GCoD: Graph Convolutional Network Acceleration via Dedicated Algorithm and Accelerator Co-Design

Haoran You[1], Tong Geng[2], Yongan Zhang[1], Ang Li[2], Yingyan Lin[1]

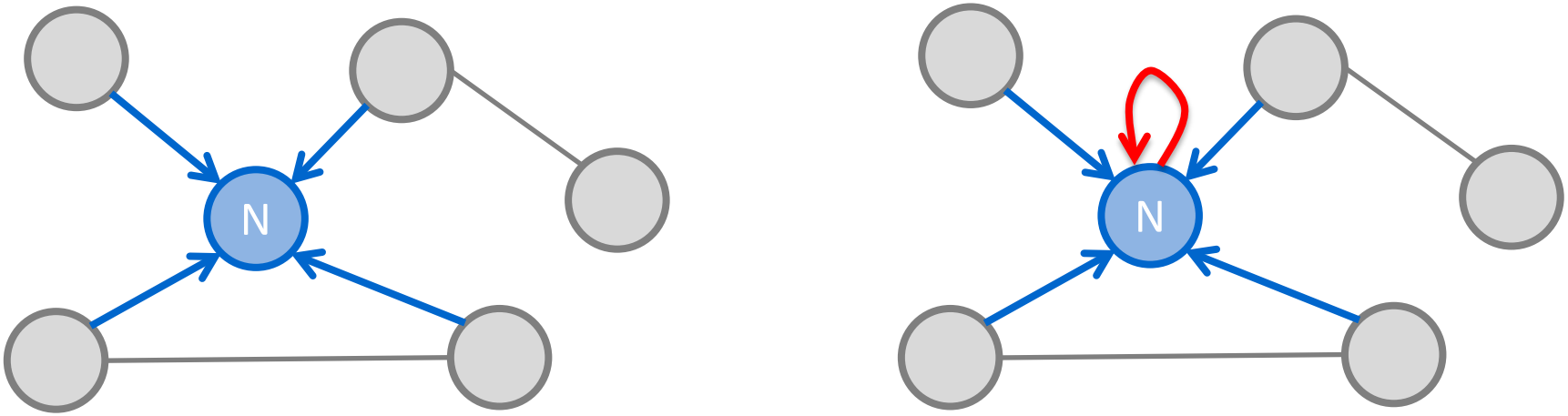
[1]Rice University

[2]Pacific Northwest National Laboratory

The 28th IEEE International Symposium on
High-Performance Computer Architecture (HPCA 2022)

Background: Graph Convolutional Networks (GCNs)

- GCNs show better performance on graph-based data than CNNs
  - Aggregation phase
    - Aggregate the information of neighbors
  - Combination phase
    - Combine the information from both neighbors and node itself

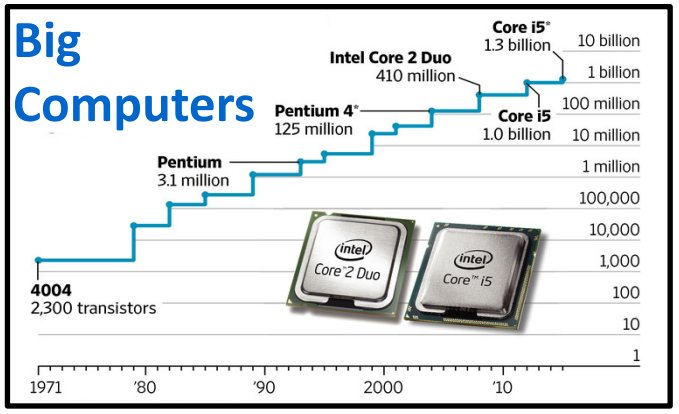


Update the feature of node N

Background: GCN Inference Needs Acceleration

- **SOTA GCNs requires inference with big graphs and big computers**

GCN Performance



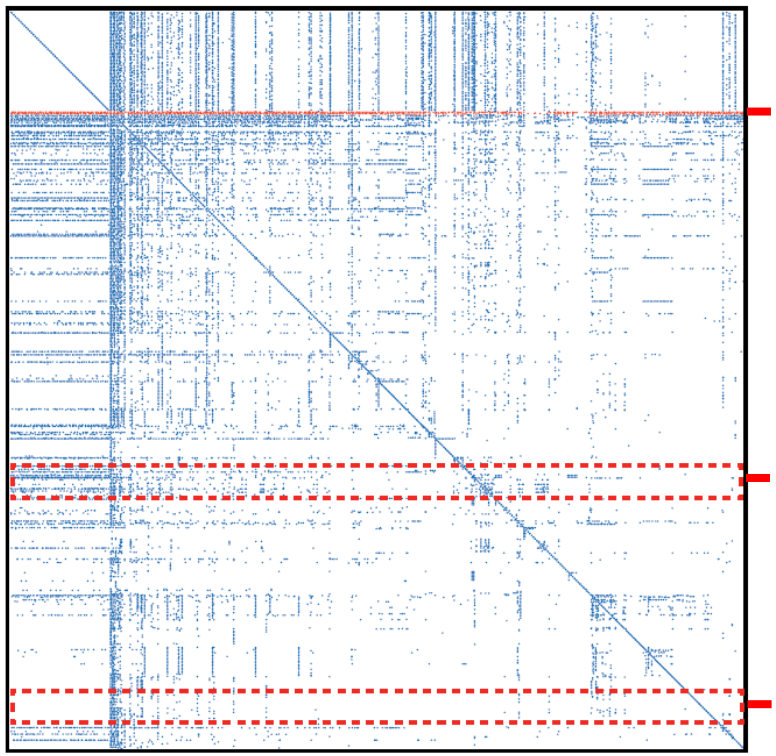
Background: GCN Inference Needs Acceleration

- **The prohibitive cost of GCNs**
  - **A mere 2-layer GCN model with 32 hidden units**
    - 19G FLOPs to process the Reddit graph
  - **A medium-scale CNN model ResNet-50**
    - 8G FLOPs to inference on ImageNet

Challenges: How to make GCN Inference Efficient?

- **The prohibitive cost of GCNs**

- **Why inference GCNs cost so much?**
  - **Real-world graphs are very large and irregular, and have complex neighbor connections**
    (E.g., 232,965 nodes in Reddit graph; each node has about 50 neighbors)



Adjacency matrix of NELL

Challenges: How to make GCN Inference Efficient?

- **The prohibitive cost of GCNs**

- **Why inference GCNs cost so much?**
  - **Real-world graphs are very large and irregular, and have complex neighbor connections**
    (E.g., 232,965 nodes in Reddit graph; each node has about 50 neighbors)

  - **The extremely high sparsity within the graph adjacency matrices**
    (E.g., often exceeds 99.9% vs. 10% ~ 80% in CNNs)

    → Workload imbalance during inference

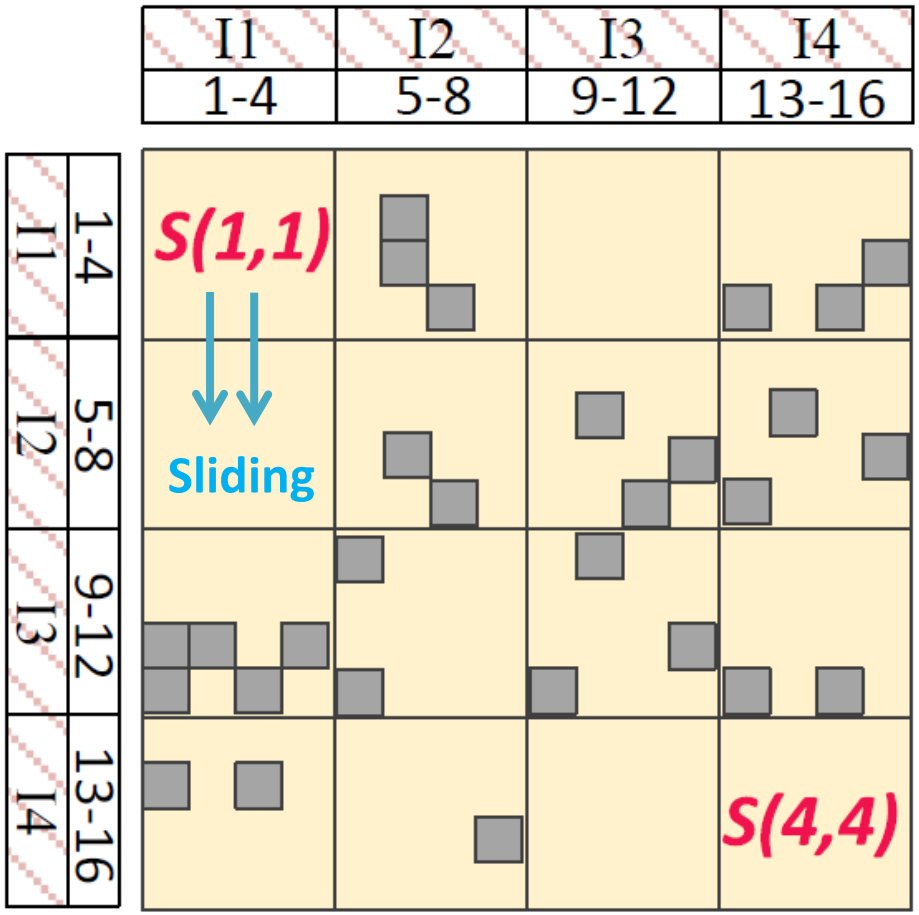
Challenges: How to make GCN Inference Efficient?

- **The prohibitive cost of GCNs**

- **Why inference GCNs cost so much?**
  - **Real-world graphs are very large and irregular, and have complex neighbor connections**
    (E.g., 232,965 nodes in Reddit graph; each node has about 50 neighbors)
  - **The extremely high sparsity within the graph adjacency matrices**
    (E.g., often exceeds 99.9% vs. 10% ~ 80% in CNNs)
  - **The dimension of node features can be very high**
    (E.g., each node in the CiteSeer graph has 3703 features)

Previous State-of-the-art GCN Inference Accelerators

- **Previous accelerators: Only consider accelerator design**
  - **HyGCN** [Yan et al., HPCA'20]
    - Skips zeros in adjacency matrices via window sliding



Workflow:

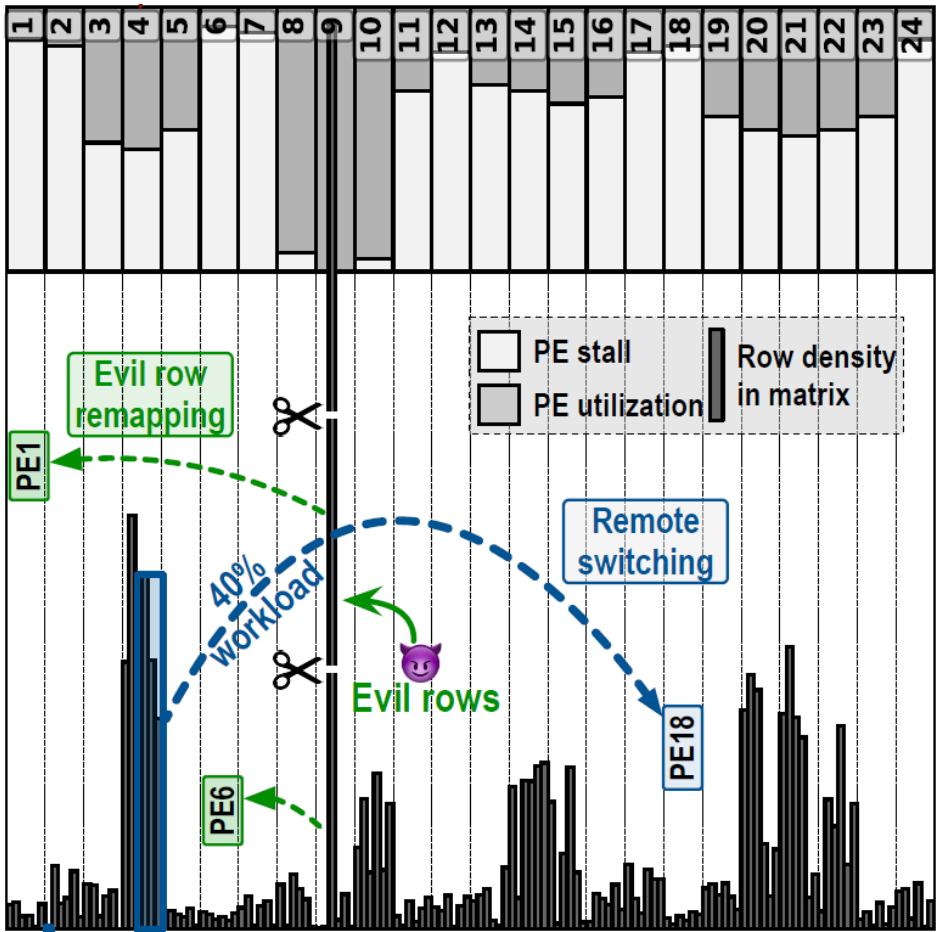
Group the vertices within the same interval together (e.g., I_i), and then aggregate their neighbor's information interval by interval (i.e., traverse I_j)

Previous State-of-the-art GCN Inference Accelerators

- **Previous accelerators: Only consider accelerator design**
  - **HyGCN** [Yan et al., HPCA'20]
    - Skips zeros in adjacency matrices via window sliding
  - **AWB-GCN** [Geng et al., MICRO'20]
    - Reduces the irregularity via auto-tuning workload balance techniques during runtime

