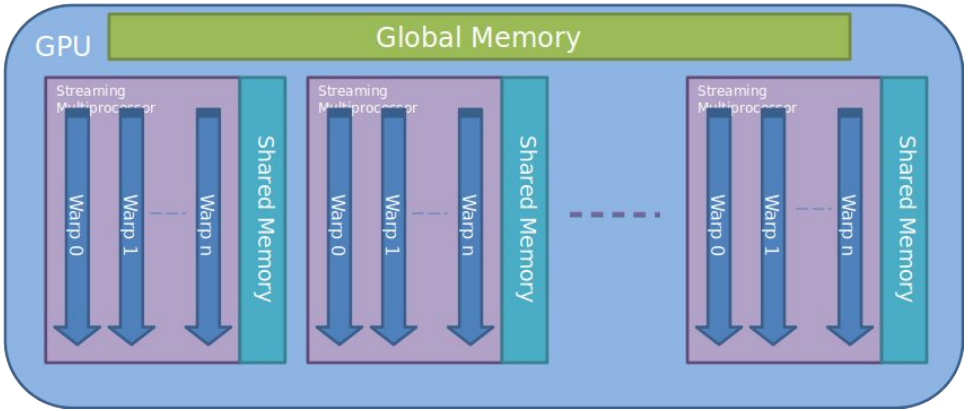
Optimizing Graph Algorithms for GPUs

By: Mayuresh Anand, Alon Albalak, Koa Sato

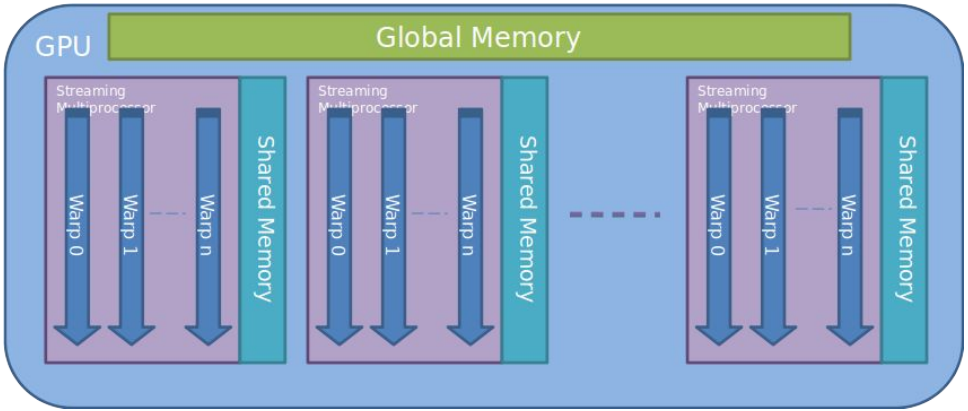
Graphical Processing Units

- **Composed of GPU cores**



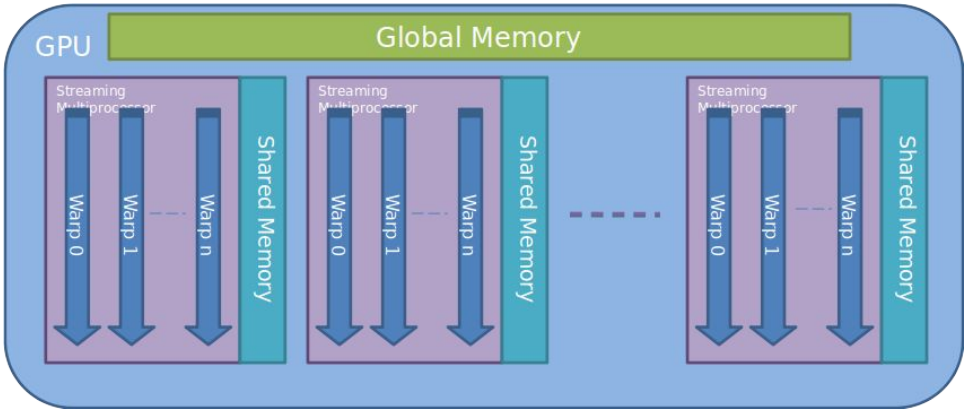
Graphical Processing Units

- Composed of GPU cores
- **Each core is composed of threads which are organized by a streaming multiprocessor (SM)**



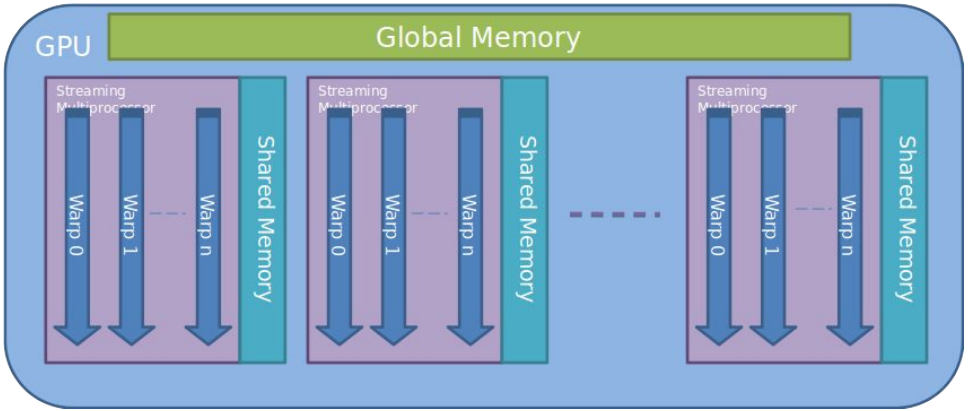
Graphical Processing Units

- Composed of GPU cores
- Each core is composed of threads which are organized by a streaming multiprocessor (SM)
- **Threads are organized into warps, such that each warp executes an instruction in parallel**



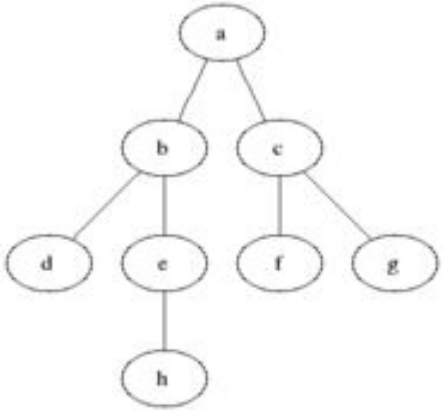
Graphical Processing Units

- Composed of GPU cores
- Each core is composed of threads which are organized by a streaming multiprocessor (SM)
- Threads are organized into warps, such that each warp executes an instruction in parallel
- **Most GPUs utilize single-instruction, multiple-threads paradigm**



