INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2020 EDMONTON/CANADA
AUGUST 17-20, 2020

Sabancı Üniversitesi

GOSH

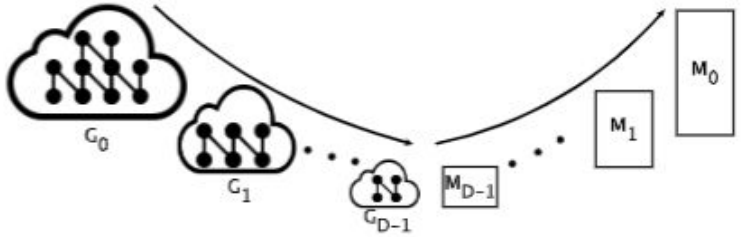
Embedding Big Graphs On Small Hardware

Taha Atahan Akyildiz
Amro Alabsi Aljundi
Kamer Kaya

FACULTY OF ENGINEERING AND NATURAL SCIENCES

GOSH: Embedding Big Graphs on Small Hardware

- **An ultra-fast** embedding on a ***single GPU*** for any **arbitrarily large graph**.
  - Provides link prediction AUC ROCs that **match the state-of-the-art** methods.
  - A parallel coarsening **with small overhead** that increases embedding performance
- A **flexible and dynamic task scheduling** for memory restricted devices.



Graph	GOSH	Speedup
Hyperlink (40m vertices, 600m edges)	**0.2 hours** (97% AUC) on a single TITAN X SotA with comparable AUC: **5.4 hours** on 4 Tesla P100	26.7x
Wiki-topcats (1.7m vertices, 28m edges)	**11 seconds** (98% AUC) on a single TITAN X SotA with comparable AUC: : **310 sec.** on a single TITAN X	27.4x

Introduction

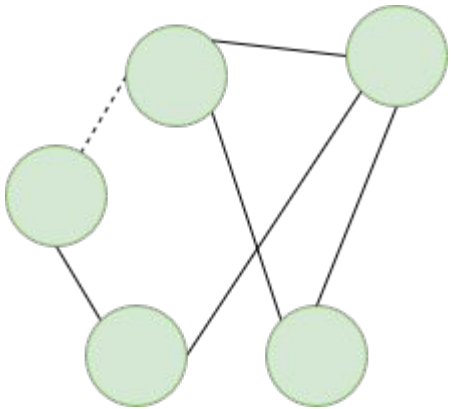
Graphs are ubiquitous

Graph types:

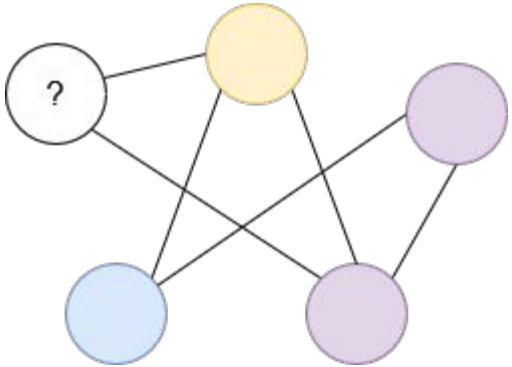
- Protein-protein interaction networks
- Citation networks
- Road networks
- Social networks

