IA^3 '17, Denver, CO, 13 November

UNIVERSITAT JAUME I

Overcoming Load Imbalance for Irregular Sparse Matrices

Goran Flegar, Hartwig Anzt

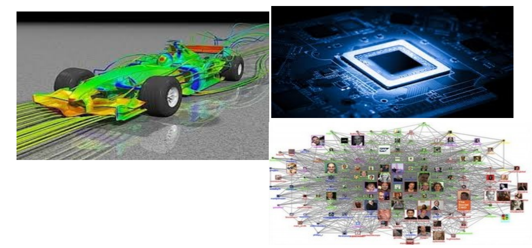
Scan me for slides!

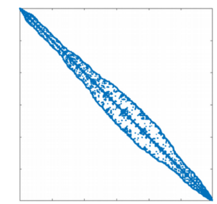
github.com/gflegar/talks/tree/master/sc_2017

flegar@uji.es

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
  - Focus: linear systems





$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; Universidad Jaume I

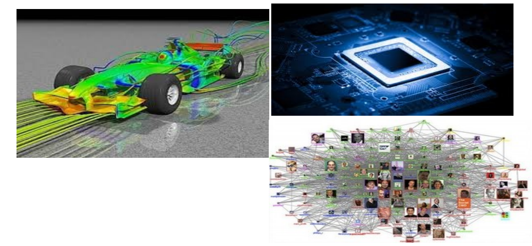
THE UNIVERSITY OF TENNESSEE KNOXVILLE

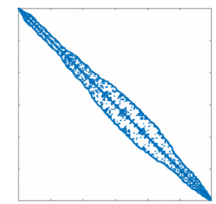
KIT Karlsruher Institut für Technologie

UNIVERSITAT JAUME I

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
  - Focus: linear systems
  - Iterative, Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations





$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; Universidad Jaume I

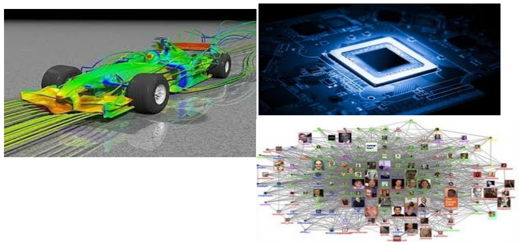
THE UNIVERSITY OF TENNESSEE KNOXVILLE

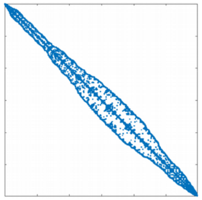
KIT Karlsruher Institut für Technologie

UNIVERSITAT JAUME I

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
  - Focus: linear systems
  - Iterative, Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations
  - Sparse matrix formats & SpMV
    - accelerate each iteration of the solver





$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; Universidad Jaume I

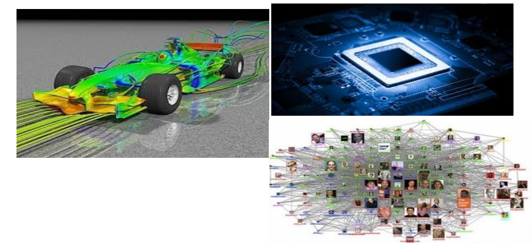
THE UNIVERSITY OF TENNESSEE KNOXVILLE

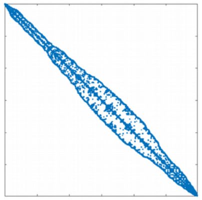
KIT Karlsruher Institut für Technologie

UNIVERSITAT JAUME I

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
  - Focus: linear systems
  - Iterative, Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations
  - Sparse matrix formats & SpMV
    - accelerate each iteration of the solver
  - Preconditioners
    - reduce the number of iterations





$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; Universidad Jaume I

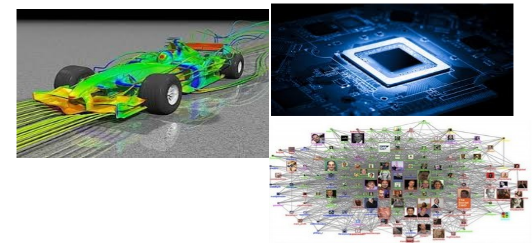
THE UNIVERSITY OF TENNESSEE KNOXVILLE

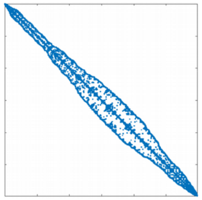
KIT Karlsruher Institut für Technologie

UNIVERSITAT JAUME I

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
  - Focus: linear systems
  - Iterative, Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations
  - Sparse matrix formats & SpMV
    - accelerate each iteration of the solver
  - Preconditioners
    - reduce the number of iterations





$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruher Institute of Technology; Universidad Jaume I

THE UNIVERSITY OF TENNESSEE KNOXVILLE

KIT Karlsruher Institut für Technologie

UNIVERSITAT JAUME I

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

→

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	
0	0	1	2	2	2	3	3	row
0	2	2	0	1	3	0	3	column

