UNIVERSITAT JAUME I

Balanced CSR Sparse Matrix-Vector Product on Graphics Processors
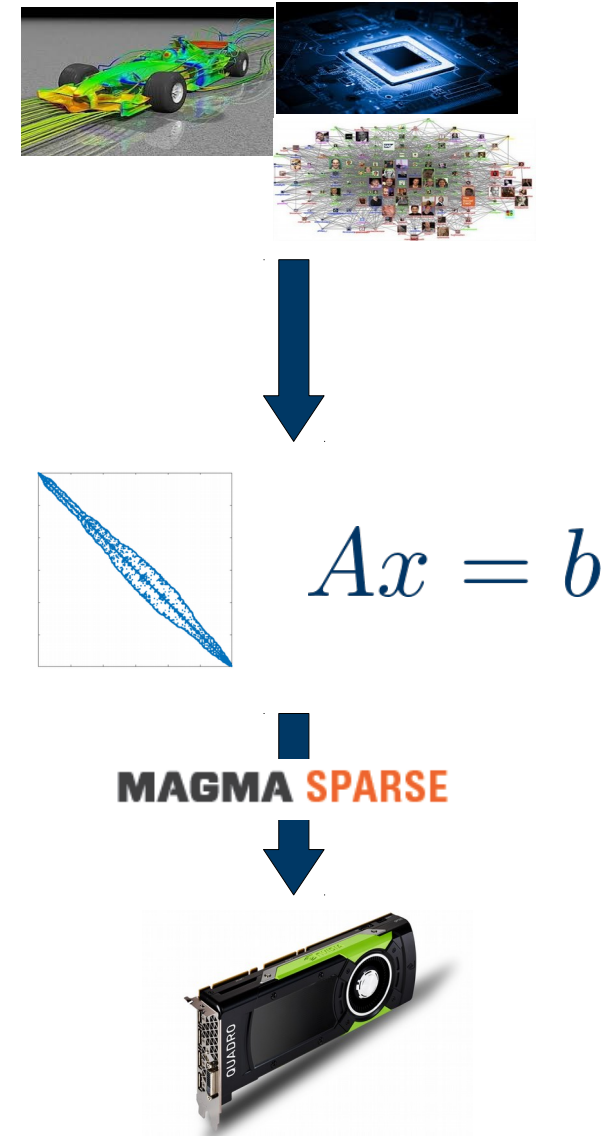
Goran Flegar, Enrique S. Quintana-Ortí

Scan me for slides!

github.com/gflegar/talks/europar_2017

flegar@uji.es

MAGMA-sparse software library

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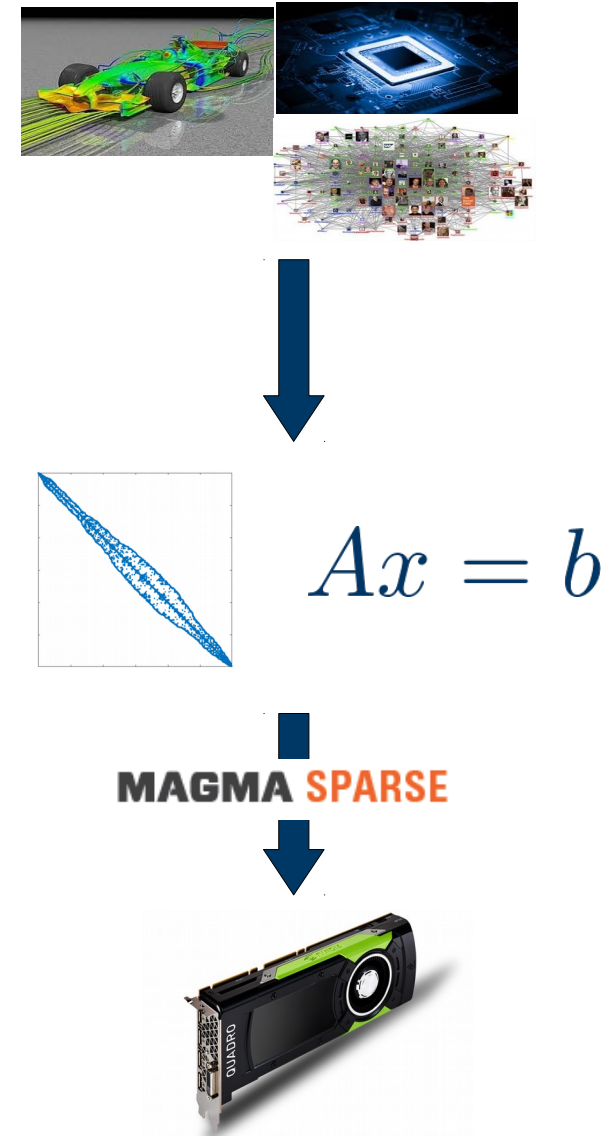
  – Focus: linear systems

$$Ax = b$$

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

$$Ax = b$$

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; University 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
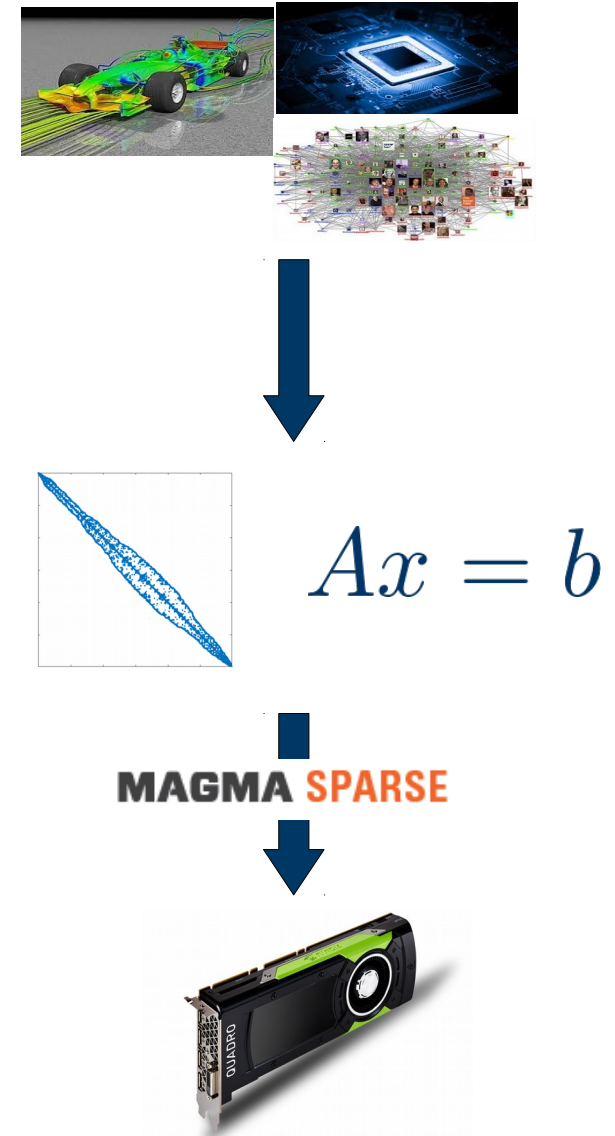
    - SpMV

    - BLAS-1 operations

  – Sparse matrix formats & SpMV
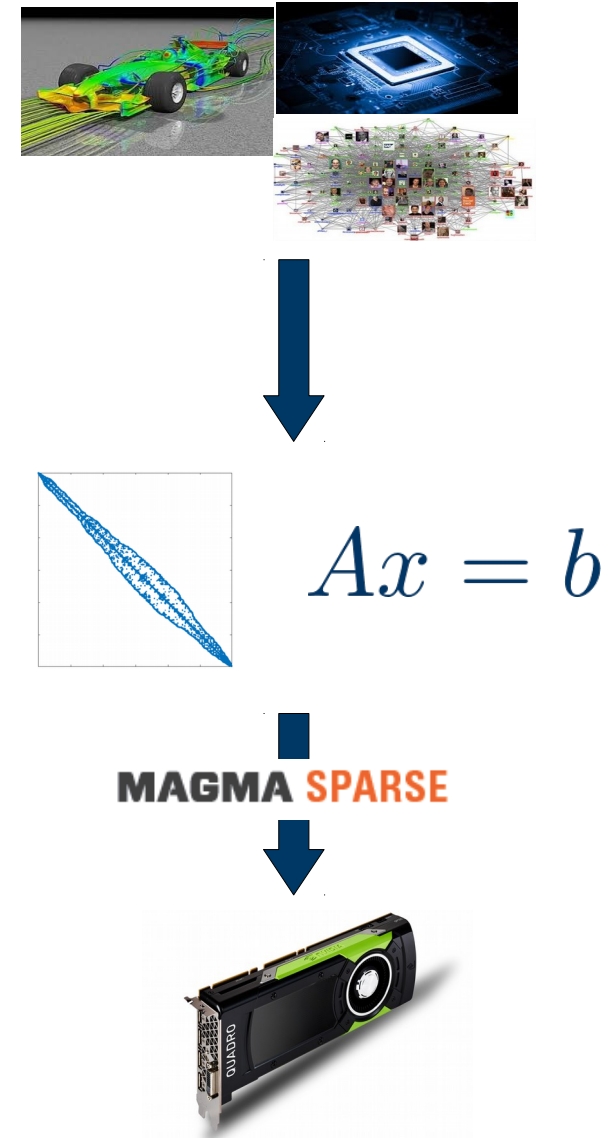
    - accelerate each iteration of the solver