Graphs can have up to **tens millions of vertices and billions of edges.**

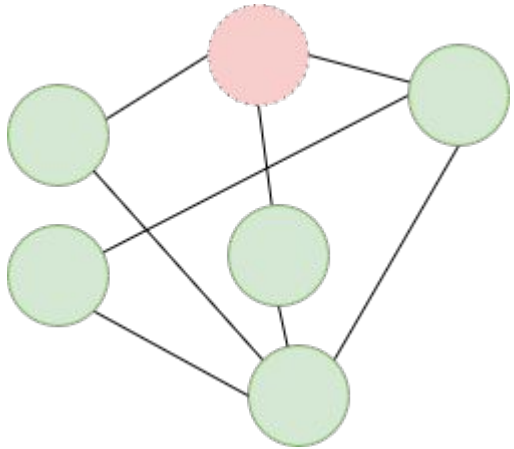
Machine learning on graphs



Link Prediction



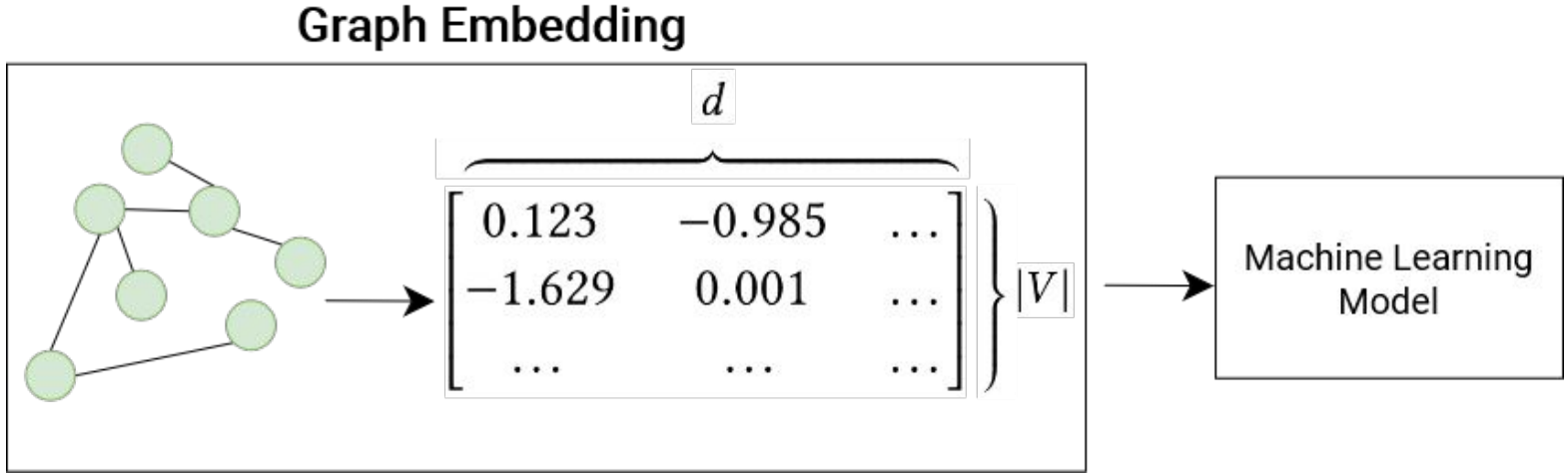
Node Classification



Anomaly Detection

Graph Embedding

- Connectivity information is **highly unstructured** and not suitable for machine learning tasks
- **Graph embedding** transforms the connectivity data of a graph into a standard d-dimensional representation that is more suitable for machine learning tasks



Graph Embedding is Expensive

Requires hours of CPU (and multi-CPU parallelization) time for graphs with 100,000's of vertices and larger graphs could require **days.** Solution?

Distributed systems - hardware requirements **scale with the graph size**

GPU parallelization - bigger graphs **require more GPUs**

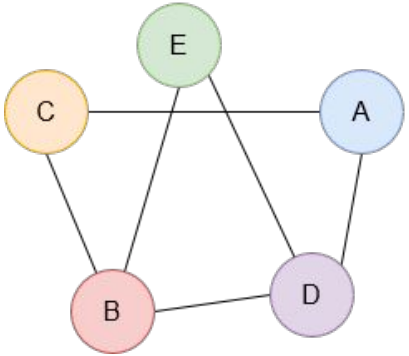
Graph coarsening (compression) - current approaches are **slow** and **inefficient**

GOSH: an **ultra-fast** embedding requiring a ***single GPU*** for any **arbitrarily large graph**.

Definitions

Given a graph $G = (V, E)$ consisting of the set of vertices V and the set of edges E

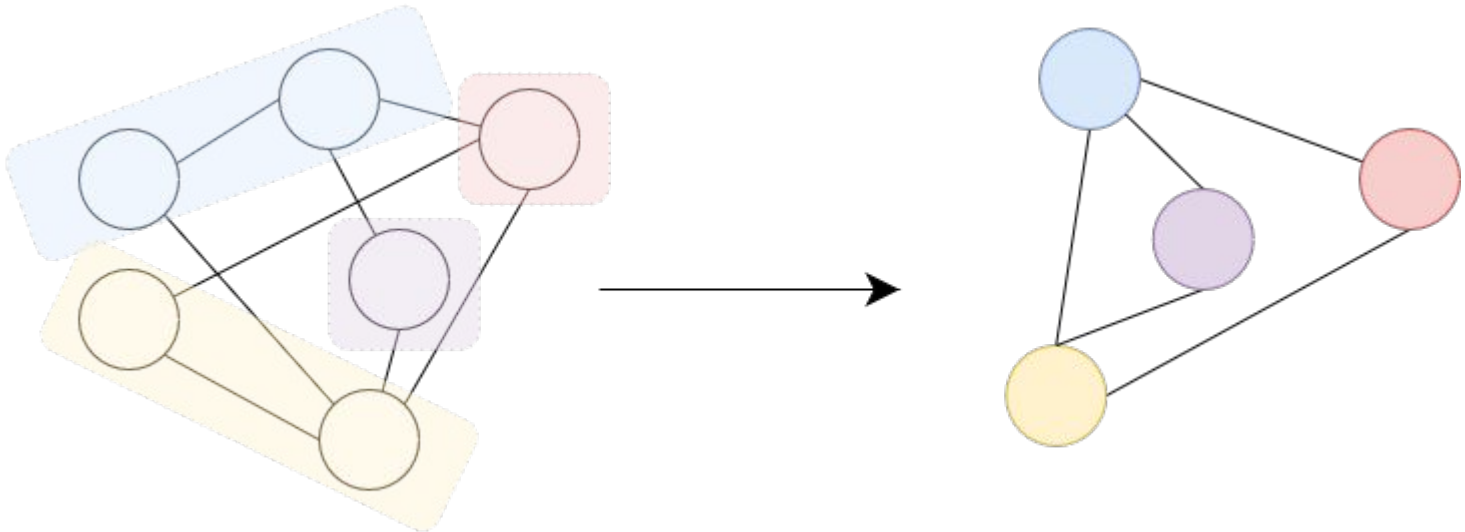
Graph embedding is the process of creating the embedding matrix $\mathbf{M}$ with $|V|$ rows and d columns (dimensions), where vector $\mathbf{M}[v]$ is the embedding of vertex v in the graph.