To achieve balanced PE workload, they consider:

- **Evil Row Remapping**
- **Remote Switching**
- **Distribution Smoothing**



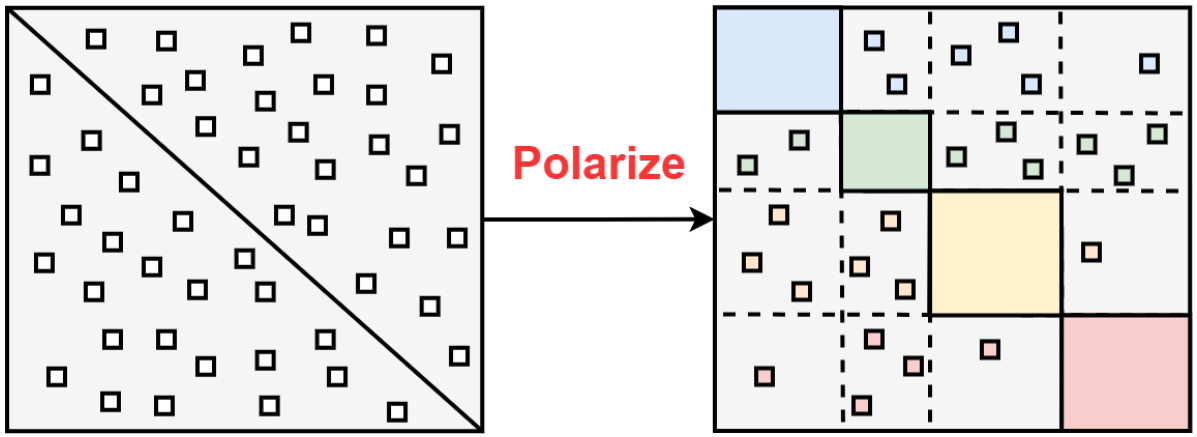
Limitation of Previous GCN Inference Accelerators

- **Previous accelerators: Only consider accelerator design**
  - **HyGCN** [Yan et al., HPCA'20]
    - Skips zeros in adjacency matrices via window sliding
  - **AWB-GCN** [Geng et al., MICRO'20]
    - Reduces the irregularity via auto-tuning workload balance techniques during runtime

- **Limitation**
  - Both HyGCN and AWB-GCN require large on-the-fly control overhead for reducing the irregularity

How to Overcome the Limitation?

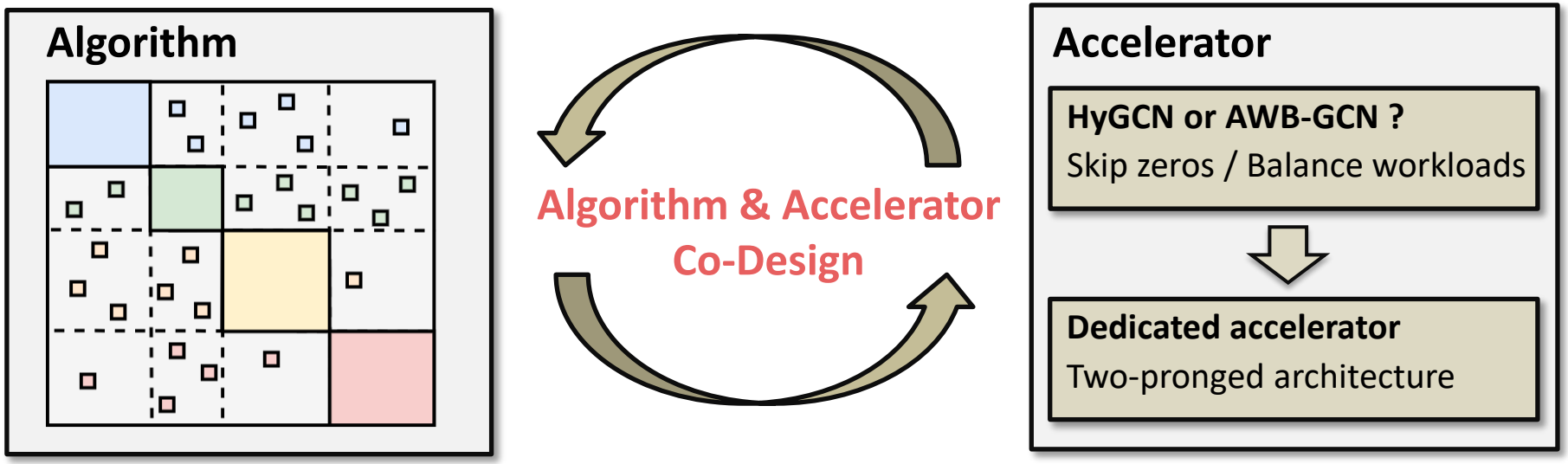
- **Previous accelerators: Only consider accelerator design**
- **Limitation**
  - Both HyGCN and AWB-GCN require large on-the-fly control overhead for reducing the irregularity
- **Can we reduce irregularity from the algorithm side?**
  - Enforce both regularity and workload balance by polarizing the graph adjacency matrices into denser or sparser parts



Graph adjacency matrix

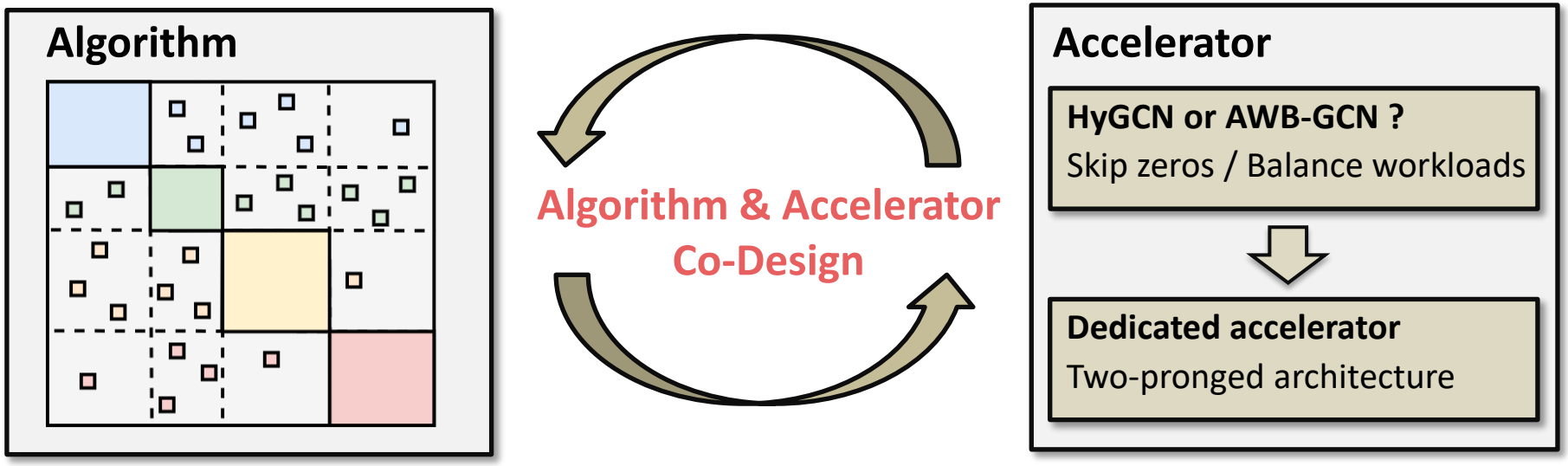
Proposed GCoD: Algorithm & Accelerator Co-Design

- **Proposed GCN algorithm & accelerator co-design (GCoD)**
  - Leverage GCN algorithm & accelerator co-design to reduce the irregularity as well as boosting the efficiency and utilization.



Proposed GCoD: Algorithm & Accelerator Co-Design

- **Proposed GCN algorithm & accelerator co-design (GCoD)**
  - Leverage GCN algorithm & accelerator co-design to reduce the irregularity as well as boosting the efficiency and utilization.



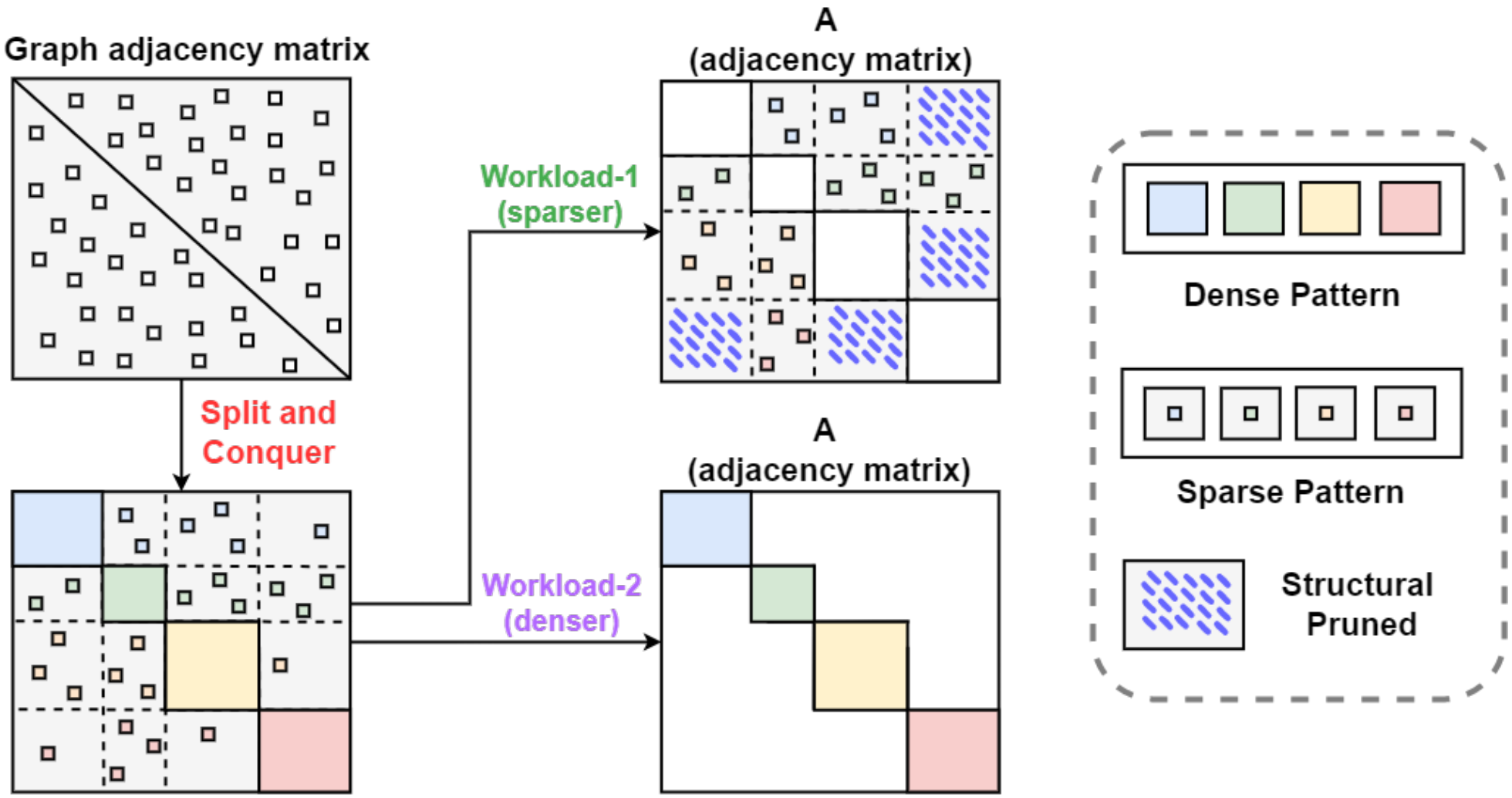
- **GCoD's Challenges**
  - On the algorithm level, how to enforce regularity?
  - On the hardware level, how to dedicatedly support the enforced regularity?

