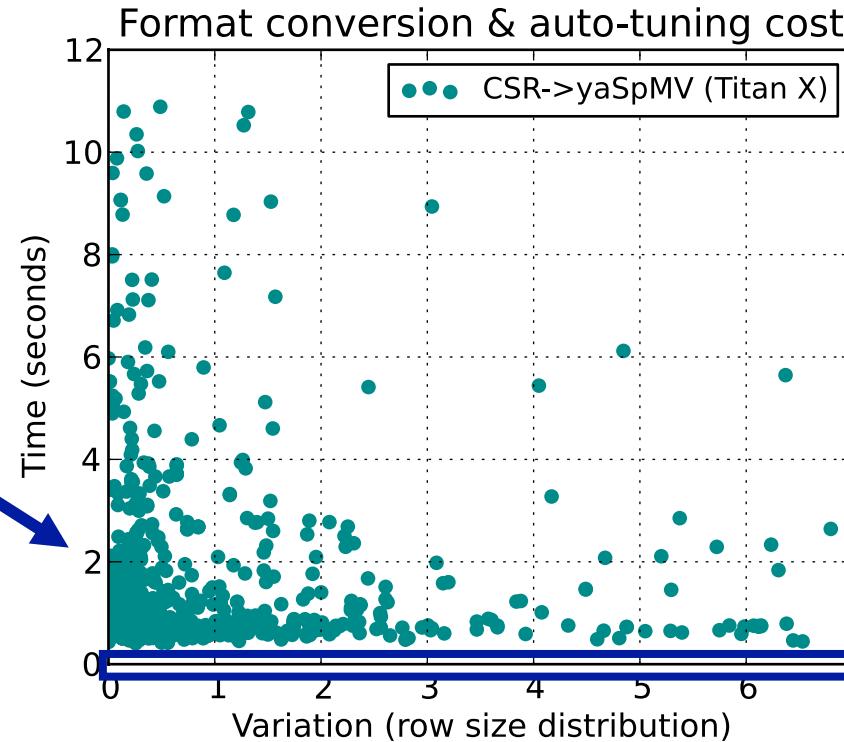
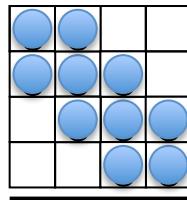
power-law
dist.



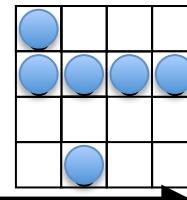
Data conv. and autotuning cost of yaSpMV

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

uniform
dist.



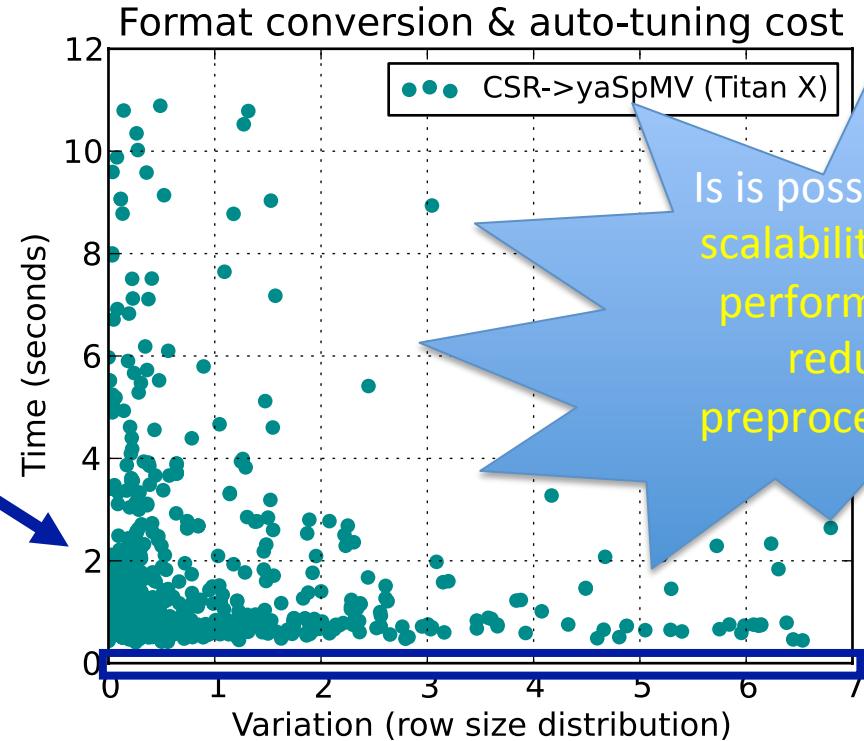
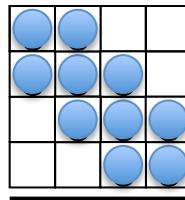
power-law
dist.



Data conv. and autotuning cost of yaSpMV

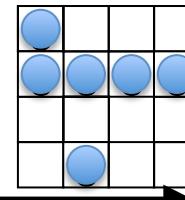
A single
SpMV needs
about
 10^{-4} seconds.

uniform
dist.



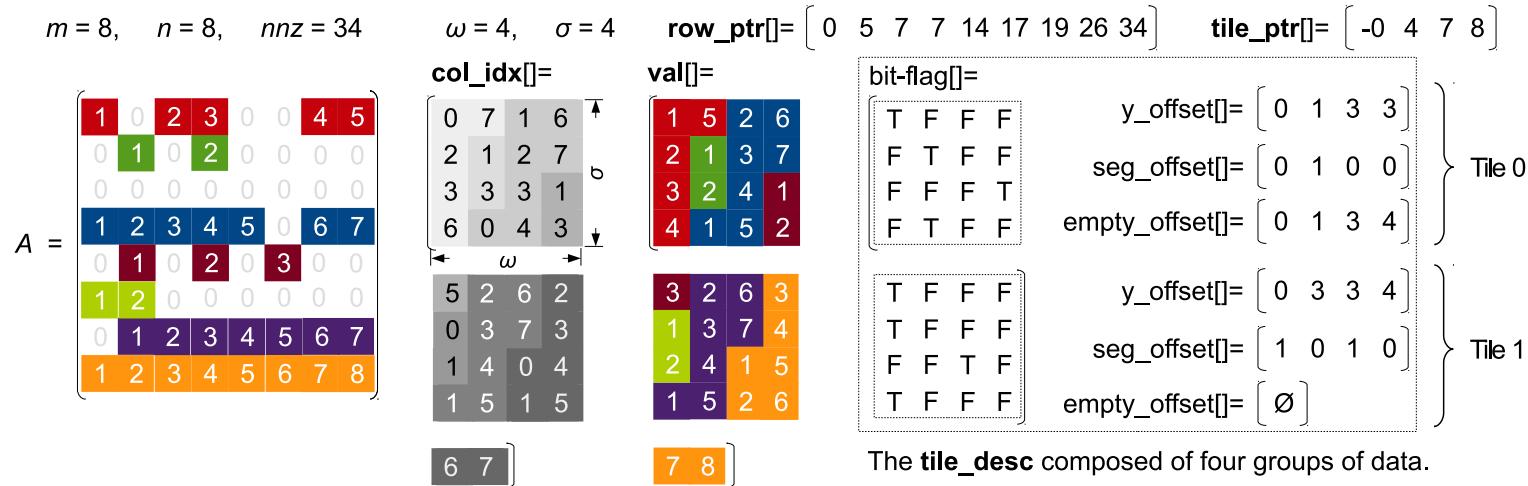
Is it possible to keep scalability and good performance, and reduce the preprocessing cost?

power-law
dist.



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

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

A (8x8), 34 nonzeros

core 0

core 1

CSR5: tiling

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

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

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

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

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

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

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

A (8x8), 34 nonzeros

column index =

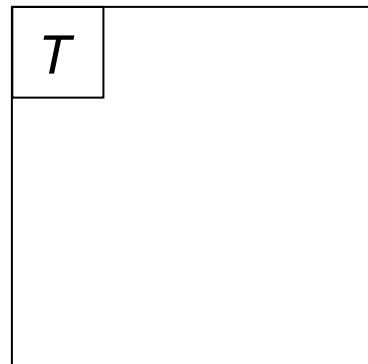
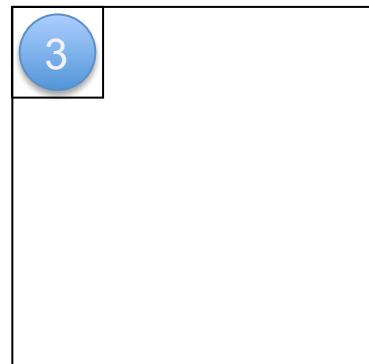
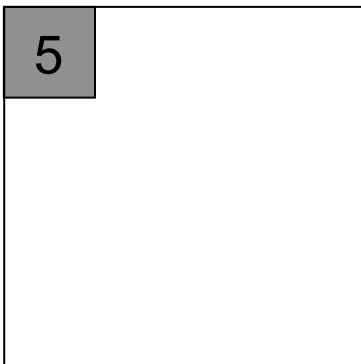
0	7	1	6
2	1	2	7
3	3	3	1
6	0	4	3

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



CSR5: tiling

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

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

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

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	2	6	
0	3	7	
1	4		
1	5		

3	2	6	
1	3	7	
2	4		
1	5		

T	F	F	
T	F	F	
F	F		
T	F		

CSR5: tiling

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

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

6 7

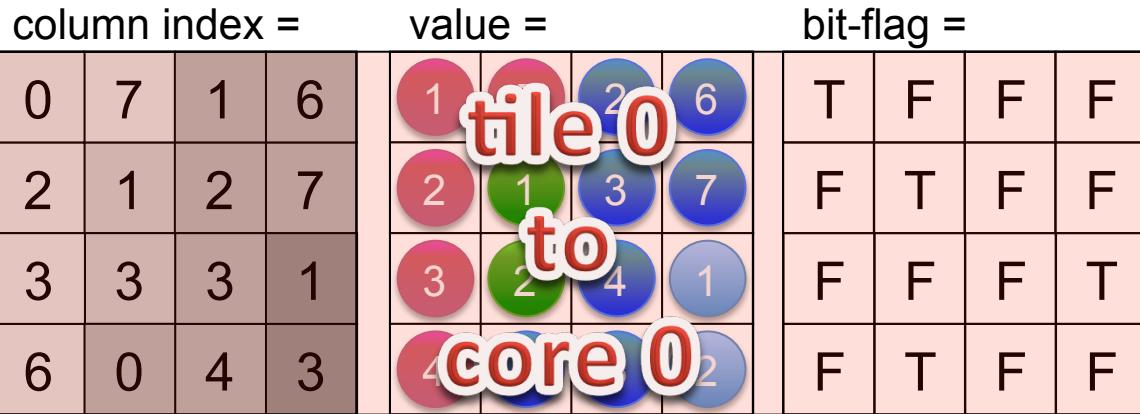
3	2	6	3
1	3	7	4
2	4	1	5
1	5	2	6

7	8
---	---

CSR5: tiling

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

A (8x8), 34 nonzeros



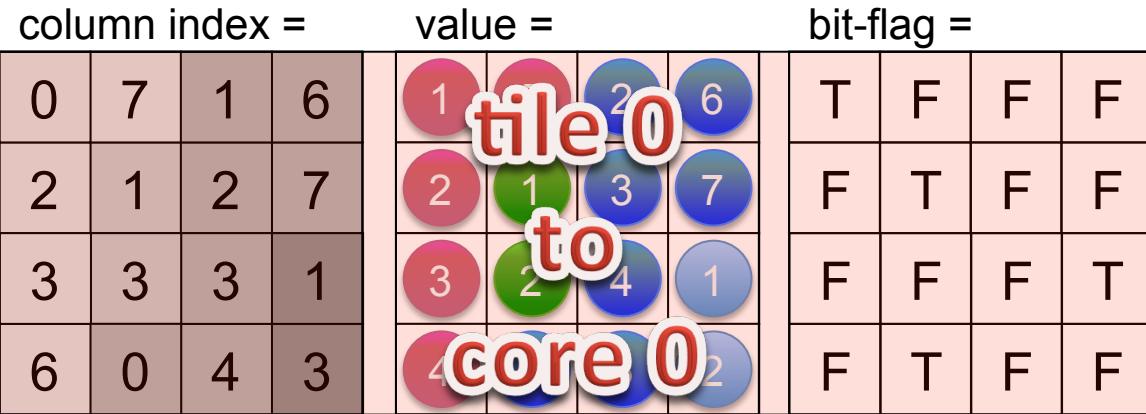
6 7

7 8

CSR5: tiling

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

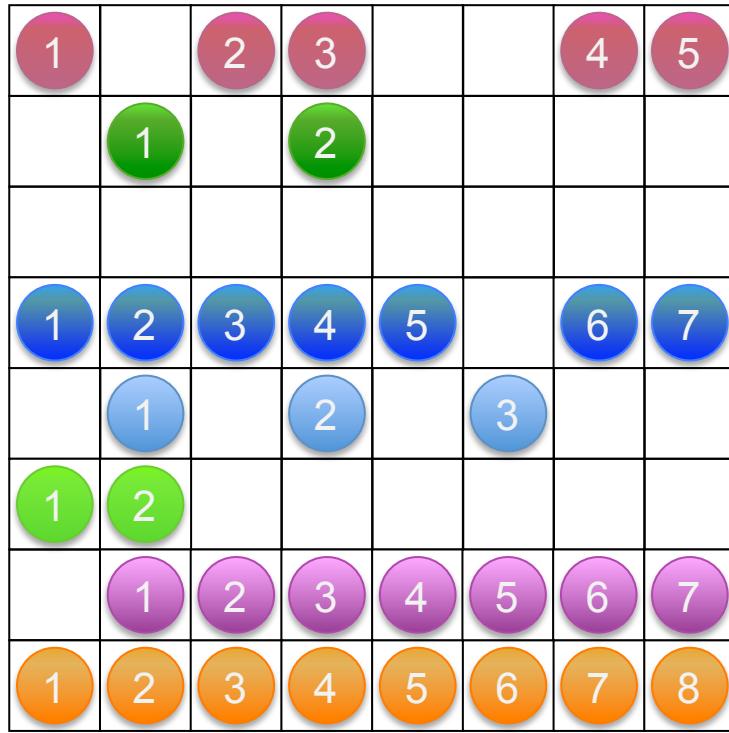
A (8x8), 34 nonzeros



5	2	6	2	3 tile 1	T F F F
0	3	7	3	1 3 7 4	T F F F
1	4	0	4	2 4 1 5	F F T F
1	5				

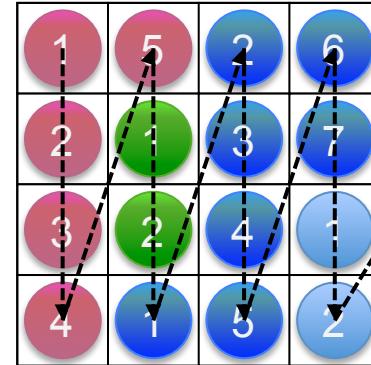
Tile, but not nonzero, is the basic work unit for load balancing.

CSR5: tile-wise transposition

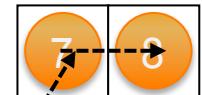
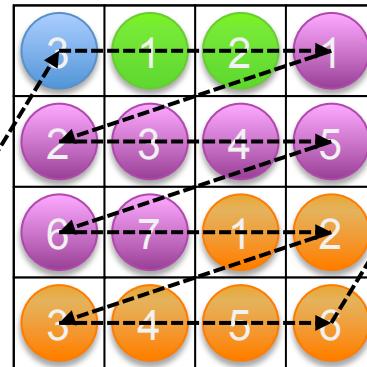
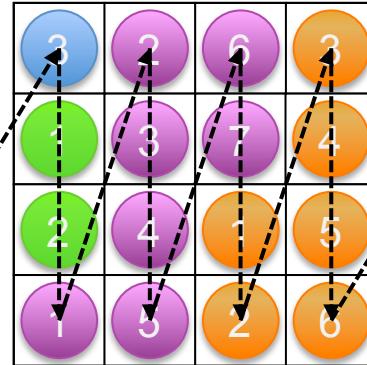
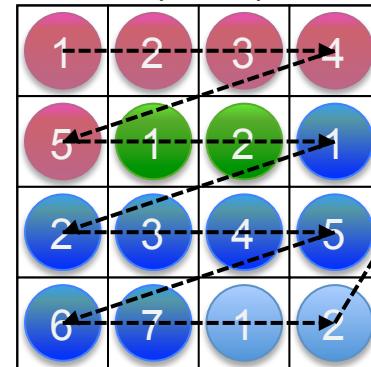


A (8x8), 34 nonzeros

value (CSR5) =



value (CSR) =

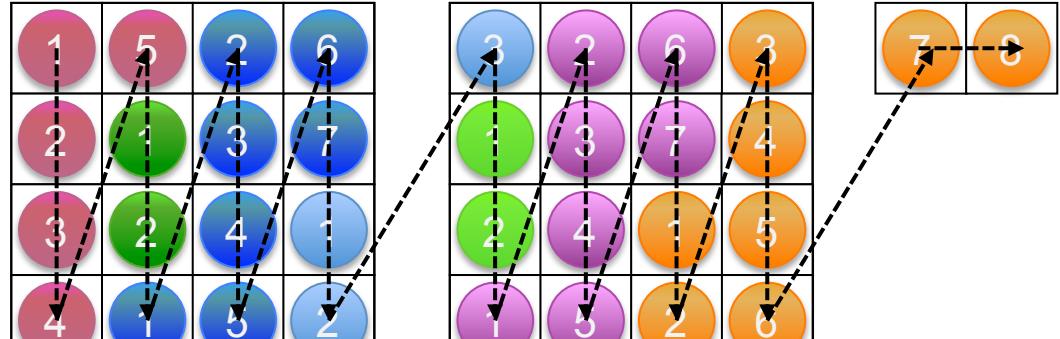


CSR5: tile-wise transposition

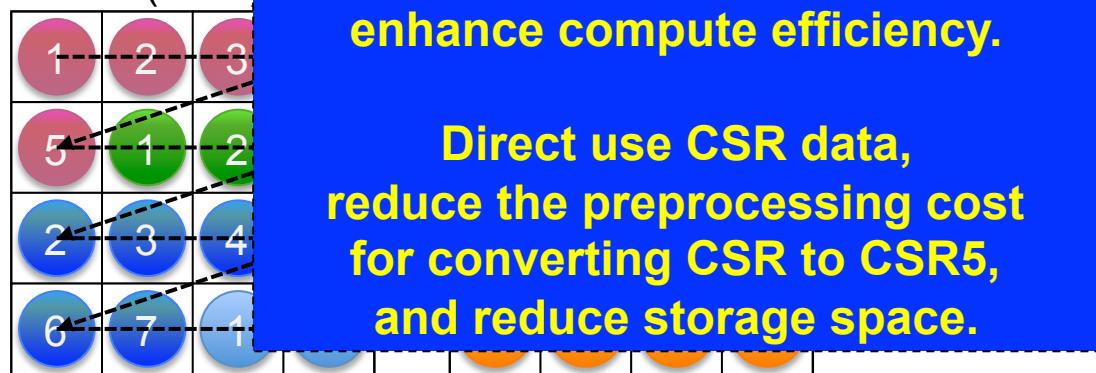
1		2	3			4	5
	1		2				
1	2	3	4	5		6	7
	1		2		3		
1	2						
1	2	3	4	5	6	7	
1	2	3	4	5	6	7	8

A (8x8), 34 nonzeros

value (CSR5) =



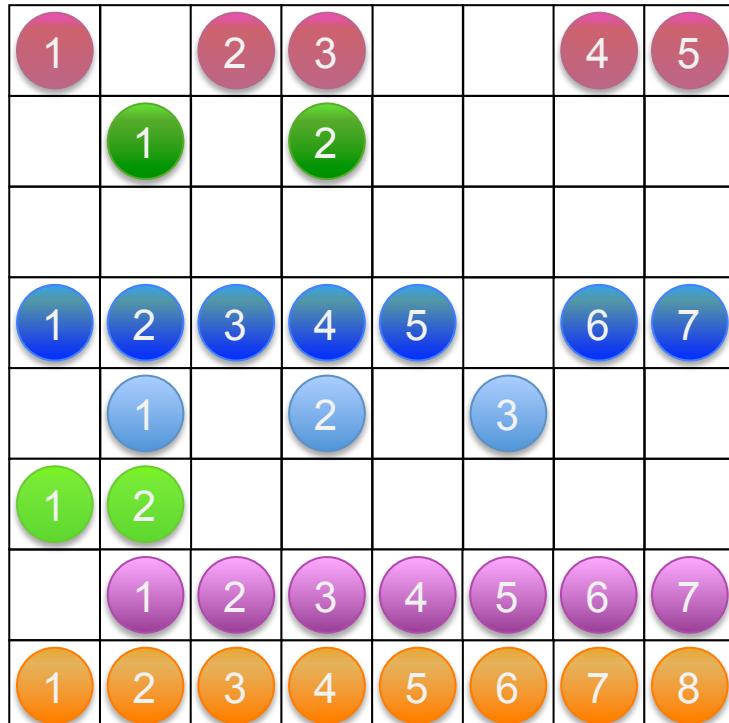
value (CSR)



Match SIMD pattern,
enhance compute efficiency.

Direct use CSR data,
reduce the preprocessing cost
for converting CSR to CSR5,
and reduce storage space.

CSR5: tile pointer array



A (8x8), 34 nonzeros

tile pointer =

0	4	7	8
---	---	---	---

value =

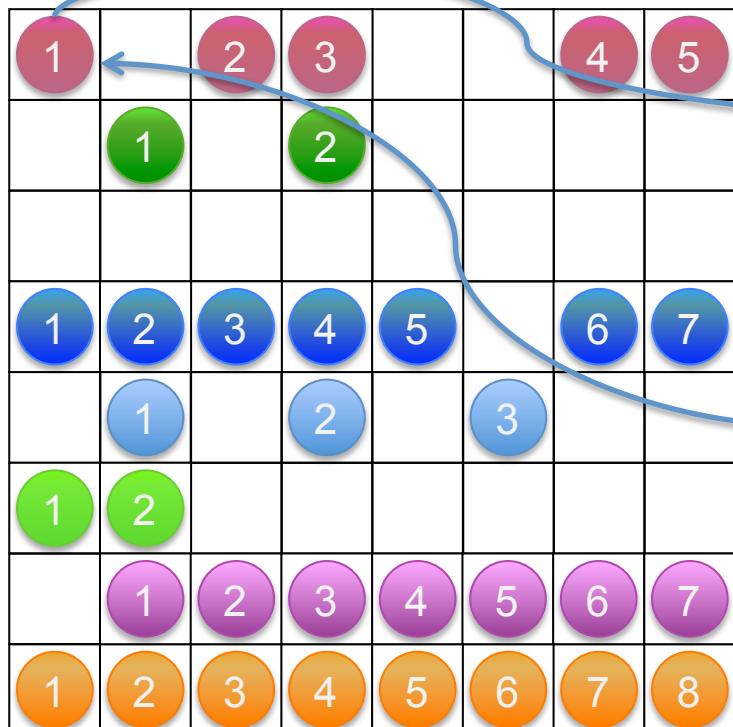
1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

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

7	8
---	---

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 =

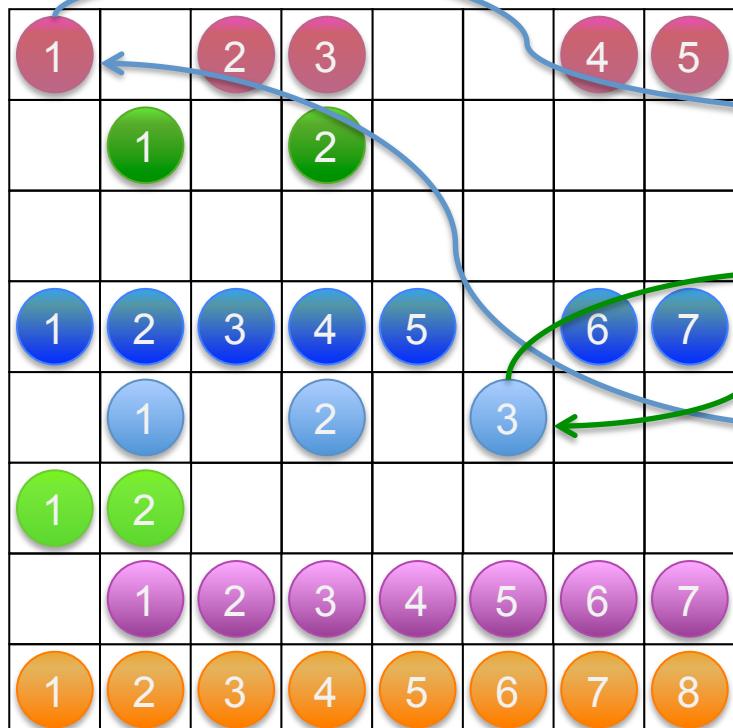
1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

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

7	8
---	---

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 =

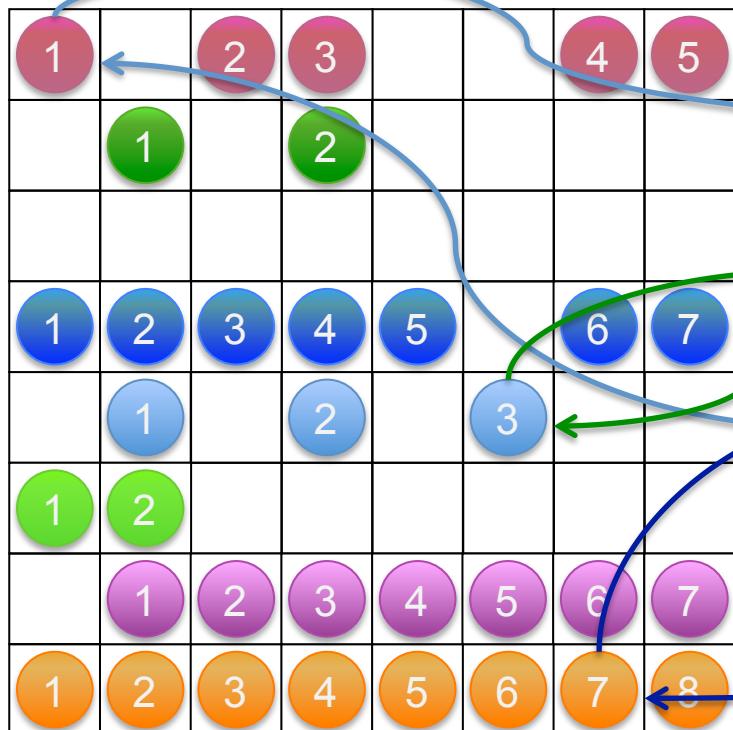
1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

