SparGNN: Efficient Joint Feature-Model Sparsity Exploitation in Graph Neural Network Acceleration

Chen Yin, Jianfei Jiang, Qin Wang, Zhigang Mao and Naifeng Jing

Department of Micro-Nano Electronics, Shanghai Jiao Tong University, Shanghai, China

2024.01.23

Outline

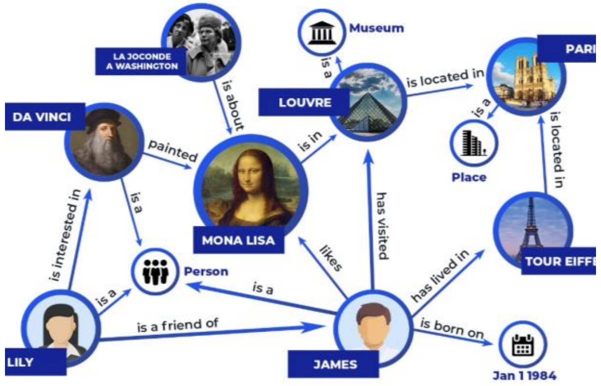
- **Background & Motivation**
- Algorithm Optimizations
- Dataflow & Hardware Design
- Evaluations

Application Scenario of Graphs

Graphs are widely used in many real-world applications

Social Network

Trading Network



Knowledge Database

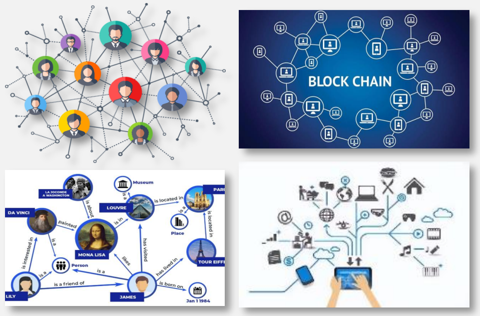
IoT Network

Millions of vertices and billions of edges in real-world graphs

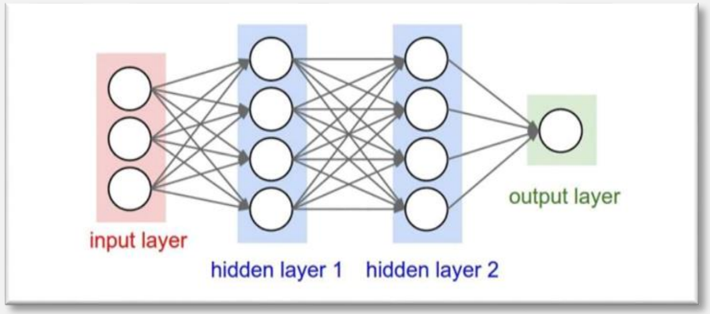
Image source: Google Images

Graph Neural Network (GNN)

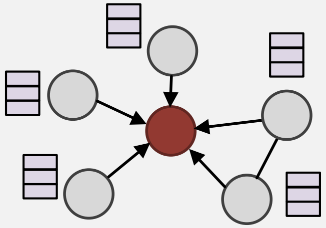
Graph Neural Network (GNN) is a powerful tool to process **Graph Data** by leveraging **Neural Network**



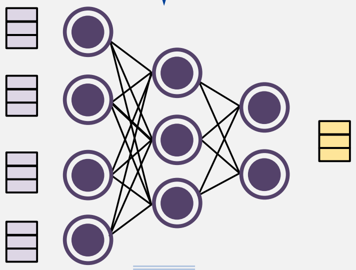
+



=



Neighbor features



Graph adjacency (A)

Vertex feature (X)

Model weight (W)

Challenges:

- Large computation & storage requirement
- E.g., (A, X, W) in *Reddit* are **over 58 GB**

Exploit Data Sparsity

Challenges in GNN Sparsity Exploitation

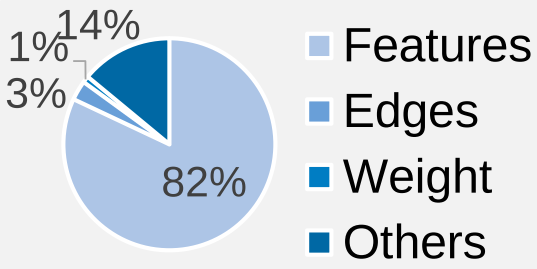
Existing works overlook the potential sparsity in features and models

Data size & density

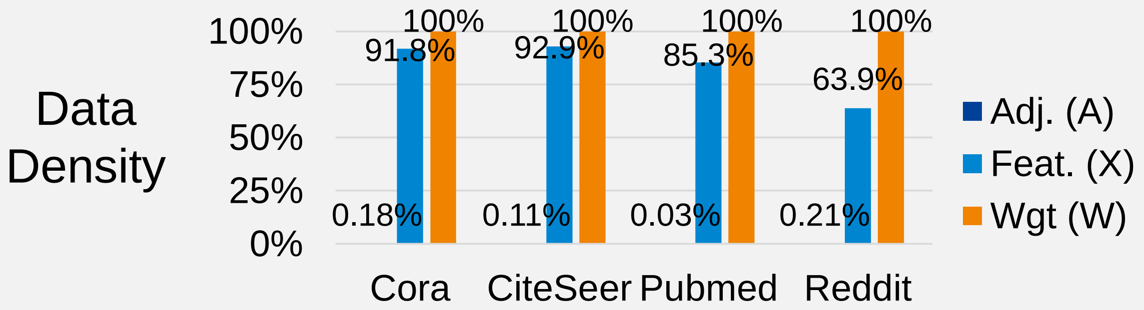
Graph adj. (A):
- Large but rather sparse

Feature (X):
- Large & dense

Model weight (W):
- Rather small but dense



Data percentage in *GraphSAGE @ Obgn-product*



Existing works

Optimize for graph sparsity
- Dataflow optimization[1]:

$$(A\times X)\times W \Rightarrow A\times(X\times W)$$

Limited to specialized GNN models
- Aim to deep layer models with residual component[2]

Overlook to exploit feature sparsity
- Graph and model co-pruning[3]

[1] Tong Geng, et al. MICRO 2020
[2] Mingi Yoo, et al. HPCA 2023
[3] Tianlong Chen, et al. ICML 2021

Challenges in GNN Sparsity Exploitation

Pruning overhead in GNNs will counteract the benefit of sparsification

High pruning overhead

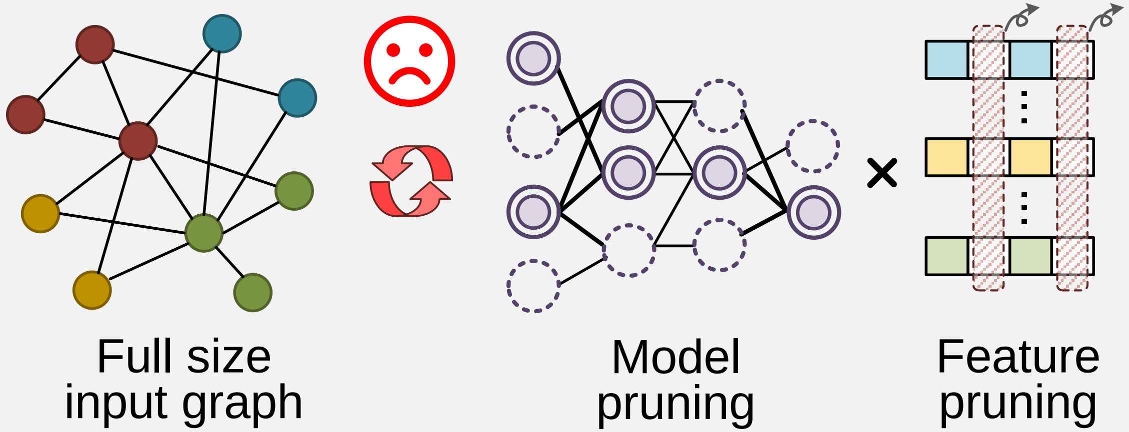
Massive operations during pruning:

