Exercises 8:
Sparse Linear Algebra

Today

- Sparse matrix-vector storage and multiplication

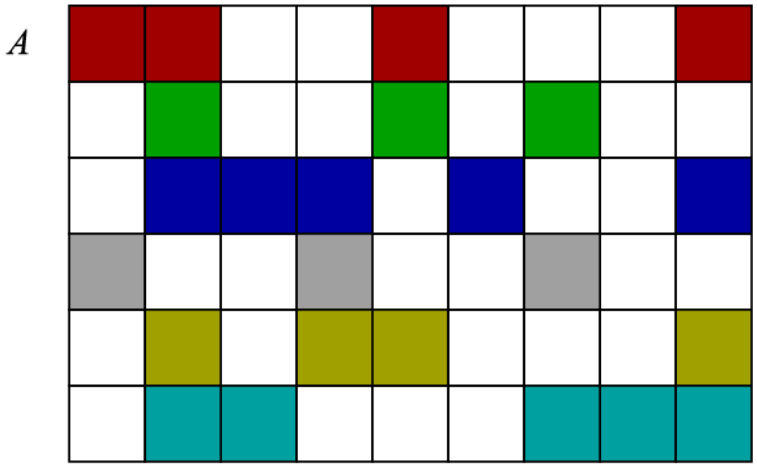
- Setup:
  - Download ex8.tar from Moodle, scp it to the cluster, and unpack the tar file
  - You should see csrmv.c and cscmv.c

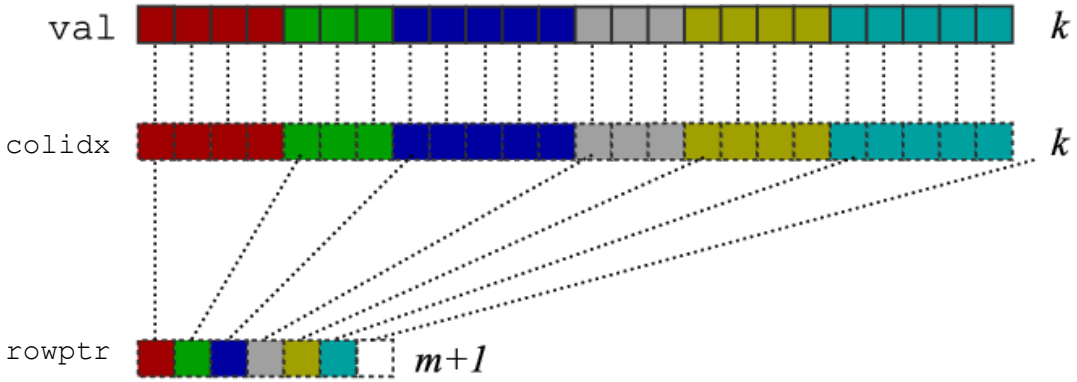
Sparse Matrix Storage Formats

- CSR: compressed sparse row
- CSC: compressed sparse column
- COO: coordinate format
- BCRS: block compressed sparse row
- DIA or CDS: Diagonal storage
- SKS: skyline storage
- ELL