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

7	8
---	---

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 =

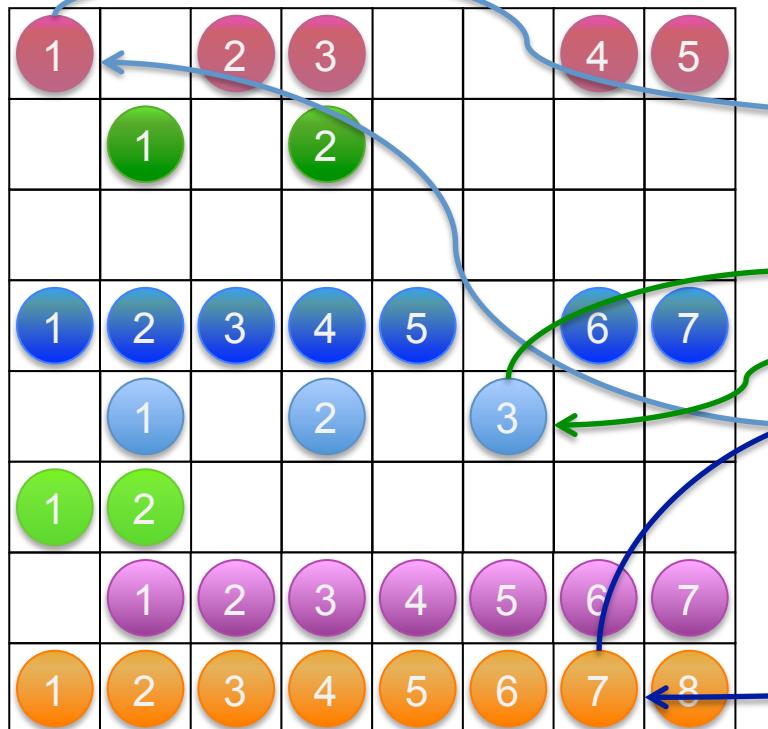
1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

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

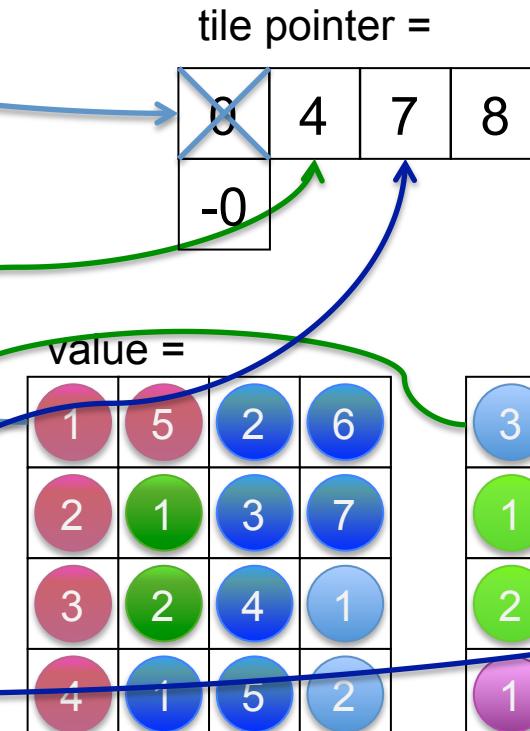
7	8
---	---

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 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

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

A (8x8), 34 nonzeros

tile
pointer =

-0	4	7	8
----	---	---	---

y offset =

0	1	3	3
---	---	---	---

value =

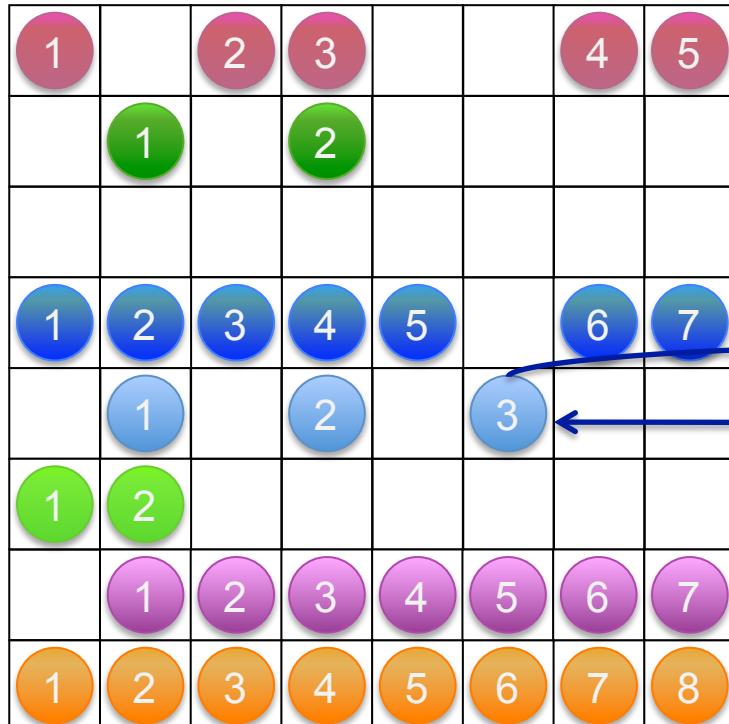
1	5	2	6
2	1	3	7
3	2	4	1
4	1	5	2

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

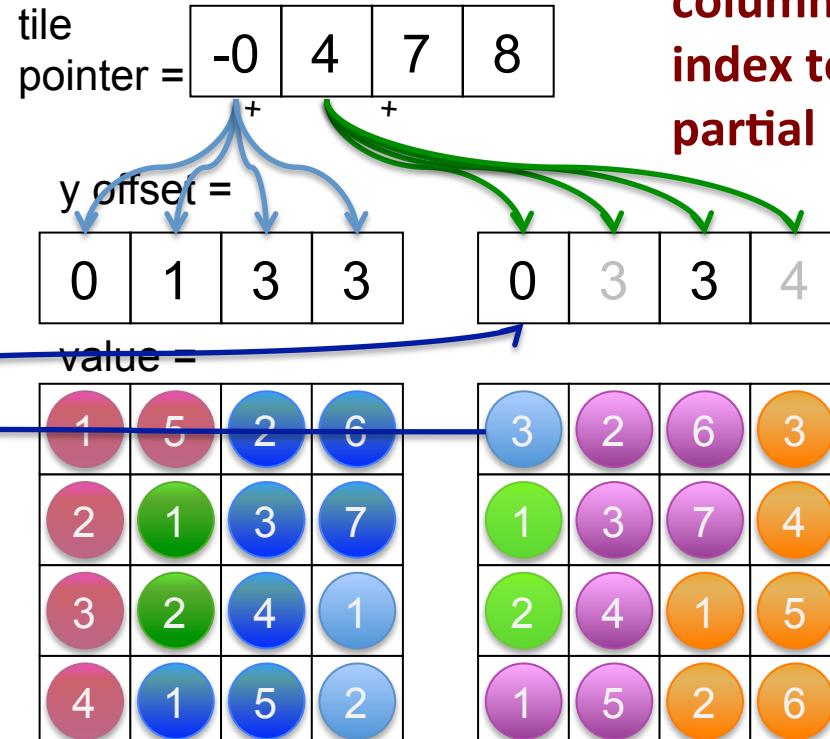
7	8
---	---

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

CSR5: tile descriptor info

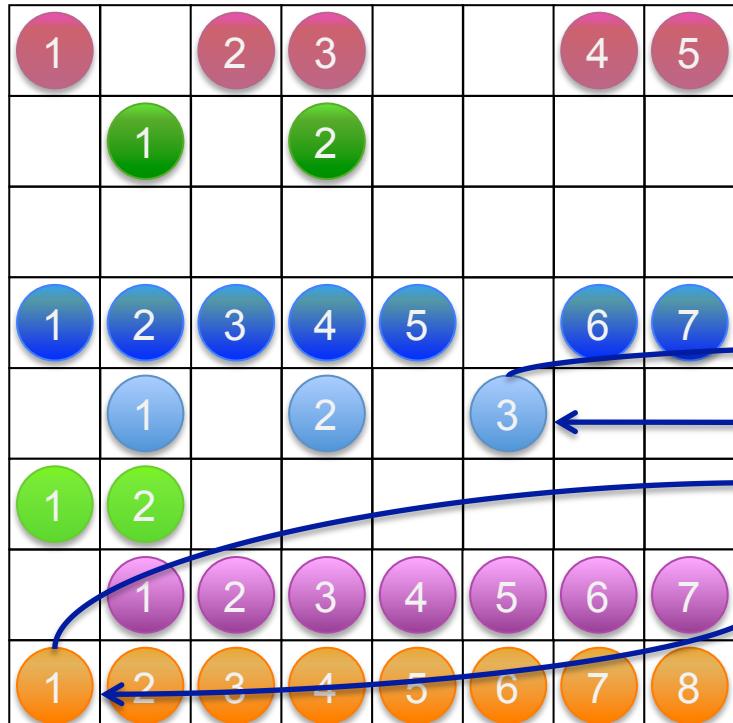


A (8x8), 34 nonzeros

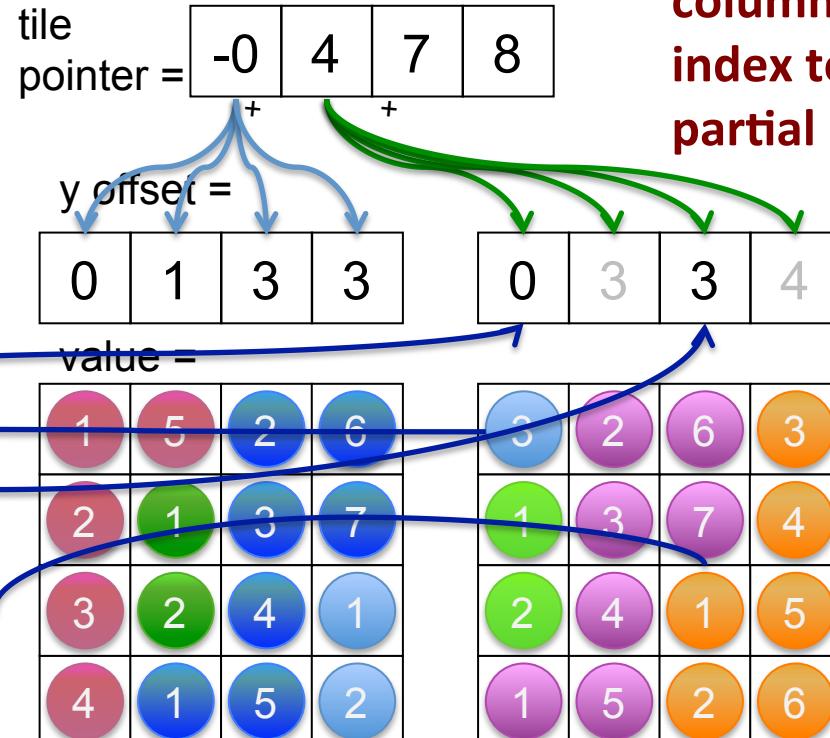


y offset lets each column know the y index to save its partial products.

CSR5: tile descriptor info

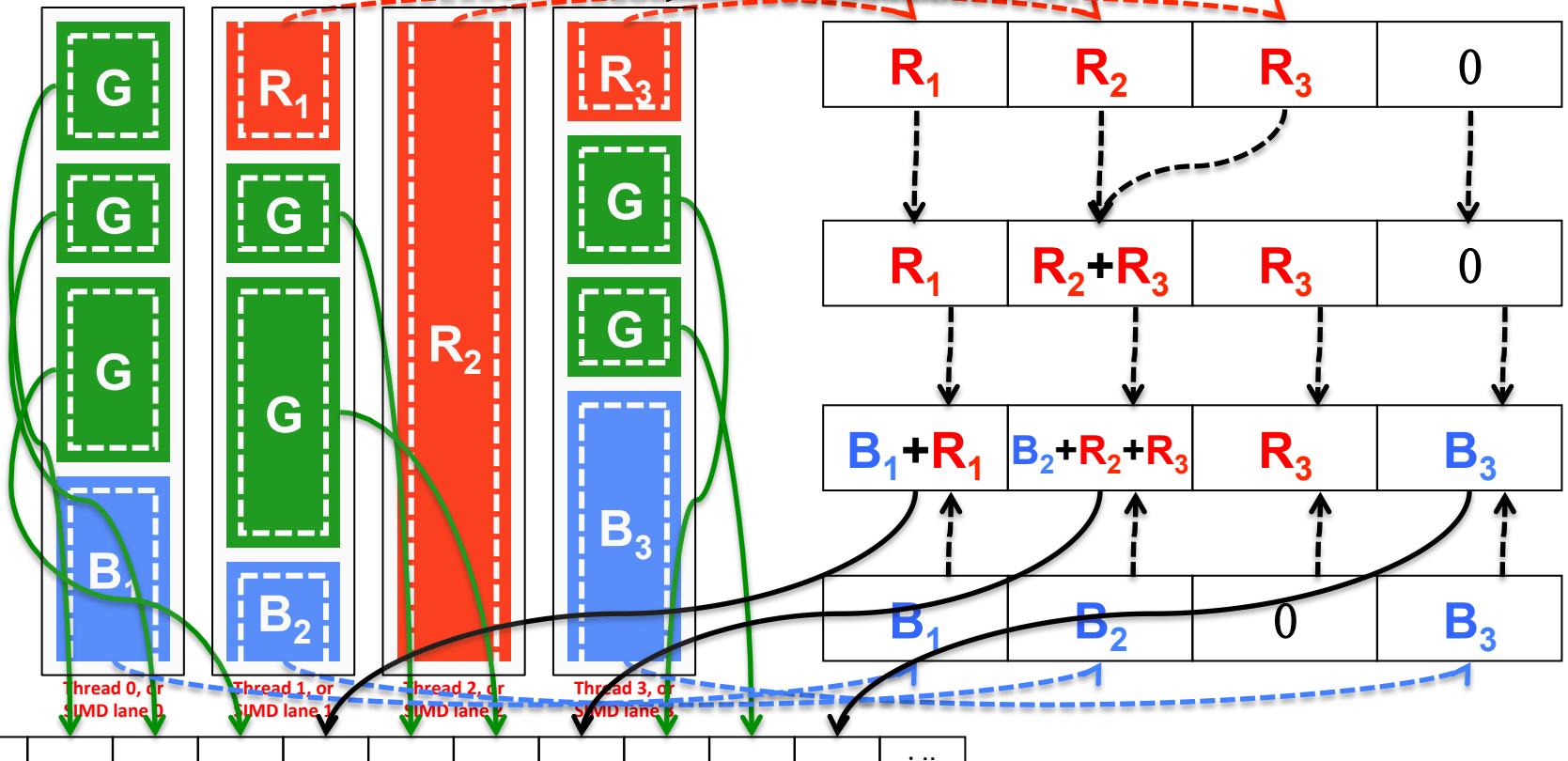


A (8x8), 34 nonzeros

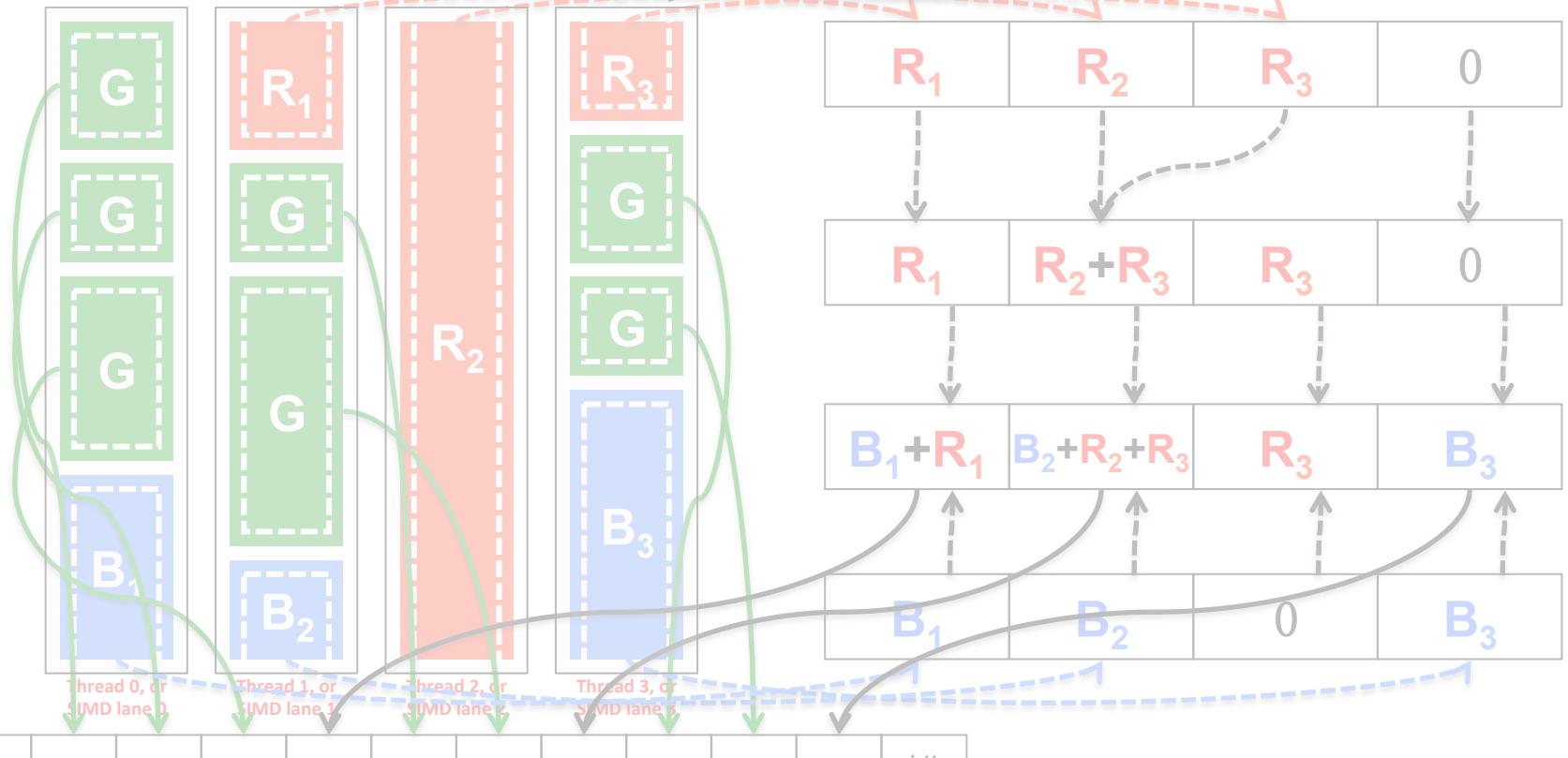


y offset lets each column know the y index to save its partial products.