$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; University 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
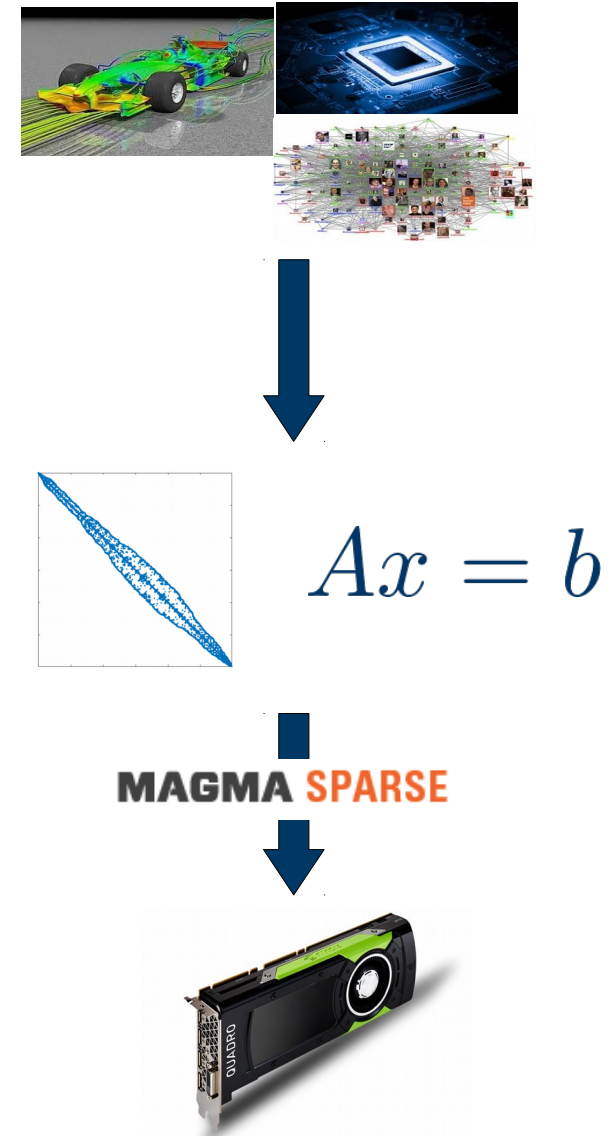
    - reduce the number of iterations

$$Ax = b$$

MAGMA SPARSE

Joint effort: Innovative Computing Lab at University of Tennessee, Knoxvile; Karlsruhe Institute of Technology; University 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
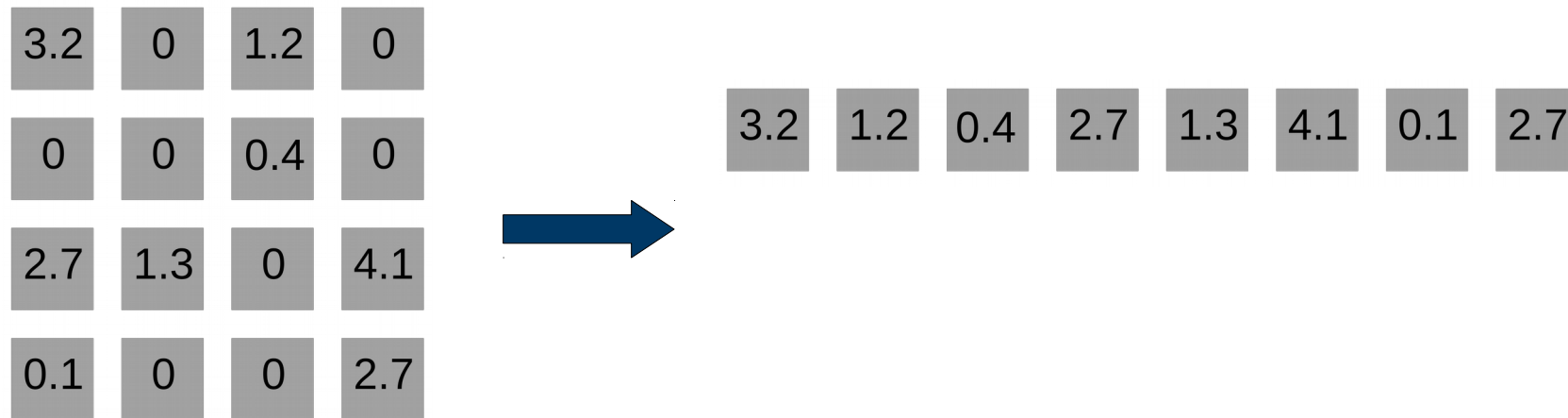
    - reduce the number of iterations

$$Ax = b$$

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

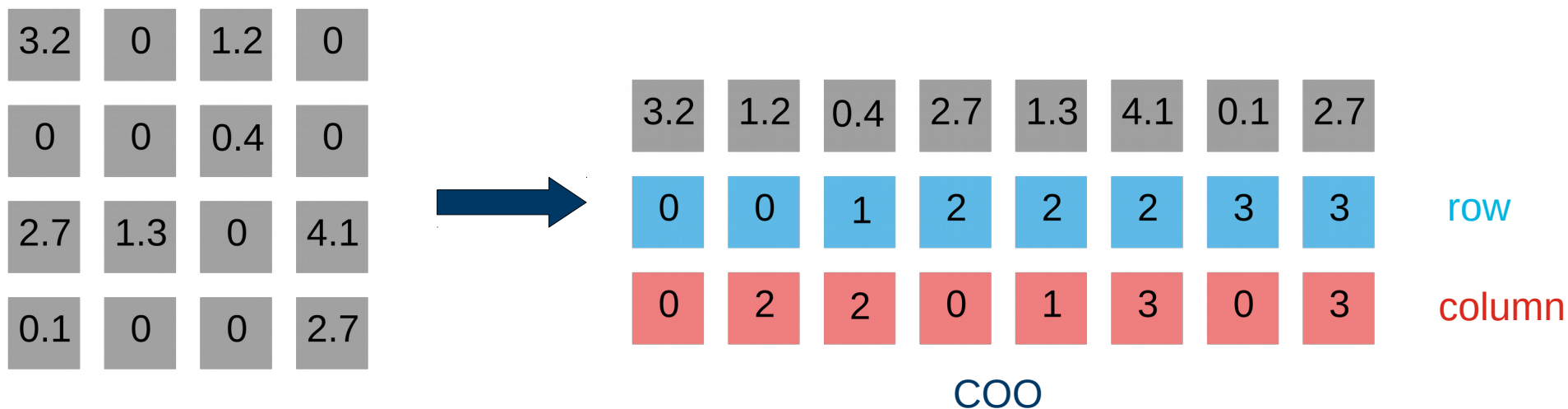THE UNIVERSITY OF TENNESSEE KNOXVILLE

KIT
Karlsruher Institut für Technologie

UNIVERSITAT JAUME I

Sparse matrix formats

$$\begin{bmatrix} 3.2 & 0 & 1.2 & 0 \\ 0 & 0 & 0.4 & 0 \\ 2.7 & 1.3 & 0 & 4.1 \\ 0.1 & 0 & 0 & 2.7 \end{bmatrix}$$