- Large input graph results in massive irregular computing & memory access

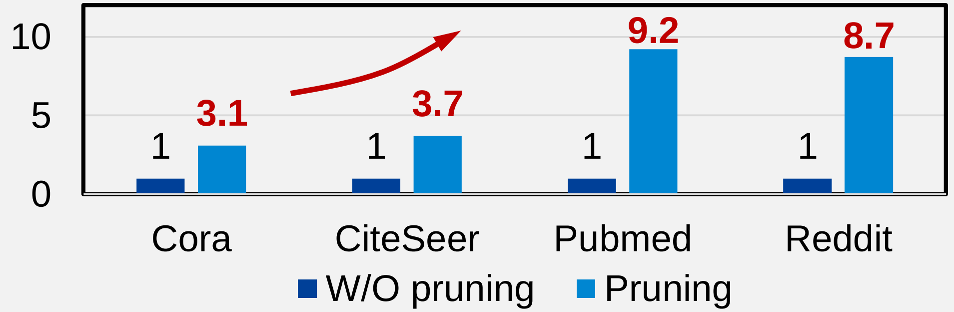
Pruning overhead cannot be amortized:

- Model parameters are related to input graphs, need re-pruning for a new graph

Pruning upon full-size input graph:
Massive irregular computation & access



Increased training time with pruning



GNN pruning will increase the training time up to 9.2×

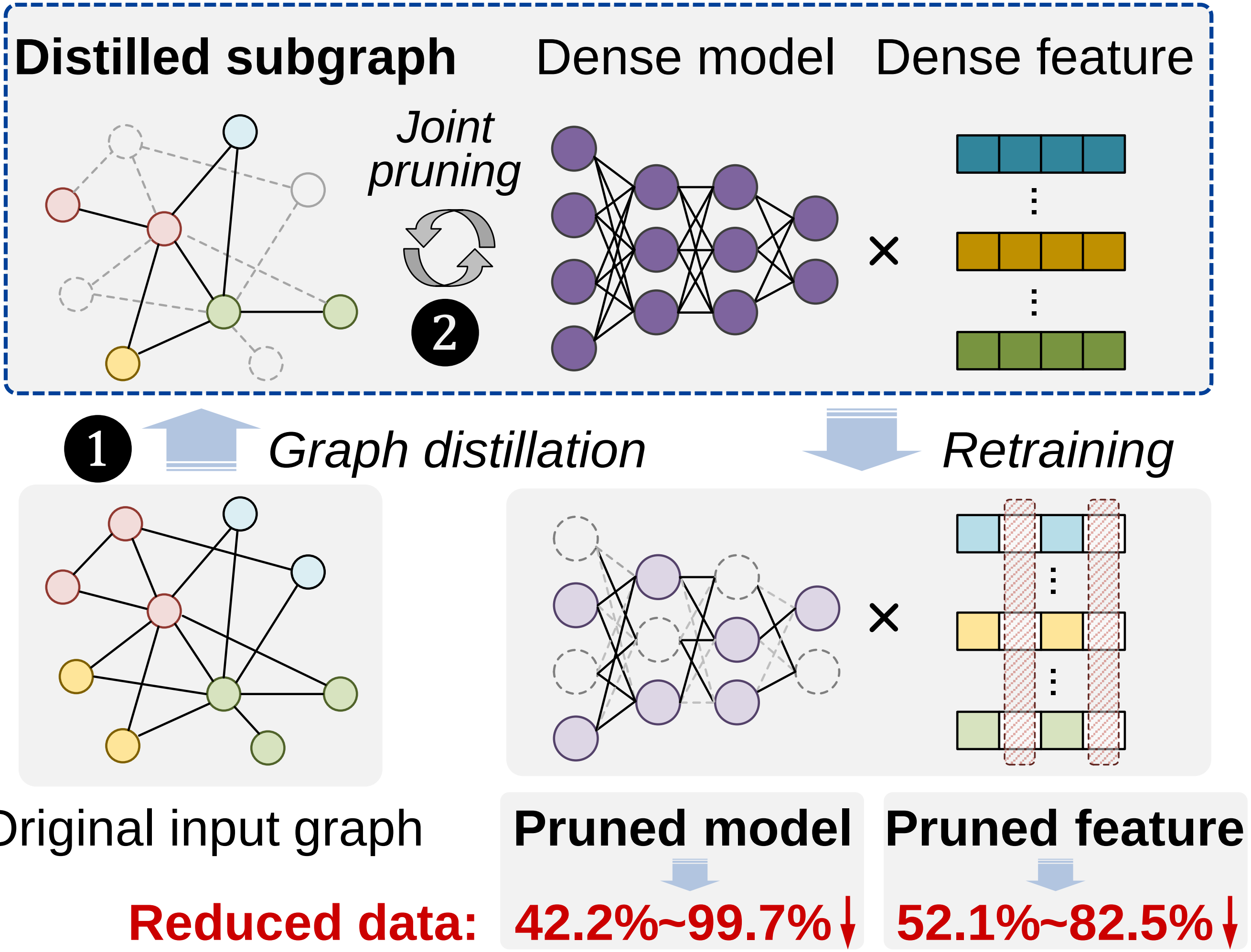
Outline

- Background & Motivation
- **Algorithm Optimizations**
- Dataflow & Hardware Design
- Evaluations

Pruning upon Distilled Subgraph

Exploit feature & model sparsity with pruning overhead reduction

Reduce graph size: ~80%



Algorithm Details

1 Graph distillation

- Reserve vertices in graph clusters
- Use *edge similarity* as the distillation metric

$$e_{sim} = \frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|}$$

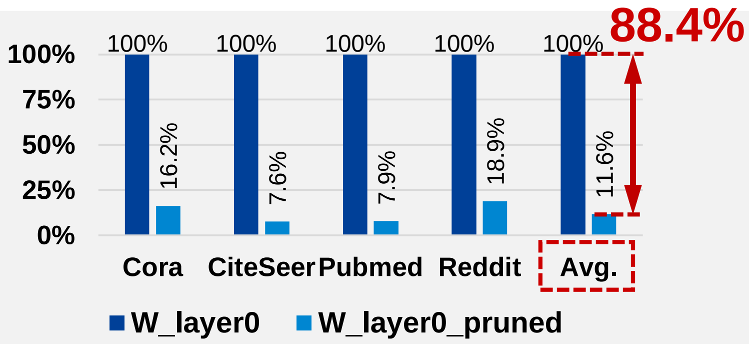
2 Feature-model joint pruning

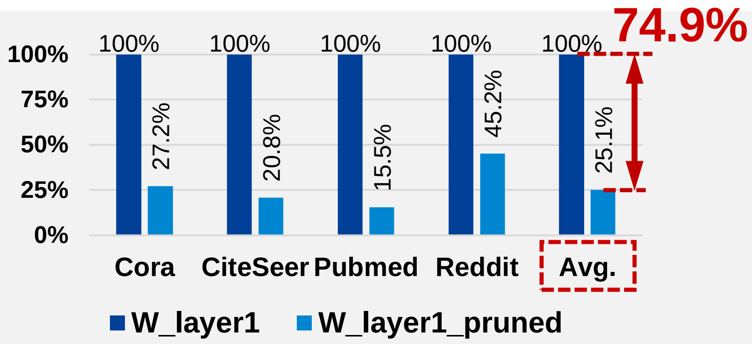
- Use dedicated feature and weight masks to indicate significant elements