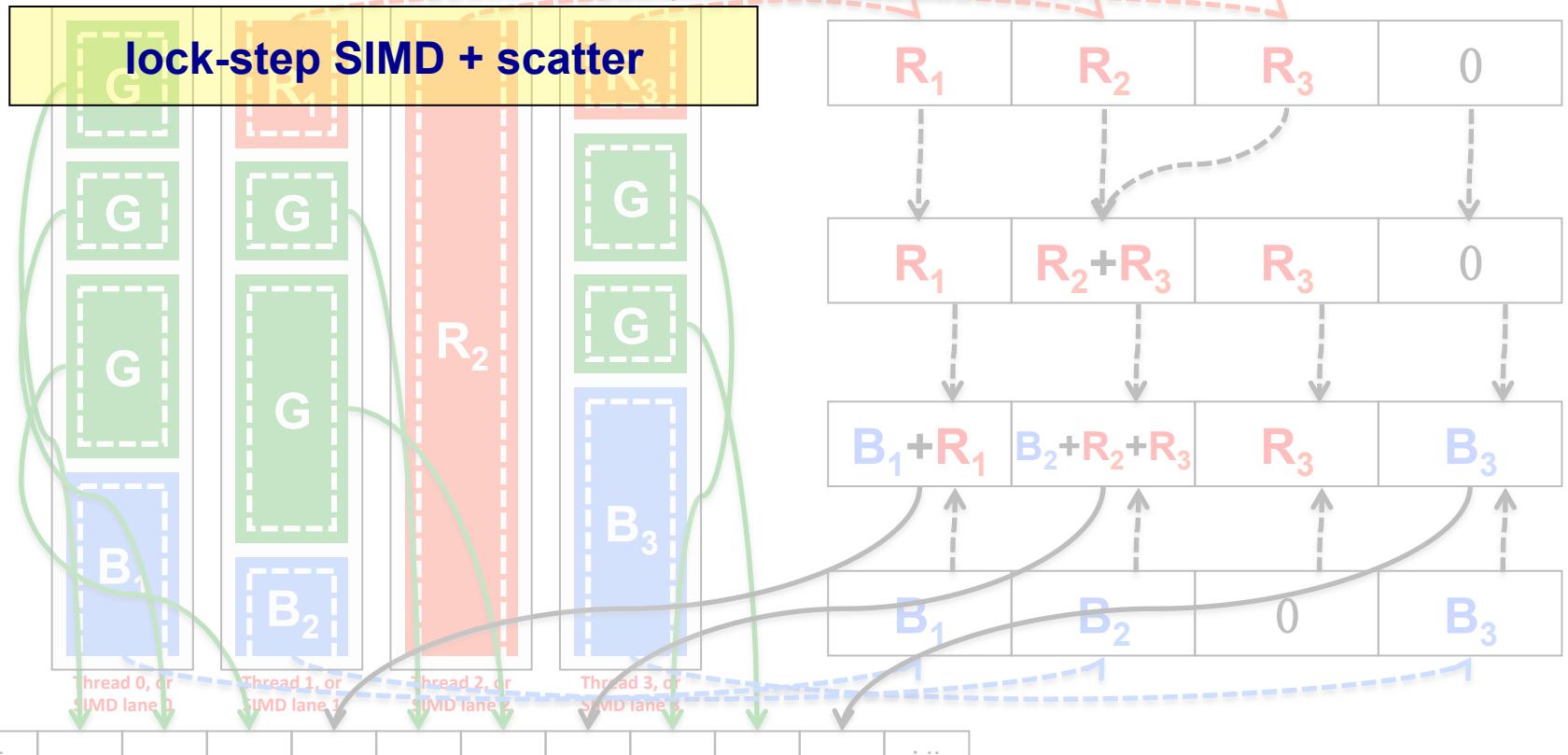
CSR5: SIMD-friendly and cross platform



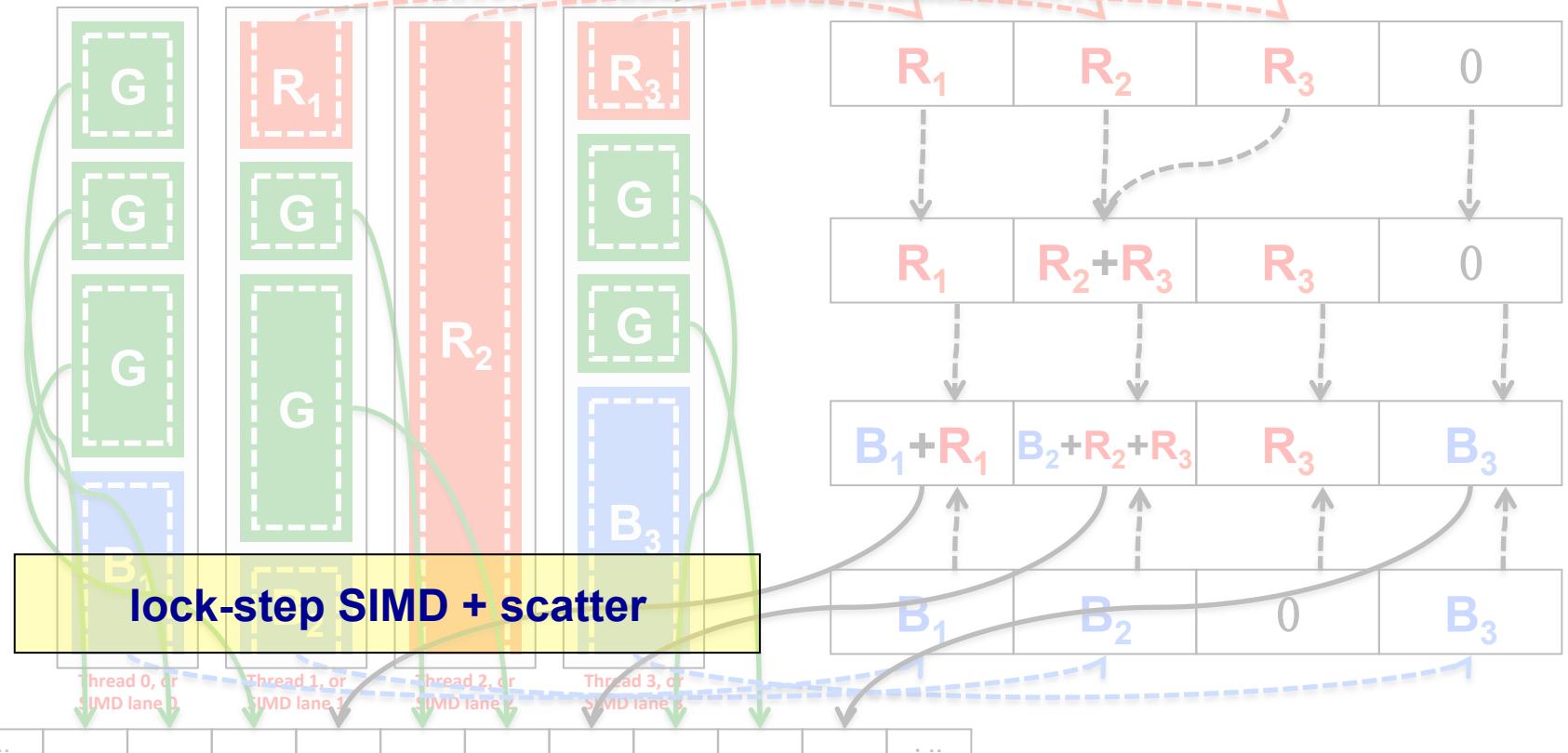
CSR5: SIMD-friendly and cross platform



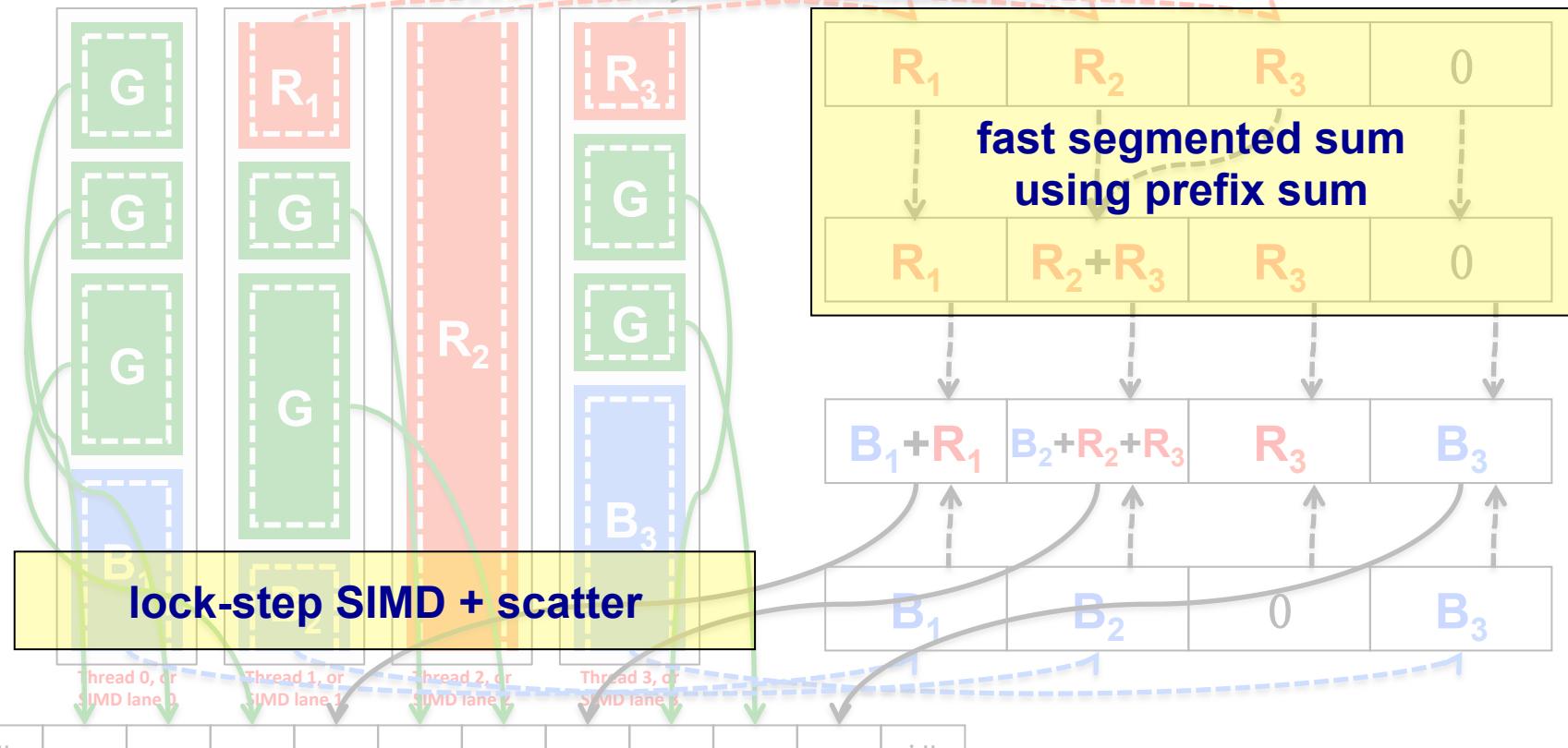
CSR5: SIMD-friendly and cross platform



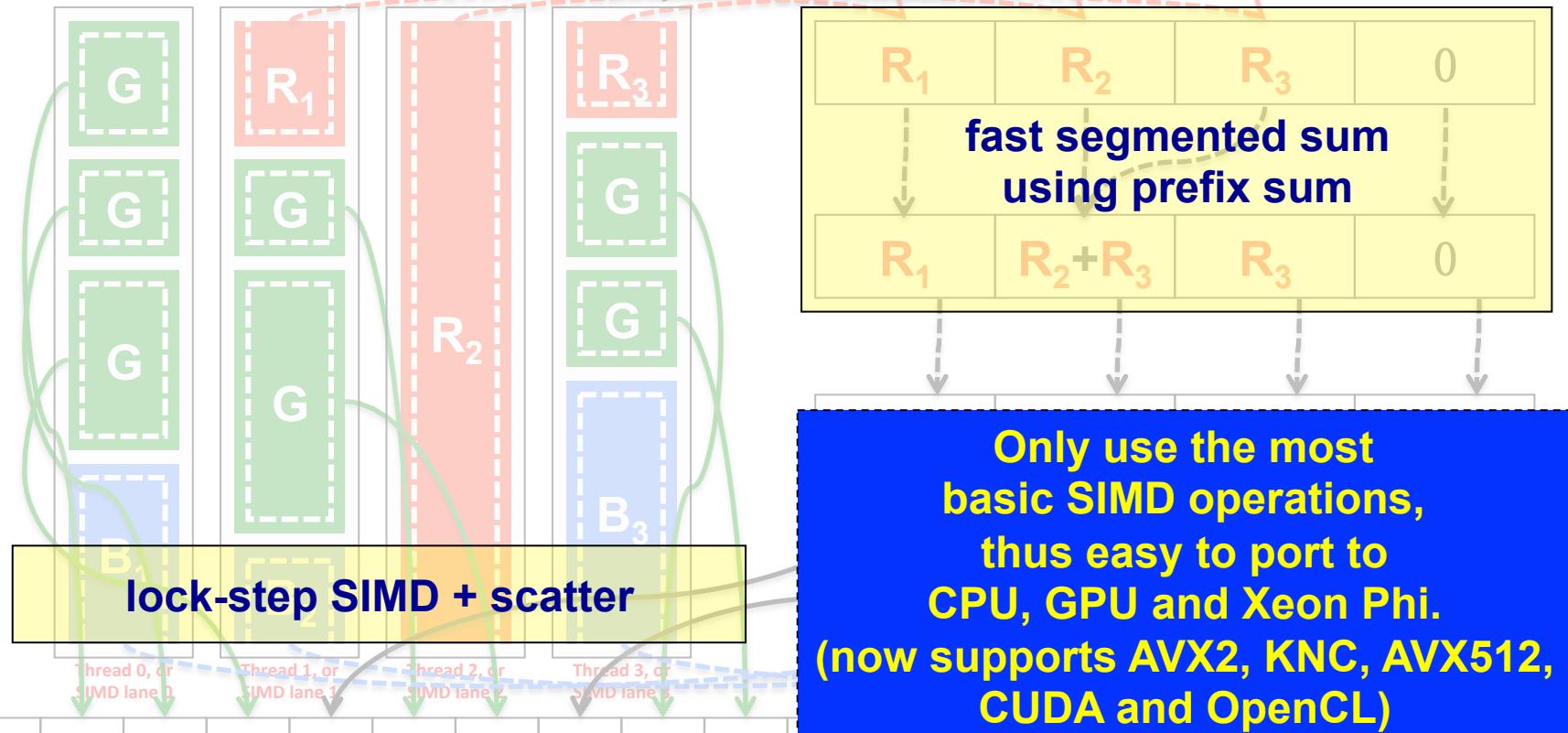
CSR5: SIMD-friendly and cross platform



CSR5: SIMD-friendly and cross platform



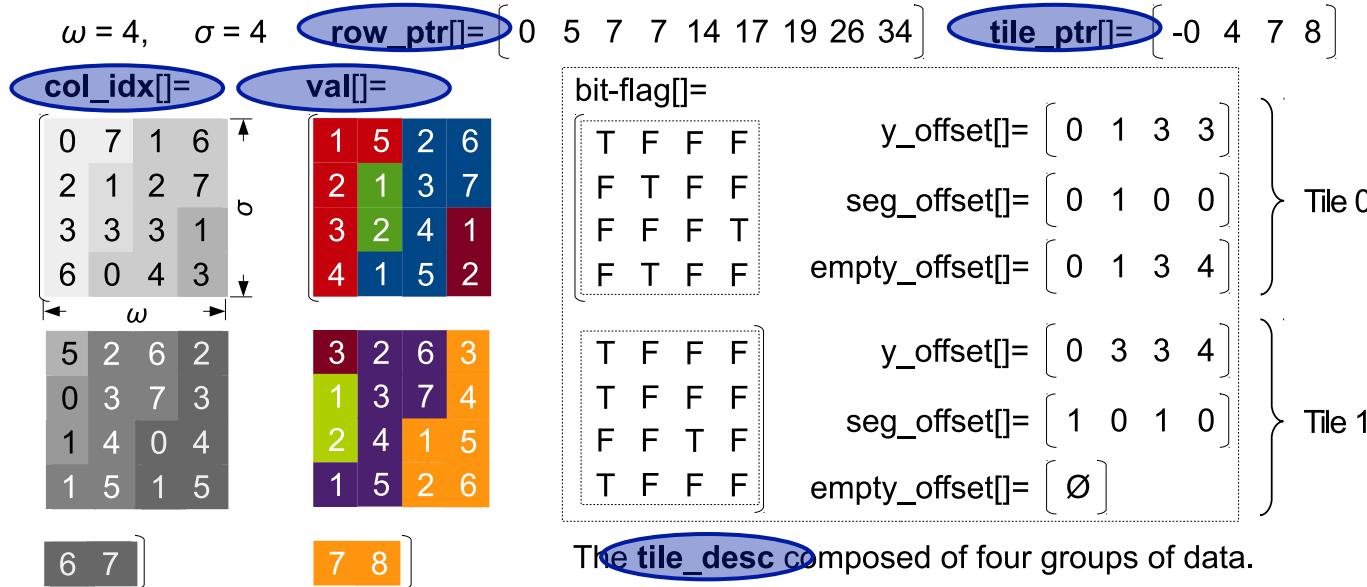
CSR5: SIMD-friendly and cross platform



CSR5: naming

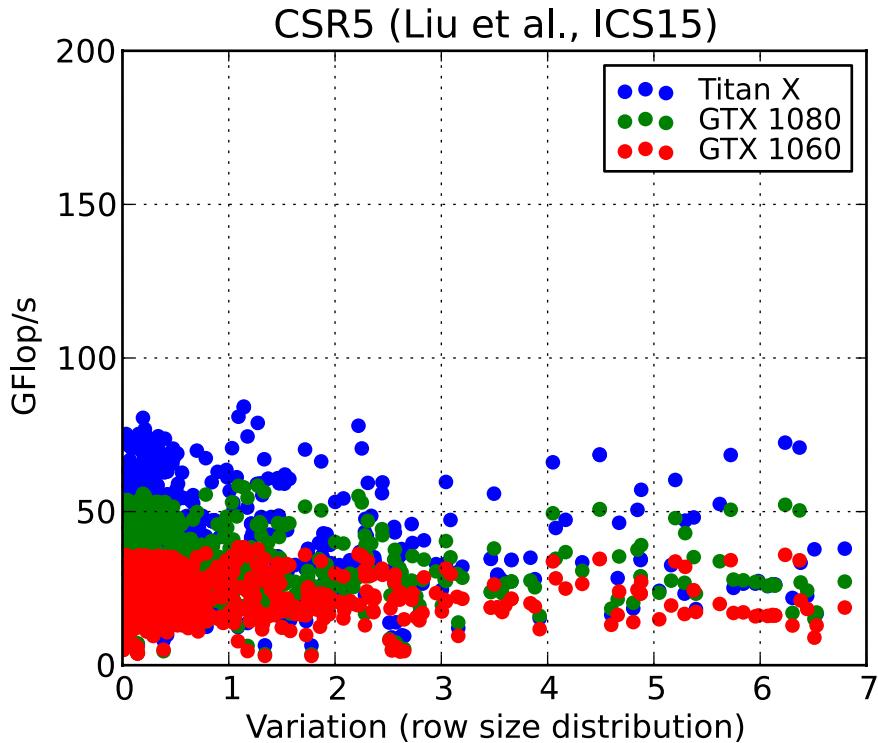
$m = 8, n = 8, nnz = 34$

1	0	2	3	0	0	4	5
0	1	0	2	0	0	0	0
0	0	0	0	0	0	0	0
1	2	3	4	5	0	6	7
0	1	0	2	0	3	0	0
1	2	0	0	0	0	0	0
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8

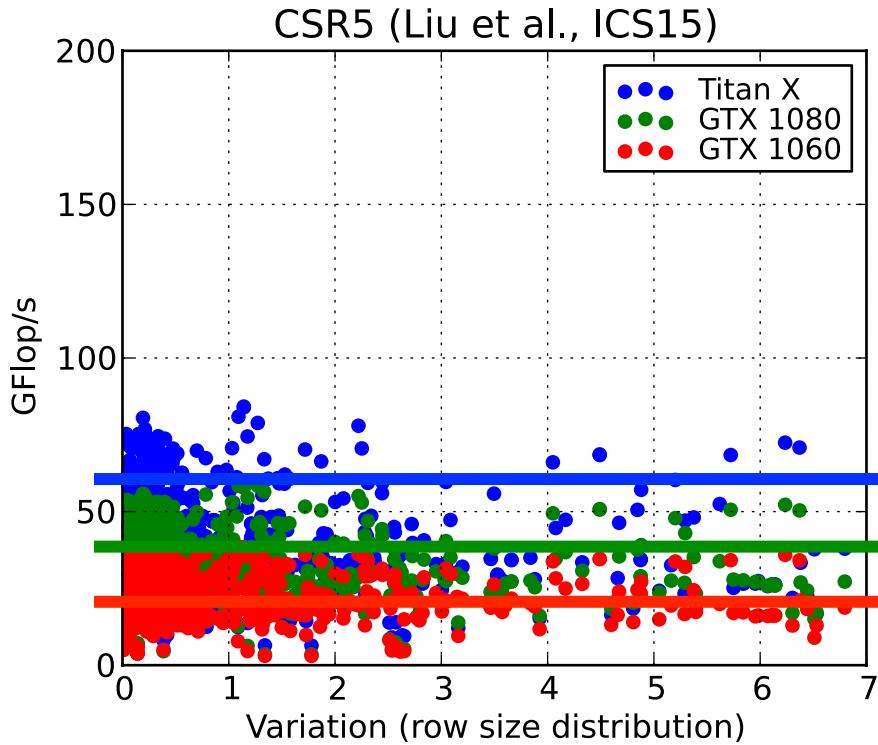


Compared three groups of info for CSR, CSR5 uses CSR data and stores five groups of info. So it is called CSR5.

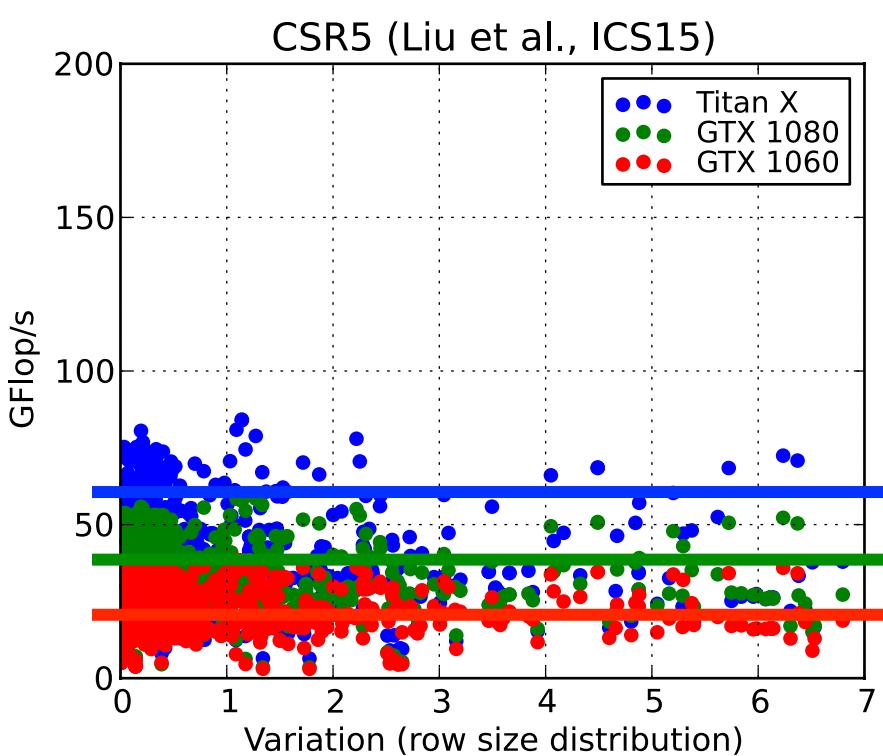
Scalability of SpMV using CSR5



Scalability of SpMV using CSR5

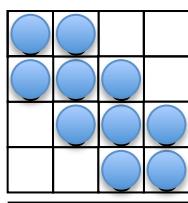


Scalability of SpMV using CSR5

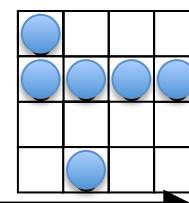


SpMV perf.
scales with
hardware perf.,
and is insensitive
to sparsity
structures.

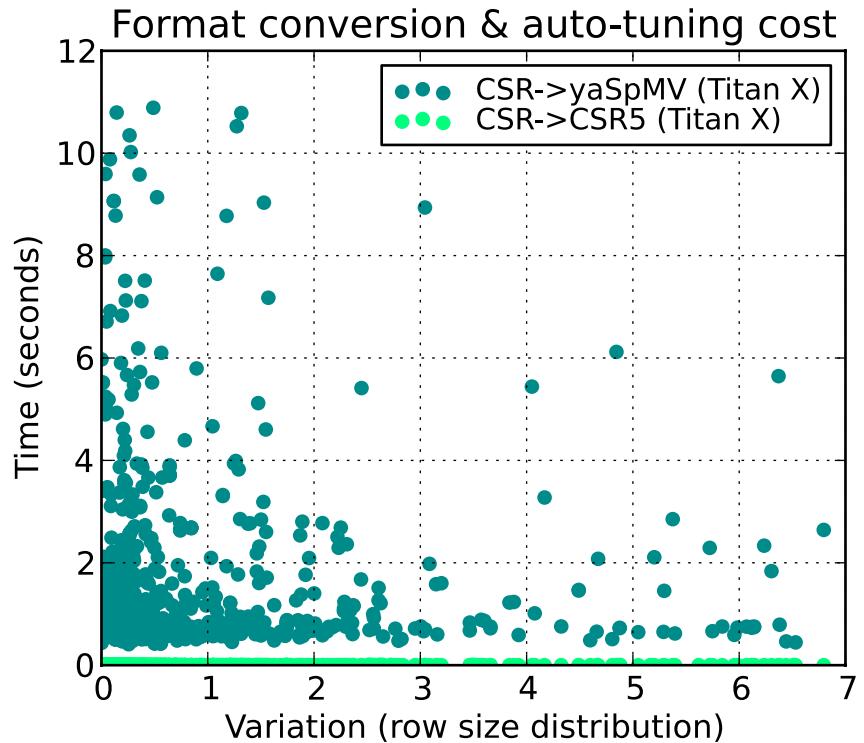
uniform
dist.



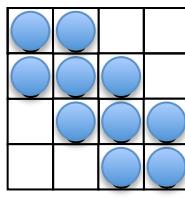
power-law
dist.



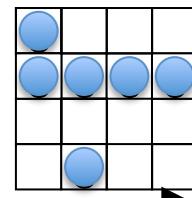
Data conv. and autotuning cost of CSR5



uniform
dist.

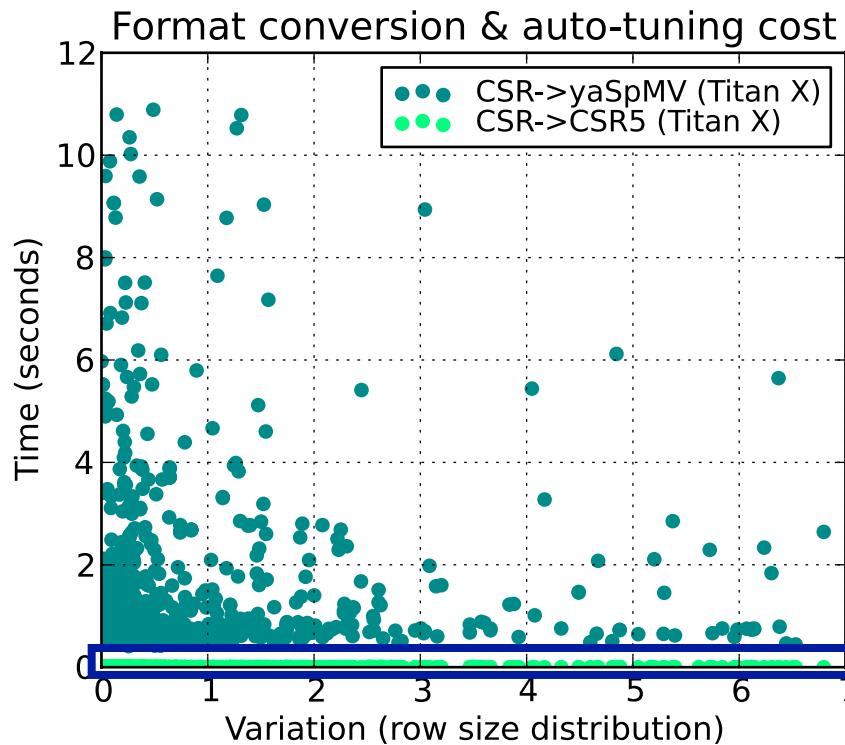
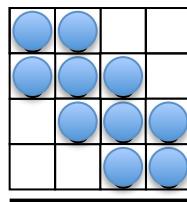


power-law
dist.



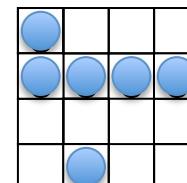
Data conv. and autotuning cost of CSR5

uniform
dist.



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

power-law
dist.



Source code of CSR5 on Github

This repository contains CSR5-based SpMV on CPUs, GPUs and Xeon Phi.

42 commits · 1 branch · 0 releases · 2 contributors · MIT

Branch: master · New pull request · Create new file · Upload files · Find file · Clone or download

Commit	Description	Date
bhSPARSE rename folder phi to knc_phi	Latest commit 7640edb on Jan 5	
CSR5_avx2	Fixed issues that resulted in wrong results with a not initialized y-...	a year ago
CSR5_avx512	update avx512 version for KNL and OpenCL version for nVidia GPUs	8 months ago
CSR5_cuda	Update anonymouslib_cuda.h	a year ago
CSR5_knc_phi	rename folder phi to knc_phi	8 months ago
CSR5_opencl_amd	Upload.	3 years ago
CSR5_opencl_nvidia	update avx512 version for KNL and OpenCL version for nVidia GPUs	8 months ago
LICENSE	Initial commit	3 years ago
README.md	Update README.md	8 months ago

https://github.com/bhSPARSE/Benchmark_SpMV_using_CSR5

CSR5 in MAGMA

MAGMA

ICL-UTK

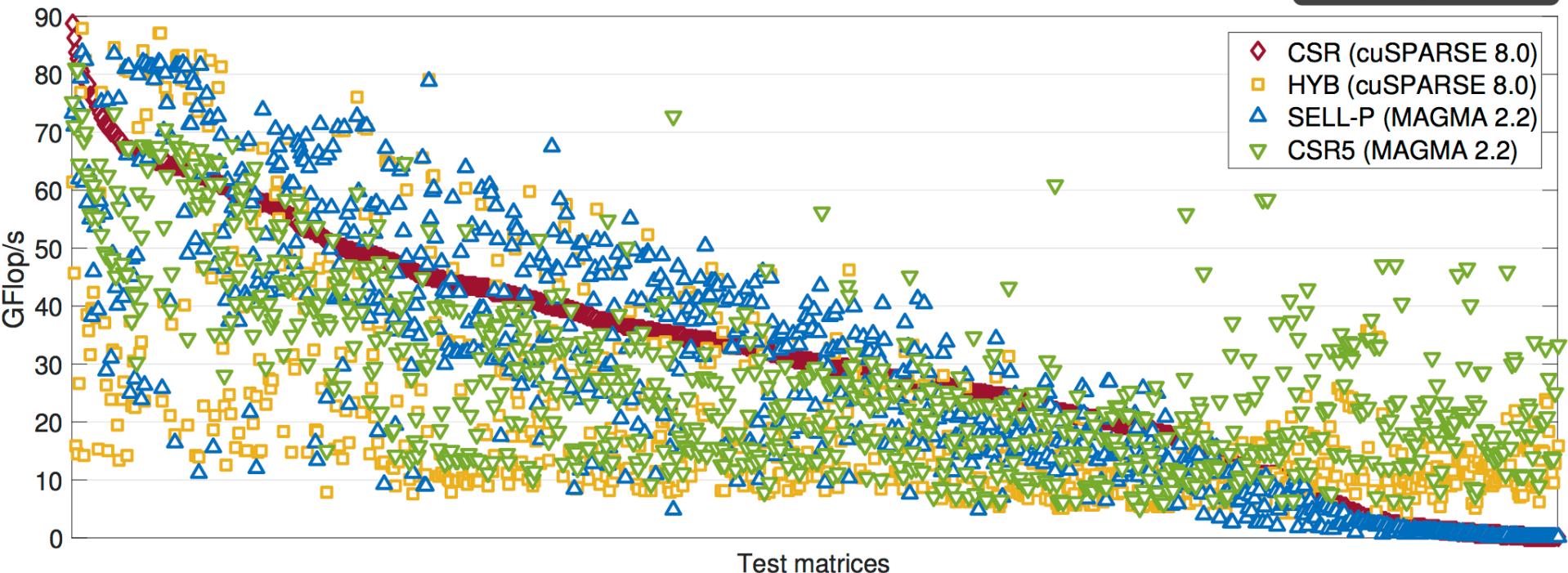
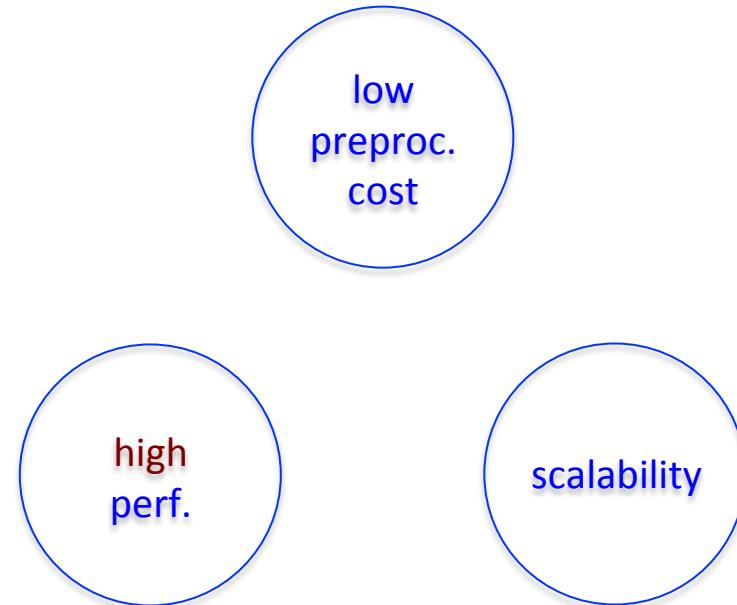


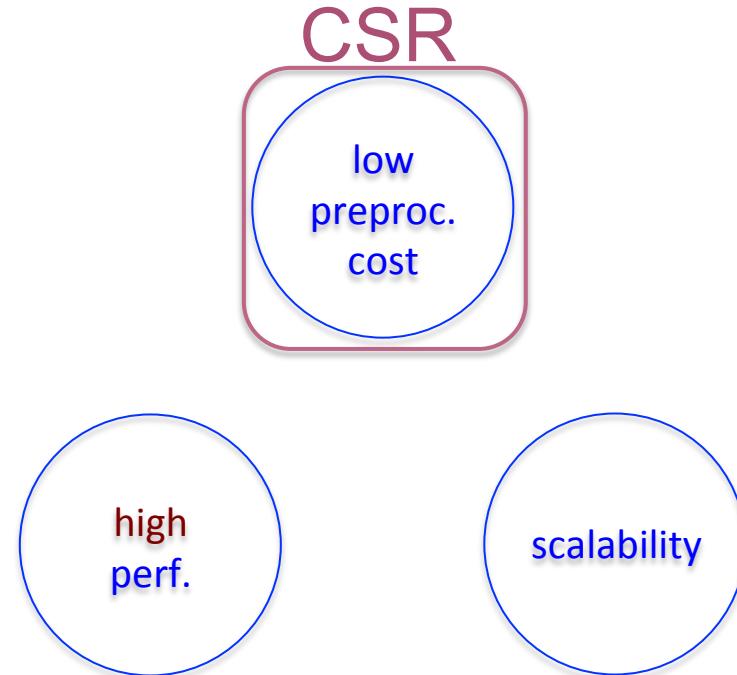
Figure from Dr. Hartwig Anzt at ICL, UTK; Device: NVIDIA Tesla P100; Precision: double.