$$\mathbf{M} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \overbrace{\begin{bmatrix} 1.216 & 0.134 & -0.982 & \dots & 0.401 \\ -0.052 & -1.004 & -0.219 & \dots & 1.881 \\ 0.0457 & 0.151 & -0.986 & \dots & 0.526 \\ 0.619 & -0.147 & -0.775 & \dots & 0.598 \\ -0.869 & -0.790 & -0.928 & \dots & 0.237 \end{bmatrix}}^{d} \Big\} |V|$$

Definitions

Coarsening a graph G_i into the graph G_{i+1} will give every group of $k \geq 1$ vertices in G_i a super node in G_{i+1} and transfer the edge information from the sub to the super nodes.



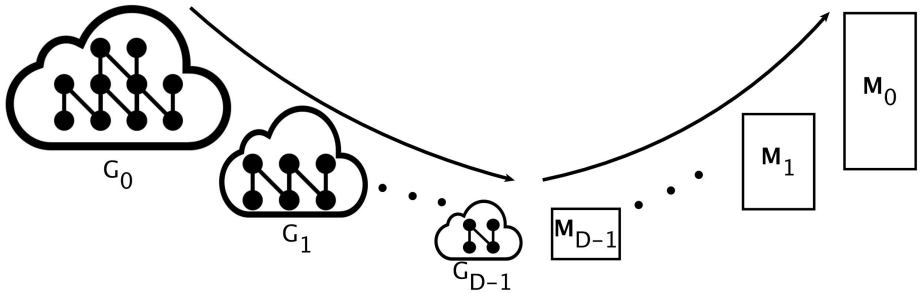
GOSH

General Flow

Algorithm starts by coarsening the input graph into smaller and smaller graphs until some ending criteria is met.

The smallest graph will be embedded, and its embeddings will be projected to the previous level by setting the embedding of every node in the finer graph to the embedding of its super node.

Process continues until the original graph has been embedded.



Embedding procedure

Embedding a graph employs the process of Noise Contrastive Estimation (NCE) through Stochastic Gradient Descent (SGD).

for $j = 0$ *to* $epochs$ **do**
  for $\forall src \in V$ **do**
    $u \leftarrow$ GetPositiveSample(src, G)
    UpdateEmbedding($\mathbf{M}[src]$, $\mathbf{M}[u]$, 1, lr)
    for $k = 1$ *to* n_s **do**
      $u \leftarrow$ GetNegativeSample(src,G)
      UpdateEmbedding($\mathbf{M}[src]$, $\mathbf{M}[u]$, 0, lr)

Embedding on a Single GPU is Memory Intensive

A social network with 100 million vertices and a standard embedding dimensionality of 128 would require around 51 GB of memory to store the embedding values **even in single precision**.

Way above the capacity of any single GPU

GOSH: a specialized embedding algorithm **for large graphs** which partitions a large graph into smaller subgraphs that can be processed by a **single GPU** to embed the whole graph in rotations.

Revised Flow

After coarsening the graph, and during the embedding stage, every graph is checked to see if its embedding matrix will fit the GPU.

- If it does, a single-kernel embedding job is dispatched to the GPU.
- If it doesn’t, the large graph embedding procedure is used.

At the end of the embedding, the embedding projection phase is carried out normally.

Work Distribution Across Graphs

When GOSH embeds a graph, it embeds a variable number of levels. How do the epochs (work done) get distributed across levels?

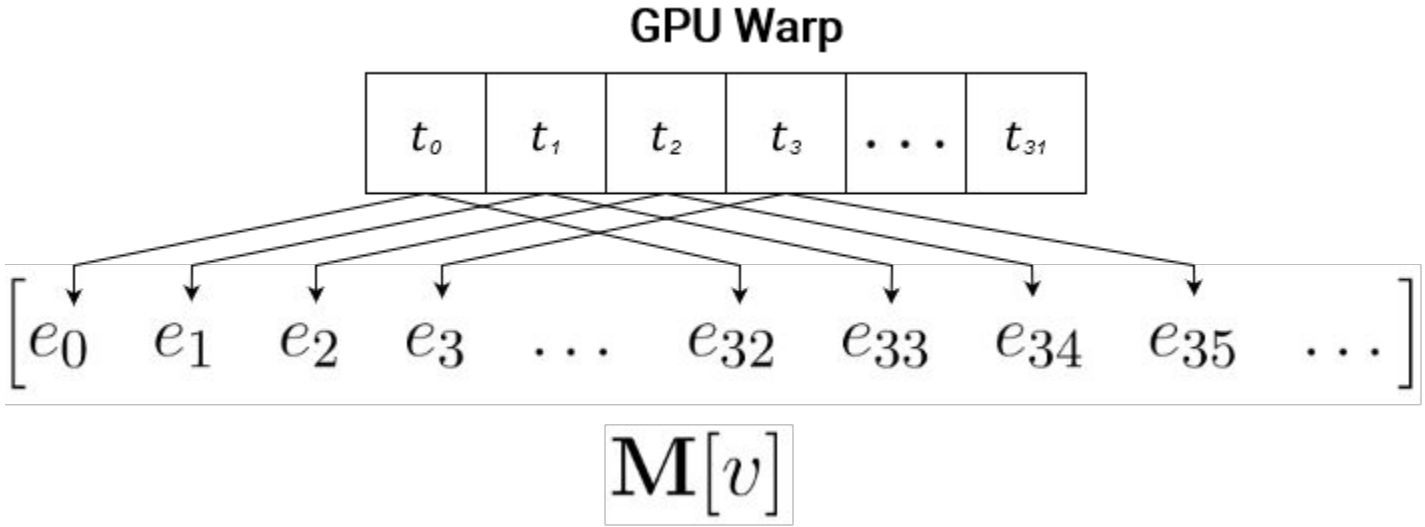
- Embedding the **coarser levels is faster** and accounts for **more updates,** but updates to the finer levels create more **finely tuned** embeddings

