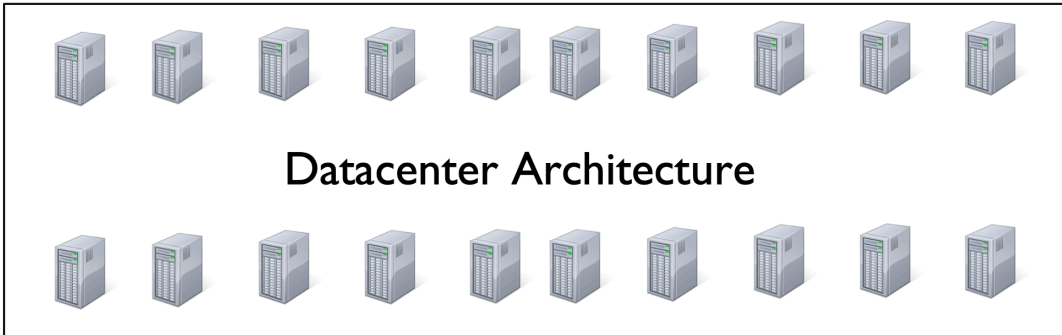


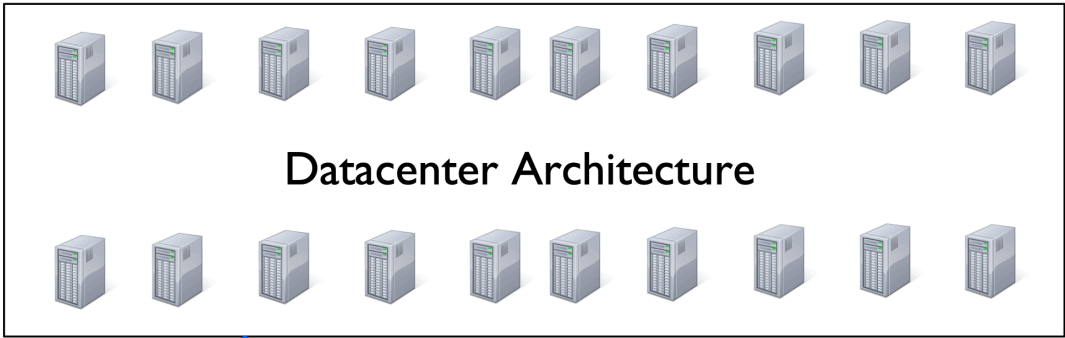
CS 744: Marius

Jason Mohoney
Spring 2025

Administrivia

- Shivaram gone for the week.
- Project check-ins pushed back. **Due April 7th**





Graph Machine Learning

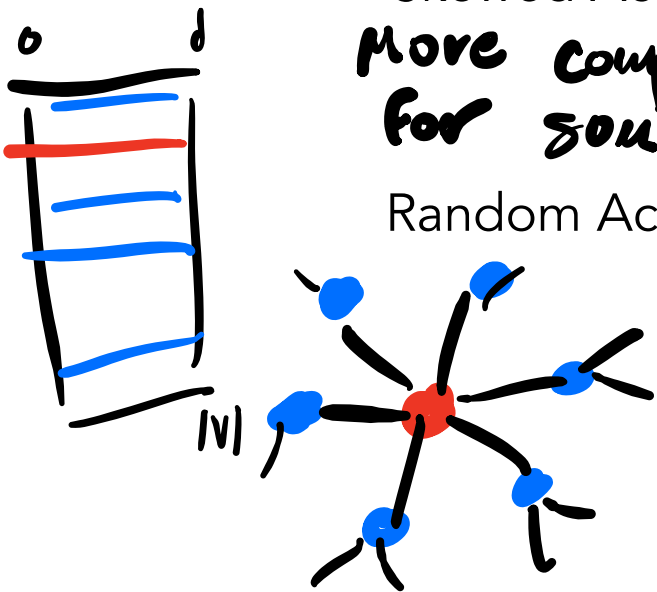
A uniquely challenging workload

Graph Processing

Skewed Access

*More compute
for some nodes*

Random Access



Machine Learning

Compute Heavy

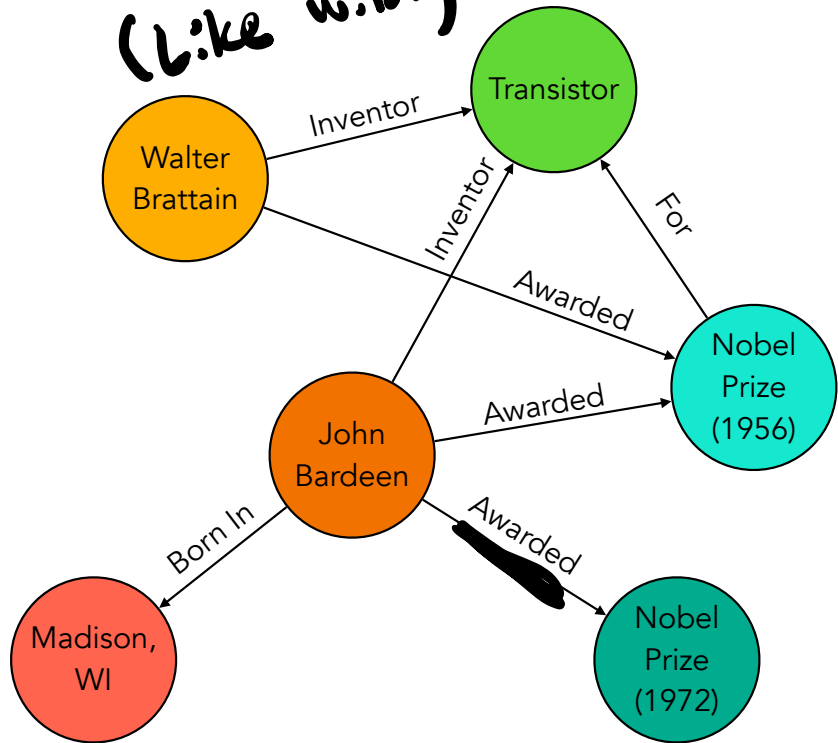
Need GPUs

Memory Heavy

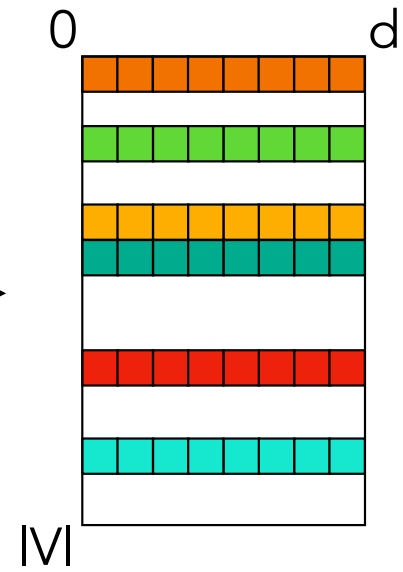
GPU memory is limited

Graph Embedding

Fact info (Like wik:)



Learn Representations!

