IEEE Cluster 2019

Albuquerque

Carnegie Mellon University

Efficient Distributed Graph Analytics using Triply Compressed Sparse Format

by

Mohammad Hasanzadeh Mofrad, Rami Melhem

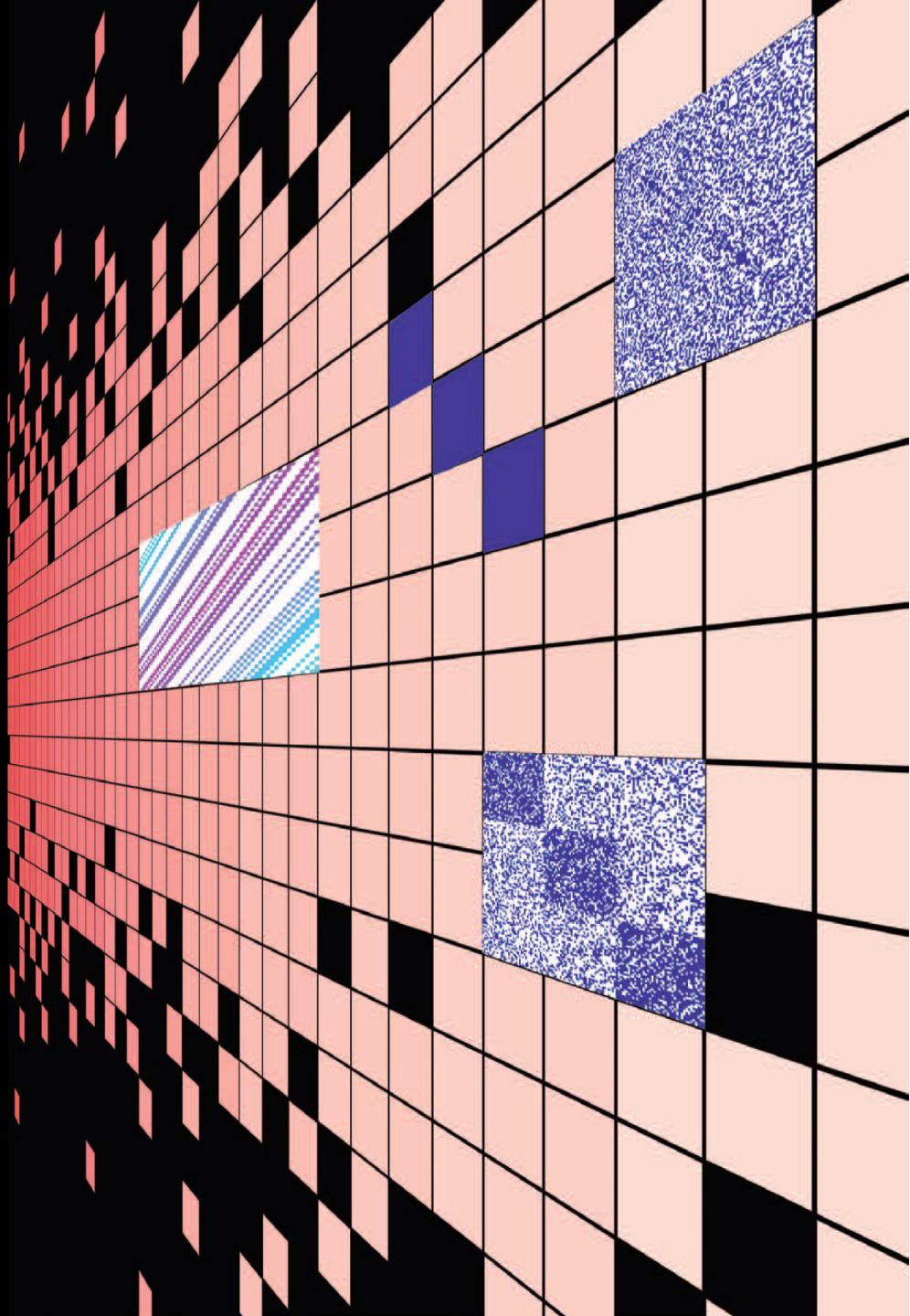
(**University of Pittsburgh**)

Yousuf Ahmad and Mohammad Hammoud

(**Carnegie Mellon University** Qatar)

Date

Wednesday, September 25, 2019



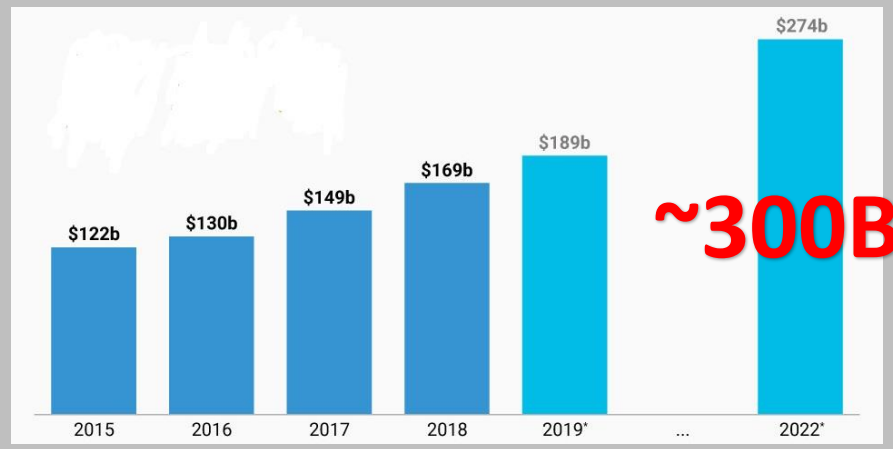
Outline

- **Motivation**
- Background
- Sparse Compression Formats
- GraphTap: Distributed Graph Analytics using Triply Compressed Sparse Column (TCSC)
- Results

Motivation – Big Data

Big Data is a Big Business

Healthcare, government, IT



Big data analytics revenue 2015 – 2022 [1]

Relational data is dominating the raw Big Data

The web, social networks, Location services

Online shopping is becoming a **norm**. **Amazon** has 119.9 M items (as of 2019) [2]

Social Media is the way to engage people [3]. **Facebook** has 2.23 B monthly active users (MAUs)

Location services are becoming **pervasive**. **Google Map** has 154.4 M MAUs (as of 2018) [4], and **Uber** has 91 M MAUs (as of 2019) [5]

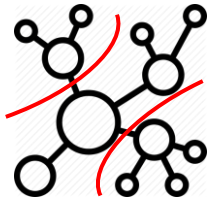
[1] https://www.pcmag.com/news/368958/the-big-data-market-is-set-to-skyrocket-by-2022
[2] https://www.scrapehero.com/number-of-products-on-amazon-april-2019/
[3] https://buffer.com/library/social-media-sites
[4] https://www.statista.com/statistics/865413/most-popular-us-mapping-apps-ranked-by-audience/
[5] https://www.reuters.com/article/us-uber-ipo-lyft-factbox/factbox-how-uber-and-lyft-compare-on-key-financial-metrics-idUSKCN1RO006

Motivation – Big Data Graph Analytics

- **Structural** and **time relationships** are scattered all over data produced by todays businesses
- **Graphs** as mathematical structure can be used to model these structures. Example **application** of graph analytics:
- A Big portion of Big Data is structured data which can be represented by **Graphs**, e.g., social networks and location services.