GOSH: a hybrid work distribution approach in which a portion *p* of the epochs is distributed evenly across the levels, and the remaining epochs are **distributed geometrically** such that **coarser levels get more epochs**.

GPU Embedding Step

To **maximize GPU utilization** and **minimize race conditions**, **GOSH** employs a **one update per GPU warp** approach

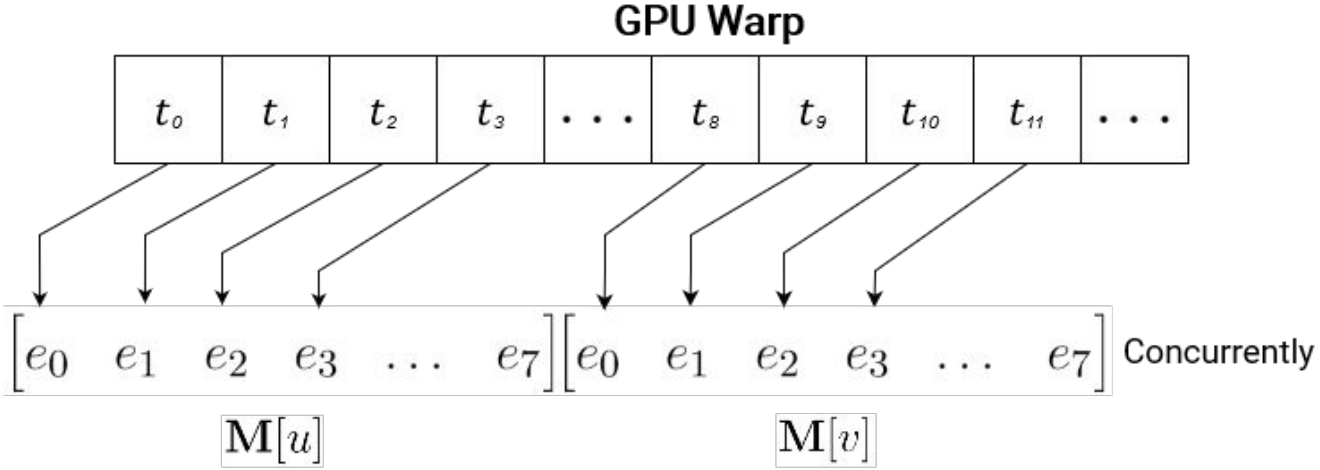
Thread t_i is responsible for elements e_i, e_{i+32},e_{i+64}… of an embedding vector



Smaller Dimensions

If an embedding has a dimensionality d <= 16, the previous method will result in (32-d) threads to be idle at any time

GOSH: A specialized method for such a case where groups of **multiples of 8 or 16 threads** are responsible for a single vertex.



Graph Coarsening

Graph Coarsening

- We aim to create a graph coarsening approach that will produce the best *coarsening effectiveness* and best *coarsening efficiency*

Corasening effectiveness is measured in terms of how well the resultant embeddings perform compare to other coarsenings

Coarsening efficiency at some level i measures the **rate of decrease in the number of vertices** from level i*-1* to level i, in other words

Coarsening efficiency of G_i = $$\frac{|V_i| - |V_{i+1}|}{|V_i|}$$

MultiEdgeCollapse Coarsening

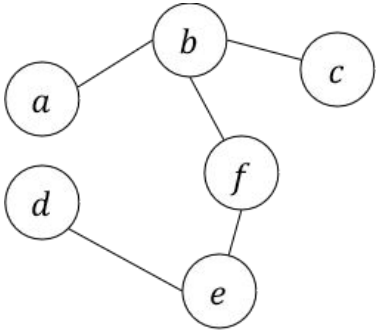
Sequentially process all vertices $v \in V$. If a vertex v is not yet “marked”, mark it to be a new super node v' and “match” any of its unmarked neighbors by marking them as v' as well.