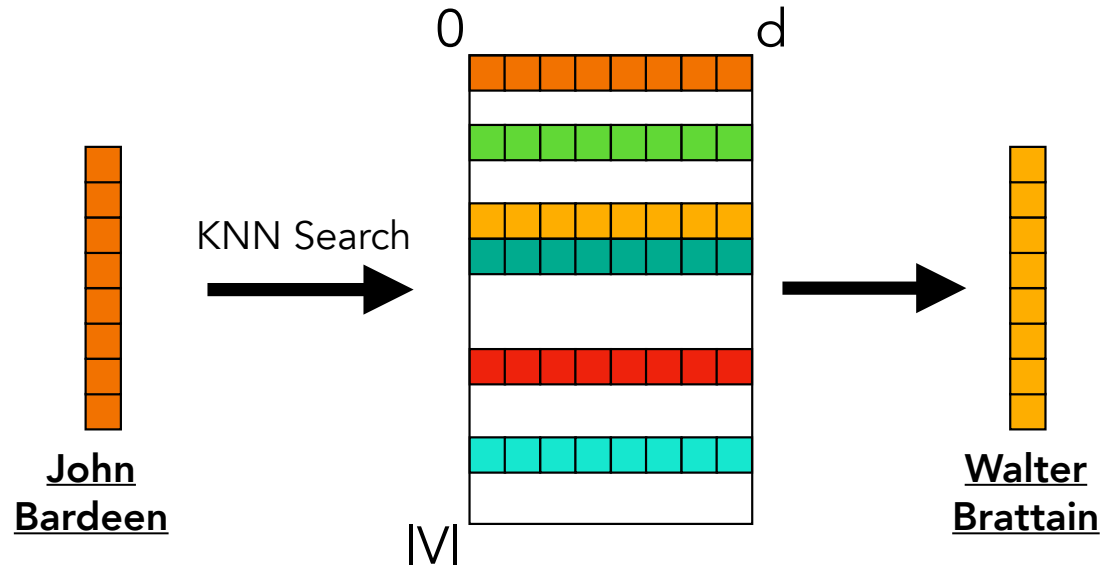
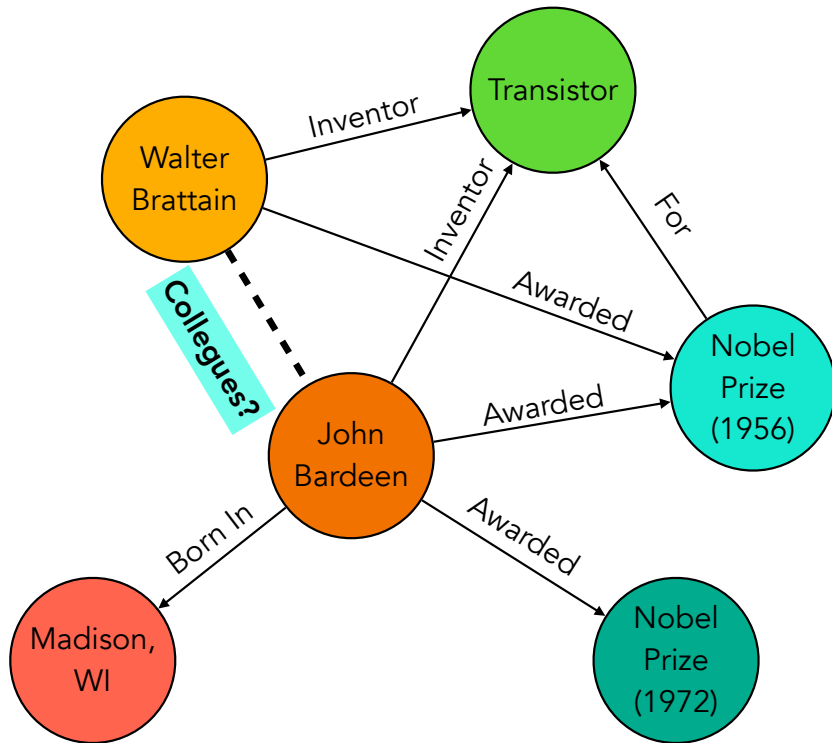


Knowledge Graph: $G = (V, R, E)$

Node Embedding Table
 Rel Embedding Table

Application: Link Prediction

Goal: Predict missing edges in the graph



Training Graph Embeddings

Contrastive Learning over edges

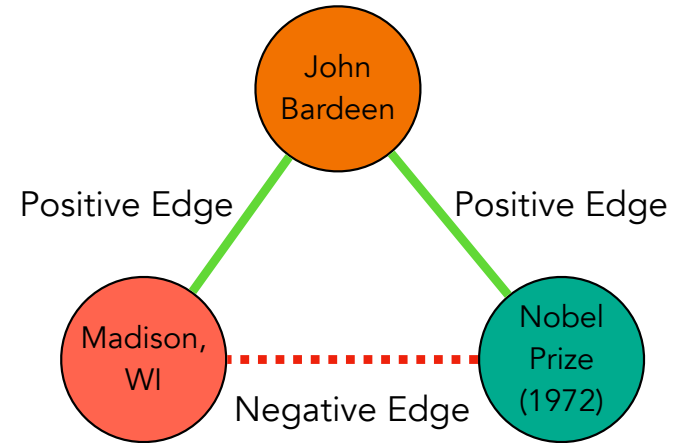
Maximize score for **positive** edges: $e \in E$

Minimize score for **negative** edges: $e' \notin E$

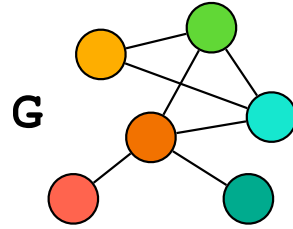
$$\mathcal{L} = \sum_{e \in E} \sum_{e' \in S'_e} \text{MSE}(f(e), f(e'))$$

DistMult: Score Function

$$f(e) = (\theta_s \odot \theta_r) \cdot \theta_d$$



Training Algorithm



```
 $\Theta$  = initEmbeddings( $G$ )  
for i range(num_batches):  
     $B$  = getBatch( $i$ )  
     $\theta$  = getEmbeddings( $B, \Theta$ )  
     $\Phi$  = computeGrads( $B, \theta$ )  
    updateEmbeddings( $\Theta, \Phi$ )
```

Training Algorithm

random init

$\Theta = \text{initEmbeddings}(G)$

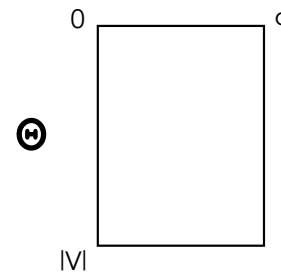
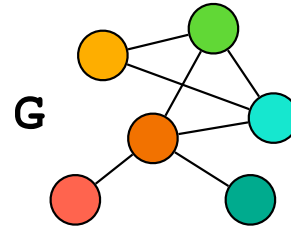
for i range(num_batches):

$B = \text{getBatch}(i)$

$\theta = \text{getEmbeddings}(B, \Theta)$

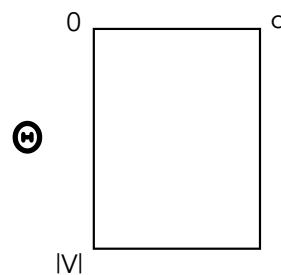
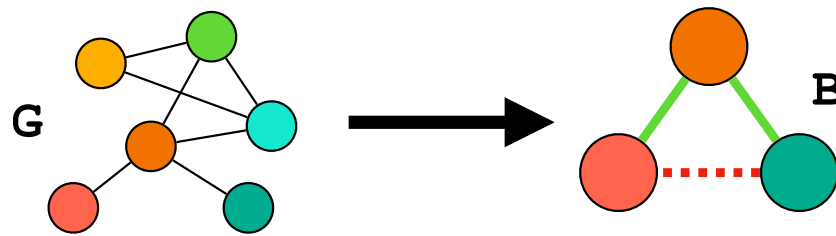
$\Phi = \text{computeGrads}(B, \theta)$