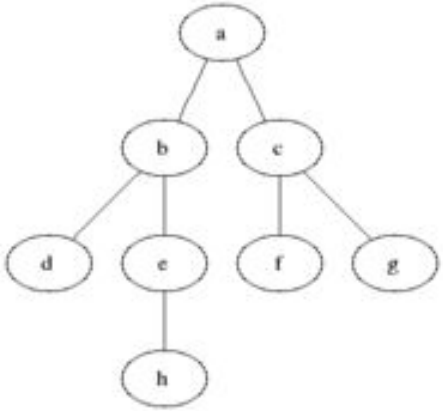
Graph Algorithms

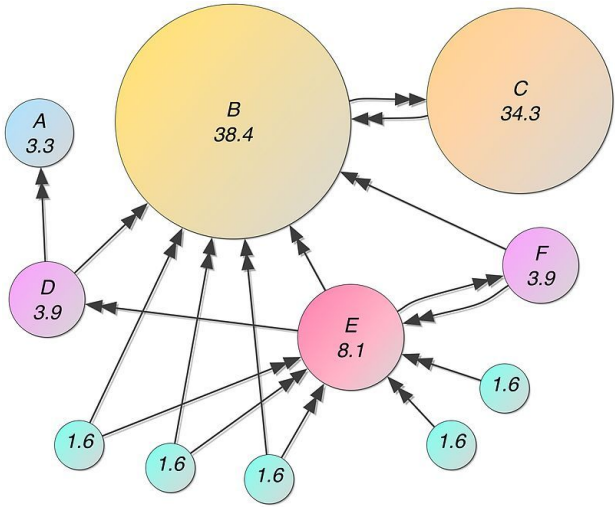
- **Breadth-first search**



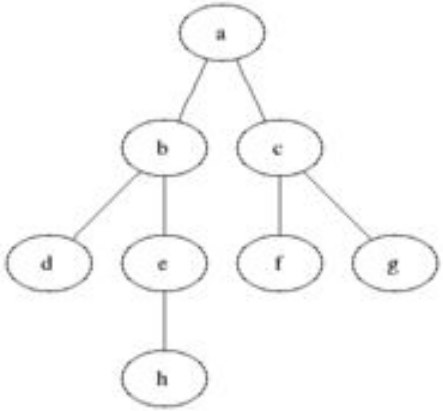
Graph Algorithms

- Breadth-first search
- **PageRank**

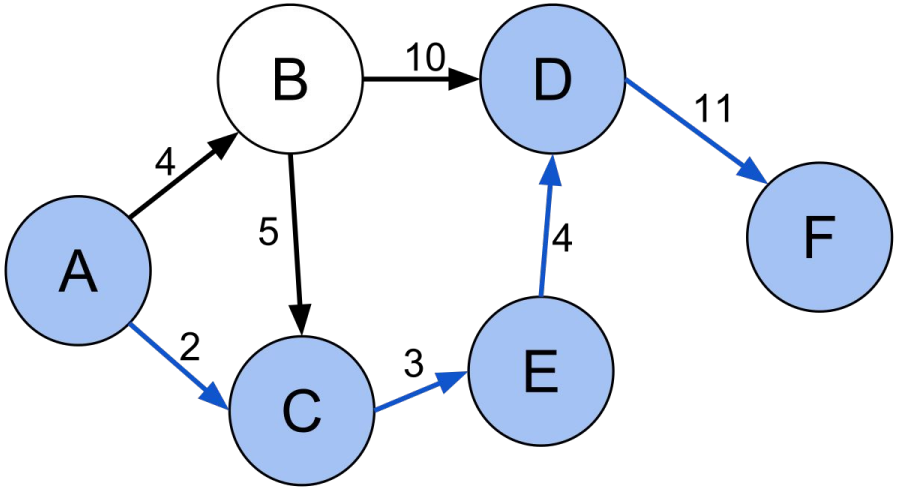


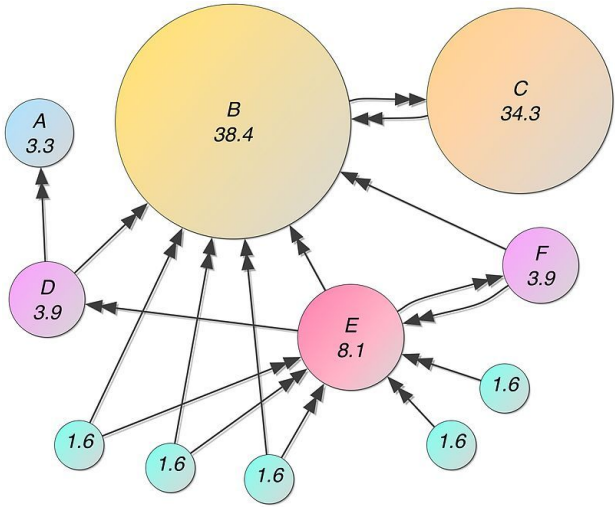


Graph Algorithms

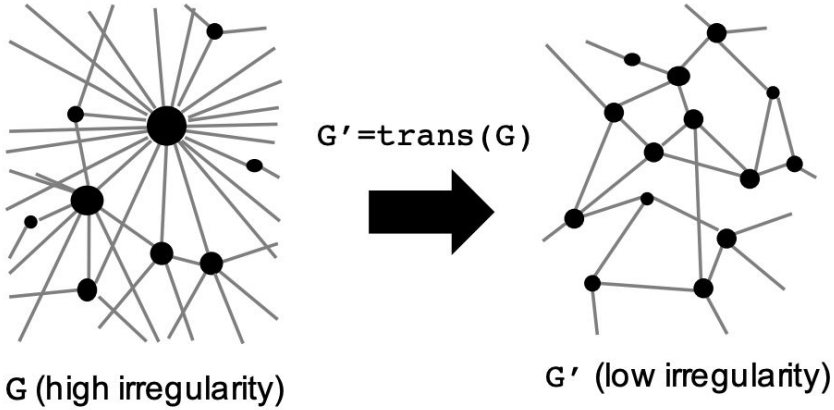


- Breadth-first search
- PageRank
- **Single-source shortest path**



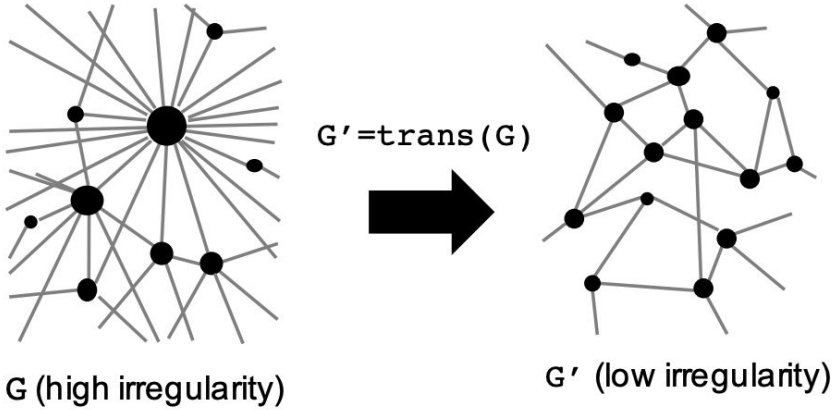


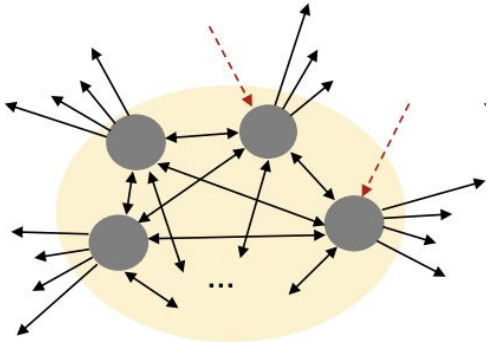
Graph Regularization



Graph Regularization

- Methods include transformations:
  - **Clique**

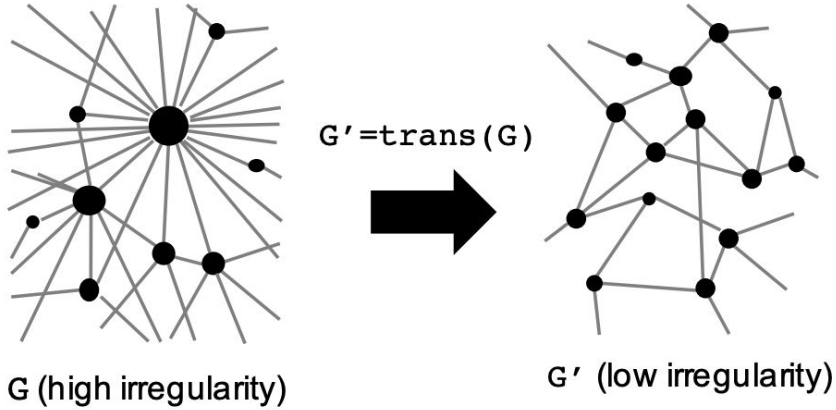


