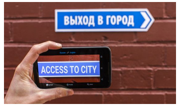
GReTA: Hardware Optimized Graph Processing for GNNs

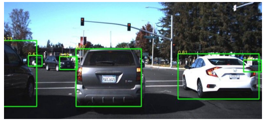
Kevin Kiningham, Phil Levis, Chris Ré
Stanford University
March 4th, 2020

Deep Neural Networks

Speech Recognition



Translation



Object Detection

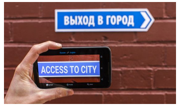
Handwriting Recognition

Traditional DNN

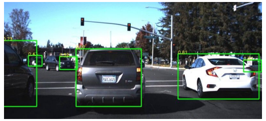
Conv Layer → … → Linear Layer → "Dog"

Deep Neural Networks + Graphs = ?


Speech Recognition

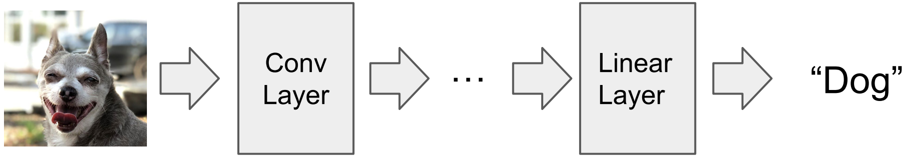


Translation

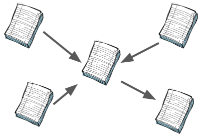

Object Detection

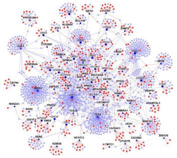

Handwriting Recognition

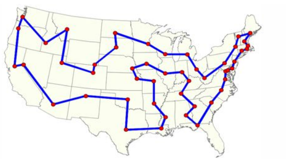
Traditional DNN




Social Networks


Citations

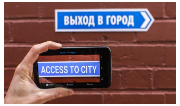

Protein Interactions


Road Networks

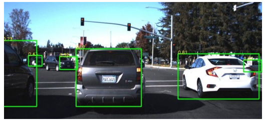
?

Deep Neural Networks + Graphs = GNNs

Speech Recognition



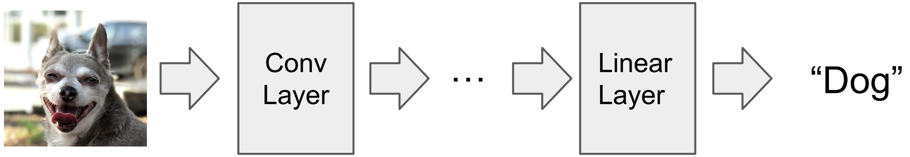
Translation

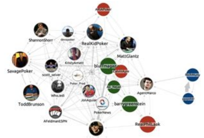


Object Detection

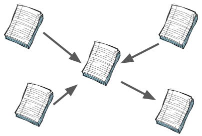
Handwriting Recognition

Traditional DNN

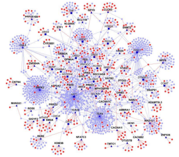




Social Networks

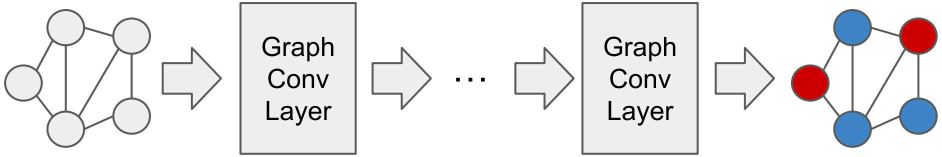


Citations



Protein Interactions

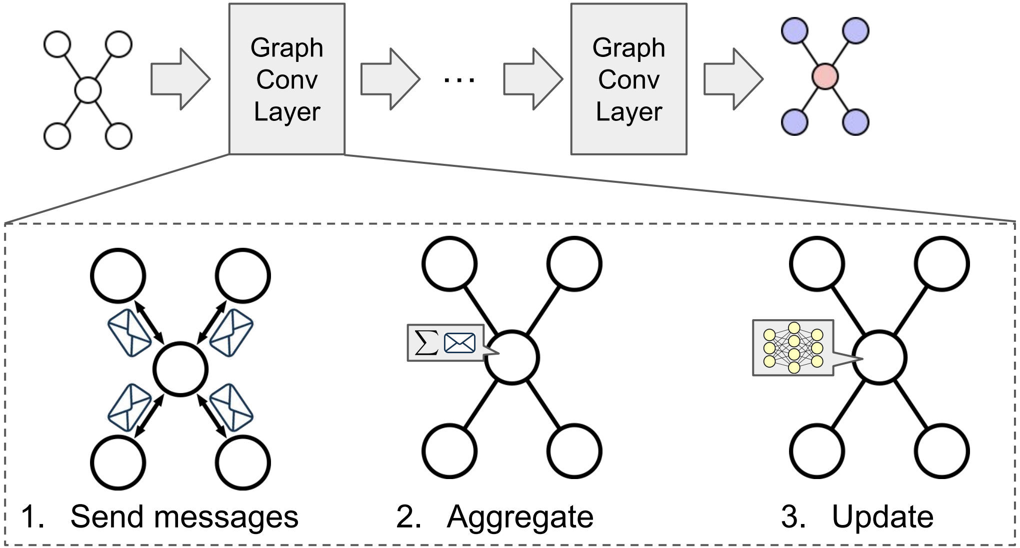
Road Networks



Graph Neural Network (GNN)

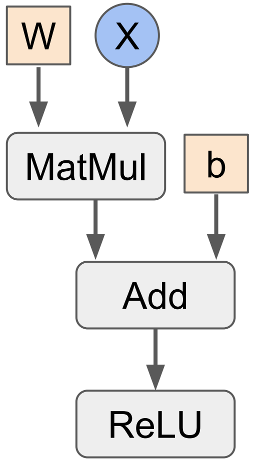
GNN Computation Is Irregular

- Computation pattern ***changes*** depending on input graph structure
- GNN layers follow message passing architecture