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



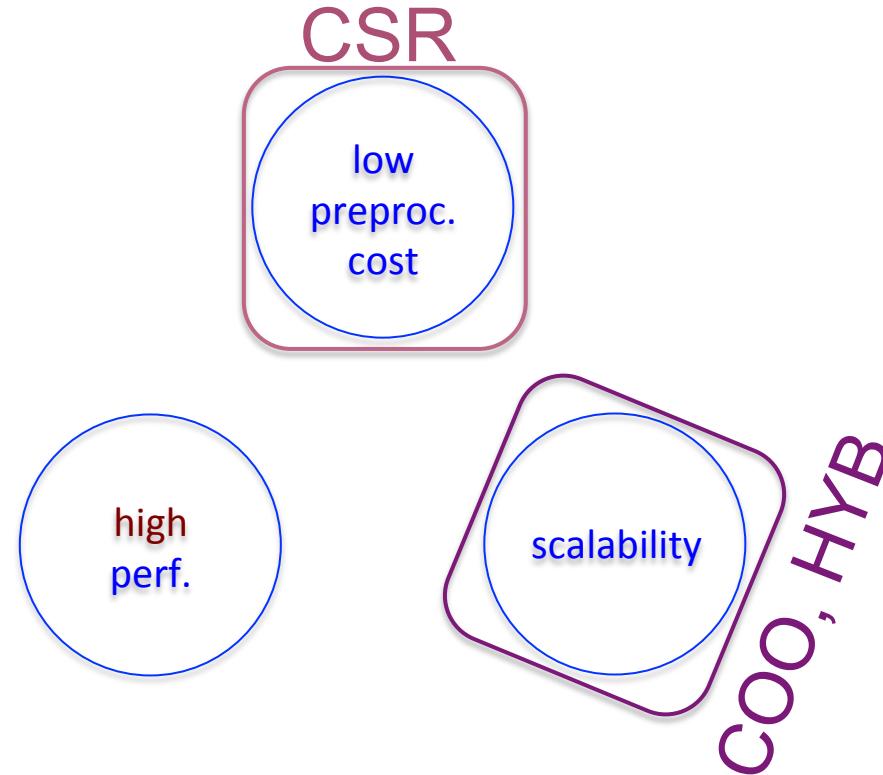
Scalability, high perf. and low preproc. cost

(choose two of the three?)



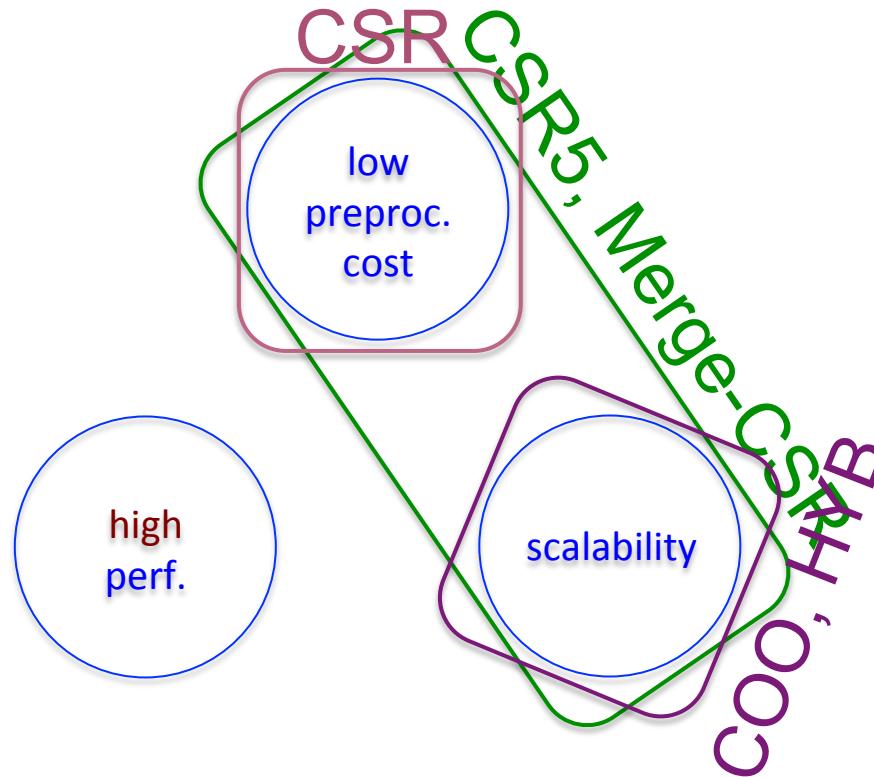
Scalability, high perf. and low preproc. cost

(choose two of the three?)



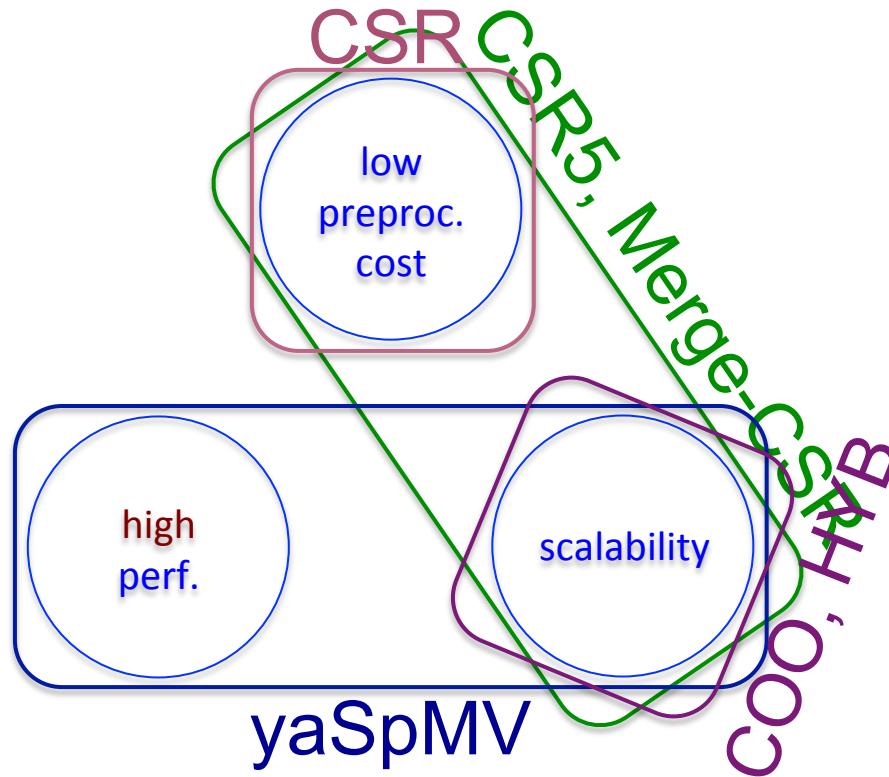
Scalability, high perf. and low preproc. cost

(choose two of the three?)



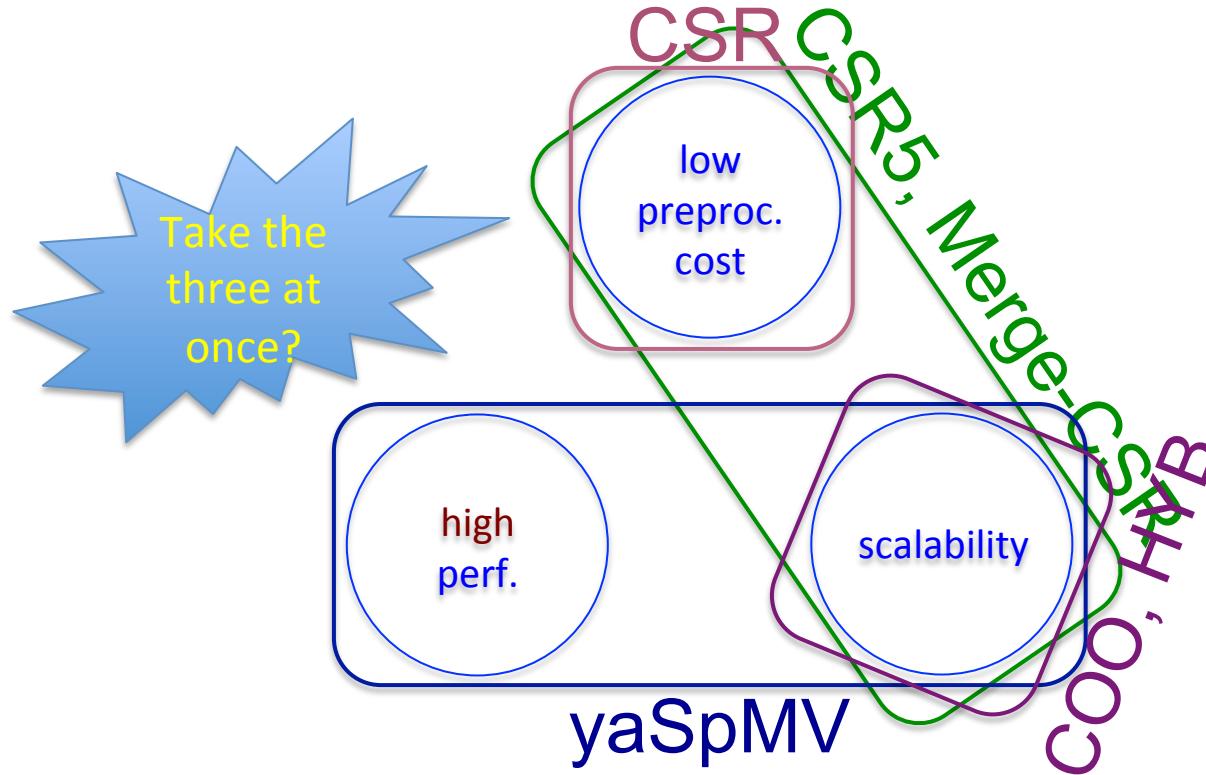
Scalability, high perf. and low preproc. cost

(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 .

$$\begin{array}{c} \begin{array}{|c|c|c|c|} \hline & & 1 & \\ \hline 2 & 3 & & \\ \hline & & & \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline & & a & \\ \hline b & & c & \\ \hline & d & & e \\ \hline & & f & \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline & 1d & & 1e \\ \hline 3b & & 3c & 2a \\ \hline & & & \\ \hline 5d & 6f & & 4a+5e \\ \hline \end{array} \end{array}$$

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.

$$\begin{array}{|c|c|c|c|} \hline & & 1 & \\ \hline 2 & 3 & & \\ \hline & & & \\ \hline 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline & & & a \\ \hline b & & c & \\ \hline & d & & e \\ \hline & & f & \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline & & & ? \\ \hline & & & \\ \hline \end{array}$$

$A, nnzA = 6$

$B, nnzB = 6$

$C, nnzC = ?$

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.

$$\begin{array}{|c|c|c|c|} \hline & & & 1 \\ \hline & 2 & 3 & \\ \hline & & & \\ \hline & 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline & & & a \\ \hline & b & & c \\ \hline & & d & e \\ \hline & & & f \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline & & & ? \\ \hline & & & \\ \hline \end{array}$$

$A, nnzA = 6$ $B, nnzB = 6$ $C, nnzC = ?$

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

$$\begin{array}{|c|c|} \hline T & T \\ \hline T & T \\ \hline \end{array} \times \begin{array}{|c|c|} \hline T & T \\ \hline T & T \\ \hline \end{array} = \begin{array}{|c|c|} \hline T & T & T \\ \hline T & T & T \\ \hline \end{array}$$

Allocate
space of size $nnzC$

$$\begin{array}{|c|c|c|c|} \hline & & 1 & \\ \hline & 2 & 3 & \\ \hline & & & \\ \hline & 4 & & 5 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline & & a & \\ \hline & b & c & \\ \hline & & d & e \\ \hline & & & f \\ \hline \end{array}$$

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.

			1	
2	3			
4		5	6	

$A, nnzA = 6$

x

			a	
b			c	
	d		e	
		f		

$B, nnzB = 6$

=

$C, nnzC = ?$

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

$$\begin{array}{|c|c|} \hline T & T \\ \hline T & T \\ \hline \end{array} \times \begin{array}{|c|c|} \hline T & T \\ \hline T & T \\ \hline T & T \\ \hline \end{array} = \begin{array}{|c|c|} \hline T & T & T \\ \hline T & T & T \\ \hline T & T & T \\ \hline \end{array}$$



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.

			1	
2	3			
4		5	6	

x

			a	
b			c	
	d		e	
		f		

=

				?

$A, nnzA = 6$

$B, nnzB = 6$

$C, nnzC = ?$

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

$$\begin{array}{|c|c|} \hline T & T \\ \hline T & T \\ \hline \end{array} \times \begin{array}{|c|c|} \hline T & T \\ \hline T & T \\ \hline \end{array} = \begin{array}{|c|c|} \hline T & T & T \\ \hline \end{array}$$

Allocate
space of $nnzC$

One SpGEMM for Two!

2	3		
4	5	6	

Expensive

SpGEMM challenge 2 - parallel insertion

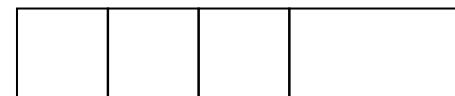
- Because of the compressed sparse format, the result nonzeros are “inserted” into C, but not “added” to predictable locations of C.