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

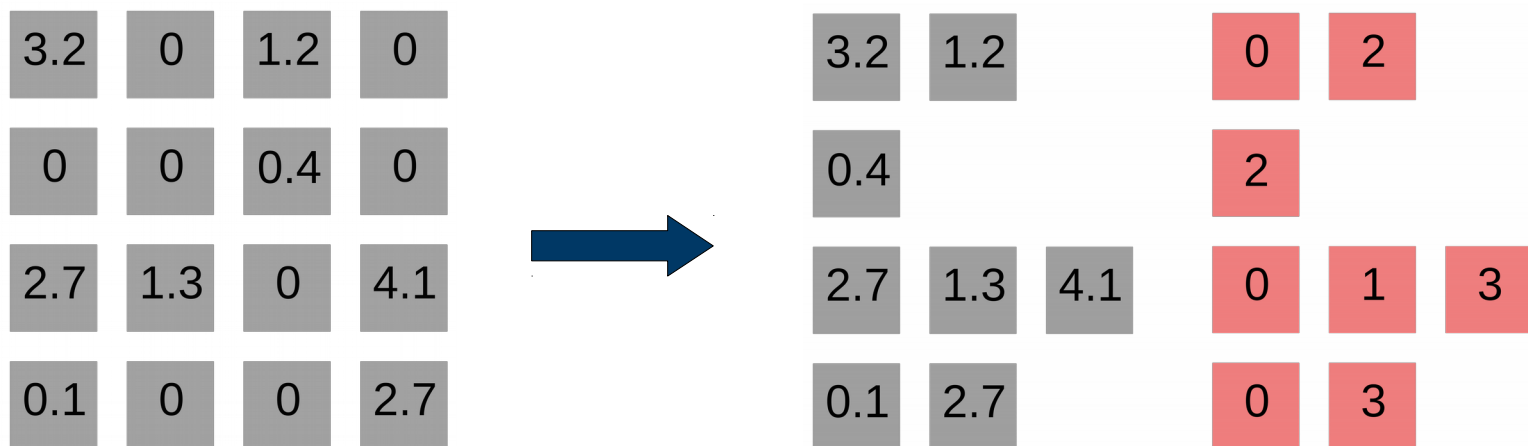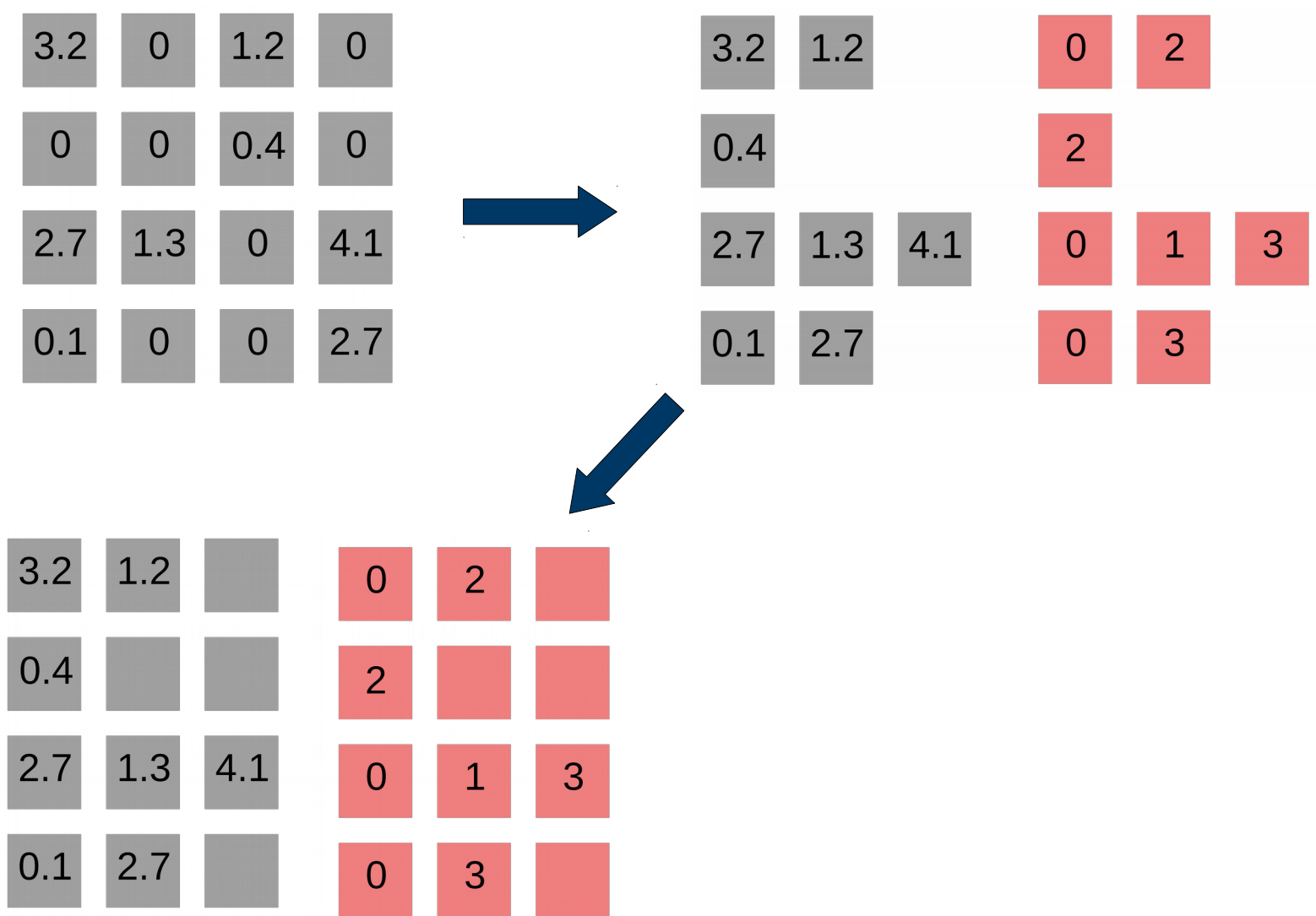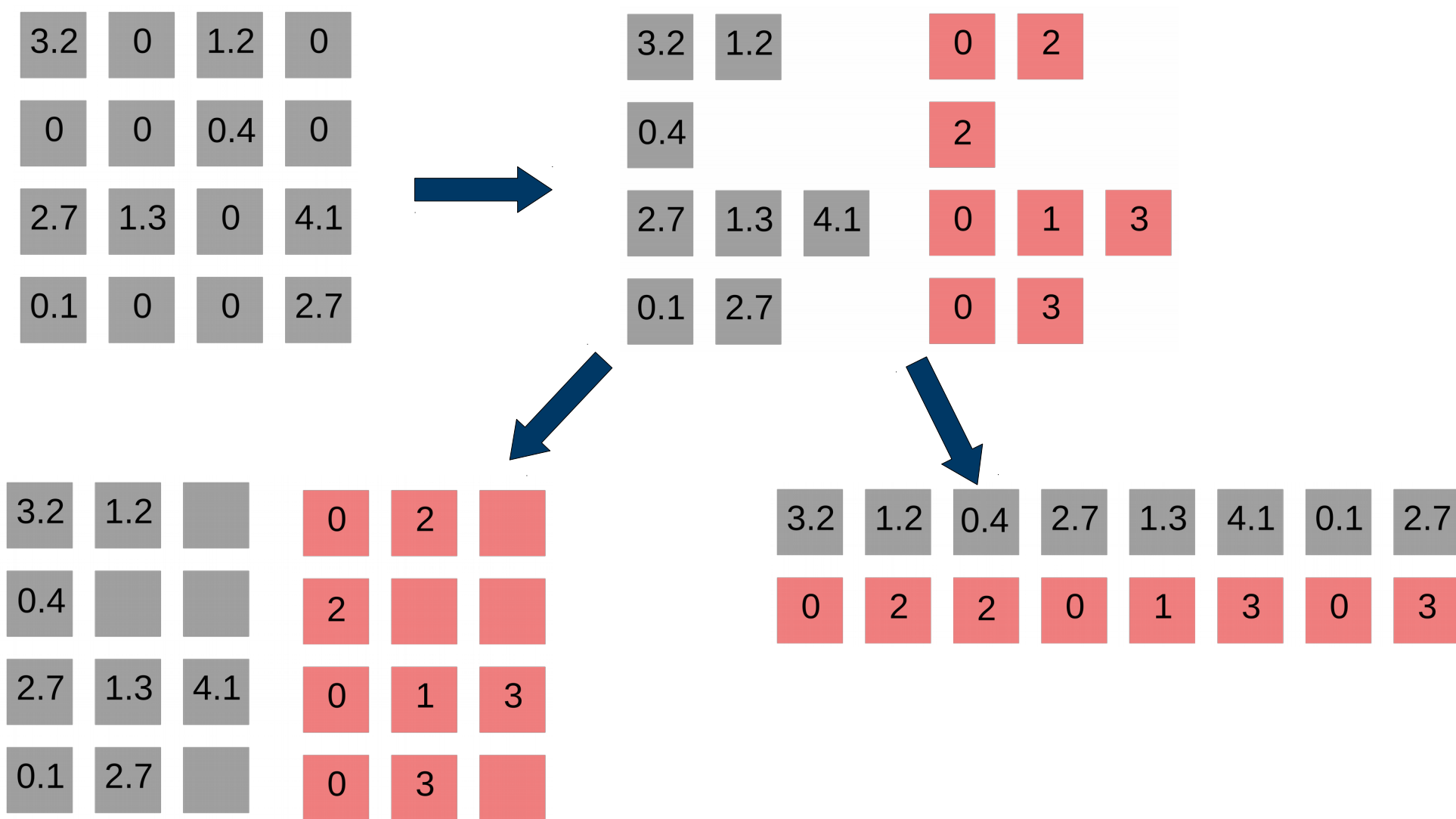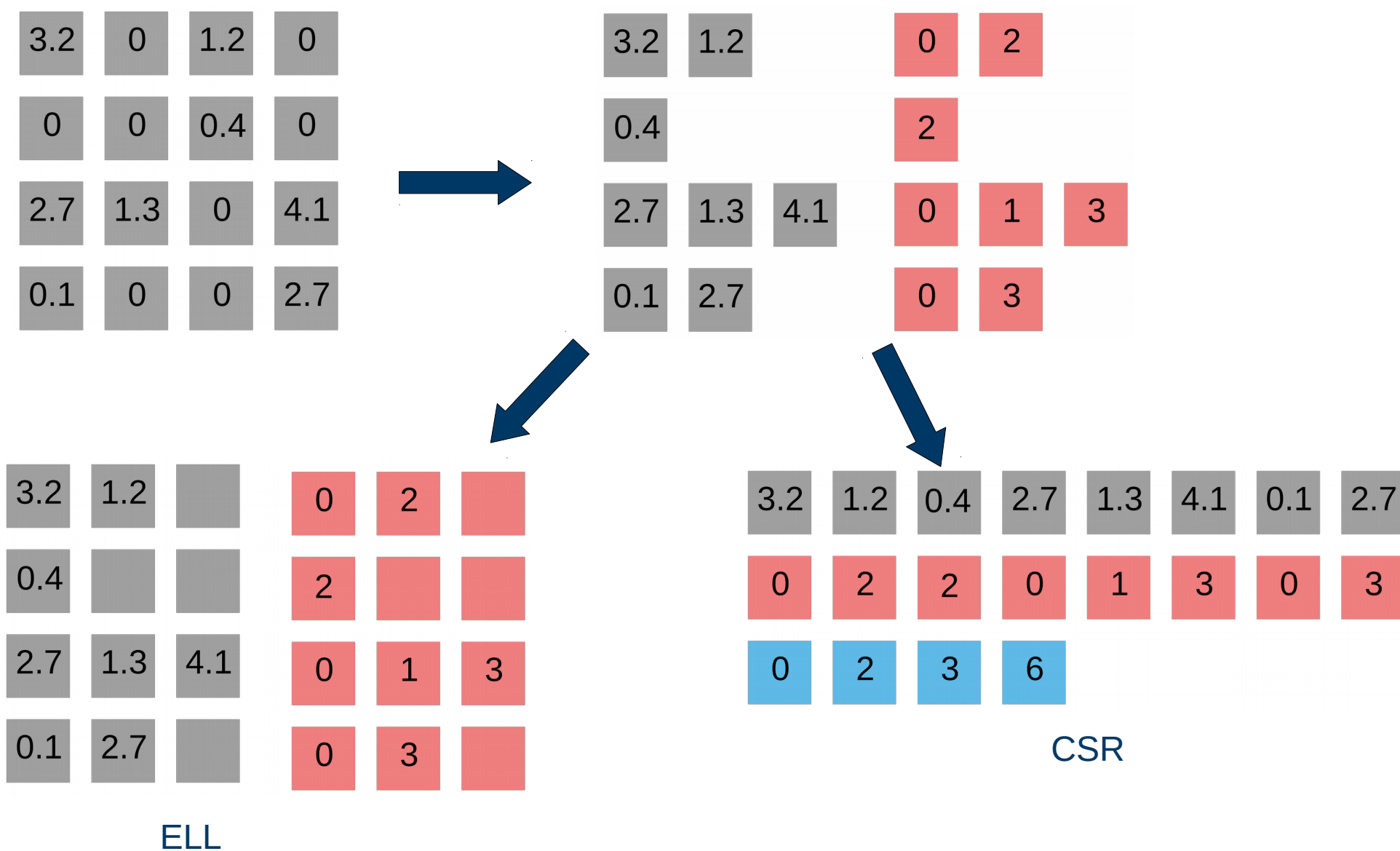| 3.2 | 1.2 | 0.4 | 2.7 | 1.3 | 4.1 | 0.1 | 2.7 |
| 0 | 0 | 1 | 2 | 2 | 2 | 3 | 3 | row |
| 0 | 2 | 2 | 0 | 1 | 3 | 0 | 3 | column |