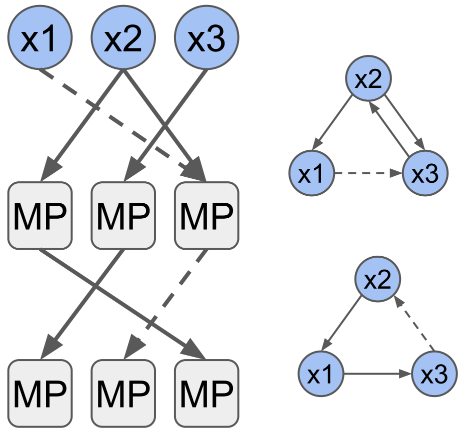


Existing DNN Representations Bad for GNNs

- Irregular computation is difficult to represent with static tensor network
  - E.g. Tensorflow
- Hard to handle large graphs
  - Must manually deal with partitioning variables
  - Hard to make efficient when graph shape can change



Static Network



Graph Neural Network

GReTA: Graph Framework for GNNs

- **Simple** to represent GNN layers
  - Computation defined on edges and vertices of input graph
  - Maps directly to message passing
- **Flexible** enough to allow a wide range of GNN models
  - Allows each execution phase to be customized
- **Efficient** execution on an accelerator
  - Partitioning: Limit accelerator memory usage without modifying user code
  - Tiling: Increase the reuse of layer weights

Talk Agenda

- Introduction
- **GReTA Overview**
- Execution Model
- Partitioning
- Experimental Results
- Conclusion

GReTA Overview

- GReTA represents computation using **graph framework**
  - Functions defined on edges and vertices
  - Can directly map message passing layer
- GNN layers implemented using four user-defined functions (UDFs)
  1. **G**ather: compute message for each edges
  2. **Re**duce: reduce incoming messages per-vertex
  3. **T**ransform: combine reduced value with per-vertex accumulator
  4. **A**ctivate: perform non-linear function

Example: Graph Convolutional Network (GCN)

$$h_v^{(\ell+1)} \leftarrow \mathrm{ReLU}\left(W^{(\ell)} \cdot \left(\sum_{u \to v} h_u\right) + b^{(\ell)}\right)$$

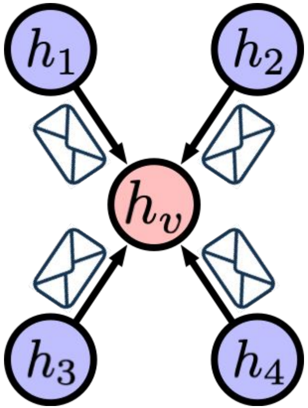
GCN layer update function

Example: Graph Convolutional Network (GCN)

$$h_v^{(\ell+1)} \leftarrow \mathrm{ReLU}\left(W^{(\ell)} \cdot \left(\sum_{u\to v} \boxed{h_u}\right) + b^{(\ell)}\right)$$

GCN layer update function

1. **Gather** messages using connected edges

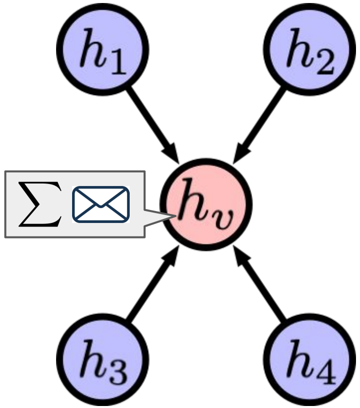


Example: Graph Convolutional Network (GCN)

$$h_v^{(\ell+1)} \leftarrow \mathrm{ReLU}\left(W^{(\ell)} \cdot \left(\boxed{\sum_{u \to v} \boxed{h_u}}\right) + b^{(\ell)}\right)$$

GCN layer update function

1. **Gather** messages using connected edges
2. **Reduce** to single vector by summation

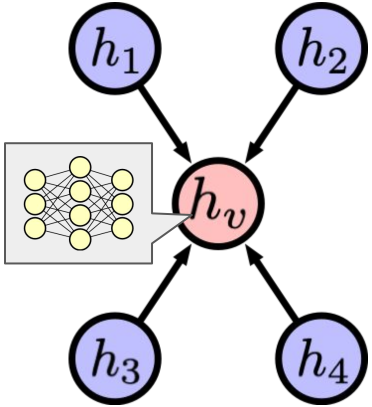


Example: Graph Convolutional Network (GCN)

$$h_v^{(\ell+1)} \leftarrow \mathrm{ReLU}\left(W^{(\ell)} \cdot \left(\sum_{u \to v} h_u\right) + b^{(\ell)}\right)$$

GCN layer update function

1. **Gather** messages using connected edges
2. **Reduce** to single vector by summation
3. **Transform** result using linear transformation

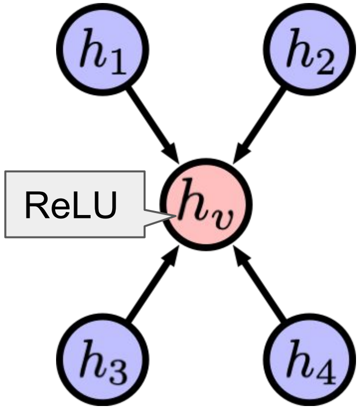


Example: Graph Convolutional Network (GCN)

$$h_v^{(\ell+1)} \leftarrow \mathrm{ReLU}\left(W^{(\ell)} \cdot \left(\sum_{u\to v} h_u\right) + b^{(\ell)}\right)$$

GCN layer update function

1. **Gather** messages using connected edges
2. **Reduce** to single vector by summation
3. **Transform** result using linear transformation
4. **Activate** output using element-wise ReLU



GCN Implementation Pseudocode

$$h_v^{(\ell+1)} \leftarrow \boxed{\mathrm{ReLU}\left(\boxed{W^{(\ell)} \cdot \left(\boxed{\sum_{u \to v} \boxed{h_u}}\right) + b^{(\ell)}}\right)}$$

GCN layer update function

```
class GCNLayer(GretaInterface):
  def gather(h_u, h_v, h_uv):
    return h_u

  def reduce(a_v, m_v):
    return a_v + m_v

  def transform(z_v, a_v, W, b):
    return z_v + W * a_v + b

  def activate(z_v):
    return relu(z_v)
```

Talk Agenda

- Introduction
- GReTA Overview
- **Execution Model**
- Partitioning
- Experimental Results
- Conclusion