		1	
2	3		
4		5	6

A

		a	
b		c	
	d		e
		f	

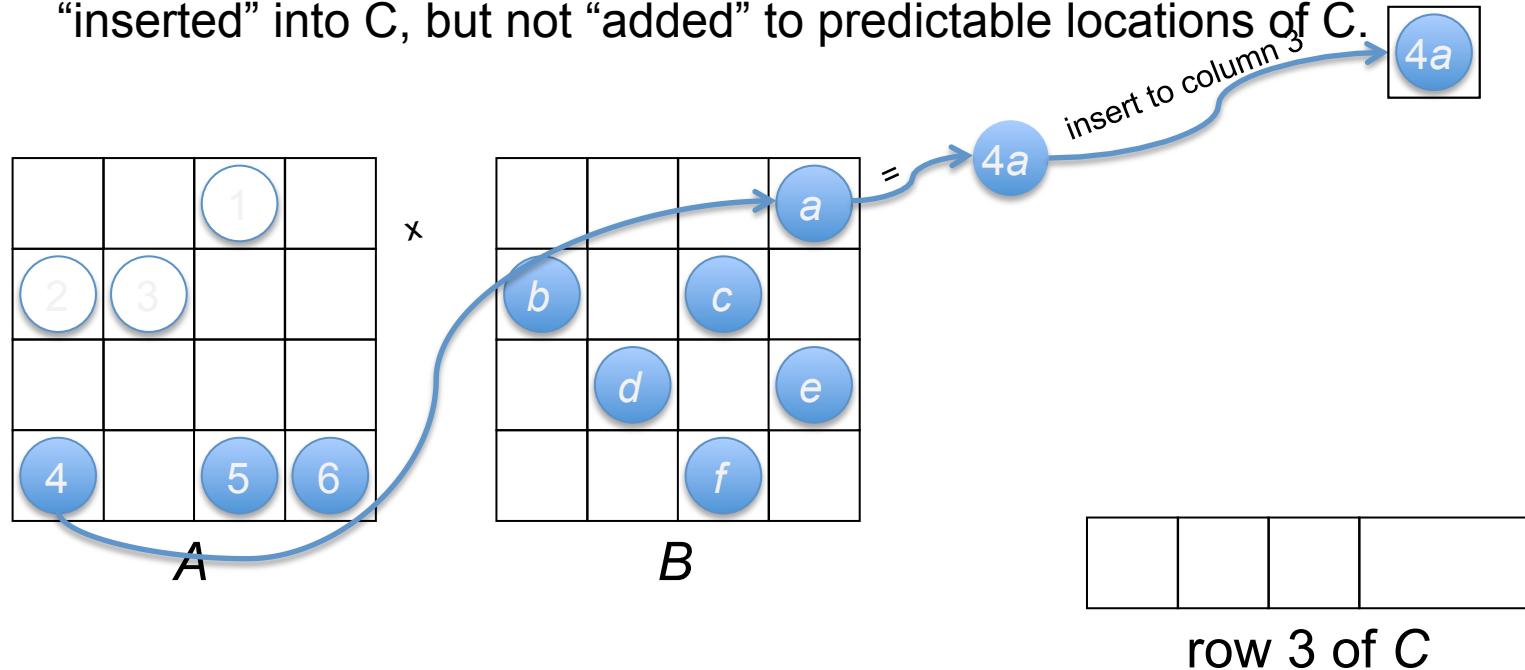
B



row 3 of C

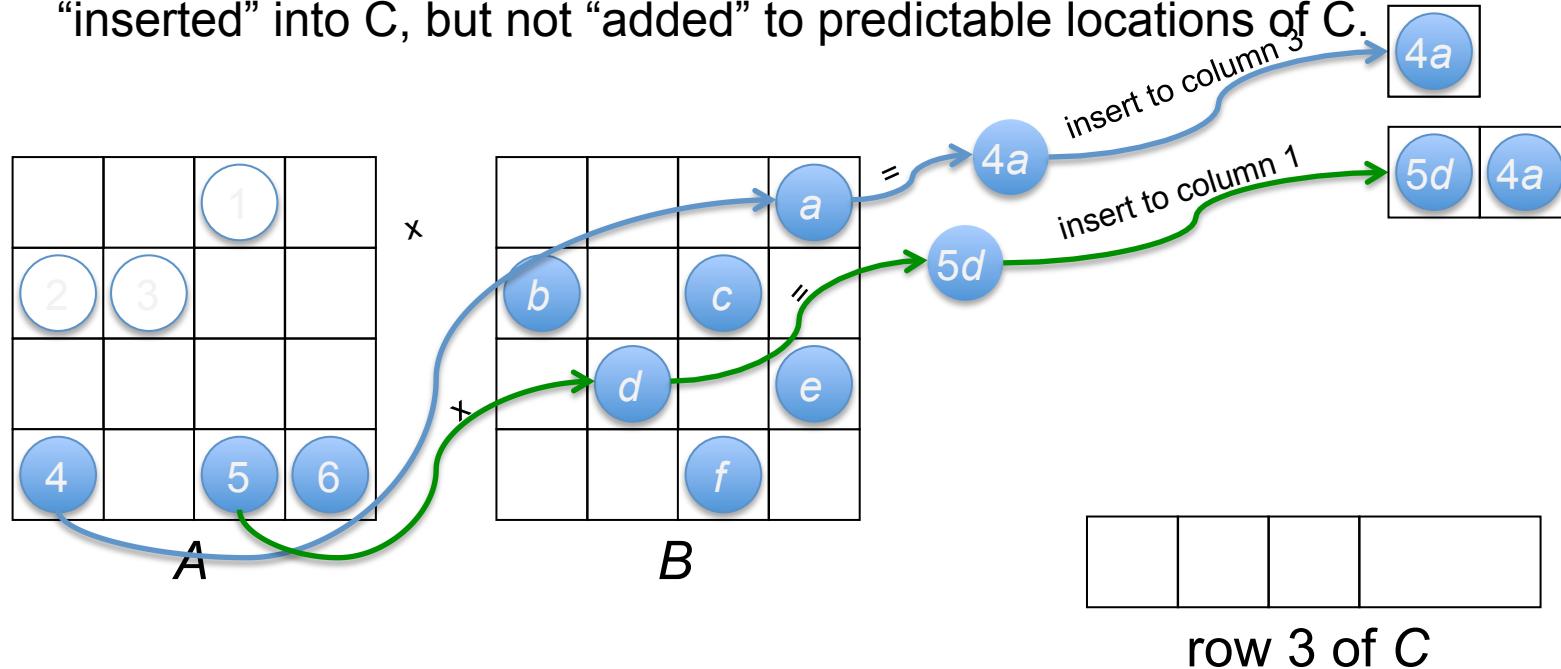
SpGEMM challenge 2 - parallel insertion

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



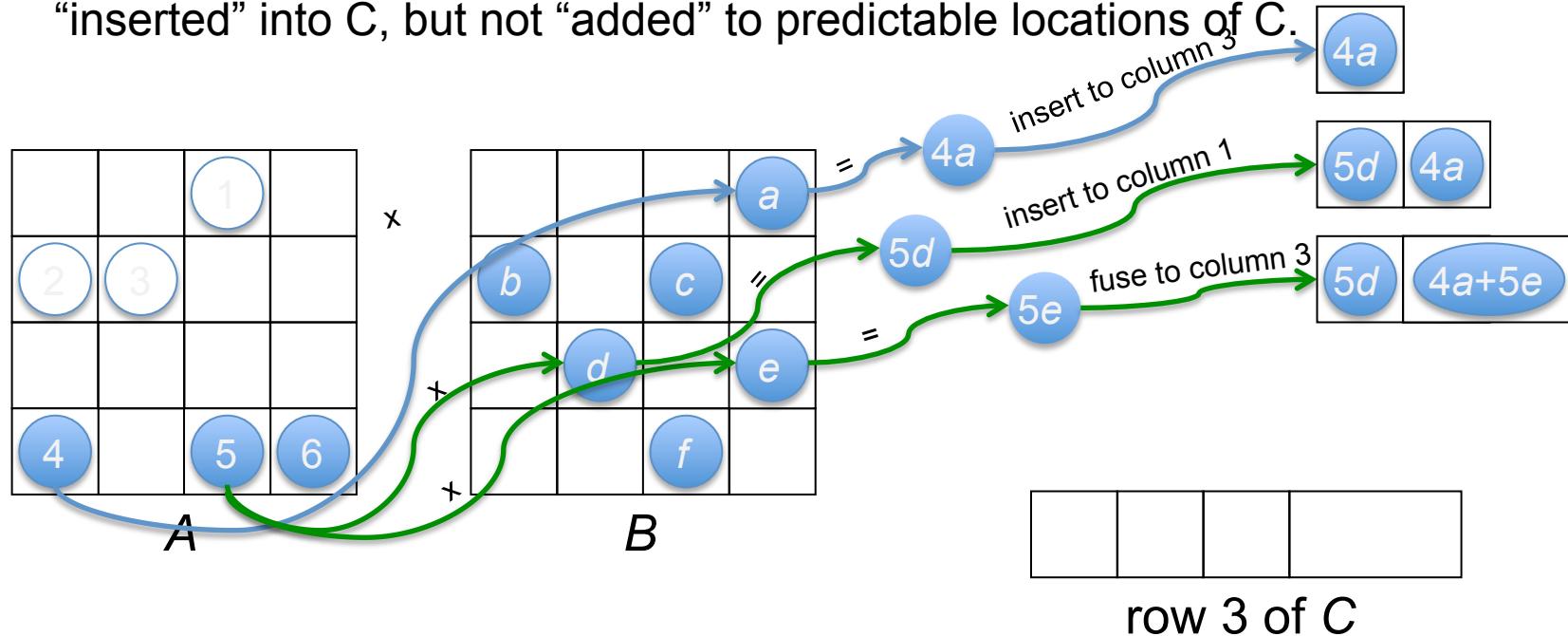
SpGEMM challenge 2 - parallel insertion

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



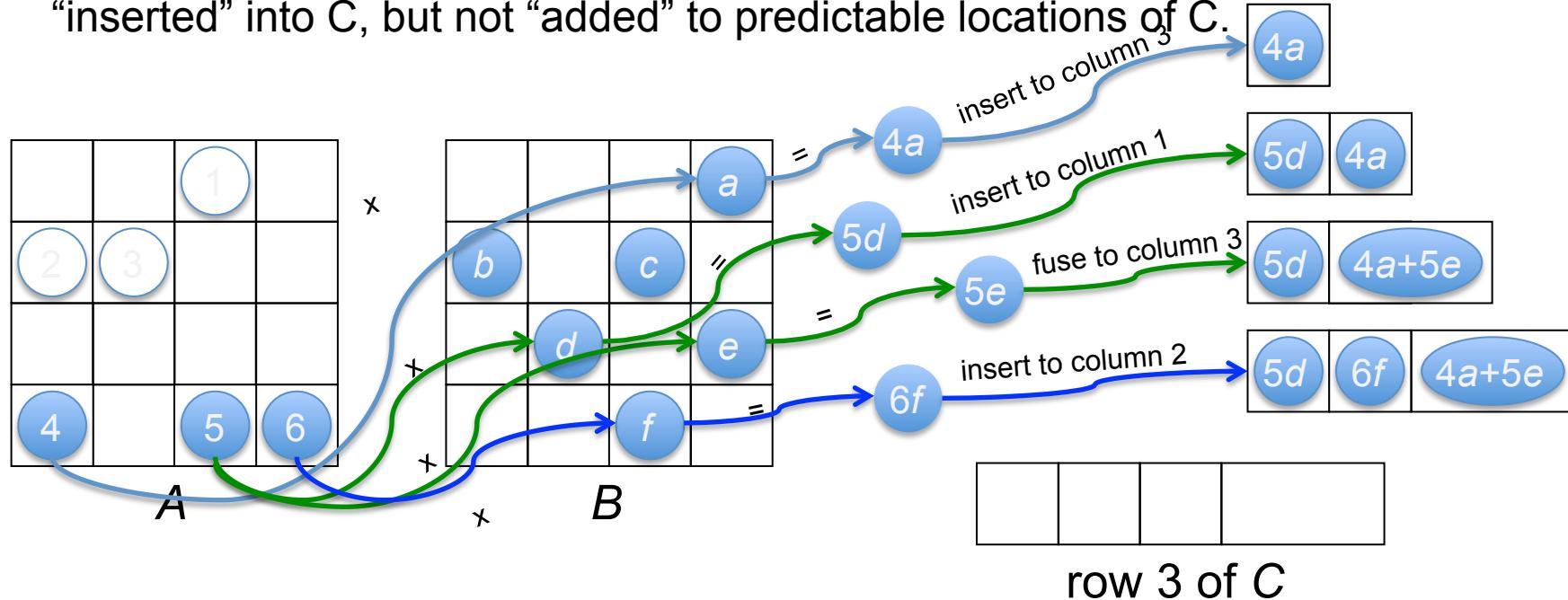
SpGEMM challenge 2 - parallel insertion

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



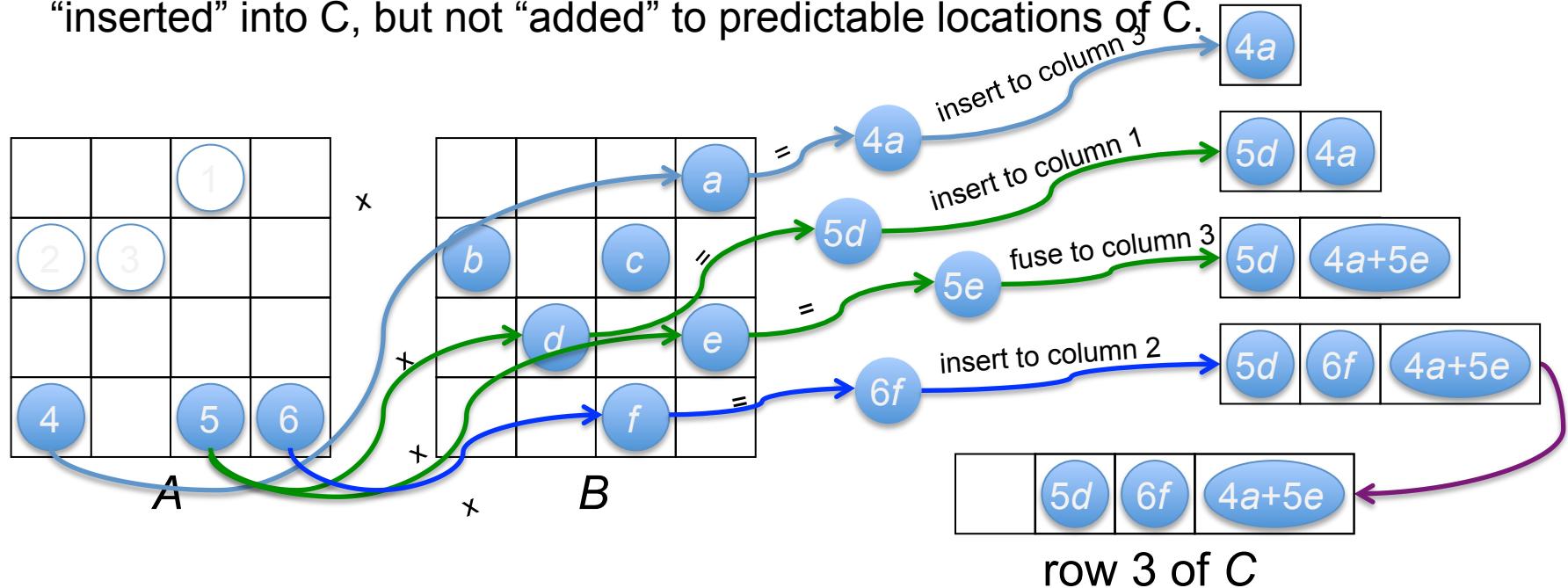
SpGEMM challenge 2 - parallel insertion

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



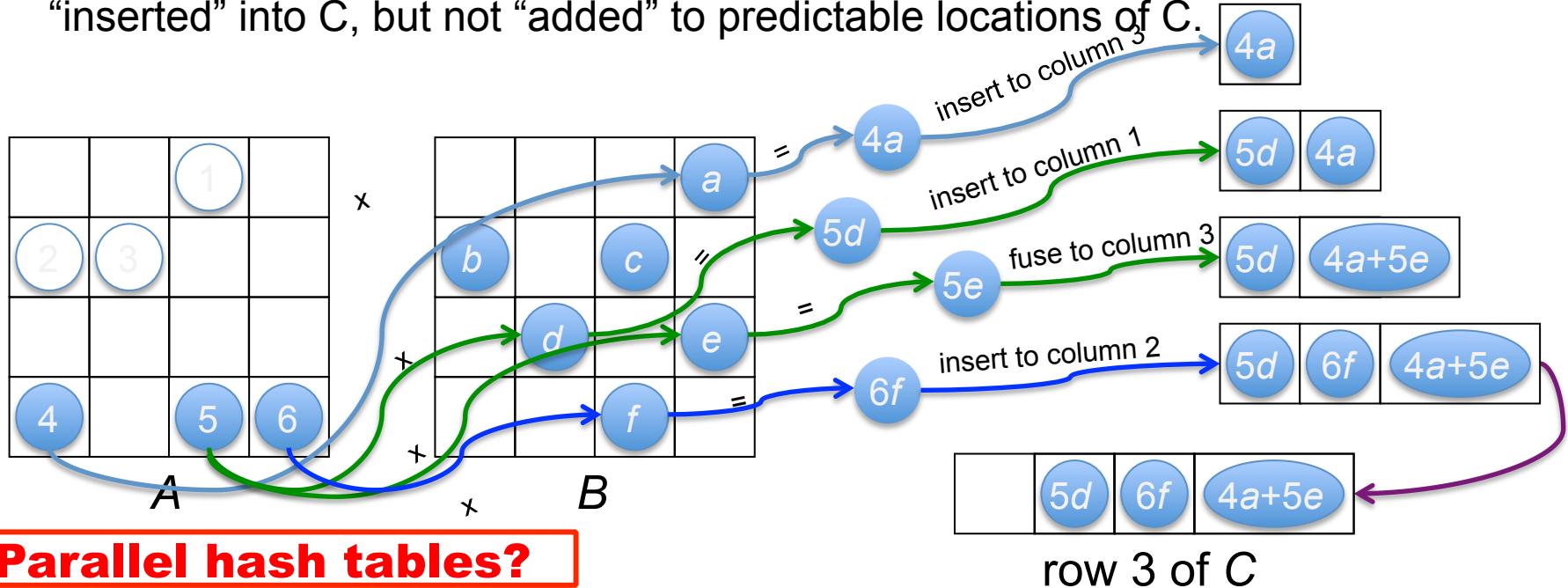
SpGEMM challenge 2 - parallel insertion

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



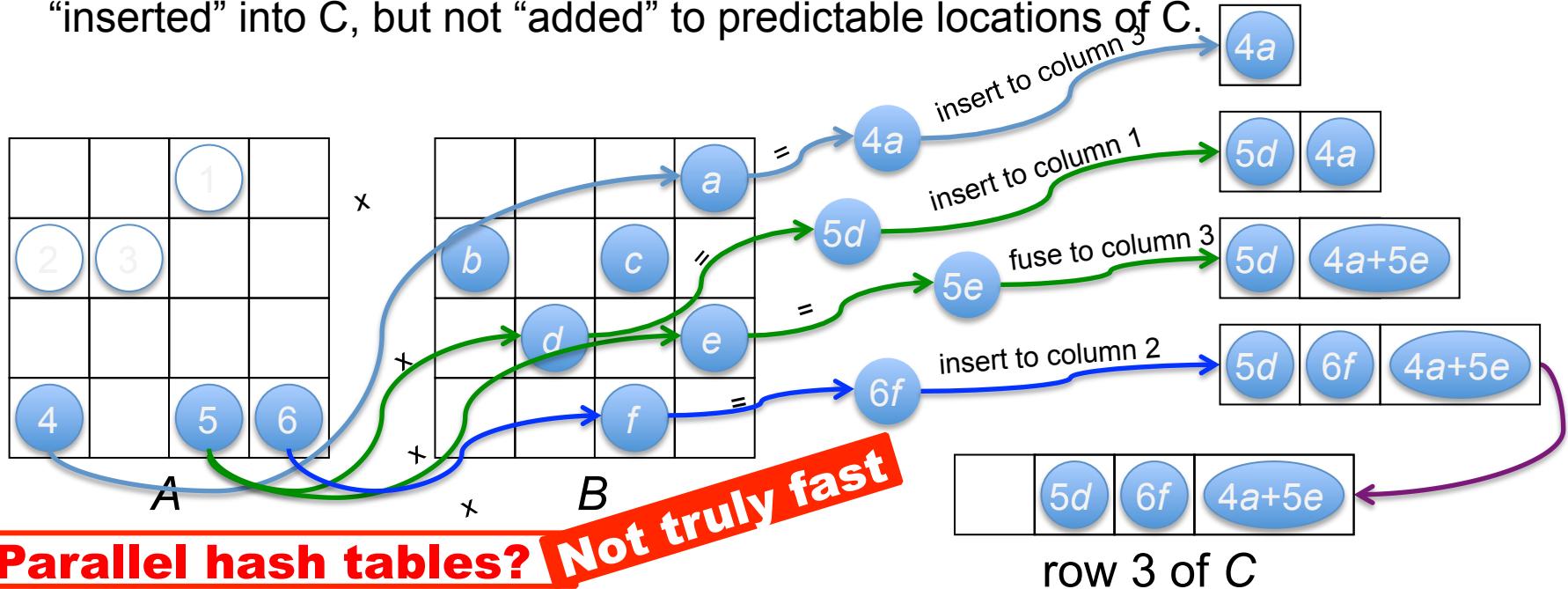
SpGEMM challenge 2 - parallel insertion

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

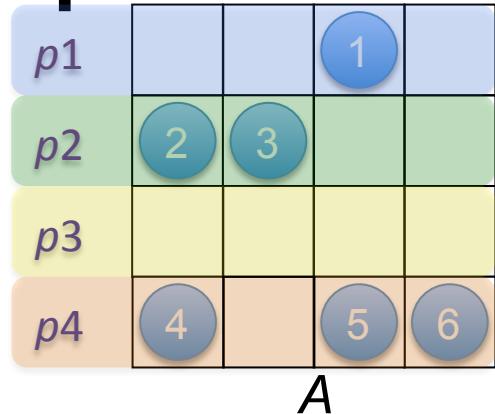


SpGEMM challenge 2 - parallel insertion

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

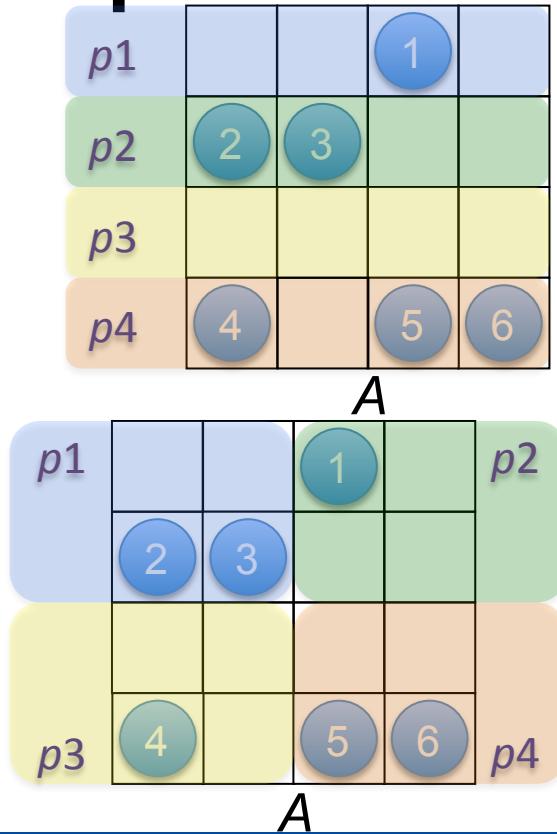


SpGEMM challenge 3 - load balancing

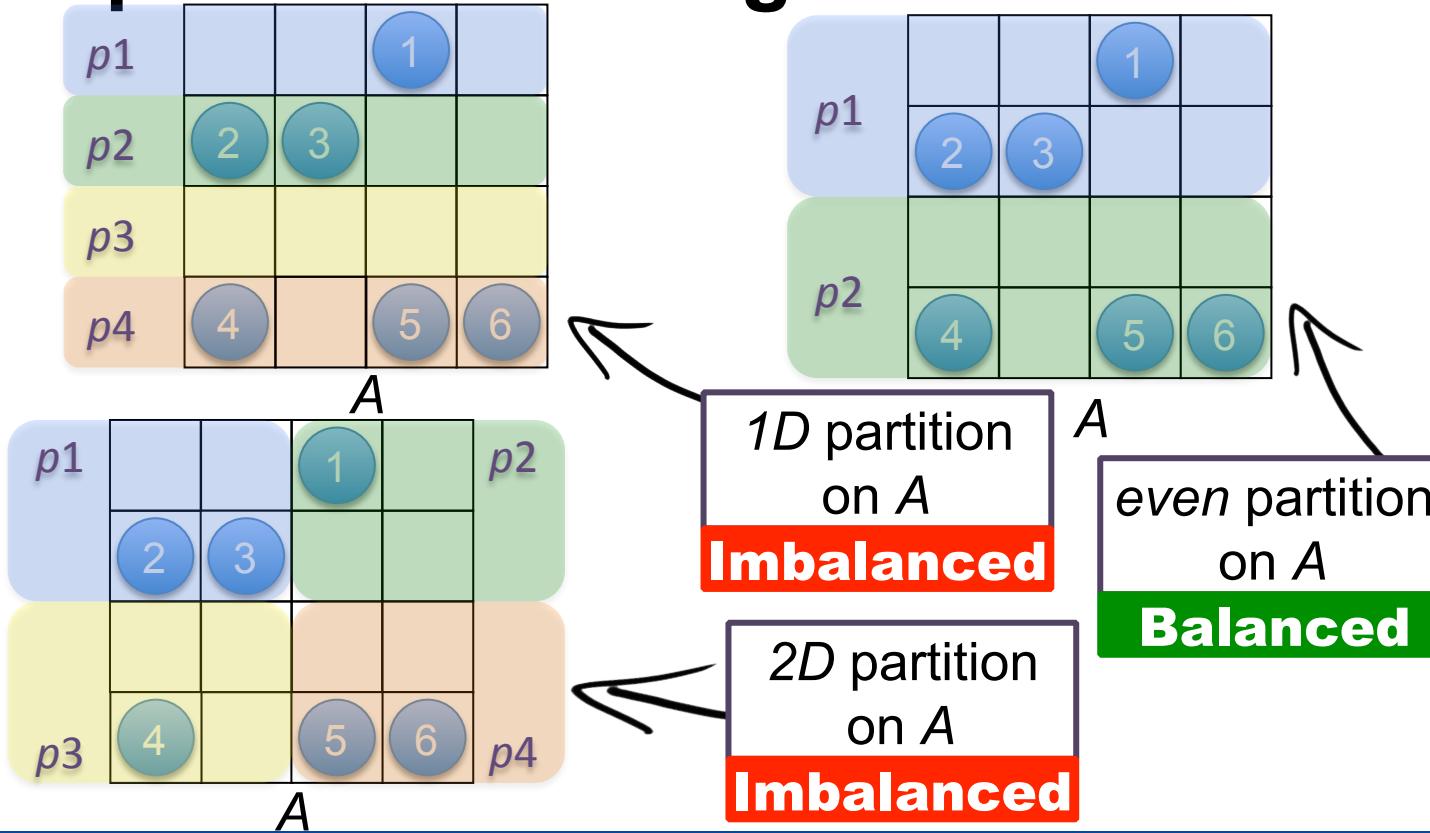


1D partition
on A
Imbalanced

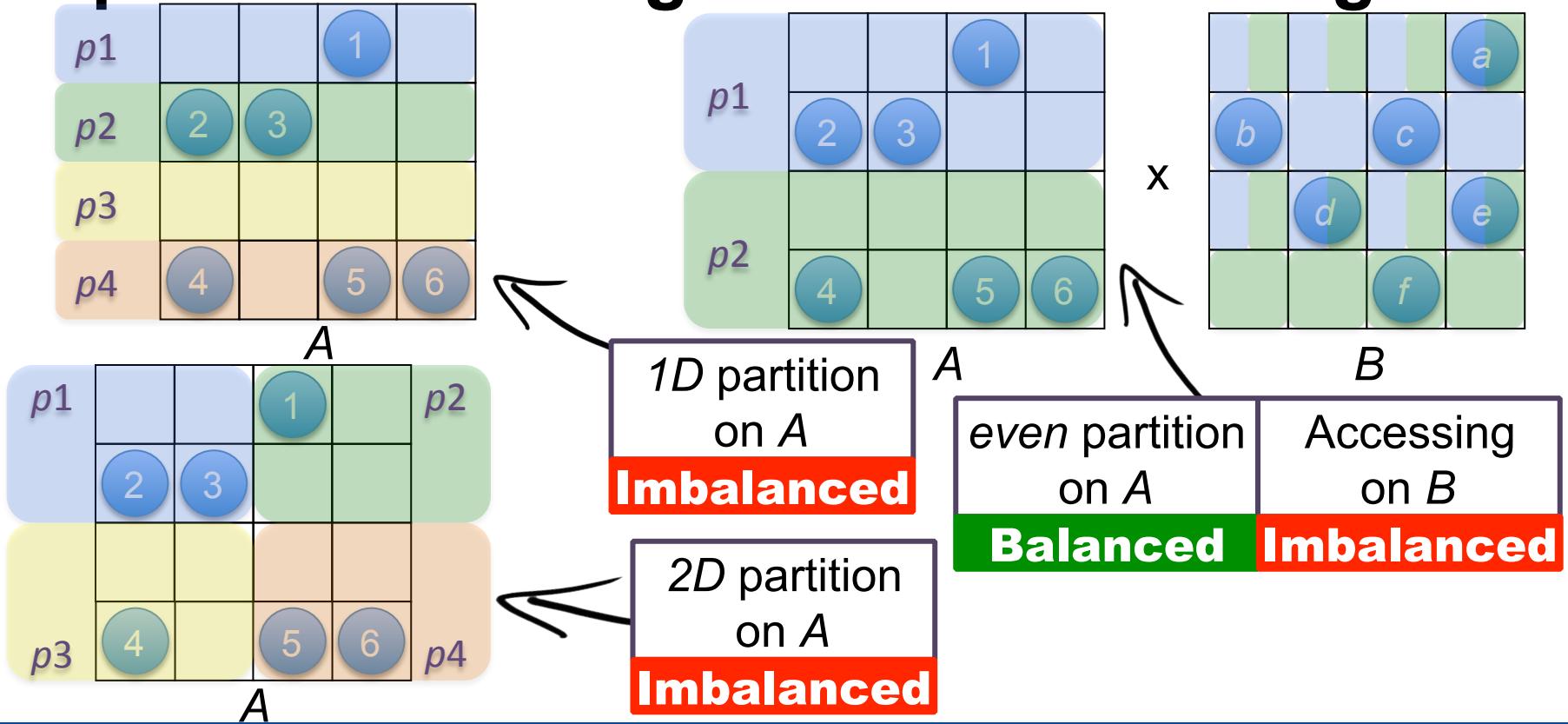
SpGEMM challenge 3 - load balancing



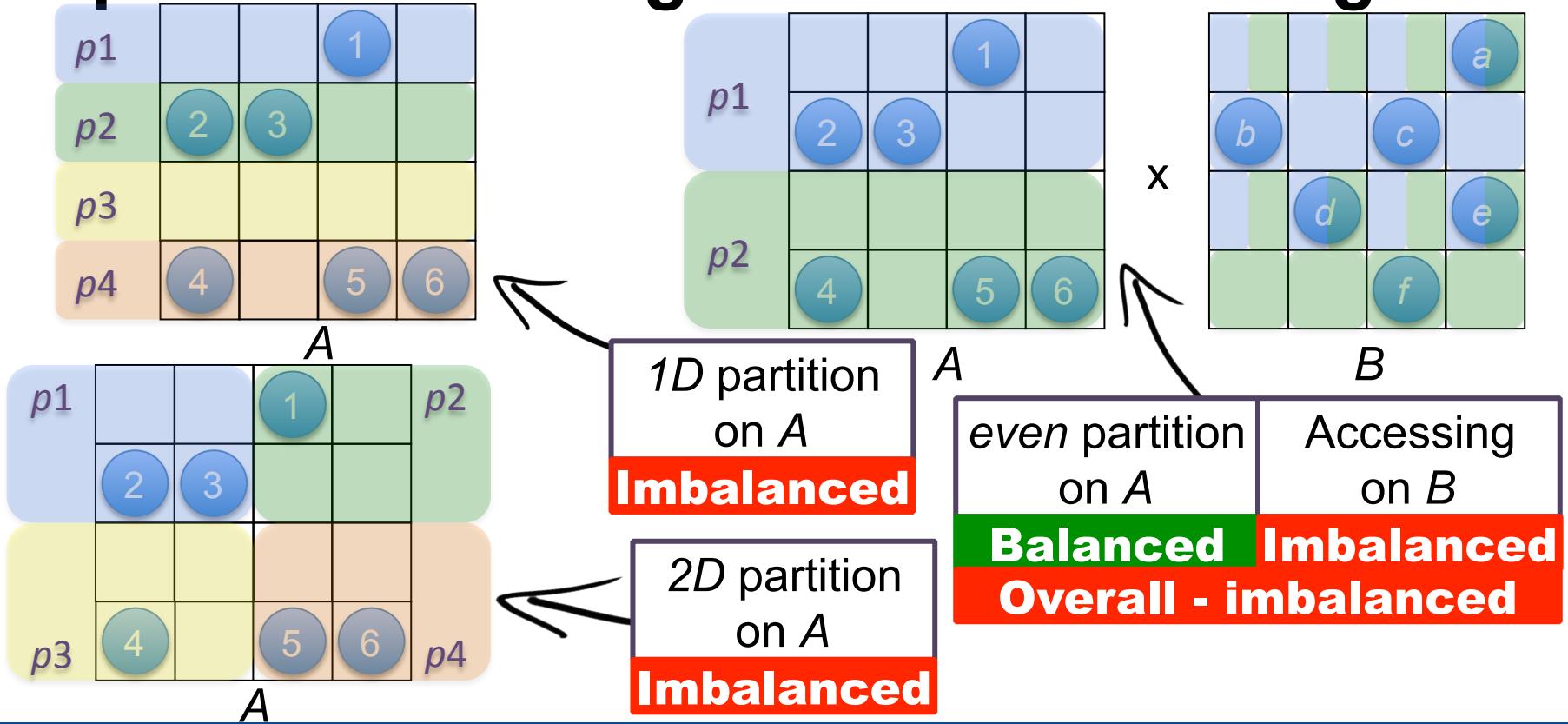
SpGEMM challenge 3 - load balancing



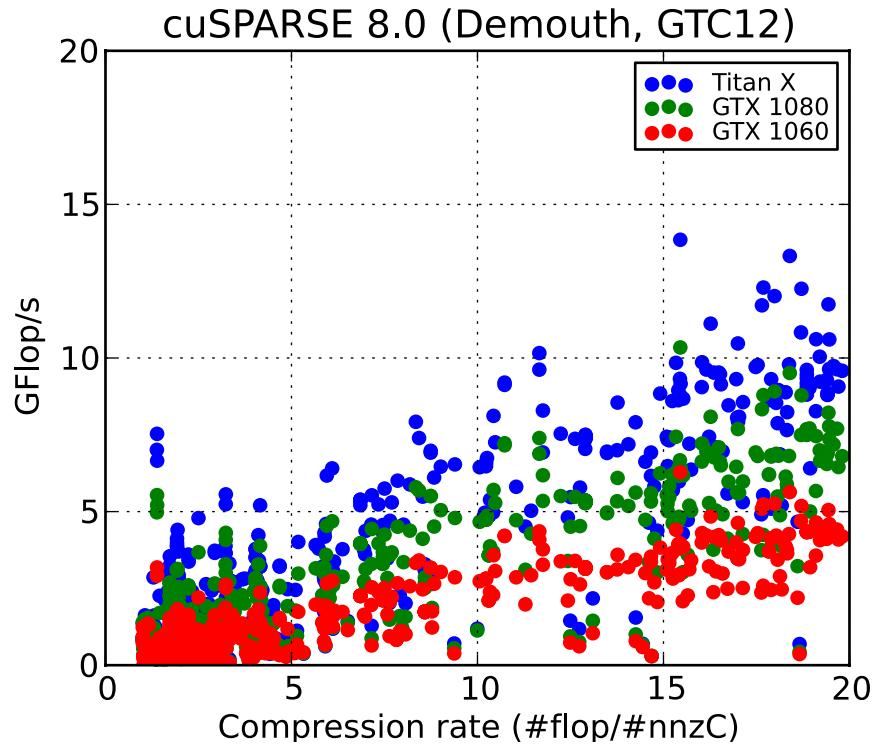
SpGEMM challenge 3 - load balancing



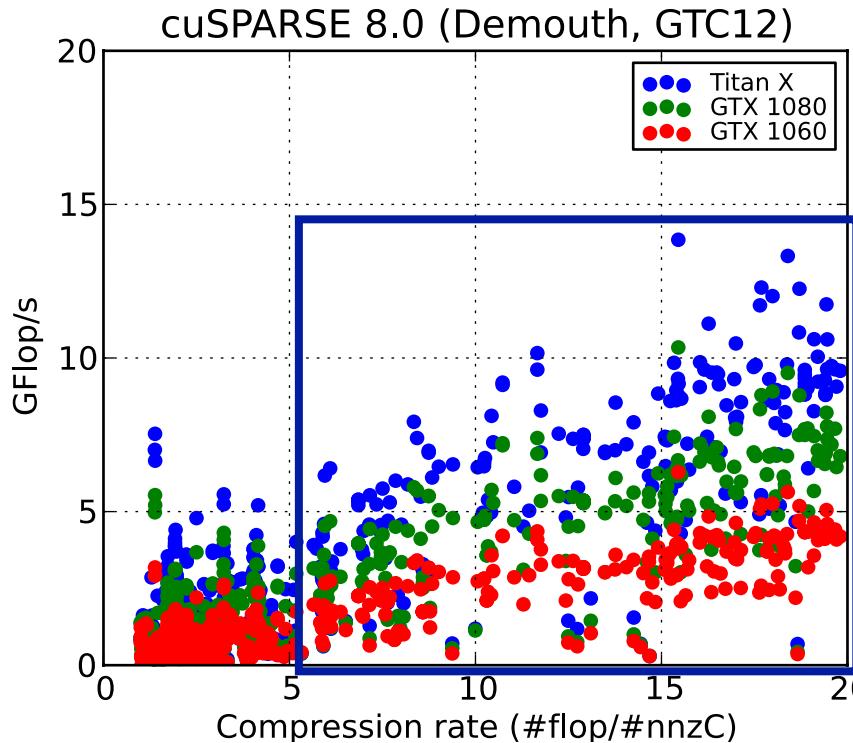
SpGEMM challenge 3 - load balancing



Scalability of SpGEMM in cuSPARSE



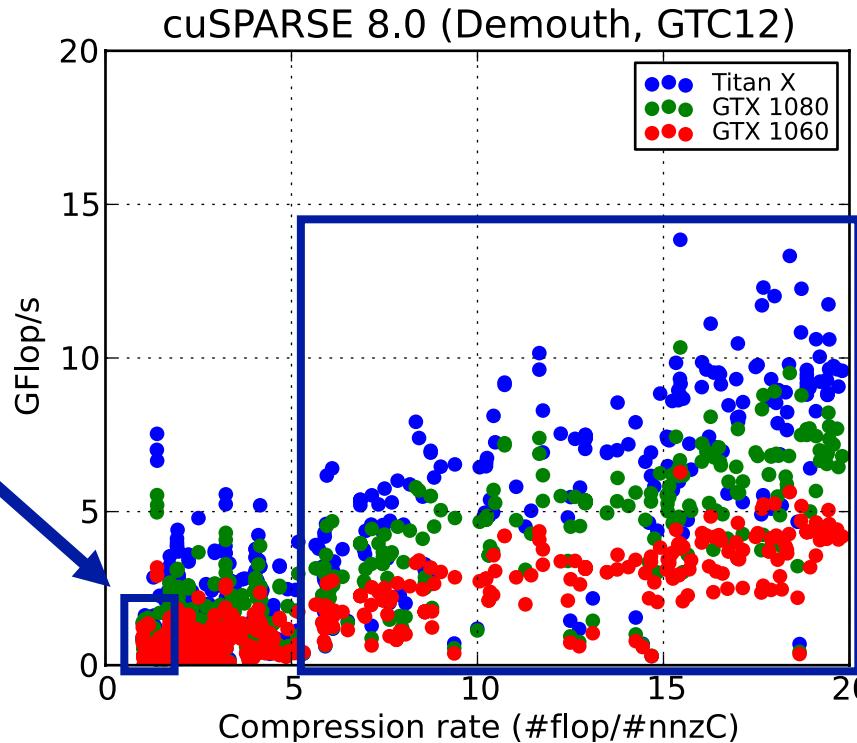
Scalability of SpGEMM in cuSPARSE



SpGEMM perf.
in general
scales with
