Eliminating Irregular Patterns for Compressed Sparse Matrix Primitives on Manycore Processing

Hao Wang (hwang121@vt.edu)

VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

Summary

- Include
  - CSR data structure
  - Parallel CSR Transpose on Multicore and Manycore (Xeon Phi)
- Not included
  - Parallel CSR spMV

Why Primitives of Sparse Matrix

- Core computational steps in graph processing and deep learning
  - Dense matrix
  - Sparse matrix
- Case studies: after applying deep learning
  - On the image recognition, **Baidu** got the best result of ImageNet Competition

CSR Data Structure

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

- Compressed Sparse Row (CSR)

val	a	b	c	d	e	f	g	h	i	j	k	l	m	n
col	1	3	5	2	0	1	5	4	0	6	2	2	1	7
row	0	3	4	7	8	10	11	12	14					

- CSR Storage usage:

  2nnz + n + 1

  nnz: number of non-zero data

  n: number of row

- Number of non-zero data at row i:

  k = row[i+1] – row[i]

- Column index at row i:

  col[row[i]]

  …

  col[row[i] + k - 1]

- Similar to the value

CSR Transpose from A to A^T

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

val	a	b	c	d	e	f	g	h	i	j	k	l	m	n
col	1	3	5	2	0	1	5	4	0	6	2	2	1	7
row	0	3	4	7	8	10	11	12	14					

$A \rightarrow A^T$

val	e	i	a	f	m	d	k	l	b	h	c	g	j	n
row	2	4	0	2	7	1	5	6	0	3	0	2	4	7
col	0	2	5	8	9	10	12	13	14					

Observations -- Parallel CSR Transpose

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

- NNZ for each *row* in A^T (NNZ for each column of *A*) is independent with *row* in *A*
  - e.g., the number of elements for row 1 in A^T (column 1 in *A*) is 3, a, f, m
- Relative offsets for NZ in *val* (with the same column index) of *A* will keep the same relative offsets in A^T
  - e.g., a, f, m with relative offsets 0, 1, 2, will keep the same relative offset in A^T
- Scattered elements of *val* in *A* will be gathered to *val* in A^T

val	a	b	c	d	e	f	g	h	i	j	k	l	m	n
col	1	3	5	2	0	1	5	4	0	6	2	2	1	7
row	0	3	4	7	8	10	11	12	14					

$A \rightarrow A^T$

val	e	i	a	f	m	d	k	l	b	h	c	g	j	n
row	2	4	0	2	7	1	5	6	0	3	0	2	4	7
col	0	2	5	8	9	10	12	13	14					

Design Objectives

- Keep balance in multiple parallel threads
  - Divide workload based on NNZ instead of row or column number

Parallel CSR Transpose

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

- Divide *col* evenly and do accumulation on column index (intra-thread offset)

to	0	1	1	1	0	1	0	0
t1	1	1	0	0	1	1	0	0
t2	1	0	2	0	0	0	1	0
t3	0	1	0	0	0	0	0	1

	to			t1			t2					t3		
val	a	b	c	d	e	f	g	h	i	j	k	l	m	n
col	1	3	5	2	0	1	5	4	0	6	2	2	1	7
row	0	3	4	7	8	10	11	12	14					