Our Contributions

For the first time, we

- **Propose a GCN algorithm & accelerator co-design framework, dubbed as GCoD**
- **On the algorithm level, GCoD integrates a split and conquer training to polarize the graphs to be either denser or sparser without compromising the accuracy**
- **On the hardware level, GCoD further develops a dedicated two-pronged accelerator with separated engines to process each of enforced workloads**

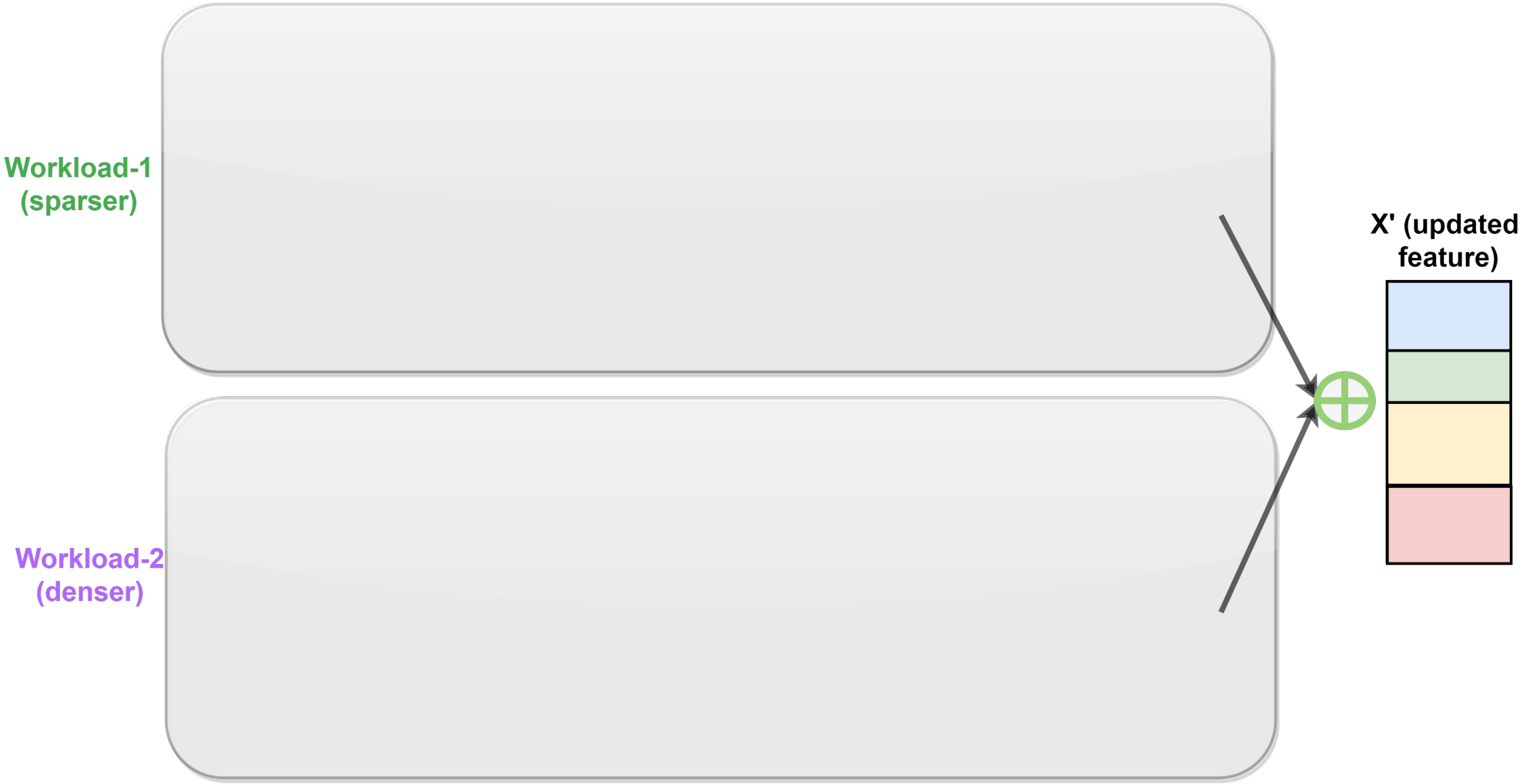
GCoD Overview: Algorithm & Accelerator Co-Design



GCN Algorithm & Accelerator Co-Design:

The core idea in the algorithm side is to split the workload into two parts, i.e., denser and sparser.

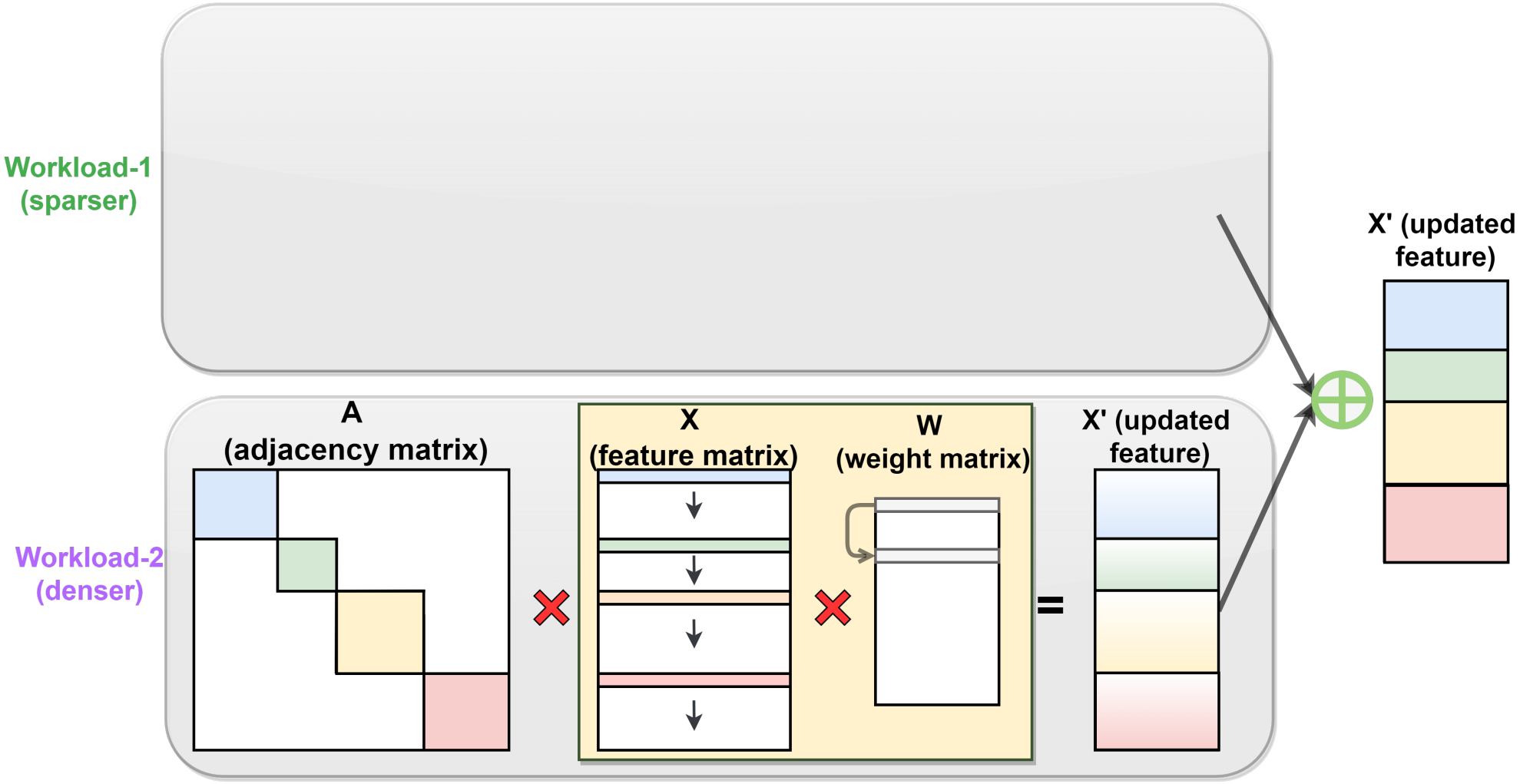
GCoD Overview: Algorithm & Accelerator Co-Design



GCN Algorithm & Accelerator Co-Design:

The core idea in the accelerator side is to handle the divided denser or sparser workloads separately.

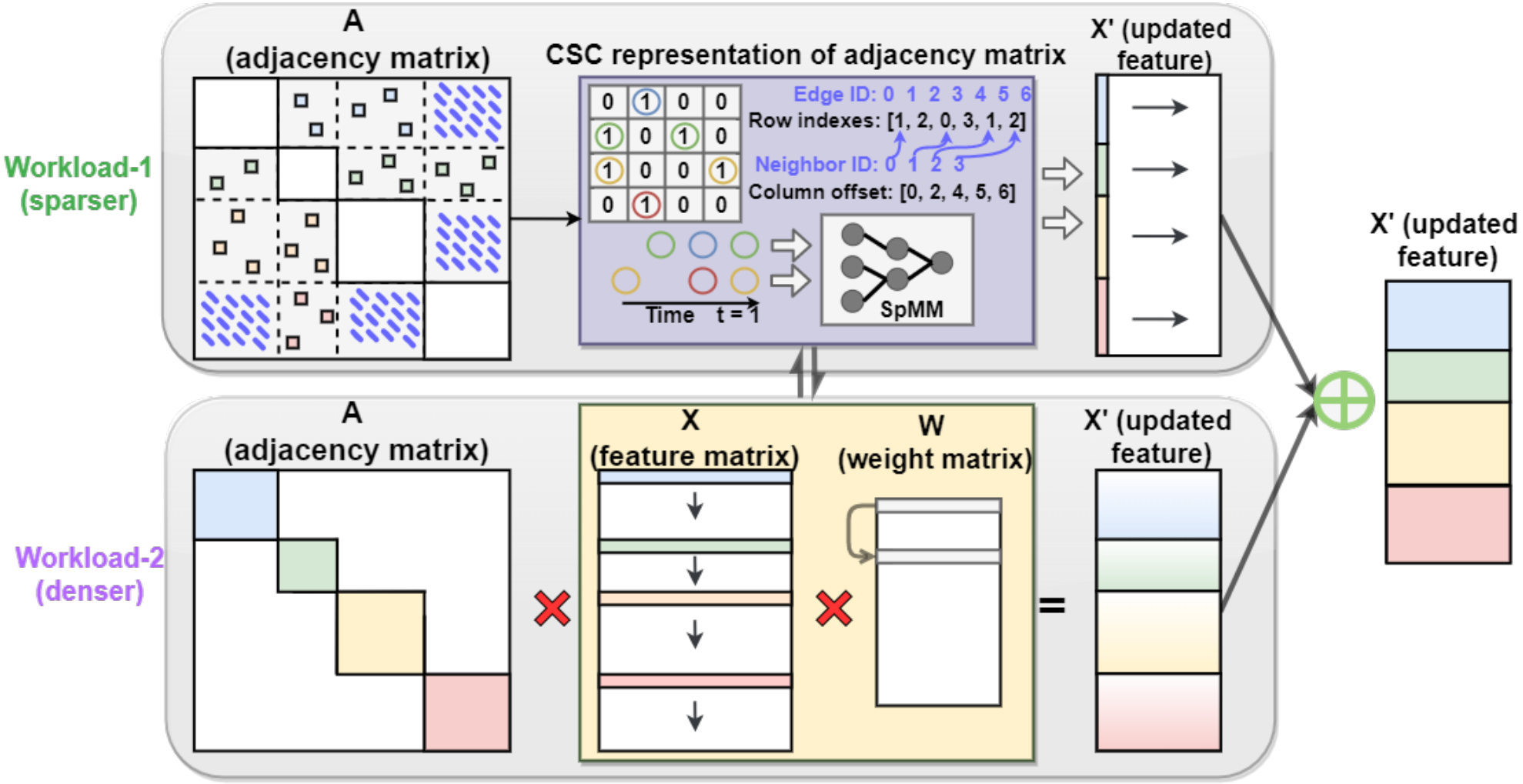
GCoD Overview: Algorithm & Accelerator Co-Design



GCN Algorithm & Accelerator Co-Design:

The core idea in the accelerator side is to handle the divided denser or sparser workloads separately.

GCoD Overview: Algorithm & Accelerator Co-Design

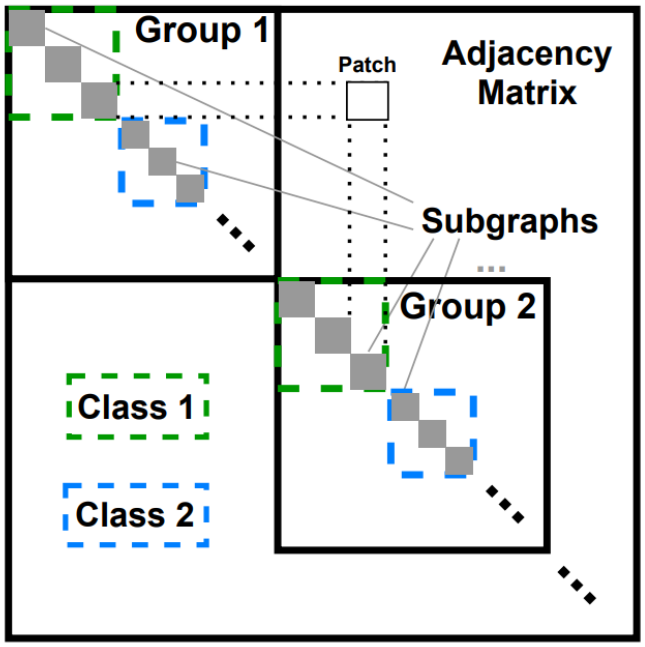


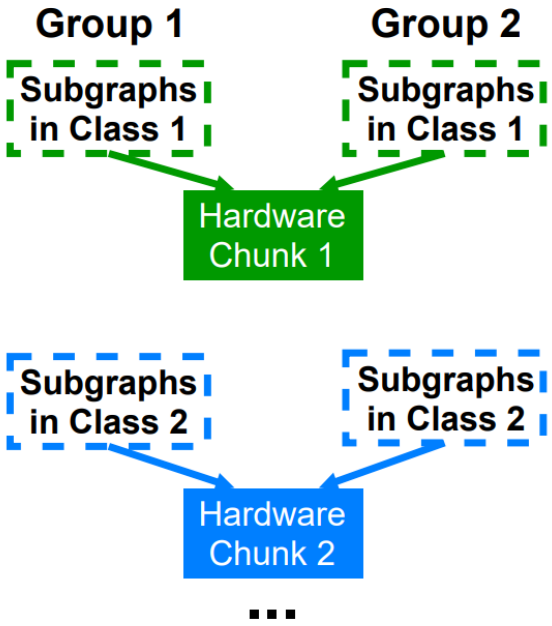
GCN Algorithm & Accelerator Co-Design:

The core idea in the accelerator side is to handle the divided denser or sparser workloads separately.