Graph Applications

Partitioning

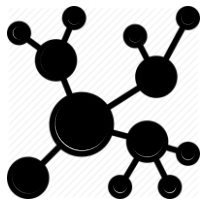


Clustering

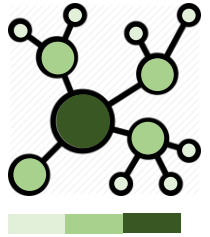
Search

Shortest/widest path

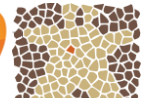
Connected component



PageRank



These applications require distributed solutions

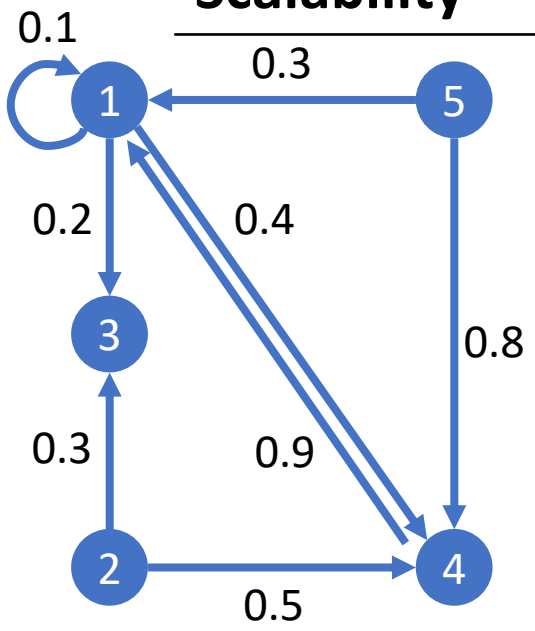


Outline

- Motivation
- **Background**
- Sparse Compression Formats
- GraphTap: Distributed Graph Analytics using Triply Compressed Sparse Column (TCSC)
- Results

Background - Duality Between Graphs and Sparse Matrices

	Graph Theory	Linear Algebra
Representation	Graph	Adjacency Matrix
Data structure	Adjacency list	Sparse matrix compression
Primitive	Fan-in/ Fan-out operations	Sparse Matrix – Vector (SpMV)
Scalability	Graph partitioning	Sparse matrix partitioning