$\text{updateEmbeddings}(\Theta, \Phi)$



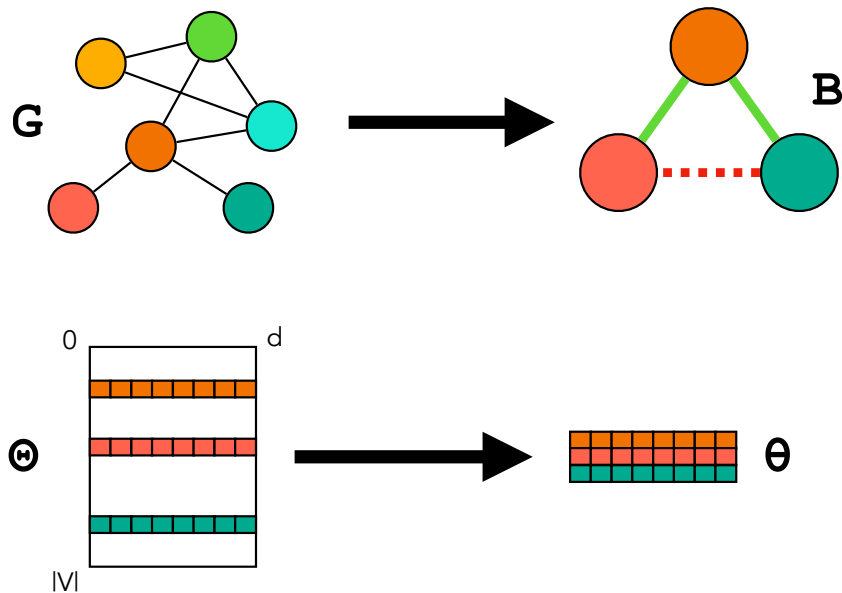
Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$   
for  $i$  range(num_batches):  
     $B = \text{getBatch}(i)$   
     $\theta = \text{getEmbeddings}(B, \Theta)$   
     $\Phi = \text{computeGrads}(B, \theta)$   
     $\text{updateEmbeddings}(\Theta, \Phi)$ 
```



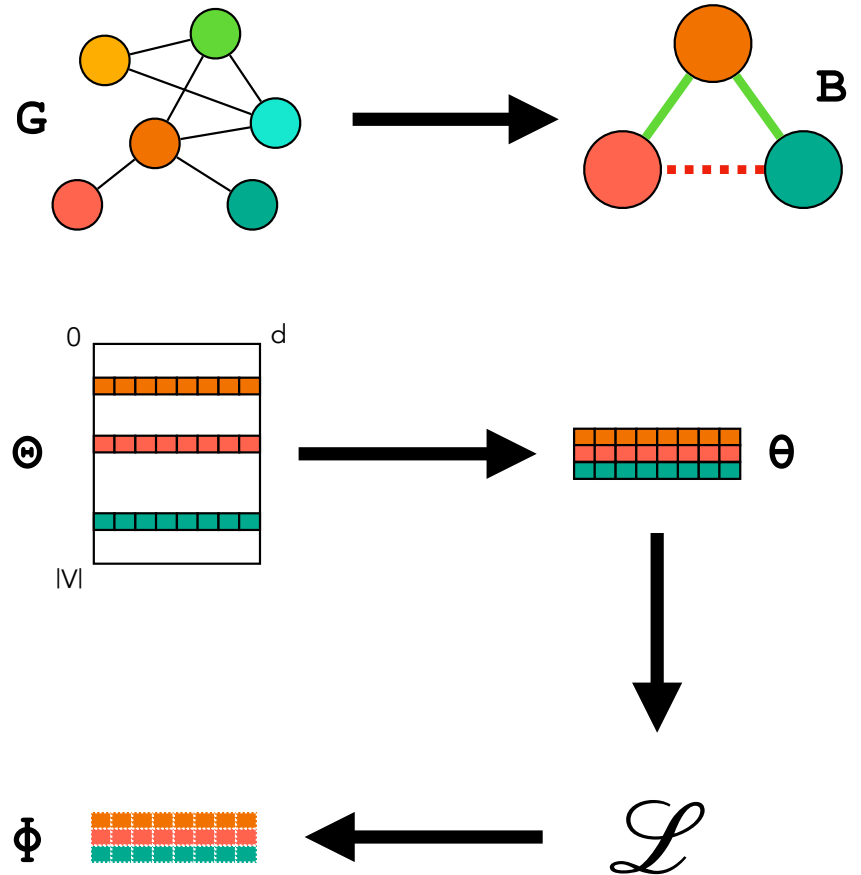
Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$   
for  $i$  range(num_batches):  
     $B = \text{getBatch}(i)$   
     $\theta = \text{getEmbeddings}(B, \Theta)$   
     $\Phi = \text{computeGrads}(B, \theta)$   
     $\text{updateEmbeddings}(\Theta, \Phi)$ 
```



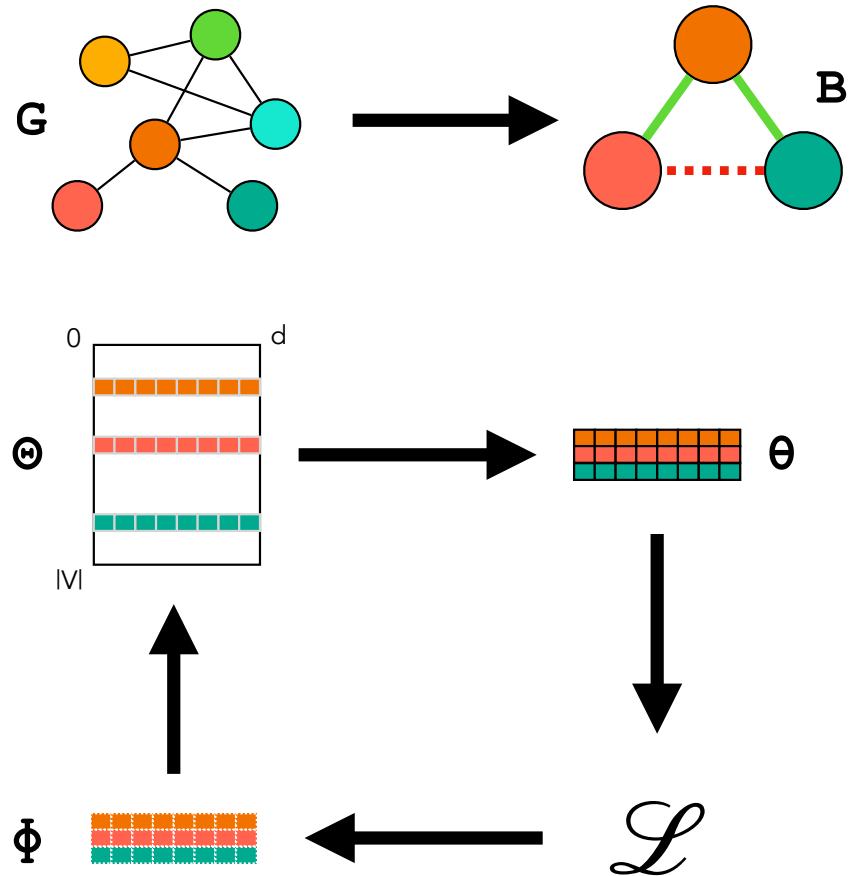
Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$   
for  $i$  range(num_batches):  
     $B = \text{getBatch}(i)$   
     $\theta = \text{getEmbeddings}(B, \Theta)$   
     $\phi = \text{computeGrads}(B, \theta)$   
     $\text{updateEmbeddings}(\Theta, \phi)$ 
```



Training Algorithm