Matching criteria: A vertex v cannot mark a vertex u, and vice versa, if both u and v have more edges than the average number of edges per vertex in the graph.

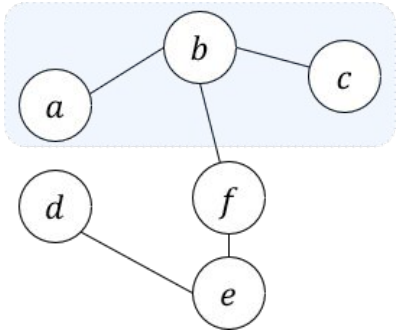
- Prevents two hub vertices from being coarsened into a single super vertex

Sorting vertices: Before processing the nodes, they are sorted in descending order based on the number of edges.

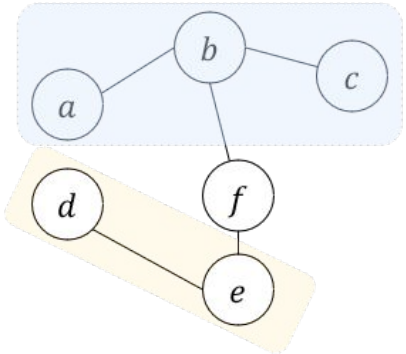
- Prevents hub vertices from being matched early on in the coarsening process.



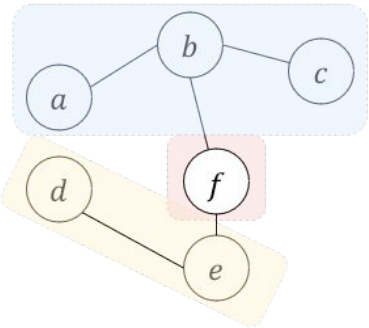
0. Initial, uncoarsened graph



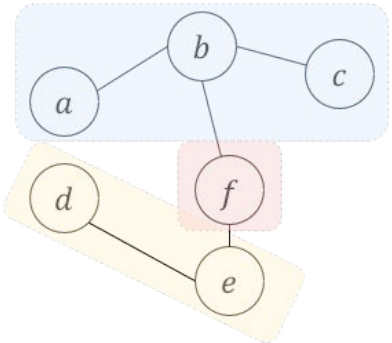
1. Mark b and match its neighbors c and a. Cannot match f due to matching rule.



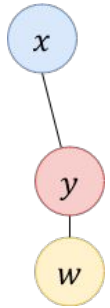
3. Mark e and match its neighbor d. Cannot match f due to matching rule.



4. Mark f. Since all its neighbor are marked, none of them are matched.



5. Try to mark a, c, and d, but they are all marked already.



6. Generate the coarsened graph and reconstruct edges

21

Parallelizing Coarsening

- MultiEdgeCollapse has a space and time complexity of O(|V|+|E|) and has the potential of parallelization

Matching phase: process vertices in *parallel* using T threads, but place locks on every vertex which the processing threads must acquire before marking or matching.

Reconstruction phase: when reconstructing the edges, every thread from the T threads will reconstruct the edges of a range of super vertices. Afterwards, the edge information generated by the threads are merged sequentially in O(T) time

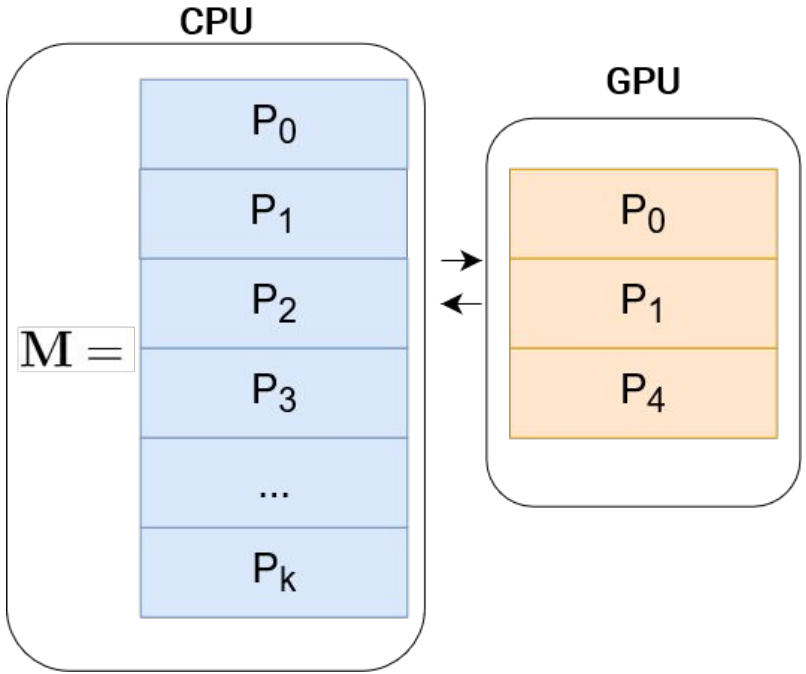
Large Graphs

Partitioning The Work

Partition the embeddings into smaller parts **such that 3 parts can fit inside the GPU concurrently.**

