Accelerating Graph Sampling for Graph Machine Learning using GPUs

Abhinav Jangda  
*University of Massachusetts Amherst*

Sandeep Polisetty  
*University of Massachusetts Amherst*

Arjun Guha  
*Northeastern University*

Marco Serafini  
*University of Massachusetts Amherst*
Graph Data is Everywhere

Social Networks

Relational Data

Recommendation Systems

Fraud and Financial Data

Knowledge Bases
Machine Learning on Graphs

Graph Machine Learning uses the network structure of the underlying data to improve predictive outcomes.

Personalization and Recommendation Systems

Search, Q&A, semantic web

*Induction from text*  
*Deduction from knowledge graph*

Combine features from multiple users and products
Graph Neural Network maps vertices of graphs to an embedding in a N-dimensional space.

**Encoding Phase**

**Decoding Phase**

- Vertex Embeddings
- Decoder
- Product Recommendation
- Clustering

2-D Vertex Embeddings
Types of Graph Neural Networks (GNNs)

Two types of Graph Neural Networks:
- Sampling based GNNs samples the input graph and train using these samples
- Whole Graph based GNNs train on the input graph directly
A Sampling based GNN first samples the graph using a Graph Sampling Algorithm and use these samples for data parallel training.
Graph Sampling is an Active Area of Research

DeepWalk and node2vec are random walks of fixed length starting from a vertex.

GraphSAGE samples 2-Hop Neighborhood of a vertex

FastGCN and LADIES samples vertices for each layer in their Neural Network.

ClusterGCN divides a graph into many clusters and samples one or more of these clusters.
ML Domain Experts Implement Graph Sampling on CPU

Graph Sampling on CPU  Neural Network on GPU

GNN implementations use CPU for Graph Sampling and GPU for Neural Network
Graph Sampling is a Major Overhead in GNNs

Experiments Performed on 32-Core CPU with 1 NVIDIA Tesla V100

How can domain experts have best of both worlds: easy to implement and fast Graph Sampling?
NextDoor: A System to Accelerate Graph Sampling on GPUs

1. A simple yet powerful API to express diverse graph sampling algorithms.
3. Load balancing and caching to optimize GPU utilization.
4. Improves end-to-end training time of GNNs by up to 4x.

Write Graph Sampling using NextDoor’s API

```c
Vertex next(s, trn, trnEdges) {
    return trnEdges[randInt(0, 10)];
}

int steps() {
    return 3;
}

int sampleSize(int step) {
    return 1;
}
```

NextDoor’s Runtime

Optimized GPU Kernels

Input Graph

Samples of Input Graph
An Abstraction for Graph Sampling Applications

- A graph sampling application runs for $k$ steps.
- In the beginning each sample has one or more root vertices.
- At each step $i$
  - A transit vertex for $i$ is a vertex whose neighbors may be added to the sample
  - Sample $m_i$ of those neighbors

![Diagram of graph sampling](image)

- $step = 0$
  - $m_0 = 2$
  - Transit Vertex at step 0

- $step = 1$
  - $m_1 = 2$
  - Transit Vertices at step 1

2-Hop Neighborhood of 2

- NextDoor’s API
- Sample Parallel Sampling
- Transit Parallel Sampling
- NextDoor’s Runtime
- Experiments
Graph Sampling using NextDoor’s API

- Simple yet powerful API based on the abstraction.
- Implementations need only few lines of code
- We implemented diverse algorithms
  - DeepWalk, PPR, and node2vec
  - K-hop Neighborhood Sampling
  - Importance Sampling
  - ClusterGCN Sampling
  - Minimal Variance Sampling
  - Layer Sampling

```cpp
Vertex next(s, trn, trnEdges) {
    int idx = randInt(0, 
                     trnEdges.size()-1);
    return trnEdges[idx];
}
```

```cpp
int steps() {return 2;}
int sampleSize(int step)
{return (step == 0) ? 25:10;}
```

GraphSAGE’s k-hop Neighborhood in NextDoor

The API provides information to perform effective load balancing and caching:
- It provides a distinction between a sample and a transit vertex
- Provides the number of steps and number of vertices sampled at each step.

How can we implement this API on a GPU?
A GPU Can Execute Thousands of Threads Simultaneously

High Latency (200 - 800 cycles) Global Memory (2 - 32 GB)

Random accesses take more cycles than consecutive accesses

Low Latency (10 cycles)
Shared Memory (< 48 KB)
Can be utilized as a software managed cache.