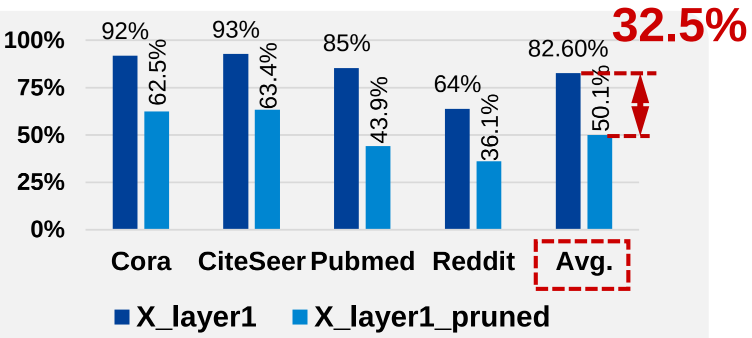
$$\mathcal{L}_{PDS} = \mathcal{L}(\mathcal{G}_{sparse}, X, m_\theta \odot \Theta) + \gamma_1 \underbrace{\|m_x\|_1}_{\text{Feature mask}} + \gamma_2 \underbrace{\|m_\theta\|_1}_{\text{Weight mask}}$$

Benefits of GNN Sparsification

Feature & model pruning reduces operation requirement





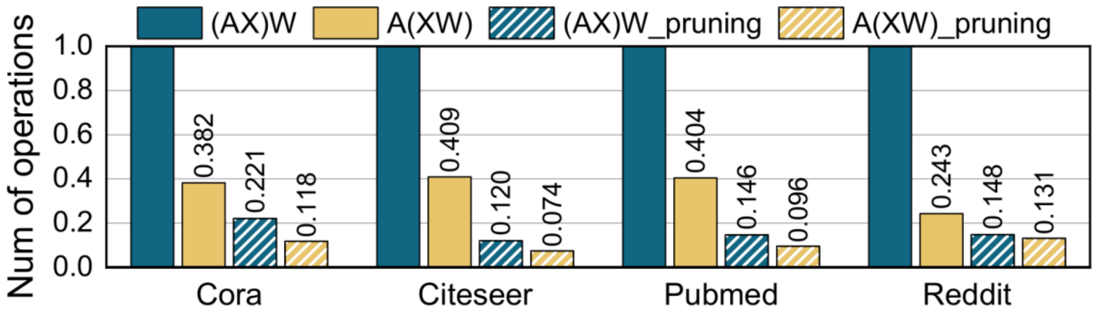


Sparsity exploitation @GCN (2-layer) on average:

Pruning ratio: Wgt@layer1: **88.4%**; Wgt@layer2: **74.9%**; Feature@layer1: **32.5%**

Accuracy	Cora	CiteSeer	Pubmed	Reddit
W/O (%)	81.7	71.4	78.8	90.9
Pruning (%)	81.1	71.9	78.5	90.3

Retain model accuracy (**±0.6%**)



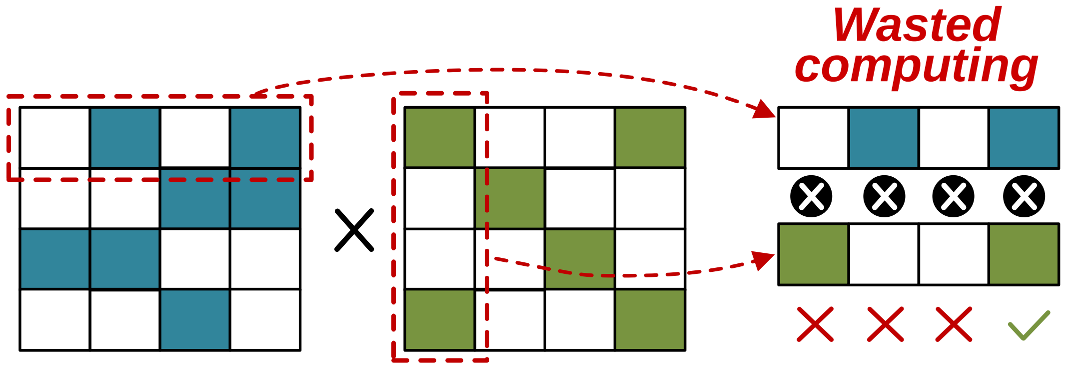
Reduce up to **5.5×** MAC operations

Challenges of GNN Sparsification

Sparsification results in hardware inefficiency

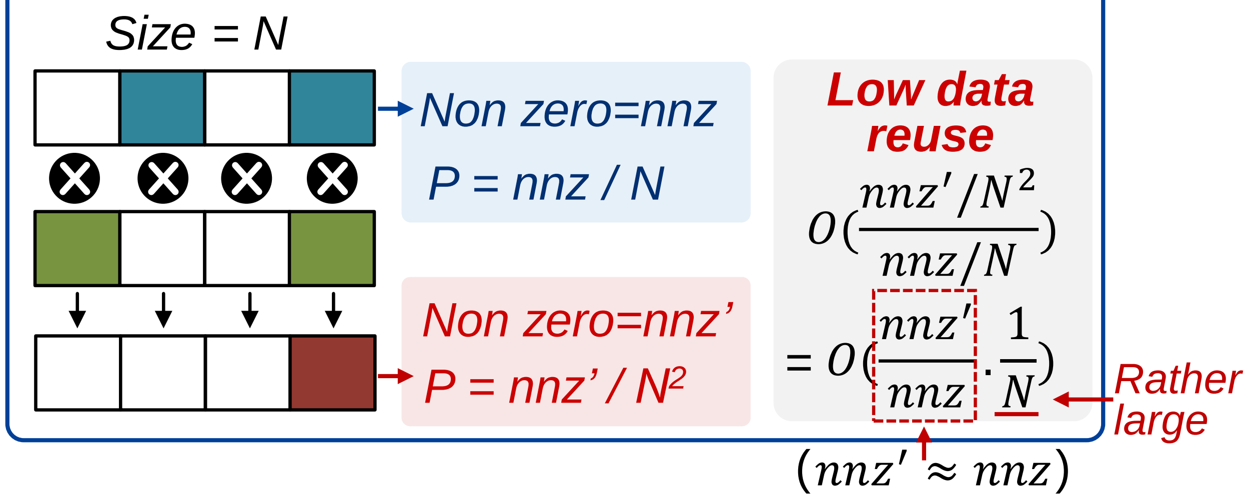
Inefficient Product

- Wasted computation on zero elements
- Costly index match for selecting non-zero elements