Src	Dst	Wgt
1	1	0.1
1	3	0.2
1	4	0.4
2	3	0.3
2	4	0.5
4	1	0.9
5	1	0.3
5	4	0.8

	0	1	2	3	4	5
0						
1		.1		.2	.4	
2				.3	.5	
3						
4		.9				
5		.3			.8	

	0	1	2	3	4	5
0						
1		.1			.9	.3
2						
3		.2	.3			
4		.4	.5			.8
5						

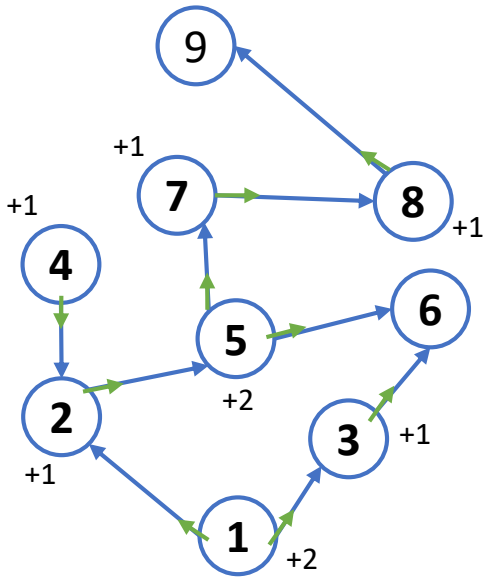
Graph *G*

Adjacency List

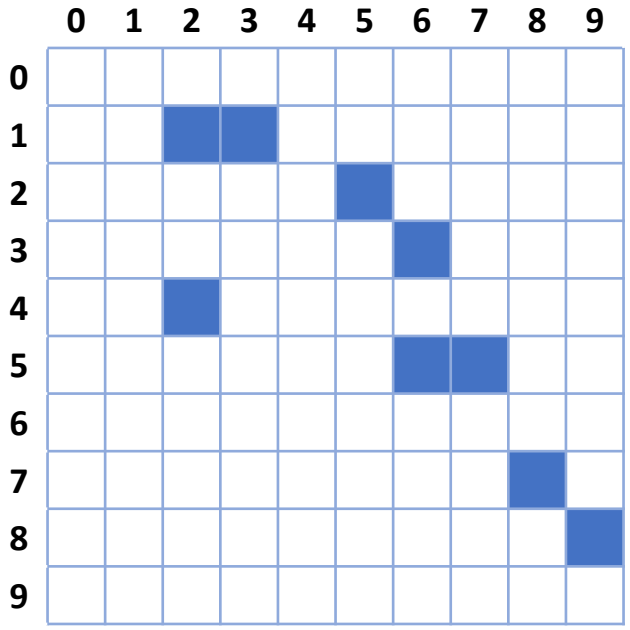
Adjacency Matrix

Transposed Adjacency Matrix *A*

Sparse Matrix – Dense Vector (SpMV) Primitive

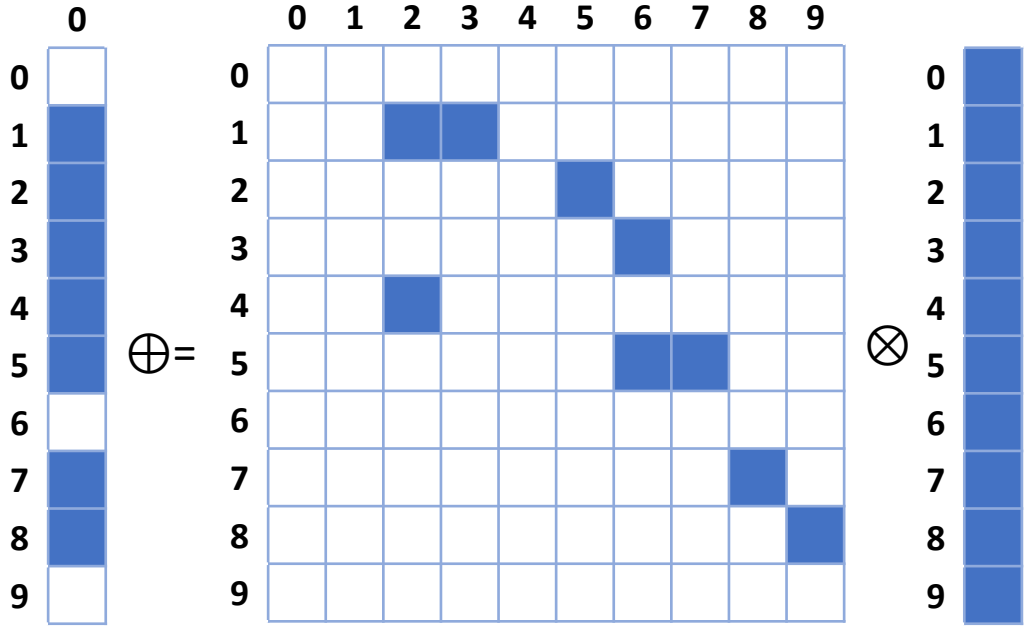


Graph *G*



Adjacency matrix of *G*, *A*

SpMV



Output vector *y* **Sparse matrix *A*** **Input vector *x***

$y = A \oplus . \otimes x$

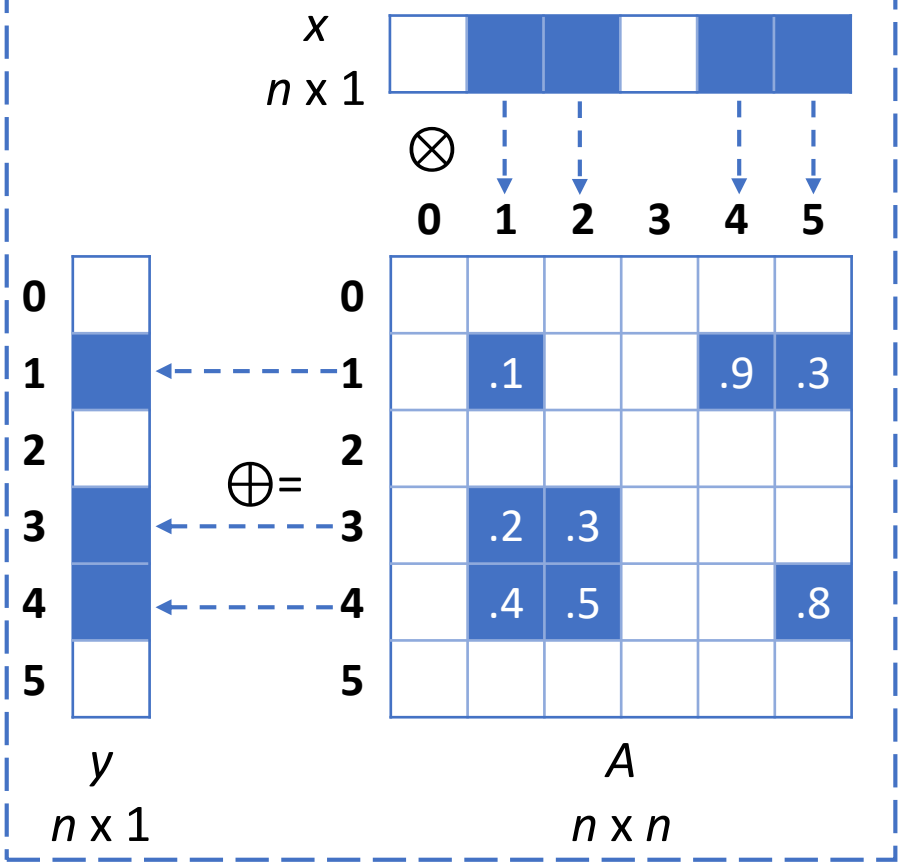
Legend	
⊗	Multiply *op*
⊕	Add *op*
SpMV	$y = A \oplus . \otimes x$
	Empty entry
	Empty row or col
	Empty row & col
	Nonzero entry

Outline

- Motivation
- Background
- **Sparse Compression Formats**
- GraphTap: Distributed Graph Analytics using Triply Compressed Sparse Column (TCSC)
- Results

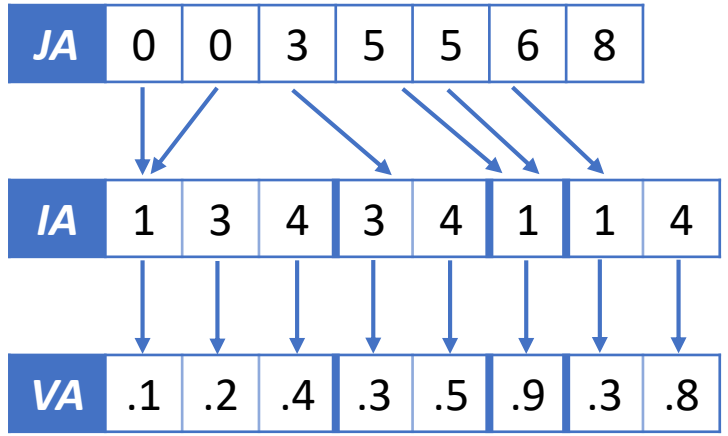
Compressed Sparse Column (CSC)

CSC space requirement is ***n* + 2*nnz* + 1** where *nnz* is the number of nonzero entries
+ Sequential column-major access (No indirection for SpMV)
- Length of *JA* and, *x* and *y* vectors equals *n*



CSC SpMV Diagram ($y = A \oplus .\otimes x$)

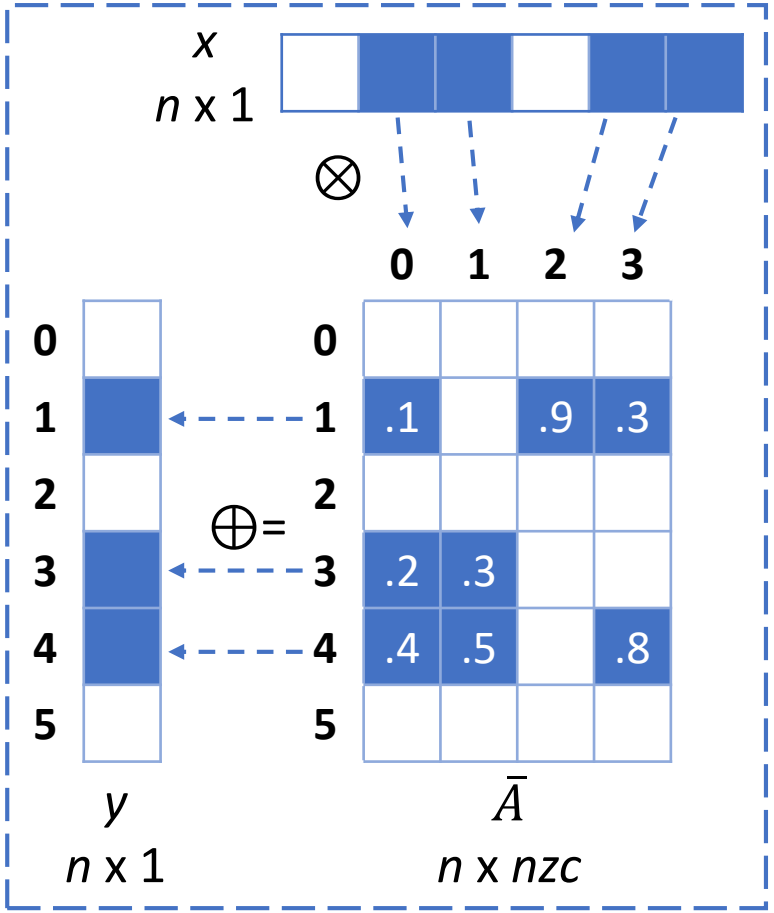
The column (or row) compressed sparse formats store and traverse a 2D matrix using a set of 1D arrays. **Compressed Sparse Column (CSC)** is the vanilla compressions (Matlab & SciPy).