COO

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

3.2	1.2	
0.4		
2.7	1.3	4.1
0.1	2.7	

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

$\rightarrow$

3.2	1.2	
0.4		
2.7	1.3	4.1
0.1	2.7	

0	2	
2		
0	1	3
0	3	

Sparse matrix formats

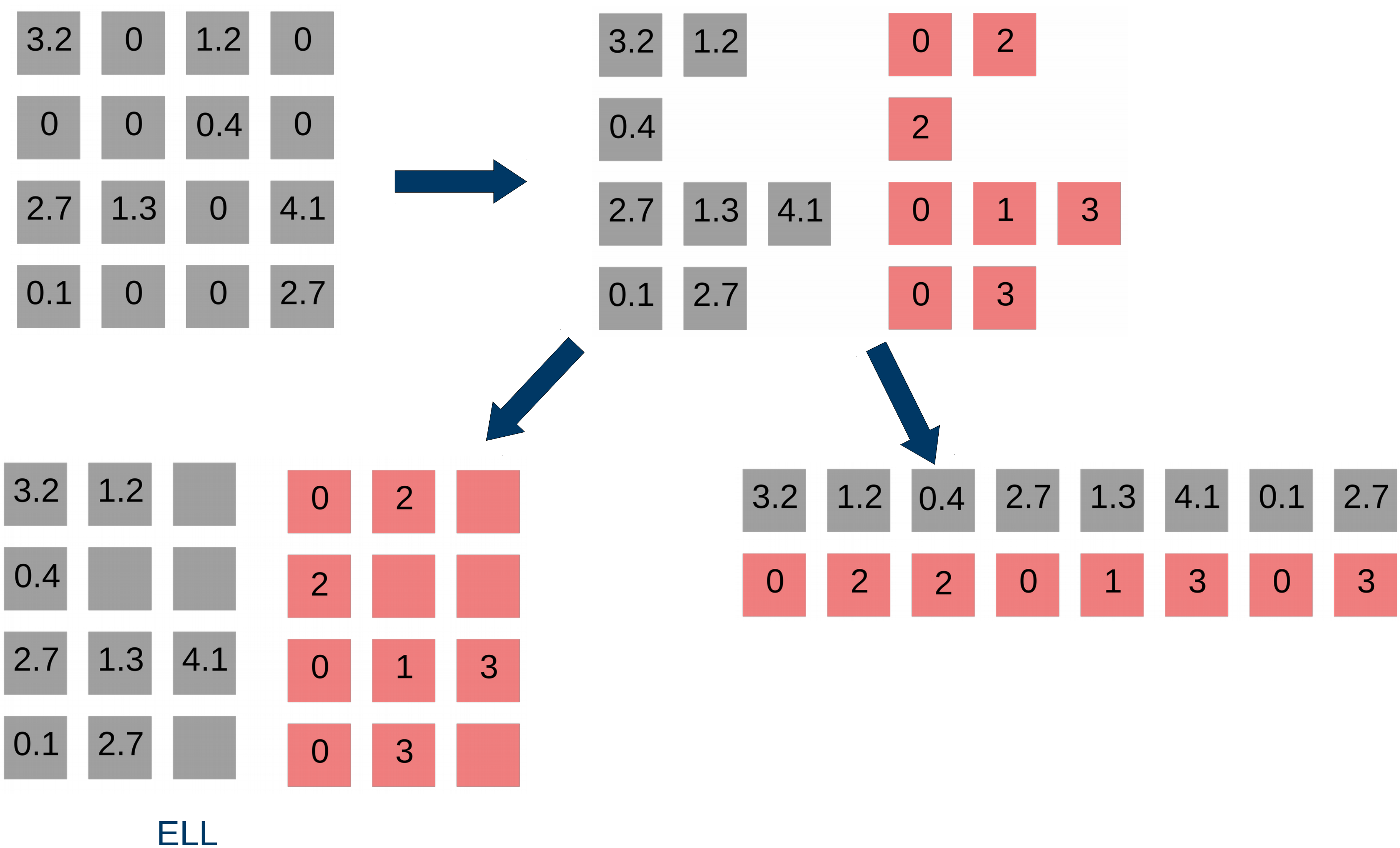
3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

3.2	1.2		0	2	
0.4			2		
2.7	1.3	4.1	0	1	3
0.1	2.7		0	3	

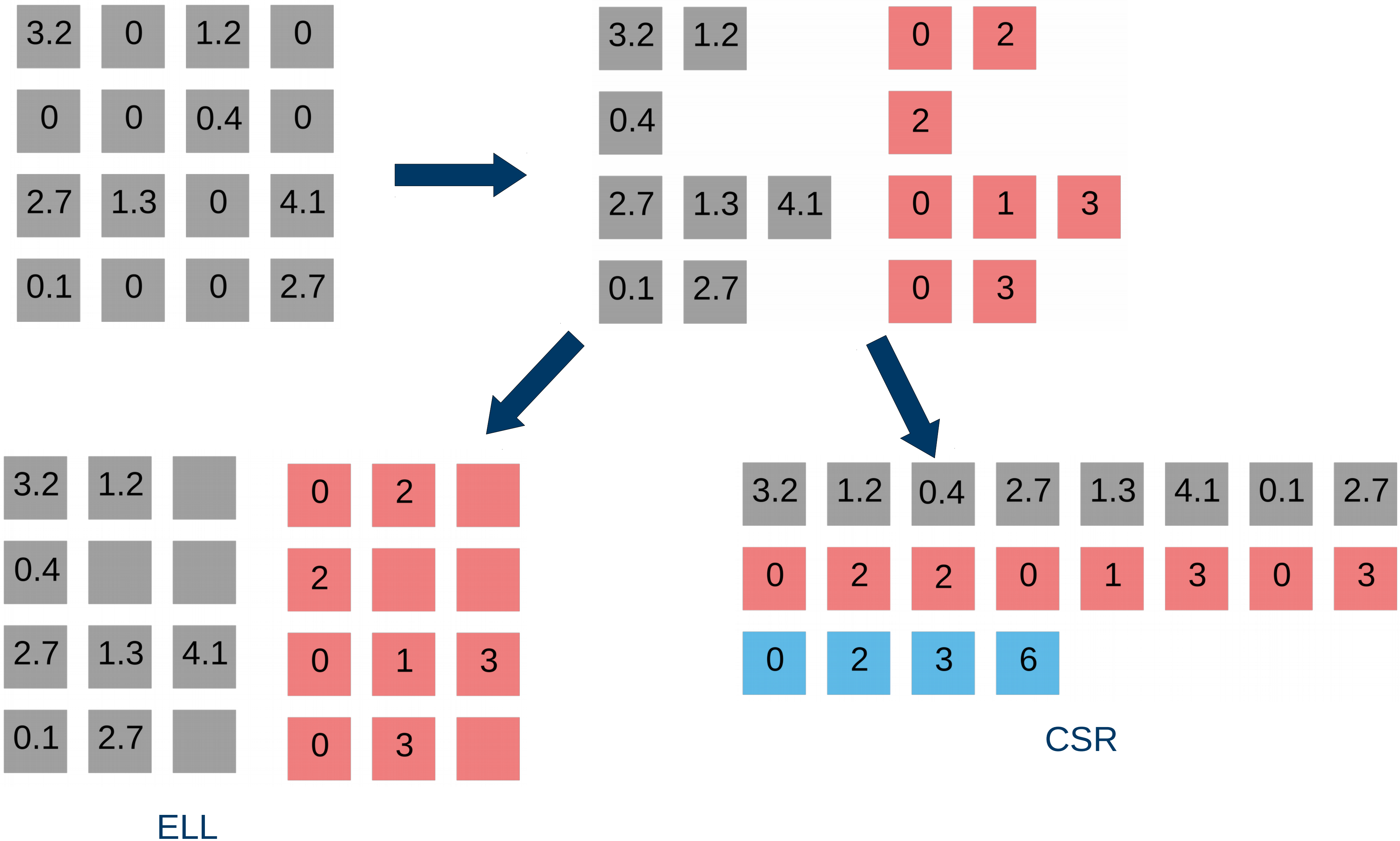
3.2	1.2		0	2	
0.4			2		
2.7	1.3	4.1	0	1	3
0.1	2.7		0	3	