Low data reuse

- Data reuse is rather low in conventional feature-weight matrix product



Need efficient dataflow and hardware supporting

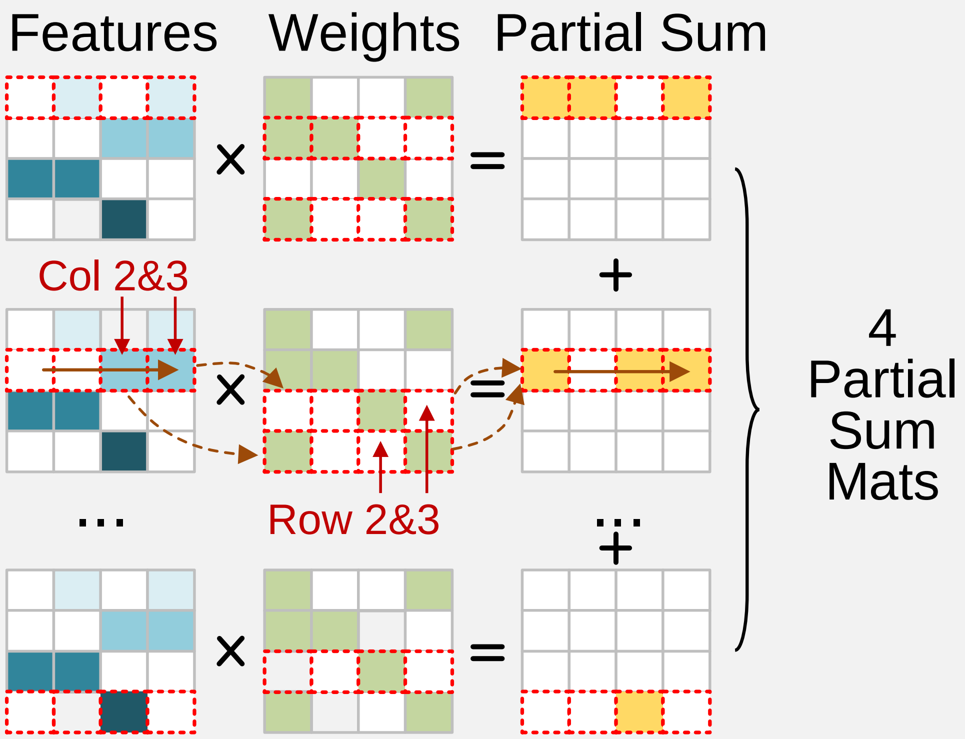
Outline

- Background & Motivation
- Algorithm Optimizations
- **Dataflow & Hardware Design**
- Evaluations

Compressed Product Dataflow

Compressed row(col)-wise product to improve data reuse

Conventional row-wise product



Limitation: low access locality in the weight matrix

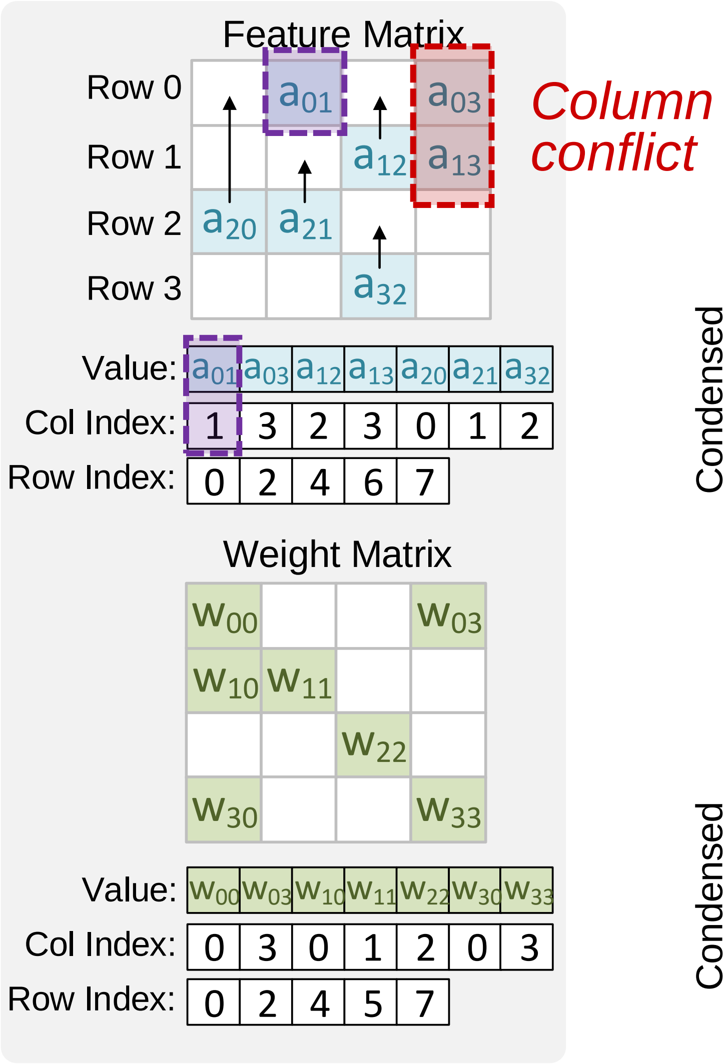
Compressed row-wise product

Advantages:

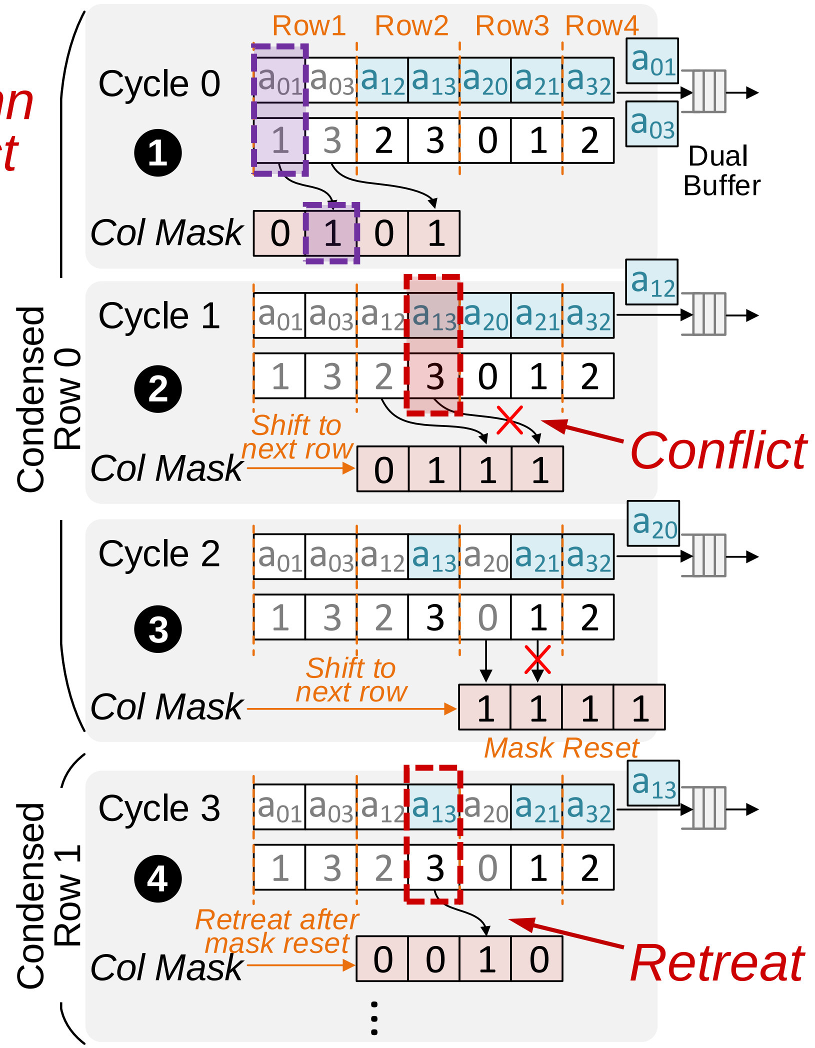
- Improve access locality and data reuse
- Reduce the number of partial sum matrix