GCoD Algorithm: Split and Conquer Training

- ***Challenge: How to reduce the irregularity?***
  - **Design insight for polarizing graphs into denser or sparser workloads**
    - **Subgraph classification**
      - Cluster nodes with similar degrees into the same class
      - Divide each class into subgraphs (of similar edges)
    - **Group partitioning (optional)**
      - Uniformly distribute subgraphs into different groups (avoid too clustered dense areas → increase reuse opportunity)

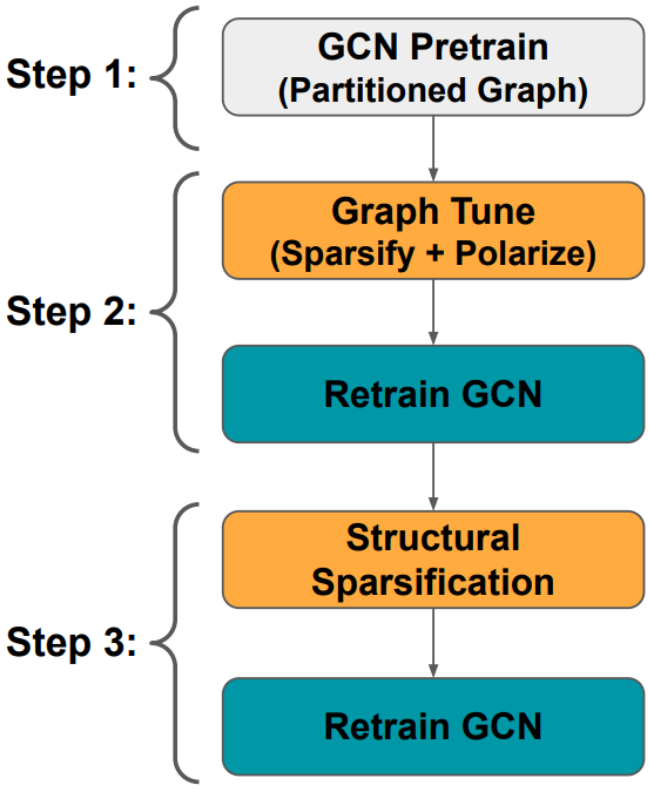




GCoD Algorithm: Split and Conquer Training

- **The proposed split and conquer training**
  - **Step 1: GCNs pretraining on the partitioned graphs**
    - Extract C subgraph classes according to the degrees
    - Use METIS to partition each class into workload balanced subgraphs
    - Pretrain GCNs on the partitioned graphs

$$\mathcal{L}_{GCN}(W) = -\sum_{n \in \mathcal{Y}_N} \sum_{f} Y_{nf} \, ln(Z_{nf})$$

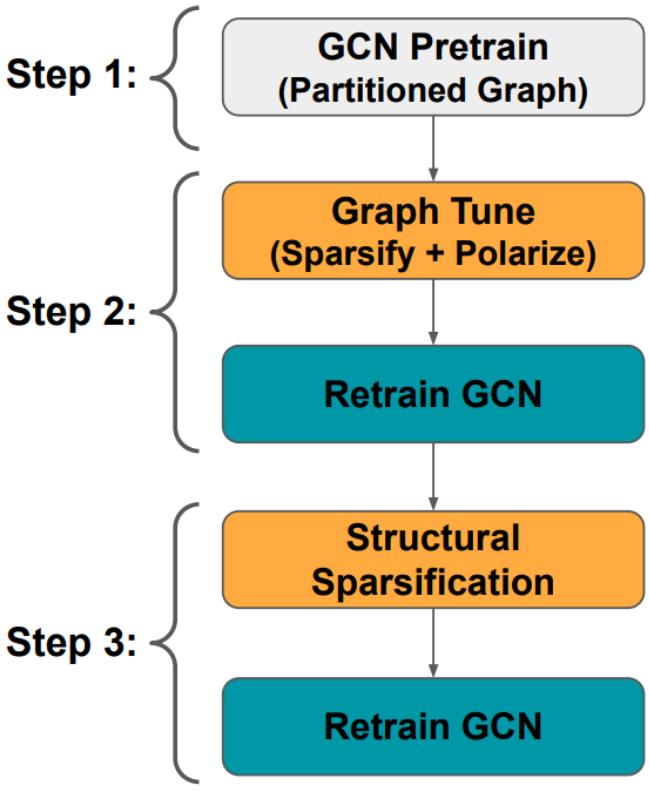


GCoD Algorithm: Split and Conquer Training

- **The proposed split and conquer training**
  - **Step 1: GCNs pretraining on the partitioned graphs**
    - Extract C subgraph classes according to the degrees
    - Use METIS to partition each class into workload balanced subgraphs
    - Pretrain GCNs on the partitioned graphs
  - **Step 2: Sparsify and diagonalize the adjacency matrices**
    - Sparse regularization term
    - Polarization regularization term

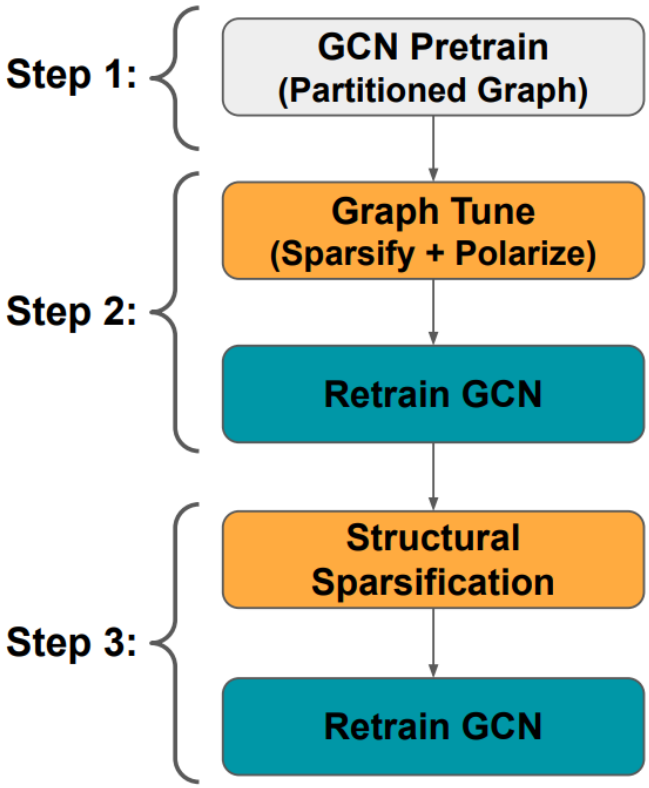
$$\mathcal{L}_{Graph}(A) = \mathcal{L}_{GCN}(A) + \mathcal{L}_{SP}(A) + \mathcal{L}_{Pola}(A)$$

$$1/M \cdot \sum \|i - j\|$$



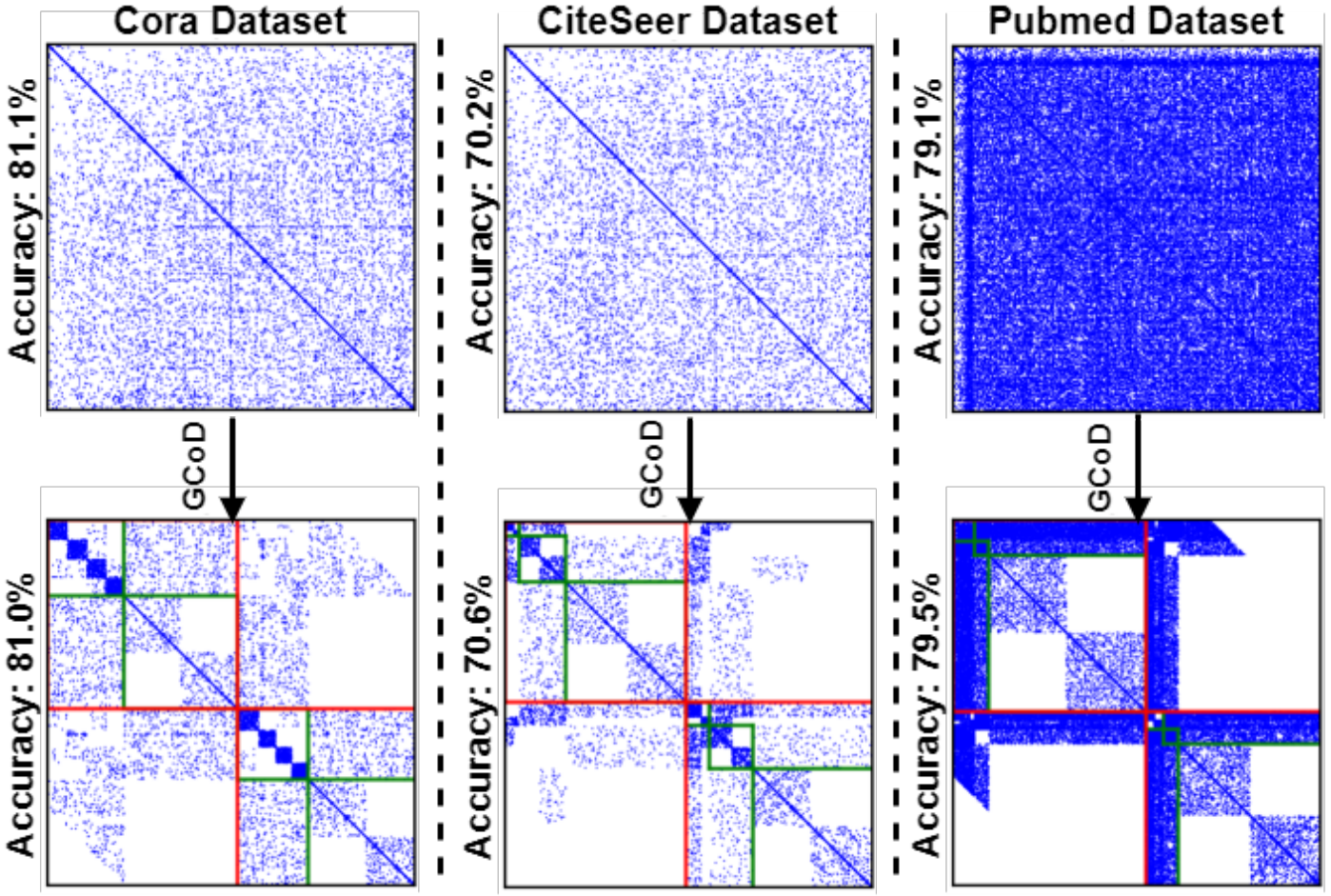
GCoD Algorithm: Split and Conquer Training

- **The proposed split and conquer training**
  - **Step 1: GCNs pretraining on the partitioned graphs**
    - Extract C subgraph classes according to the degrees
    - Use METIS to partition each class into workload balanced subgraphs
    - Pretrain GCNs on the partitioned graphs
  - **Step 2: Sparsify and diagonalize the adjacency matrices**
    - Sparse regularization term
    - Polarization regularization term
  - **Step 3: Structural sparsification**
    - Prune patches where the number of non-zeros is smaller than a threshold (ranging from 10 to 30 for different graphs)



GCoD Algorithm: Split and Conquer Training

- **The proposed split and conquer training**
  - **Results visualization**
    - Denser along diagonal line
    - Sparser along off-diagonal line



GCoD Accelerator: Two-pronged Acceleration

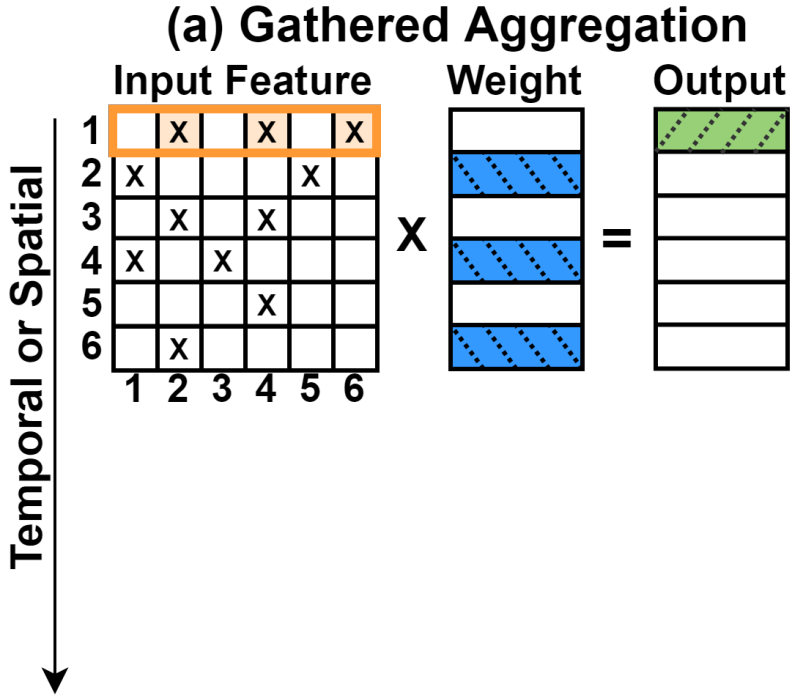
- ***Challenge: How to fully exploit the potential of such regularity?***

- **Design principles and considerations**
  - **A two-pronged accelerator**
    - Separately process sparser and denser branches (echo with GCoD algorithm)