GPU Threads

Simultaneous Multiprocessor (SM)

Thread Block

Thread Block

Simultaneous Multiprocessor (SM)

Simultaneous Multiprocessor (SM)
Sample Parallel Graph Sampling on GPUs

**Sample Parallel:** Assign samples to consecutive threads.

![Diagram showing sample parallel graph sampling on GPUs]

- **Samples:** \( S_1, S_2, S_3 \)
- **Transits:** \( T_1, T_2, T_3 \)

**Step 1:**
- 2-Hop Neighborhood of \( S_1, S_2, S_3 \)

Sample Parallelism suffers from irregularity:
- Leads to random memory accesses
- Cannot cache in shared memory and registers

**GPU Sampling Kernel**

- Edge List in Global Memory
- Consecutive threads access edges of different transit vertices

**Experiments**

- NextDoor's API
- Sample Parallel Sampling
- Transit Parallel Sampling
- NextDoor's Runtime
Transit Parallel: Rethinking Parallel Graph Sampling

Transit Parallel: Assign samples with common transits to consecutive threads.

Transit Parallelism achieves regularity:
✓ Helps to coalesce memory accesses
✓ Can cache in shared memory and registers

Group Samples by Transit

Transits

GPU Sampling Kernel

Edge List in Global Memory

Consecutive threads access edges of same transit vertices

NextDoor's API
Sample Parallel Sampling
Transit Parallel Sampling
NextDoor's Runtime
Experiments

Experiments
Load Balanced Transit Parallelism in NextDoor

NextDoor performs load balancing for different transits and cache of neighbors of a transit.
Implementation

1. NextDoor utilizes efficient parallel radix sort and prefix scan for group by operation and load balancing.
2. NextDoor is implemented in C++14 and CUDA 11.
Experiments

Benchmarks:
- DeepWalk
- Personalized Page Rank (PPR)
- node2vec
- Multi Dimensional Random Walks
- K-hop Neighborhood Sampling
- ClusterGCN Sampling
- FastGCN Sampling
- LADIES Sampling
- Minimal Variance Sampling (MVS)
- Layer Sampling

CPU only Baselines:
- KnightKing
- Samplers in existing GNN Systems

<table>
<thead>
<tr>
<th>Graph Name</th>
<th>Abrv</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protein-Protein Interactions</td>
<td>PPI</td>
<td>50K</td>
<td>1.4M</td>
</tr>
<tr>
<td>com-Orket</td>
<td>Orkut</td>
<td>3M</td>
<td>117M</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>Patents</td>
<td>3.77M</td>
<td>16.5M</td>
</tr>
<tr>
<td>soc-LiveJournal1</td>
<td>LiveJ</td>
<td>4.8M</td>
<td>68.9M</td>
</tr>
</tbody>
</table>
Experiments

Evaluation System:
- Dual Socket 16-Core Intel Xeon Silver CPUs
- 4x NVIDIA Tesla V100
- 128 GB of RAM
- Ubuntu 18.04
- CUDA 11.2
NextDoor: End-to-End Speedups with GNN Training

NextDoor’s fast Graph Sampling implementations significantly improves training time of GNNs.
NextDoor against existing Graph Sampling Implementations

NextDoor’s API
- Sample Parallel Sampling
- Transit Parallel Sampling
- NextDoor’s Runtime

Experiments

NextDoor achieves orders of magnitude speedup over CPU baselines
Summary

- Graph Sampling takes up to 62% of training time in Graph Neural Networks.
- Efficient Graph Sampling on GPUs is hard due to irregular nature of graphs.
- NextDoor: accelerate Graph Sampling using GPUs.
  - ✓ API to easily write efficient graph sampling applications.
  - ✓ Runtime that optimizes memory accesses in GPUs and efficiently balances load.
- NextDoor achieves orders of magnitude improvement over existing solutions.
- NextDoor improves end-to-end training time of GNNs by up to 4 times on large graphs.
Existing Implementations Have Limited Parallelism

Is it possible to increase parallelism in the standard approach to parallel Graph Sampling?

One thread of a sample is assigned to expand different vertices of the sample.