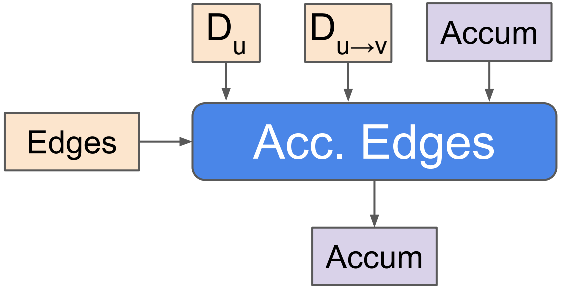
GReTA Execution Model

Execution conceptually split into three phases

GReTA Execution Model

Execution conceptually split into three phases

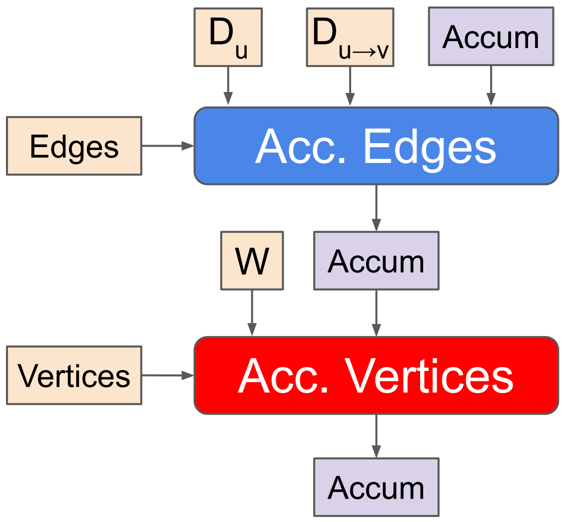
1. **Accumulate Edges**
   - Gather/compute message for each edge
   - Reduce to single value per vertex



GReTA Execution Model

Execution conceptually split into three phases

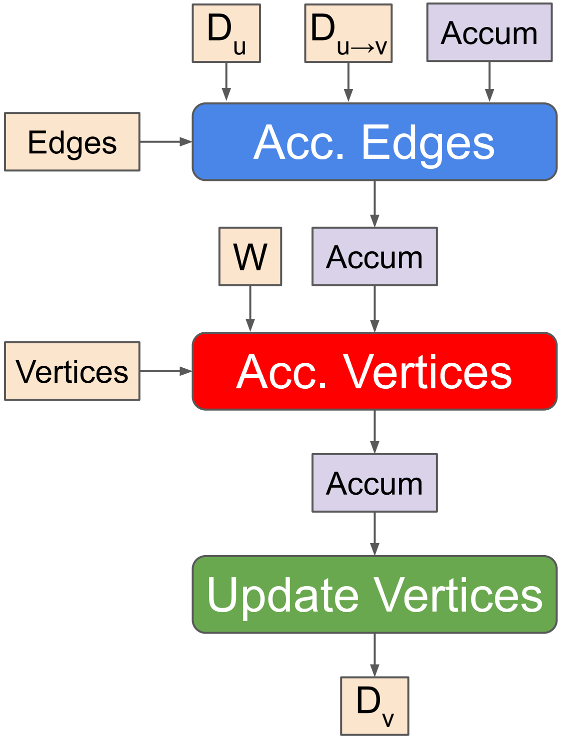
1. **Accumulate Edges**
   - Gather/compute message for each edge
   - Reduce to single value per vertex
2. **Accumulate Vertices**
   - Combine reduced value with prior vertex accumulator state



GReTA Execution Model

Execution conceptually split into three phases

1. **Accumulate Edges**
   - Gather/compute message for each edge
   - Reduce to single value per vertex
2. **Accumulate Vertices**
   - Combine reduced value with prior vertex accumulator state
3. **Update Vertices**
   - Apply activate to accumulator



Talk Agenda

- Introduction
- GReTA Overview
- Execution Model
- **Partitioning**
- Experimental Results
- Conclusion

Optimizations for Hardware Implementation

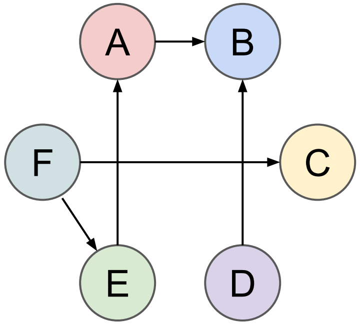
Execution Partitioning

- Problem: Large graphs do not fit into limited accelerator memory
    - E.g. social media graphs with millions of users
- Solution: Partition graph and execute GReTA on each partition separately
- Results combined via vertex accumulators

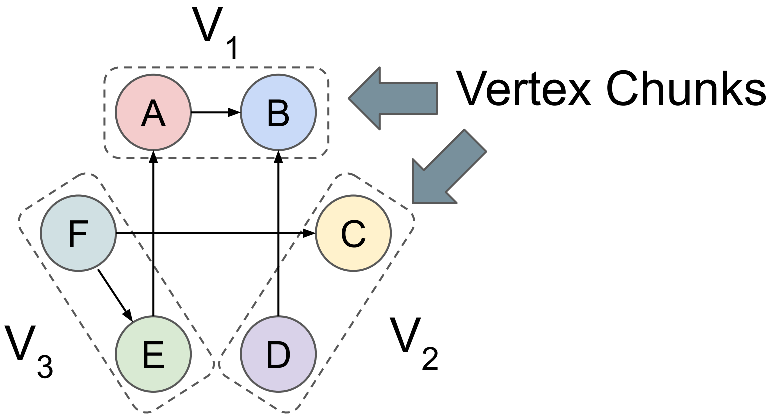
Weight Tiling

- Problem: Bandwidth bottlenecks when layer weights are large
- Solution: Improve reuse by splitting weights into tiles
- Tiles can be reused across multiple vertices