Compressed Product Dataflow

Feature & weight matrix



Compress workflow



1 Select non-zero elements

- Use a column mask to identify non-zeros in parallel

2 Skip conflict and shift

- A column has been occupied if its mask bit has been set

3 Reset mask if all bits are set

- Once all bits have been set, a compressed row is generated

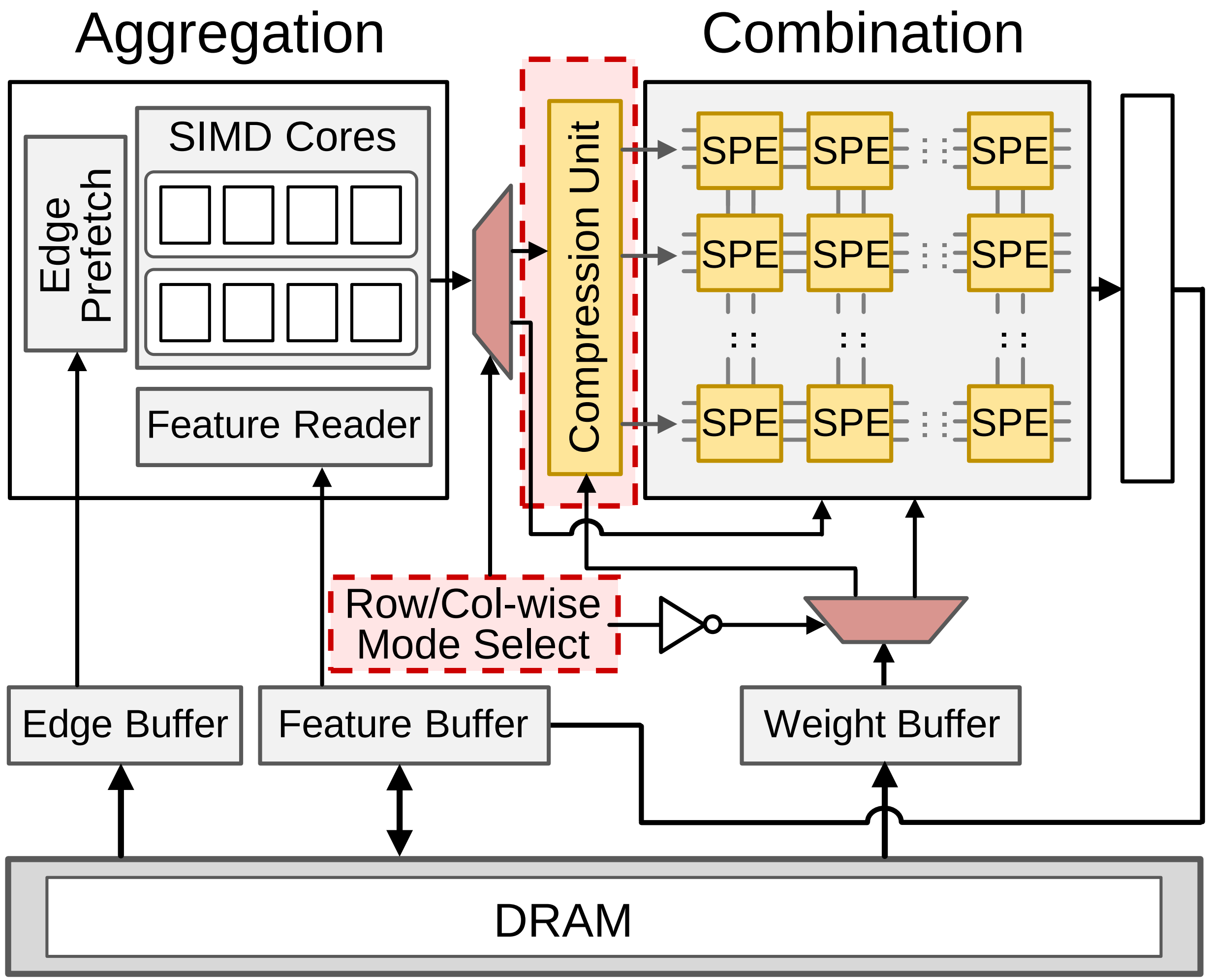
4 Retreat after mask reset

- Retreat the mask to rows where still remains uncompressed elements

Hardware Design

Overall Architecture

- Based on a classical GNN accelerator[1]
- Add a ***Compression Unit*** to compress sparsified features and weights
- Model select: support compressed row-wise and col-wise product