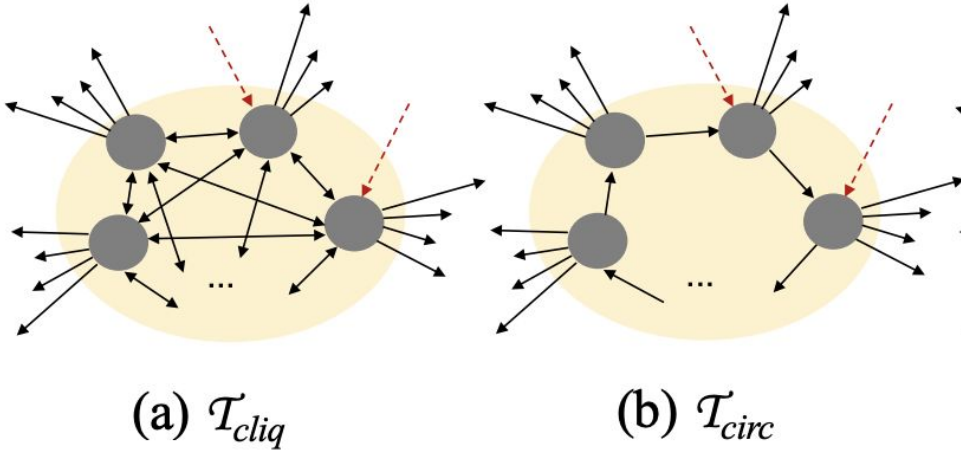


(a) $\mathcal{T}_{cliq}$

Graph Regularization

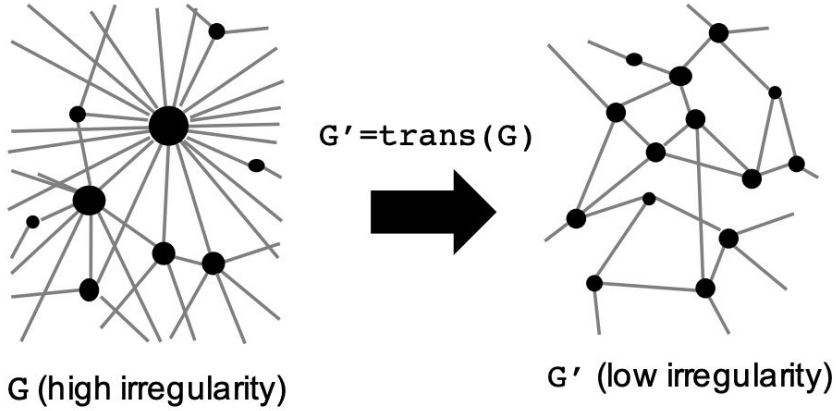
- Methods include transformations:
  - Clique
  - **Circular**

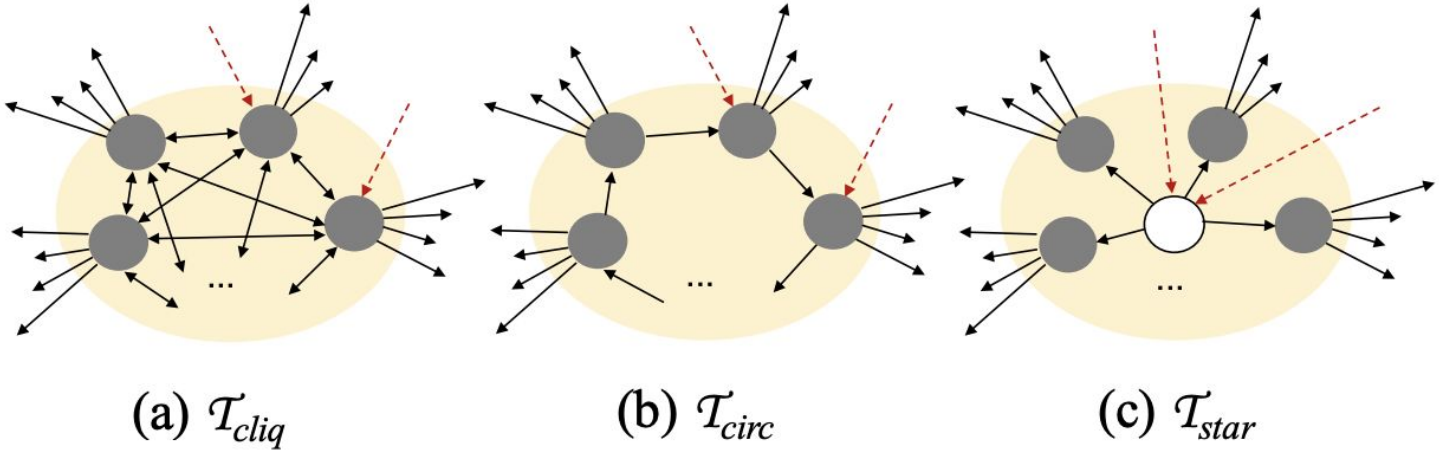




Graph Regularization

- Methods include transformations:
  - Clique
  - Circular
  - **Star**





Tigr - Transforming Irregular Graphs into more Regular ones

- **Claim that GPU-based graph algorithms suffer due to power law structure**

http://www.cs.ucr.edu/~anode001/asplos18.pdf

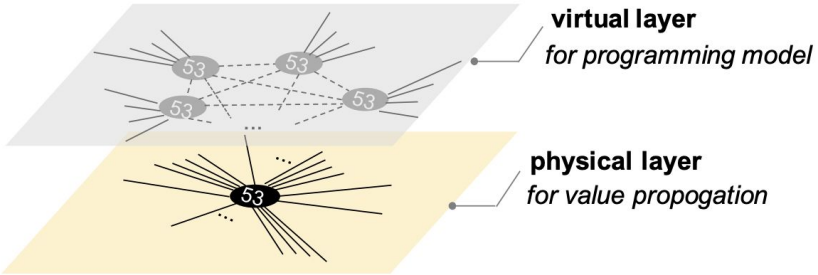
Tigr - Transforming Irregular Graphs into more Regular ones