COO

Sparse matrix formats

Sparse matrix formats

Sparse matrix formats

ELL

Sparse matrix formats

ELL

Sparse matrix formats

ELL

CSR

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

UNIVERSITAT JAUME I

Sparse matrix formats

… leads to CSC
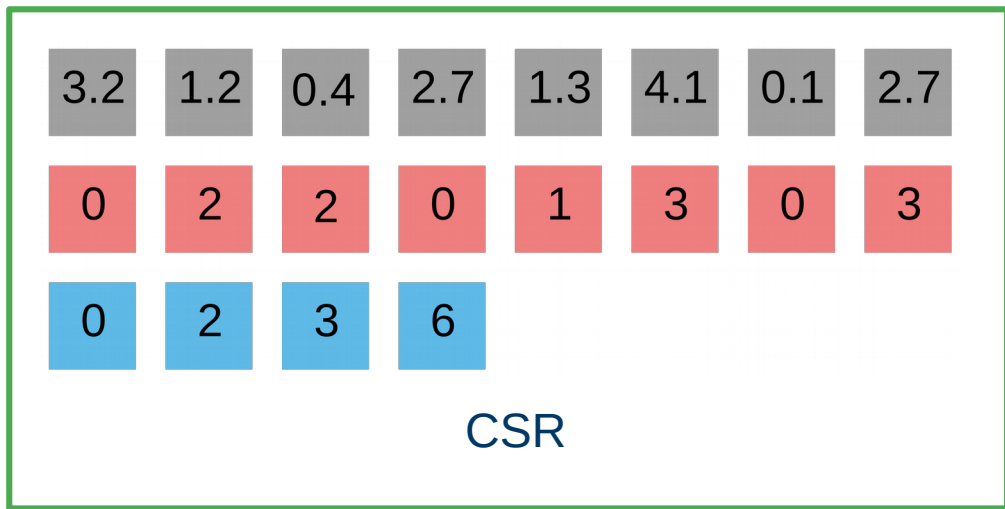
"Standard" approach

CSR

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) |

$$y := Ax$$

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

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) |

$$y := Ax$$

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] += val[j] * x [ colidx[j] ];
5    }
6  }
```

Bell & Garland '08
- parallelize outer loop

~ cuSPARSE SpMV

UNIVERSITAT JAUME I

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)

$$y := Ax$$

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

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)

$$y := Ax$$

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

Bell & Garland '08
- parallelize outer loop

~ cuSPARSE SpMV

Load imbalance!
Non-coalescence!

Specialized formats
- HYB – ELL + COO
- SELL-P – good memory access, parallelizes well
- … a few new ones every year

UNIVERSITAT JAUME I

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) |

$$y := Ax$$

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] += val[j] * x[ colidx[j] ];
5    }
6  }
```

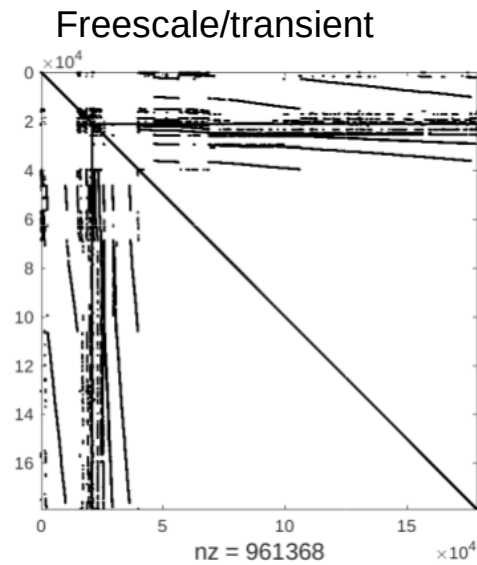Bell & Garland '08
- parallelize outer loop

~ cuSPARSE SpMV

Load imbalance!
Non-coalescence!

Specialized formats
- HYB – ELL + COO
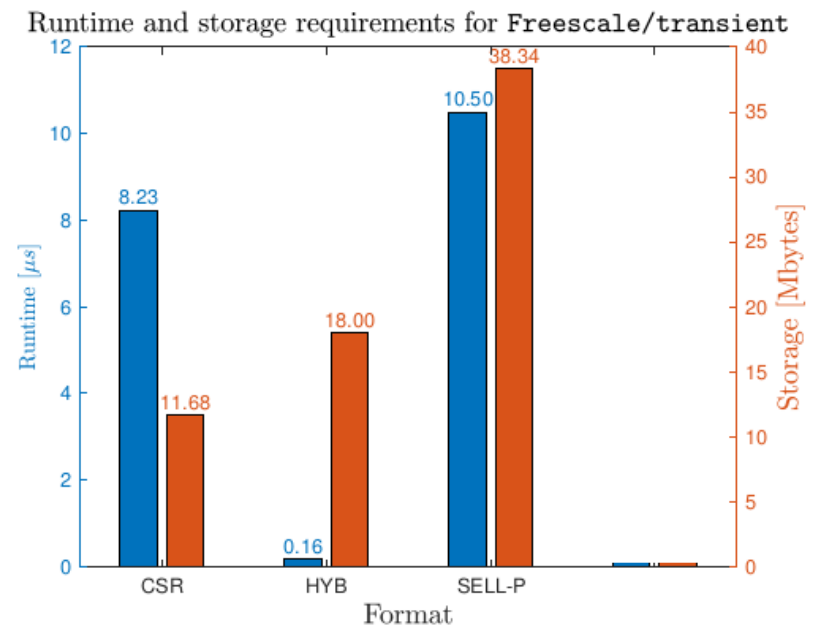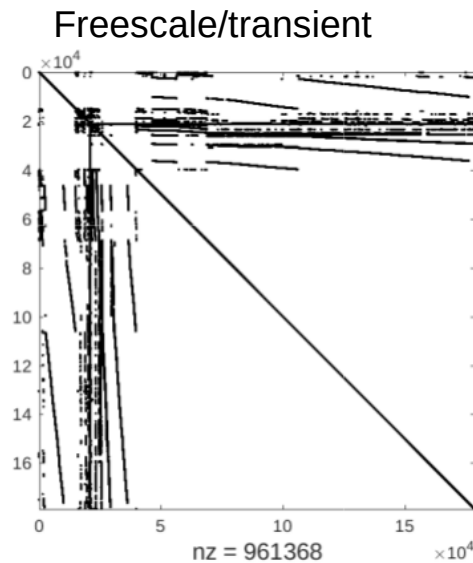- SELL-P – good memory access, parallelizes well
- … a few new ones every year

Library code – SpMV is not the only kernel!