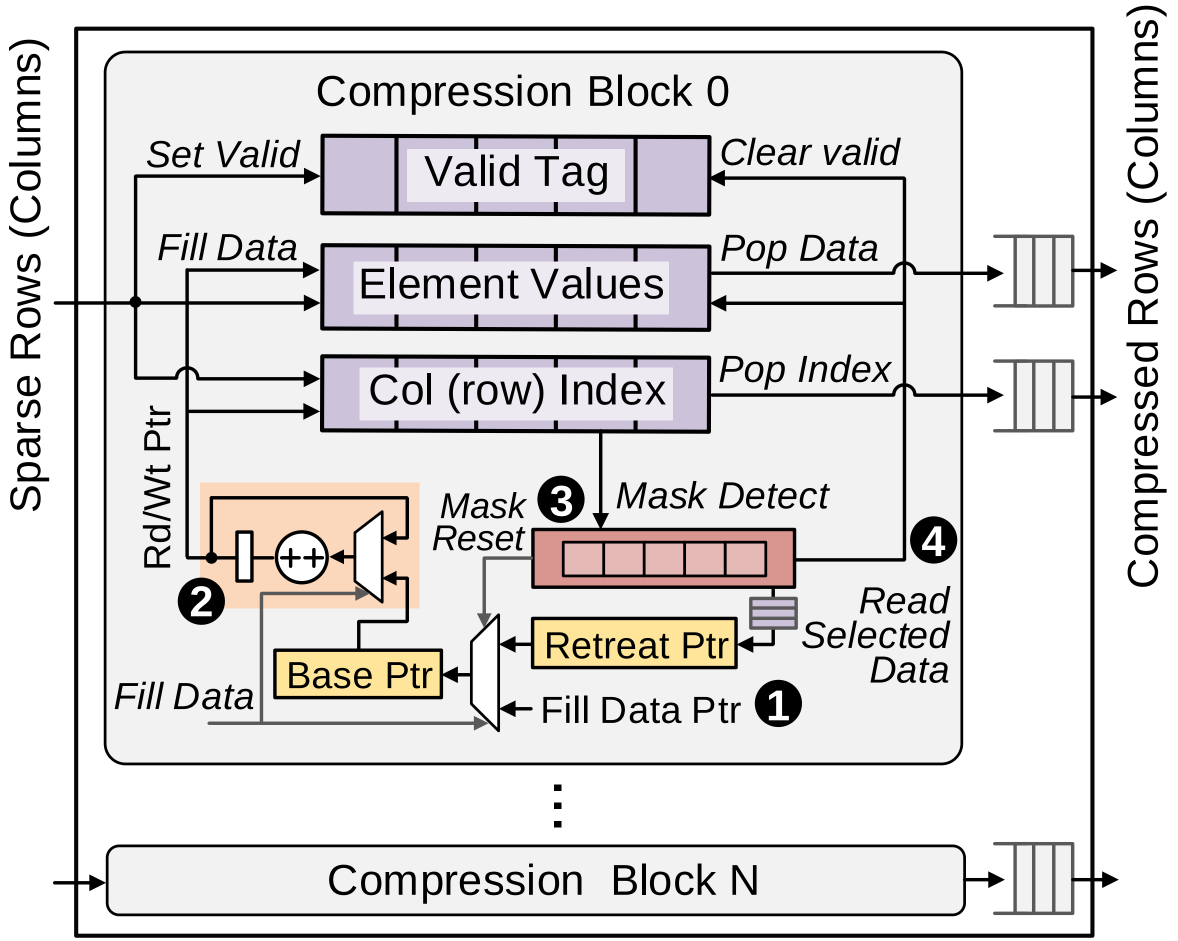


[1] Mingyu Yan, et al. HyGCN: A GCN Accelerator with Hybrid Architecture (HPCA 2020)

Hardware Design

Compression Unit

- ❶ Fill non-zero element into three cyclic buffer (Tag, Val, Index) as FIFO
- ❷ Update read and write pointer (Only keep one set pointers for these three buffer)
- ❸ Mask detection:
  - Detect column conflict
  - Reserve retreat pointers
- ❹ Read out compressed data



Outline

- Background & Motivation
- Algorithm Optimizations
- Dataflow & Hardware Design
- **Evaluations**

Evaluations

Experimental Setup

GNN Model

- GCN, GraphSAGE, GIN, GAT
- Layer=2, hidden=512 as GLT[1]

Dataset

Accuracy	Cora	CiteSeer	Pubmed	Reddit
# Vertex	2,708	3,327	19,717	232,965
# Edge	13,264	12,431	108,365	1.148E8
Avg. deg	4.9	3.7	5.5	493
Input length	1433	3703	500	602
Output length	7	6	3	41

[1] Tianlong Chen, et al. ICML 2021

System Configuration

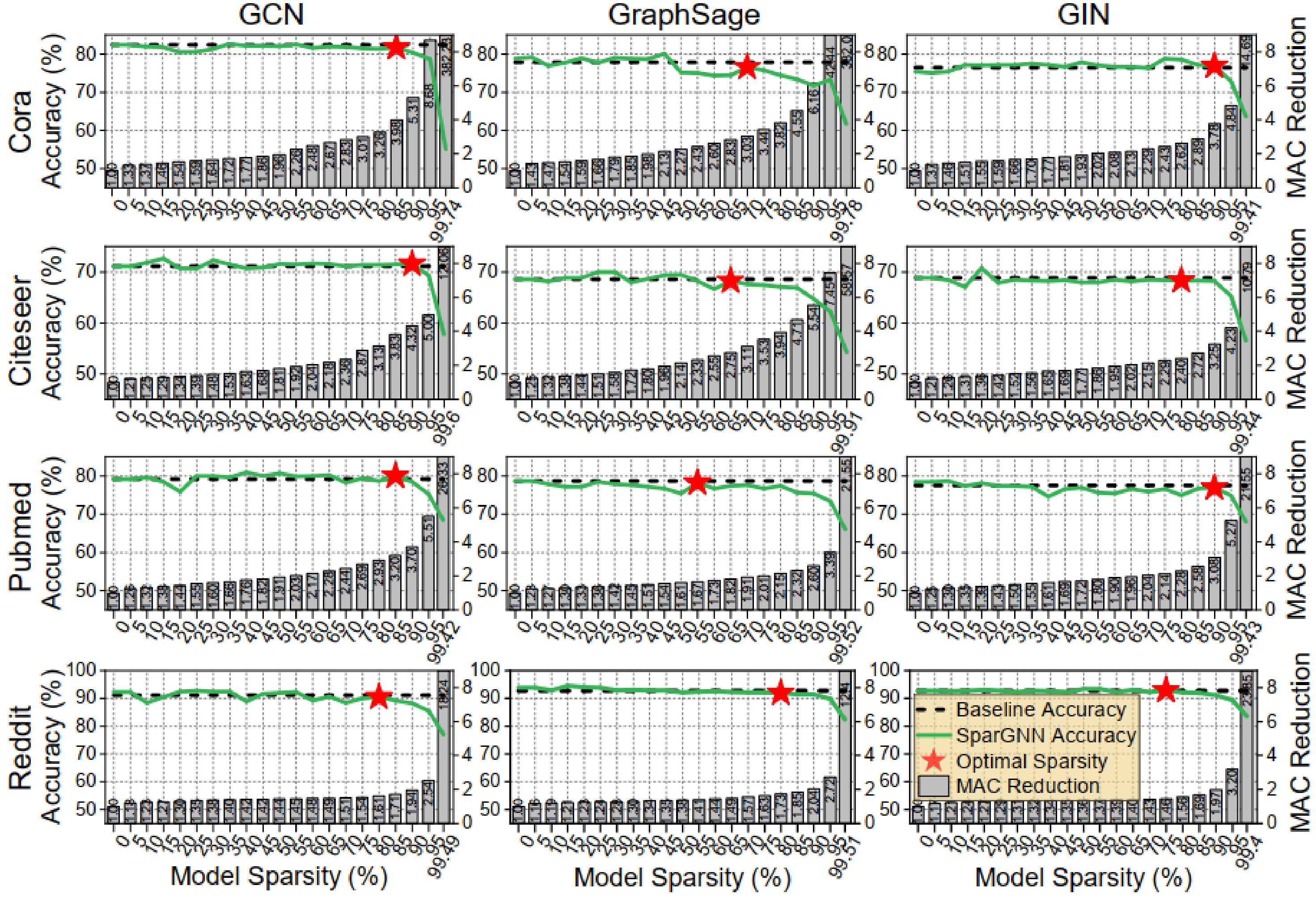
- Cycle accurate simulator + DRAMSim3[5]