	to			t1			t2					t3		
intra	0	0	0	0	0	0	0	0	0	0	0	1	0	0

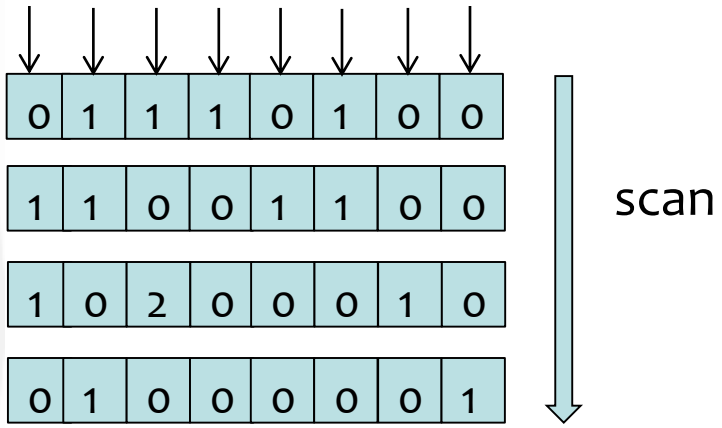
Design Objectives

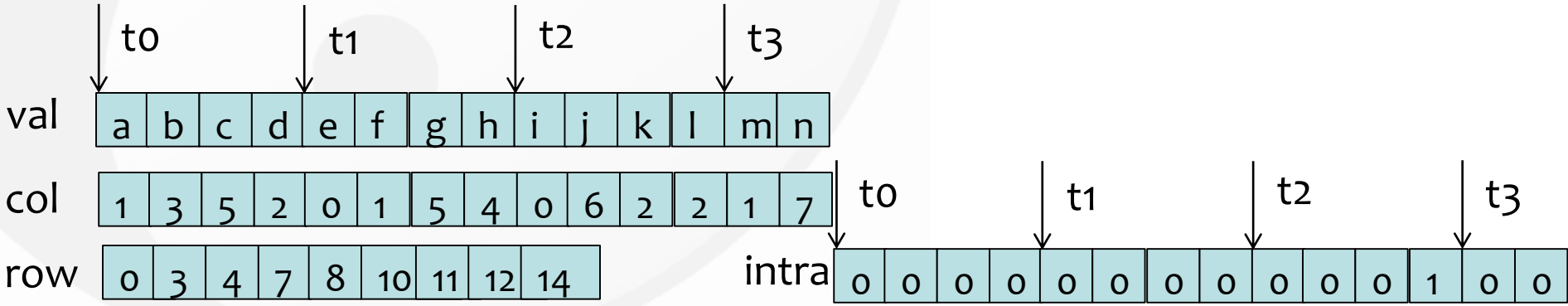
- Keep balance in multiple parallel threads
  - Divide workload based on NNZ instead of row or column number
- Change multiple data access to offset calculation
  - Compute the offset in A^T for each NZ in A

Parallel CSR Transpose

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

- Calculate prefix sum (inter-thread offset)