- Many, many others

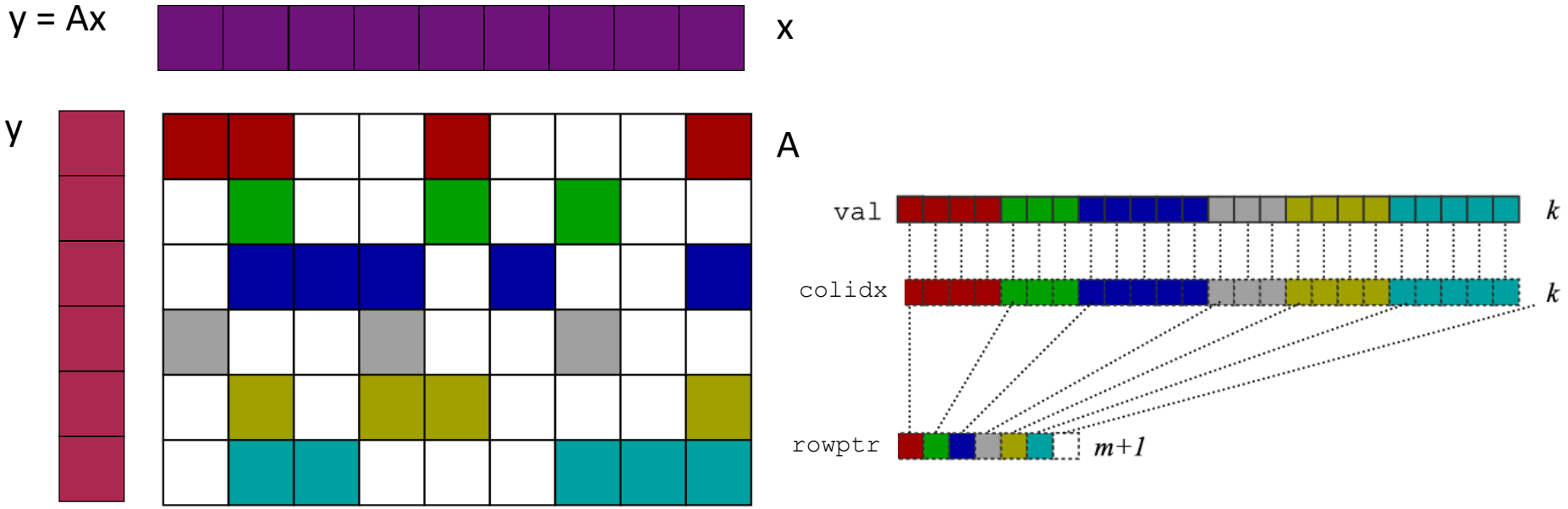
Compressed Sparse Row (CSR) Storage





- CSR has:
  - Array of the nonzero values (val) of size nnz = number of nonzeros
  - Array of the column indices (colidx) for each value of size nnz
  - Array of row start pointers (rowptr) of size n = number of rows

SpMV with Compressed Sparse Row (CSR)

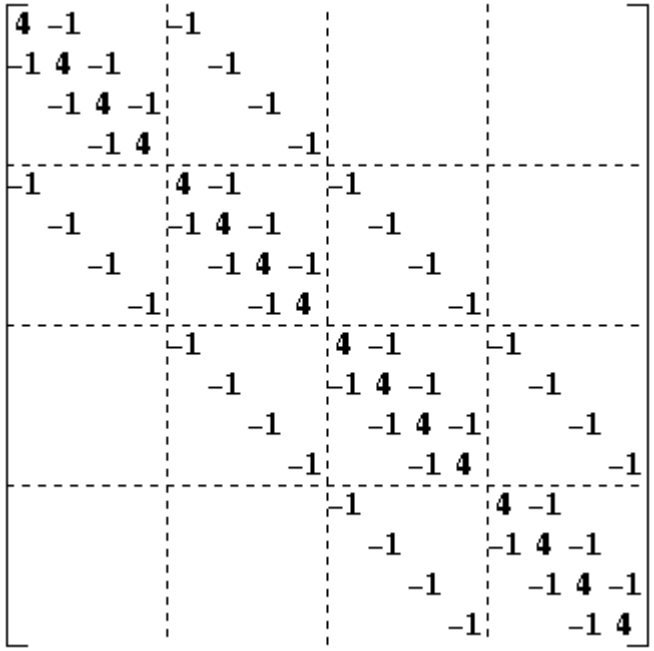


Matrix-vector multiply kernel: y(i) ← y(i) + A(i,j)*x(j)

```
for each row i
   for k=rowptr[i] to rowptr[i+1]-1 do
       y[i] = y[i] + val[k]*x[colidx[k]]
```

Task 1: Implement CSR-MV

- Open the file csrmv.c
- This file takes 2 command line inputs: n and Ntrials
- This file creates a 2D Poisson matrix (a square matrix with dimension N, where $N = n^2$) in CSR format and will multiply it by a vector of all ones
- Measures the time to do the matrix-vector product, averaged over Ntrials trials

