Graphite: Optimizing Graph Neural Networks on CPUs Through Cooperative Software-Hardware Techniques

Zhangxiaowen Gong†*, Houxiang Ji†, Yao Yao†, Christopher W. Fletcher†, Christopher J. Hughes*, Josep Torrellas†

†University of Illinois at Urbana-Champaign, *Intel Labs
Session 10B, ISCA 2022
June 22, 2022
Graph Neural Network (GNN)

- Traditional DNNs (e.g. CNNs) can hardly process graphs
- GNN specializes in processing graphs
- Application domains:
  - Recommender systems
  - Social networks
  - Knowledge graphs
  - Physics
  - Life science
  - And many more
GNN Characteristic: Alternating Phases

- Two alternating phases: **Aggregation** and **Update**

  - **Aggregation**: each vertex gathers and reduces features from neighbors/edges
    
    $a^k_v = \text{AGGREGATE}(h_u^{(k-1)} \mid \forall u \in N(v) \cup \{v\})$

  - **Update**: each vertex computes its output features from the aggregation outputs with a DL op (e.g. MLP)
    
    $h^k_v = \text{UPDATE}(a^k_v)$

  - **Sparse** connections
  - **Irregular** memory access patterns
  - **Poor** locality
  - **Memory intensive**
  - **Variable** execution time for each vertex, correlated with the vertex’s degree

  - **Dense** computation
  - **Regular** memory access patterns
  - **Good** locality
  - **Compute intensive**
  - **Similar** execution time for each vertex
GNN Characteristic: Activation (Feature) Sparsity

- Sparsity: zeros in the working sets
- ReLU: 20-80% sparse
- Dropout in training: often 50% dropped
- Combined: often >80% sparse
- Dynamic and unstructured
- Operating on zeros: ineffectual

Example: feature sparsity during 3-layer GraphSAGE training
Other GNN Characteristics

- Long feature length
  - Traditional graph analytics: often scalar feature
  - GNN: often hundreds to thousands

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vertex feature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>1,433</td>
</tr>
<tr>
<td>Citeseer</td>
<td>3,703</td>
</tr>
<tr>
<td>Reddit</td>
<td>602</td>
</tr>
<tr>
<td>Ogbn-products</td>
<td>100</td>
</tr>
</tbody>
</table>

- Reuses input graphs in training
Motivation: GNNs on CPUs

- Real-world graphs are often huge
  - Millions to billions of vertices and edges

- CPUs: viable platforms for GNNs
  - Terabyte-level memory capacity
  - Have high availability

- GNNs on CPUs are memory bandwidth bound
  - 3-layer GraphSAGE training on CPUs:
    - 10% of pipeline slots do useful work
    - 62% of pipeline slots are stalled waiting for memory

| Graph          | |V|   | |E|   |
|----------------|-----|-----|-----|
| Ogbn-products  | 2.5M| 124M|
| Ogbn-papers100M| 111M| 1.6B|
| wikipedia      | 3.6M| 45M |
| twitter        | 62M | 1.5B|
Contribution: Graphite

- Graphite: cooperative SW-HW techniques that optimize GNNs on CPUs

- Software techniques:
  - Layer fusion: overlap compute and memory
  - Feature compression: reduce memory traffic
  - Input preprocessing: increase locality
  - Inference 1.8x, training 1.9x speedup

- HW-SW co-design techniques:
  - Enhanced DMA engine: offload aggregation
  - Inference 1.8x, training 2.4x speedup
Graphite Software Techniques
Basic Optimized Implementation

- **Aggregates** all vertices then **updates** them
- **Aggregation**
  - JIT-assembled kernel
  - Output parallelized
  - Hand vectorized
  - Software prefetch
  - OpenMP dynamic scheduling
- **Update**
  - Stock library GEMM

![Diagram showing batch of vertices dynamically distributed to processors and all vertices evenly distributed to processors.](image)
Layer Fusion

- Goal: overlap memory-bound and compute-bound operations
- Fusion: interleave **aggregation** and **update** of vertex batches
Overlapping Compute-Memory: Within a Processor

- Prefetches the features needed by the aggregation of the next batch
- Ongoing prefetch overlaps with the update
Overlapping Compute-Memory: Among Processors

- **Aggregation**: variable time
- **Update**: fixed time
- Executions on different processors naturally go out-of-phase
- Memory bandwidth: a shared resource
Feature Compression

Goal: reduce memory traffic
Avoid loading/storing zeros
Fast vector comparison and (de)compression instructions

Compression

Step 1: generate bit-mask
- Zero vector: 0 0 0 0 0 0 0
- Input vector: 10 7 0 43 0 0 0 22
- Bit mask: 1 1 0 1 0 0 0 1

Step 2: bubble collapse
- Bit mask: 1 1 0 1 0 0 0 1
- Input vector: 10 7 0 43 0 0 0 22
- Compressed vector: 10 7 43 22

Decompression

- Bit mask: 1 1 0 1 0 0 0 1
- Compressed vector: 10 7 43 22
- Decompressed vector: 10 7 0 43 0 0 0 22
- Bubble expand
Increasing Locality in Aggregation

- Aggregation: each vertex gathers neighbors’ features
  - Features span multiple cache lines
  - Temporal locality of features is important
- Goal: increase temporal reuse of vertex features
Increasing Locality in Aggregation: Algorithm