Graph Partitioning Example

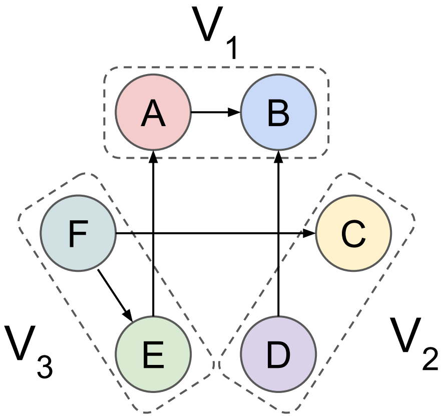


Graph Partitioning Example

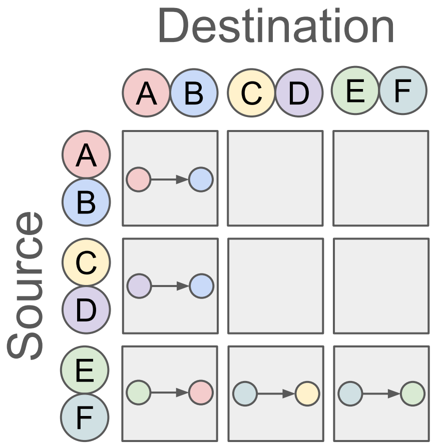


Vertex Partition

Graph Partitioning Example

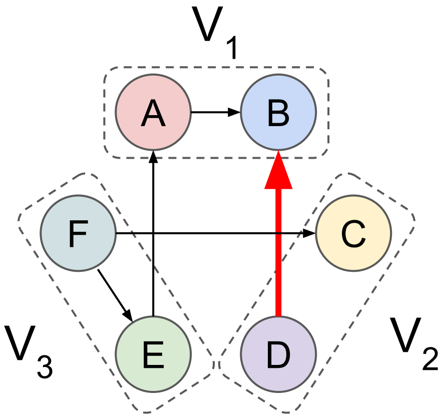


Vertex Partition

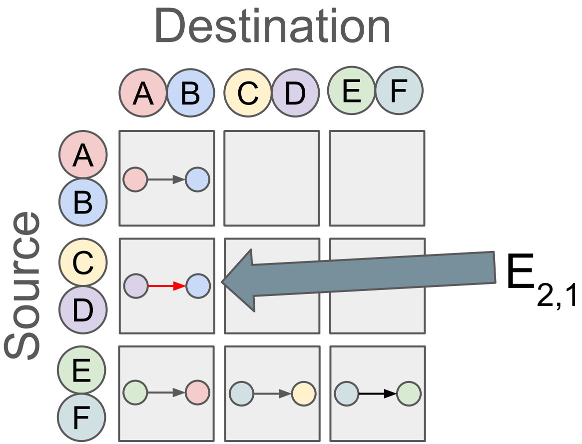


Edge Partition

Graph Partitioning Example

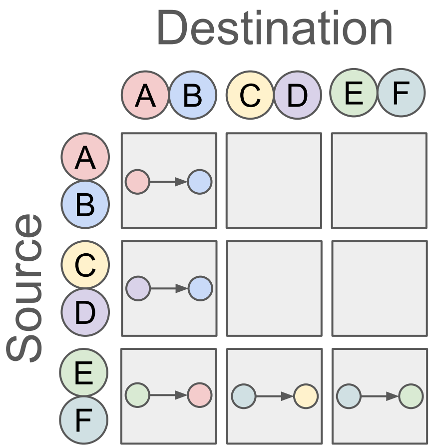


Vertex Partition

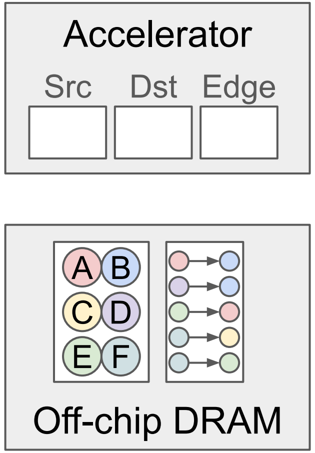


Edge Partition

Execution Partitioning Example

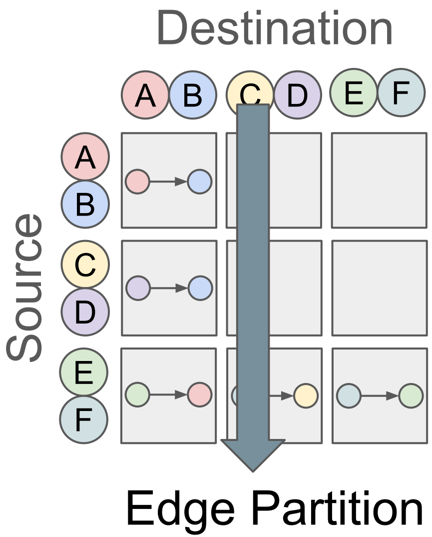


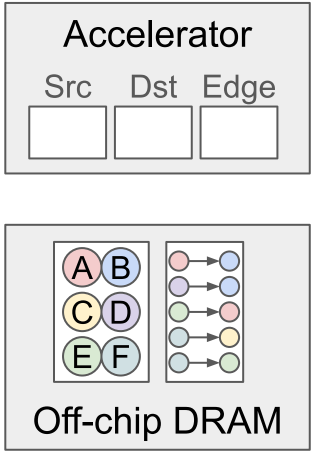
Edge Partition



Execution Partitioning Example

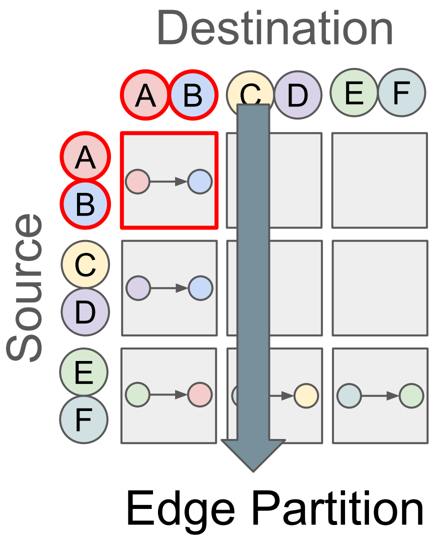
Execution follows columns

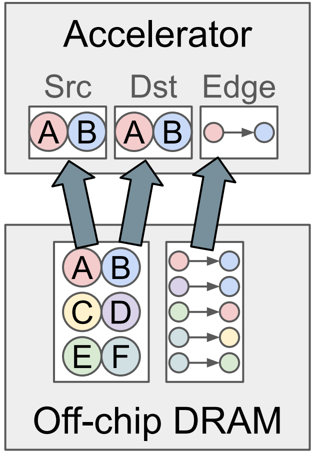