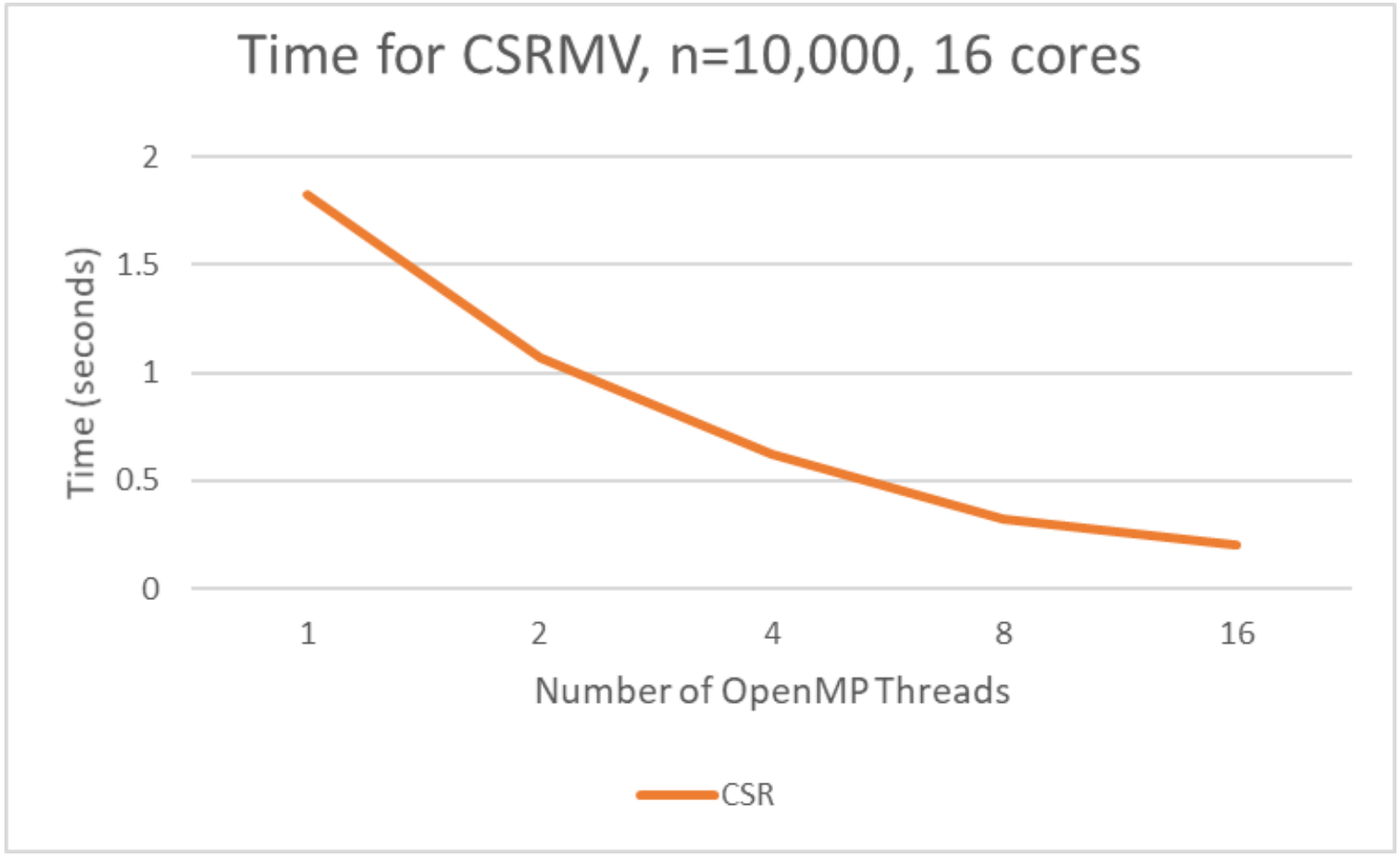


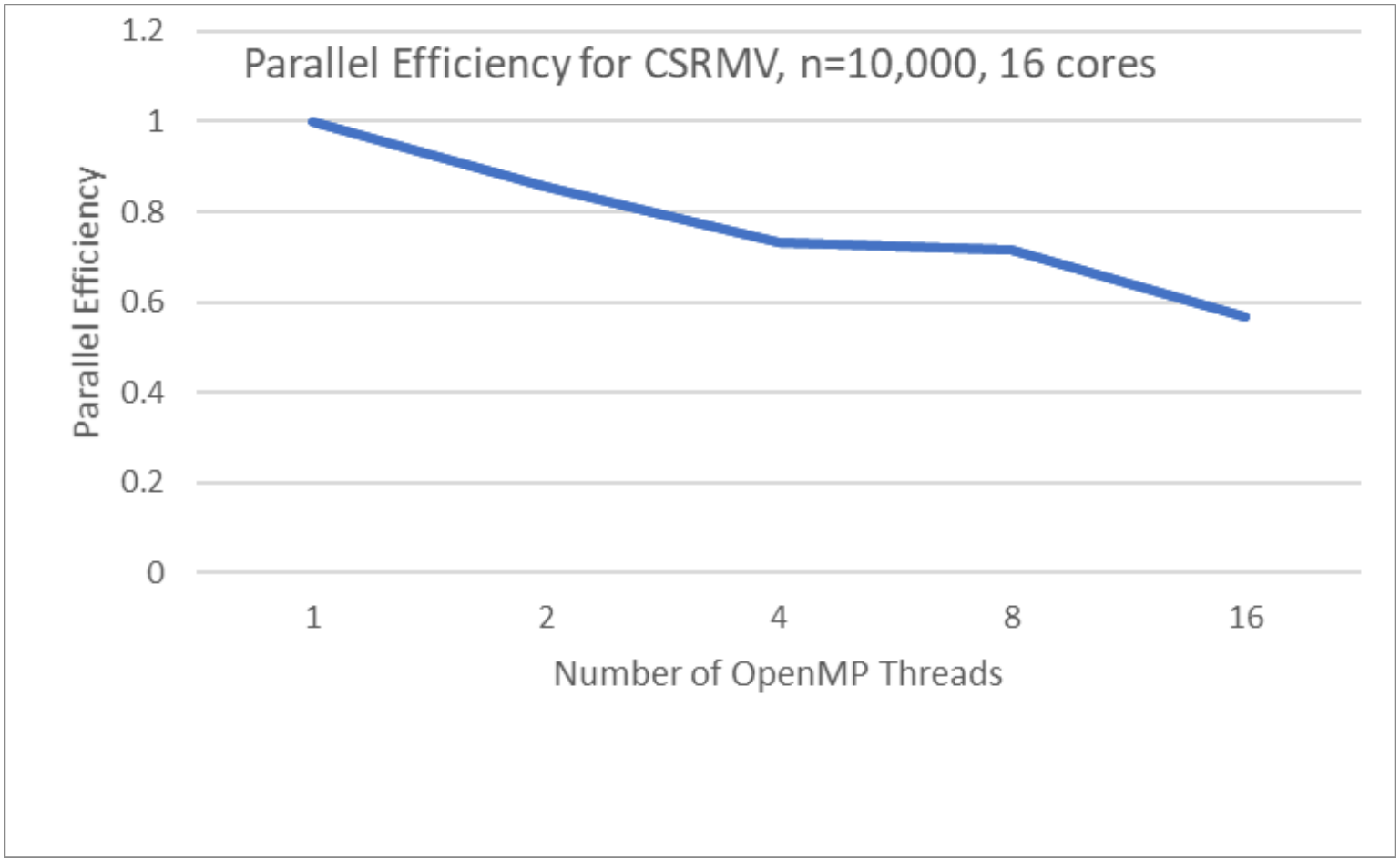
$n = 4$

- Your task: implement the code that does the CSR matrix vector multiplication
- To compile: gcc –fopenmp –o csrmv csrmv.c
- To run: ./csrmv n Ntrials

Task 2: OpenMP Parallelization

- Parallelize the CSR matrix-vector multiplication part using OpenMP
  - (should be simple)
- Test your implementation:
- Measure runtimes (using fixed problem size) for different numbers of threads (e.g., p=1,2,4,8,16 threads)
  - Try a few different problem sizes (small, too big for cache, etc.)
- You can use the example batch script job.sh (or just run on r3d3)


Time for CSRMV, n=10,000, 16 cores
Time (seconds)
2
1.5
1
0.5
0
1
2
4
8
16
Number of OpenMP Threads
CSR



Parallel efficiency = (sequential time)/(p*parallel time)

Compressed Sparse Column MV

$$\begin{bmatrix}\times\\ \times\\ \times\\ \times\\ \times\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 2\\ 2 & 3 & 2 & 0 & 3\\ 0 & 1 & 0 & 1 & 1\\ 0 & 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}\times\\ \times\\ \times\\ \times\\ \times\end{bmatrix}$$

```
val =      [1,2,1,3,1,2,1,1,1,2,3,1]
rowidx =   [0,2,1,2,3,2,4,0,3,1,2,3]
colptr =         [0,2,5,7,9,25]
```

Matrix-vector multiply kernel: y(i) ← y(i) + A(i,j)*x(j)

```
for each col i
   for k=colptr[i] to colptr[i+1]-1 do
      y[rowidx[k]] += val[k]*x[i]
```

Task 3: CSC Format

- In cscmv.c, we have given you an implementation of CSC matrix-vector multiply, parallelized with OpenMP, but there is an error in the way we have used OpenMP.
- Find and fix the error!