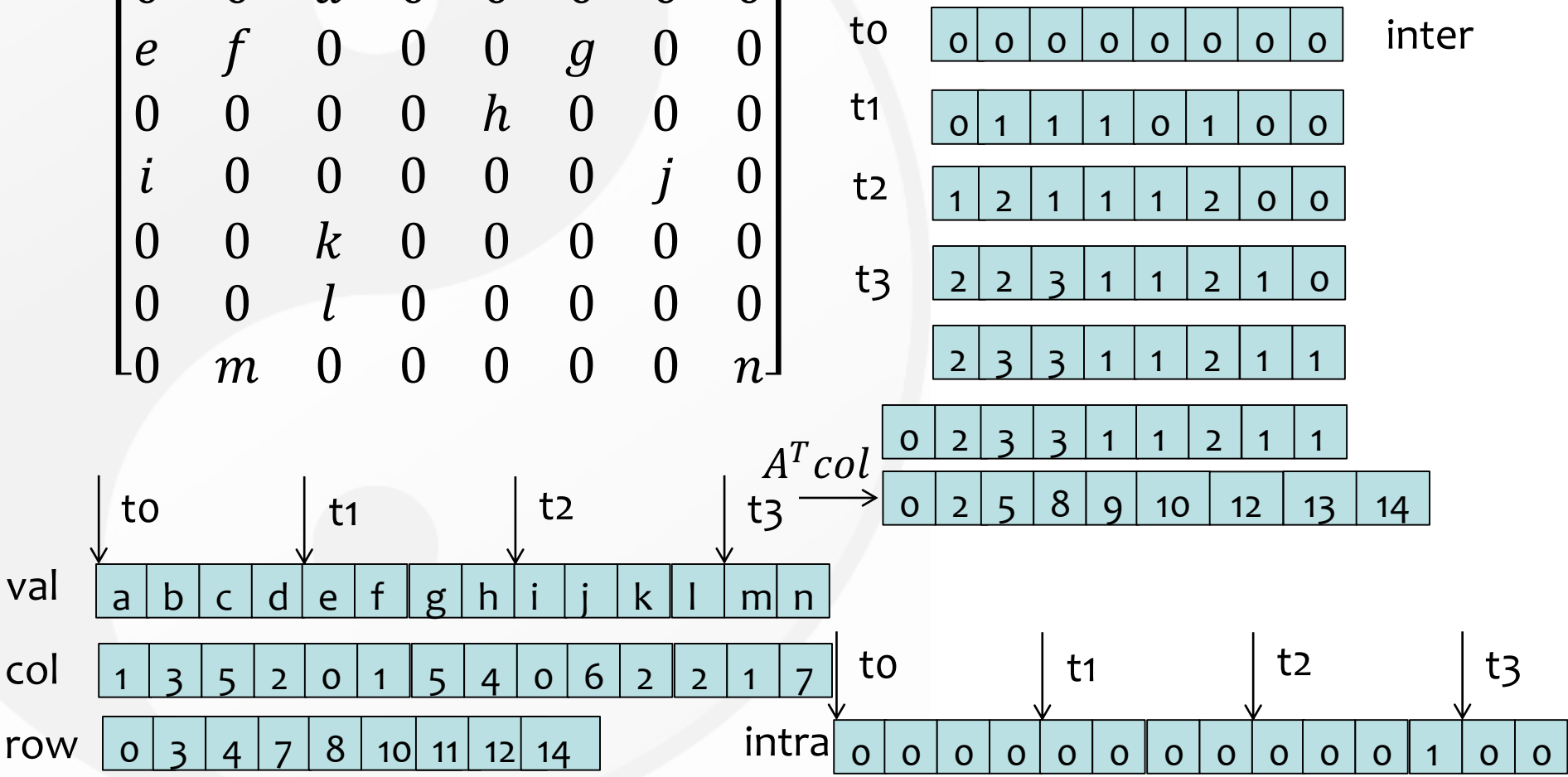




Parallel CSR Transpose

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