```
 $\Theta = \text{initEmbeddings}(G)$   
  
for  $i$  range(num_batches):  
     $B = \text{getBatch}(i)$   
     $\theta = \text{getEmbeddings}(B, \Theta)$   
     $\phi = \text{computeGrads}(B, \theta)$   
     $\text{updateEmbeddings}(\Theta, \phi)$ 
```



Challenge: Scale

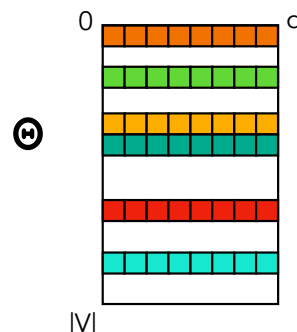
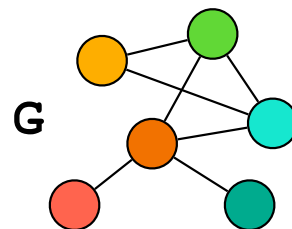
Example: Hyperlink-2012

Link structure of the internet circa 2012

$|V| = 3.5$ billion

$|E| = 128$ billion edges

A100: 80 GB



Edge List Size
 $|E| * 2 * 4 \text{ bytes} = \mathbf{1TB}$

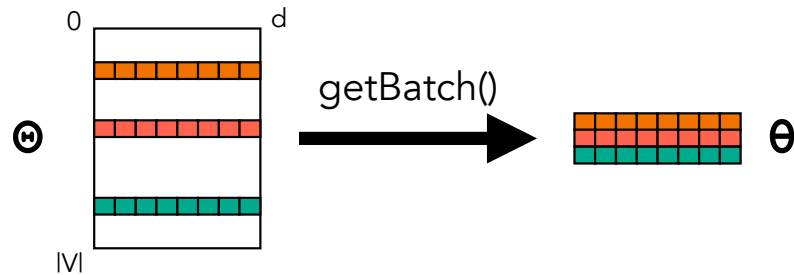
Embedding Table Size
 $d=200$
 $|V| * d * 4 \text{ bytes} = \mathbf{2.8TB}$

Large number of edges => Large amount of model computation => Need GPUs!

But edges and embeddings exceed GPU memory capacity!

Challenge: Random Access

Need to perform random access of the embedding table when forming batches



Okay if embeddings are in GPU or CPU memory, but prohibitively slow on disk!

Prior Solutions Bottlenecked by Data Movement

DGL-KE (Amazon)

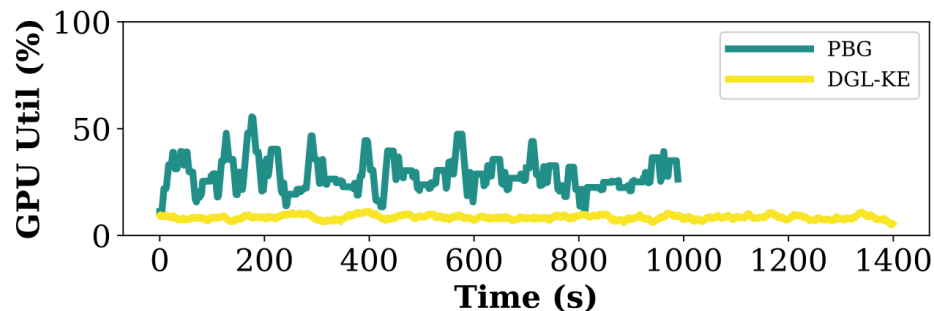


Figure 1: The GPU utilization of DGL-KE and PBG for one training epoch of ComplEx embeddings on the Freebase86m knowledge graph.

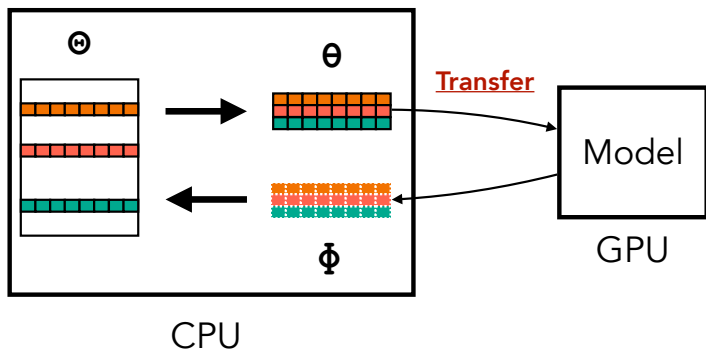
PBG (Facebook)

Why such poor utilization?

Prior Solutions Bottlenecked by Data Movement