Task 3: CSC Format

- In cscmv.c, we have given you an implementation of CSC matrix-vector multiply, parallelized with OpenMP, but there is an error in the way we have used OpenMP.
- Find and fix the error!

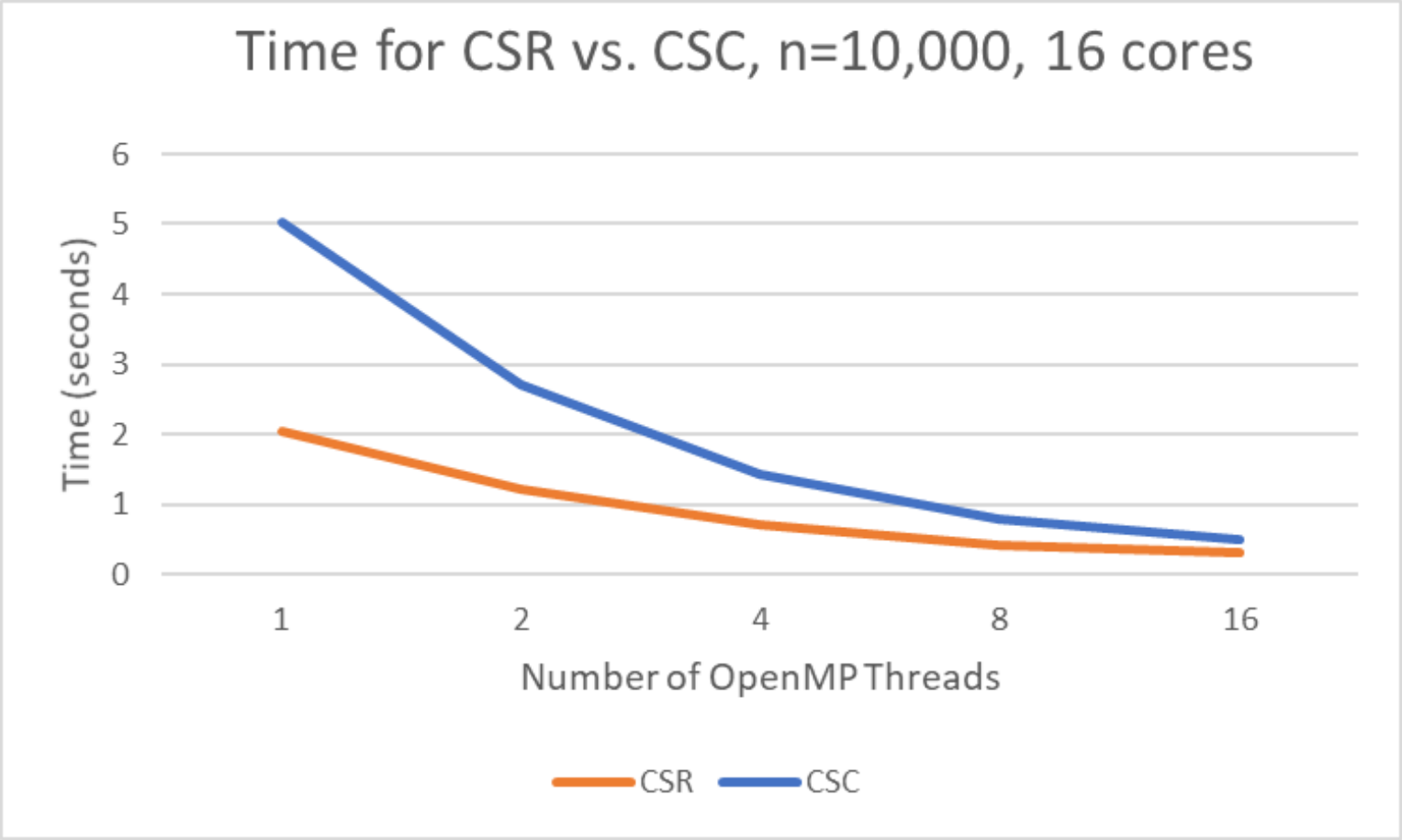
- Hints: Write code that prints out the solution y=Ax
  - Given that x is a vector of all 1's, you know what the vector y=Ax should look like
  - Try running on a small problem size, for different numbers of threads