2-Hop Neighborhood of 1, 2, & 3

Is it possible to increase parallelism in the standard approach to parallel Graph Sampling?
Regular Code achieves High Performance on GPU

**Regular Computation Thread Block**
- Consecutive Loads ✓
- Non-Divergent Control Flow ✓
- Cache in Shared Memory ✓

**Irregular Computation Thread Block**
- Random Memory Access ❌
- Divergent Control Flow ❌
- Cannot Cache in Shared Memory ❌

Global Memory

Shared Memory

Consecutive Loads ✓
Non-Divergent Control Flow ✓
Cache in Shared Memory ✓

Irregular Computation Thread Block

Different neighbors of vertices

200 cycles

800 cycles

Cannot cache random accesses
Wait due to divergent control flow

NextDoor's API
Sample Parallel Sampling
Transit Parallel Sampling
NextDoor's Runtime
Experiments
Sample Parallel Graph Sampling is Irregular

- Sample Parallelism suffers from irregularity:
  - Leads to random memory accesses
  - Cannot cache in shared memory and registers

---

2-Hop Neighborhood of 1, 2 & 3

- Consecutive threads access edges of different transit vertices

- NextDoor's API
- Sample Parallel Sampling
- Transit Parallel Sampling
- NextDoor's Runtime
- Experiments
Scheduling Transit Parallel in NextDoor

The NextDoor API exposes *three degrees of parallelism* that match the GPU architecture:

1. **1st degree**: Each *transit* is mapped to a *threadblock*.
2. **2nd degree**: Each *sample* is assigned to a group of $m_i$ threads at step $i$.
3. **3rd degree**: Each *thread* samples one *neighbor*.

---

**GPU Sampling Kernel for 2nd step of 2-Hop Neighborhood with $m_1 = 2$**

- Edge List in Global Memory
- 1st degree: Each *transit* is mapped to a *threadblock*.
- 2nd degree: Each *sample* is assigned to a group of $m_i$ threads at step $i$.
- 3rd degree: Each *thread* samples one *neighbor*.

- NextDoor's API
- Sample Parallel Sampling
- Transit Parallel Sampling
- NextDoor's Runtime
- Experiments
Additional Overhead of Transit Parallel over Sample Parallel

- Transit Parallel uses a Group By operation.
- Random Walks spend up to 40% time in grouping operation.
- Despite overhead Transit Parallel still achieves up to 2x speedup over Sample Parallel in Random Walks.
- Other Application spend less than 10% time in grouping operation.
- Random Walks spends more time because they sample only one neighbor of the transit vertex at each step.

Percentage of time spent in *group by* operation over total time.
Standard Approach to Parallel Graph Sampling

Sample Parallel Graph Sampling

- Samples can be expanded in parallel by assigning samples to a single thread.
- Approach adopted by existing systems.

Can we use this approach for a GPU based parallel graph sampling?
Leverage GPUs for Graph Sampling is hard!

Regular Computations

\[
\begin{bmatrix}
  a_{1} & a_{2} & a_{3} \\
  a_{4} & a_{5} & a_{6} \\
  a_{7} & a_{8} & a_{9}
\end{bmatrix}
+ 
\begin{bmatrix}
  b_{1} & b_{2} & b_{3} \\
  b_{4} & b_{5} & b_{6} \\
  b_{7} & b_{8} & b_{9}
\end{bmatrix} = 
\begin{bmatrix}
  c_{1} & c_{2} & c_{3} \\
  c_{4} & c_{5} & c_{6} \\
  c_{7} & c_{8} & c_{9}
\end{bmatrix}
\]

Irregular Computations

Different neighbors of vertices

✓ Consecutive Memory Accesses
✓ Convergent Control Flow
✓ Utilize faster shared memory

❌ Random Memory Access
❌ Divergent Control Flow
❌ Cannot Utilize faster memory
NextDoor speedup over Transit Parallel

![Graph showing speedup for different datasets and models](image-url)
Workflow of Graph Neural Network Training

Input Graph

Random Walk starting at 1

Random Walk starting at 2

Random Walk starting at 3

Each Random Walk is a mini-batch

Neural Network
An Abstraction for Graph Sampling Applications

- A graph sampling application runs for \( k \) steps.
- Each execution of application produces one sample of the graph.
- In the beginning each sample has root vertex(s).
- At step \( i \), the application samples \( m_i \) vertices.
- Function \( \text{next} \) describes the sampling procedure.
- A transit vertex at a step \( i \) is a vertex whose neighbors may be sampled at step \( i \).

A Random Walk of length 4 starting from 1