DGL-KE (Amazon)

Synchronous training with batch prep on CPU



PBG (Facebook)

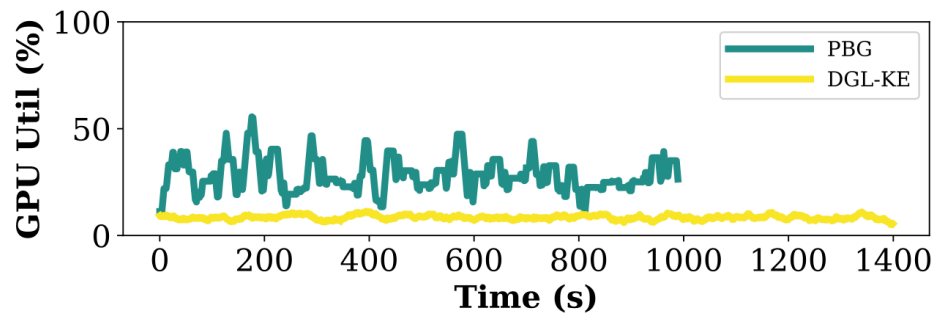


Figure 1: The GPU utilization of DGL-KE and PBG for one training epoch of ComplEx embeddings on the Freebase86m knowledge graph.

Why such poor utilization?

Prior Solutions Bottlenecked by Data Movement

DGL-KE (Amazon)

Synchronous training with batch prep on CPU

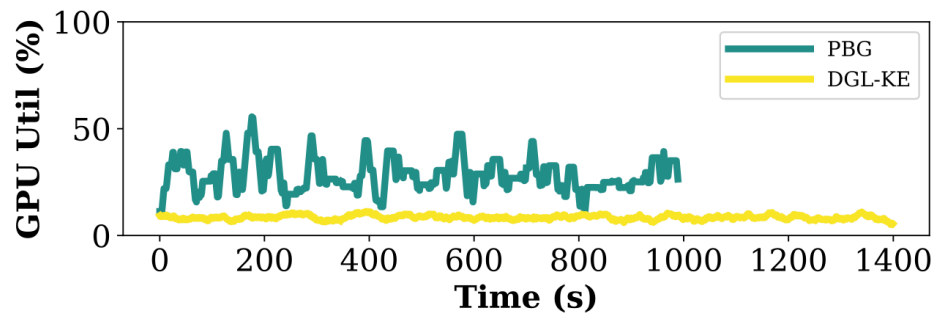
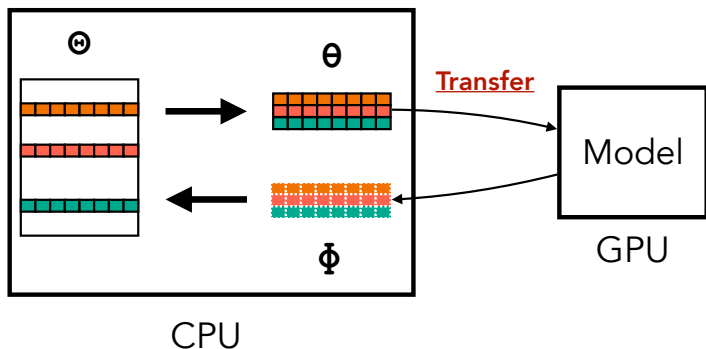
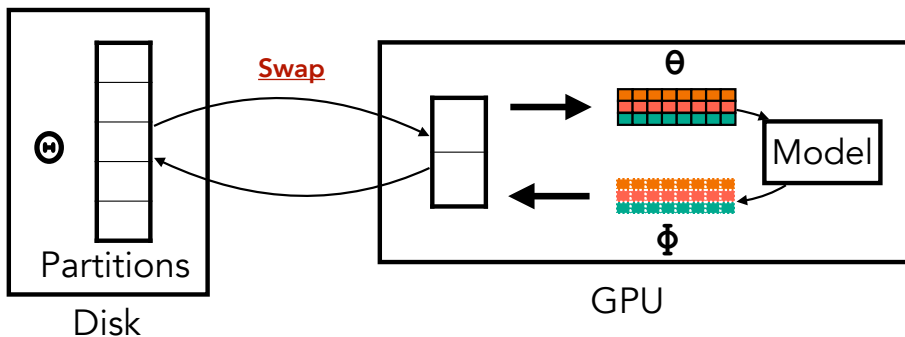


Figure 1: The GPU utilization of DGL-KE and PBG for one training epoch of Complex embeddings on the Freebase86m knowledge graph.

PBG (Facebook)

Partitioned synchronous training with batch prep on GPU



Why such poor utilization?

DGL-KE: Sync training
+ Transfer & batch prep on CPU

PBG: Swapping of partitions

Marius

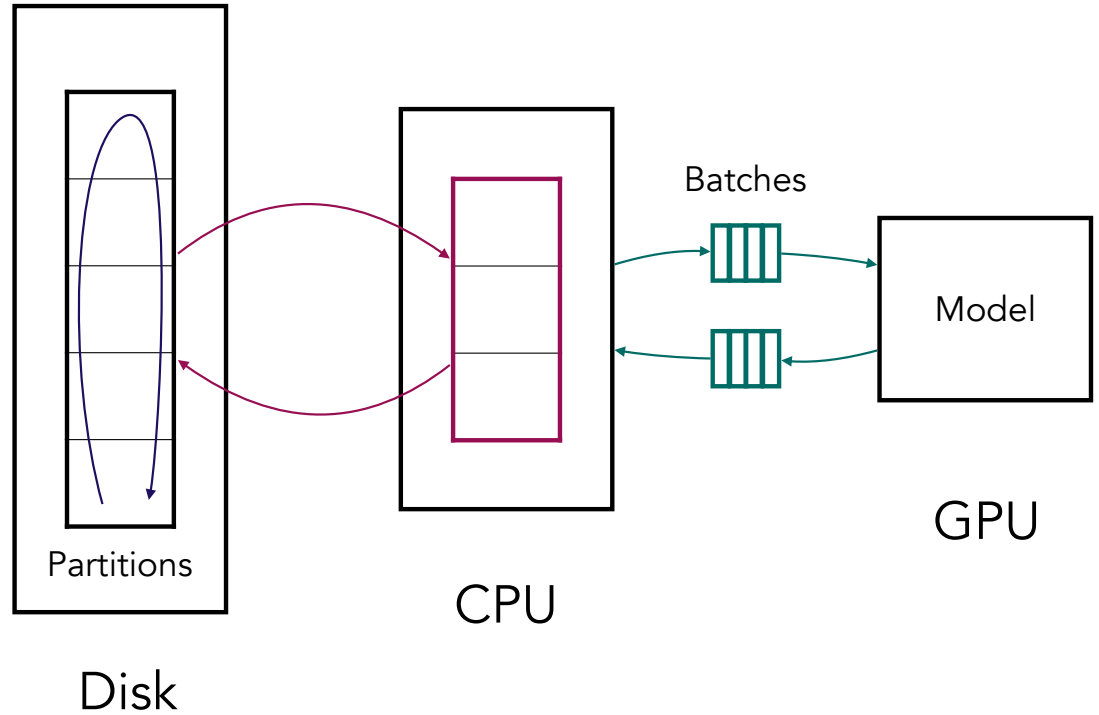
1. Mitigate CPU-GPU transfer stalls through pipelined training

Fixes DGL-KE

2. Mitigate Disk-CPU transfer stalls through partition buffer

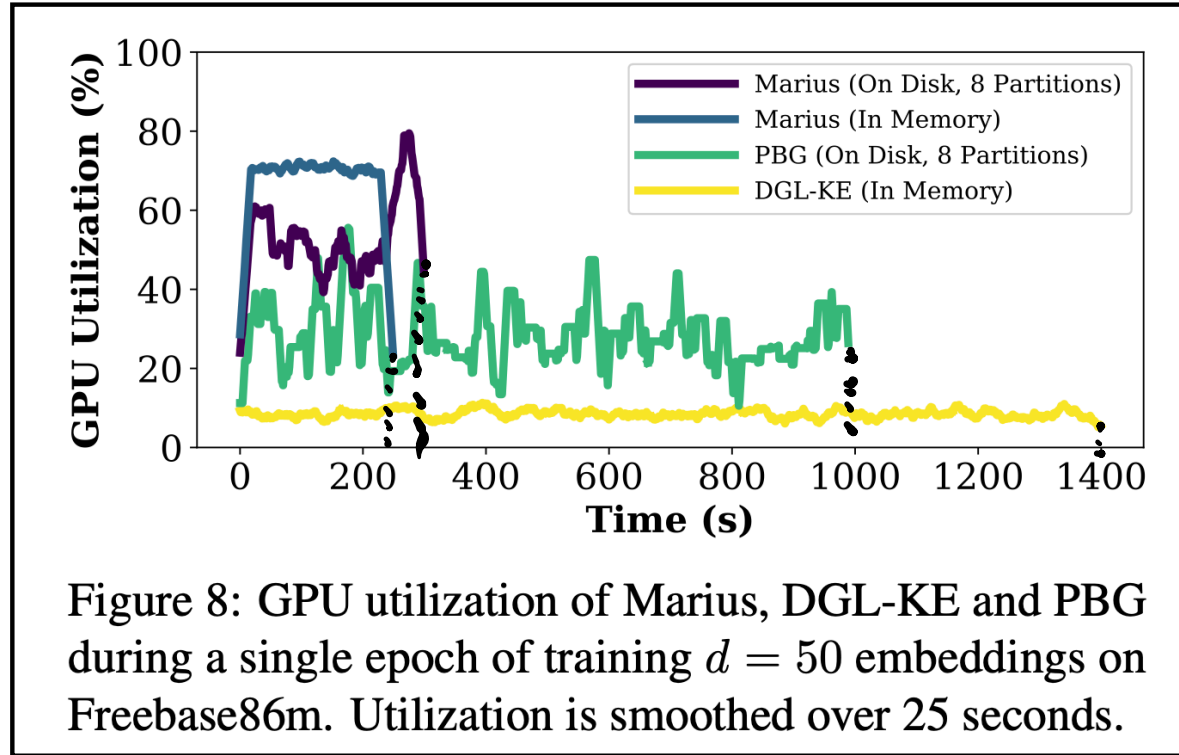
Fixes PBG

3. Minimize Disk IO through BETA ordering



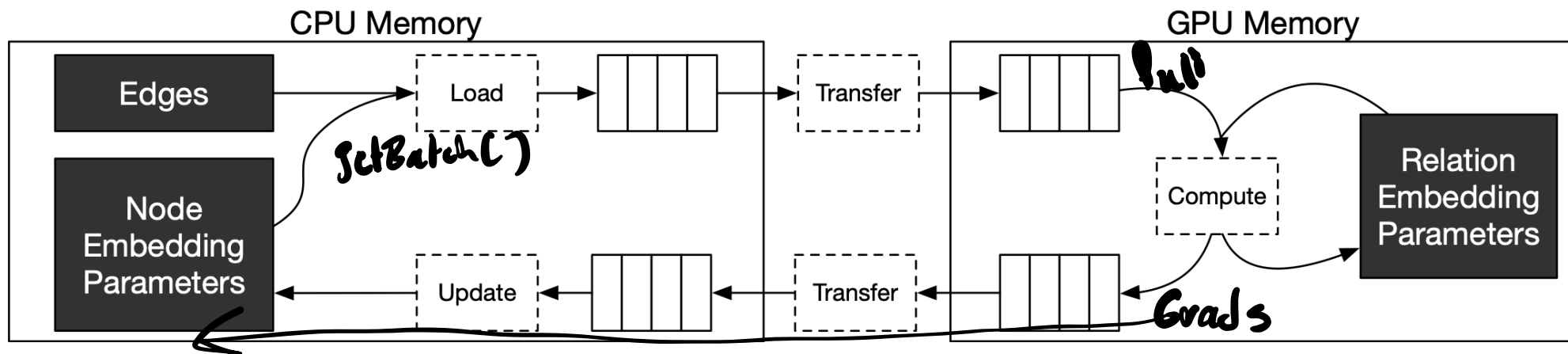
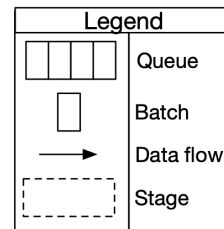
Marius

1. Mitigate CPU-GPU transfer stalls through **pipelined training**
2. Mitigate Disk-CPU transfer stalls through **partition buffer**