Task 4: CSR vs. CSC

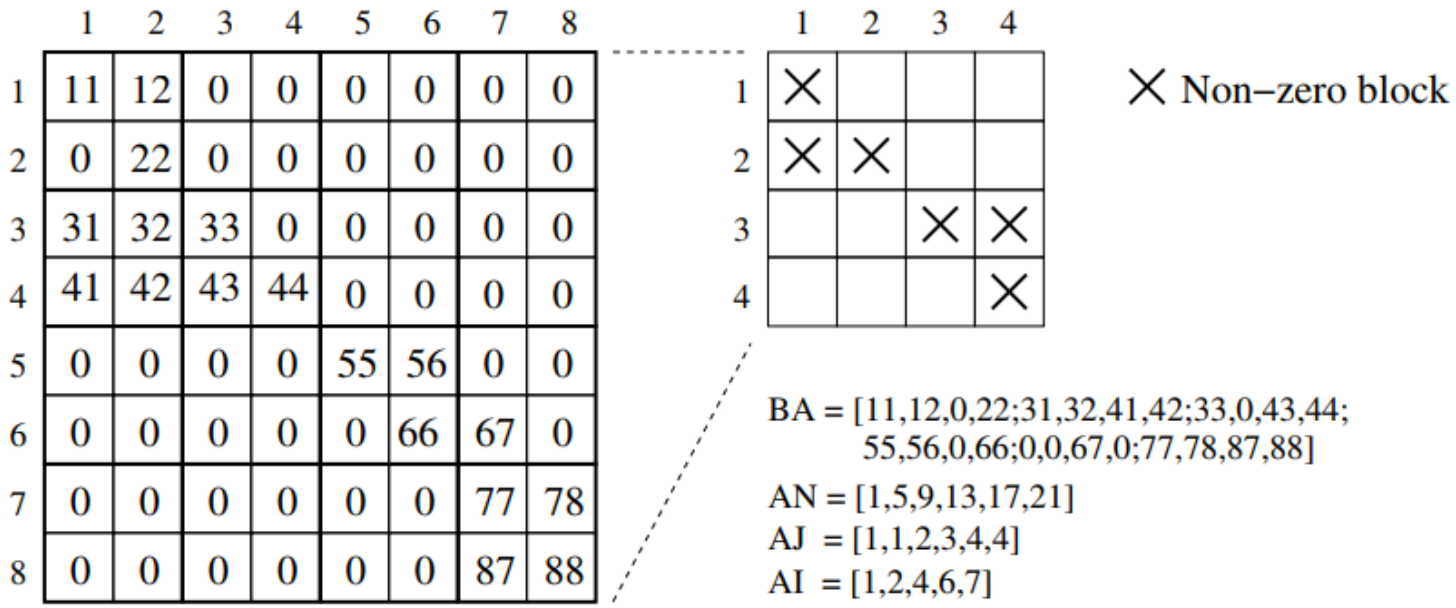
- With the corrected CSC code, compare the performance using CSR and CSC formats.

- Choose a problem size
- Measure runtimes for csrmv and cscmv for different numbers of threads (e.g., p=1,2,4,8,16,32 threads)
- Which is faster? Why?