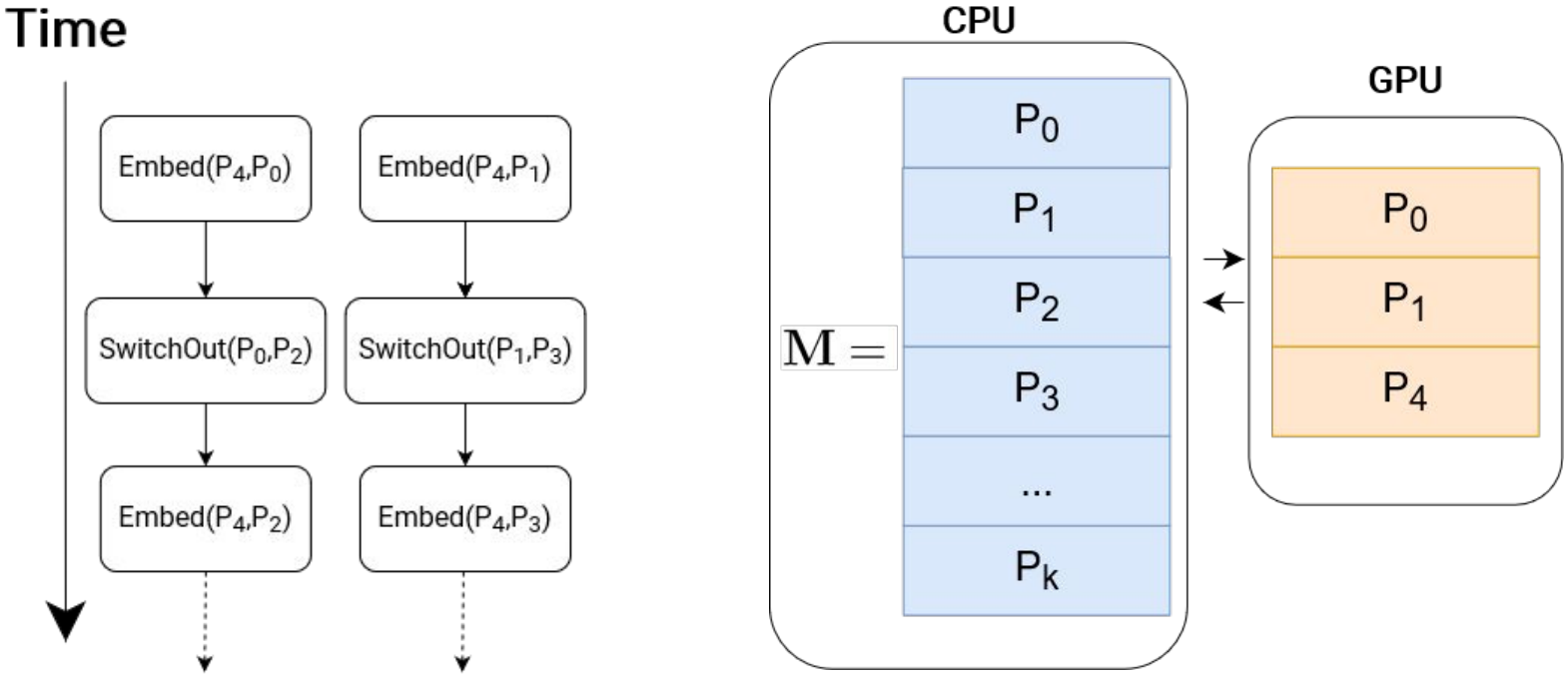
Execute batches of small jobs, each of which carries out **updates between the vertices in two parts**.

A single rotation of jobs **executes a job for every possible pair of parts.**



Partitioning The Work

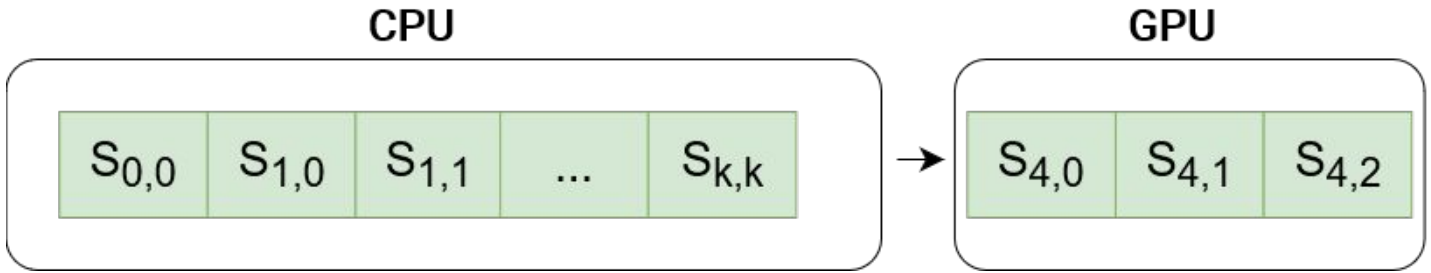
When a part is used up on the GPU, **it is switched out for another one from the CPU.**



Sample Space Partitioning

Storing the graph on the GPU is expensive, therefore **edges are sampled on the CPU concurrently with the embedding and sent to the GPU.**

The GPU will store **3 or more sample pools on the GPU** and the CPU will bring new sample pools depending on the running embedding.