	HyGCN[2]	I-GCN[3]	FlowGCN[4]	Ours
Compute (@1GHz)	32 SIMD cores + 4K PEs	64 traversal engines + 4K PEs	32 multicasting adapter + 4K PEs	32 SIMD cores + 4K PEs
On-chip Memory (MB)	Edge/Wgt: 2 Feature: 2 Coordinator: 16	Edge/Wgt: 2 Feature: 2 Hub cache: 16	Edge/Wgt: 2 Feature: 2 Msg buffer: 16	Edge/Wgt: 2 Feature: 2 Compression: 16
Off-chip Memory	256GB/s HBM	256GB/s HBM	256GB/s HBM	256GB/s HBM

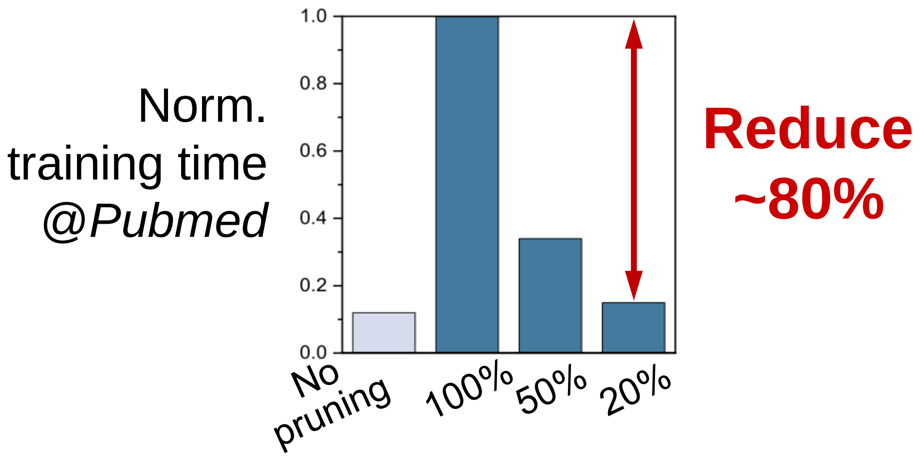
[2] Mingyu Yan, et al. HPCA 2020
[3] Tong Geng, et al. MICRO 2021
[4] Rishov Sarkar, et al. HPCA 2023
[5] Shang Li, et al. IEEE CAL 2020

Evaluations

Accuracy with pruning



Reduced pruning overhead



Model accuracy

- Sustain model accuracy with **42.2%~99.7%** model and **52.1%~82.5%** feature pruning

Pruning overhead

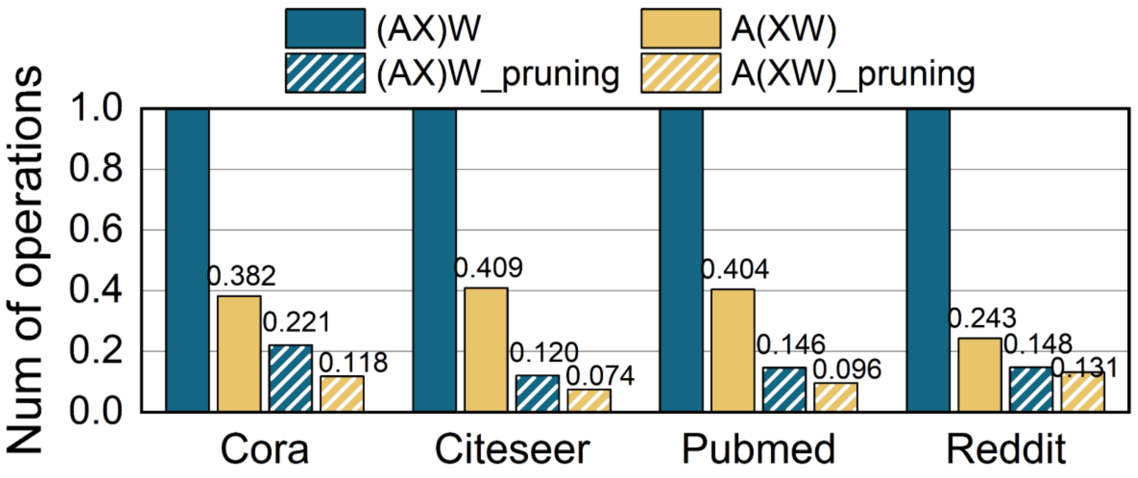
- Reduce **~80%** training time overhead

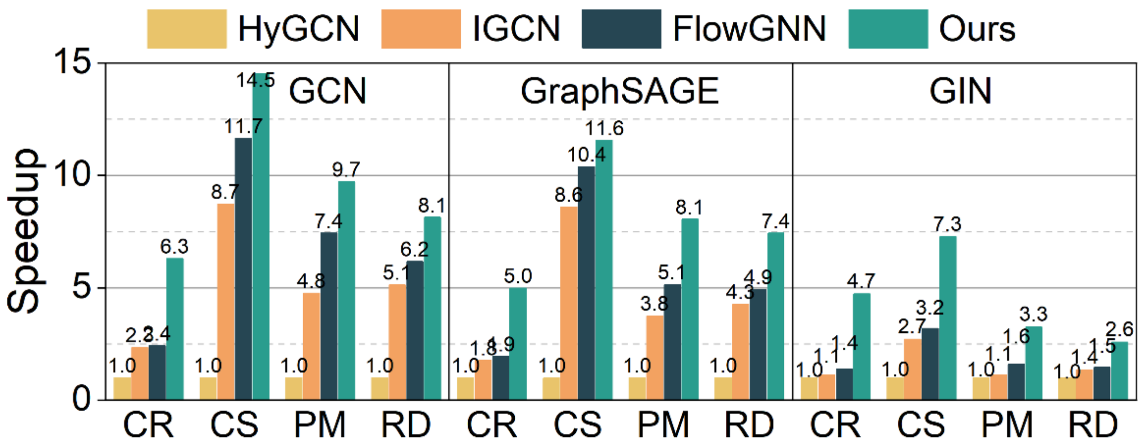
Evaluations

Reduced MAC Operations & Speedup

- Our work reduces **1.8×~5.5×** MAC operation by joint features & model pruning

- Our work achieves **6.8×, 2.3× and 1.8×** speedup compared to HyGCN, I-GCN and FlowGNN