CSC Data Structures

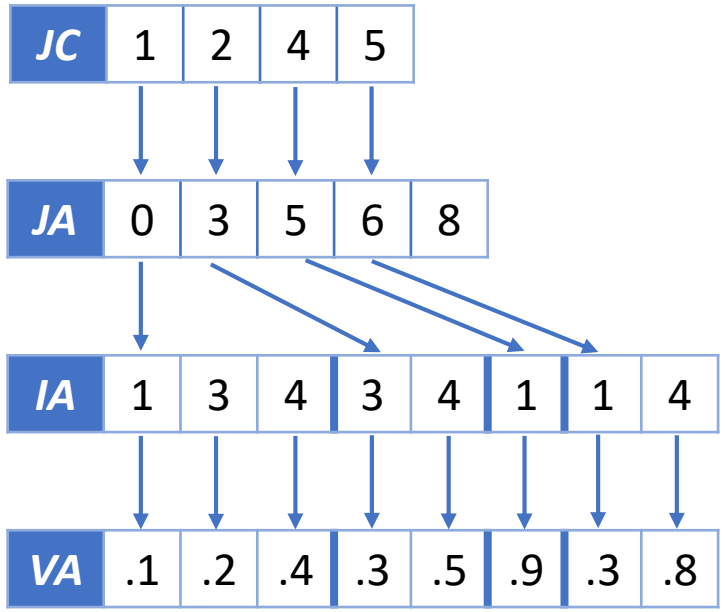
Doubly Compressed Sparse Column (DCSC)

CSC space requirement is ***n* + 2*nnz* + 1** where *nnz* is the number of nonzero entries
+ Sequential column-major access (No indirection for SpMV)
- Length of *JA* and, *x* and *y* vectors equals *n*



DCSC SpMV Diagram ($y = \bar{A} \oplus . \otimes x$)

Doubly Compressed Sparse Column (DCSC) is a flavor that removes nonzero columns (PETSc & CombBLAS).

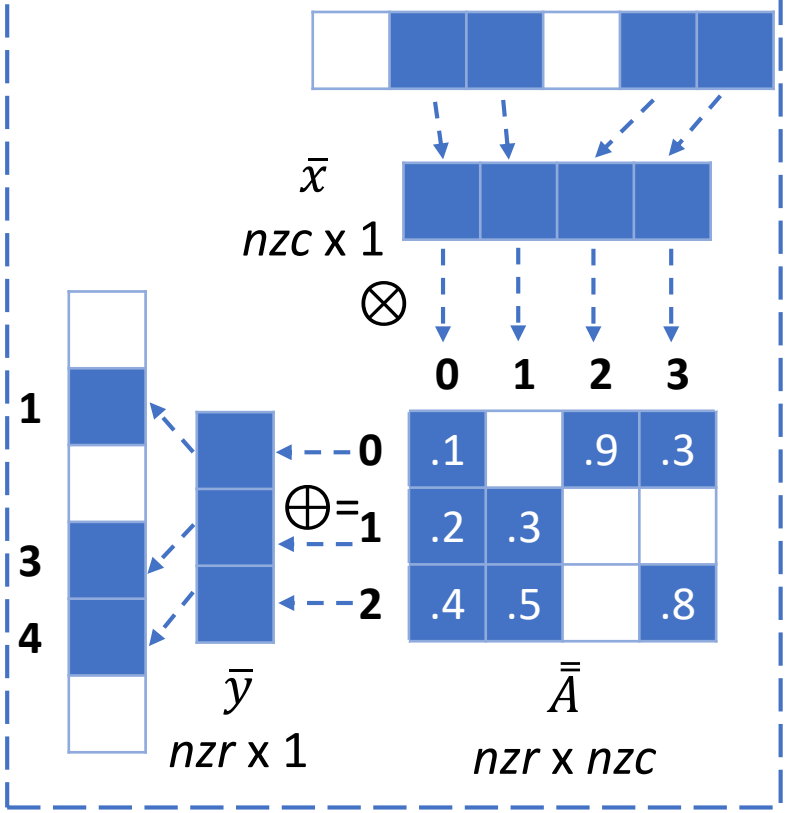


DCSC Data Structures

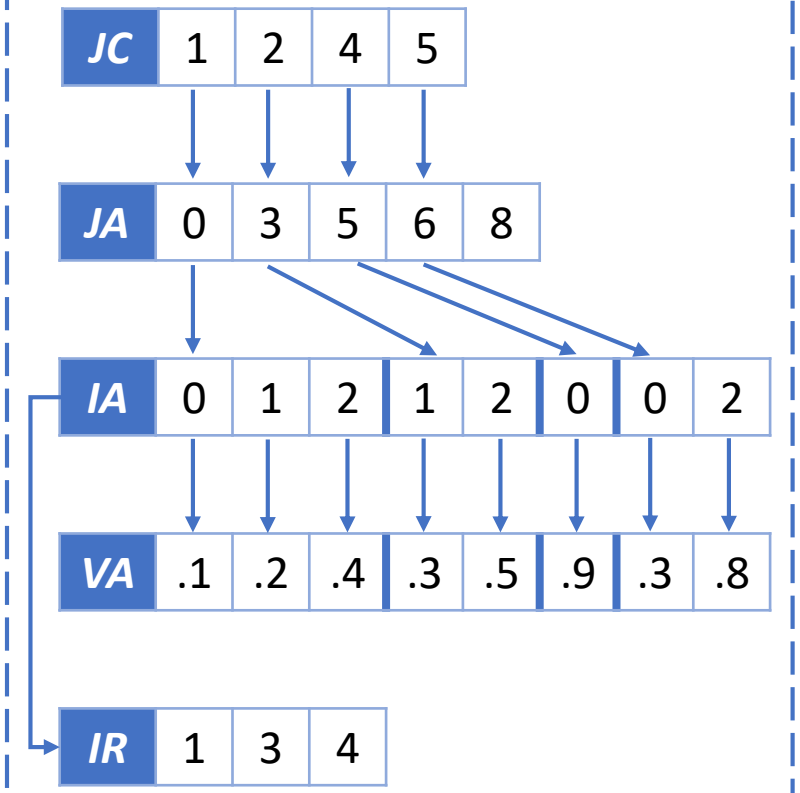
DCSC Space requirement is **2*nzc* + 2*nnz* + 1** where *nzc* is the number of nonzero columns
+ One indirection for SpMV (*n* → *nzc*)
+ Length of *JA* equals *nzc*
- Length of *x* and *y* vectors equals *n*

Triply Compressed Sparse Column (TCSC)

Conventional formats compress the matrix independent of the vectors.
Triply Compressed Sparse Column (TCSC) is a new flavor that removes both nonzero columns and rows and co-compresses the matrix and vectors
TCSC runs Sparse Matrix – Sparse input and output Vectors (**SpMSpV²**)



TCSC SpMSpV² Diagram ($\bar{y} = \bar{\bar{A}} \oplus . \otimes \bar{x}$)