- Claim that GPU-based graph algorithms suffer due to power law structure
- **Previous works addressed the issue by:**
  - **Warp segmentation**
  - **Altering the graph algorithms**

http://www.cs.ucr.edu/~anode001/asplos18.pdf

Tigr - Transforming Irregular Graphs into more Regular ones

- Claim that GPU-based graph algorithms suffer due to power law structure
- Previous works addressed the issue by:
  - Warp segmentation
  - Altering the graph algorithms
- **Tigr targets the fundamental issue of irregularity by transforming the graph *virtually***



http://www.cs.ucr.edu/~anode001/asplos18.pdf

CuSha

- **Use shards!!!**

https://dl.acm.org/doi/10.1145/2600212.2600227

CuSha

- Use shards!!!

- Representing graph with shards has improved I/O performance for disk based graph processing

https://dl.acm.org/doi/10.1145/2600212.2600227

CuSha

- Use shards!!!

- Representing graph with shards has improved I/O performance for disk based graph processing

- Shards allow contiguous placement of graph data required by a set of computation.

https://dl.acm.org/doi/10.1145/2600212.2600227

CuSha

- Use shards!!!

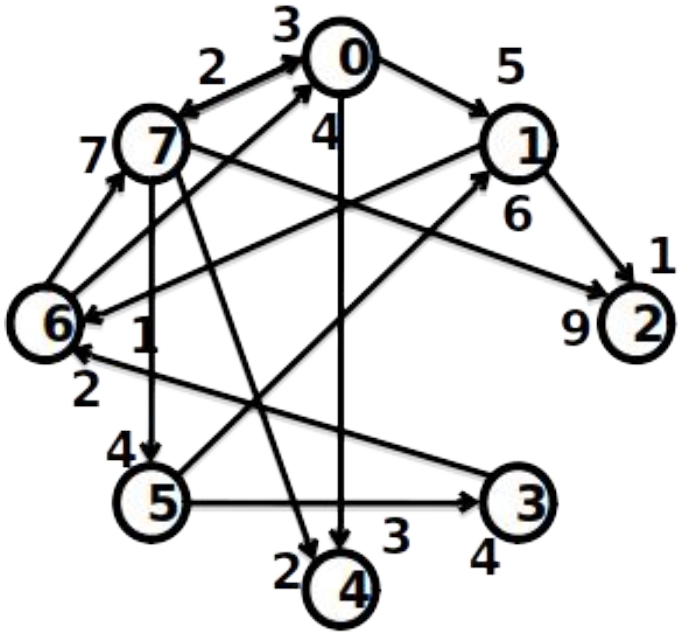
- Representing graph with shards has improved I/O performance for disk based graph processing

- Shards allow contiguous placement of graph data required by a set of computation.

- G-Shards representation employs shard concept and modifies it appropriately for GPU.

https://dl.acm.org/doi/10.1145/2600212.2600227

CuSha



Shard 0

SrcIndex	SrcValue	EdgeValue	DestIndex
0	x_0	5	1
1	x_1	1	2
5	x_5	6	1
5	x_5	4	3
6	x_6	4	0
7	x_7	3	0
7	x_7	9	2

Shard 1

SrcIndex	SrcValue	EdgeValue	DestIndex
0	x_0	3	4
0	x_0	2	7
1	x_1	1	6
3	x_3	2	6
6	x_6	7	7
7	x_7	2	4
7	x_7	4	5

VertexValues	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7

Motivation

- **We want to make graph algorithms run faster**

Motivation

- We want to make graph algorithms run faster
- **CUSHA uses G-shards to appropriately manage GPU resources**

Motivation

- We want to make graph algorithms run faster
- CUSHA uses G-shards to appropriately manage GPU resources
- **What if we convert graph to more regularized one using an intermediate representation and input to CUSHA**

Motivation

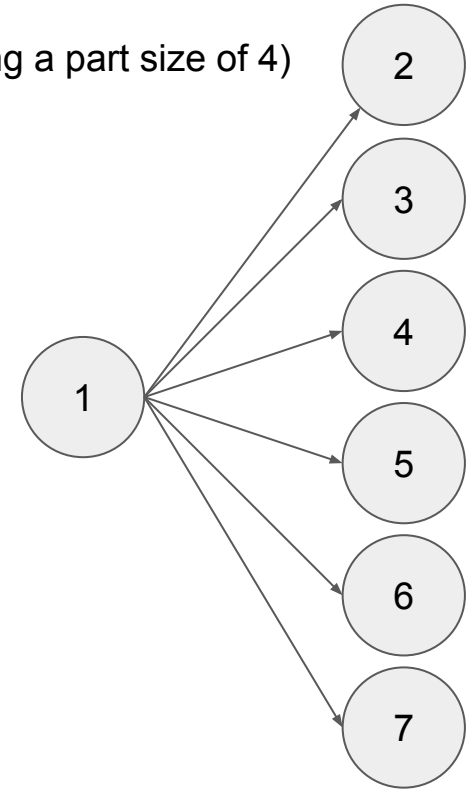
- We want to make graph algorithms run faster
- CUSHA uses G-shards to appropriately manage GPU resources
- What if we convert graph to more regularized one using an intermediate representation and input to CUSHA
- **This representation has lower out degree hence making it larger in size but regularized and efficient to process**

Hypothesis

- Can we achieve better performance from CuSha if we perform a physical graph transformation (UDT) that regularizes the graph before inputting it into CuSha?