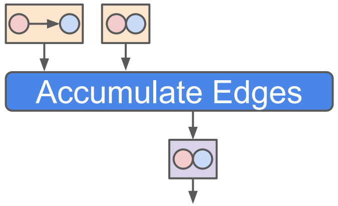


Execution Partitioning Example

Execution follows columns

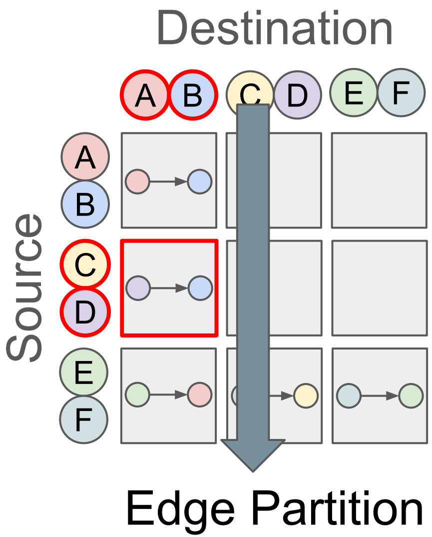


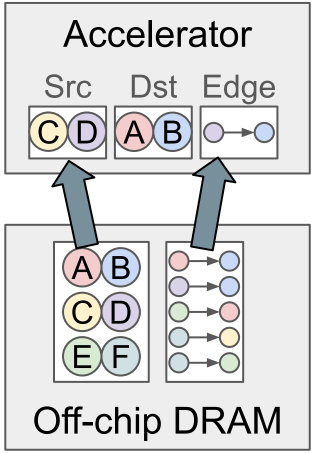


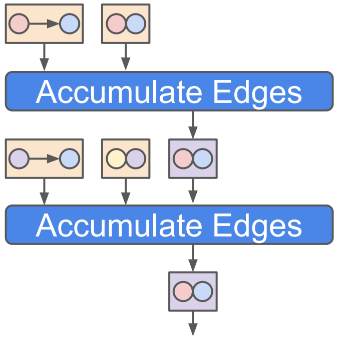


Execution Partitioning Example

Execution follows columns

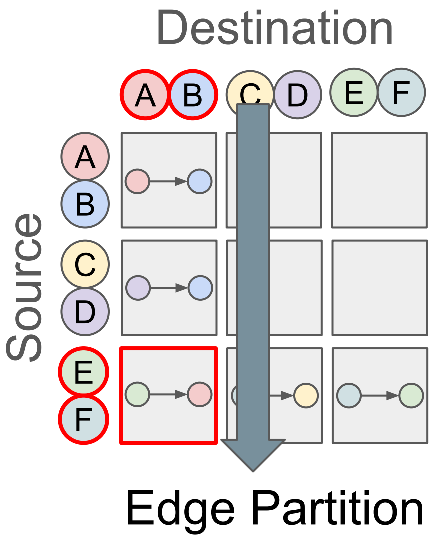


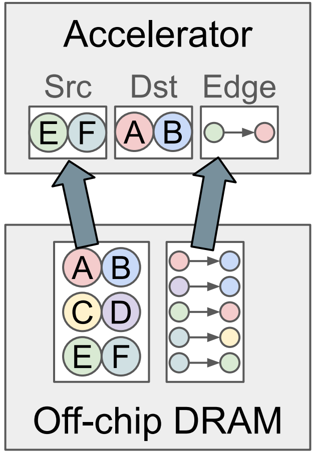


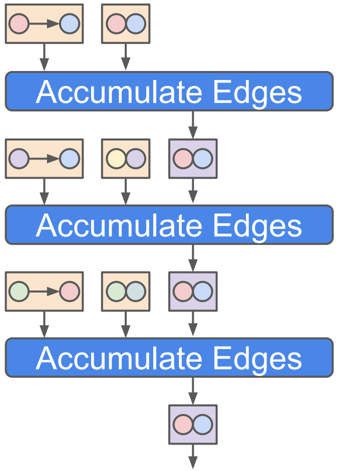


Execution Partitioning Example

Execution follows columns

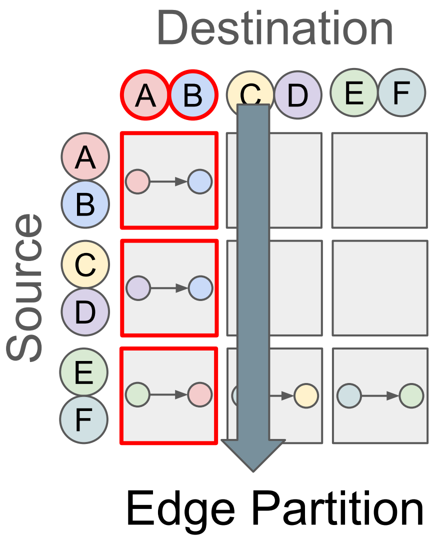


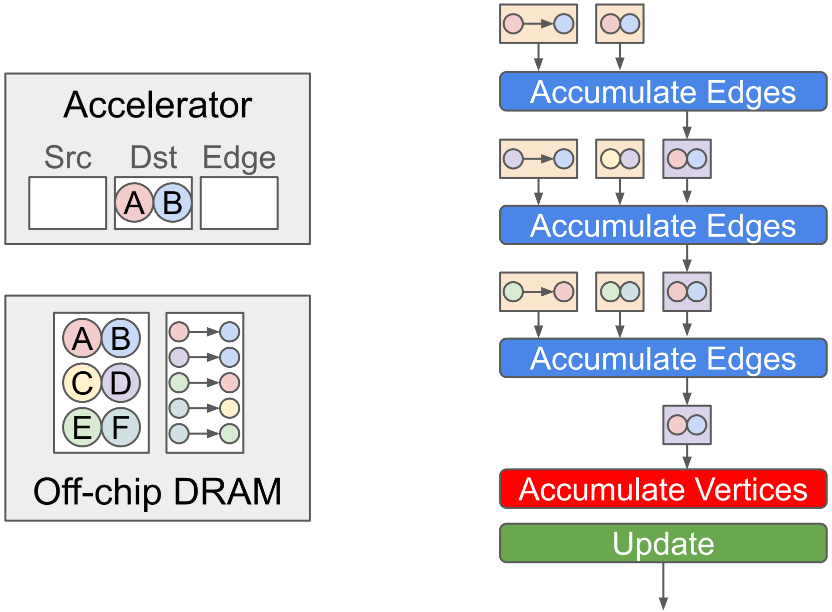




Execution Partitioning Example

Execution follows columns





Talk Agenda

- Introduction
- GReTA Overview
- Execution Model
- Partitioning
- **Experimental Results**
- Conclusion