GCoD Accelerator: Two-pronged Acceleration

- ***Challenge: How to fully exploit the potential of such regularity?***

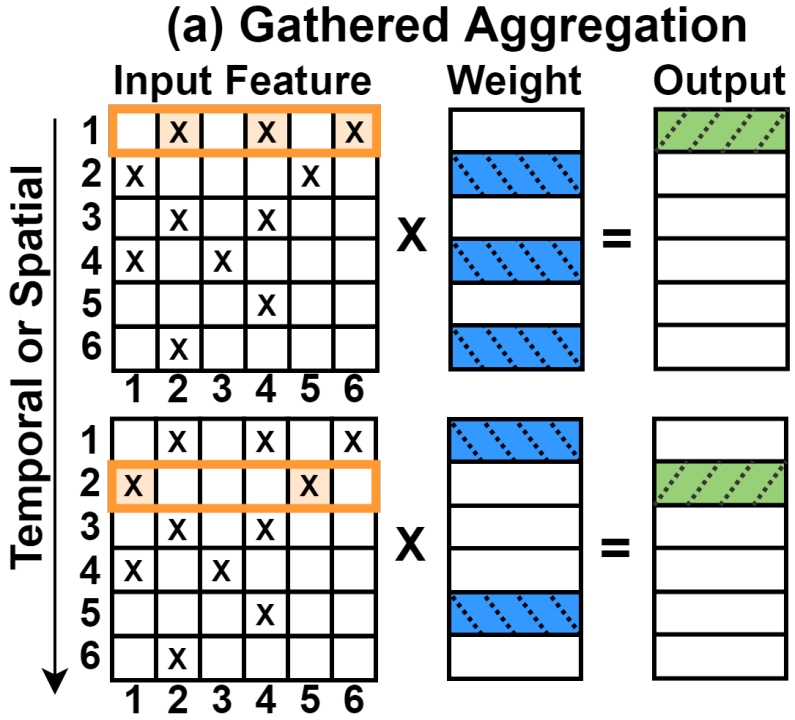
- **Design principles and considerations**
  - A two-pronged accelerator
  - **Gathered aggregation vs. Distributed aggregation**
    - Given that bandwidth is more limited when processing GCNs of poor data locality, the distributed aggregation better favors our design



GCoD Accelerator: Two-pronged Acceleration

- ***Challenge: How to fully exploit the potential of such regularity?***

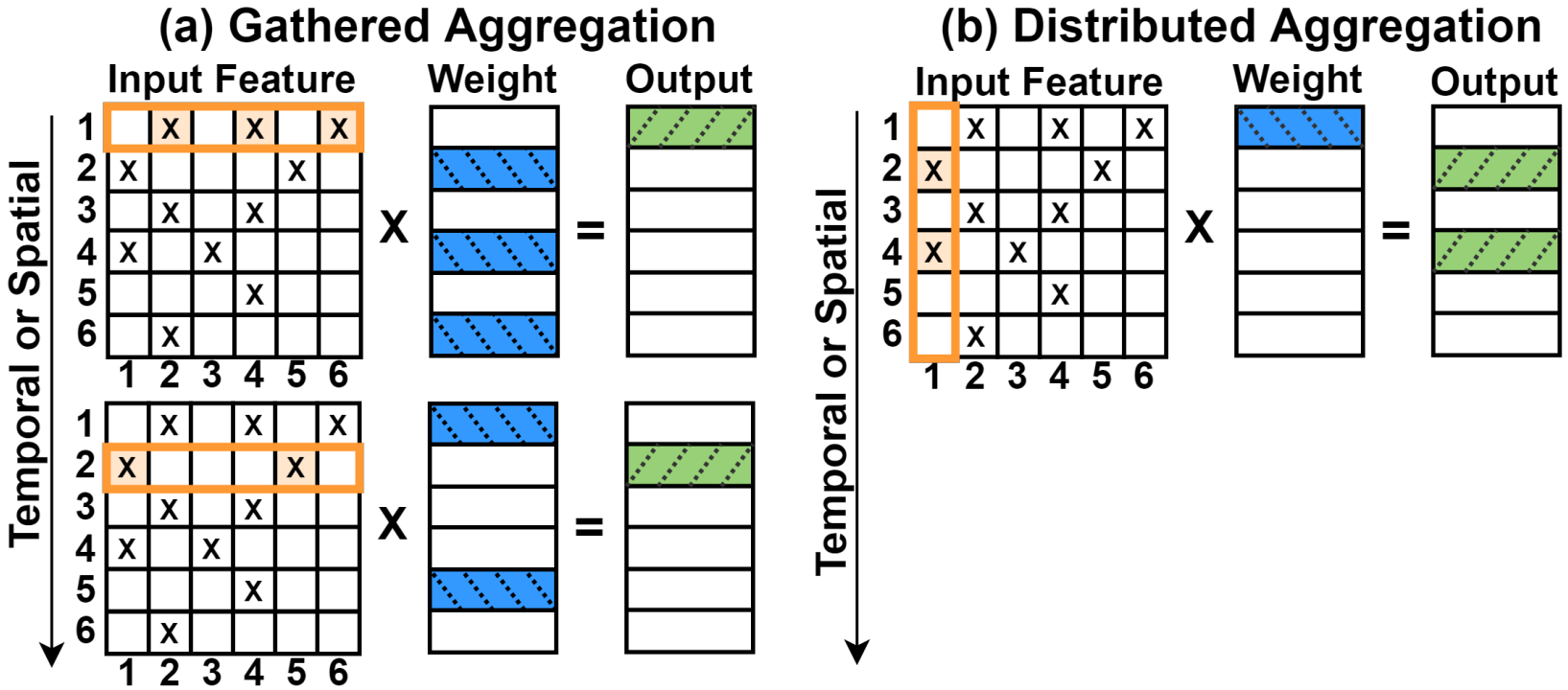
- **Design principles and considerations**
  - A two-pronged accelerator
  - **Gathered aggregation vs. Distributed aggregation**
    - Given that bandwidth is more limited when processing GCNs of poor data locality, the distributed aggregation better favors our design



GCoD Accelerator: Two-pronged Acceleration

- ***Challenge: How to fully exploit the potential of such regularity?***

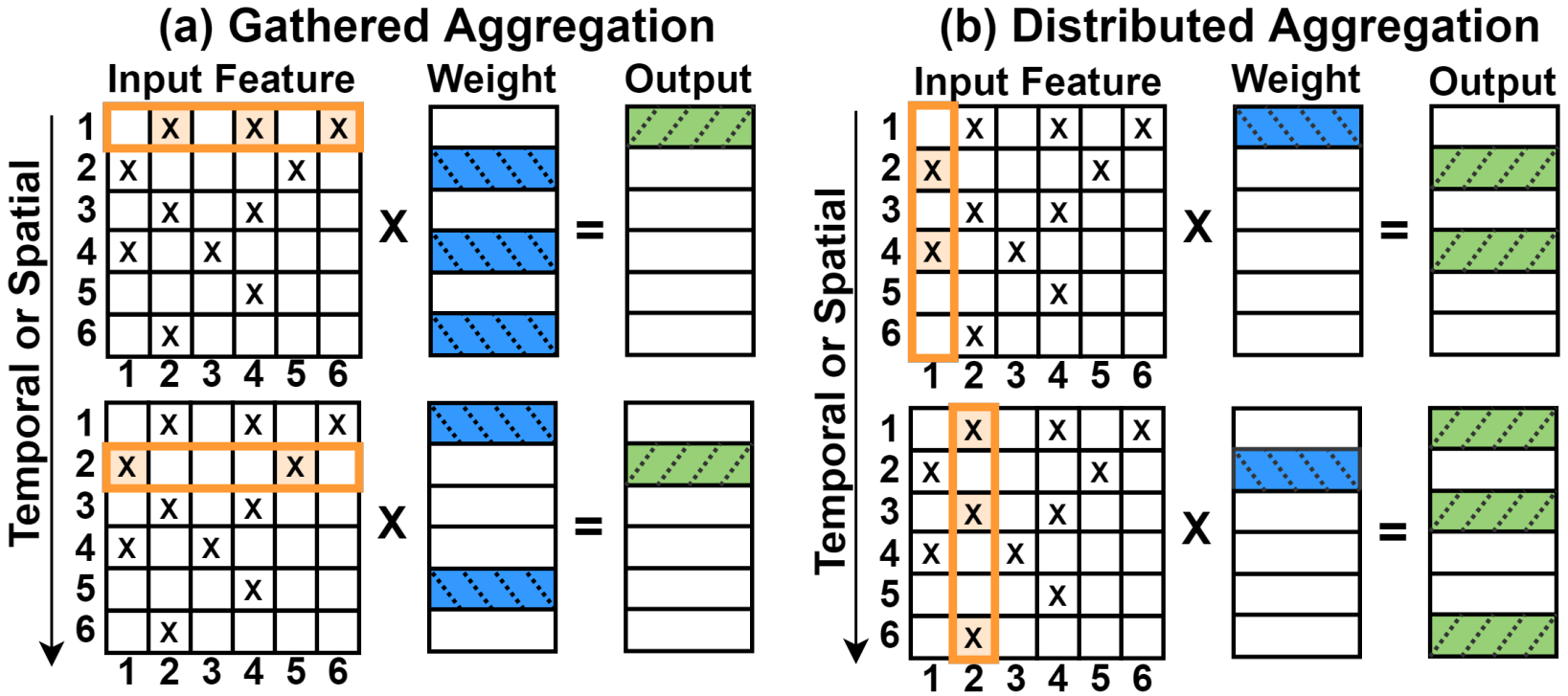
- **Design principles and considerations**
  - A two-pronged accelerator
  - **Gathered aggregation vs. Distributed aggregation**
    - Given that bandwidth is more limited when processing GCNs of poor data locality, the distributed aggregation better favors our design



GCoD Accelerator: Two-pronged Acceleration

- ***Challenge: How to fully exploit the potential of such regularity?***

- **Design principles and considerations**
  - A two-pronged accelerator
  - **Gathered aggregation vs. Distributed aggregation**
    - Given that bandwidth is more limited when processing GCNs of poor data locality, the distributed aggregation better favors our design



GCoD Accelerator: Two-pronged Acceleration

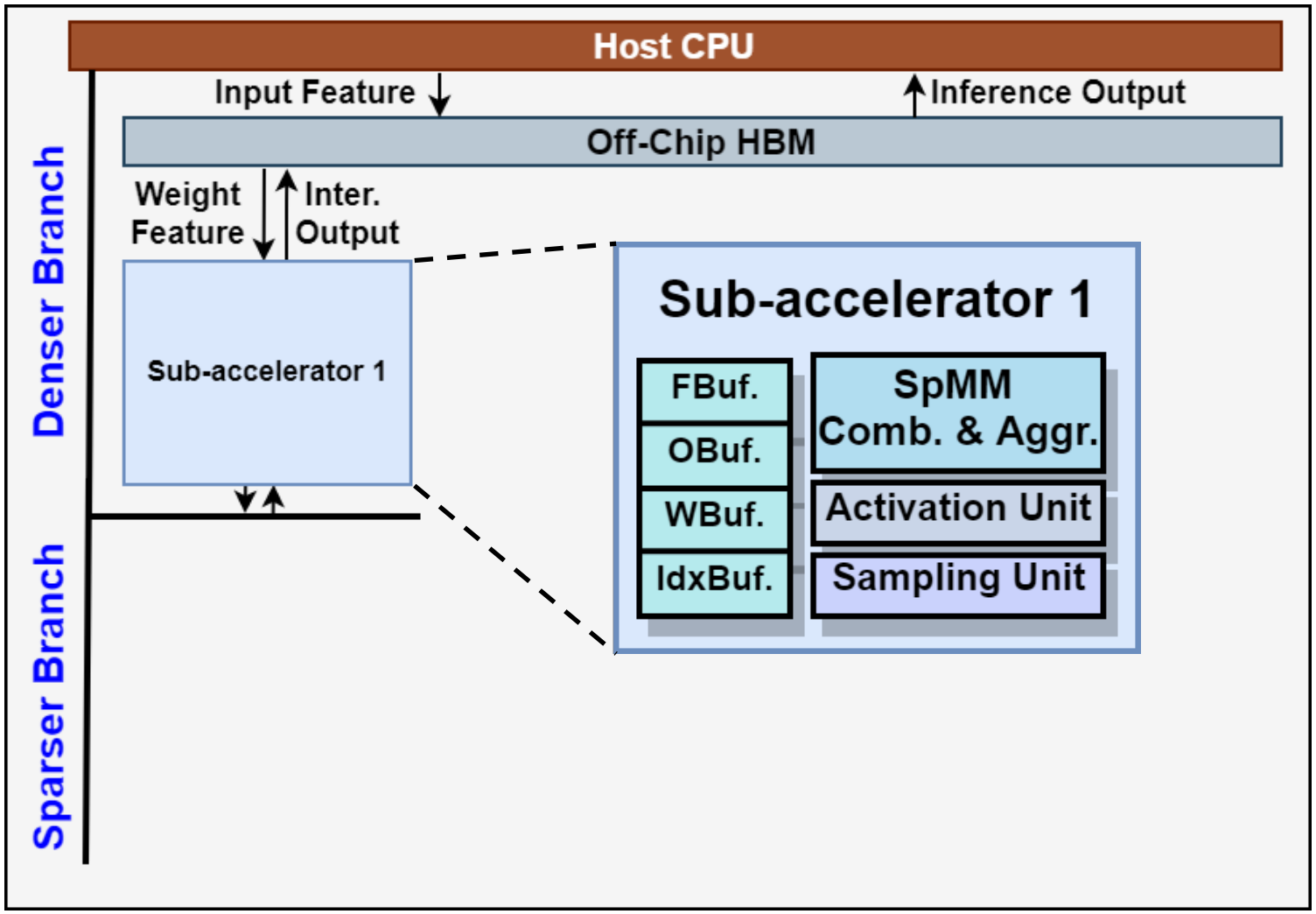
- ***Challenge: How to fully exploit the potential of such regularity?***

- **Design principles and considerations**
  - A two-pronged accelerator
  - Gathered aggregation vs. Distributed aggregation
  - **A unified architecture**
    - If using (1),
      - A * X is sparse-sparse matrix multiplication;
      - (A * X) * W is dense-dense matrix multiplication;
    - If using (2),
      - X * W is **sparse-dense** matrix multiplication (SpMM);
      - A * (X * W) is also **sparse-dense** matrix multiplication (SpMM).