Motivating example

Freescale/transient
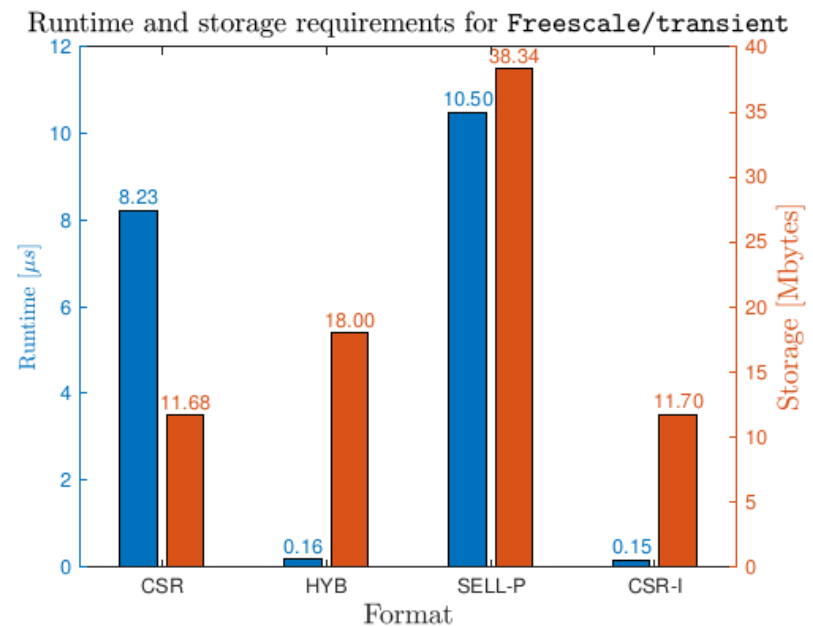
nz = 961368

Motivating example

Freescale/transient

Runtime and storage requirements for Freescale/transient

Can we do better than HYB using CSR?

Motivating example

Freescale/transient

Can we do better than HYB using CSR?

YES!

55x speedup

Motivating example

Freescale/transient

Runtime and storage requirements for Freescale/transient

55x speedup

Can we do better than HYB using CSR?

YES!

New **CSR-I** algorithm (and format).

Motivating example

Freescale/transient

Runtime and storage requirements for Freescale/transient
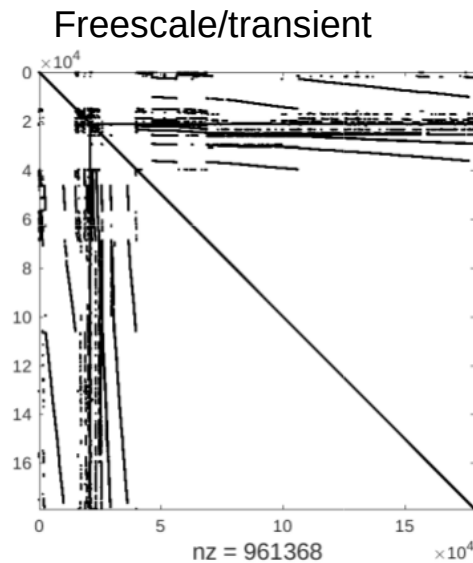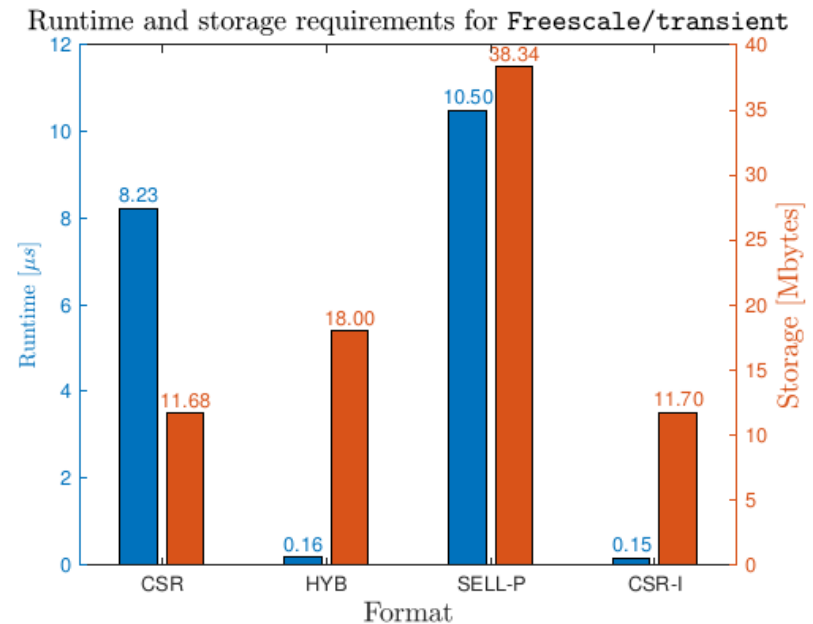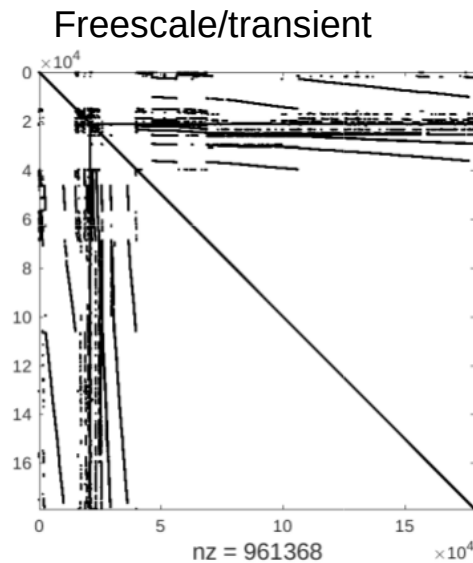
Can we do better than HYB using CSR?

55x speedup

YES!

New **CSR-I** algorithm (and format).

Only need to add a small amount of data to vanilla CSR. Add - but not modify the existing format!

UNIVERSITAT JAUME I

How to do it? By force!

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

How to do it? By force!

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

Merge the two loops into one.

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    int row = -1, next_row = 0, nnz = rowptr[m];
3    for (int i = 0; i < nnz; ++i) {
4        while (i >= next_row) next_row = rowptr[++row+1];
5        y[row] += val[i] * x[ colidx[i] ];
6  }}
```

UNIVERSITAT JAUME I

How to do it? By force!

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

Merge the two loops into one.

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    int row = -1, next_row = 0, nnz = rowptr[m];
3    for (int i = 0; i < nnz; ++i) {
4        while (i >= next_row) next_row = rowptr[++row+1];
5        y[row] += val[i] * x[ colidx[i] ];
6  }}
```

Split the loop into equal chunks.

```
1  const int T = thread_count;
2  void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
3    int row = -1, next_row = 0, nnz = rowptr[m];
4    for (int k = 0; k < T; ++k) {
5      for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
6        while (i >= next_row) next_row = rowptr[++row+1];
7        y[row] += val[i] * x[ colidx[i] ];
8  }}}
```

How to do it? By force!

```
1 void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2   for (int i = 0; i < m; ++i) {
3     for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4       y[i] += val[j] * x [ colidx[j] ];
5   }
6 }
```

Merge the two loops into one.

```
1 void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2   int row = -1, next_row = 0, nnz = rowptr[m];
3   for (int i = 0; i < nnz; ++i) {
4     while (i >= next_row) next_row = rowptr[++row+1];
5     y[row] += val[i] * x[ colidx[i] ];
6 }}
```

Split the loop into equal chunks.

```
1 const int T = thread_count;
2 void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
3   int row = -1, next_row = 0, nnz = rowptr[m];
4   for (int k = 0; k < T; ++k) {        Parallelize this!
5     for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
6       while (i >= next_row) next_row = rowptr[++row+1];
7       y[row] += val[i] * x[ colidx[i] ];
8 }}}
```

Race conditions!