ELL

Sparse matrix formats



Sparse matrix formats



Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

3.2	1.3	1.2	4.1
2.7		0.4	2.7
0.1			

0	2	0	2
2		1	3
3			

… leads to CSC

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

3.2	1.3	1.2	4.1
2.7		0.4	2.7
0.1			

0	2	0	2
2		1	3
3			

… leads to CSC

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	2	2	0	1	3	0	3
0	2	3	6				

CSR

“Standard” approach

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

→

3.2	1.3	1.2	4.1
2.7		0.4	2.7
0.1			

0	2	0	2
2		1	3
3			

… leads to CSC

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	2	2	0	1	3	0	3
0	2	3	6				

CSR

"Standard" approach

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	0	1	2	2	2	3	3
0	2	2	0	1	3	0	3

COO

Considered inferior to CSR (memory consumption)

Sparse matrix formats

3.2	0	1.2	0
0	0	0.4	0
2.7	1.3	0	4.1
0.1	0	0	2.7

→

3.2	1.3	1.2	4.1
2.7		0.4	2.7
0.1			

0	2	0	2
2		1	3
3			

… leads to CSC

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	2	2	0	1	3	0	3
0	2	3	6				

CSR

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	0	1	2	2	2	3	3
0	2	2	0	1	3	0	3

COO

"Standard" approach

Considered inferior to CSR (memory consumption)

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

$$y := Ax$$

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

$$y := Ax$$

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

$$y := Ax$$

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Bell & Garland '08

- parallelize outer loop

~ cuSPARSE SpMV

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

$$y := Ax$$

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Bell & Garland ‘08
- parallelize outer loop

~ cuSPARSE SpMV

Load imbalance!
Non-coalescence!

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

$$y := Ax$$

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Bell & Garland ‘08
- parallelize outer loop

~ cuSPARSE SpMV

Load imbalance!
Non-coalescence!

Specialized formats
- HYB (ELL + COO) [cuSPARSE]
- CSR5 [Liu, Vinter ‘15], CSR-I [Flegar, Quintana ‘17]
- SELL-P [Kreutzer et al.] – good memory access, parallelizes well
- … a few new ones every year

CSR SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	2	2	0	1	3	0	3	Column indexes (colidx)
0	2	3	6					Row pointers (rowptr)

$$y := Ax$$

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Bell & Garland ‘08
- parallelize outer loop

~ cuSPARSE SpMV

Load imbalance!
Non-coalescence!

Specialized formats
- HYB (ELL + COO) [cuSPARSE]
- CSR5 [Liu, Vinter ‘15], CSR-I [Flegar, Quintana ‘17]
- SELL-P [Kreutzer et al.] – good memory access, parallelizes well
- … a few new ones every year

CSR-I idea

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

CSR-I idea

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Collapse the two loops into one.

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int i = 0; i < nnz; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}
```

CSR-I idea

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Collapse the two loops into one.

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int i = 0; i < nnz; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}
```

Split the loop into equal chunks.

```
const int T = thread_count;
void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}}
```

CSR-I idea

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < m; ++i) {
    for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
      y[i] +=  val[j] * x [ colidx[j] ];
  }
}
```

Collapse the two loops into one.

```
void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int i = 0; i < nnz; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}
```

Split the loop into equal chunks.

```
const int T = thread_count;
void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}}
```

Parallelize this!

COO SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	0	1	2	2	2	3	3	Row indexes (rowidx)
0	2	2	0	1	3	0	3	Column indexes (colidx)

COO SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	0	1	2	2	2	3	3	Row indexes (rowidx)
0	2	2	0	1	3	0	3	Column indexes (colidx)

```
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < nnz; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

COO SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	0	1	2	2	2	3	3	Row indexes (rowidx)
0	2	2	0	1	3	0	3	Column indexes (colidx)

```
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < nnz; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

Split the loop into equal chunks.

```
const int T = thread_count;
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

COO SpMV

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7	Values (val)
0	0	1	2	2	2	3	3	Row indexes (rowidx)
0	2	2	0	1	3	0	3	Column indexes (colidx)

```
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int i = 0; i < nnz; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

Split the loop into equal chunks.

```
const int T = thread_count;
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

Parallelize this!

COO vs CSR-I SpMV

COO:

```
const int T = thread_count;
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

CSR-I:

```
const int T = thread_count;
void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}}
```

COO vs CSR-I SpMV

COO:

```
const int T = thread_count;
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

CSR-I:

```
const int T = thread_count;
void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}}
```

Race conditions!

COO vs CSR-I SpMV

COO:

```
const int T = thread_count;
void SpMV_COO(int m, int *rowidx, int *colidx, float *val, float *x, float *y) {
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      y[rowidx[i]] += val[i] * x[colidx[i]];
}}
```

CSR-I:

```
const int T = thread_count;
void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
  int row = -1, next_row = 0, nnz = rowptr[m];
  for (int k = 0; k < T; ++k) {
    for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
      while (i >= next_row) next_row = rowptr[++row+1];
      y[row] += val[i] * x[ colidx[i] ];
}}}
```

Race conditions!

- Use atomics
- Accumulate partial result into registers

Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp) – shared cache lines!

Spreading out threads causes strided memory access.

Assign one warp per chunk.

Getting good performance on GPUs

CSR-I

3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	2	2	0	1	3	0	3

Getting good performance on GPUs

CSR-I

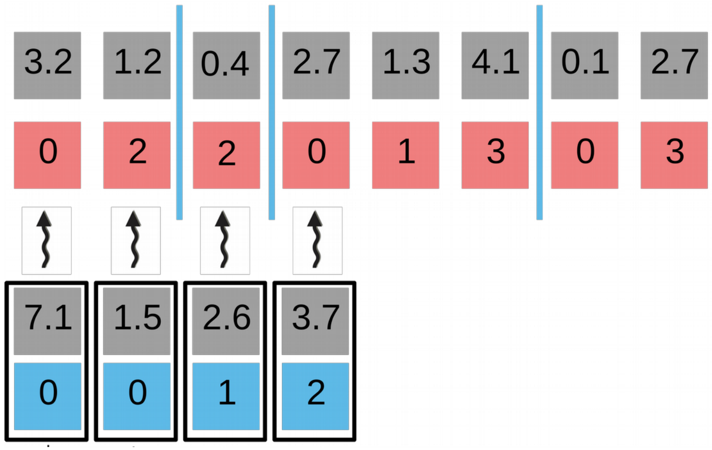
3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	2	2	0	1	3	0	3