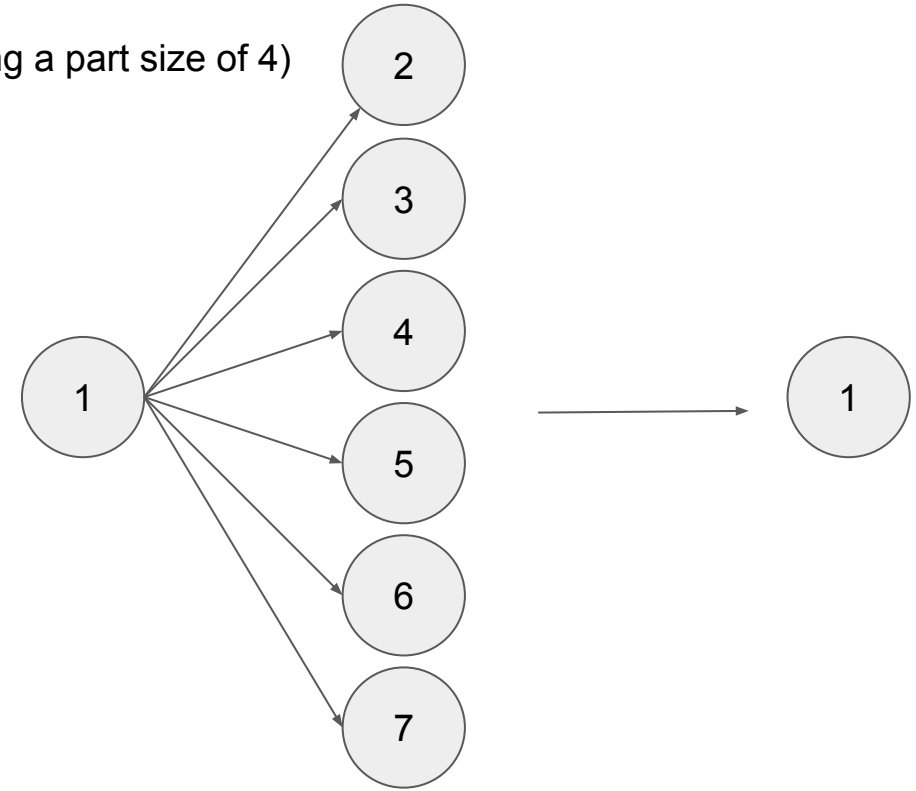
Uniform-Degree Tree Transformation

(Using a part size of 4)



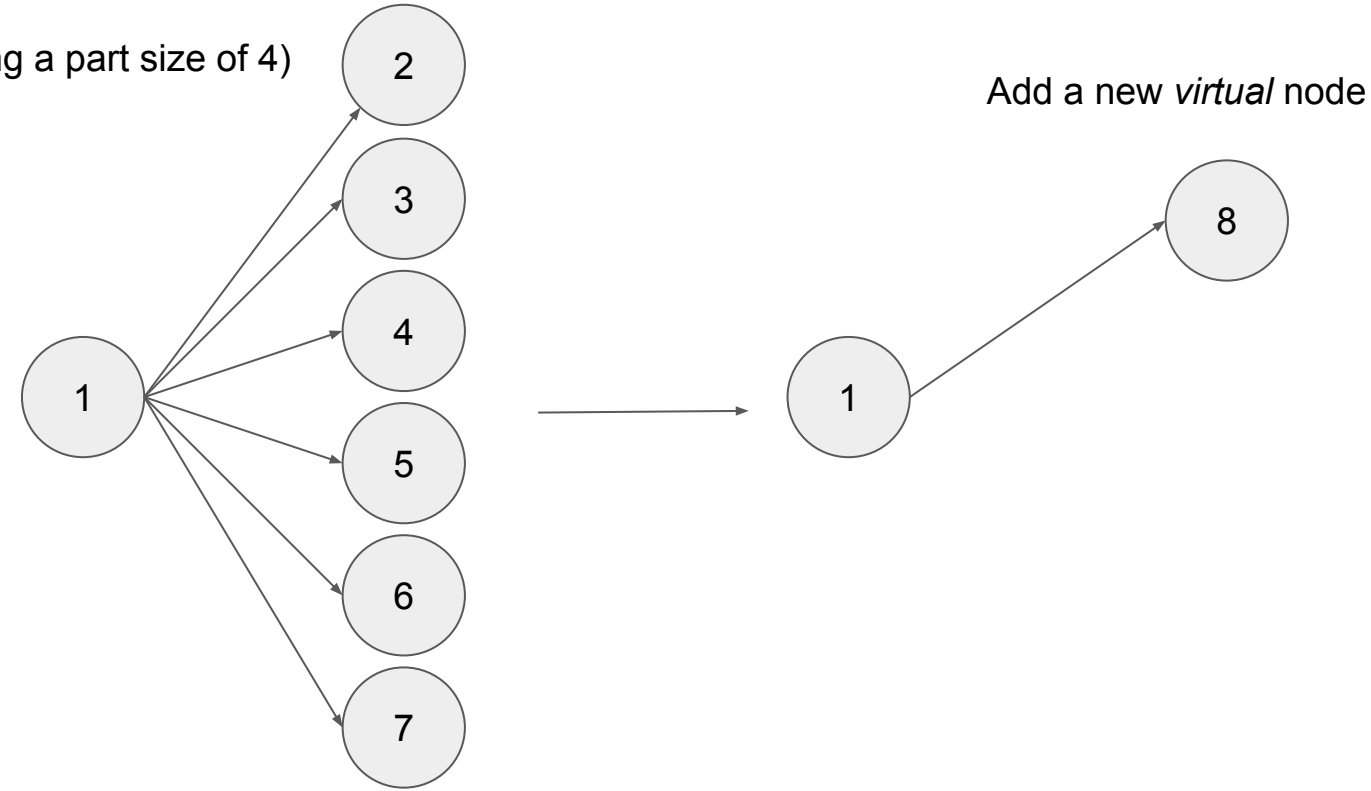
Uniform-Degree Tree Transformation

(Using a part size of 4)