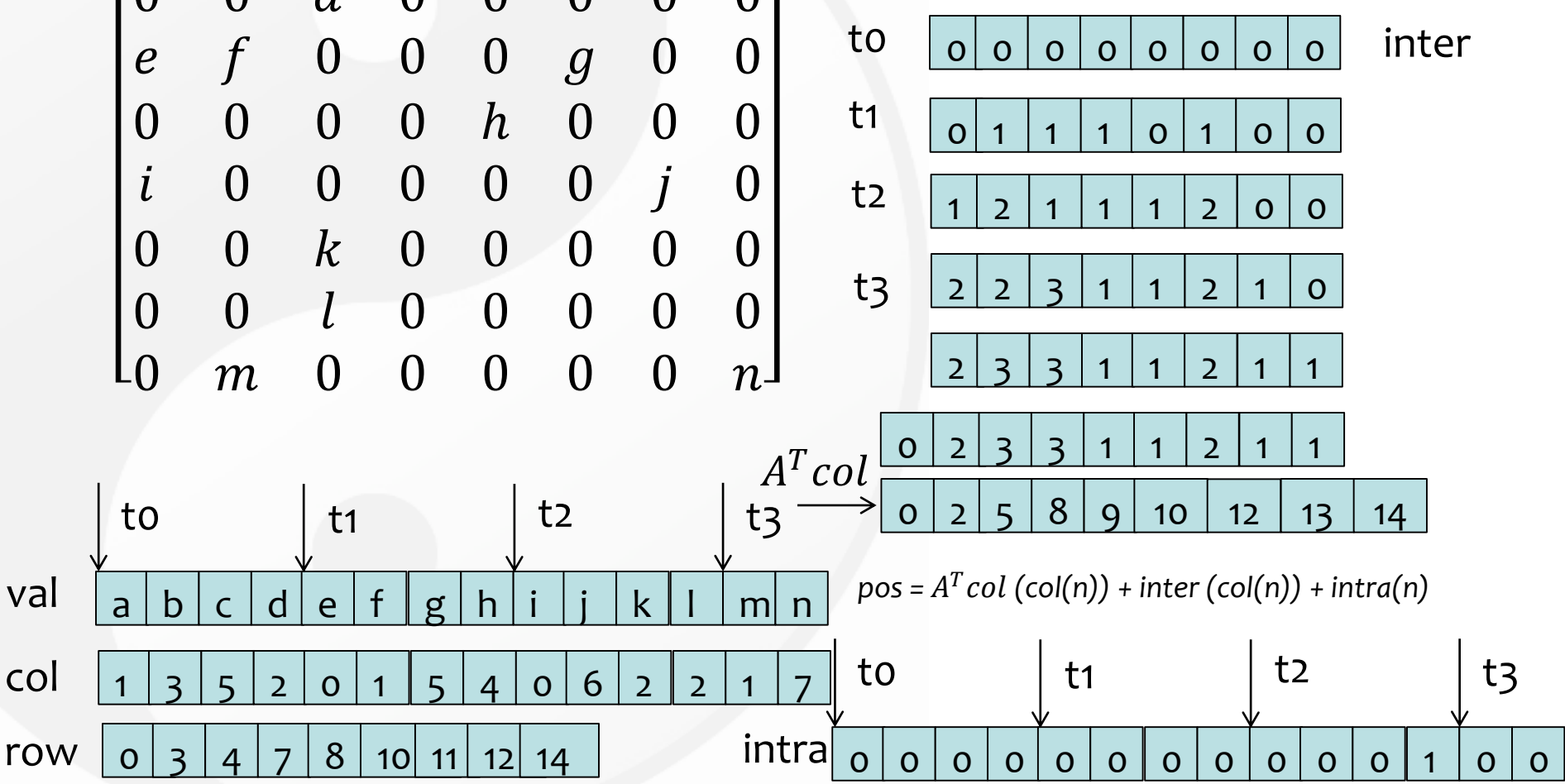
- Calculate prefix sum (inter-thread offset)