COO

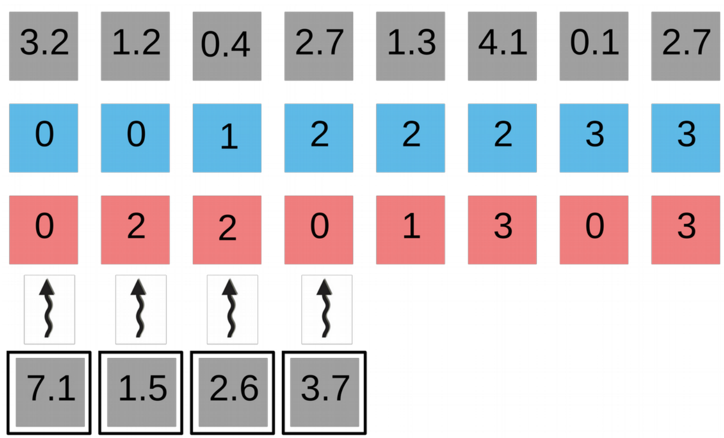
3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	0	1	2	2	2	3	3
0	2	2	0	1	3	0	3

Getting good performance on GPUs

CSR-I

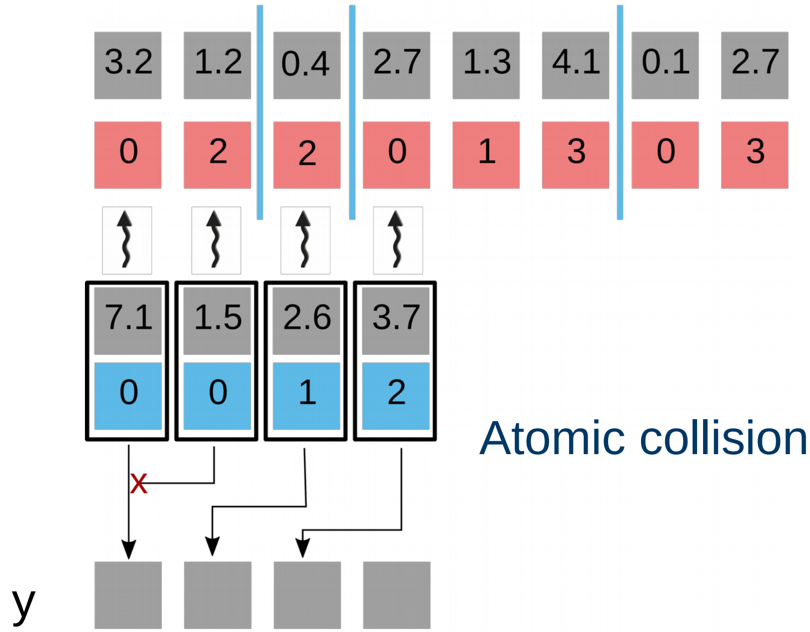


COO



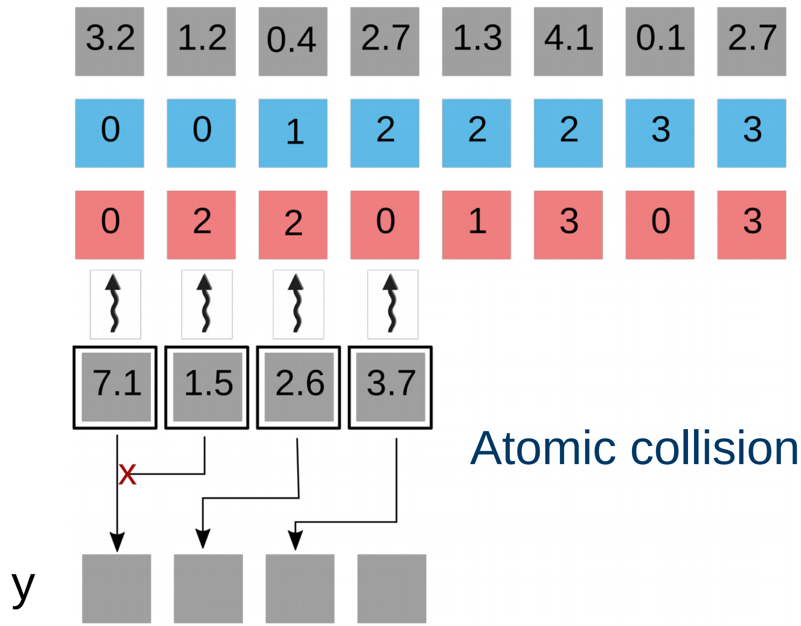
Getting good performance on GPUs

CSR-I



Atomic collision!