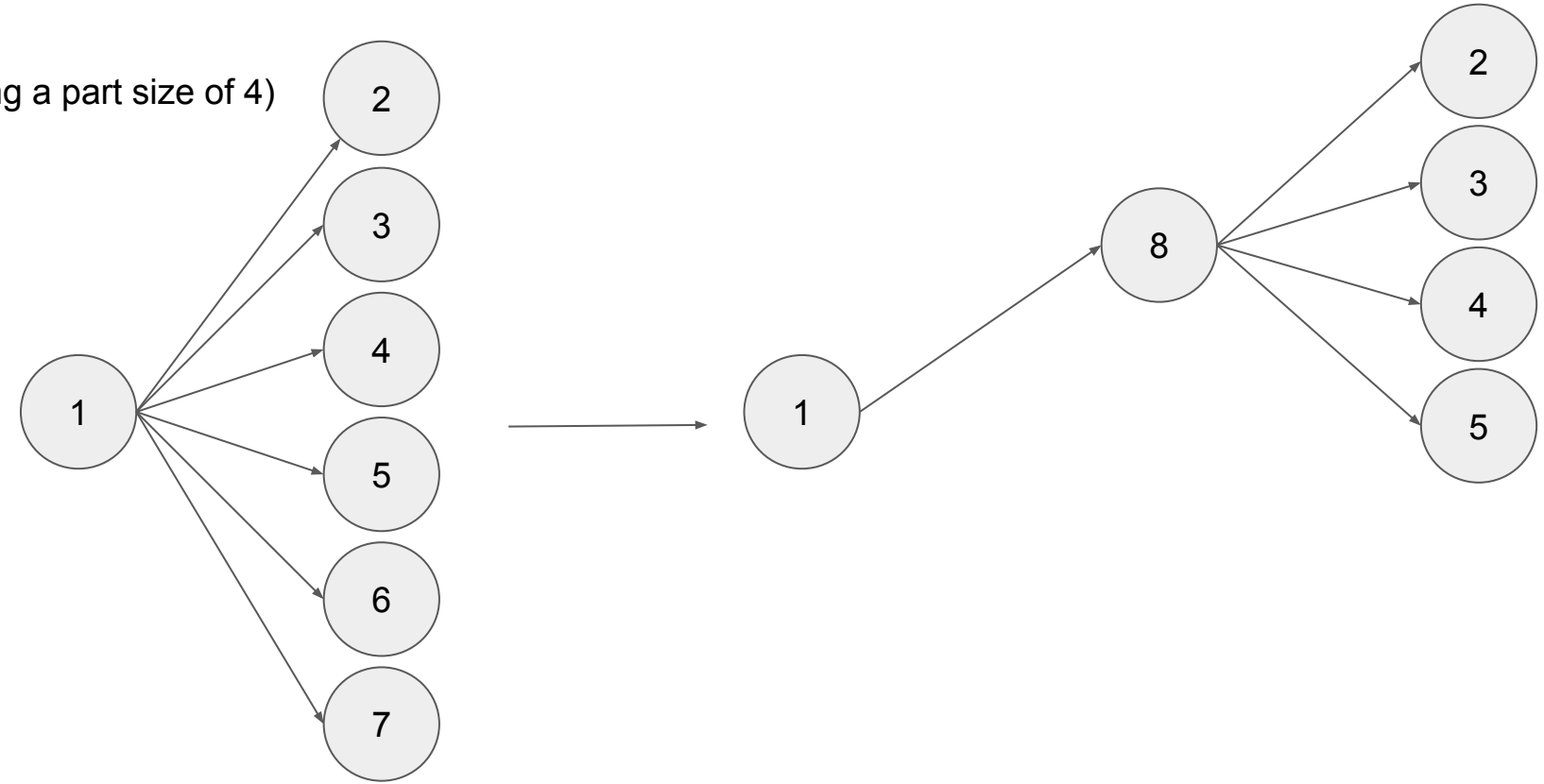
Uniform-Degree Tree Transformation

(Using a part size of 4)



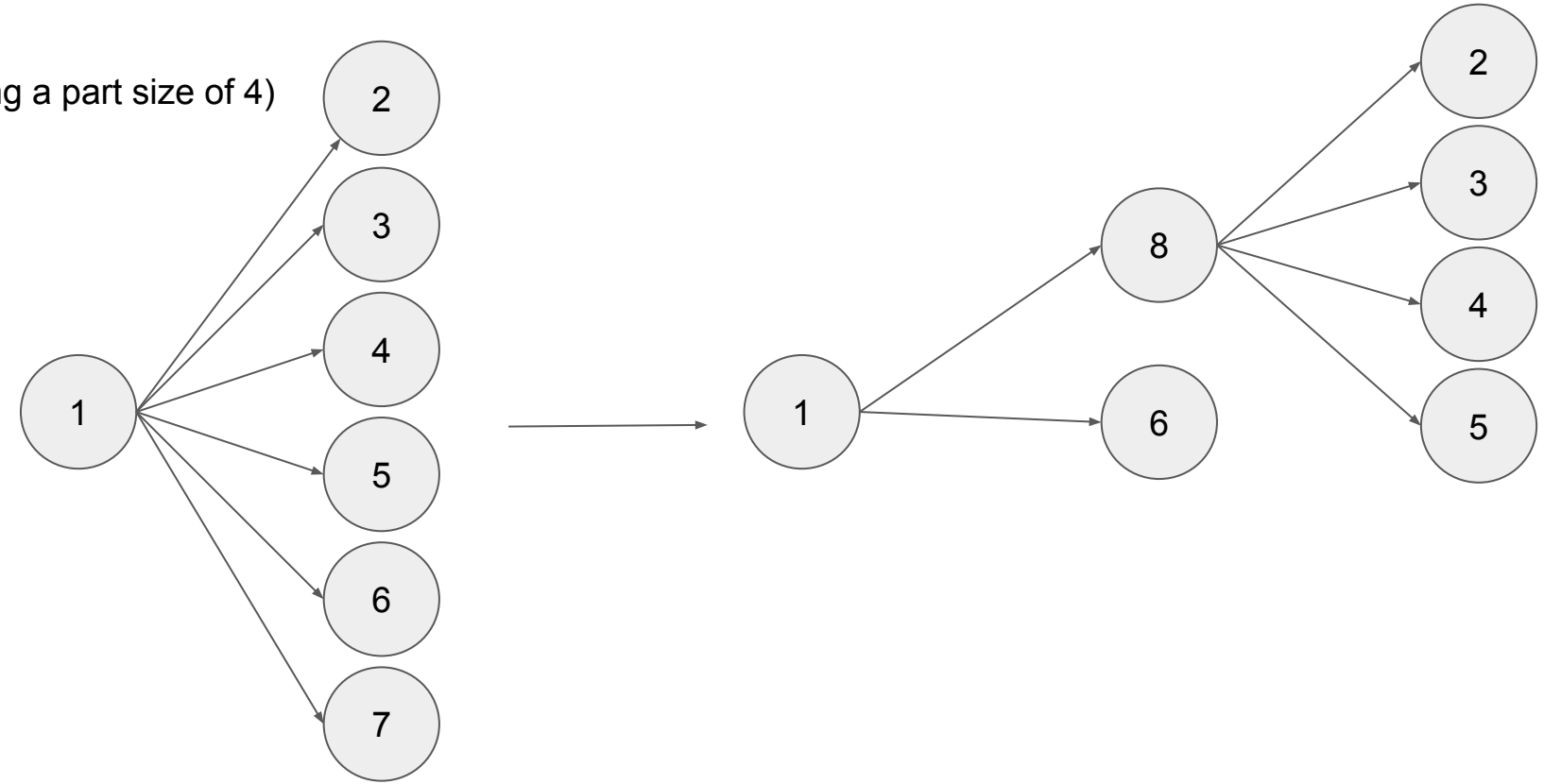
Uniform-Degree Tree Transformation

(Using a part size of 4)