TCSC Data Structures

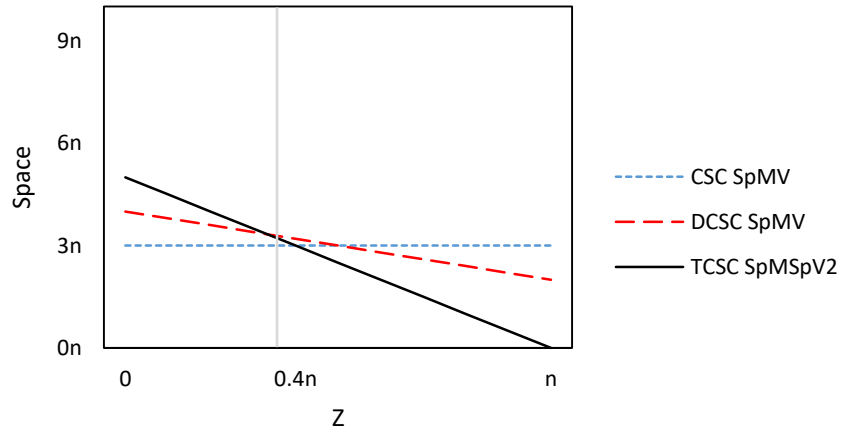
CSC space requirement is ***n* + 2*nnz* + 1** where *nnz* is the number of nonzero entries
+ Sequential column-major access (No indirection for SpMV)
- Length of *JA* and, *x* and *y* vectors equals *n*

DCSC Space requirement is **2*nzc* + 2*nnz* + 1** where *nzc* is the number of nonzero columns
+ One indirection for SpMV (*n* → *nzc*)
+ Length of *JA* equals *nzc*
- Length of *x* and *y* vectors equals *n*

TCSC Space requirement is **2*nzc*+ *nzr*+ 2*nnz*+1** where *nzr* is the number of nonzero rows
+ No indirection for SpMSpV²
+ Length of *JA* equals *nzc*
+ Length of *x* and *y* vectors equals *nzc* and *nzr*

Comparison of Space Requirements

- Simpler forms of Space requirements*:
  - CSC SpMV: $3n$
  - DCSC SpMV: $2n + 2(n - z) = 4n - 2z$
  - TCSC SpMSpV^2: $3(n - z) + 2(n - z) = 5n - 5z$



- Observation: If more than 40% of rows/columns ($z = 0.4\ n$) are empty TCSC can save space.

*Term $2nnz + 1$ can be eliminated from all space formulas , and $nzr \approx nz$ and $nz = n - z$

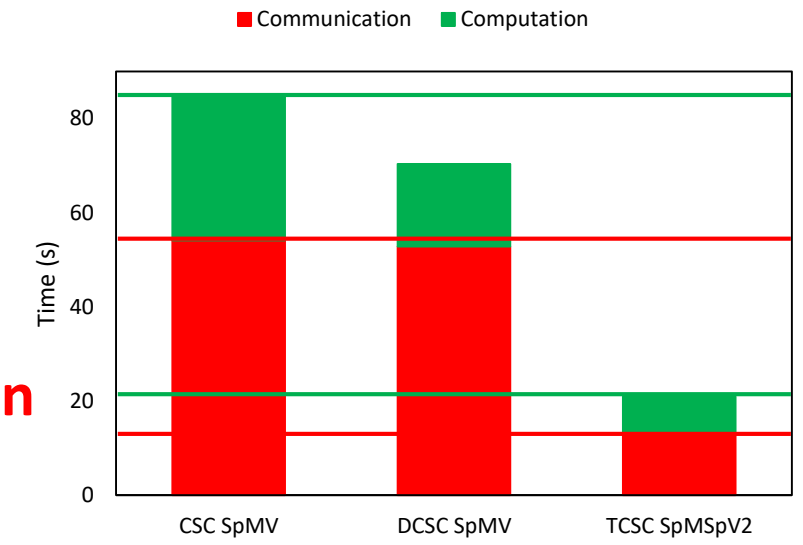
Distributed SpMV vs SpMSpV^2

- ✓ **Matrix size** translates into **computation**
- ✓ **Compact loops and more indirections** increases the **computation time**
- ✓ (especially for synthetic graphs)

SpMV is a compute-intensive & memory-intensive primitive.

In a distributed setting:

- ✓ **Vector sizes** translates into **communication**
- ✓ **Larger vectors** increases the **communication time**
- ✓ (especially for real-world graphs)



PageRank on Twitter with 1.46B edges

Outline

- Motivation
- Background
- Sparse Compression Formats
- **GraphTap: Distributed Graph Analytics using Triply Compressed Sparse Column (TCSC)**
- Results

GraphTap: Distributed Graph Analytics using Triply Compressed Sparse Column (TCSC)

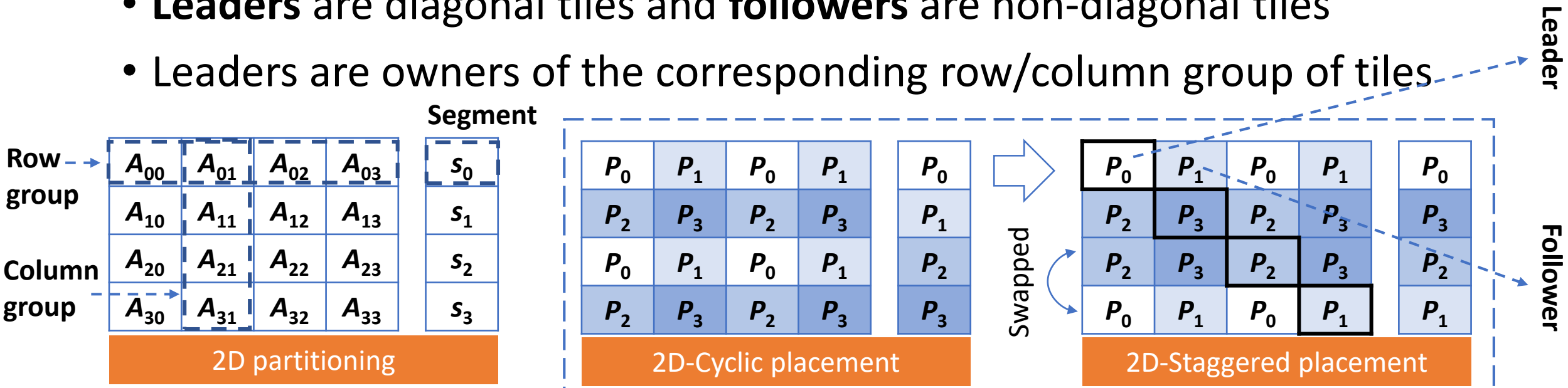
- **GraphTap** uses sparse matrix compression to store each tile
  - It incorporates TCSC as its core compression format.

- **GraphTap** is a new distributed graph analytics system
  - It uses matrix partitioning to break a 2D matrix into square tiles
  - It uses process placement to distribute tiles among processes (machines)

- **GraphTap** uses a variant of GAS (Gather, scatter, and Apply) graph computing model that can be overload with user-defined code.
  - Scatter-gather, combine, and apply computing model.