Experimental Setup

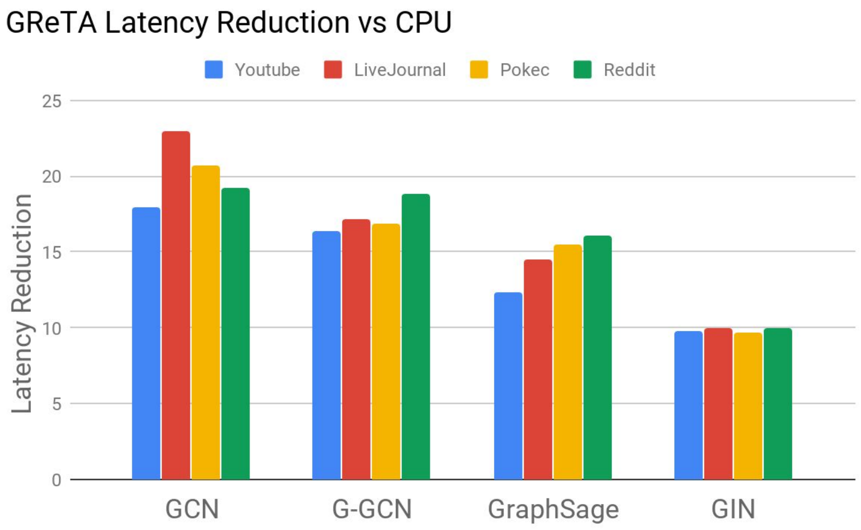
- Implemented range of GNN models
  - GCN (simple, classic GNN model)
  - GraphSage (max-reduce instead of sum)
  - GIN (MLP in transform layer)
  - G-GCN (per-edge computation)
- Baseline
  - CPU: 2.6 GHz Intel Xeon E5-2690v4
  - GPU: Nvidia Tesla P100
  - Models implemented using Tensorflow
- Compared to custom 32nm GReTA accelerator
- Key performance metric: Total inference latency for batch size of 1

	Dataset	Nodes	Edges	2-Hop
YT	YouTube	1.13M	2.98M	25
LJ	LiveJournal	3.99M	34.6M	65
PO	Pokec	1.63M	30.6M	167
RD	Reddit	232K	47.4M	239

Evaluation Datasets

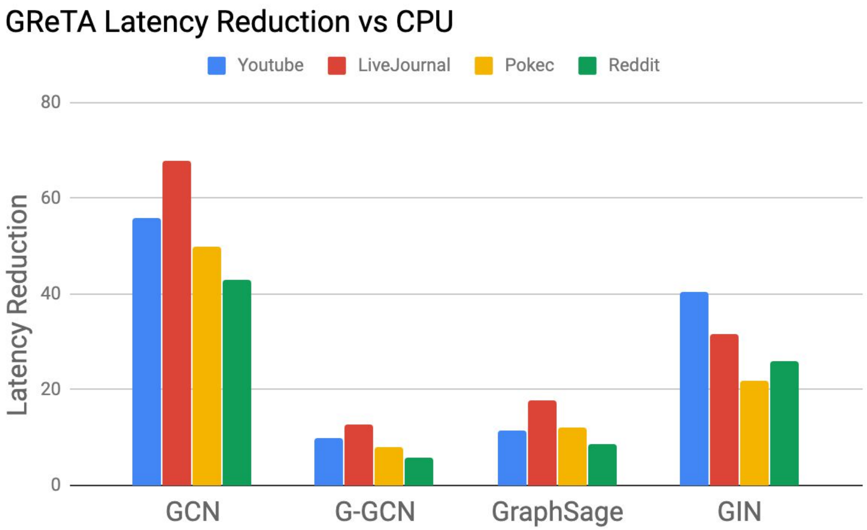
9-23x Latency Reduction vs CPU

- **15x** g.mean across all datasets/models
- Best results on models where message passing dominates (GCN, G-GCN)



6-67x Latency Reduction vs GPU

- **21x** g.mean across all datasets/models
- Best speedup on models with low overall latency (GCN, GIN)
- Small batch size means data transfer latency often dominates



Conclusion

Key features of GReTA:

1. **Simple** representation using a graph framework
2. **Expressive** enough to allow for a wide range of GNNs
3. **Efficient** execution on an accelerator

Future work: Apply GReTA beyond GNNs? Integration with existing frameworks?

Conclusion

Key features of GReTA:

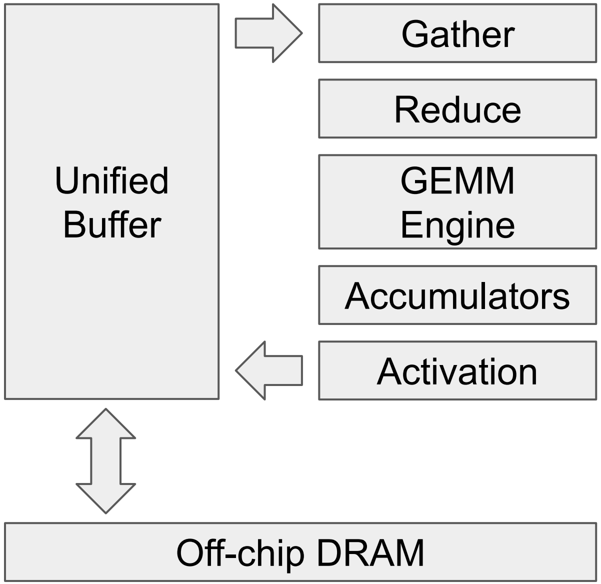
1. **Simple** representation using a graph framework
2. **Expressive** enough to allow for a wide range of GNNs
3. **Efficient** execution on an accelerator

Future work: Apply GReTA beyond GNNs? Integration with existing frameworks?

