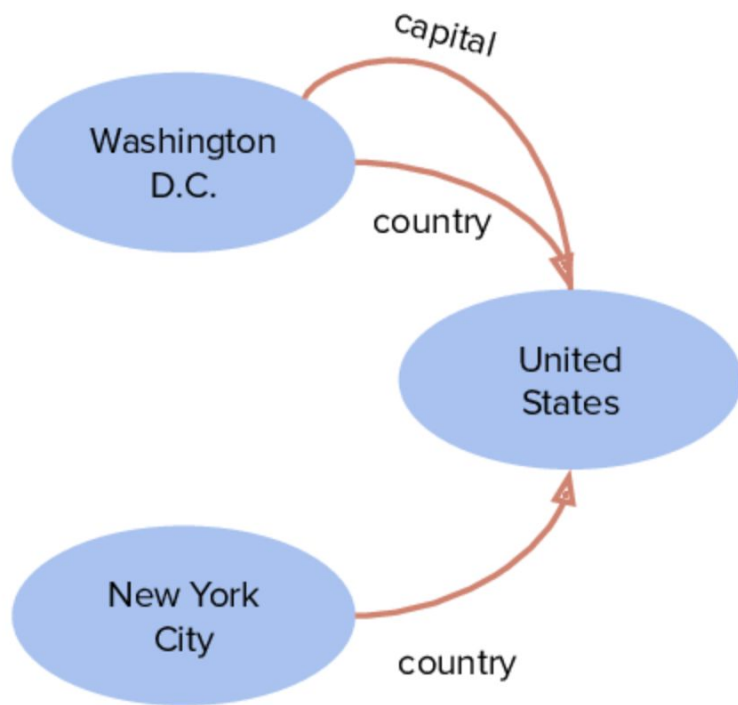


Marius: Learning Massive Graph Embeddings on a Single Machine

**Jason Mohoney and Roger Waleffe, *University of Wisconsin-Madison*;
Henry Xu, *University of Maryland, College Park*; Theodoros Rekatsinas
and Shivaram Venkataraman, *University of Wisconsin-Madison***

Review by Kliment Serafimov
For 6.827
April 26th 2022



Embedding

Washington D.C.	New York City	United States	country	capital
1	1	5	1	2
2	2	6	2	2
4	5	4	4	5
5	4	4	1	1
6	7	8	4	5
7	7	7	5	5

Figure 2. Starting from a knowledge graph, embedding methods generate representations of the elements of the knowledge graph that are embedded in a vector space. For example, these representations could be vectors... [Expand](#)



Problem and Motivation: Learning Graph Embeddings

- Learning graph embeddings is widely used in industry for a variety of downstream tasks.
- This paper focuses on **link prediction**.

Set up:

- Parameters: 1 high-dimensional vector $\in \mathbb{R}^{400}$ per node and per edge type.
- Training: minimize contrastive loss:
- Example $f(\theta_s, \theta_r, \theta_d) = \theta_s \text{diag}(\theta_r) \theta_d$; makes $f(e) \sim 1$ if $e \in E$, ~ 0 otherwise
- Second term is approximated.

$$\mathcal{L} = - \sum_{s,r,d \in E} (f(\mathbf{e}_\theta) - \log(\sum_{s',r',d' \notin E} e^{f(\mathbf{e}_{\theta'})})) \quad (1)$$

where $\mathbf{e}_\theta = (\theta_s, \theta_r, \theta_d)$ and $\mathbf{e}_{\theta'} = (\theta'_s, \theta'_r, \theta'_d)$.

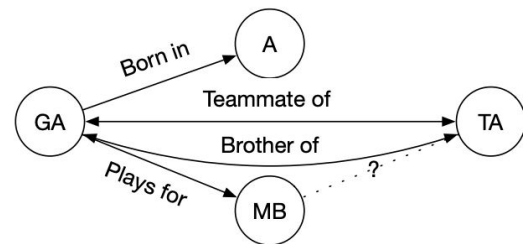
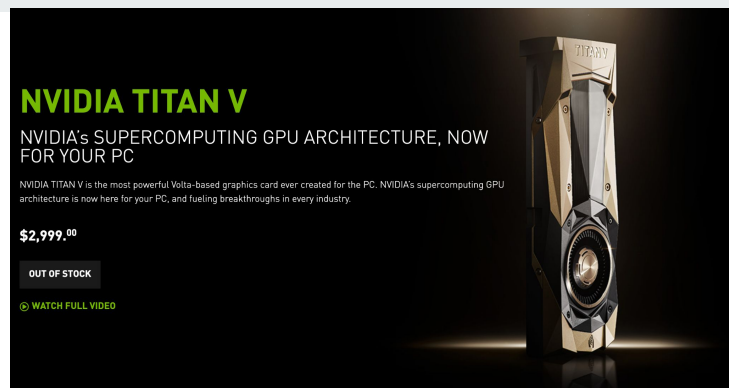


Figure 2: A sample knowledge graph.