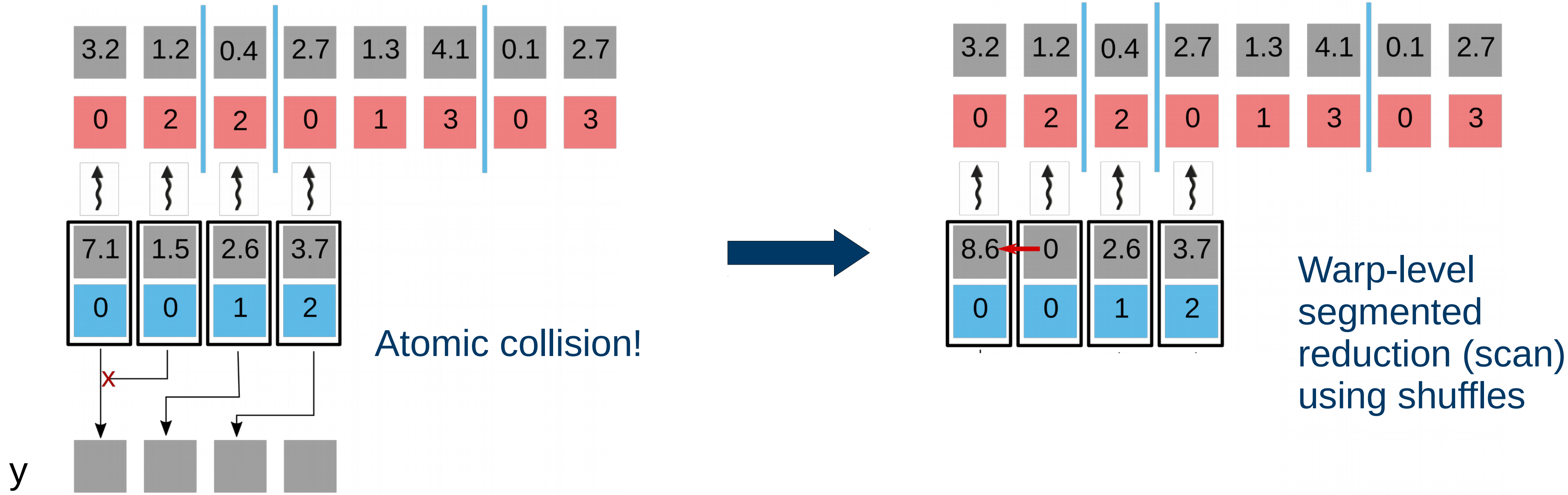
COO



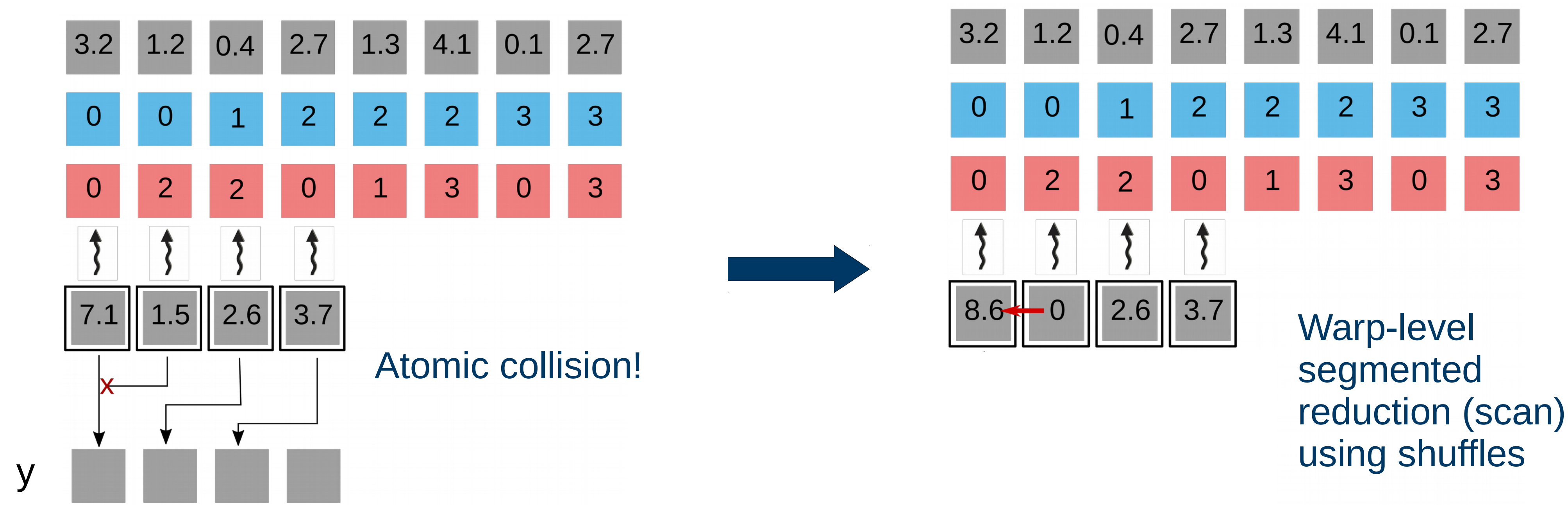
Atomic collision!