hardware perf.

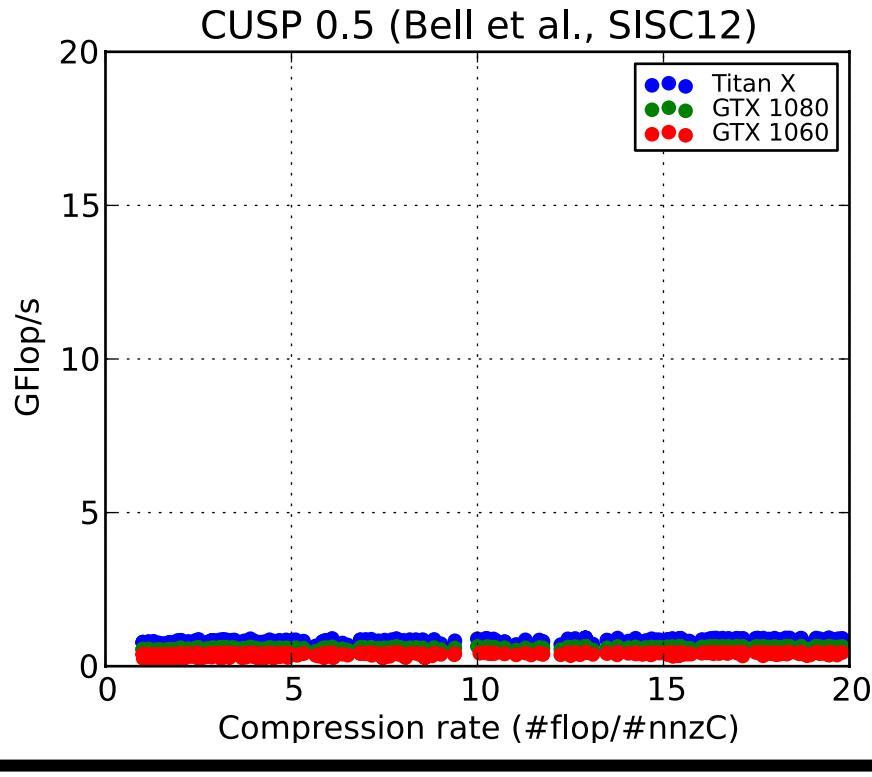
Scalability of SpGEMM in cuSPARSE

SpGEMM perf.
does not
scale with
hardware perf.



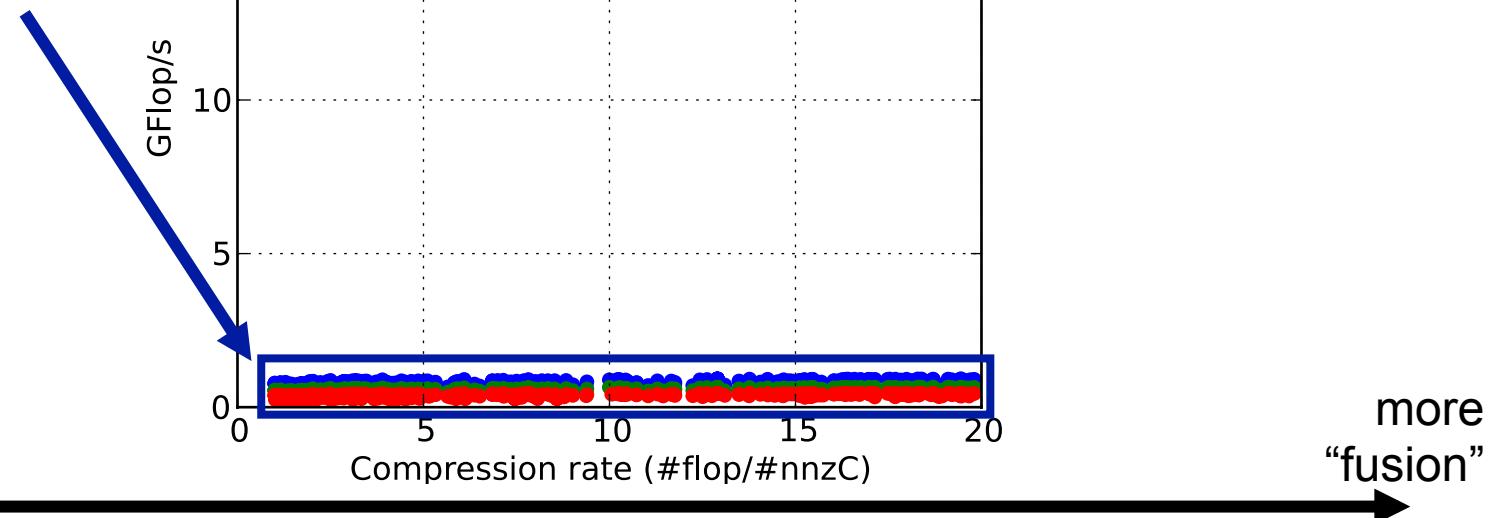
SpGEMM perf.
in general
scales with
hardware perf.

Scalability of SpGEMM in CUSP

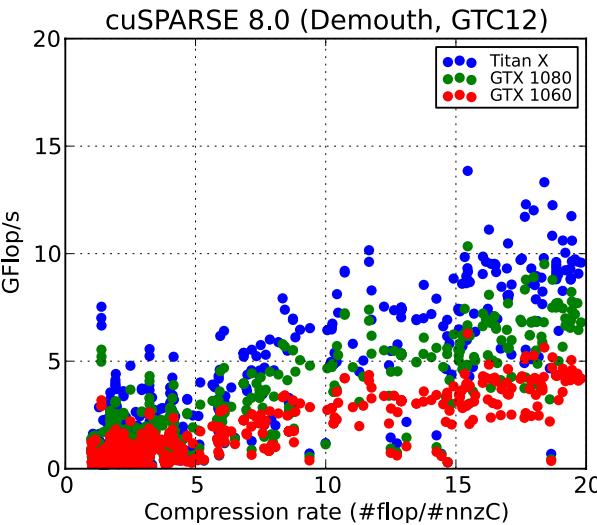


Scalability of SpGEMM in CUSP

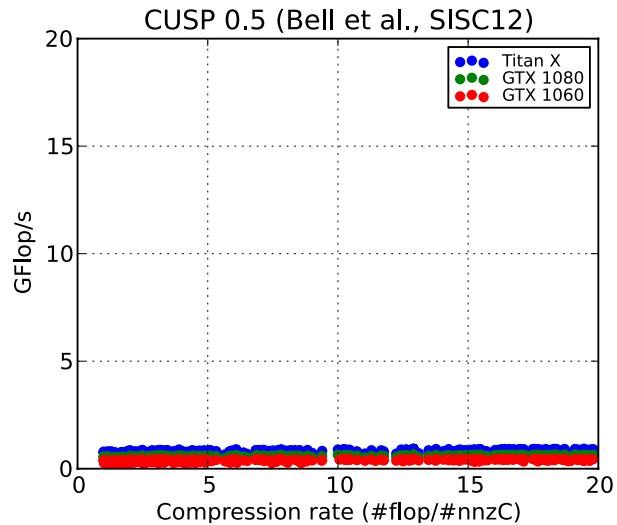
SpGEMM perf.
always
scales with
hardware perf.



Contradiction btw. scalability and high perf.

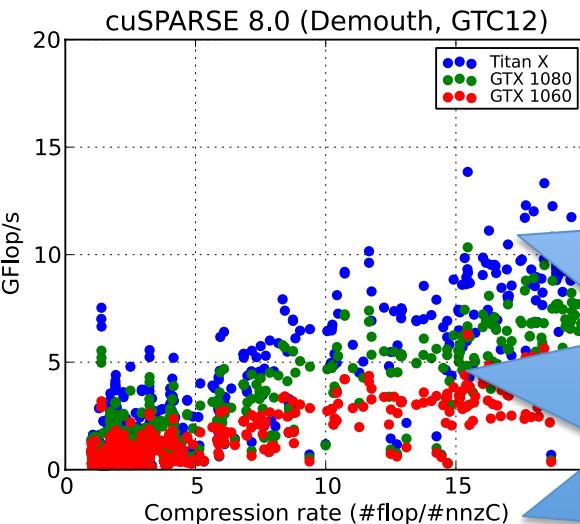


For regular matrices,
high performance
method cannot scale
or scale imperfectly.



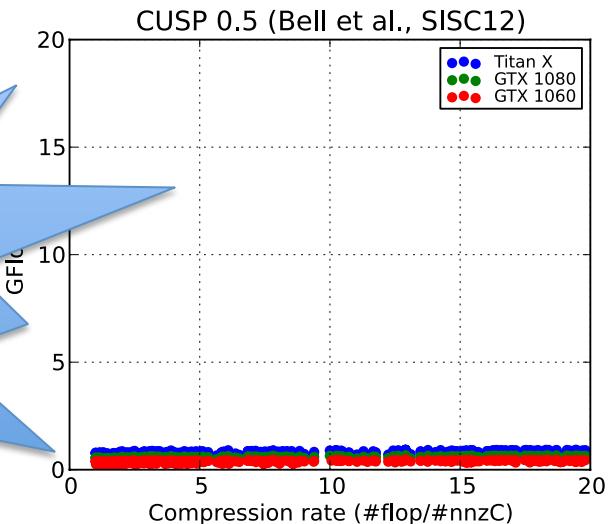
For both regular and
irregular matrices,
scalable method gives
low performance.

Contradiction btw. scalability and high perf.



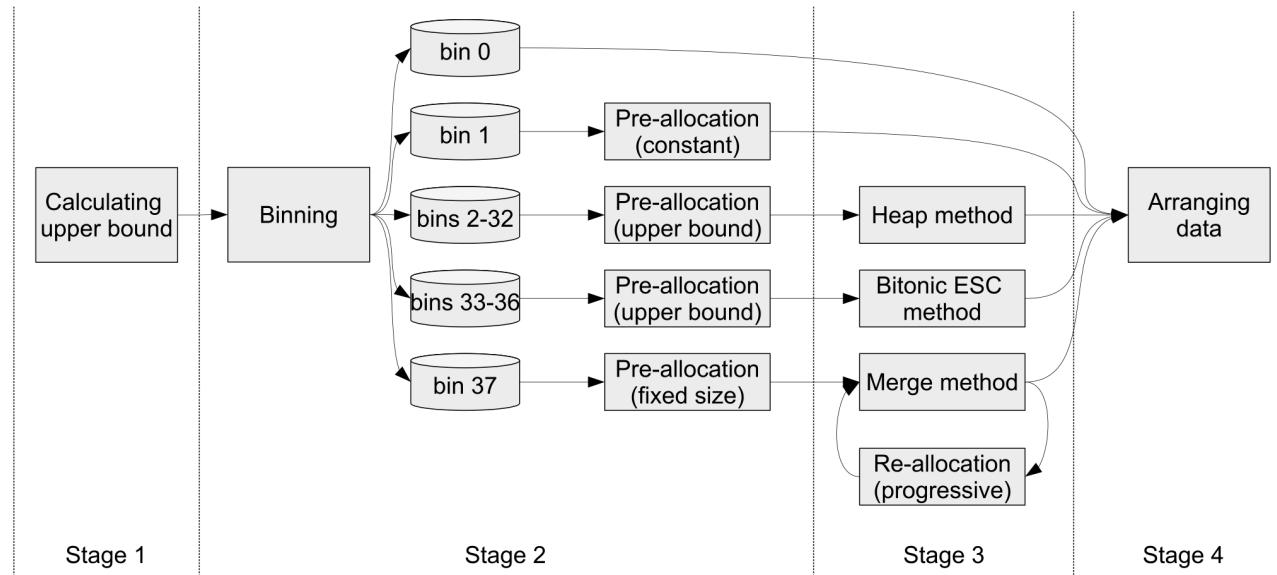
For regular matrices,
high performance
method cannot scale
or scale imperfectly.

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



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

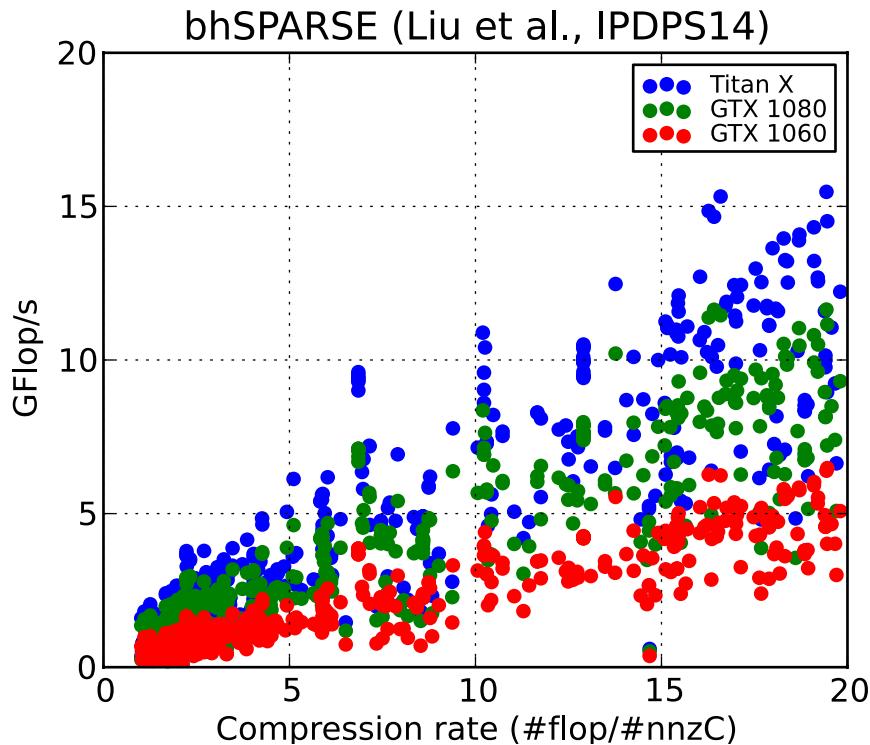
An SpGEMM framework (our work)



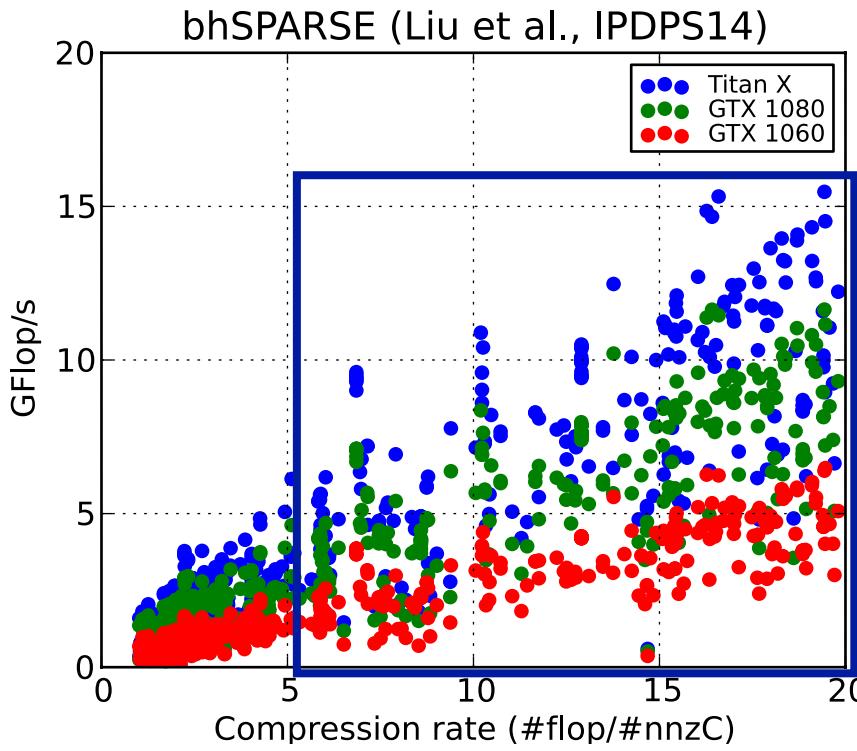
Weifeng Liu, Brian Vinter. **A Framework for General Sparse Matrix-Matrix Multiplication on GPUs and Heterogeneous Processors.** *JPDC*. 2015. (extended from *IPDPS14*)

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

Scalability of SpGEMM in bhSPARSE



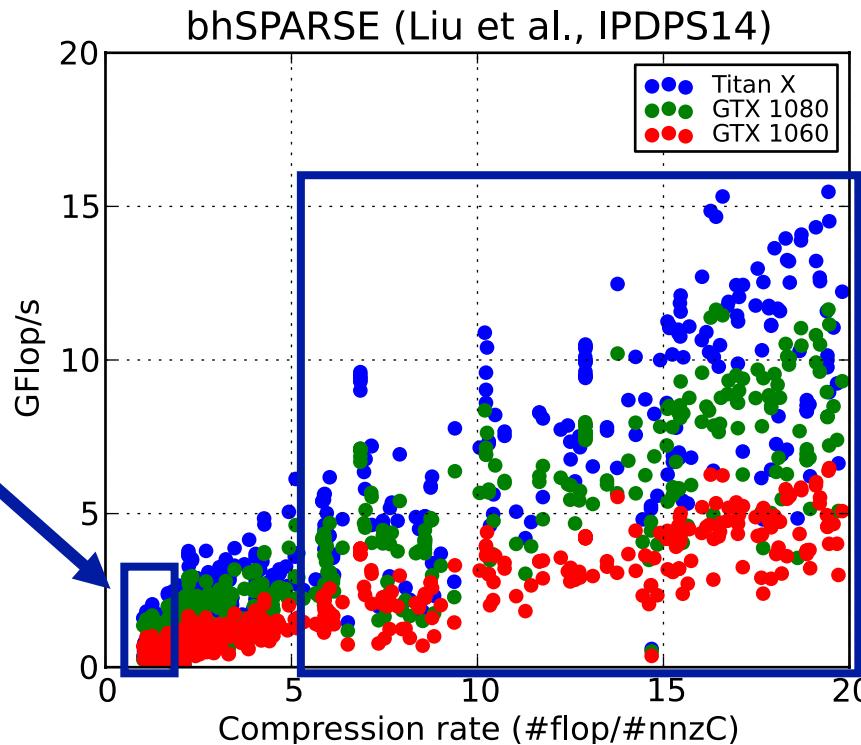
Scalability of SpGEMM in bhSPARSE



SpGEMM perf.
in general
scales with
hardware perf.

Scalability of SpGEMM in bhSPARSE

SpGEMM perf.
in general
scales with
hardware perf.



more
“insertion”

more
“fusion”

SpGEMM perf.
in general
scales with
hardware perf.