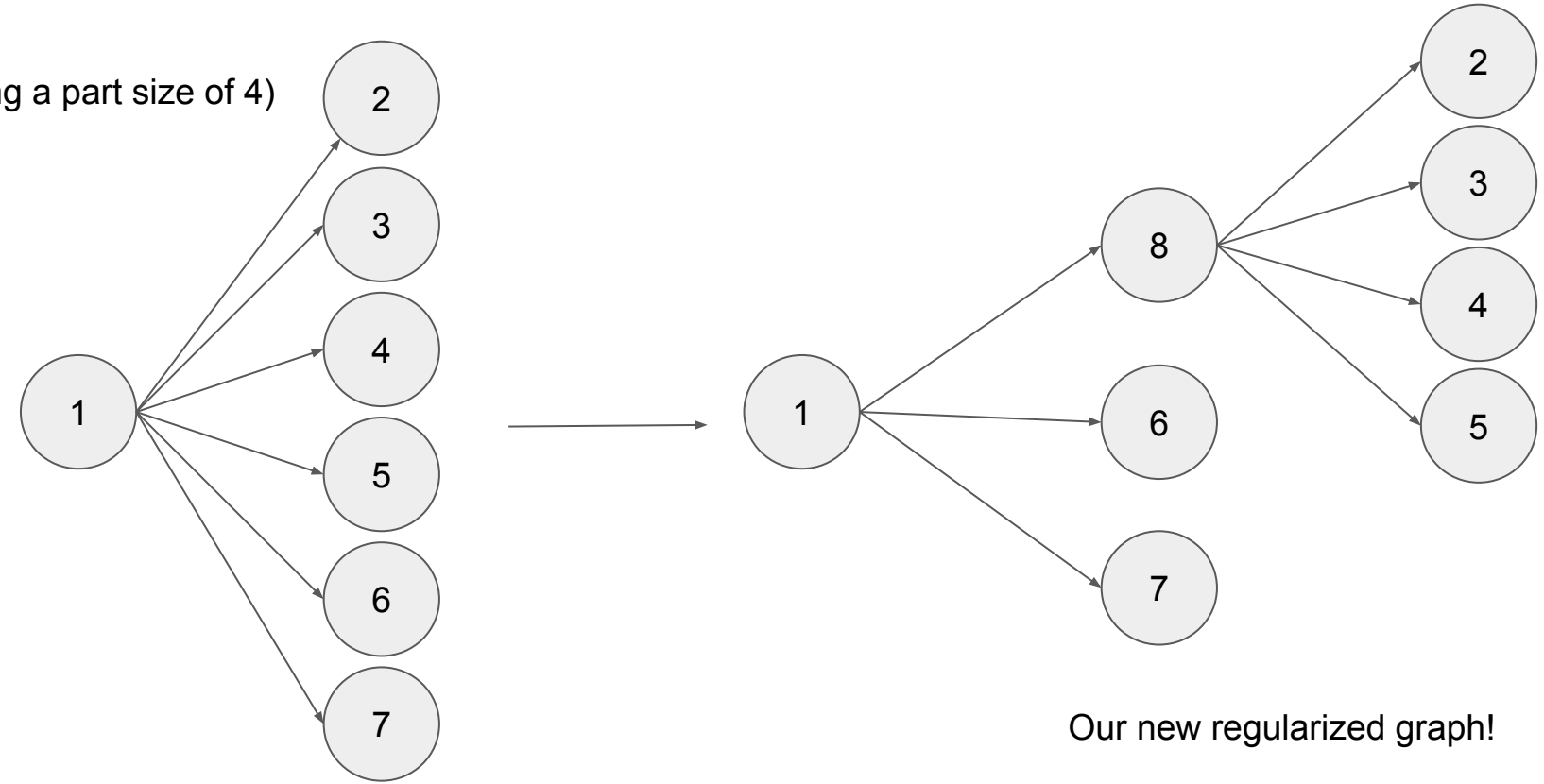
Uniform-Degree Tree Transformation

(Using a part size of 4)



Uniform-Degree Tree Transformation

(Using a part size of 4)



Our new regularized graph!

MACHINE CONFIGURATION

- **CPU: i5 6600k (3.50 GHz, 4 cores, 4 threads)**

MACHINE CONFIGURATION

- CPU: i5 6600k (3.50 GHz, 4 cores, 4 threads)
- **GPU: GTX 1060 6GB**

MACHINE CONFIGURATION

- CPU: i5 6600k (3.50 GHz, 4 cores, 4 threads)
- GPU: GTX 1060 6GB
- **Memory: 16 GB**

MACHINE CONFIGURATION

- CPU: i5 6600k (3.50 GHz, 4 cores, 4 threads)
- GPU: GTX 1060 6GB
- Memory: 16 GB
- **OS: Ubuntu 18.04**

Datasets

- Pokec social network[1]
  - 1632803 vertices, 30622564 edges
- Higgs twitter[2]
  - 456627 vertices, 14855842 edges
- Amazon product co-purchasing network[3]
  - 403394 vertices, 3387388 edges

1 - https://snap.stanford.edu/data/soc-Pokec.html
2 - https://snap.stanford.edu/data/higgs-twitter.html
3 - https://snap.stanford.edu/data/amazon0601.html

EXPERIMENTATION

- **Computation time for UDT-CuSha on BFS and SSSP**

EXPERIMENTATION

- Computation time for UDT-CuSha on BFS and SSSP
- **Computation time on Higgs, Pokec and Amazon**

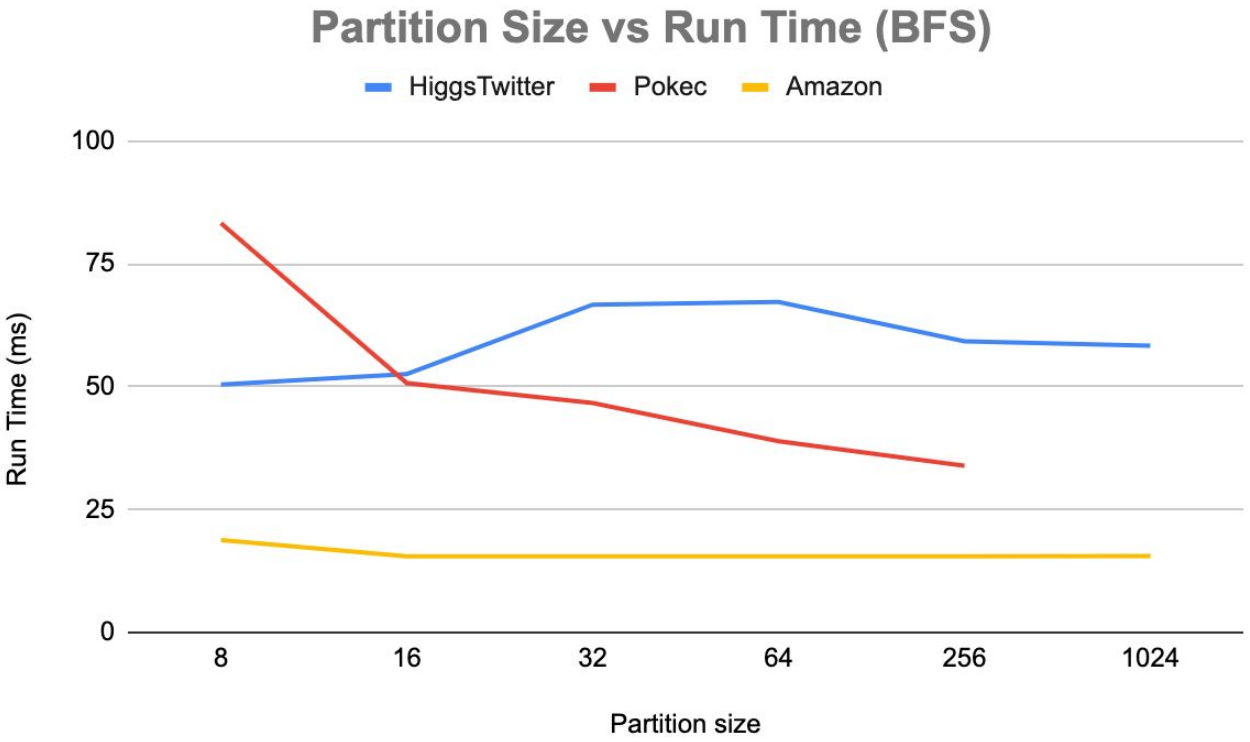
EXPERIMENTATION

- Computation time for UDT-CuSha on BFS and SSSP
- Computation time on Higgs, Pokec and Amazon
- **Computation time on 5 different part sizes**