(1) $Y = (A * X) * W$	(2) $Y = A * (X * W)$
First aggregation then combination	First combination then aggregation

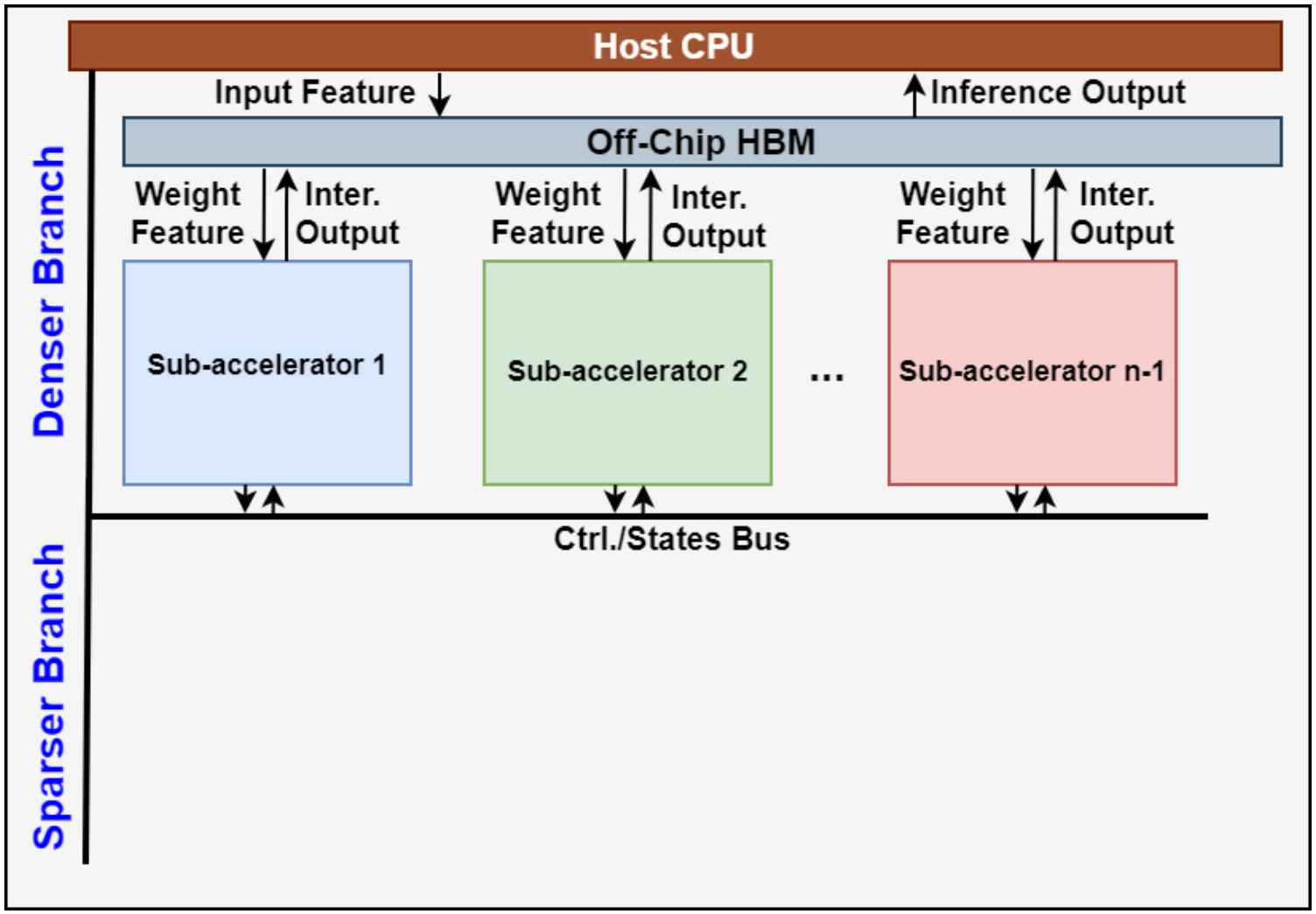
GCoD Accelerator: Two-pronged Acceleration

- **Micro-architecture designs**
  - **Denser branch:** Workload balance
  - **Sparser branch:** Reduced workload



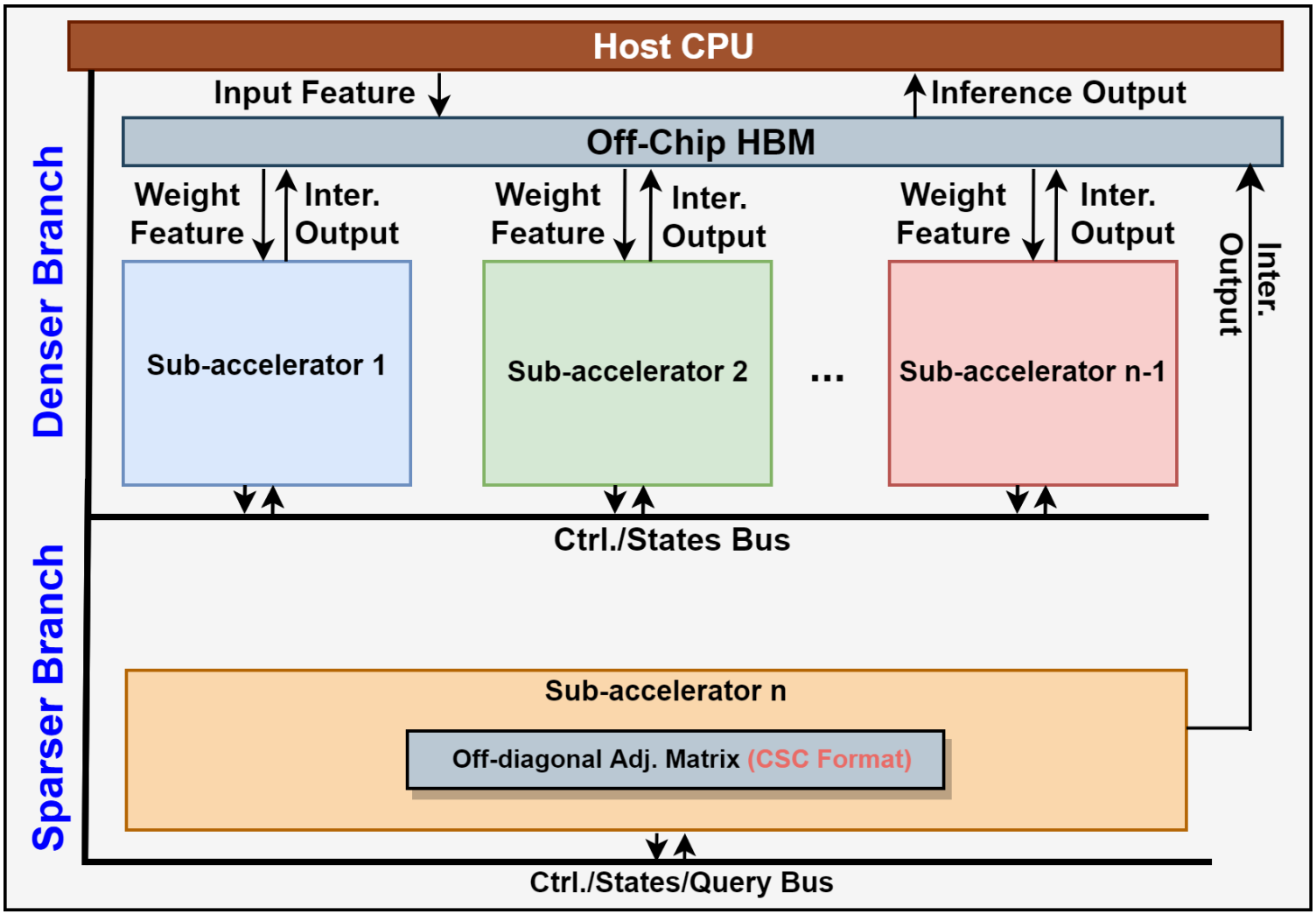
GCoD Accelerator: Two-pronged Acceleration

- **Micro-architecture designs**
  - **Denser branch:** Workload balance
  - **Sparser branch:** Reduced workload



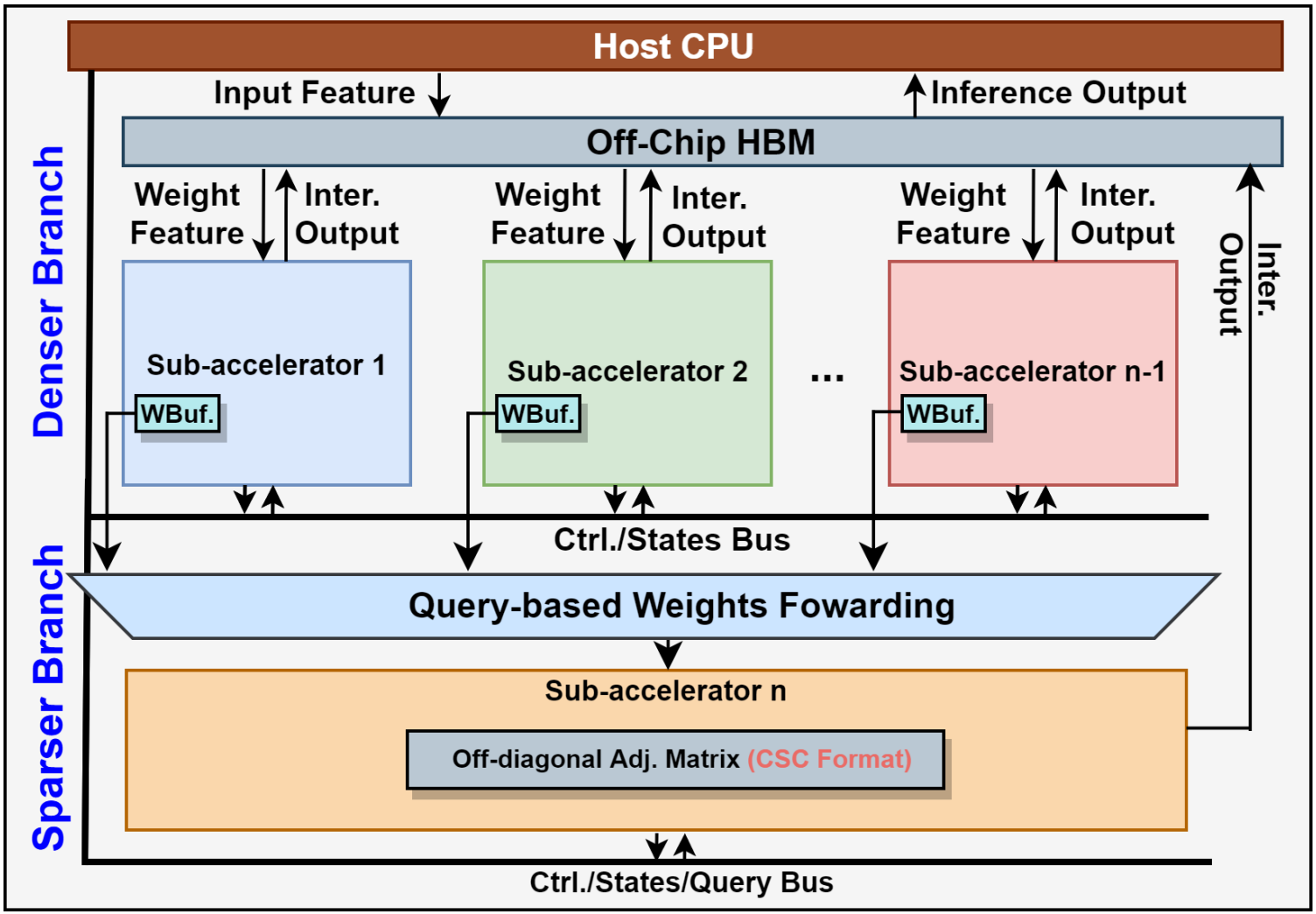
GCoD Accelerator: Two-pronged Acceleration

- **Micro-architecture designs**
  - **Denser branch:** Workload balance
  - **Sparser branch:** Reduced workload



GCoD Accelerator: Two-pronged Acceleration

- **Micro-architecture designs**
  - **Denser branch:** Workload balance
  - **Sparser branch:** Reduced workload