🙋 **Q&A?** 🤔

GReTA Accelerator

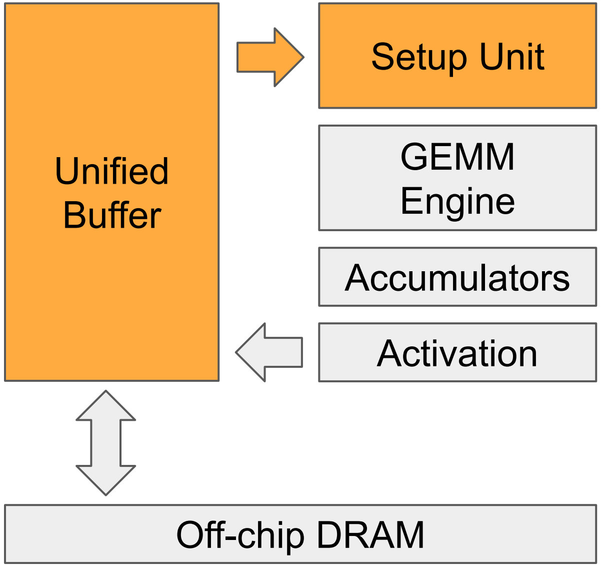
- Replace setup with unit for `Gather`-ing edge/vertex values
  - Uses graph adjacency info stored in Unified Buffer
- New accumulator unit for `Reduce`
- Note: Existing NN ops can still run on new architecture!
  - Gather unit just performs single load
  - Reduce unit performs no-op



Compiling GReTA to a TPU-like Architecture

Execution in four stages

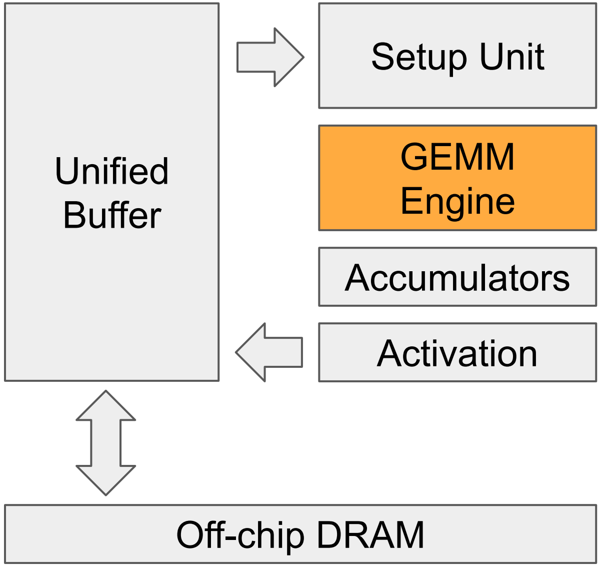
1. **Load:** Move data from unified buffer into setup unit



Traditional DNN Accelerator Model

Execution in four stages

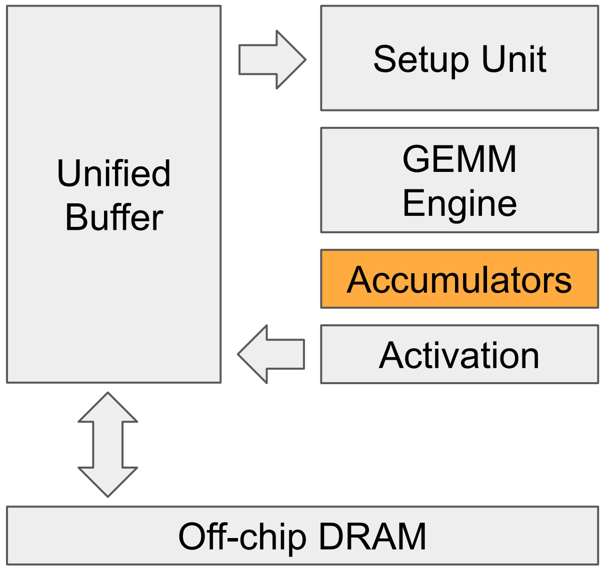
1. **Load:** Move data from unified buffer into setup unit
2. **Compute:** Multiply setup data by pre-loaded weight values



Traditional DNN Accelerator Model

Execution in four stages

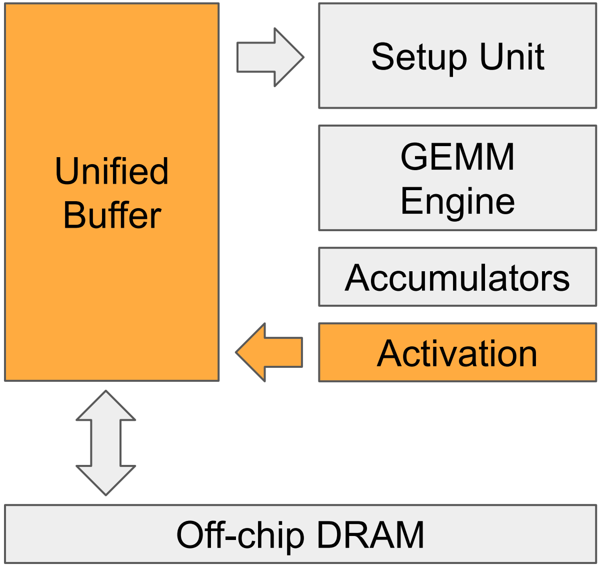
1. **Load:** Move data from unified buffer into setup unit
2. **Compute:** Multiply setup data by pre-loaded weight values
3. **Accumulate:** Collect output from compute over *N* cycles