3. Minimize Disk IO through **BETA ordering**



Pipelined Architecture

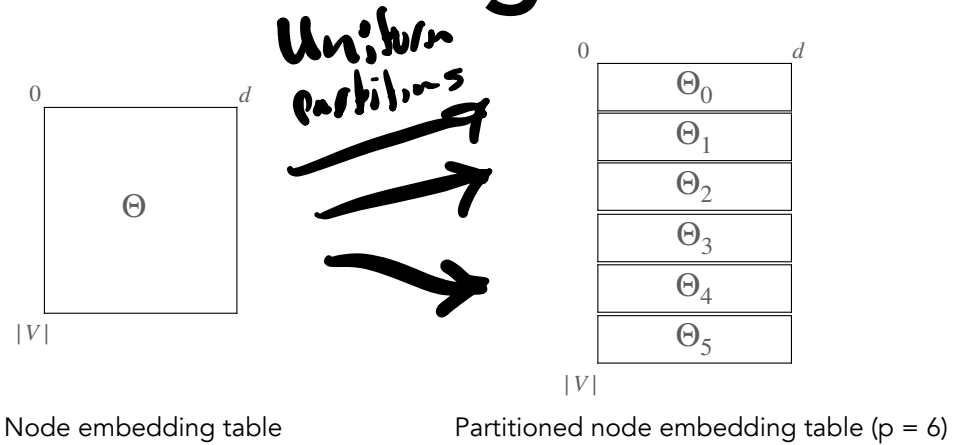
1. Bounded Staleness
2. Putting Relation Embeds on GPU



Out-of-core Training

Node Embedding Partitions

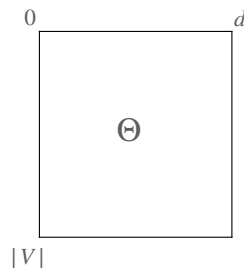
Node embeddings are partitioned uniformly into p disjoint partitions.



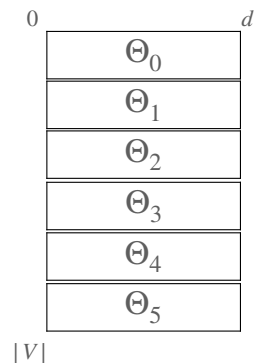
Out-of-core Training

Node Embedding Partitions

Node embeddings are partitioned uniformly into p disjoint partitions.



Node embedding table

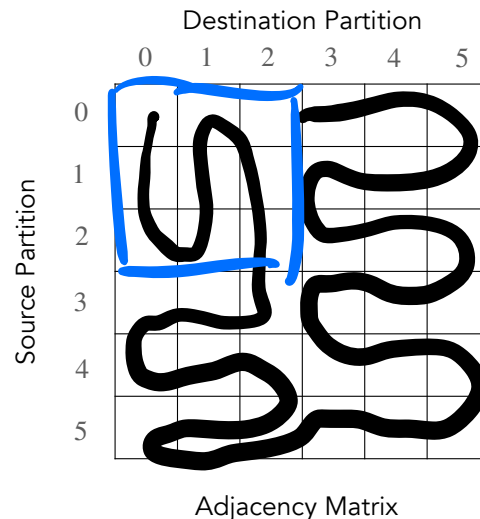


Partitioned node embedding table ($p = 6$)

Edge Buckets

Edge bucket (i,j) contains all edges with a source in partition i and a destination in partition j

To train, we need to iterate over all edges, we need to iterate over all edge buckets



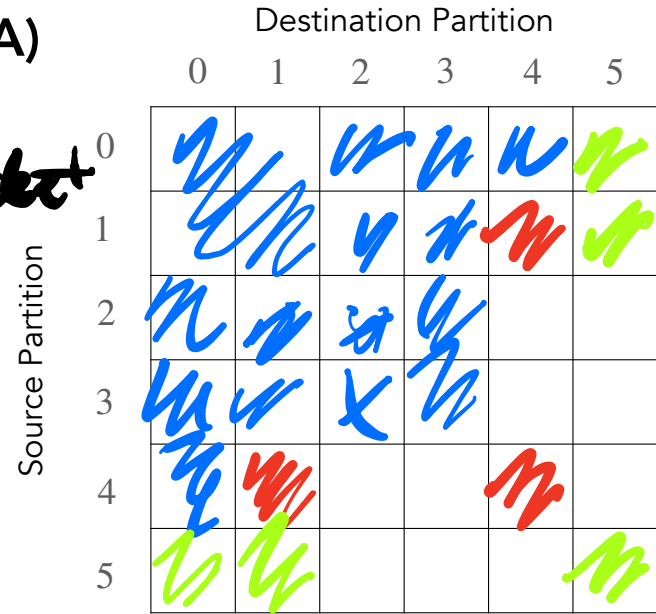
Adjacency Matrix

Buffer-aware Edge Traversal Algorithm (BETA)

Idea: Swap in partition that brings the most edge buckets

BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed



Partitions in Buffer

0	1	
---	---	--

 $c = 3$

4

5

Partitions on disk

Θ_0	Θ_1	Θ_2	Θ_3	Θ_4	Θ_5
------------	------------	------------	------------	------------	------------

 $p = 6$

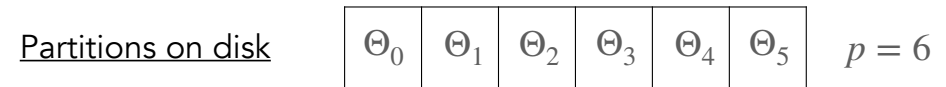
Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed

0 swaps*