How to do it? By force!

```
1 │ void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2 │   for (int i = 0; i < m; ++i) {
3 │     for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4 │       y[i] +=  val[j] * x [ colidx[j] ];
5 │   }
6 │ }
```

Merge the two loops into one.

```
1 │ void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2 │   int row = -1, next_row = 0, nnz = rowptr[m];
3 │   for (int i = 0; i < nnz; ++i) {
4 │     while (i >= next_row) next_row = rowptr[++row+1];
5 │     y[row] += val[i] * x[ colidx[i] ];
6 │ }}
```

Split the loop into equal chunks.

```
1 │ const int T = thread_count;
2 │ void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
3 │   int row = -1, next_row = 0, nnz = rowptr[m];
4 │   for (int k = 0; k < T; ++k) {        Parallelize this!
5 │     for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
6 │       while (i >= next_row) next_row = rowptr[++row+1];
7 │       y[row] += val[i] * x[ colidx[i] ];
8 │ }}}
```

- Use atomics
Race conditions! · Accumulate partial result into registers

How to do it? By force!

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] += val[j] * x [ colidx[j] ];
5    }
6  }
```

Merge the two loops into one.

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    int row = -1, next_row = 0, nnz = rowptr[m];
3    for (int i = 0; i < nnz; ++i) {
4      while (i >= next_row) next_row = rowptr[++row+1];
5      y[row] += val[i] * x[ colidx[i] ];
6  }}
```

Split the loop into equal chunks.

```
1  const int T = thread_count;
2  void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
3    int row = -1, next_row = 0, nnz = rowptr[m];
4    for (int k = 0; k < T; ++k) {          Parallelize this!
5      for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
6        while (i >= next_row) next_row = rowptr[++row+1];     State between outer
7        y[row] += val[i] * x[ colidx[i] ];                   loop iterations!
8  }}}
```

Race conditions!
- Use atomics
- Accumulate partial result into registers

How to do it? By force!

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    for (int i = 0; i < m; ++i) {
3      for (int j = rowptr[i]; j < rowptr[i+1]; ++j)
4        y[i] +=  val[j] * x [ colidx[j] ];
5    }
6  }
```

Merge the two loops into one.

```
1  void SpMV_CSR(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
2    int row = -1, next_row = 0, nnz = rowptr[m];
3    for (int i = 0; i < nnz; ++i) {
4      while (i >= next_row) next_row = rowptr[++row+1];
5      y[row] += val[i] * x[ colidx[i] ];
6  }}
```

Split the loop into equal chunks.