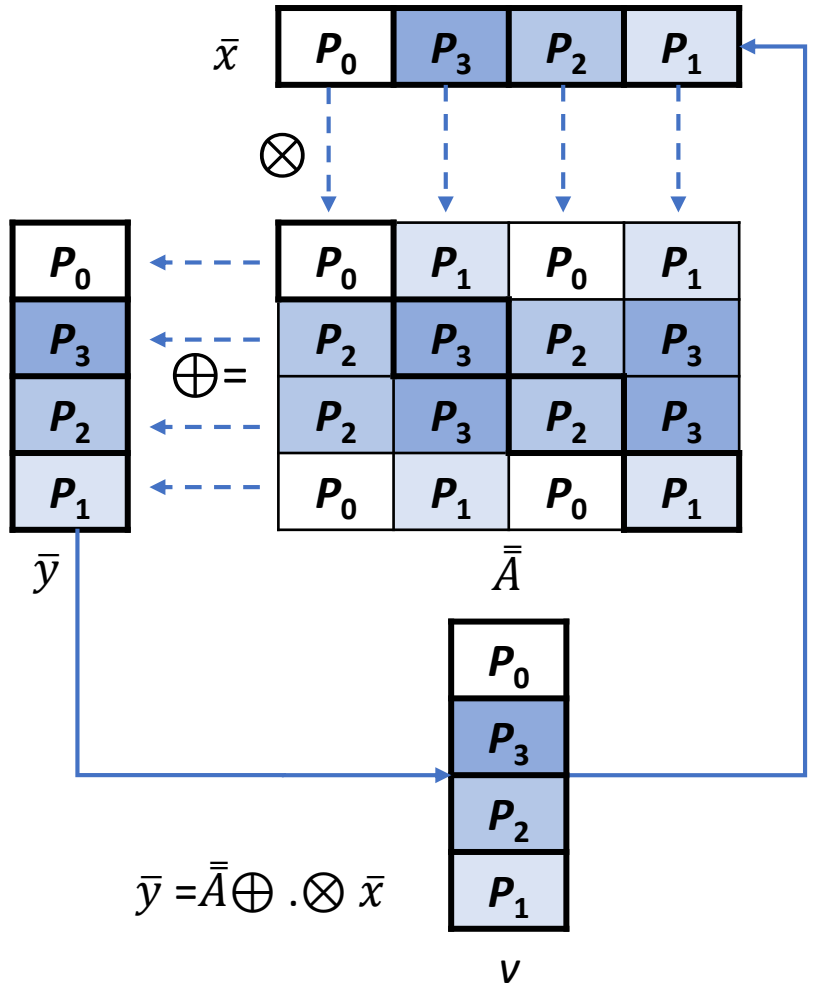
Matrix Partitioning and Process Placement for Scalability

- 2D Partitioning: Given *p* processes create a *p* x *p* grid of tiles
- Process Placement: 2D-Staggered is used for process placement (2D-Cyclic + unique process at diagonal tiles)
- **Leaders** are diagonal tiles and **followers** are non-diagonal tiles
- Leaders are owners of the corresponding row/column group of tiles



Computing Model

- **Scatter-gather**
  - Scatter: Leaders construct and send new $\bar{x}$ segments
  - Gather: Followers receive the new $\bar{x}$ segments
- **Combine**
  - All processes execute the SpMSpV^2 primitive
  - Per row group, leaders receive partial results $\hat{\bar{y}}$ from followers and accumulate with their $\bar{y}$
- **Apply**
  - Leaders construct and store new values *v* from $\bar{y}$



Outline

- Motivation
- Background
- Sparse Compression Formats
- GraphTap: Distributed Graph Analytics using Triply Compressed Sparse Column (TCSC)
- **Results**

Experimental Settings to evaluate TCSC

Single Thread

- A machine with 12-core and 512 GB RAM
- Compared CSC, DCSC, and TCSC
- Graph applications
  - PageRank (PR)

Distributed

- A cluster of 32 machines each with 28-core, 192 GB RAM and Intel Omni-path network
- Compared **GraphTap** (TCSC), GraphPad (DCSC), and LA3 (DCSC) systems
- Graph applications:
  - PageRank (PR)
  - Single Source Shortest Path (SSSP)
  - Breadth First Search (BFS)
  - Connected Component (CC)

Datasets

Graph	|V|	|E|	Z_c	Z_r	T	N
UK'05 (UK5)	39.4 M	0.93 B	0	0.12	Web	4
IT'04 (IT4)	41.2 M	1.15 B	0	0.13	Web	4
Twitter (TWT)	41.6 M	1.46 B	0.09	0.14	Soc	8
GSH'15 (G15)	68.6 M	1.8 B	0	0.19	Web	8
UK'06 (UK6)	80.6 M	2.48 B	0.01	0.14	Web	16
UK Union (UKU)	133 M	**5.5 B**	0.05	0.09	Web	24
Rmat26 (R6)	67.1 M	1.07 B	0.55	0.72	Syn	4
Rmat27 (R27)	134 M	2.14 B	0.57	0.73	Syn	8
Rmat28 (R28)	268 M	4.29 B	0.59	0.74	Syn	16
Rmat29 (R29)	536 M	8.58 B	0.61	0.75	Syn	24
Rmat30 (R30)	1.07 B	**17.1 B**	0.62	0.76	Syn	**32**