Parallel CSR Transpose

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

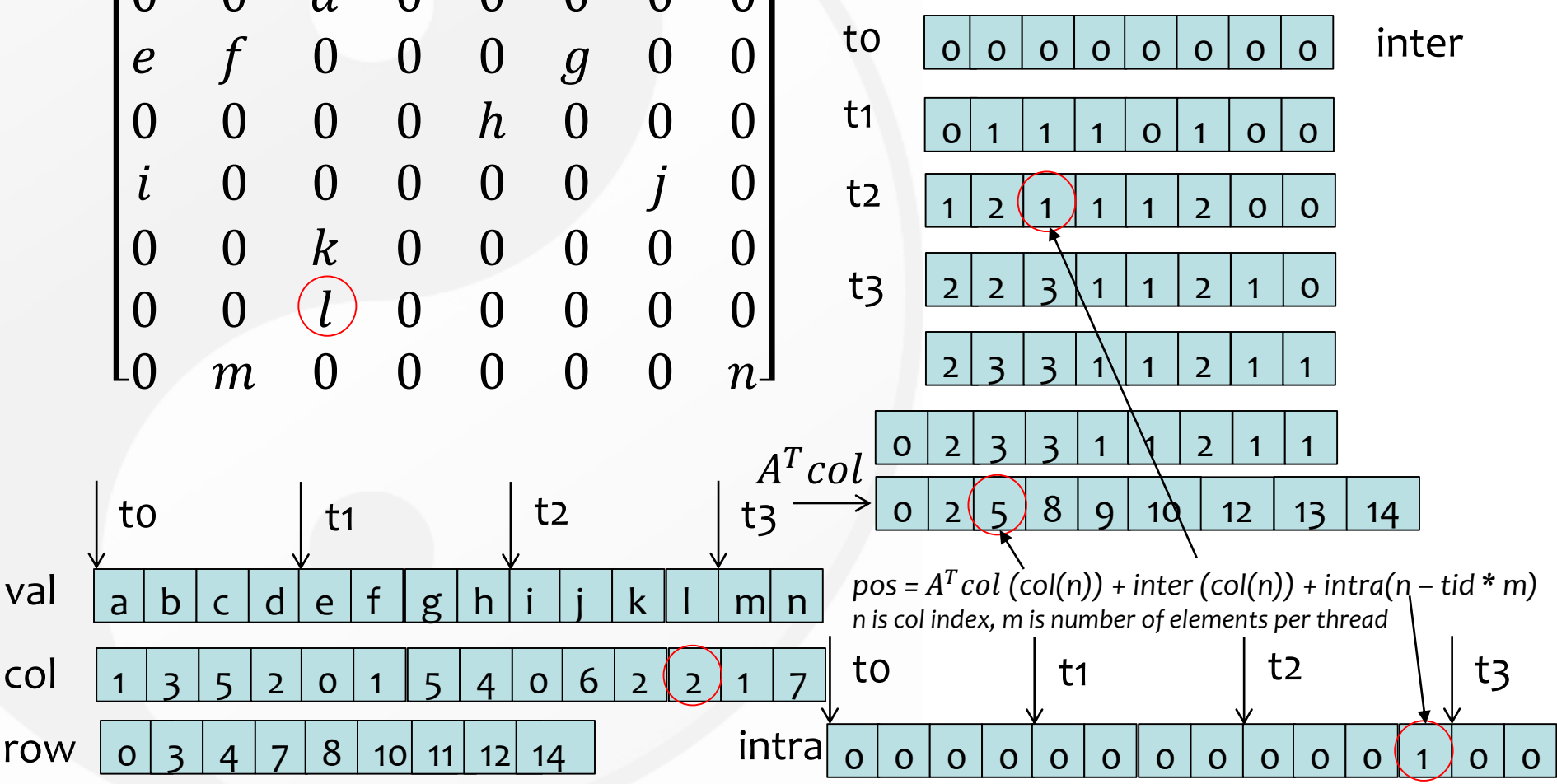
- Calculate position of *A col* in $A^T row$