Workflow

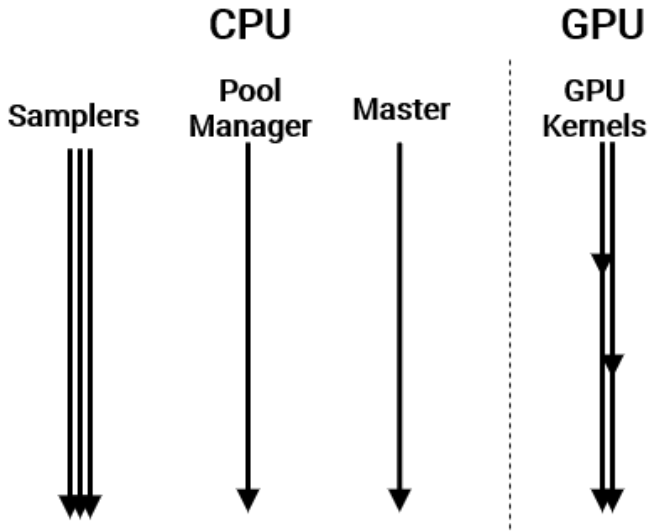
Four concurrent tasks:

1 - **Samplers:** Team of **CPU samplers** concurrently generating positive samples

2 - **Pool manager:** CPU thread that will dispatch **copies of sample pools** to the GPU

3 - **Master:** CPU thread which will copy embedding parts and dispatch embedding kernels

4 - **Kernels:** GPU jobs that execute the embedding jobs



Evaluation

Evaluation

Link prediction as the machine learning task to evaluate model and used **AUC ROC** as a metric

Evaluated systems:

- **GOSH-fast, -normal, -slow:** tweaked the smoothing factor, epochs and learning rate to demonstrate flexibility of the model
- **VERSE:** multi-core CPU embedding which **GOSH** is based on
- **Graphvite-fast, -slow:** state-of-the-art multi-GPU graph embedding running with two different epoch settings
- **MILE:** coarsening based graph embedding
- **GOSH-NoCoarse:** Direct embedding using **GOSH** without coarsening.

Datasets

Graph	|V|	|E|	Density
com-dblp [9]	317,080	1,049,866	3.31
com-amazon [9]	334,863	925,872	2.76
youtube [13]	1,138,499	4,945,382	4.34
soc-pokec [9]	1,632,803	30,622,564	18.75
wiki-topcats [9]	1,791,489	28,511,807	15.92
com-orkut [9]	3,072,441	117,185,083	38.14
com-lj [9]	3,997,962	34,681,189	8.67
soc-LiveJournal [9]	4,847,571	68,993,773	14.23
hyperlink2012 [12]	39,497,204	623,056,313	15.77
soc-sinaweibo [18]	58,655,849	261,321,071	4.46
twitter_rv [18]	41,652,230	1,468,365,182	35.25
com-friendster [9]	65,608,366	1,806,067,135	27.53

Use the embeddings to train a **Logistic Regression** model

80%-20% train-test split

Rows in the training and test sets are **edges (50% positive and 50% negative)**

The row of edge (u, v) is constructed by **element-wise multiplying the embeddings of u and v**

Evaluating Coarsening Efficiency

	i	Time (s)	$\lvert V_i \rvert$
MILE	0	-	3056838
	1	249.77	1535168
	2	237.39	768804
	3	184.72	384752
	4	151.24	192507
	5	139.23	96308
	6	128.47	48183
	7	117.75	24107
	8	99.73	12062
	Total	1308.31	-