```
1  const int T = thread_count;
2  void SpMV_CSRI(int m, int *rowptr, int *colidx, float *val, float *x, float *y) {
3    int row = -1, next_row = 0, nnz = rowptr[m];
4    for (int k = 0; k < T; ++k) {                               Parallelize this!
5      for (int i = k*nnz / T; i < (k+1)*nnz / T; ++i) {
6        while (i >= next_row) next_row = rowptr[++row+1];        State between outer
7        y[row] += val[i] * x[ colidx[i] ];                      loop iterations!
8  }}}
```

- Use atomics
- Accumulate partial
  result into registers

Race conditions!

Precompute starting value
of "row" for each thread.

Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

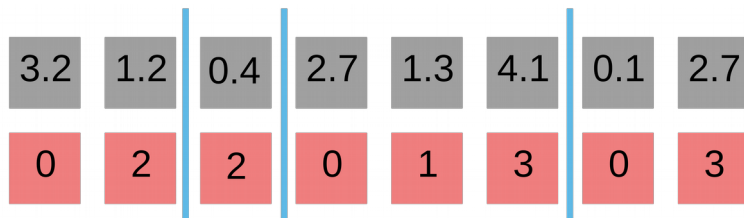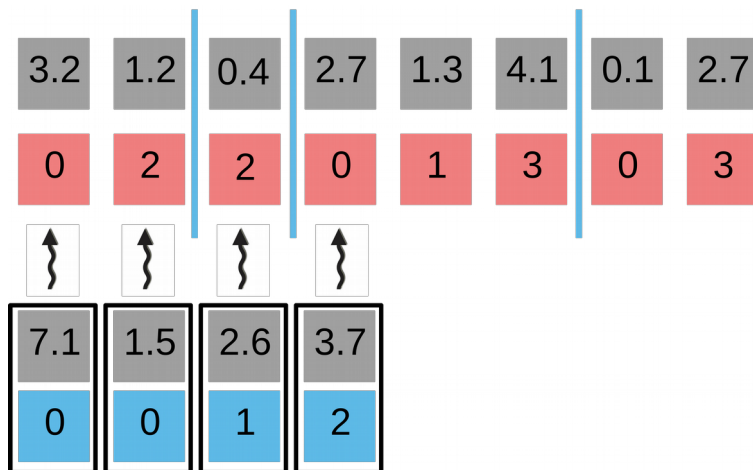Spreading out threads causes strided memory access.

Assign one warp per chunk.

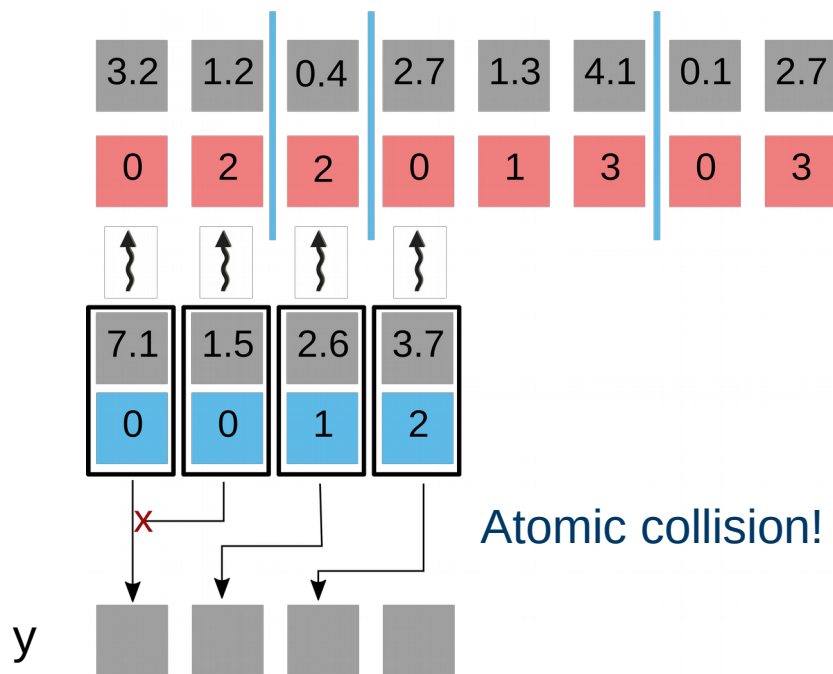# Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

Spreading out threads causes strided memory access.

Assign one warp per chunk.

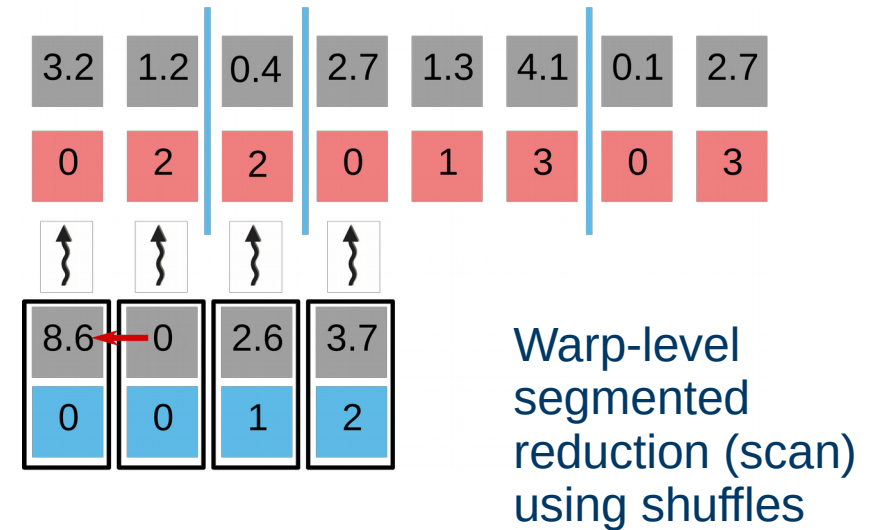| 3.2	1.2	0.4	2.7	1.3	4.1	0.1	2.7
0	2	2	0	1	3	0	3

Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

Spreading out threads causes strided memory access.

Assign one warp per chunk.

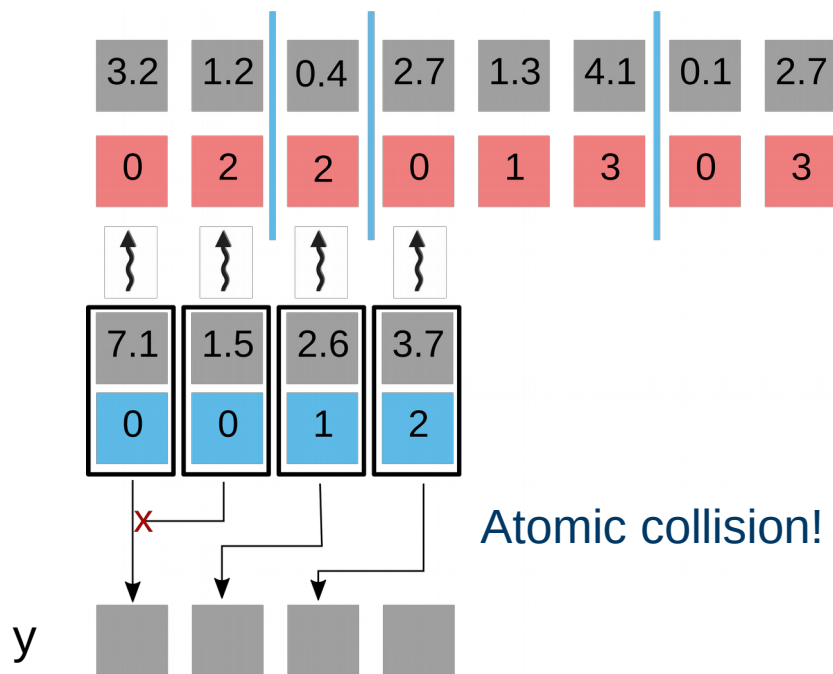# Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

Spreading out threads causes strided memory access.

Assign one warp per chunk.

Atomic collision!

Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

Spreading out threads causes strided memory access.

Assign one warp per chunk.

Atomic collision!

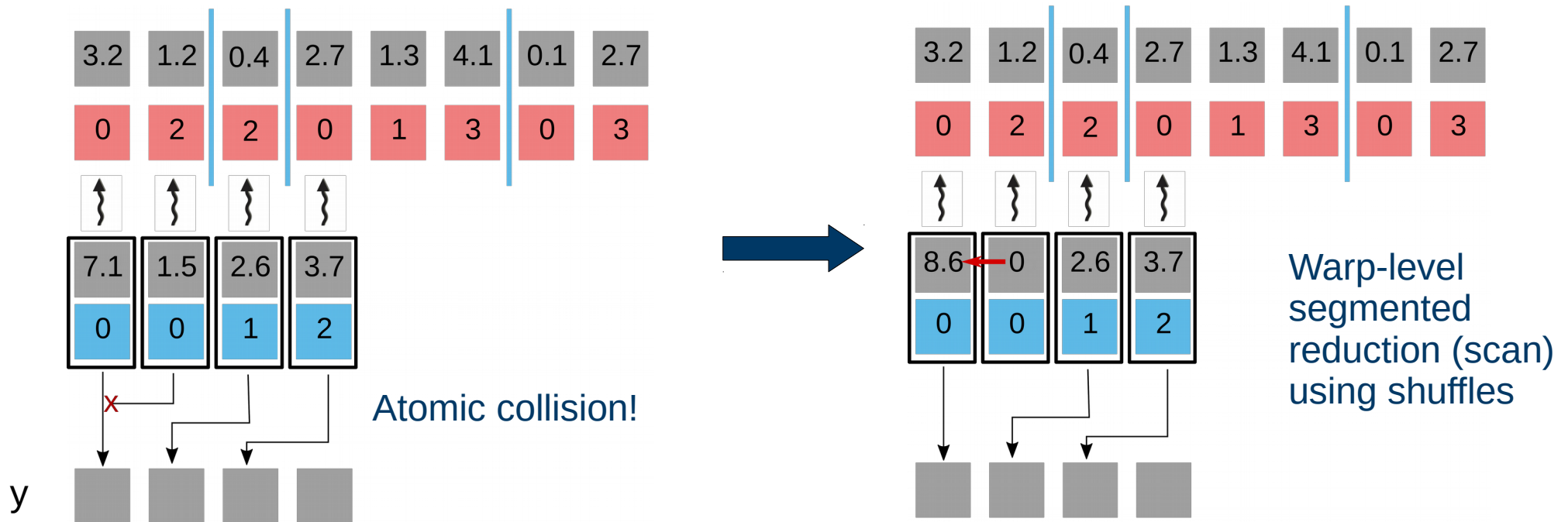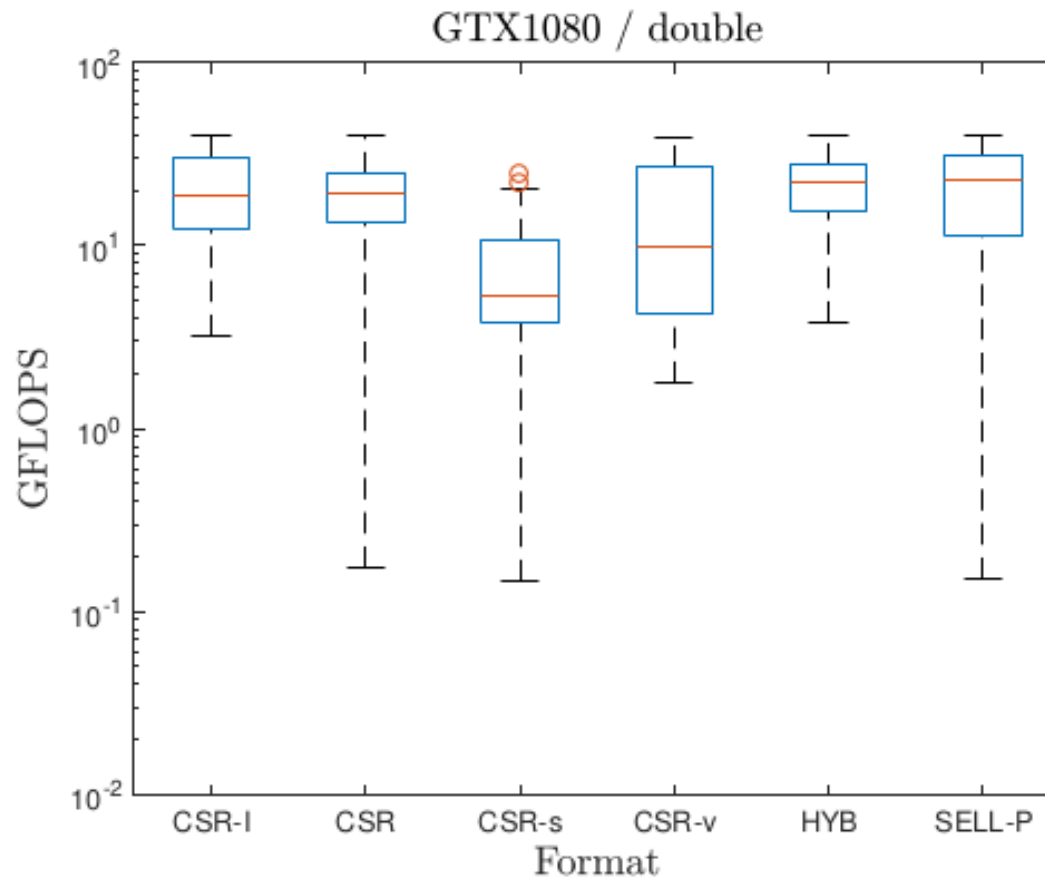Warp-level segmented reduction (scan) using shuffles

Getting good performance on GPUs

CUDA thread = 1 lane of a 32-wide SIMD unit (warp)

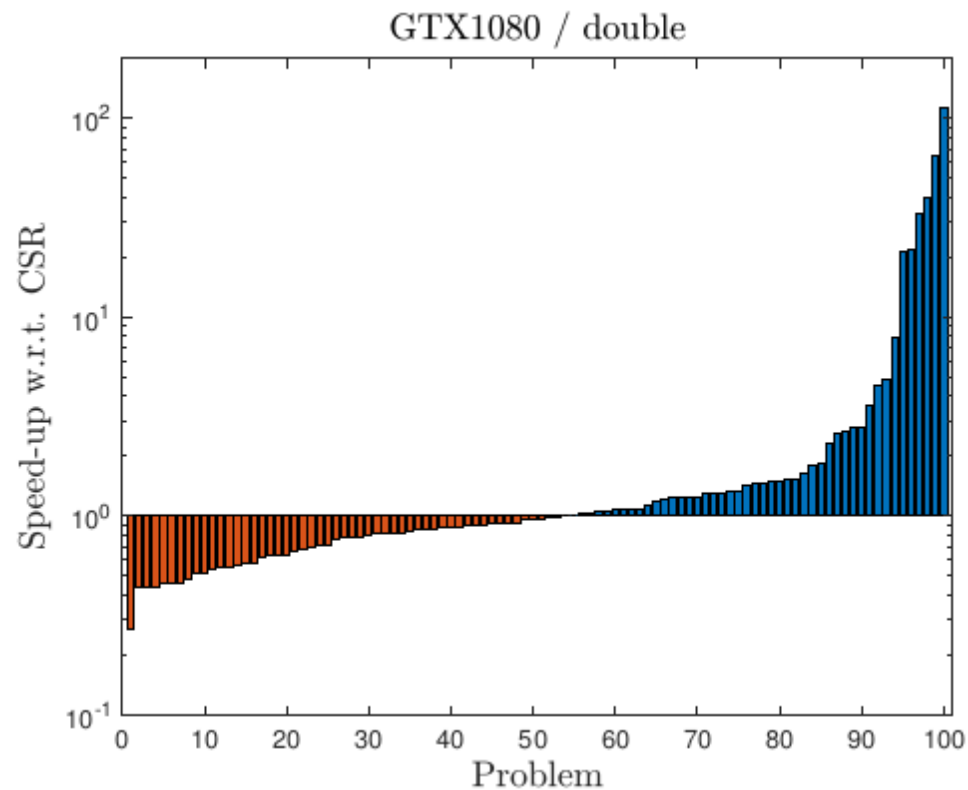Spreading out threads causes strided memory access.
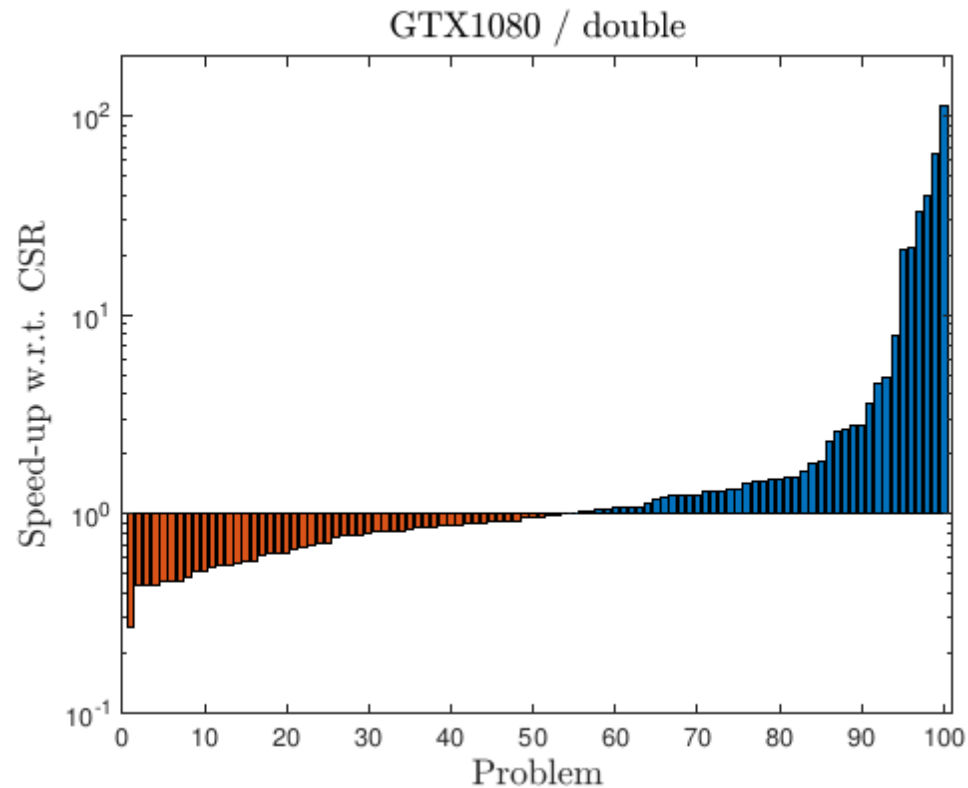
Assign one warp per chunk.

Atomic collision!