Parallel CSR Transpose

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

- Calculate position of *A col* in $A^T row$



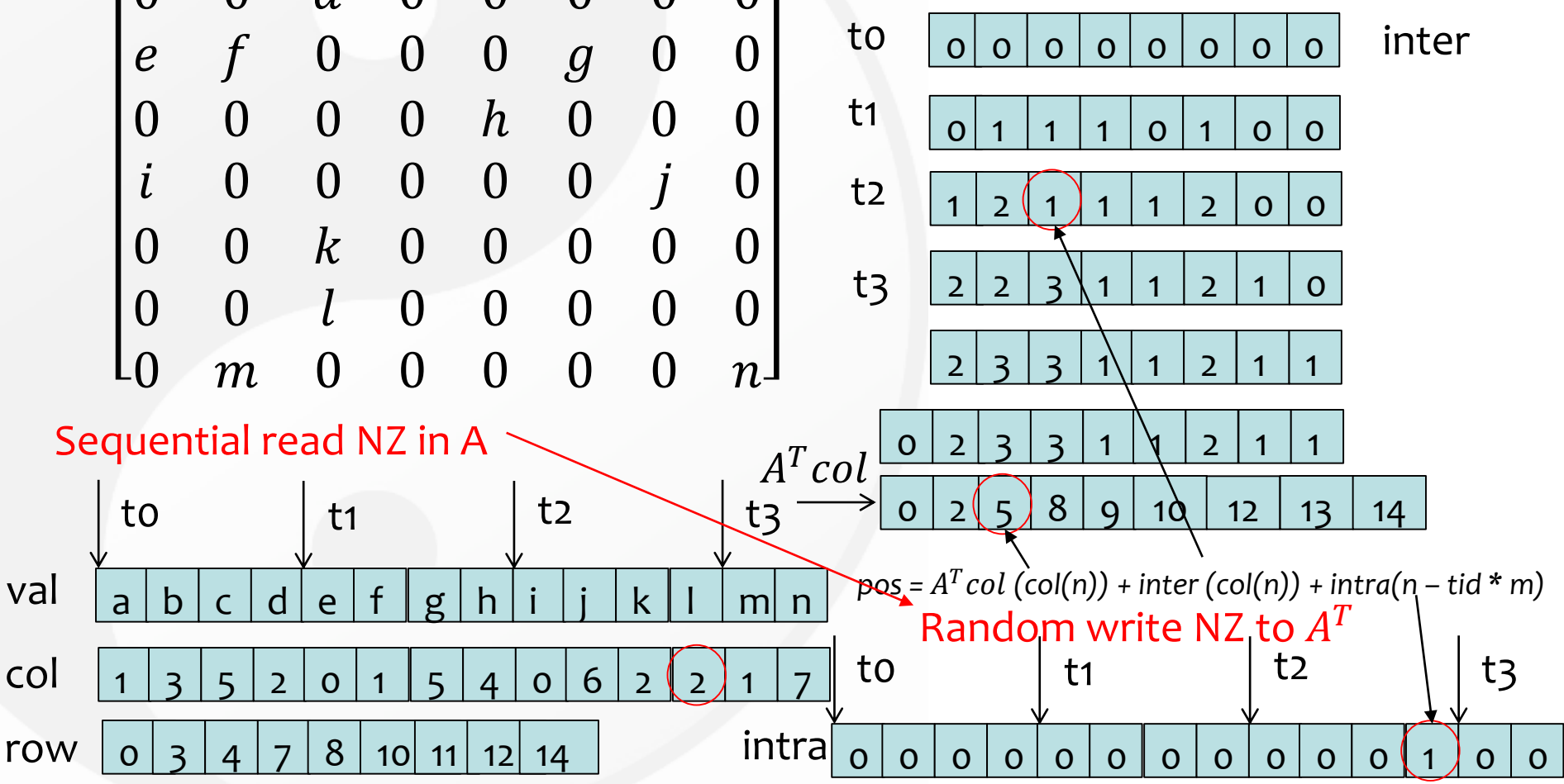
Design Objectives

- Keep balance in multiple parallel threads
  - Divide workload based on NNZ instead of row or column number
- Change multiple data access to offset calculation
  - Compute the offset in A^T for each NZ in A
- Dynamic select the random data access
  - Random data write vs Random data read

Random Write and Sequential Read

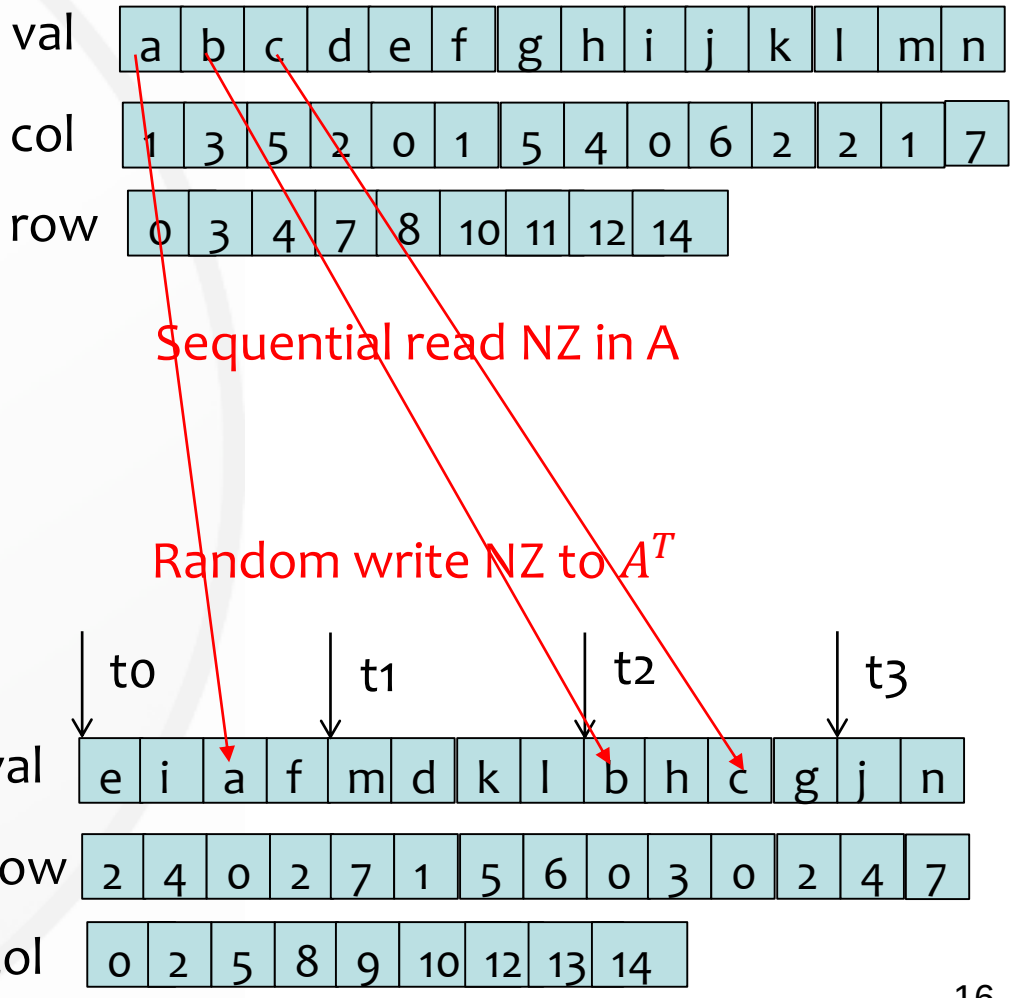
$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