Scaling issues

- Very memory intensive!
 - For a graph of 100M nodes, and 400 floats per embedding vector, 4 bytes per float:
 - ~160GB! Just for the node embeddings.
 - Highest GPU memory: 80 GB available (NVIDIA A100)
 - Modest GPU: 12GB (NVIDIA TITAN V)
- Key challenge:
 - How do you facilitate data movement to and from GPU?
 - CPU RAM <--> GPU? (Amazon's DGL-KE, 2020)
 - DISK <--> GPU? (PyTorch BigGraph, 2019)
 - Distributed GPU Compute? (both ^)
 - How to overcome memory bottleneck and IO bottleneck?



NVIDIA TITAN V
NVIDIA's SUPERCOMPUTING GPU ARCHITECTURE, NOW FOR YOUR PC

NVIDIA TITAN V is the most powerful Volta-based graphics card ever created for the PC. NVIDIA's supercomputing GPU architecture is now here for your PC, and fueling breakthroughs in every industry.

\$2,999.00

OUT OF STOCK

WATCH FULL VIDEO

The image shows a vertical, gold and black NVIDIA TITAN V graphics card standing upright against a dark background. The card has a prominent circular fan at the bottom and a textured, armor-like top section.



**NVIDIA A100
TENSOR CORE GPU**

Unprecedented acceleration at every scale

The image shows a top-down view of the NVIDIA A100 Tensor Core GPU. It is a large, square chip mounted on a green printed circuit board (PCB). The chip itself is a dark, square die with a grid of gold pins around its perimeter. The PCB is populated with various components, including capacitors and other integrated circuits.

Solution!

- Pipelining for hiding IO overhead
- Disk <--> CPU RAM (used as cache) <--> GPU
 - Partition nodes
 - Load and store partitions with smart replacement schedule

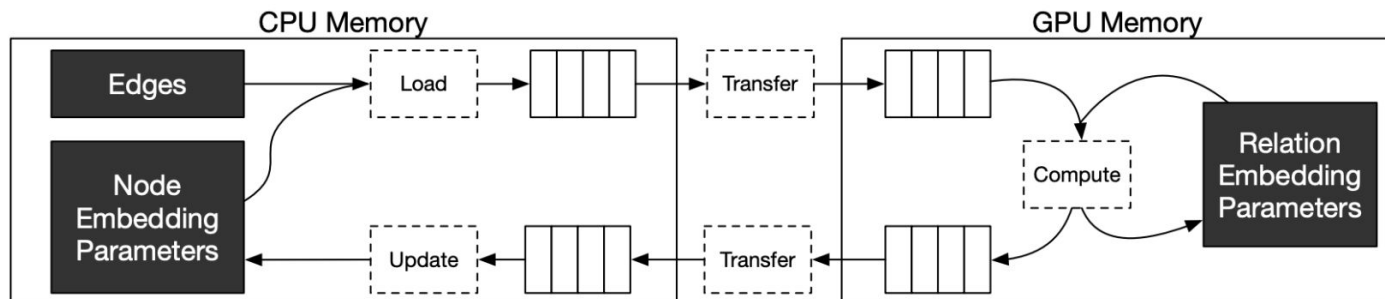
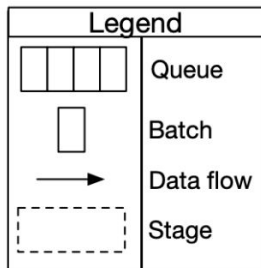
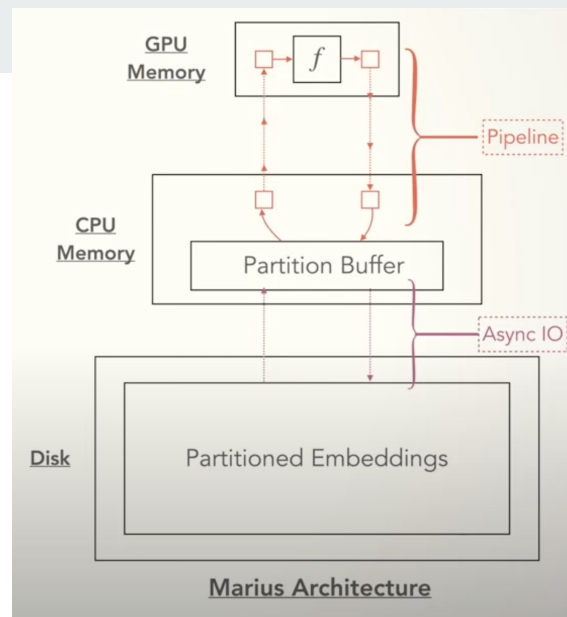