Getting good performance on GPUs

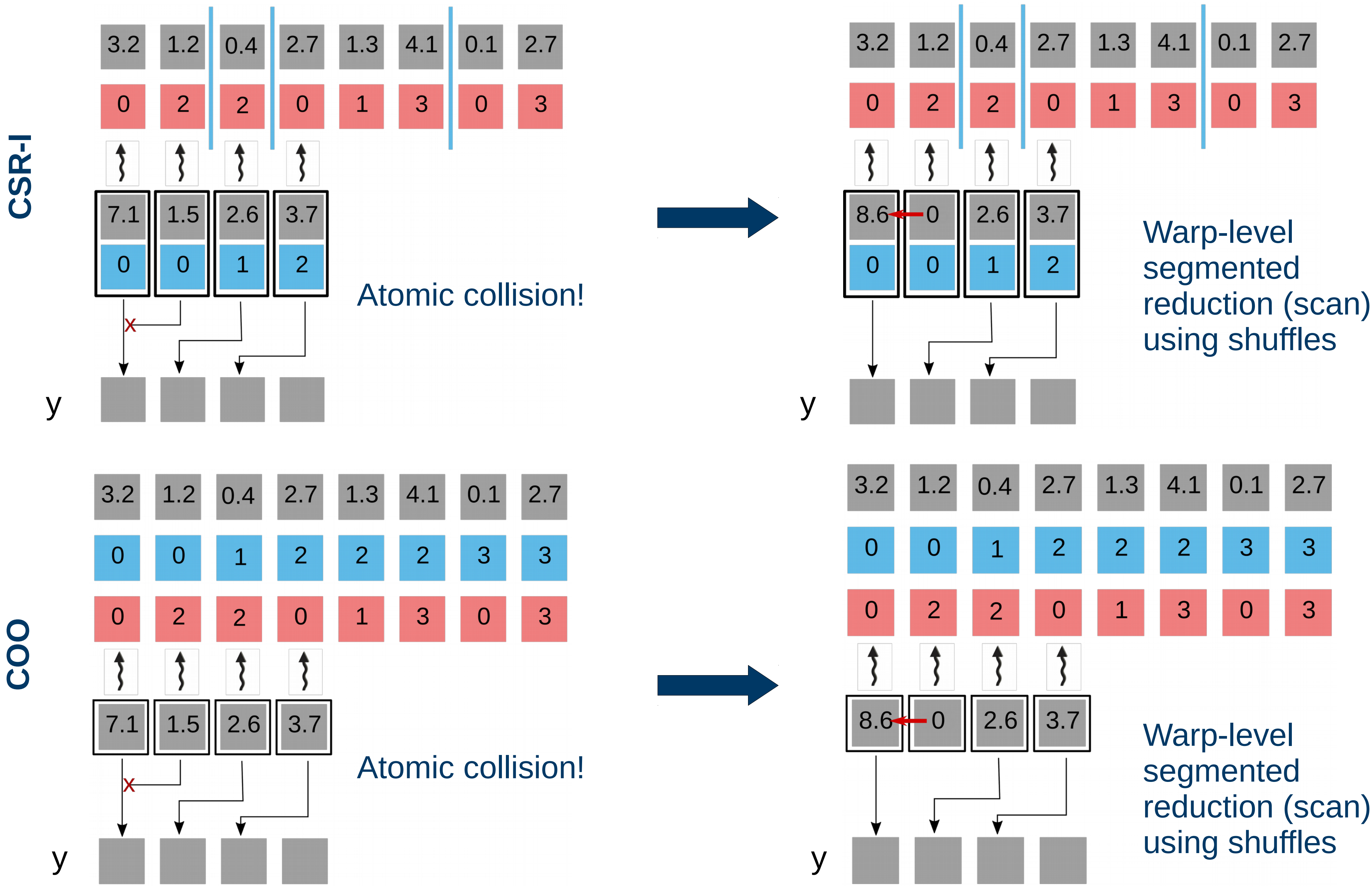
CSR-I



COO

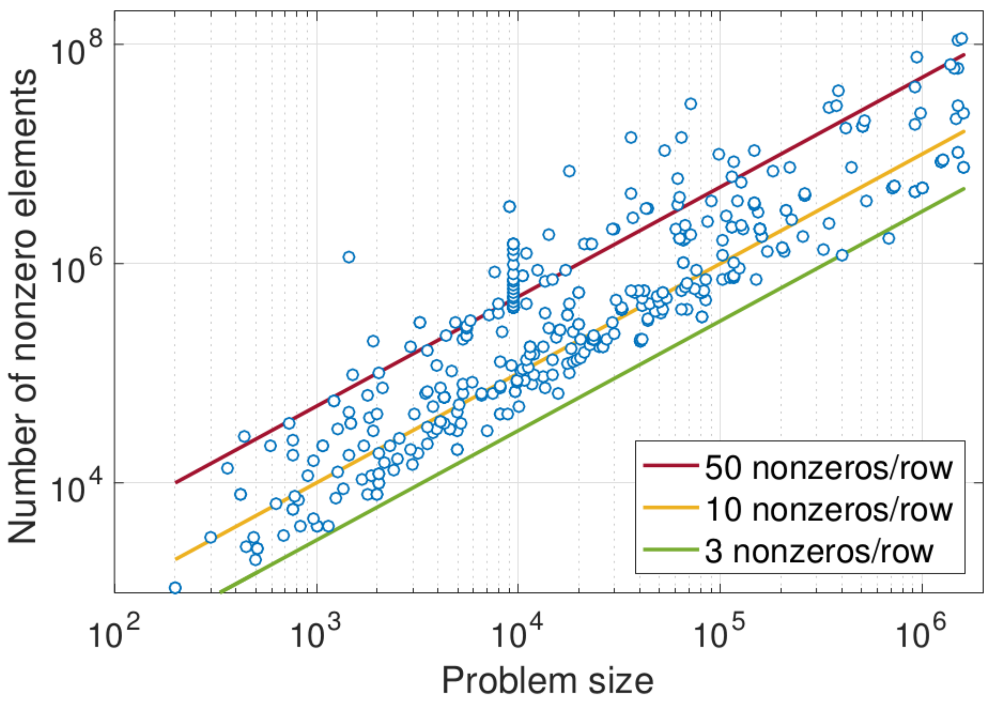


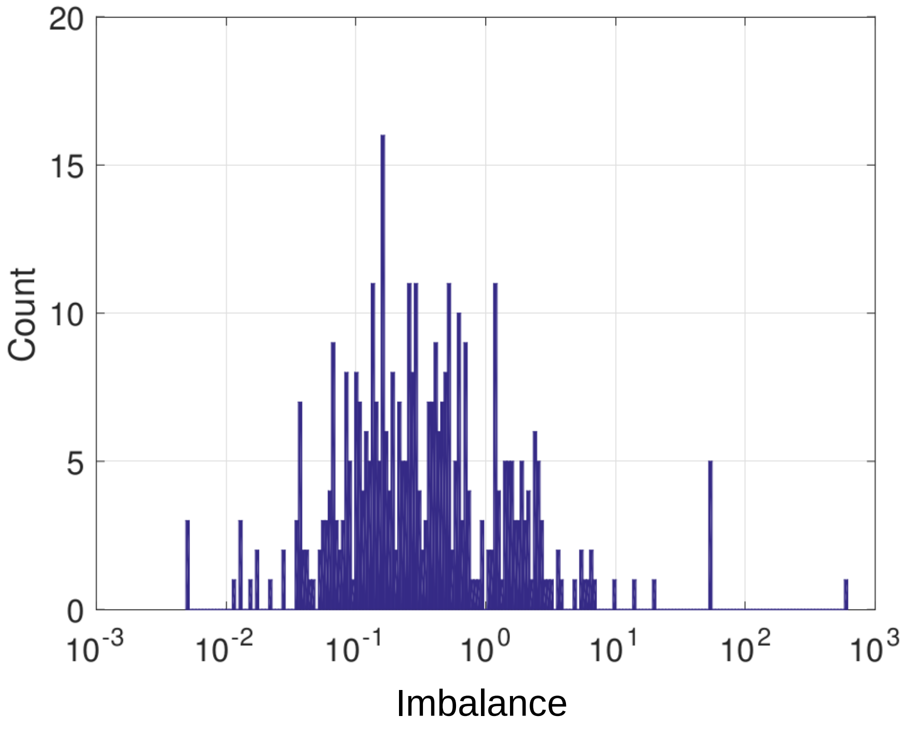
Getting good performance on GPUs



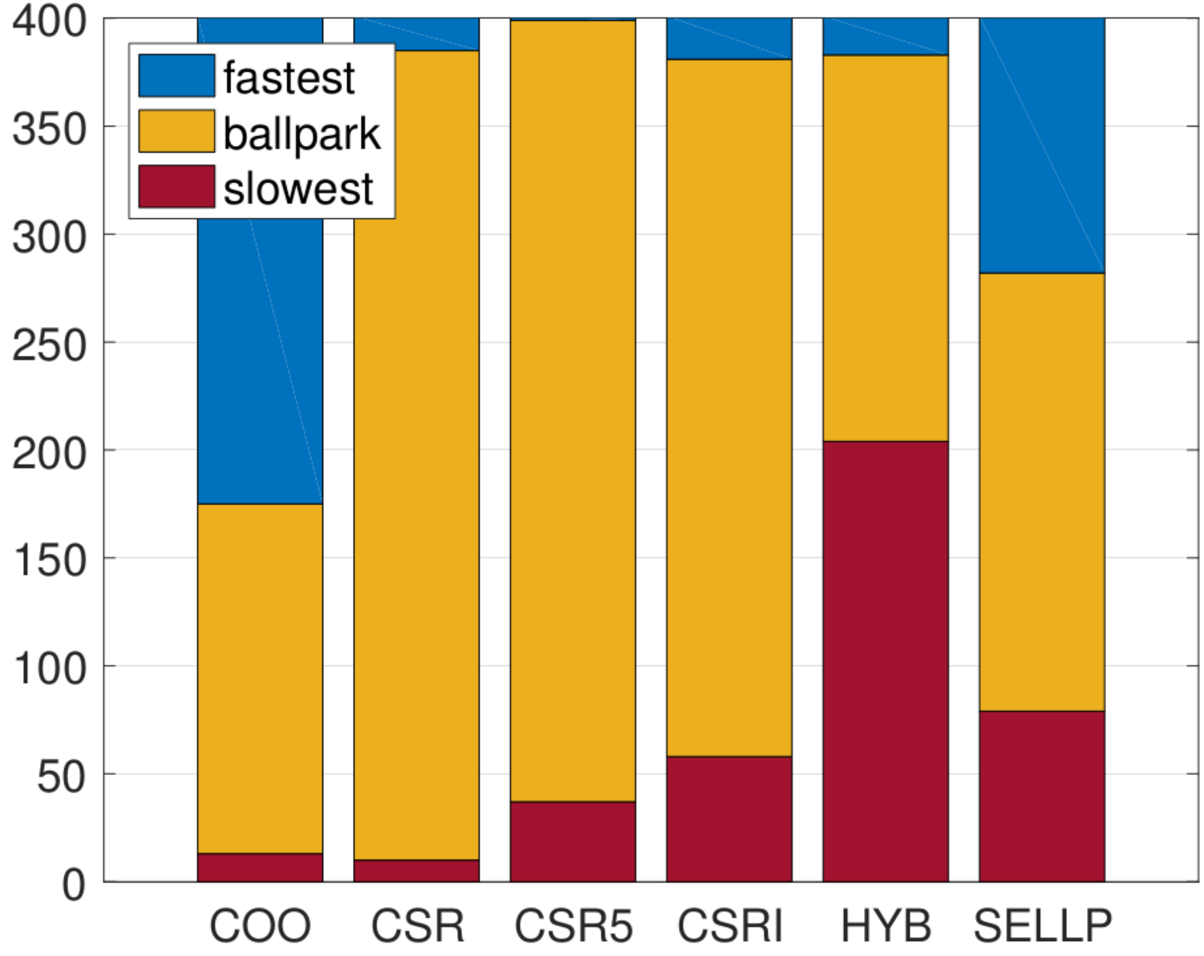
Test matrices