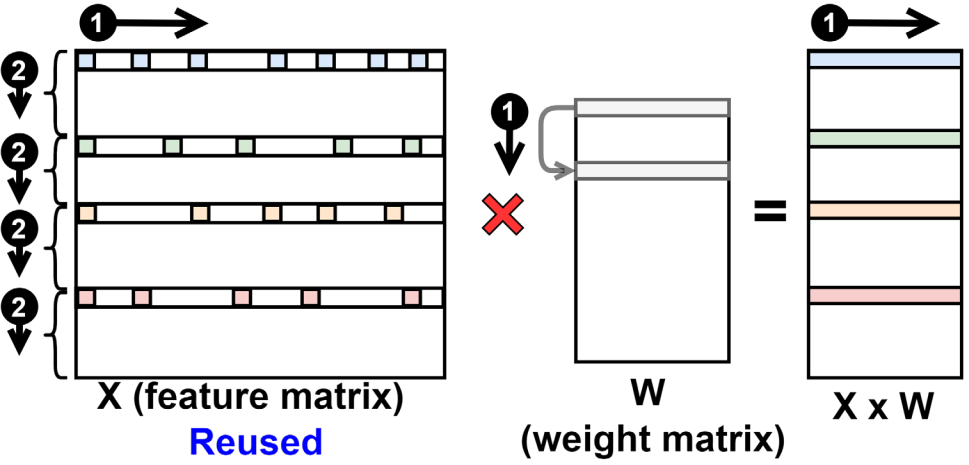
GCoD Accelerator: Two-pronged Acceleration

- **Micro-architecture designs**
  - **Denser branch:** Workload balance
  - **Sparser branch:** Reduced workload

	Multi-Chunks	On-chip Storage	Off-chip Access	Arch. Reuse	Data Reuse	Workloads
W/o GCoD	N	H	H	N	N	Heavy & Imbalanced
GCoD Denser	Y	L	L	Y	Y	**Balanced**
GCoD Sparser	N	H	L	Y	Y	**Light**

GCoD Accelerator: Fine-grained Pipeline

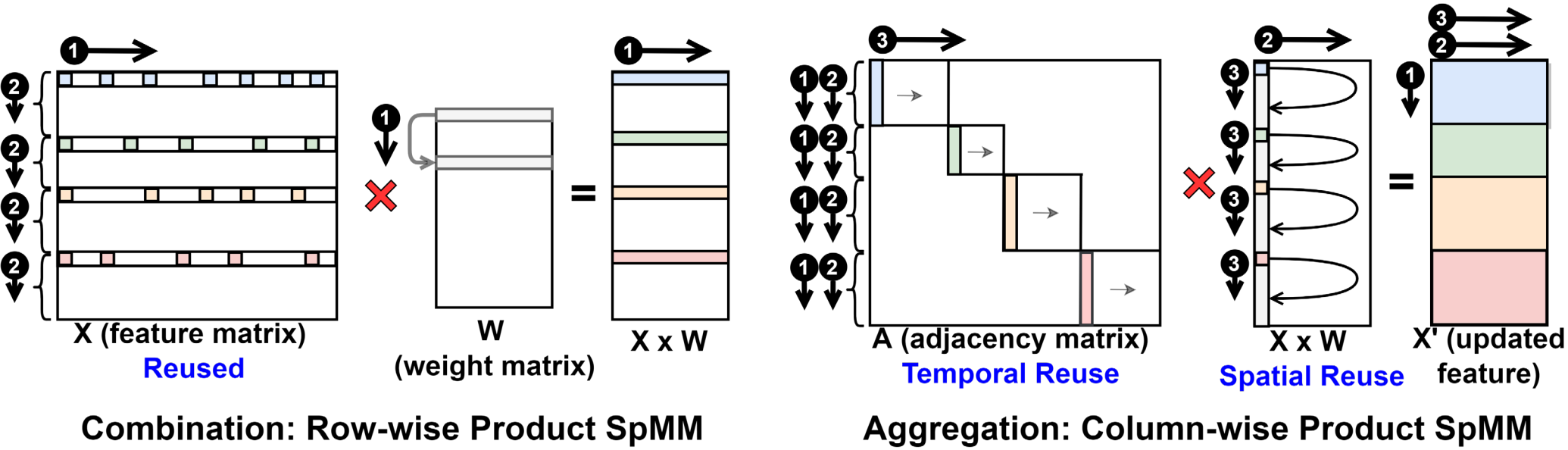
- **Fine-grained pipelines**
  - **Efficiency-aware pipeline**
    - Combination is row-wise; Aggregation is column-wise
    - Advantages:
      - Maximize the data reusability in X, XW, and A
    - Disadvantages:
      - Require a large on-chip output buffer



Combination: Row-wise Product SpMM

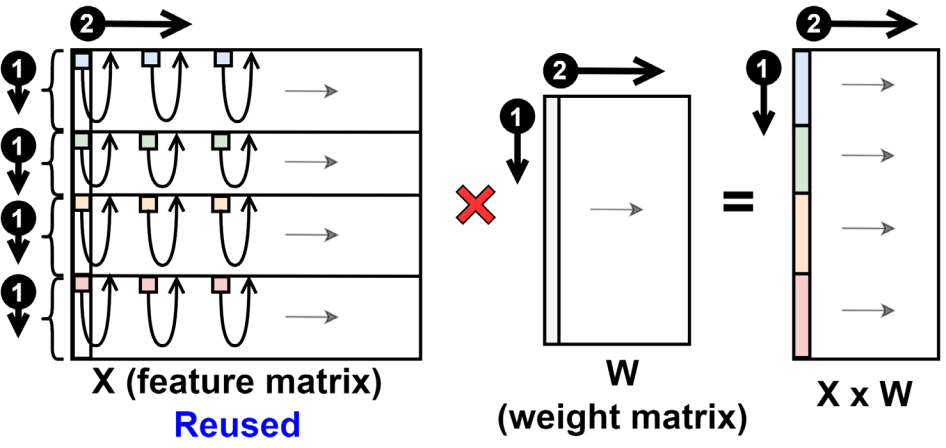
GCoD Accelerator: Fine-grained Pipeline

- **Fine-grained pipelines**
  - **Efficiency-aware pipeline**
    - Combination is row-wise; Aggregation is column-wise
    - Advantages:
      - Maximize the data reusability in X, XW, and A
    - Disadvantages:
      - Require a large on-chip output buffer



GCoD Accelerator: Fine-grained Pipeline

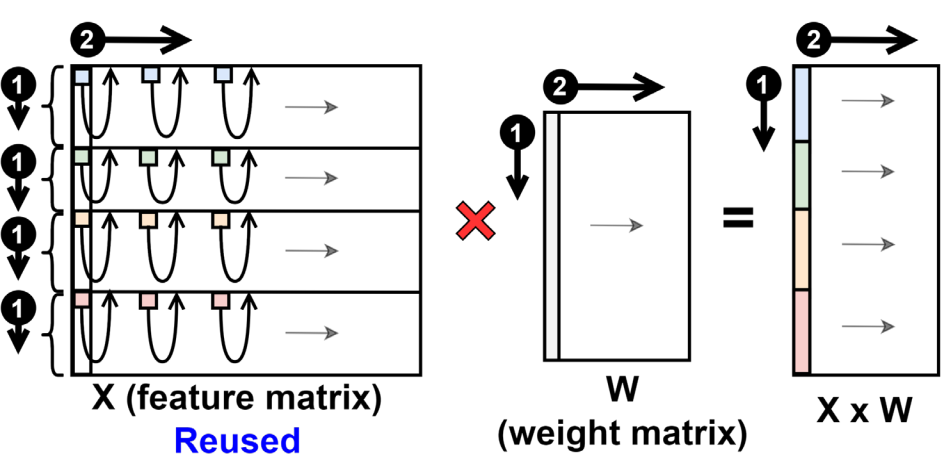
- **Fine-grained pipelines**
  - Efficiency-aware pipeline
  - **Resource-aware pipeline**
    - Combination is column-wise; Aggregation is column-wise
    - Advantages:
      - Only need to store one column of outputs
    - Disadvantages:
      - Need frequent accesses to the buffer storing A



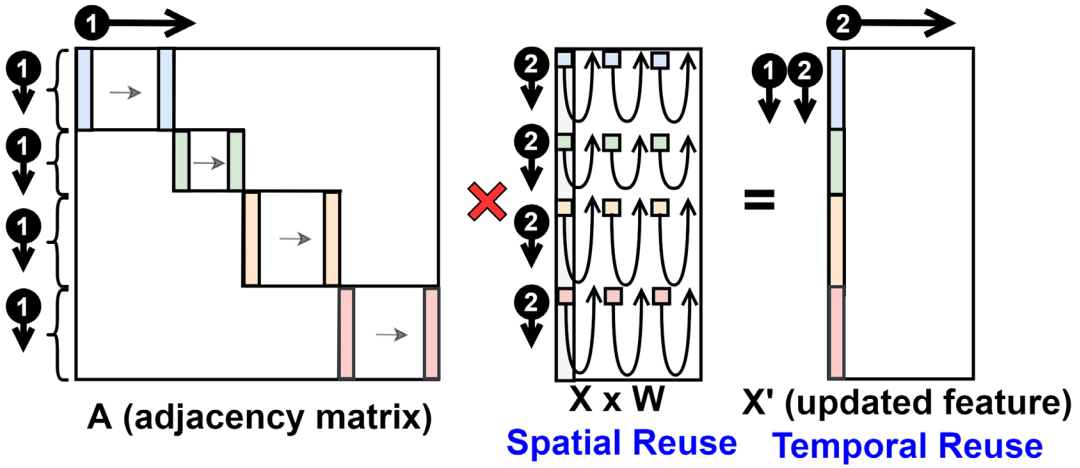
Combination: Column-wise Product SpMM

GCoD Accelerator: Fine-grained Pipeline

- **Fine-grained pipelines**
  - Efficiency-aware pipeline
  - **Resource-aware pipeline**
    - Combination is column-wise; Aggregation is column-wise
    - Advantages:
      - Only need to store one column of outputs
    - Disadvantages:
      - Need frequent accesses to the buffer storing A



Combination: Column-wise Product SpMM



Aggregation: Column-wise Product SpMM

GCoD Accelerator: Fine-grained Pipeline

- **Fine-grained pipelines**
  - **Efficiency-aware pipeline**
    - Advantages:
      - Maximize the data reusability in X, XW, and A
    - Disadvantages:
      - Require a large on-chip output buffer
  - **Resource-aware pipeline**
    - Advantages:
      - Only need to store one column of outputs
    - Disadvantages:
      - Need frequent accesses to the buffer storing A

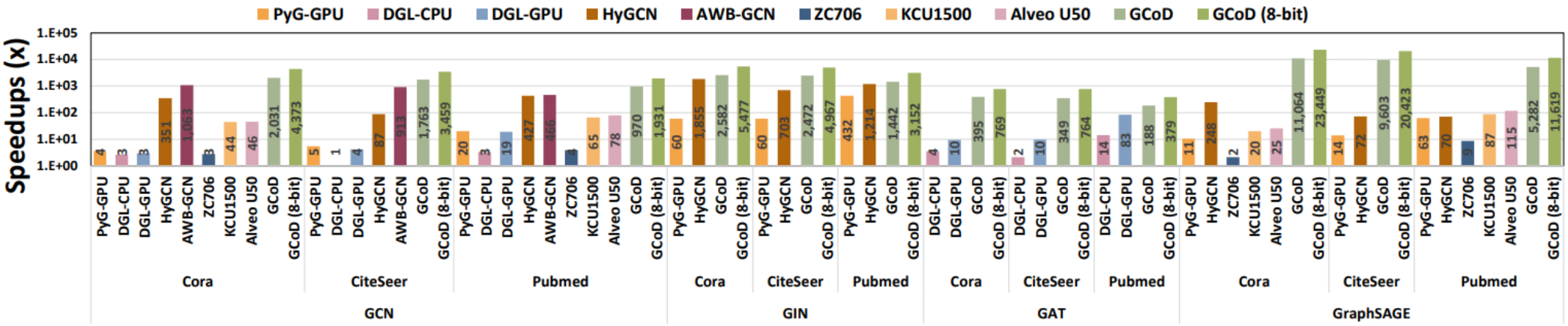
	Comb. SpMM	Aggr. SpMM	On-chip Storage	Off-chip Access	Data Reuse	Fit for Graphs
Efficiency-aware	RW	CW	H	L	X; XW; A	**Medium**
Resource-aware	CW	CW	L	L	X; XW; X’	**Large**

Evaluation Setup and Baselines

- **Evaluation Setup**
  - **Five GCN Models:**
    - **GCN, GraphSAGE, GAT, GIN, 28-layer ResGCNs**
  - **Six Datasets:**
    - **Cora, CiteSeer, Pubmed, NELL, Reddit, Ogbn-ArXiv, Ogbl-Collab**
  - **Metrics:**
    - **Node classification accuracy, Latency speedups**

- **Benchmark Baselines**
  - **Accelerator Baselines**
    - **PyG-CPU/GPU, HyGCN, AWB-GCN, Deepburning-GL on three FPGAs (i.e., ZC706, KCU1500, Alveo U50)**
  - **Algorithm Baselines**
    - **Random pruning, SGCN, QAT, Degree-Quant**