Source code of bhSPARSE on Github

The screenshot shows the GitHub repository page for 'bhSPARSE / Benchmark_SpGEMM_using_CSR'. The repository has 1 star, 5 forks, and 33 commits. It contains 1 branch and 0 releases. There is 1 contributor. The license is MIT. The latest commit was on April 9, 2016. The repository description is 'CSR-based SpGEMM on nVidia and AMD GPUs'. The commit history includes:

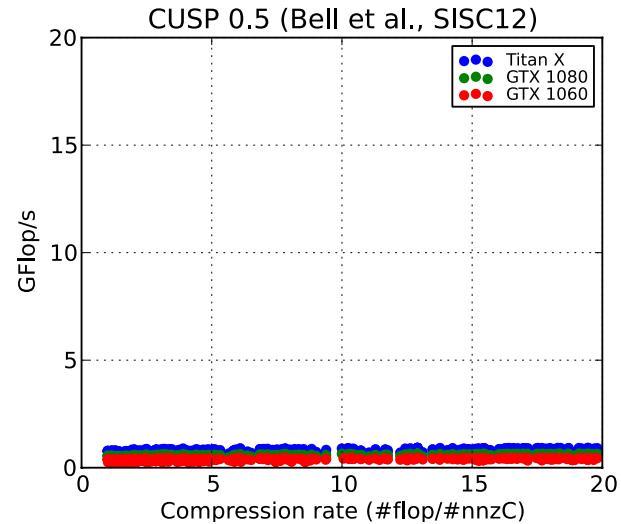
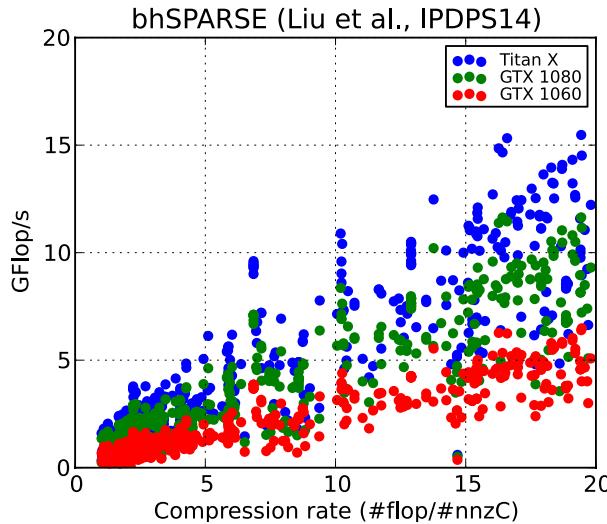
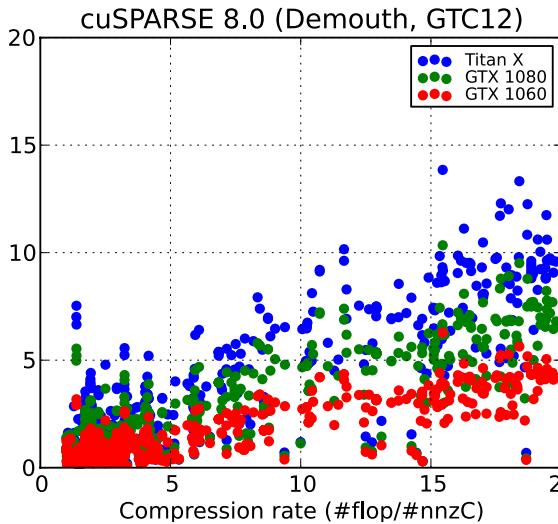
File	Commit Message	Date
SpGEMM_cuda	update.	a year ago
SpGEMM_opencl	Update Makefile	2 years ago
LICENSE	Initial commit	3 years ago
README.md	Update.	a year ago

https://github.com/bhSPARSE/Benchmark_SpGEMM_using_CSR

Has been integrated into AMD's clSPARSE library.
<https://github.com/clMathLibraries/clSPARSE>



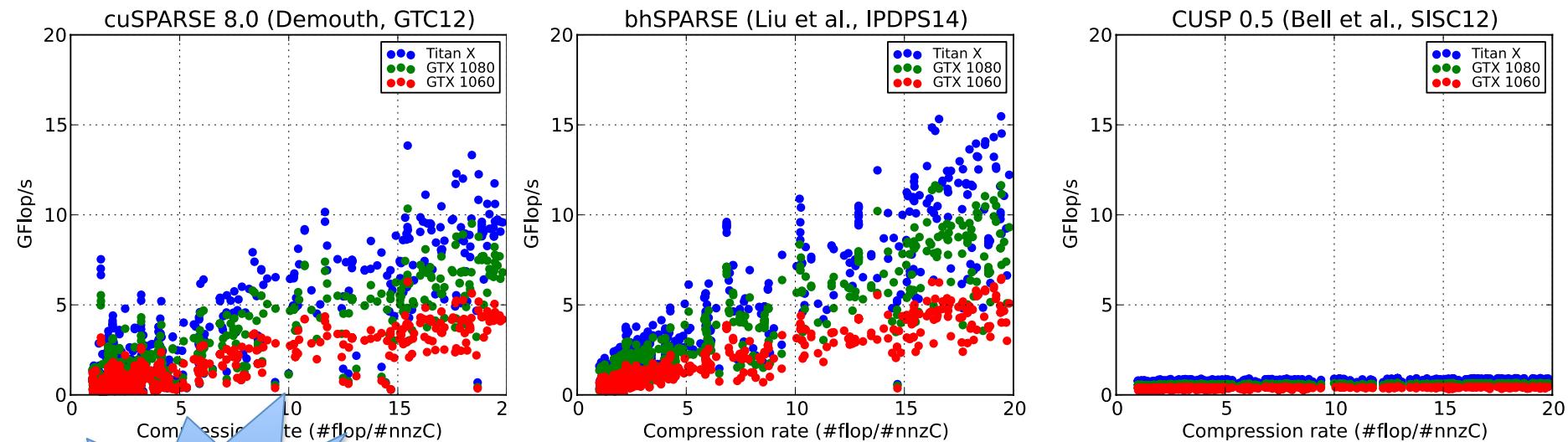
Contradiction btw. scalability and high perf.



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

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

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.

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.

row pointer =

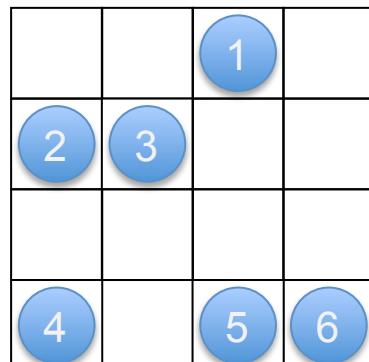
0	1	3	3	6
---	---	---	---	---

column index =

2	0	1	0	2	3
---	---	---	---	---	---

value =

1	2	3	4	5	6
---	---	---	---	---	---

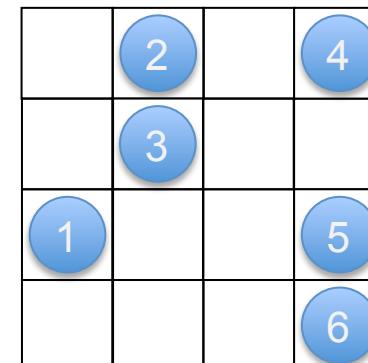


A

(4x4)

sparse, 6 nonzeros

->



A^T

(4x4)

row pointer =

0	2	3	5	6
---	---	---	---	---

column index =

1	3	1	0	3	3
---	---	---	---	---	---

value =

2	4	3	1	5	6
---	---	---	---	---	---

ScanTrans and MergeTrans (our work)

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

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

core 0 core 1

0	0	0	0
core 0	1	1	1
core 1	1	0	1

0	0	0	0
1	1	1	0
2	1	2	1

core 0 core 1

0	2	3	5	6
---	---	---	---	---

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

0	1	2	3	3
1	1	0		
2	3	1		

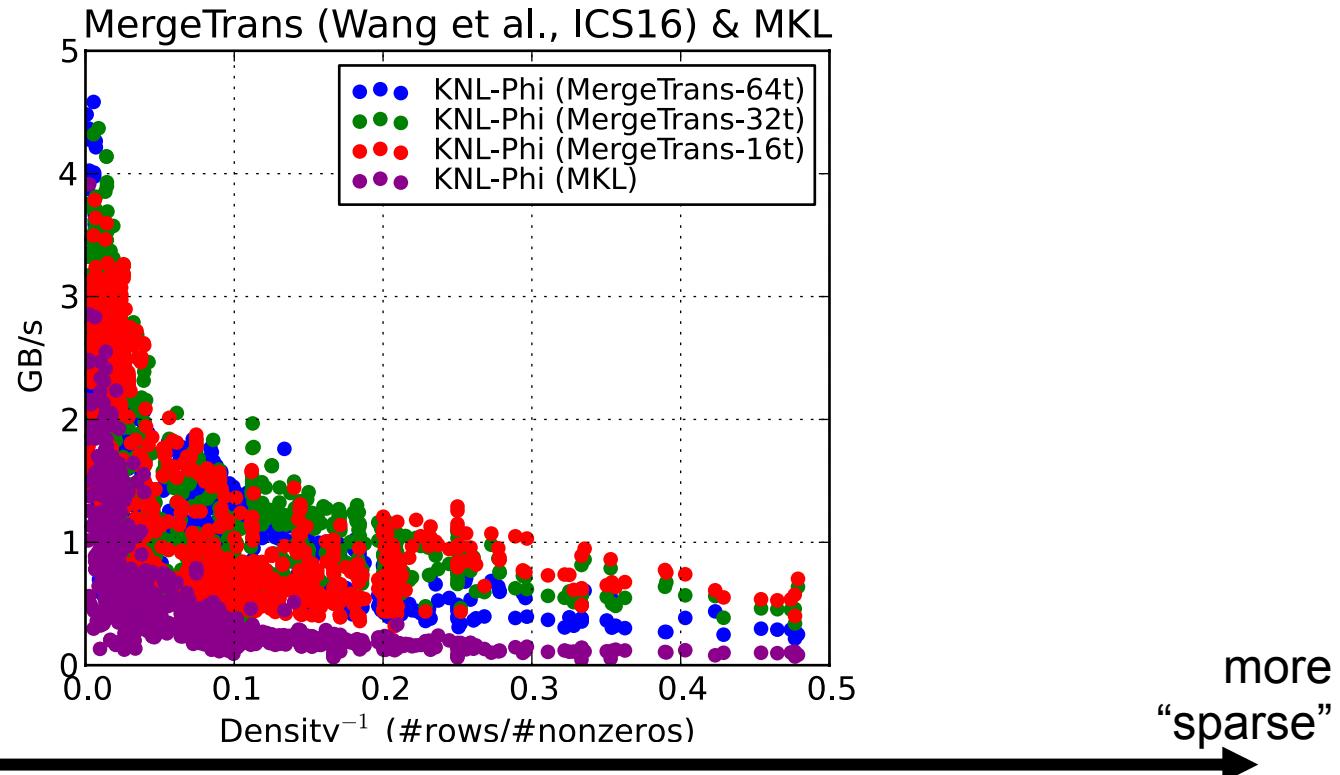
core 0

0	1	1	2	3
3	3	3		
4	5	6		

core 1

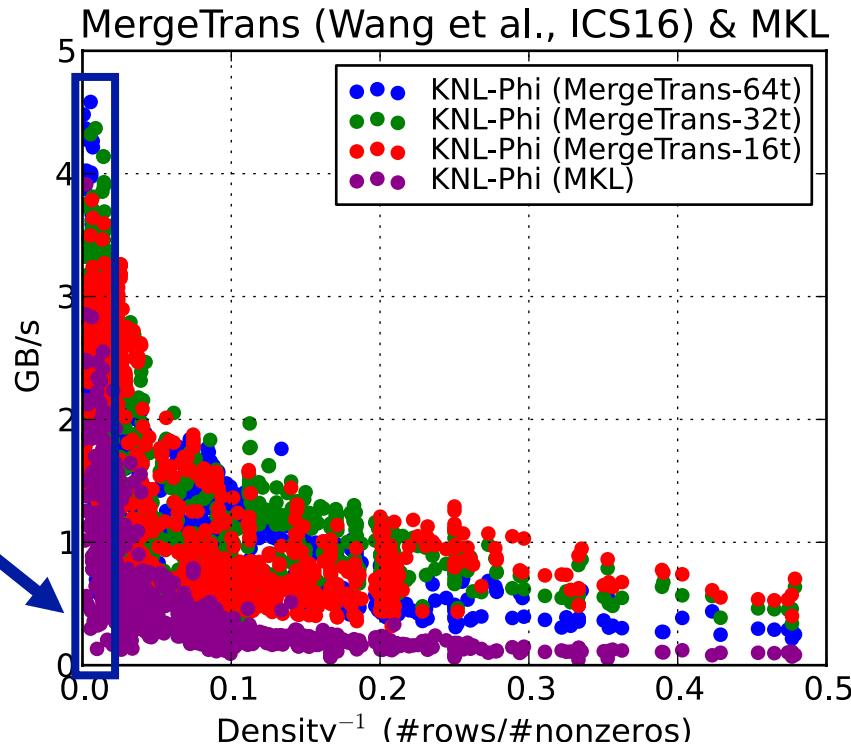
Hao Wang, Weifeng Liu, Kaixi Hou, Wu-chun Feng. **Parallel Transposition of Sparse Data Structures.** ICS16.

Scalability of MergeTrans on KNL



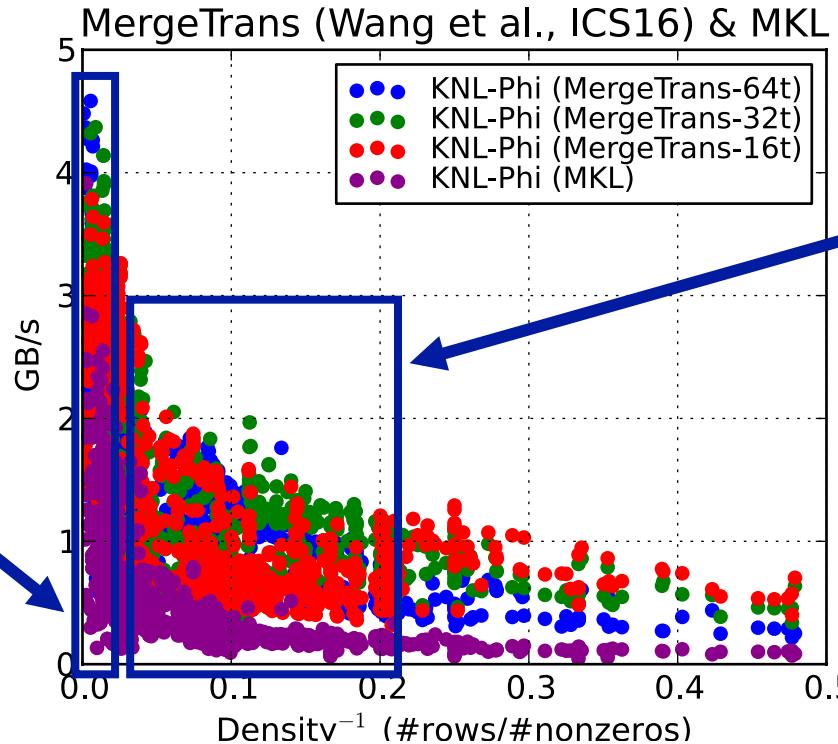
Scalability of MergeTrans on KNL

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



Scalability of MergeTrans on KNL

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



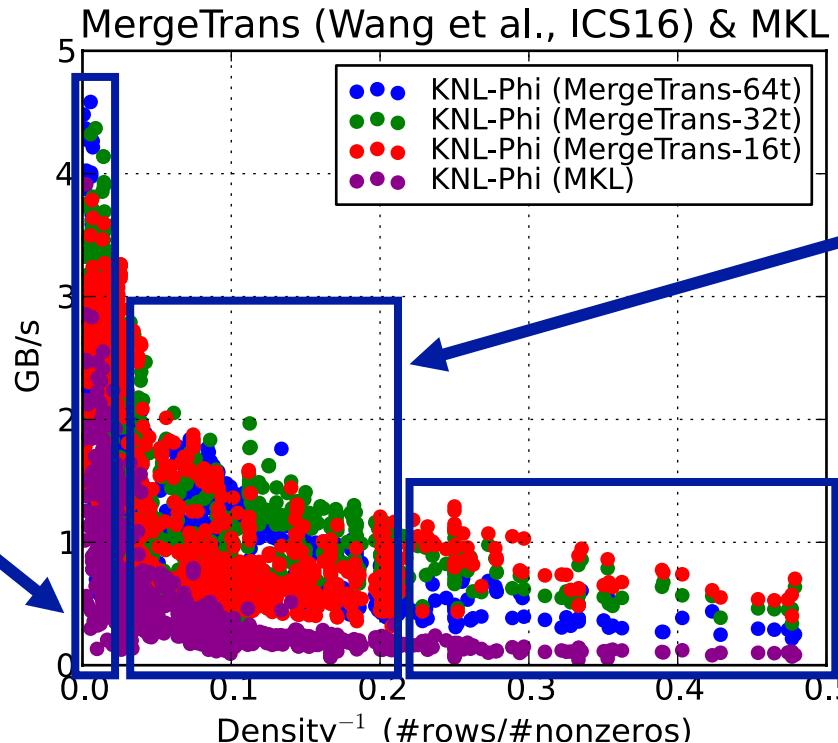
more
“dense”

more
“sparse”

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

Scalability of MergeTrans on KNL

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



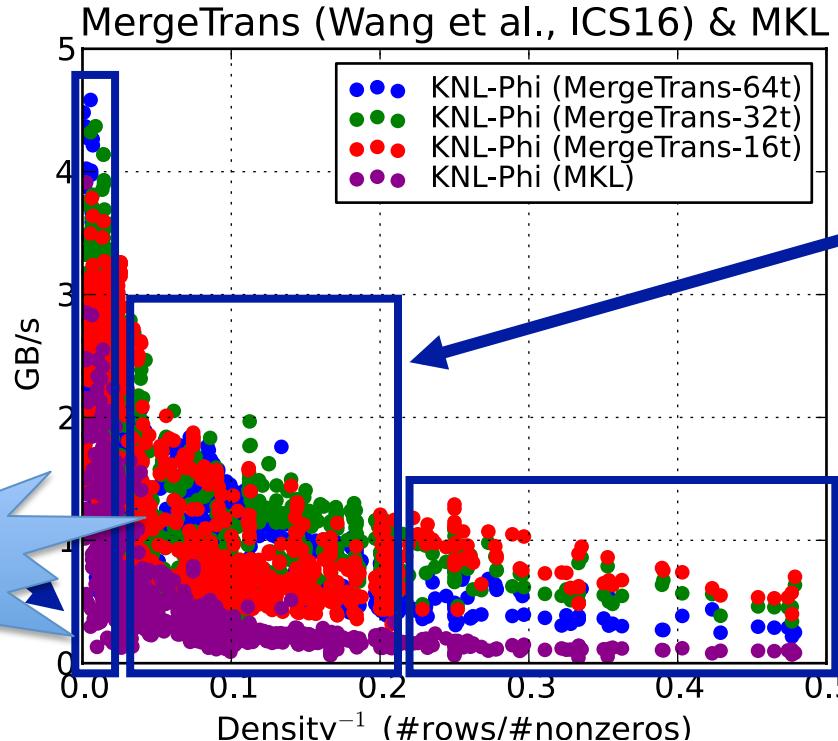
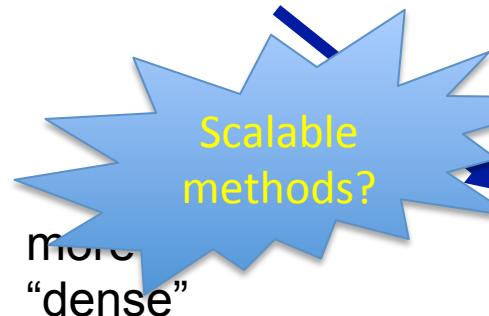
more
“dense”

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

SpTRANS perf.
inversely
scales with
hardware perf.
(16t is the best)
more
“sparse”

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
"sparse"

Source code of Scan/MergeTrans on Github

The screenshot shows the GitHub repository page for `vtsynergy / sptrans`. The page includes a navigation bar with links for This repository, Search, Pull requests, Issues, Marketplace, and Explore. Below the navigation bar, there are buttons for Watch (4), Star (0), and Fork (0). The repository has 6 commits, 1 branch, 0 releases, and 1 contributor. The license is listed as LGPL-2.1. A dropdown menu shows the current branch is master, and there is a link to New pull request. Below the summary, a list of files shows they were all uploaded by Kaixi Hou, with the latest commit being 73ed57d 27 days ago. The files listed are LICENSE, Makefile, README.md, main.cpp, matio.h, mmio.h, and sptrans.h.

File	Commit Message	Time Ago
LICENSE	Init upload	27 days ago
Makefile	Init upload	27 days ago
README.md	Update README.md	27 days ago
main.cpp	Init upload	27 days ago
matio.h	Init upload	27 days ago
mmio.h	Init upload	27 days ago
sptrans.h	Init upload	27 days ago

<https://github.com/vtsynergy/sptrans>

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.

$$L \quad (4 \times 4)$$

$\begin{matrix} 1 & & & \\ & 1 & & \\ & & 2 & 1 \\ 3 & & & 1 \end{matrix}$

$nnzL = 6$
known

$$\begin{array}{c} x \\ = \\ b \end{array}$$

$x \quad (4 \times 1) \quad \text{dense}$
 $b \quad (4 \times 1) \quad \text{dense}$
 $a \quad (4 \times 1) \quad \text{known}$

system:

$$\begin{cases} 1*x_0 = a \\ 1*x_1 = b \\ 2*x_1 + 1*x_2 = c \\ 3*x_0 + 1*x_3 = d \end{cases}$$

solution:

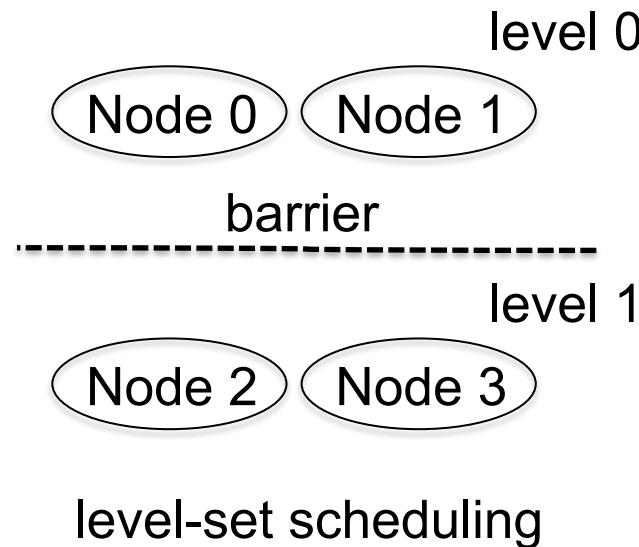
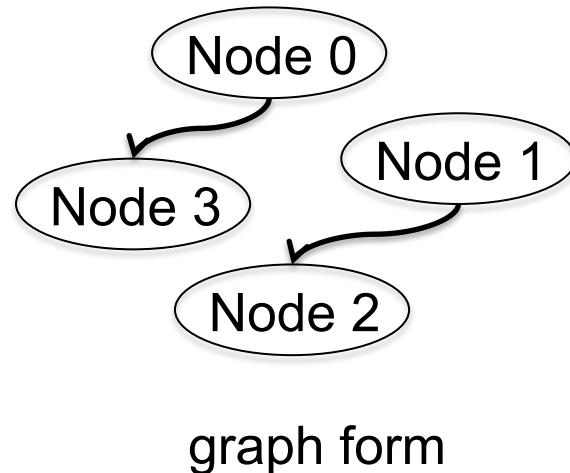
$$\begin{cases} x_0 = a \\ x_1 = b \\ x_2 = c - 2b \\ x_3 = d - 3a \end{cases}$$

Level-set method for SpTRSV

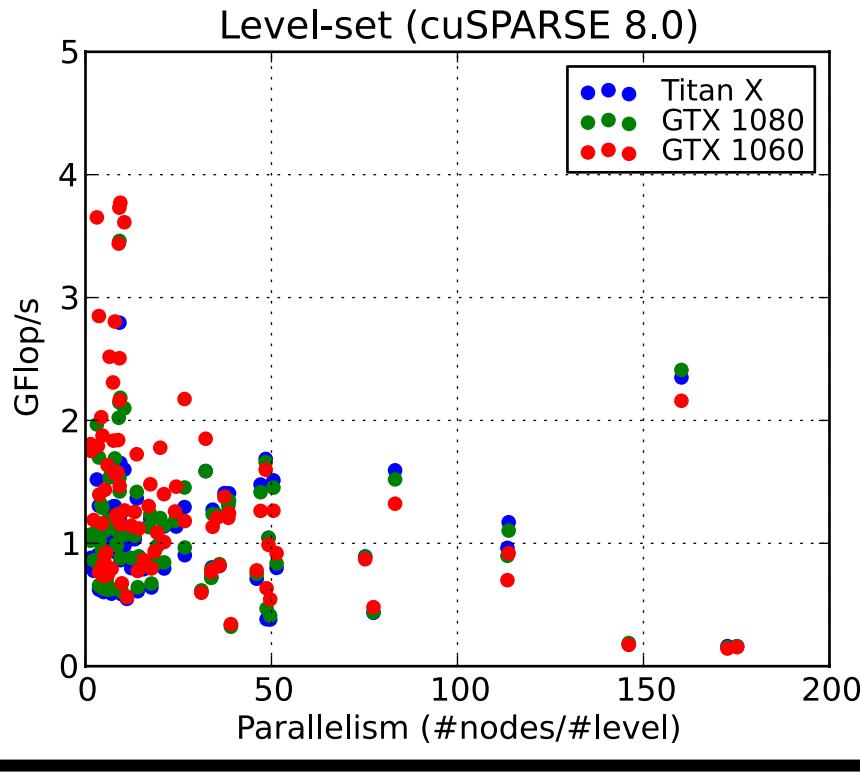
- The level-set method has two phases: (1) group nodes (x_0-x_{n-1}) that can be consumed in parallel, and (2) solve nodes group by group with barriers between.

1			
	1		
	2	1	
3			1

matrix form

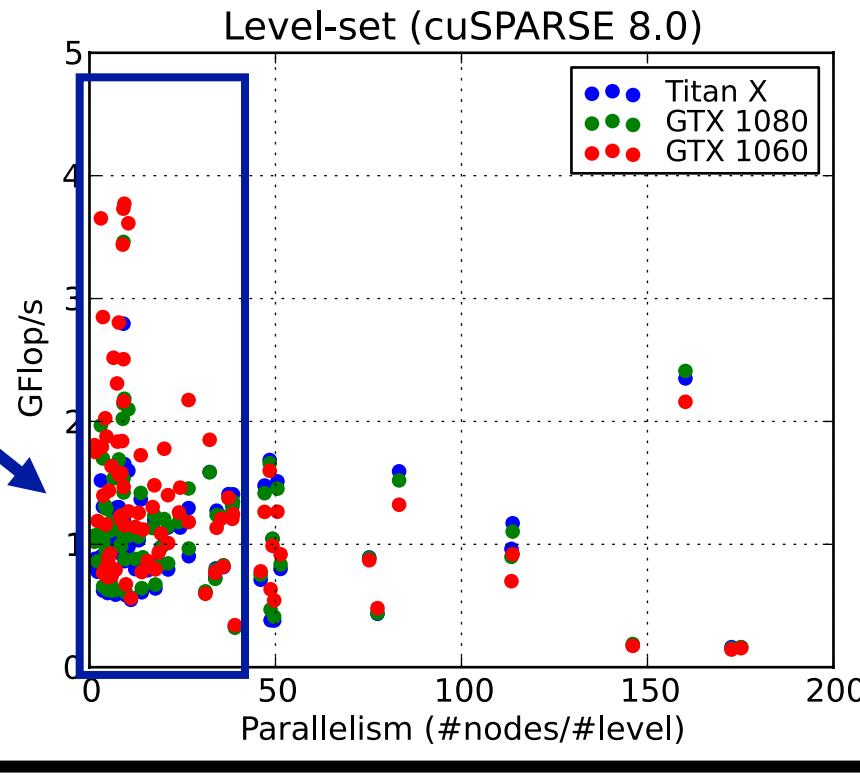


Scalability of level-set method for SpTRSV



Scalability of level-set method for SpTRSV

SpTRSV perf.
inversely
scales with
hardware perf.