400 matrices from SuiteSparse matrix collection





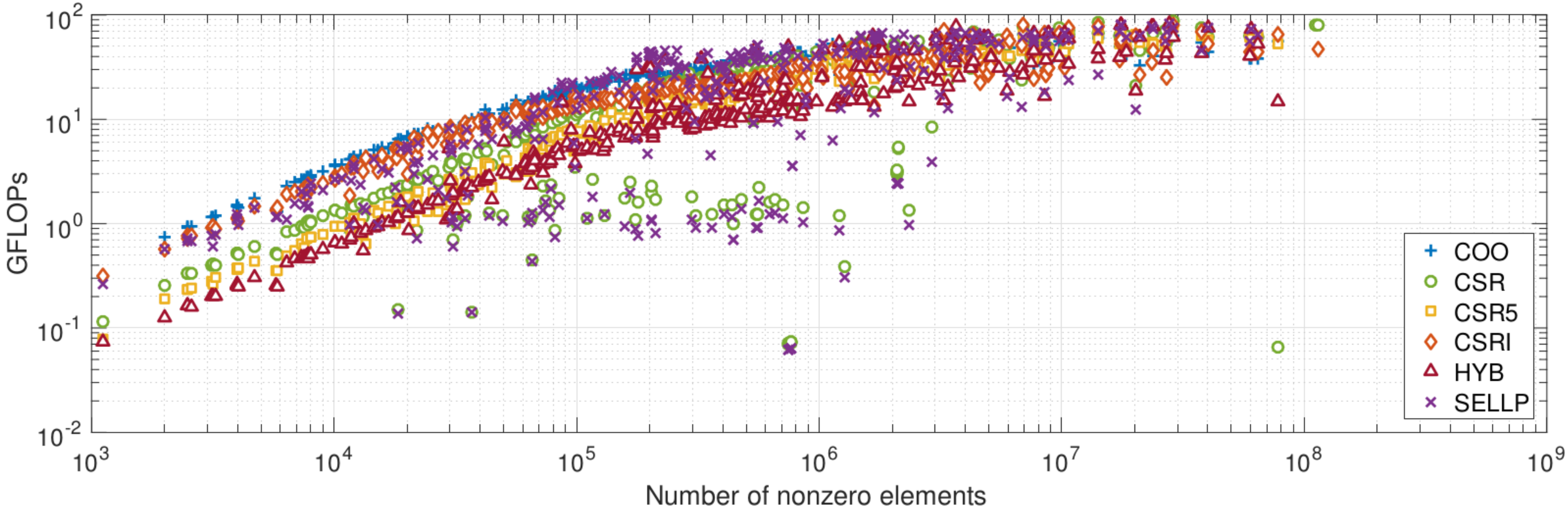
Performance / format histogram



COO wins most of the cases!

* P100 on Piz-Daint supercomputer @ CSCS

(Too) detailed performance plot



COO is superior for small matrices!