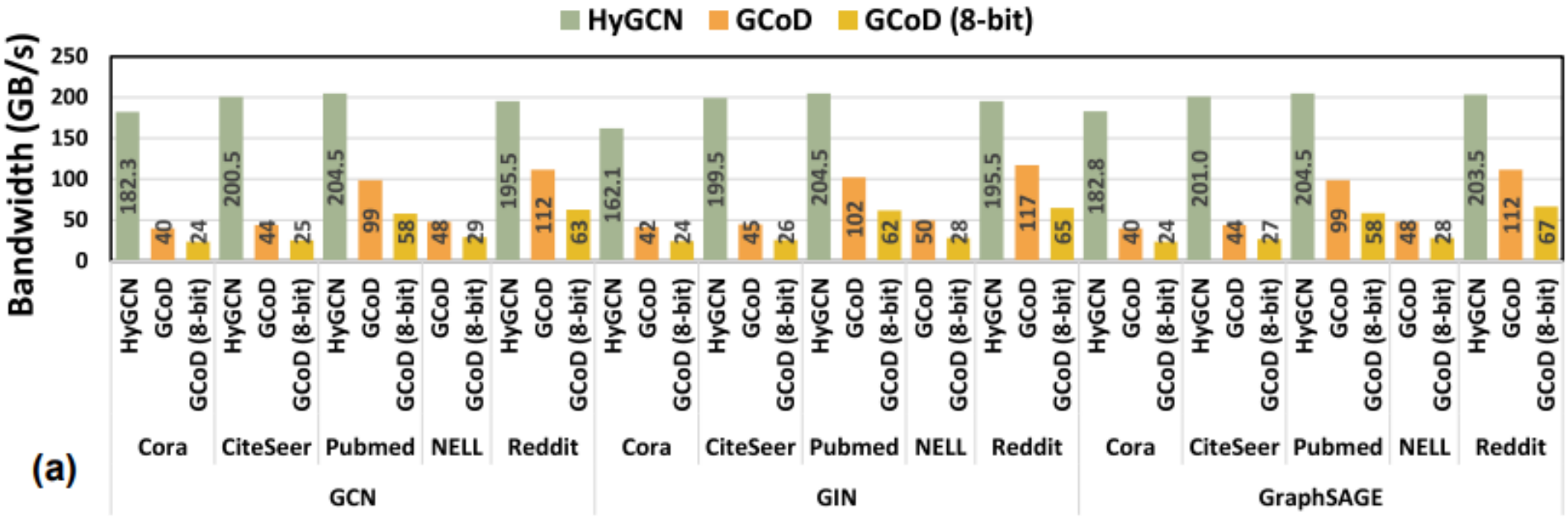
Evaluation: GCoD over SOTA GCN Accelerators



- **GCoD over CPU/GPU platforms**
  - GCoD achieves up to 15286x and 294x speedups over PyG-CPU and PyG-GPU
  - GCoD (8-bit) more aggressively achieve 32158x and 607x speedups over PyG-CPU and PyG-GPU

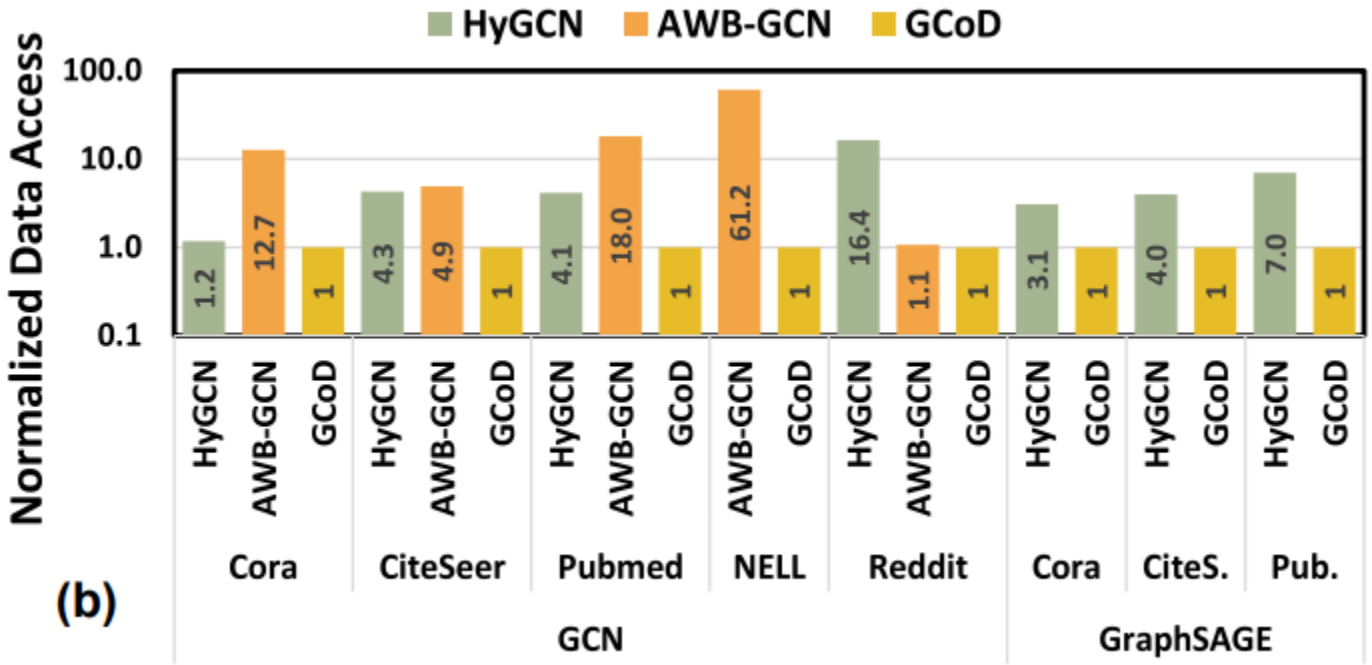
- **GCoD over SOTA GCN accelerators**
  - GCoD achieves on average 7.8x, 2.5x, 2532x, 165x, and 115x speedups over HyGCN, AWB-GCN, ZC706, KCU1500, Alveo U50, respectively.

Evaluation: GCoD over SOTA GCN Accelerators



- **GCoD over SOTA GCN accelerators**
  - Off-chip memory Bandwidth Consumption
    - GCoD and GCoD (8-bit) only require on average 48% and 26% off-chip memory bandwidth as compared to HyGCN, respectively

Evaluation: GCoD over SOTA GCN Accelerators



- **GCoD over SOTA GCN accelerators**
  - Off-chip Memory Access
    - GCoD requires much less (as low as 16.4x and 61.2x) off-chip memory accesses as compared to HyGCN and AWB-GCN

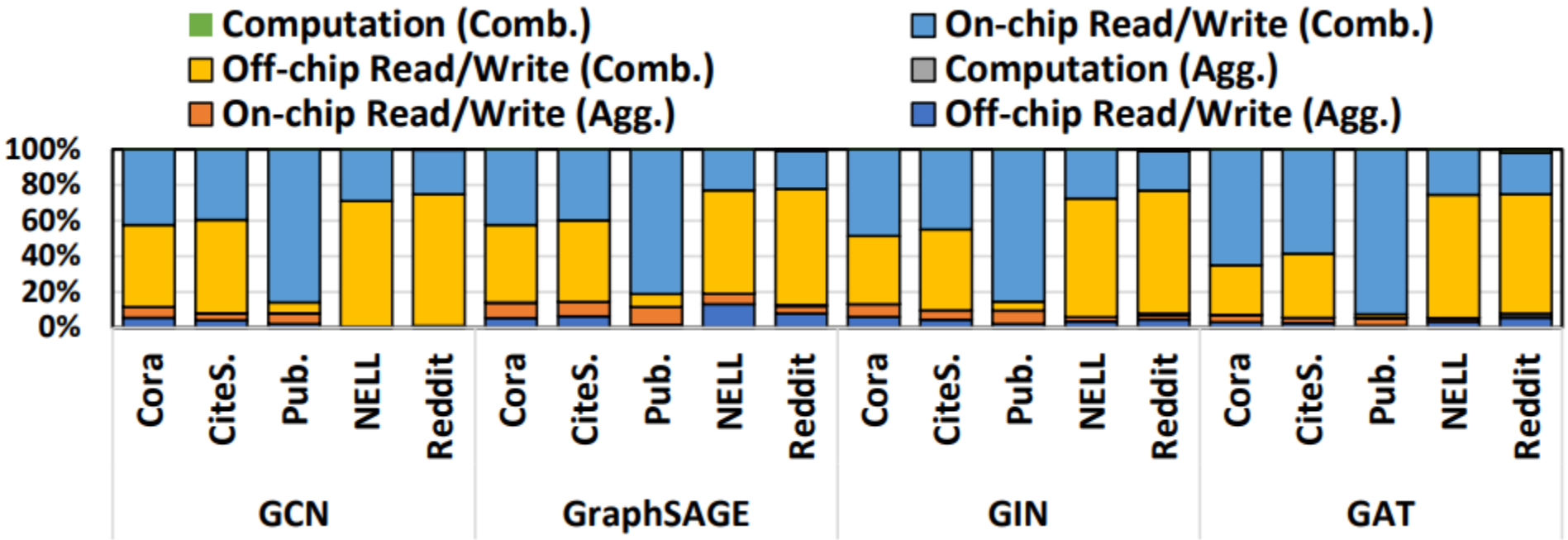
Evaluation: GCoD Algorithm over SOTA Baselines

Models	Methods	Accuracy (%)				
		Cora	CiteSeer	Pubmed	NELL	Reddit
GCN	Vanilla	81.1±1.2	70.2±0.8	79.1±0.6	65.6±0.7	92.2±1.1
	RP	79.6±0.8	70.4±0.5	78.4±0.6	63.5±2.3	91.2±2.2
	SGCN	80.2±0.7	70.4±0.7	79.1±0.1	64.2±1.2	91.3±1.3
	QAT	81.0±0.7	71.3±1.0	79.0±0.2	65.1±1.4	92.4±0.9
	Degree-Quant	81.7±0.7	71.0±0.9	79.1±0.1	65.2±0.8	92.6±1.5
	GCoD	**81.9±0.8**	**71.7±0.5**	79.5±0.3	**66.3±0.5**	**93.4±0.9**
	GCoD (8-bit)	81.0±0.9	70.6±0.3	**79.5±0.2**	66.0±0.3	93.2±1.3
	GCoD Improv.	**↑0.2% ~ ↑2.8%**				
GAT	Vanilla	83.1±0.4	72.2±0.7	78.8±0.3	66.6±0.3	94.2±0.3
	RP	80.9±0.6	69.8±0.8	78.2±0.1	64.5±1.2	93.1±1.2
	SGCN	81.9±0.3	71.9±0.2	78.4±0.1	64.9±1.0	93.4±0.9
	QAT	81.9±0.7	71.2±1.0	78.3±0.5	65.1±0.8	93.8±0.5
	Degree-Quant	82.7±0.7	71.6±1.0	78.6±0.3	65.9±0.6	94.0±0.7
	GCoD	**83.2±0.3**	**72.2±0.4**	**79.0±0.1**	**66.7±0.4**	94.5±0.2
	GCoD (8-bit)	82.6±0.2	71.8±0.1	78.8±0.2	66.5±0.2	**94.5±0.4**
	GCoD Improv.	**↑0.1% ~ ↑2.2%**				
GIN	Vanilla	78.6±0.9	67.5±1.5	78.5±0.2	65.2±0.2	92.8±2.2
	RP	74.6±0.4	64.5±0.5	76.9±0.6	64.2±0.5	92.0±0.5
	SGCN	78.0±0.1	67.0±0.1	77.2±1.1	64.8±0.4	92.3±0.9
	QAT	75.6±1.2	63.0±2.6	77.5±0.2	64.7±0.3	92.9±0.4
	Degree-Quant	78.7±1.4	67.5±1.4	78.1±0.5	65.2±0.3	93.1±0.6
	GCoD	**78.9±0.5**	68.6±0.8	**78.5±0.3**	**65.8±0.2**	93.3±0.9
	GCoD (8-bit)	78.4±0.2	**68.7±1.3**	78.3±0.2	65.6±0.4	**93.3±1.1**
	GCoD Improv.	**↑0.3% ~ ↑4.2%**				
GraphSAGE	Vanilla	81.2±0.2	71.1±0.3	78.7±0.2	66.2±0.4	93.8±2.9
	RP	77.7±0.7	66.1±2.2	76.0±0.3	65.5±0.4	90.7±1.8
	SGCN	79.2±0.5	70.9±0.3	78.5±0.1	66.1±0.3	93.9±0.9
	GCoD	**81.4±0.6**	71.3±0.3	**79.0±0.4**	**66.7±0.6**	**94.5±1.2**
	GCoD (8-bit)	80.8±0.4	**71.3±0.1**	78.8±0.1	66.4±0.8	94.3±1.5
	GCoD Improv.	**↑0.2% ~ ↑3.8%**				

GCoD over SOTA Compression Baselines

- GCoD consistently achieve comparable or even better accuracy (↑0.2% ~ ↑4.2%) over standard GCNs and all compression baselines, while leading to 5% ~ 15% structural sparsity and workload balanced subgraph partitions

Evaluation of GCoD Accelerators



- **Energy Breakdown of GCoD**
  - The combination phase consumes most of the energy than the aggregation phase after GCoD acceleration
    (vs. PyG-CPU on which aggregation occupies 80% ~ 99%)
  - The energy costs of accessing off-chip memory remain reasonable along with the growing of graph sizes → good scalability

Summary

For the first time, we

- **Propose a GCN algorithm & accelerator co-design framework, dubbed as GCoD**
- **On the algorithm level, GCoD integrates a split and conquer training to polarize the graphs to be either denser or sparser without compromising the accuracy**
- **On the hardware level, GCoD further develop a dedicated two-pronged accelerator with separated engines to process each of enforced workloads**

Acknowledge: NSF RTML & NeTS program