Warp-level segmented reduction (scan) using shuffles

UNIVERSITAT JAUME I

Performance of CSR-I

100 matrices from SuiteSparse

UNIVERSITAT JAUME I

Speed-up / slowdown over cuSPARSE CSR

Speed-up / slowdown over cuSPARSE CSR

GTX1080 / double

No format conversion!
- try both, and use the fastest later on!
- sometimes 1 cuSPARSE SpMV = 100 CSR-I SpMVs

UNIVERSITAT JAUME I

Choosing the winner a priori

CSR-I designed for irregular patterns

Choosing the winner a priori

CSR-I designed for irregular patterns

How to measure irregularity?

Deviation of row lengths from the mean.

UNIVERSITAT
JAUME I

Choosing the winner a priori

CSR-I designed for irregular patterns

How to measure irregularity?

Deviation of row lengths from the mean.

Is "5" regular or irregular?

Depends on the density of the matrix (mean #rows)

UNIVERSITAT
JAUME I

Choosing the winner a priori

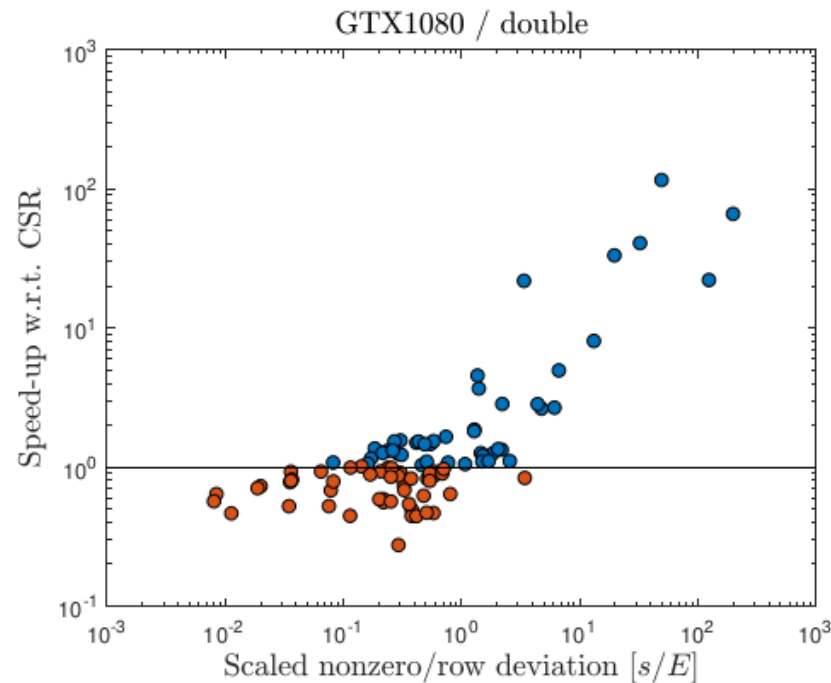CSR-I designed for irregular patterns   How to measure irregularity?

Deviation of row lengths from the mean.

Is "5" regular or irregular?

Depends on the density of the matrix (mean #rows)

Scatter plot of speedup vs normalized std. dev.

Choosing the winner a priori

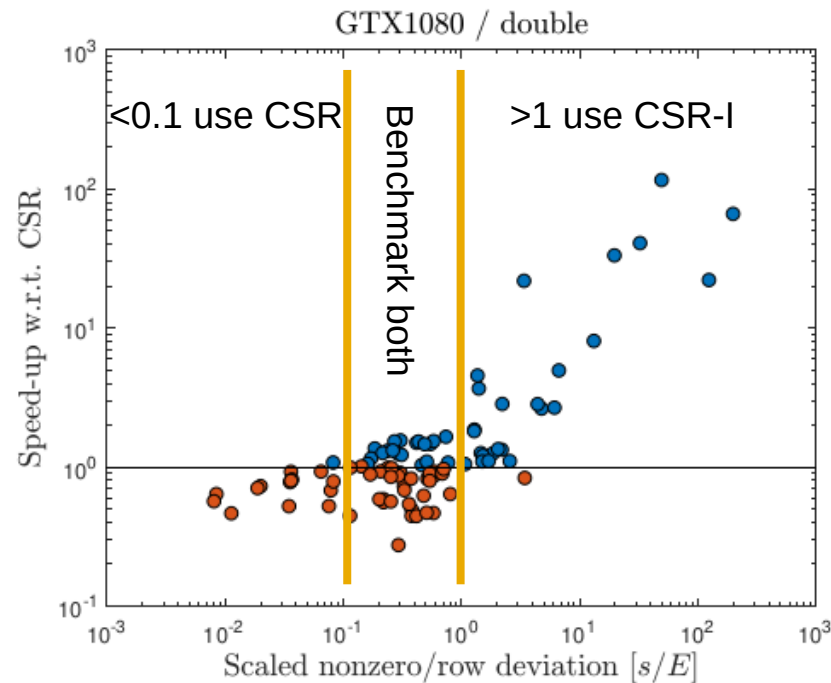CSR-I designed for irregular patterns   How to measure irregularity?

Deviation of row lengths from the mean.

Is "5" regular or irregular?

Depends on the density of the matrix (mean #rows)

Scatter plot of speedup vs normalized std. dev.

Conclusion

Use atomics and warp shuffles to tackle irregular sparsity patterns.

Determine a priori when CSR-I is faster than standard algorithm.
- keep cuSPARSE performance for regular patterns
- CSR-I for irregular ones

UNIVERSITAT
JAUME I

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/europar_2017

This research is based on a cooperation between Hartwig Anzt, Jack Dongarra (University of Tennessee), Goran Flegar and Enrique S. Quintana-Ortí (Universidad Jaume I).

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

UNIVERSITAT
JAUME·I