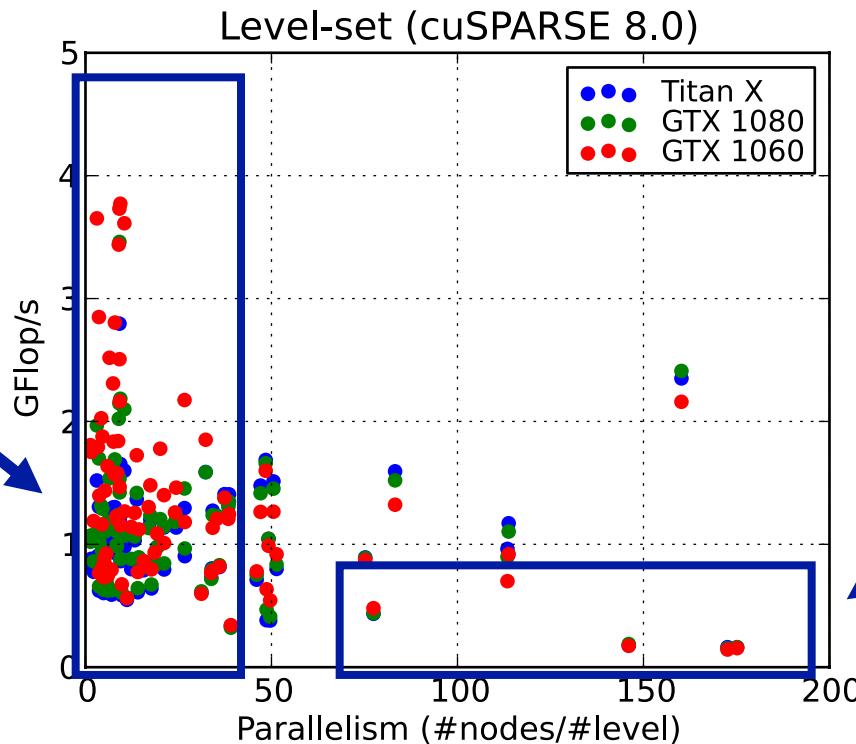
Scalability of level-set method for SpTRSV

SpTRSV perf.
inversely
scales with
hardware perf.

SpTRSV perf.
does not
scale with
hardware perf.

more
“levels”
less
“nodes per level”

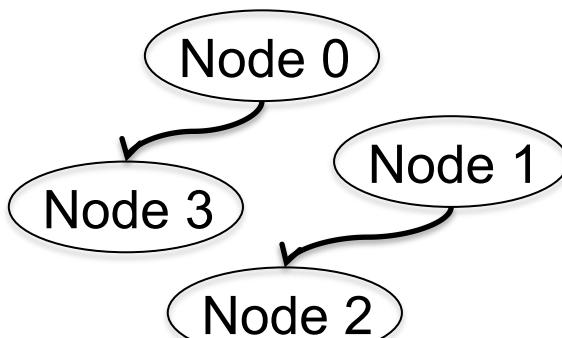
more
“nodes per level”
less
“levels”



Sync-free method for SpTRSV (our work)

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

1			
	1		
	2	1	
3			1



CSR row_ptr:

0	1	2	4	6
0	2	4	5	6

CSC column_ptr:

0	2	4	5	6
0	2	4	5	6

In-degree:

1	1	2	2
2	2	1	1

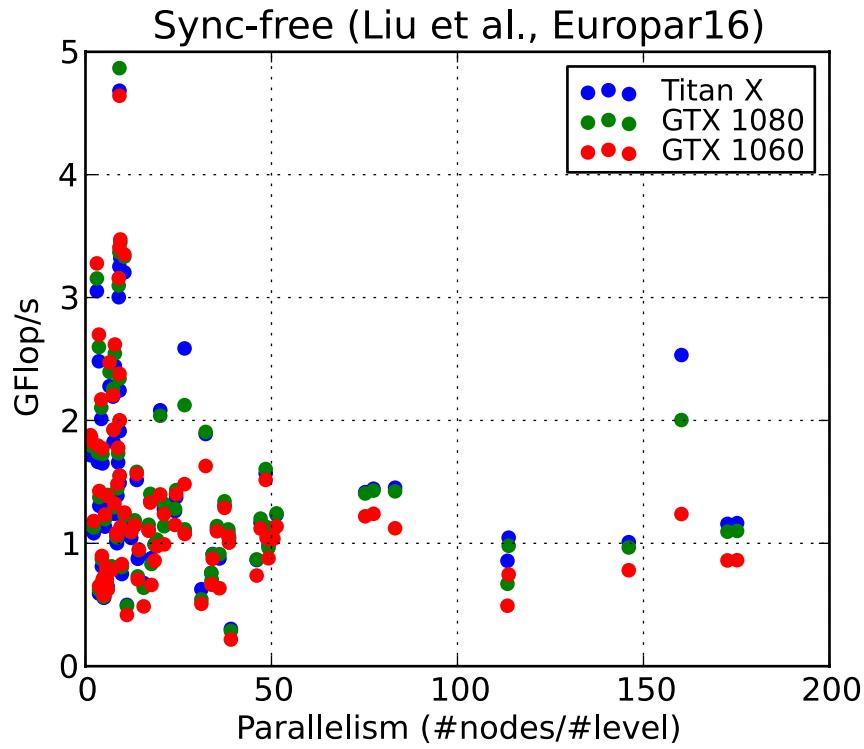
Out-degree:

2	2	1	1
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

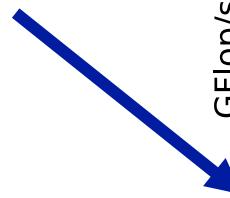


more
“levels”
less
“nodes per level”

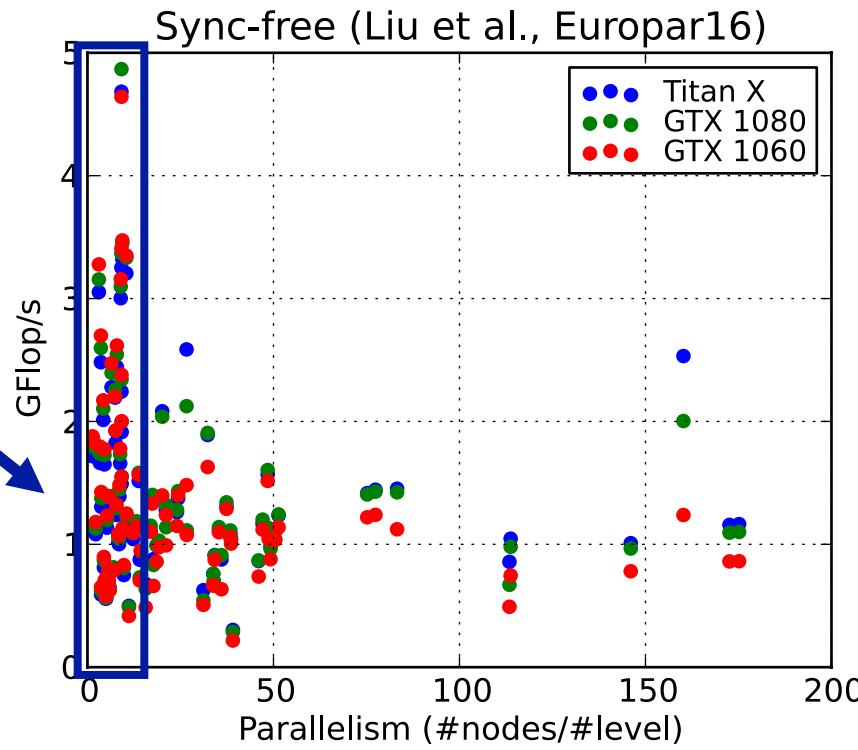
more
“nodes per level”
less
“levels”

Scalability of sync-free method for SpTRSV

SpTRSV perf.
inversely
scales with
hardware perf.



more
“levels”
less
“nodes per level”

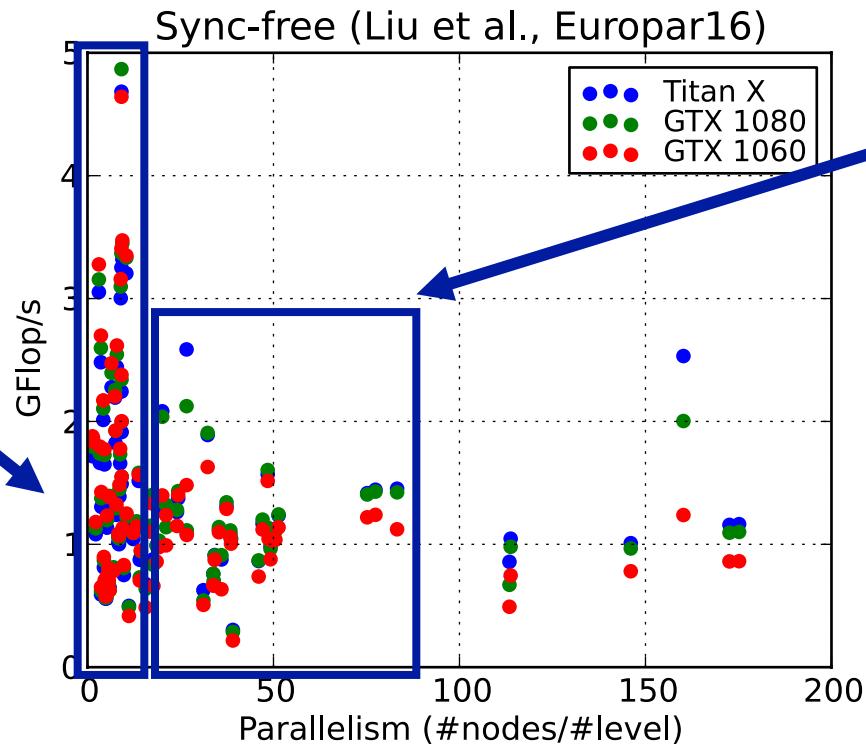


more
“nodes per level”
less
“levels”

Scalability of sync-free method for SpTRSV

SpTRSV perf.
inversely
scales with
hardware perf.

more
“levels”
less
“nodes per level”



SpTRSV perf.
does not
scale with
hardware perf.

more
“nodes per level”
less
“levels”

Scalability of sync-free method for SpTRSV

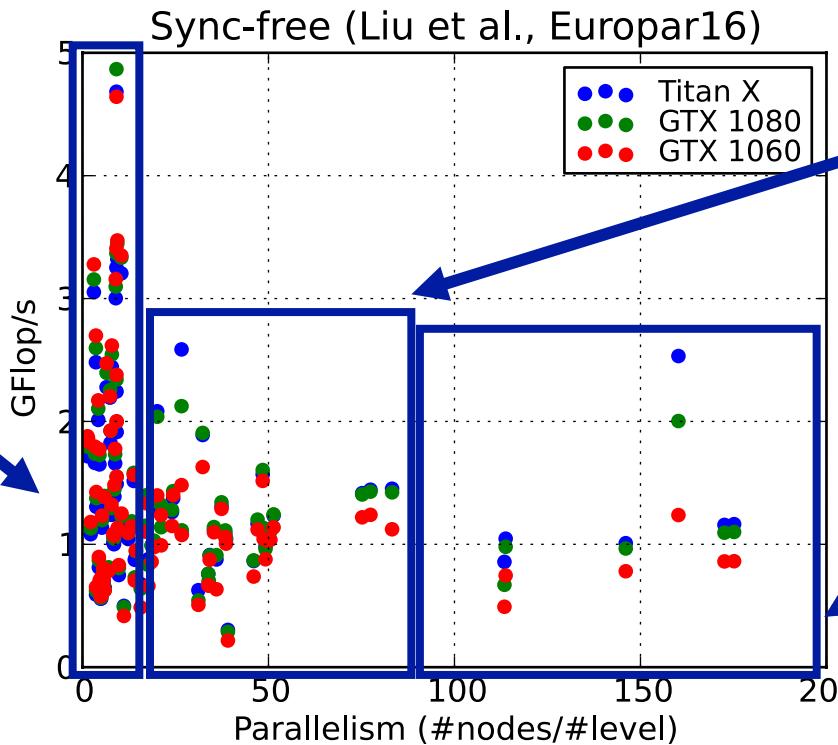
SpTRSV perf.
inversely
scales with
hardware perf.

SpTRSV perf.
does not
scale with
hardware perf.

more
“levels”
less
“nodes per level”

SpTRSV perf.
scales with
hardware perf.

more
“nodes per level”
less
“levels”



Source code of sync-free SpTRSV on Github

bhSPARSE / Benchmark_SpTRSM_using_CSC

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Fast Synchronization-Free Algorithms for Parallel Sparse Triangular Solves with Multiple Right-Hand Sides (SpTRSM)

21 commits 1 branch 0 releases 1 contributor MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

Commit	Message	Date
bhSPARSE committed on GitHub Update README.md	Latest commit 61046d6 on Jul 3	
SpTRSV_cuda	Update sprsv_syncfree_cuda.h	7 months ago
SpTRSV_opencl_amd	Upload.	9 months ago
LICENSE	Initial commit	9 months ago
README.md	Update README.md	2 months ago

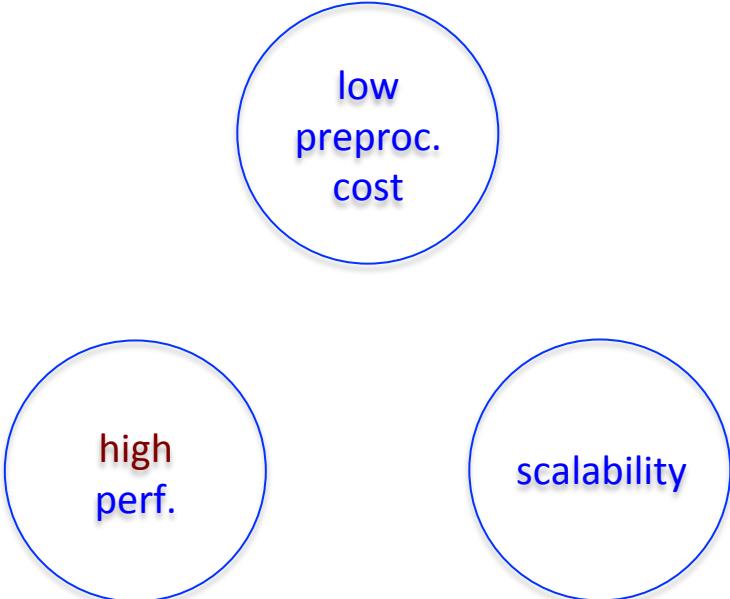
https://github.com/bhSPARSE/Benchmark_SpTRSM_using_CSC

Has been integrated into the MAGMA library.
<https://bitbucket.org/icl/magma>



Scalability, high perf. and low preproc. cost

(choose one of the three?)



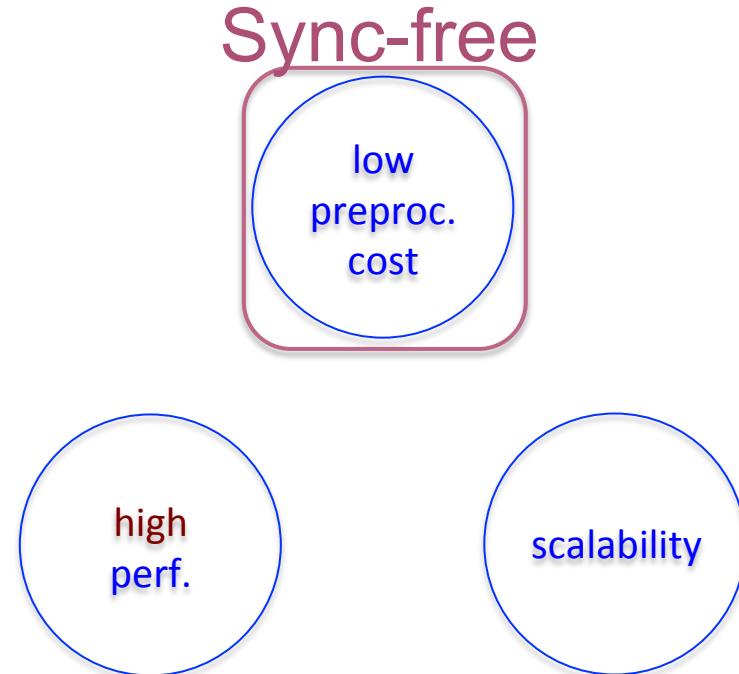
low
preproc.
cost

high
perf.

scalability

Scalability, high perf. and low preproc. cost

(choose one of the three?)



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

Sync-free

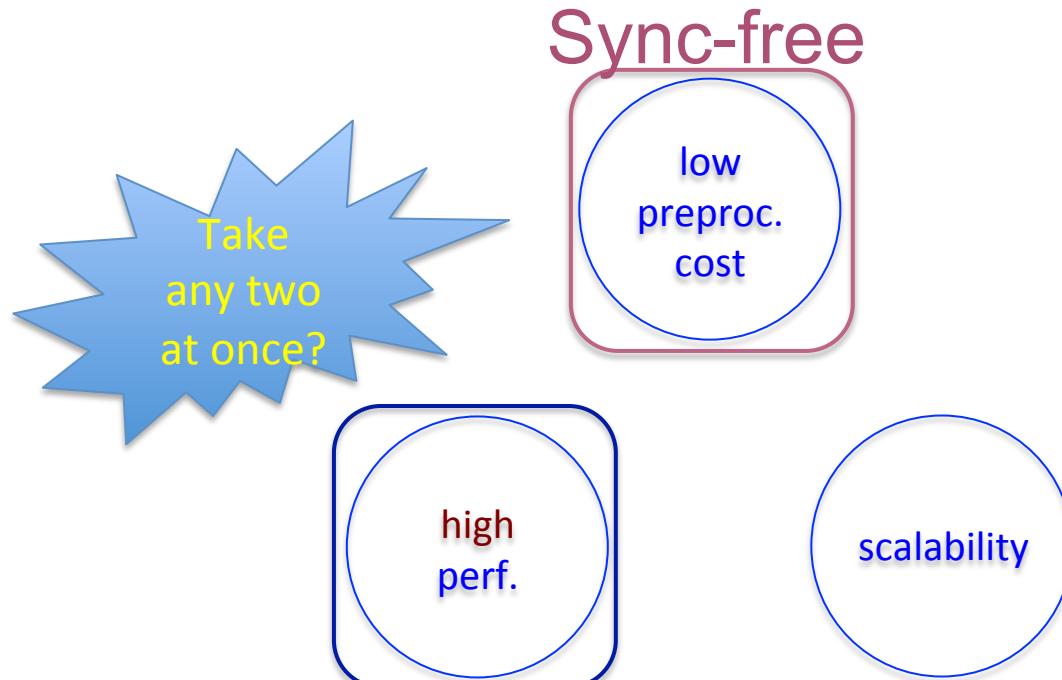
low
preproc.
cost

high
perf.

scalability

P2P (CPU) [Park et al.]

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

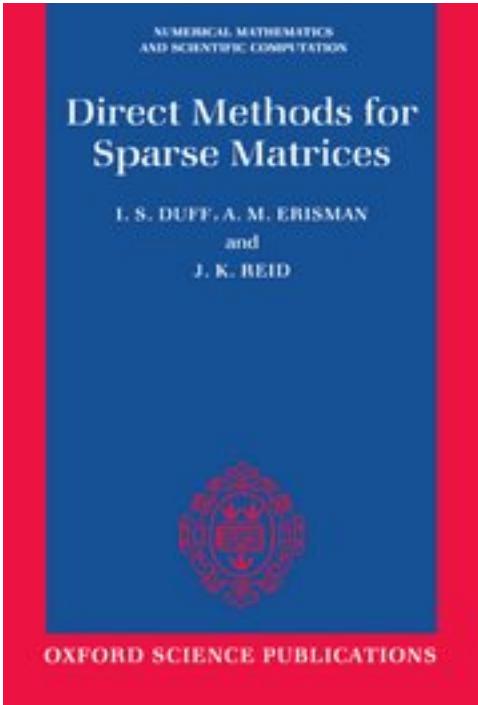


P2P (CPU) [Park et al.]

Observations (Summary)

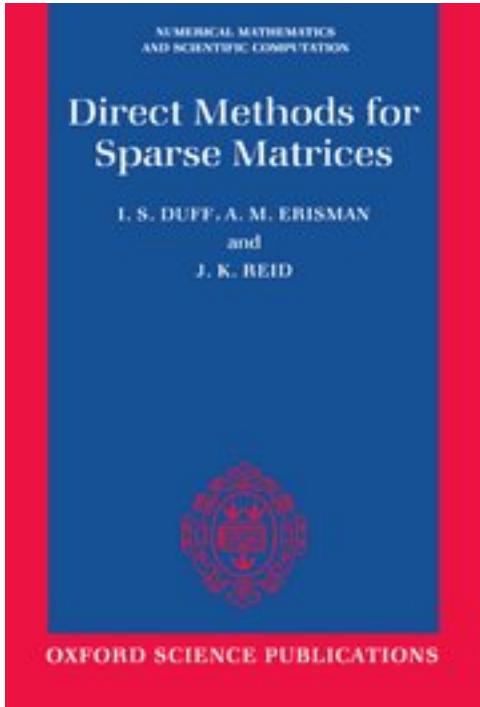
- The emergence of many-core processors magnifies the scalability problem brought by sparsity structures, and brings new research opportunities.
- There is no “enough good” SpMV implementation (low preprocessing cost + scalability + high performance); as for other operations, scalability and high performance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

Direct Methods for Sparse Matrices

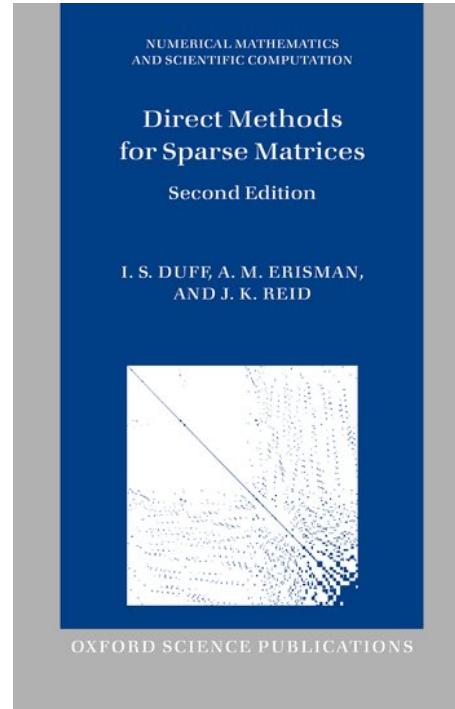


1st Edition, 1986

Direct Methods for Sparse Matrices

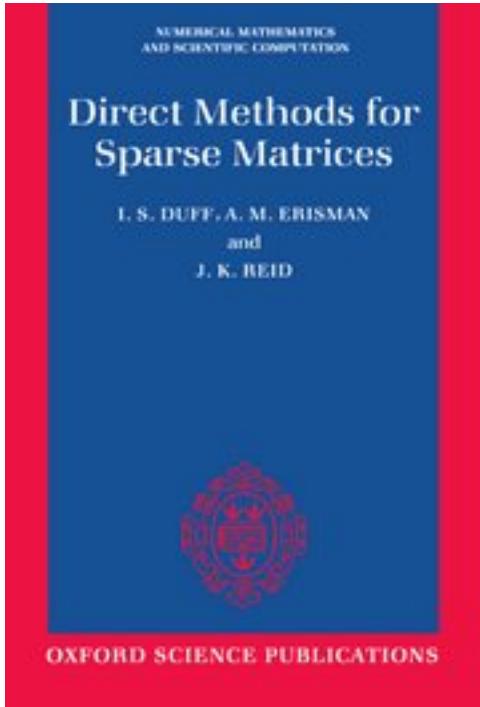


1st Edition, 1986

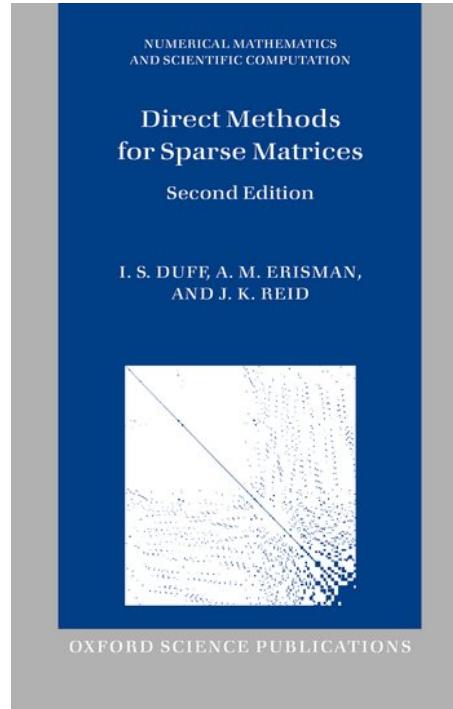


2nd Edition, 2017

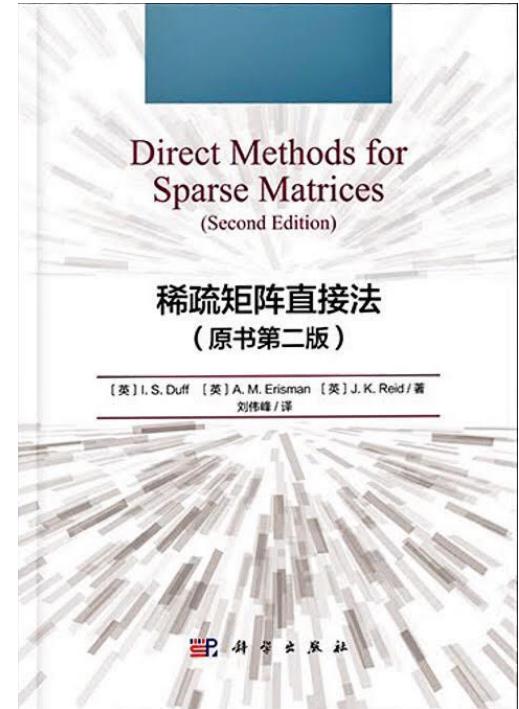
Direct Methods for Sparse Matrices



1st Edition, 1986



2nd Edition, 2017



Chinese Edition, 2018

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.