GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs

Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, Yufei Ding.

UC Santa Barbara
Graphs are everywhere...

Social Networks

Financial Services

Molecular chemistry

Point Cloud

Power Grid

Molecular Biology

Credict: Google Image
Graph Analytics: Goals and Methods

- **Extract more insights from graphs structure.**
  - Generate the feature vectors (embeddings) for nodes, edges, and graphs.
  - **Link prediction:** friend recommendation.
  - **Graph prediction:** drug classification.
  - **Node classification:** power-grid failure detection.

- **GNN Vs. Traditional graph algorithms (e.g., random walks).**
  - High classification accuracy.
  - Better generality for diverse graph inputs.
  - Lower computation complexity.
  - Ease of parallelization.
GNN: Graph Neural Networks
GNN: Graph Neural Networks

Kipf, Thomas N., Max Welling. Semi-supervised classification with graph convolutional networks. ICLR’17

Hamilton et al. Inductive Representation Learning on Large Graphs. NeurIPS’17
Existing GNN Acceleration Solutions

- **Graph Processing Framework [Gunrock]:**
  - Optimizations tailored for graph algorithms.
  - Missing operators for NN computation.
  - Lack of programmability and portability.

- **Deep Learning Frameworks [PyG, DGL]:**
  - Focusing on programmability and generality.
  - Lack of efficient backend for sparse operators.
  - Hard-coded designs with poor input adaptability.
Overview of GNNAdvisor

Overall, we are the first to

- Explore the benefits of input properties (e.g., GNN model architectures and input graphs).
- Give an in-depth analysis of their importance in guiding system optimizations for GPU-based GNN computing.
Graph Information.

1. Node Degree.
Real-world graphs follow the power-law distribution of node degrees.

2. Embedding Dimensionality.
GNN input graphs demonstrate various node embedding sizes.

3. Graph community
Skewed edge distribution widely exists in many real-world graphs.

(a) Graph Community  
(b) Loading without Community  
(c) Loading with Community
Input Extraction (cont’d)

- **GNN model information.**
  - The order of neighbor aggregation and node update.
  - The types of aggregation method, such as sum, mean.

---

| GCN Layer | GIN Layer |
|-----------|-----------|
| Node Update | Node Update |
| Neighbor Aggregation | Neighbor Aggregation |
2D Workload Management

- **Coarse-grained Neighbor Partitioning**
  It is a novel workload balance technique tailored for GNN computing on GPUs. It aims to tackle the challenge of inter-node workloads imbalance and redundant atomic operations.

- **Fine-grained Dimension Partitioning**
  It further distributes the workloads of a neighbor group along the embedding dimension to improve the aggregation performance.
2D Workload Management (cont’d)

- **Warp-aligned Thread Mapping:**
  This is in collaborating with our neighbor and dimension partitioning to systematically capitalize on the performance benefits of balanced workloads.
Specialized Memory Optimization

Community-aware Node Renumbering:
We reorder node IDs to improve the temporal/spatial locality at the GNN aggregation without changing the output correctness to explore the performance benefits of graph community.
Warp-centric Shared Memory Optimization:
We customize GPU shared memory layout according to the block-level warp organization pattern, therefore, significantly reducing the number of atomic operations and global memory access.
Design Optimization

\[ WPT = ngs \times \frac{Dim}{dw} \]
\[ SMEM = \frac{tpb}{tpw} \times Dim \times FloatS \]

- **Analytical Modeling:**
The performance/resource analytical model of GNNAvisor has two variables, workload per thread (WPT), and shared memory usage per block (SMEM).

- **Parameter Auto Selection:**
To determine the value of the neighbor-group size (ngs) and dimension-worker (dw), we follow two steps.
- First, we determine the value of \( dw \) based on \( tpw \) (thread-per-warp) and \( \text{dim} \) (embedding dimension).
- Second, we determine the value of \( ngs \) based on the selected \( dw \) and the thread-per-block (tpb).
GNN Models.

- **Graph Convolutional Network (GCN):**
  - 2 layers with 16 hidden dimensions.

- **Graph Isomorphism Network (GIN):**
  - 5 layers with 64 hidden dimensions.

Evaluation Platform.

A server with an 8-core 16-thread Intel Xeon Silver 4110 CPU and a Quadro P6000 GPU. Also study on the DGX-1 system with Tesla V100 GPU.
Evaluation (cont’d): Overall Performance

Averaged 4.03x and 2.02x speedup in comparison with DGL on GCN and GIN in inference.

Averaged 1.61x and 2.00x speedup in comparison with DGL on GCN and GIN in training.
Evaluation (cont’d): Optimization Analysis

(c) Node Renumbering.

up to 1.74x and 1.49x speedup in GCN and GIN, respectively.

(d) Block-level Optimizations.

average 47.85% and 57.93% reduction in atomic operations and DRAM access, respectively.
Evaluation (cont’d): Additional Studies

(a) Dimension Analysis on GCN.

(b) Dimension Worker.

(c) Performance Quadro P6000 Vs. Tesla V100.
Key Focus & Contributions

☑ Efficient sparse kernel design for GNN computation on GPUs

☑ Design flexibility for handling different inputs.

☑ Seamless integration with the existing NN frameworks.

- 2D workload management.
- Specialized memory optimization.

GNN Input properties (e.g., graph structure, node embedding size) for guiding system-level optimizations.

PyTorch-based front-end design with high programmability and portability.
Thank You

Q & A

[GitHub] https://github.com/YukeWang96/OSDI21_AE.git