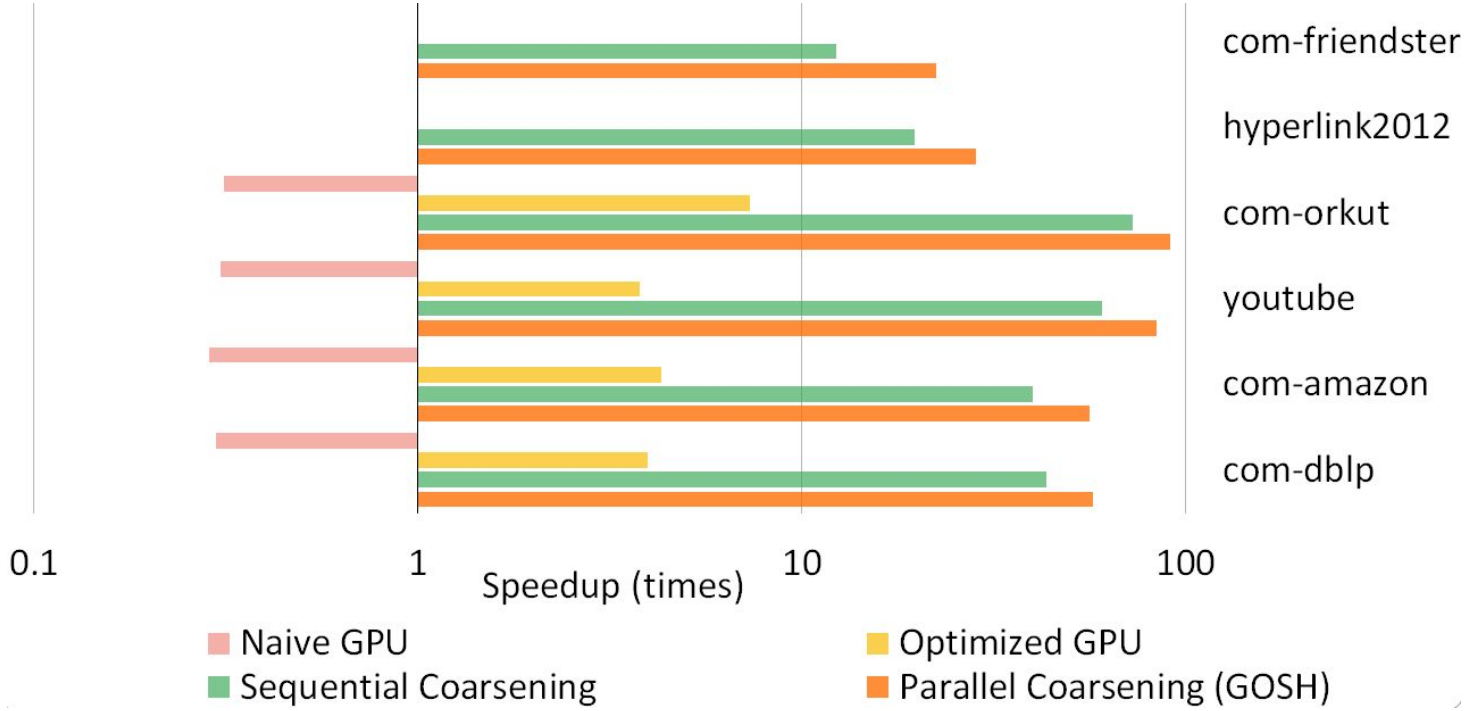
	i	Time (s)	$\lvert V_i \rvert$
GOSH	0	-	3056838
	1	4.44	975132
	2	1.23	213707
	3	0.62	46667
	4	0.16	8084
	5	0.03	2000
	6	0.01	701
	7	< 0.01	375
	8	< 0.01	275
	Total	6.60	-

Orders of magnitude faster

Better coarsening throughout levels

Better prediction accuracy (shown in a later slide)

Speedup Breakdown



Accuracy Comparison - Medium Scale Graphs

Graph	Algorithm	Time (s)	Speedup	AUCROC(%)
com-lj	VERSE	12502.72	1.00×	**98.86**
	MILE	3948.62	3.17×	80.19
	GRAPHVITE-fast	373.58	33.47×	98.04
	GRAPHVITE-slow	644.43	19.40×	98.33
	GOSH-fast	16.27	768.45×	96.82
	GOSH-normal	55.01	227.28×	98.33
	GOSH-slow	153.72	81.33×	**98.46**
	GOSH-NoCoarse	675.25	18.52×	98.32
youtube	VERSE	1365.36	1.00×	**98.04**
	MILE	1328.62	1.03×	94.17
	GRAPHVITE-fast	63.90	21.37×	97.07
	GRAPHVITE-slow	104.76	13.03×	97.45
	GOSH-fast	2.76	494.70×	96.16
	GOSH-normal	7.15	190.96×	97.78
	GOSH-slow	15.32	89.12×	**97.93**
	GOSH-NoCoarse	158.60	8.61×	97.16

Maintain accuracy while being faster than other models

Flexibility between speed and accuracy

Accuracy Comparison - Large Scale Graphs

Graph	Algorithm	Time (s)	Speedup	ROC (%)
soc-sinaweibo	VERSE	20397.79	1.00×	99.89
	GOSH-fast	48.88	417.30×	70.27
	GOSH-normal	352.86	57.81×	97.00
	GOSH-slow	759.85	26.84×	99.37
twitter_rv	VERSE	Timeout	-	-
	GOSH-fast	261.08	-	91.78
	GOSH-normal	994.46	-	97.36
	GOSH-slow	2128.70	-	98.50

Graphvite cannot run large scale graphs using a single GPU

MILE and VERSE time out after 12 hours of runtime

Small Dimensions results

Graph	SM	d	Time (s)	Graph	SM	d	Time (s)
com-orkut	No	8	63.72	soc-LiveJournal	No	8	40.13
		16	64.20			16	40.46
		32	64.95			32	41.22
	Yes	8	24.27		Yes	8	14.86
		16	34.98			16	21.82
		32	64.54			32	40.93

When d is smaller than 32, assigning a single update to a warp is a waste of resources.

We assign 32/d updates to a single warp to fully utilize all the GPU cores, where d = 16, 8.

Conclusion

Conclusion

GOSH provides **high quality embeddings** using **minimal hardware** and **at a fraction of the time**

Employs a **novel parallel graph coarsening** algorithm and a special **scheduling schema** to embed **large graphs** using a **single GPU**

Provides **flexibility** between **quality and speed** of embedding