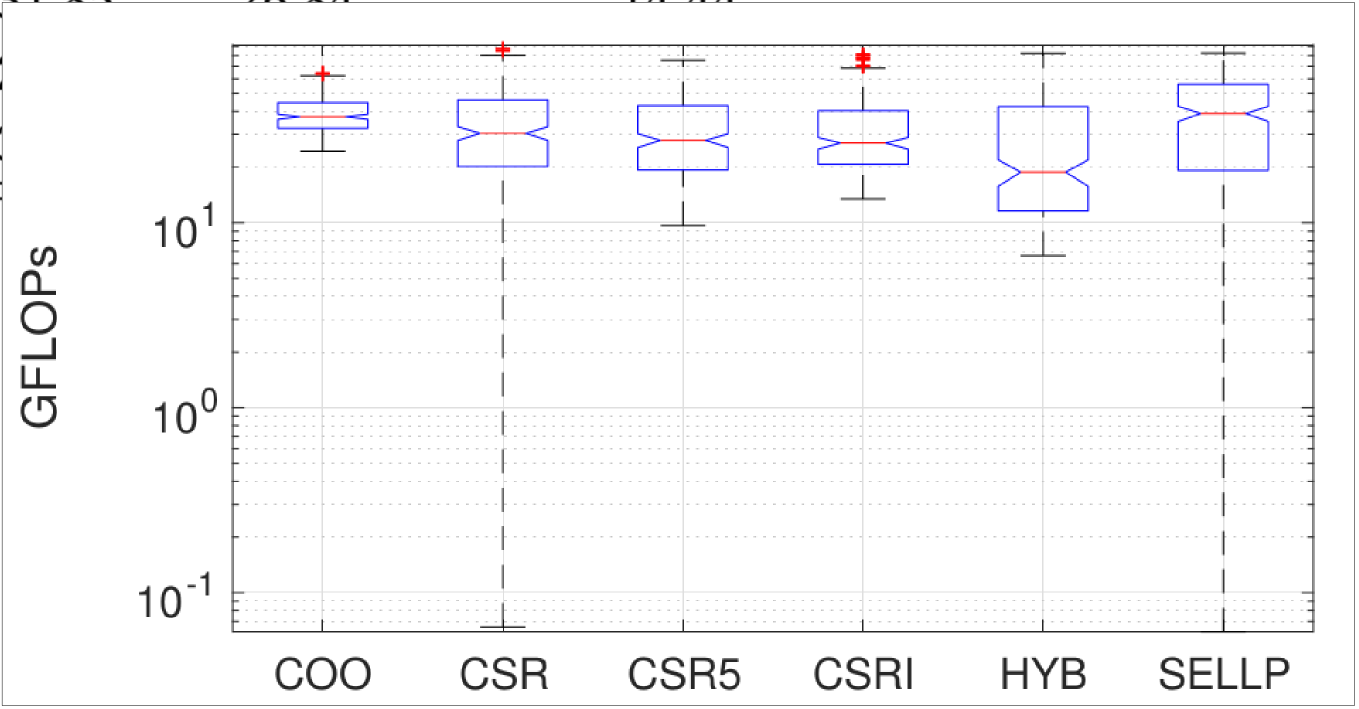
Basic statistics

Kernel	min	max	average	median	standard-dev.
COO	24.29	64.32	38.86	37.24	9.16
CSR	0.07	87.43	32.77	30.43	20.07
CSR5	9.66	75.56	31.79	27.15	15.58
CSRI	13.47	81.21	31.85	26.84	14.44
HYB	6.64	82.43	27.98	18.74	20.22
SELLP	0.06	82.62	36.42	38.64	22.46

COO has good average & small deviation!

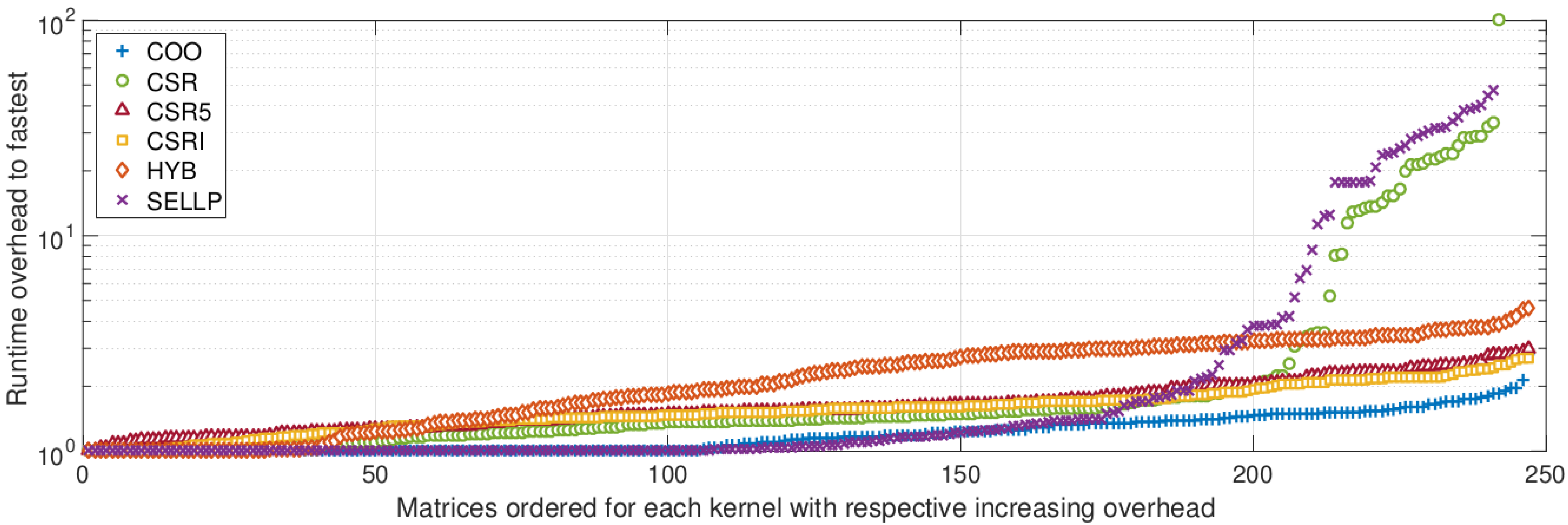
Basic statistics

Kernel	min	max	average	median	standard-dev.
COO	24.29	64.32	38.86	37.24	9.16
CSR	0.07	87.43	32.77	30.43	20.07
CSR5	9.66	75.56	31.79	27.15	15.58
CSRI	13.47	81.21	31.85	26.84	14.44
HYB	6.64	82.43	[illegible]	[illegible]	[illegible]
SELLP	0.06	82.62	[illegible]	[illegible]	[illegible]



COO has good average & small deviation!

Runtime comparison



COO is never “slow”!

Conclusion

No “holy grail” format / algorithm for SpMV.

- COO is a good all-rounder.

Re-visiting “forgotten” formats may pay of.

Atomics + warp shuffles are sometimes a good alternative to reduction.

Thank you! Questions?

All functionalities are part of the MAGMA-sparse project.

MAGMA SPARSE

ROUTINES	BiCG, BiCGSTAB, Block-Asynchronous Jacobi, CG, CGS, GMRES, IDR, Iterative refinement, LOBPCG, LSQR, QMR, TFQMR
PRECONDITIONERS	ILU / IC, Jacobi, ParILU, ParILUT, Block Jacobi, ISAI
KERNELS	SpMV, SpMM
DATA FORMATS	CSR, ELL, SELL-P, CSR5, HYB

http://icl.cs.utk.edu/magma/

Scan me for slides!

github.com/gflegar/talks/tree/master/sc_2017

This research is based on a cooperation between Hartwig Anzt (Karlsruhe Institute of Technology, University of Tennessee) and Goran Flegar (Universidad Jaume I).

Karlsruher Institut für Technologie

THE UNIVERSITY OF TENNESSEE KNOXVILLE

UNIVERSITAT JAUME I