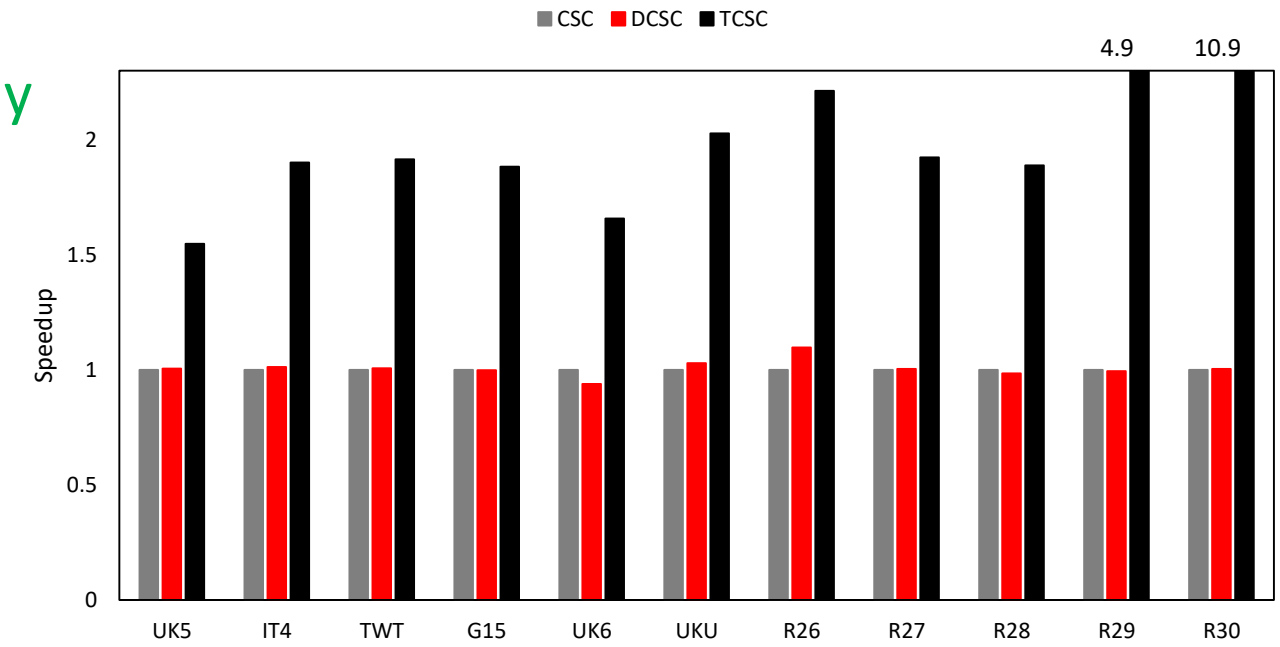
Real-world datasets
1 – 5.5 B

Synthetic datasets
1 – 17 B

Number of machines 4 – 32 machines

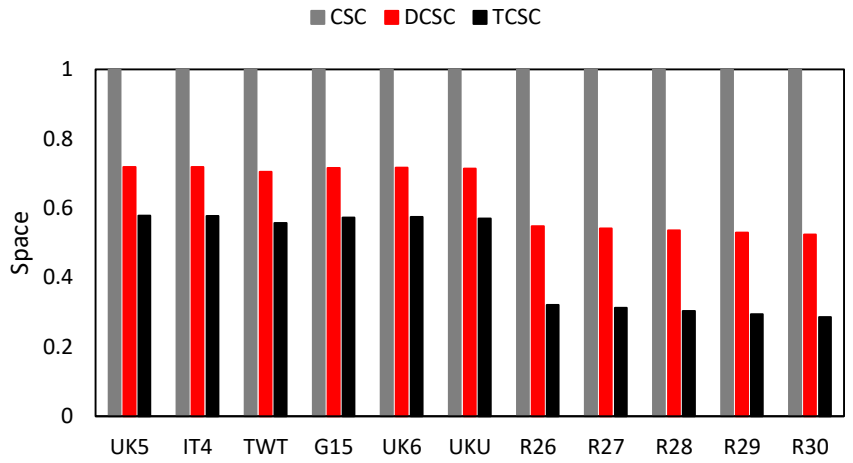
Results – Single Thread PR

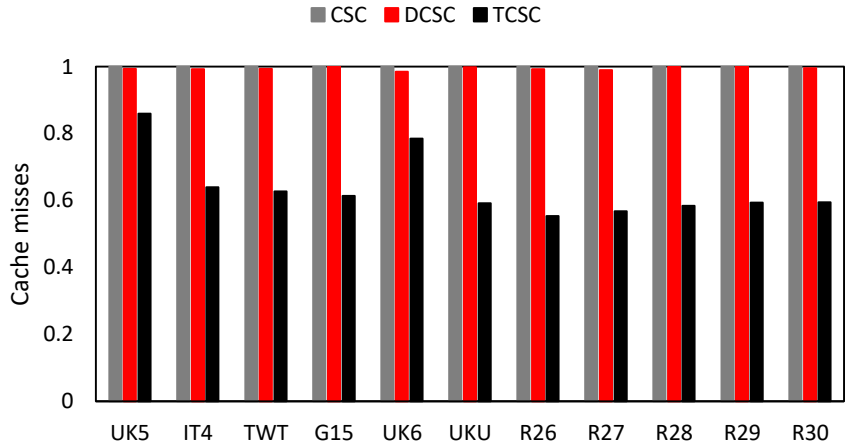
- Speedup
  - TCSC has 2.2 – 11 x speedup compared to CSC and DCSC
  - No indirections avoid polluting L1 cache
  - Smaller vectors fit in L2 cache
  - SpMSpV^2 primitive is cache friendly
  - TCSC is space efficient



Results – Single Thread PR

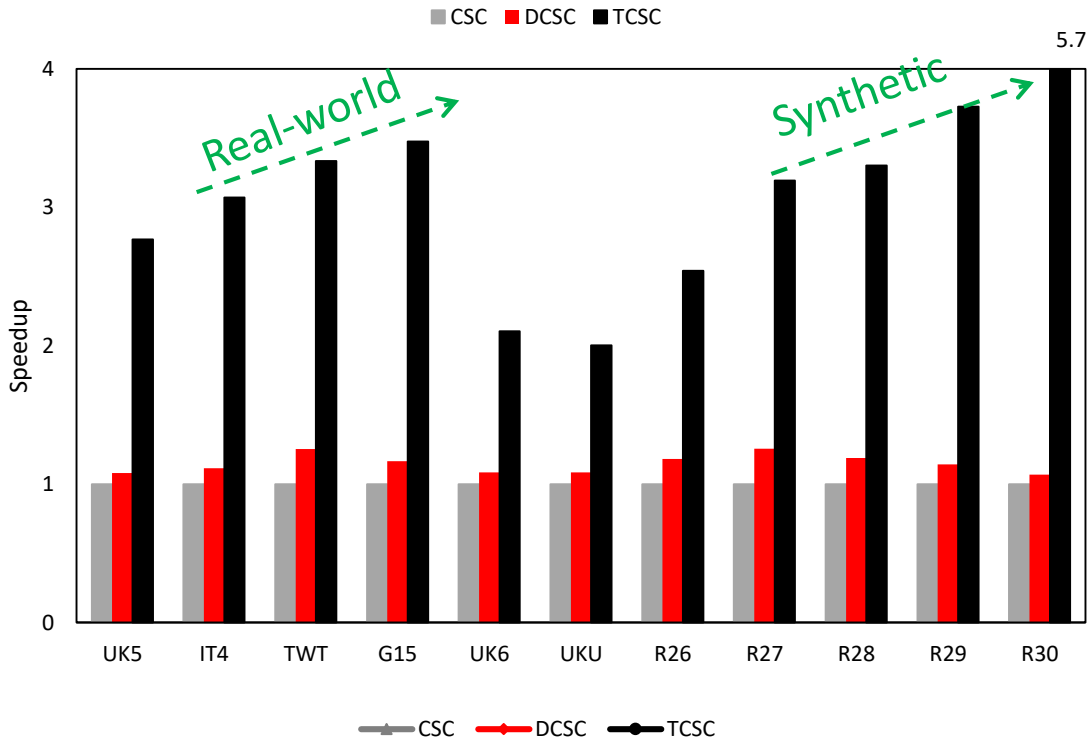
- Space
  - TCSC requires 45 - 75% less space compared to CSC and 15 - 25% compared to DCSC
  - Smaller vectors
- Cache misses
  - TCSC has 20 - 40% less cache misses compared to CSC and DCSC.
  - Less indirections



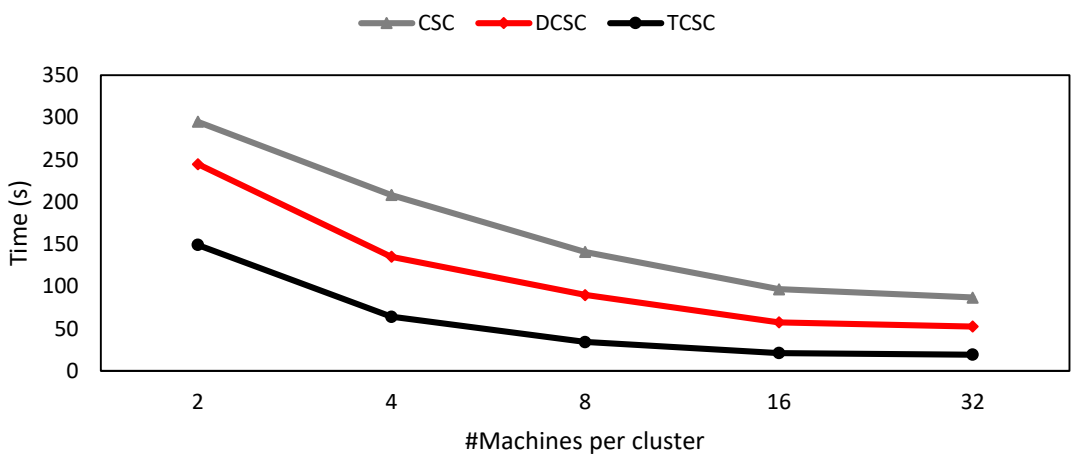


Results – Distributed PR (GraphTap)