- Calculate position of A *col* in A^T *row*



Random Write and Sequential Read

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$

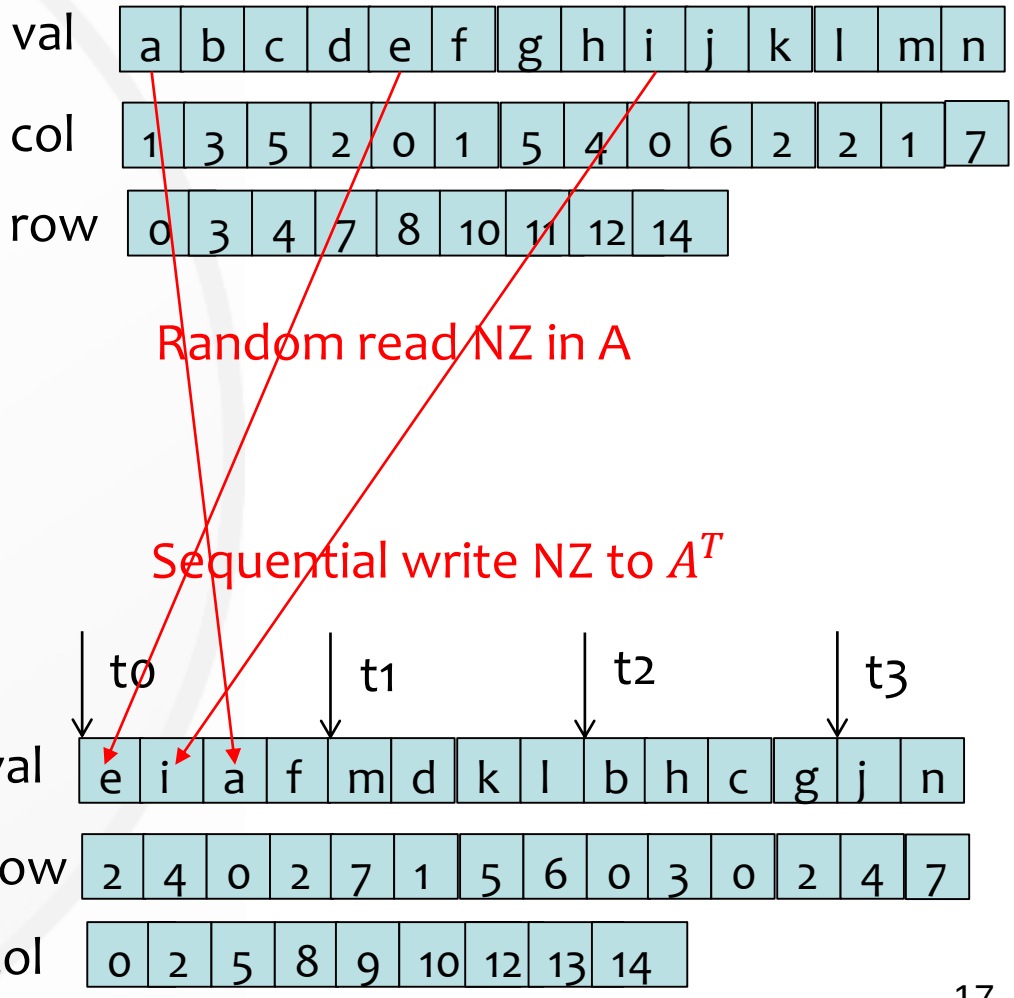


- Calculate position of *A col* in $A^T row$

pos = A^T *col* (*col*(n)) + *inter* (*col*(n)) + *intra*(n – *tid* * *m*)

Sequential Write and Random Read

$$\begin{bmatrix} 0 & a & 0 & b & 0 & c & 0 & 0 \\ 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\ e & f & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & j & 0 \\ 0 & 0 & k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & n \end{bmatrix}$$



- Calculate position of $A^T row$ from A
  - Given pos, compute n from the equation below

$$pos = A^T col\,(col(n)) + inter\,(col(n)) + intra(n - tid * m)$$

Current Results and TODO

- Random write is faster than random read on both Ivy bridge and Xeon Phi
  - Additional overhead on inter- and intra- offset operations
  - Bad cache locality
- TODO
  - Wait for Skylake (with AVX 512 support) and KNL (merged CPU and manycore)