Example output

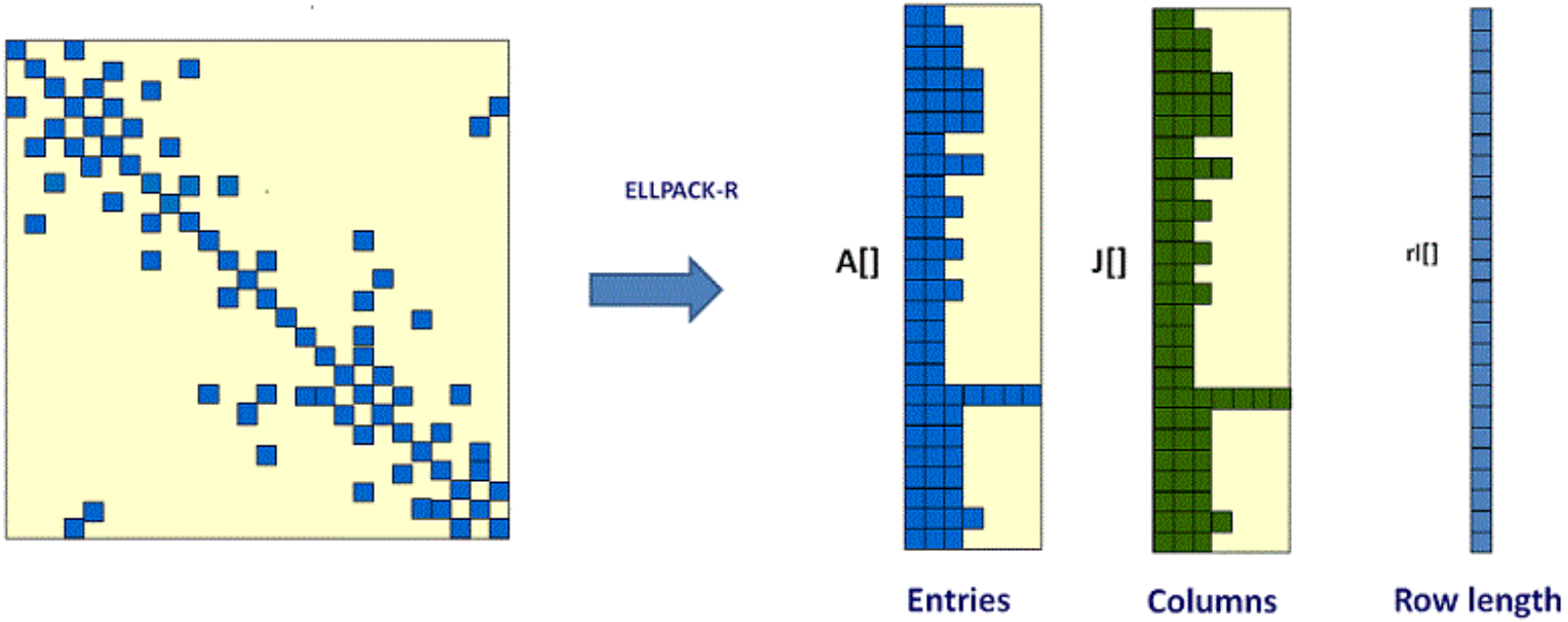


Other Sparse Matrix Formats



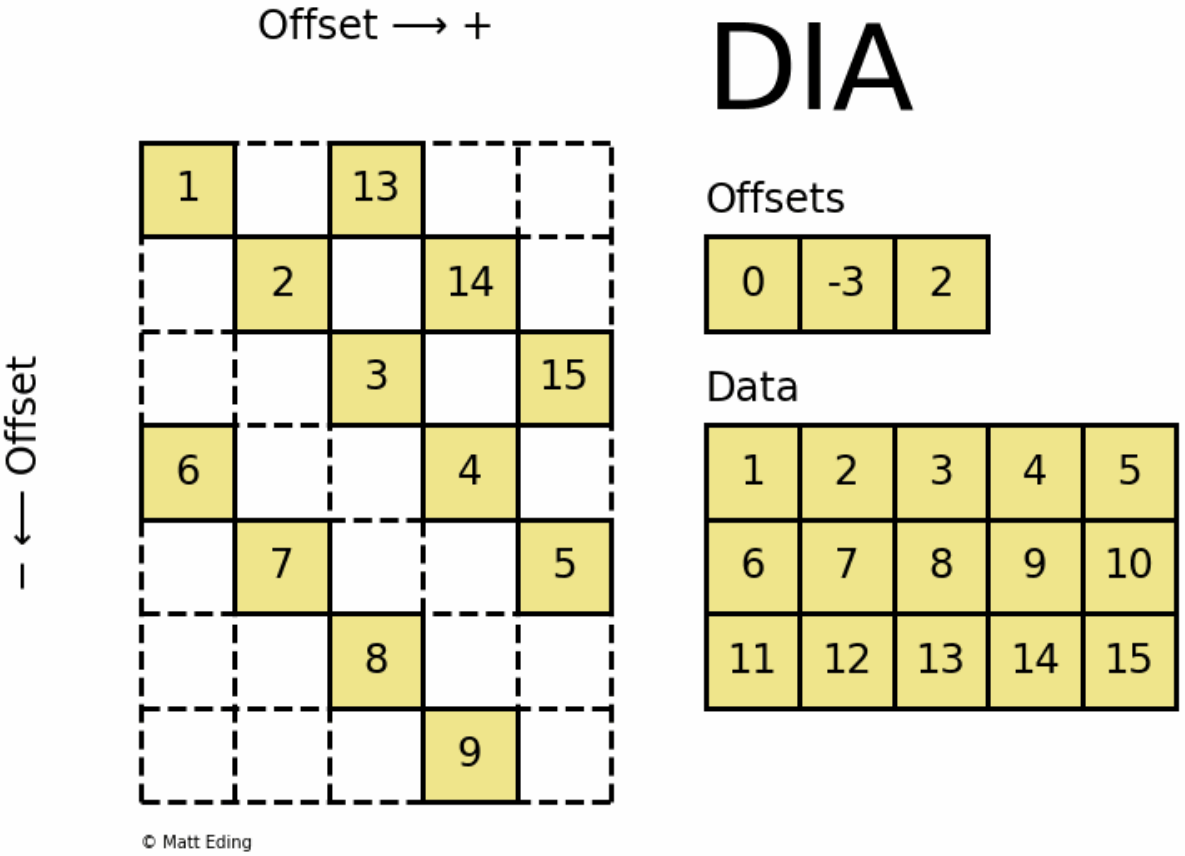
Block Compressed Row Storage Format

Other Sparse Matrix Formats



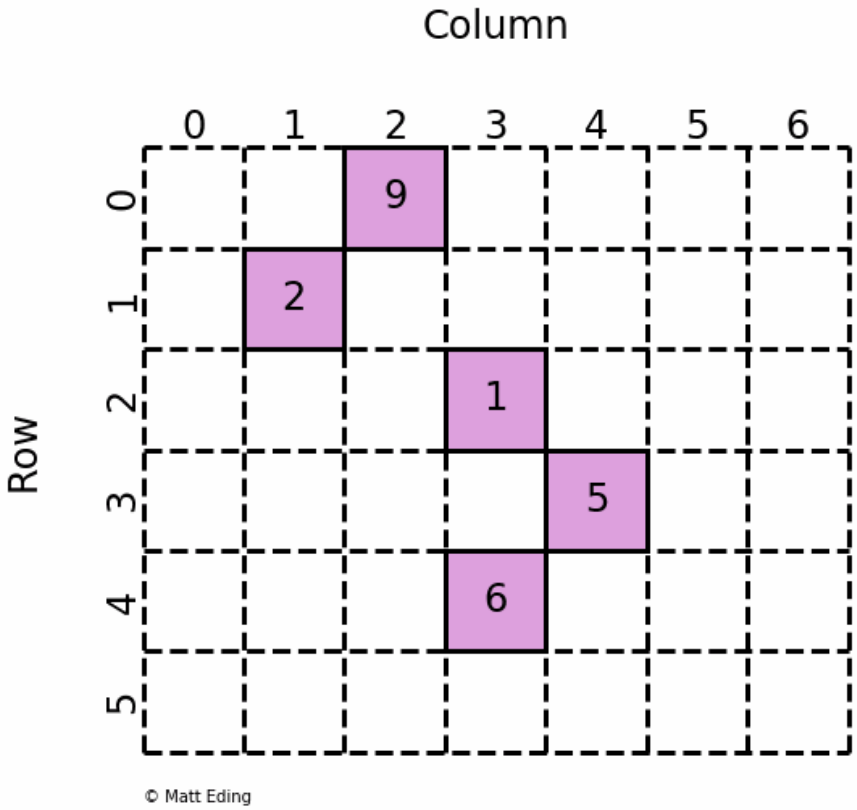
https://sites.google.com/site/mcfastsparse/

Other Sparse Matrix Formats



https://matteding.github.io/2019/04/25/sparse-matrices/

Other Sparse Matrix Formats



COO

Row

1	3	0	2	4

Column

1	4	2	3	3

Data

2	5	9	1	6

https://matteding.github.io/2019/04/25/sparse-matrices/

Current Research

- New storage formats
  - For SIMD architectures
    - Kreutzer, Moritz, et al. "A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units“, *SIAM SISC,* 36.5 (2014): C401-C423.
  - For GPUs
    - Kreutzer, Moritz, et al. "Sparse matrix-vector multiplication on GPGPU clusters: A new storage format and a scalable implementation“, *IPDPS*, 2012.
  - For portability across architectures
    - Liu, Weifeng, and Brian Vinter. "CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication“, *Supercomputing, 2015.*

- Using Machine Learning to select the best sparse matrix format:
  - Zhao, Yue, et al. "Bridging the gap between deep learning and sparse matrix format selection“, PPoPP 2018.
  - Benatia, Akrem, et al. "Sparse matrix format selection with multiclass SVM for SpMV on GPU“, *ICPP, 2016.*

High-Performance Libraries Implementing Sparse LA

- Trilinos (Sandia National Lab), https://trilinos.github.io/
- PETSc, Portable, Extensible Toolkit for Scientific Computation (Argonne National Lab), https://www.mcs.anl.gov/petsc/
- Many storage formats, many numerical algorithms (LU, CG, SPMV, ...) implemented, tested and ready to use!