- Speedup
  - TCSC is up to 5.7x faster than CSC & DCSC
  - Co-compressing matrix & vectors

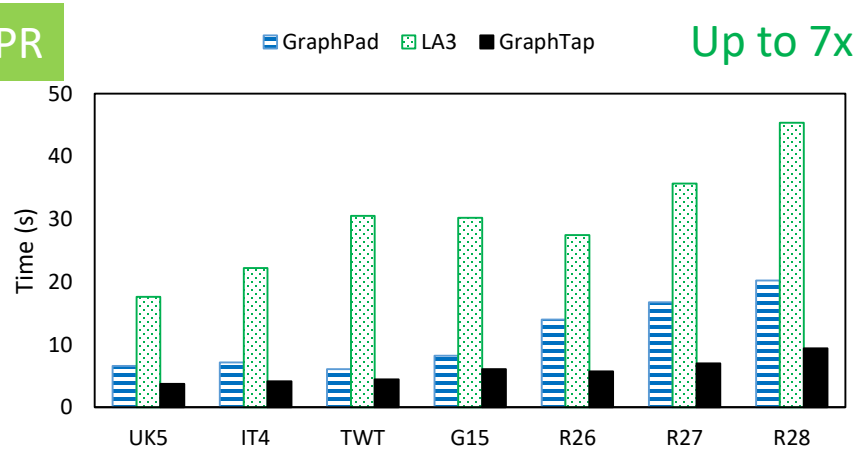


- TCSC is more scalable in **process scalability** test (16 processes per machine on Rmat29 with 2 → 32 #machines on Rmat29)

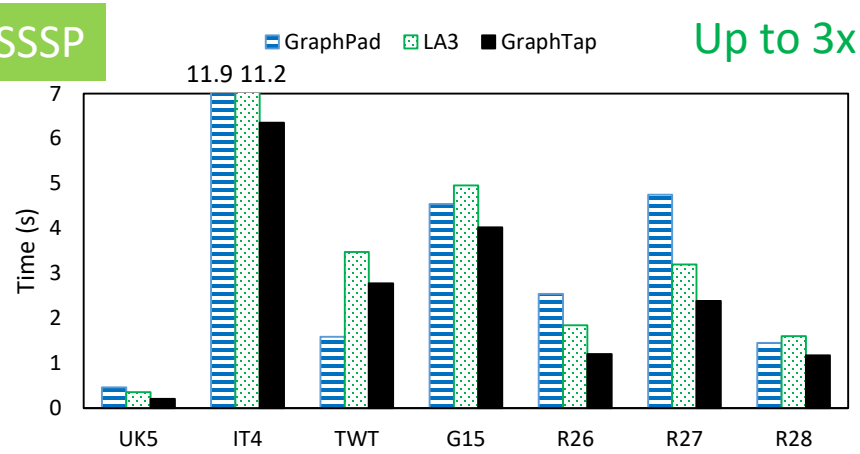


Results – GraphTap Versus GraphPad & LA3

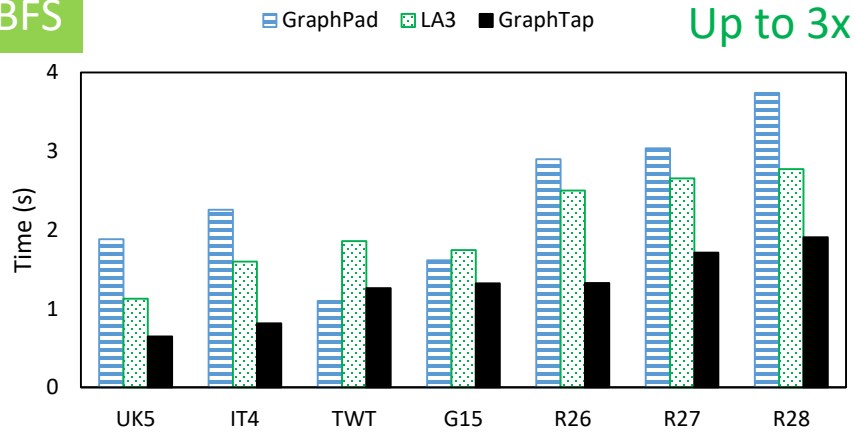
PR



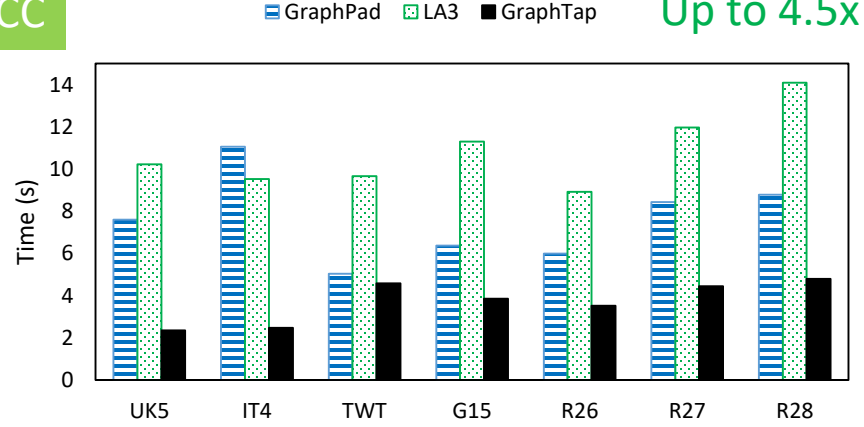
SSSP



BFS



CC



- GraphTap is more cache friendly than GraphPad and LA3 because of TCSC SpMSpV^2 primitive
- GraphTap is more scalable than GraphPad and LA3 because, TCSC has less communication volume

Summary & Conclusion

- We proposed a co-compression technique called **Triply Compressed Sparse Column (TCSC)** which compresses matrix and vectors cooperatively and runs atop **Sparse Matrix – Sparse input and output Vectors (SpMSpV^2) primitive**.
- Compared to CSC and DCSC, TCSC is:
  - more space efficient because it inherently compresses vectors[1]
  - cache friendly because it has less indirections
  - faster because it offers an efficient SpMSpV^2 primitive
- **GraphTap** (our new distributed graph analytics) outperforms GraphPad and LA3
  - TCSC performs significantly better compared to CSC and DCSC due to its computation and communication efficiencies.

1. Often, compressed vectors are shown using pairs of (index, value) or bitvector accompanying with a full-length value vector

Thank You!

Want to know more about me! Checkout my homepage
http://cs.pitt.edu/~moh18