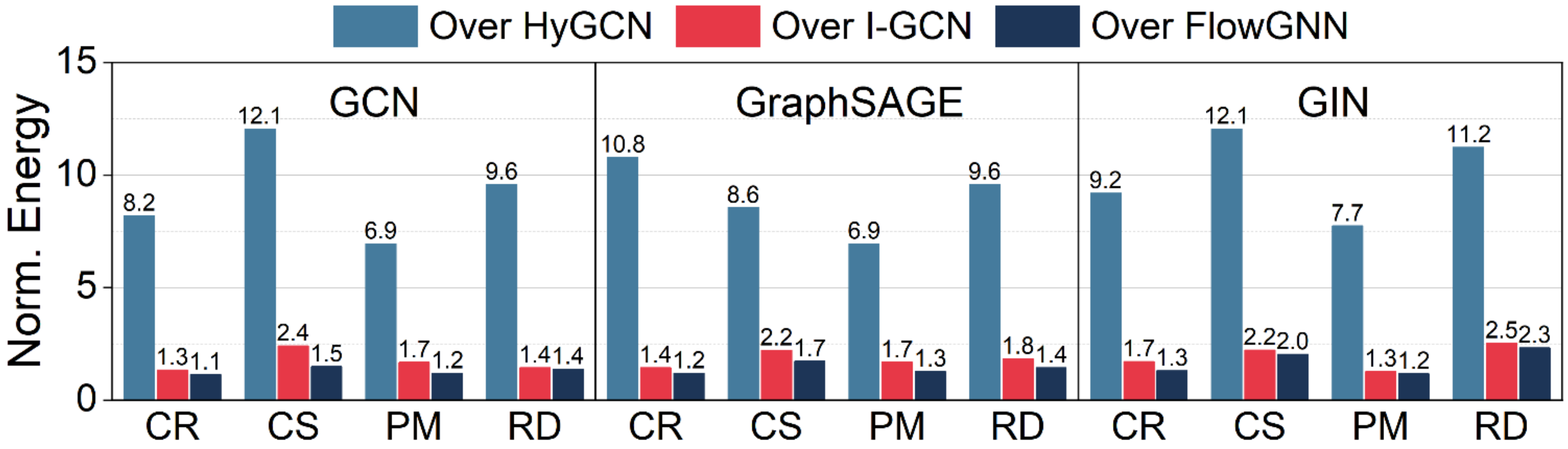




Evaluations

Energy Efficiency Improvement

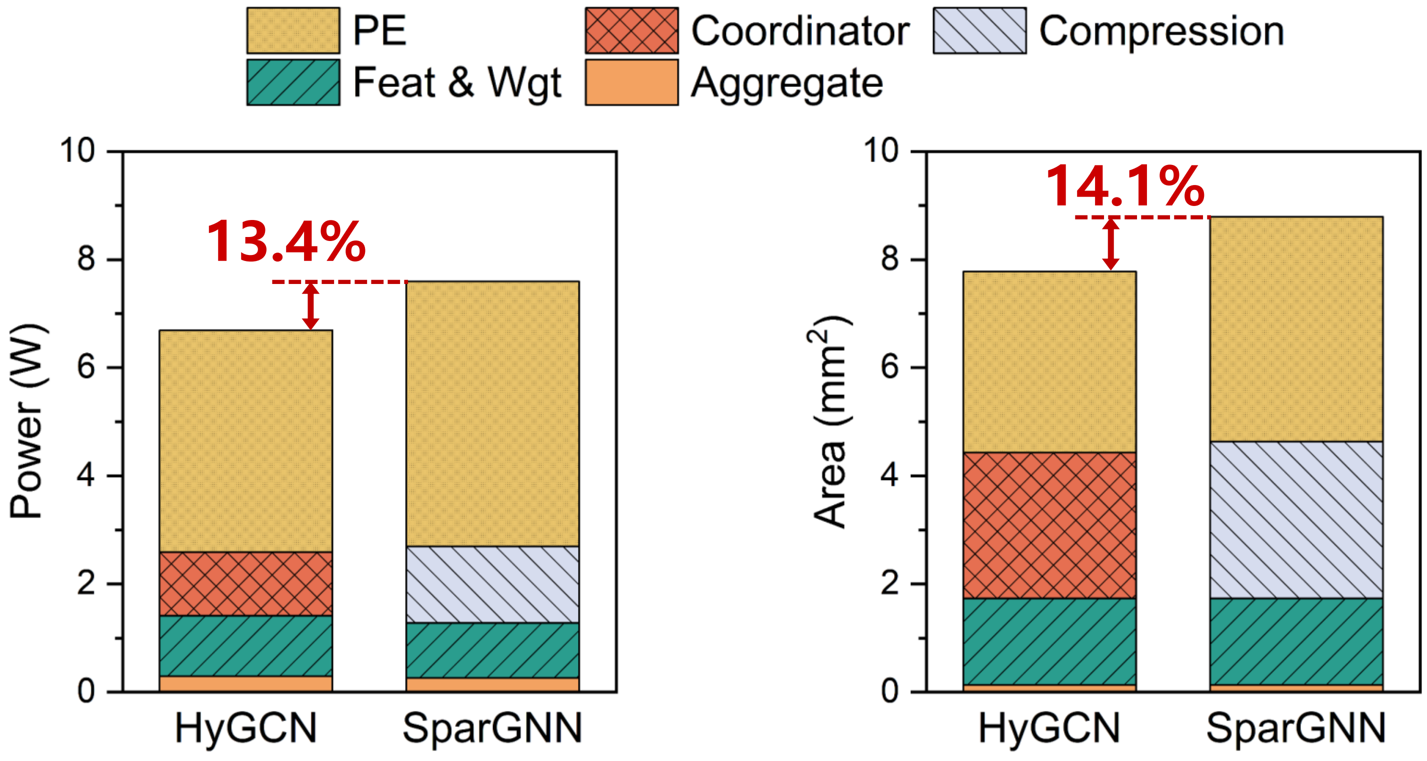
- Compared to HyGCN, I-GCN and FlowGNN, our work can achieve **9.2×, 1.8× and 1.4×** energy efficiency on average



Evaluations

Power & Area

- The total power and area of our work are **7.6 *W*** and **8.9 *mm*²**, which increase **13.4%** and **14.1%** compared to HyGCN



Conclusion

Challenges:

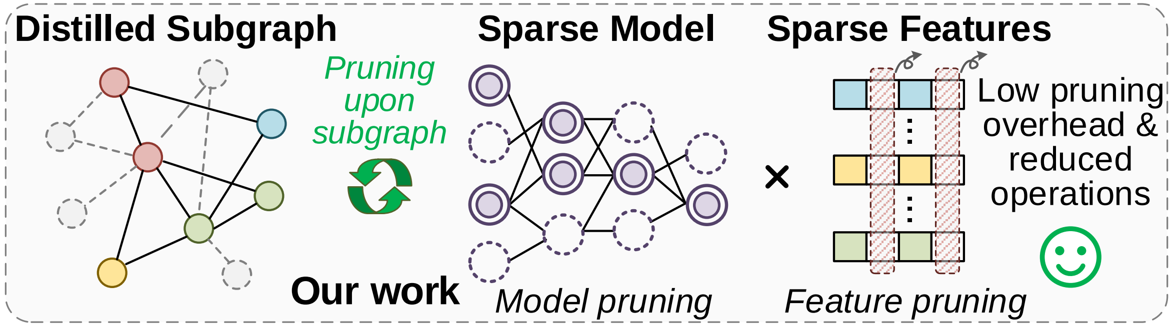
- In algorithm: pruning upon large input graph significantly **increases training overhead**
- In hardware: GNN sparsification **exacerbates hardware inefficiency**

Solution:

- In algorithm: feature & model **joint pruning upon a sparsified subgraph**
- In hardware: **compressed row(col)-wise product** for data reuse improvement

Achievement:

- Reduce MAC operation: **1.85×~5.52×**
- Compared to SOTA accelerators: **1.8×~6.8×** speedup**;** **1.4×~9.2×** energy efficiency



SparGNN: Efficient Joint Feature-Model Sparsity Exploitation in Graph Neural Network Acceleration

Thank you !

Q&A

Presenter: Chen Yin

Email: yinchen@sjtu.edu.cn