- Computes a new processing order of vertices
- Grouping: assigns each vertex to the group of its highest-degree neighbor
- Vertices in a group are processed temporally closely and reuse at least one feature vector

![Graph Diagram]

Original processing order: v0, v1, v2, v3, v4, v5
New processing order: v0, v2, v3, v4, v1, v5

Group of v1: v0, v2, v3, v4
Group of v4: v1, v5
Increasing Locality in Aggregation: Algorithm

- Computes a new processing order of vertices
- Grouping: assigns each vertex to the group of its highest-degree neighbor
- Vertices in a group are processed temporally closely and reuse at least one feature vector

Original processing order: v0, v1, v2, v3, v4, v5
New processing order: v0, v2, v3, v4, v1, v5

Group of v1: v0, v2, v3, v4
Group of v4: v1, v5
reuse the features of v1
Increasing Locality in Aggregation: Algorithm

- Computes a new processing order of vertices
- Grouping: assigns each vertex to the group of its highest-degree neighbor
- Vertices in a group are processed temporally closely and reuse at least one feature vector

Original processing order: v0, v1, v2, v3, v4, v5
New processing order: v0, v2, v3, v4, v1, v5
Increasing Locality in Aggregation: Overhead

- Linear complexity $O(V+E)$, good scalability
- **We only apply the optimization in GNN training**
  - Training contains many epochs
  - The cost of preprocessing the inputs is amortized
Graphite HW-SW Co-design Techniques
GNN Aggregation and DMA

- Aggregation: gather and reduce
- Gathered features have low reuse
- Scatter-gather is a common DMA operation
- Graphite enhances DMA to perform aggregation
Graphite DMA Structure

- One DMA engine per processor
  - Connected to NoC
  - Virtual address: L2 STLB for address translation
  - Works in user-space
  - Reuses function units in existing DMA
  - Adds a narrow vector unit to perform reductions

- Descriptor-based programming model
  - 64B descriptor encodes an aggregation
  - Easily built from CSR encoded adjacency matrices

- Incompatible with feature compression for cost reason
DMA Aggregation

P0
Core + L1
L2
DMA
L3 + directory

P1
Core + L1
L2
DMA
L3 + directory

P2
Core + L1
L2
DMA
L3 + directory

NoC
DMA Aggregation

P0: Core + L1 → L2 → DMA → L3 + directory

P1: Core + L1 → L2 → DMA → L3 + directory

P2: Core + L1 → L2 → DMA → L3 + directory

core issues descriptor
DMA Aggregation

P0: Core + L1 → L2 → DMA → L3 + directory

P1: Core + L1 → L2 → DMA → L3 + directory

P2: Core + L1 → L2 → DMA → L3 + directory

NoC

DMA gathers input features
DMA Aggregation

DMA reduces and writes results to L2
DMA Assisted Layer Fusion

- On each processor:
  - DMA: aggregation
  - Core: update
- The **update** of a vertex batch overlaps with the **aggregation** of the next vertex batch
Evaluation Setup

- **GNN Models:**
  - 3-layer GCN and GraphSAGE

- **Datasets:**
  - 4 graphs with 2.5M-111M vertices and 45M-1.6B edges

- **Baseline:**
  - SOTA SpMM from DistGNN[1] + MKL GEMM

- **Evaluation:**
  - SW-only techniques: 28-core Cascade Lake server running 28 threads

---

Performance: SW-Only Techniques

- Feature compression @ 50% sparsity
- Locality optimization only on training
- Techniques are synergetic

**Average inference speedup**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic optimized implementation</td>
<td>1.07</td>
</tr>
<tr>
<td>Layer fusion</td>
<td>1.35</td>
</tr>
<tr>
<td>Feature compression</td>
<td>1.45</td>
</tr>
<tr>
<td>Layer fusion + feature compression</td>
<td>1.81</td>
</tr>
</tbody>
</table>

**Average training speedup**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic optimized implementation</td>
<td>1.07</td>
</tr>
<tr>
<td>Layer fusion</td>
<td>1.18</td>
</tr>
<tr>
<td>Feature compression</td>
<td>1.40</td>
</tr>
<tr>
<td>Layer fusion + feature compression</td>
<td>1.55</td>
</tr>
<tr>
<td>Layer fusion + feature compression + locality optimization</td>
<td>1.88</td>
</tr>
</tbody>
</table>
Performance: HW+SW Techniques

- DMA aggregation is incompatible with feature compression
- DMA fusion is more effective than SW-only fusion

![Average inference speedup](chart1.png)

![Average training speedup](chart2.png)
Conclusion

- GNNs on CPUs: memory bandwidth bound
- Graphite alleviates memory pressure by:
  - Fusing layers to overlap compute and memory
  - Compressing features to reduce memory traffic
  - Optimizing the vertex processing order to improve locality
  - Augmenting the DMA engine to offload aggregation
- Evaluated with 28 cores
  - SW-only techniques: inference 1.8x, training 1.9x speedup (native)
  - HW+SW techniques: inference 1.8x, training 2.4x speedup (simulated)

More in the paper:
- Algorithms of the techniques
- DMA descriptor design
- In-depth evaluation of individual techniques
- And more...
Thanks