Traditional DNN Accelerator Model

Execution in four stages

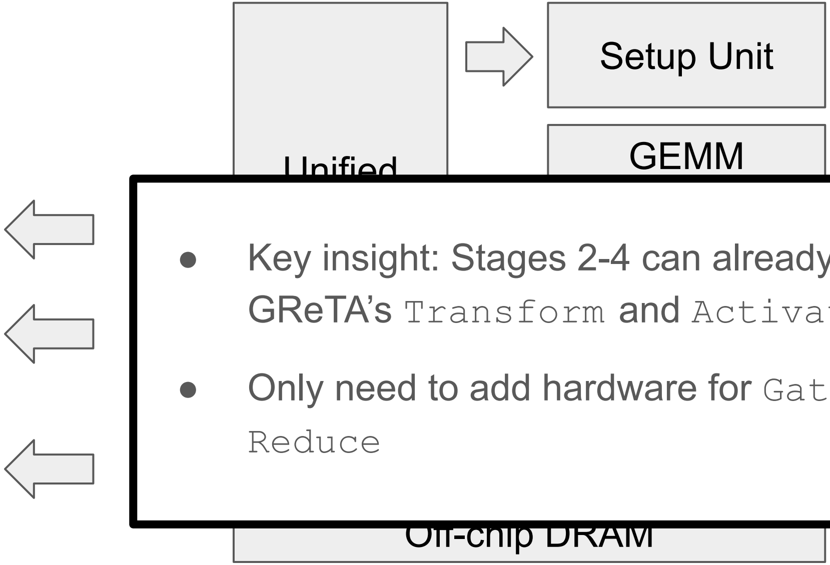
1. **Load:** Move data from unified buffer into setup unit
2. **Compute:** Multiply setup data by pre-loaded weight values
3. **Accumulate:** Collect output from compute over *N* cycles
4. **Activate:** Execute required activation/normalization and store result



Traditional DNN Accelerator Model

Execution in four stages

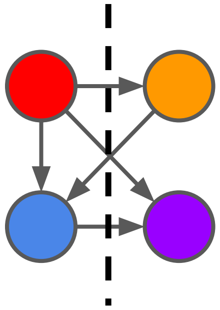
1. **Load:** Move data from unified buffer into setup unit
2. **Compute:** Multiply setup data by pre-loaded weight values
3. **Accumulate:** Collect output from compute over *N* cycles
4. **Activate:** Execute required activation/normalization and store result



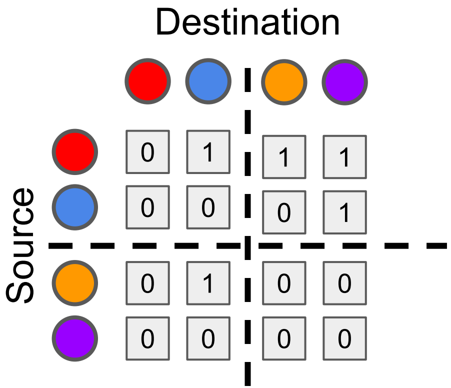
- Key insight: Stages 2-4 can already execute GReTA's `Transform` and `Activate` UDFs
- Only need to add hardware for `Gather` and `Reduce`

Graph Partitioning

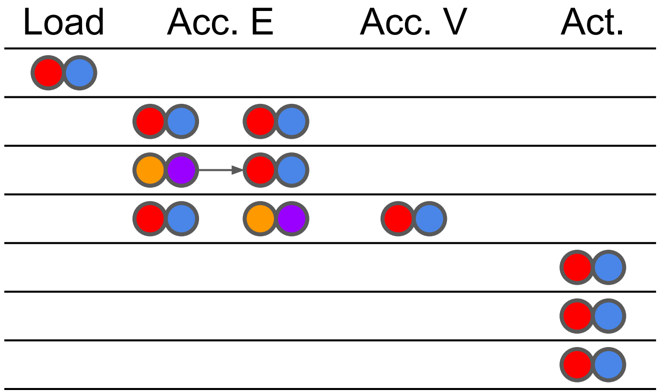
- Problem: Data for full graph may be too large to fit entirely on accelerator
- Solution: Partition graph and execute phases for each partition separately



Vertex Partition



Edge Partition

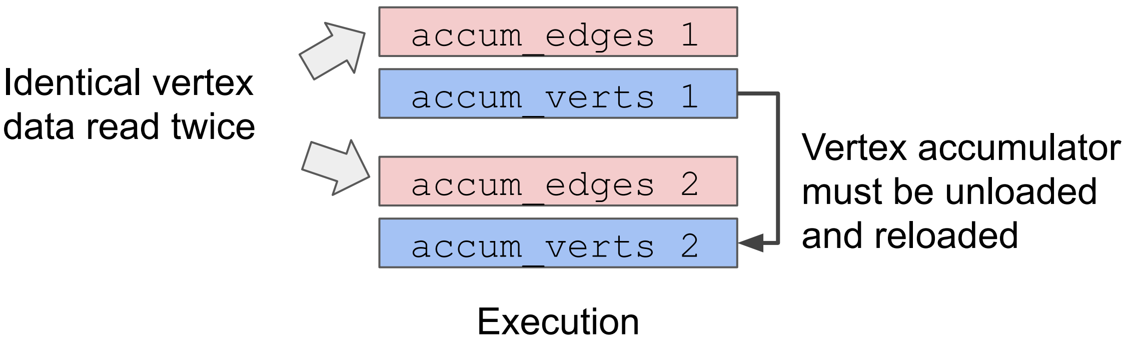


Interleaving Execution

- Multiple GReTA programs in a layer may reuse data
  - Read identical edge/vertex data
  - Reuse accumulator values
- Interleaving execution improves data locality

$$h_{v,1} \leftarrow W_1 \sum_{u \to v} h_u$$

$$h'_v \leftarrow h_{v,1} + W_2 \sum_{u \to v} h_u$$



Interleaving Execution

- Multiple GReTA programs in a layer may reuse data
  - Read identical edge/vertex data
  - Reuse accumulator values
- Interleaving execution improves data locality

$$h_{v,1} \leftarrow W_1 \sum_{u \to v} h_u$$

$$h'_v \leftarrow h_{v,1} + W_2 \sum_{u \to v} h_u$$

Vertex data read once, reused

accum_edges 1

accum_edges 2

accum_verts 1

accum_verts 2

Vertex accumulator stays loaded

Execution

Weight Tiling

- Problem: Layer weights can be too large to fully load into GEMM unit
- Existing solution: Slice weights into tiles and reloading for each new vertex
    - Unfortunately, gives worst case reuse of each tile
    - Accelerator often bottlenecked on loading/reload weight tiles