Figure 4: Marius training pipeline.

Partitions

- Nodes are partitioned into p partitions
- To calculate loss, need to cycle through p^2 pairs of partitions.

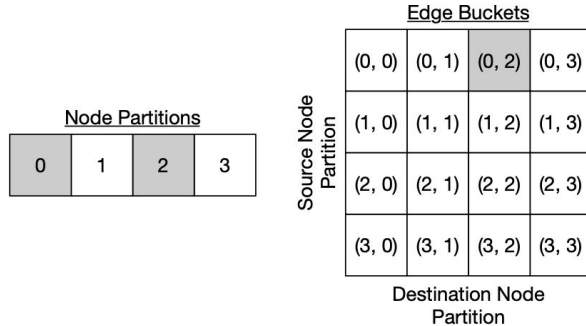


Figure 3: Partitions and edge-buckets with $p = 4$. All edges in edge-bucket (0, 2) have a source node in node-partition 0 and a destination node in node-partition 2.

Replacement Schedule Walkthrough

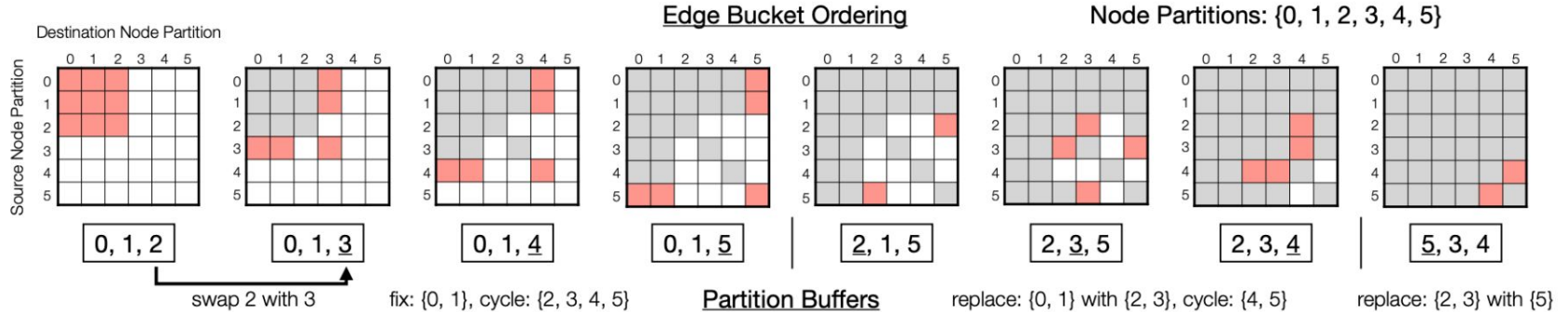


Figure 5: Example *BETA* ordering for $p = 6$ and $c = 3$. The sequence of partition buffers corresponds to first fixing $\{0, 1\}$, then replacing $\{0, 1\}$ with $\{2, 3\}$, fixing $\{2, 3\}$, and finally replacing $\{2, 3\}$ with $\{5\}$. Each successive buffer differs by one swap. A corresponding edge bucket ordering is shown above the buffers. For each partition buffer in the sequence, all previously unprocessed edge buckets which have their source and destination node partitions in the buffer are added to the ordering (red edge buckets). For each buffer, these edge buckets can be added in any order.



Results

See paper.



Discussion

- Exposing a general interface for implementing similar algorithms?
- Machine-aware self-configuring implementation?