		Destination Partition					
		0	1	2	3	4	5
Source Partition	0	■					
	1	■					
	2	■					
	3						
	4						
	5						



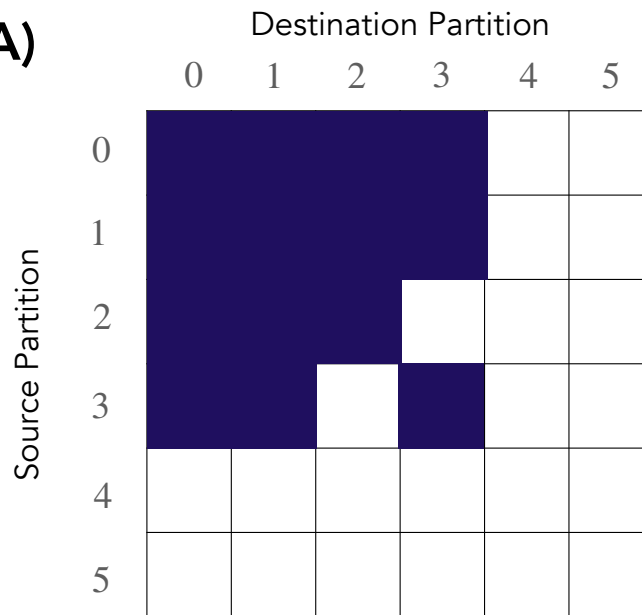
* Not counting initialized buffer

Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

1. Randomly initialize buffer
- 2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets**
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed

1 swap



Partitions in Buffer



Partitions on disk

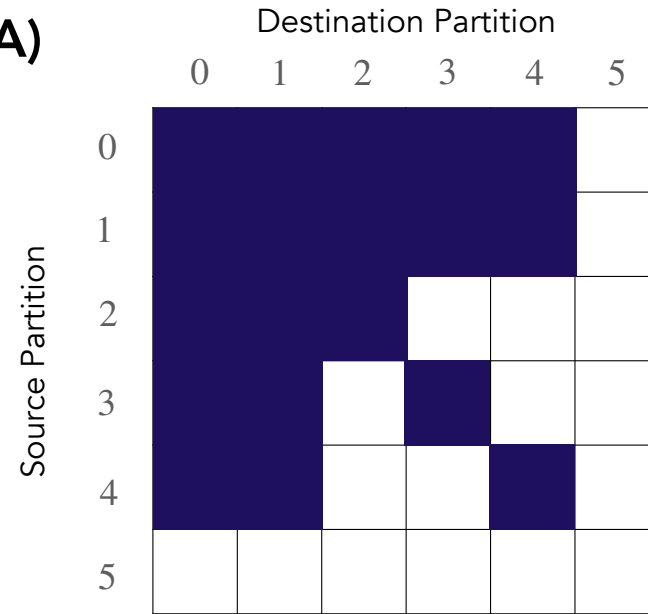


Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed

2 swaps



Partitions in Buffer

\ominus_0	\ominus_1	\ominus_4
-------------	-------------	-------------

 $c = 3$

Partitions on disk

\ominus_0	\ominus_1	\ominus_2	\ominus_3	\ominus_4	\ominus_5
-------------	-------------	-------------	-------------	-------------	-------------

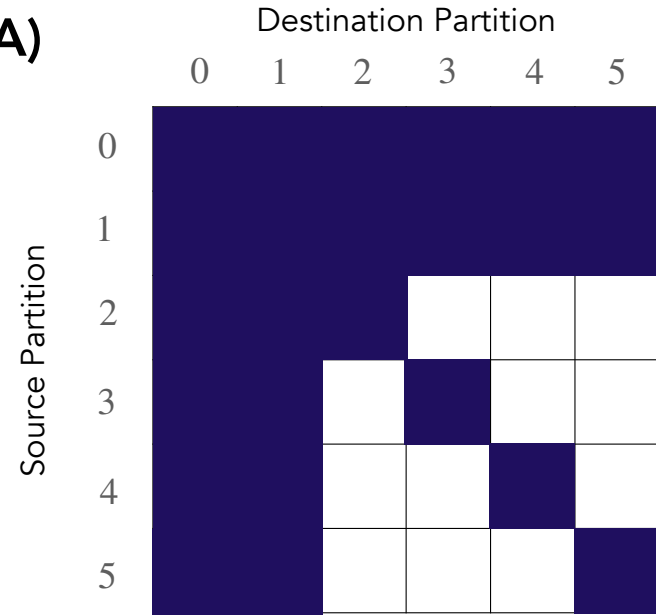
 $p = 6$

Buffer-aware Edge Traversal Algorithm (BETA)

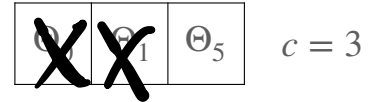
BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed

3 swaps



Partitions in Buffer



Partitions on disk

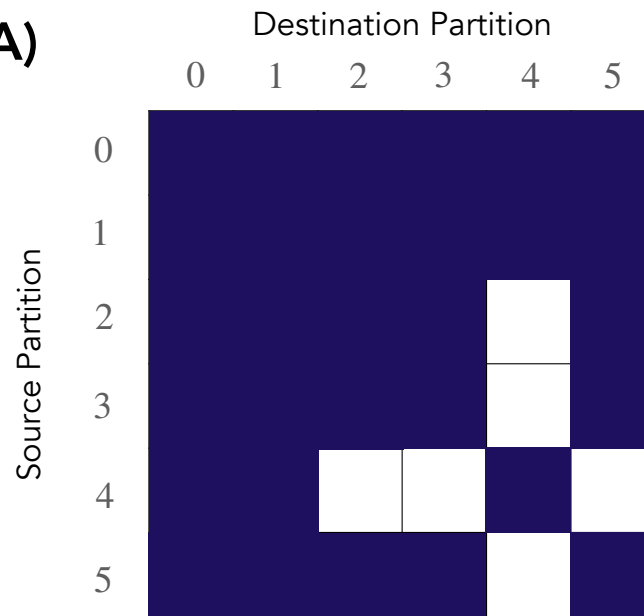


Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed

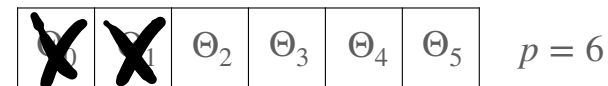
5 swaps



Partitions in Buffer



Partitions on disk

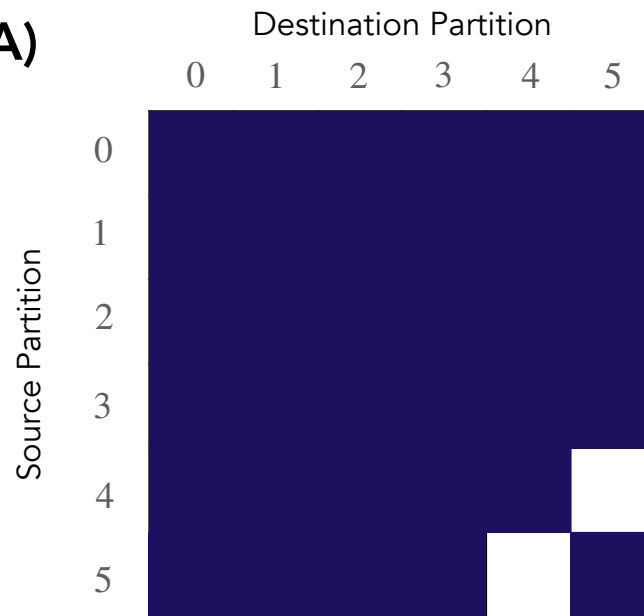


Buffer-aware Edge Traversal Algorithm (BETA)

BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed

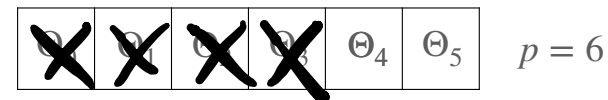
6 swaps



Partitions in Buffer



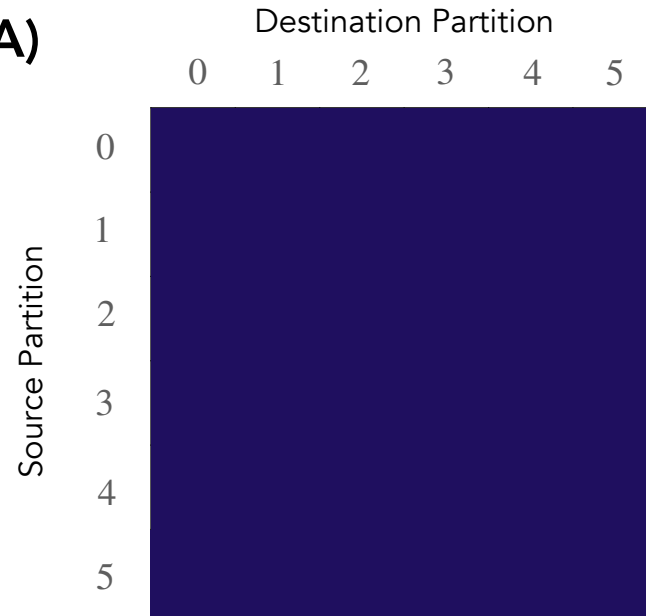
Partitions on disk



Buffer-aware Edge Traversal Algorithm (BETA)

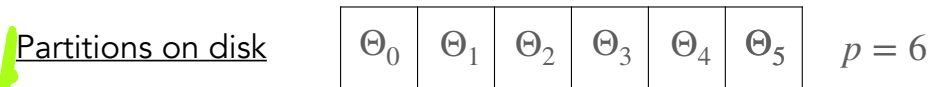
BETA Ordering

1. Randomly initialize buffer
2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets
3. Fix a new $c - 1$ partitions and repeat until all edge buckets have been processed



7 swaps

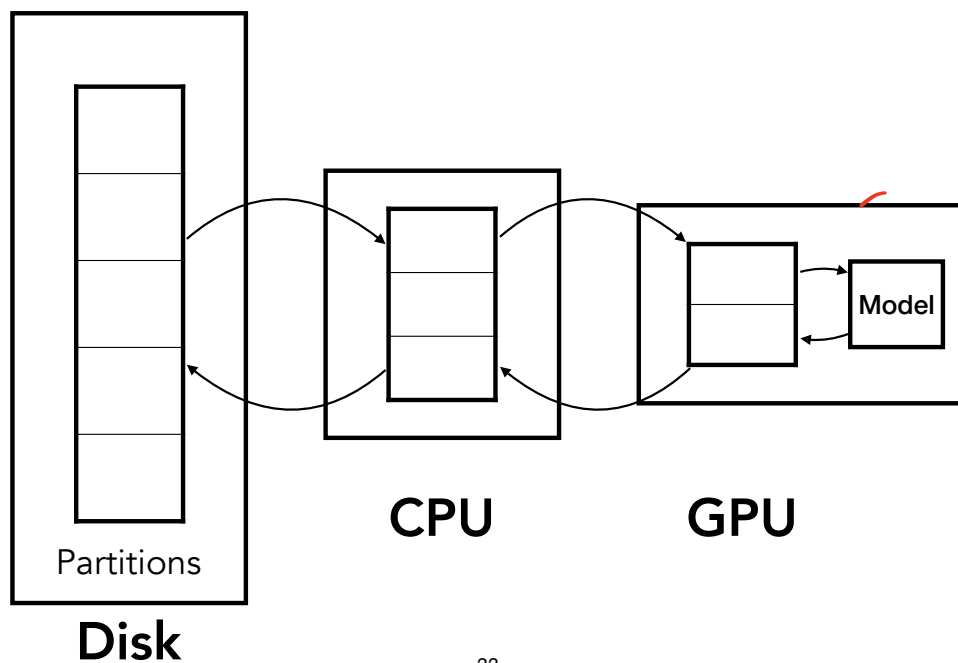
13 swaps for Hilbert



Discussion

Consider a three-tiered design where partitions are cached in the **CPU and GPU**.

1. What are the advantages & disadvantages of this design?
2. What changes are needed to the training algorithm?



Discussion

1. What are the advantages & disadvantages of the three-tier design?

+ Better GPU util b.c. less transfer

+/- reduces pipeline complexity but increases swapping complexity

- decreases GPU util during swaps - decreased working GPU memory

2. What changes are needed to the training algorithm?

- Pipeline is gone

- 3-level Beta Algorithm

↓
reduced batch size
- Reduced MRR due to batch size of sampling bias
- CPUs are idle

Next Steps

class

- Next ~~week~~: Vector Databases
- Project check-ins due April 7th

CS 7372

2:30-3:30 Tuesday

11-12 Wednesday