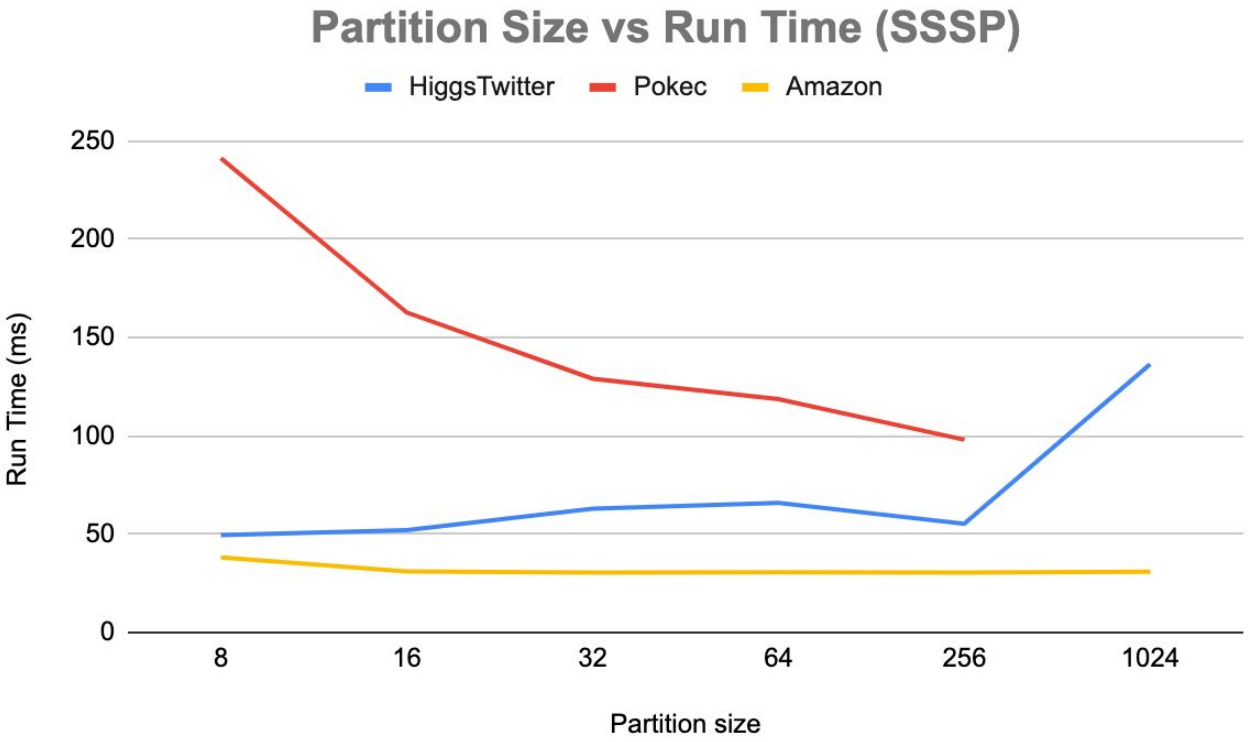
EXPERIMENTATION

- Computation time for UDT-CuSha on BFS and SSSP
- Computation time on Higgs, Pokec and Amazon
- Computation time on 5 different part sizes
- **Comparison with baseline (Tigr-V+ and CuSha)**

Experimentation Results



Experimentation Results



Comparisons with Baselines

Twitter	UDT-CuSha	UDT-CuSha	UDT-CuSha	UDT-CuSha	UDT-CuSha	UDT-CuSha	CuSha	Tigr
Part Size	8	16	32	64	256	1024	N/A	8
Time (ms)	**49.4730**	**52.0758**	**62.9978**	**65.9644**	**55.3284**	136.479	132.789	6.7826

Amazon	UDT-CuSha	UDT-CuSha	UDT-CuSha	UDT-CuSha	UDT-CuSha	UDT-CuSha	CuSha	Tigr
Part Size	8	16	32	64	256	1024	N/A	8
Time (ms)	38.1308	30.966	**30.4466**	30.5708	**30.4798**	30.8584	30.5586	7.182

Interpretations

- **Increasing part_size asymptotically leads to CuSha level performance**
  - **Intuitive as increasing part size means we have done less to alter the graph**

Interpretations

- Increasing part_size asymptotically leads to CuSha level performance
  - Intuitive as increasing part size means we have done less to alter the graph
- **Higgs vs. Amazon**
  - **As part_size increases, performance on Higgs decreases, performance on Amazon**
  - **Similar size graphs, however:**
    - **Higgs has 32.5 edges/node,**
    - **Amazon has 4.7 edges/node**
  - **Our approach works better on more dense graphs, which means regularization helps**

Interpretations

- Increasing part_size asymptotically leads to CuSha level performance
  - Intuitive as increasing part size means we have done less to alter the graph
- Higgs vs. Amazon
  - As part_size increases, performance on Higgs decreases, performance on Amazon
  - Similar size graphs, however:
    - Higgs has 32.5 edges/node,
    - Amazon has 4.7 edges/node
  - Our approach works better on more dense graphs, which means regularization helps
- **Tigr has incredible performance**
  - **Prevents physical transformation of graph while allowing computation on a normalized graph**

Interpretations

- Increasing part_size asymptotically leads to CuSha level performance
  - Intuitive as increasing part size means we have done less to alter the graph
- Higgs vs. Amazon
  - As part_size increases, performance on Higgs decreases, performance on Amazon
  - Similar size graphs, however:
    - Higgs has 32.5 edges/node,
    - Amazon has 4.7 edges/node
  - Our approach works better on more dense graphs, which means regularization helps
- Tigr has incredible performance
  - Prevents physical transformation of graph while allowing computation on a normalized graph
- **Trade-off between increased number of nodes and regularization**

Future work and things we didn’t have time for

- **UDT transformation is useful, but only for graphs with high irregularity**

Future work and things we didn’t have time for

- UDT transformation is useful, but only for graphs with high irregularity
- **Analysis of graph prior to execution of algorithms is very important**
  - **A graph that has small maximum outgoing edges will not benefit from UDT**

Future work and things we didn’t have time for

- UDT transformation is useful, but only for graphs with high irregularity
- Analysis of graph prior to execution of algorithms is very important
  - A graph that has small maximum outgoing edges will not benefit from UDT
- **Try adjusting virtual warp size in CuSha algorithm to match part size**

QUESTIONS?